20

An object–oriented CAE Framework for asynchronous Circuit Synthesis

U. Baake and S. A. Huss

Integrated Circuits and Systems Lab.
Department of Computer Science
Technical University of Darmstadt
64283 Darmstadt, Germany
{baake, huss}@vlsi.informatik.th–darmstadt.de

Abstract

The paper presents an interactive OSF/Motif–based design framework for high–level specification, analysis and synthesis of asynchronous control circuits. As novel contributions we propose an object–oriented graphical design environment including interfaces to external state of the art asynchronous circuit synthesis tools. The proposed design environment represents a powerful CAE design system including features like geometric representation of abstract graph–based input specifications, menu–based command entry, user–defined configuration of input devices and a comprehensive set of visualization and manipulation commands. Thus, by using this system, a circuit designer is able to specify and visualize asynchronous circuit behaviour on a high abstraction level and synthesize the corresponding logic using the built–in algorithms as well as other available non–graphical synthesis tools.

1 Introduction

In the domain of interface circuitry, where the use of a global clock is rarely applicable, design methods for asynchronous circuits have recently gained an increasing interest in the logic synthesis community [LL91][LS93][Men90][MM94]. By using design methodologies for locally synchronous and globally asynchronous operation of data processing [Bor89] problems related to limited throughput, clock skew, and clock distribution may be avoided. Even in this restricted domain, asynchronous design has been considered as time–consuming and difficult because of the lack of both, a good formal specification and of appropriate synthesis and verification tools. Formal specification of asynchronous circuits is not that easy because behaviour such as concurrency, sequencing, conflict, timing constraints and data–dependency is difficult to specify in a way that is both, natural for designers and at the same time appropriate for formal analysis or verification. In contrast to synchronous design, where powerful CAE environments, such as those provided by, e.g., Synopsys or Viewlogic, support the specification and synthesis process, available asynchronous circuit synthesis tools, such as e.g. SIS [BLM+92] from UCB or ASSASSIN [YVL93] from IMEC, use an alphanumeric design entry only.

In this paper we present an object-oriented, interactive design environment that supports automated asynchronous circuit design starting from a behavioural graph–based problem specification called signal transition graphs STGs [Chu87]. This formal specification form is exploited in other well–known synthesis tools, e.g. [BLM+92], too. Thus, the implementation of well defined

F. J. Rammig et al. (eds.), *Electronic Design Automation Frameworks*
© Springer Science+Business Media Dordrecht 1995

interface layers allows specification-import and data-export between the proposed framework and e.g. SIS or ASSASSIN. A powerful component for visualizing and beautifying graph–based specifications allows for a comfortable processing of imported non–graphical STG specifications. Using the OSF/Motif–based graphical front–end of the framework for flexible design capture and manipulation, a designer can use proprietary algorithms for analysis or synthesis as well as other well–known synthesis methods by exporting the specification according to their requirements.

In addition to the object–oriented representation of STGs [BH93], we introduce algorithms for constructing geometric representations from textual specifications. This is essential for importing and visualizing graph–based textual specifications within the graphical front–end of the framework. The automatic generation of drawings of graphs has important applications in key computer technologies such as software engineering, database design, and visual interfaces.

The paper is organized as follows. In Section 2, the specification form used for describing interface behaviour is briefly introduced in a formal way. In Section 3, we give an overview for the main components of our system, called Graphical Asynchronous Circuits Environment. In Section 4 a more detailed description of the GRACE front–end is outlined, while Section 5 describes the analysis and synthesis parts, respectively. In Section 6 the import and export interfaces of GRACE are described. An example of a complete design flow, processed by this CAE framework, concludes the paper.

2 Design specification

In this section we give a brief introduction into signal transition graphs and show that this specification is very well–suited to describe asynchronous circuit behaviour. STGs, introduced independently in [Chu87] and [RY85], represent a graph–based methodology for the specification of speed-independent asynchronous digital controllers. An STG provides an appropriate representation of concurrency. In addition its natural timing–diagram–like specification makes it easy to describe interface communications with signals as outlined in Figure 1.

An STG $\Sigma = \langle T, P, F, m_0 \rangle$ is an interpreted Free Choice Net (FC Net, i.e. subset of a Petri Net), which is expressive enough to specify concurrency or choice, but yet simple enough for analysis. The set of transitions T is interpreted as the set of signal transitions $S \times \{+, -\}$. S denotes a set of signals, where for every signal $s \in S$ a rising transition s^+ and a falling transition s^- is associated, respectively. s^* denotes any transition of signal s. P is a set of places that can be used to specify conflict or choice and $F \subseteq (T \times P) \cup (P \times T)$ represents the flow relation between transitions and places. The marking is an integer labeling of the places $m : P \rightarrow \mathbb{N}$, denoting the number of tokens in a place p. m_0 represents the initial marking of Σ. The *preset* of a place or a transition $x \in P \cup T$ is given by $\cdot x = \{y | (y, x) \in F\}$ and the *postset* of x is denoted $x \cdot = \{y | (x, y) \in F\}$, respectively.

A transition t is said to be enabled, if $\forall p \in \cdot t : m(p) \geq 1$. This is denoted as $m[t\rangle$. The set of all enabled transitions is then $\mathcal{F} = \{t \in T \mid m[t\rangle\}$. An enabled transition may fire according to the Petri Net firing rule.

A marked graph (MG) is a Petri Net, where each place p has exactly one predecessor transition, denoted as $source(p)$, and one successor transition, called $target(p)$. It is usually represented by a directed graph G, where the vertices correspond to transitions and the arcs to places.

An STG/MG is an STG that is derived from an MG.

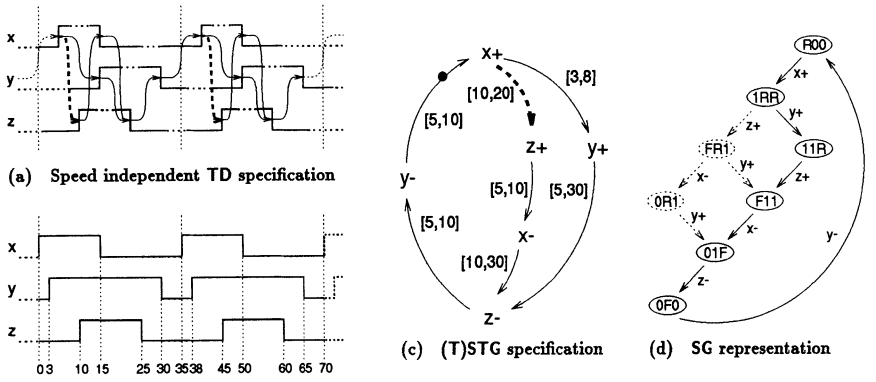


Figure 1: Behavioural design specifications

In order to include additional timing information related to signal transitions, STG models could be extended [BH94][Van93][VGL94] as to specify timed interface circuits by assigning lower and upper bounds of the firing time on each arc p of an timed STG, denoted as $b(p)$ and $B(p)$, respectively. In Figure 1a,b two different timing diagrams are presented: a speed independent version corresponding to the STG without timing information on its arcs and a timed version. For a detailed description of scheduling and firing transitions by including timing intervals in STG specifications, denoted as TSTGs, we refer to [VGD92][BH94].

A state graph (SG) $\Phi = \langle M_S, T, \delta, m_0 \rangle$ may be derived from Σ by performing a reachability analysis according to the Petri Net firing rule [Chu87]. The transformation of m into m' by firing an enabled transition t, is denoted as $m[t\rangle m' \Leftrightarrow m' = \delta(m,t)$ with the partial function δ: $M_S \times T \rightarrow M_S$. $M_S = \{m : S \rightarrow \{0,1,R,F\}\}$ denotes the set of all states, where each vector $\langle m(1), m(2), ..., m(n)\rangle$ specifies the value of each signal $s_i \in S$ of the system when the system is in that state. The initial state is denoted as m_0. Figure 1d shows the complete SG representation of the STG specification outlined in Figure 1c without including the timing information on the STG arcs. The reduced SG derived form the timed STG (TSTG) is given by removing the dotted states and arcs from the SG in Figure 1d. For more information on deriving reduced logic implementations from timed input specifications we refer to, e.g., [MM93].

Specification of delay insensitive circuit behaviour may be denoted in a completely different way to the outlined STG method. First, language–based specifications such as communicating sequential processes (CSPs) [Mar90], and second by means of asynchronous finite state machines (AFSMs). CSPs seem to be well–suited for the representation of large designs on a high abstraction level but problems may arise when attempting to specify concurrency within a process. Asynchronous finite state machines [CL93][YDN93] have already been successfully used for hazard-free synthesis of asynchronous controllers, but neither CSPs nor AFSMs incorporate timing constraints as it is usually required for a consideration of wire and environment delays [HM90][MM93][BH93].

3 The CAE framework GRACE, an overview

In the following section we give a description of our CAE framework for asynchronous circuit synthesis. The Graphical Asynchronous Circuits Environment (GRACE) consists of five main components as depicted in Figure 2.

The object-oriented data administration is central to all components. One important reason for using an object-oriented representation is the introduction of independent layers of object classes. Therefore, we separate the universal data structures, which include new object-oriented abstract graph structures as well as powerful C++ class libraries for X-Window based graphical representations, and the problem specific data objects like, e.g., STG or state graph (SG) representations. A more detailed description of the general object–oriented concepts may be found elsewhere [BBH94].

The OSF/Motif–based graphical front–end consists of the powerful STG editor SignEd and of the wavefront postprocessing tool TimEd that handles the visualization of the equivalent timing diagram while editing a protocol specification in form of an STG. Further on, TimEd visualizes the results from exercising timing analysis algorithms in order to calculate the schedule of all signal transitions of an STG specification, as outlined in Figure 1, or to determine precedence relations between particular transitions.

The import/export tool ImEx is responsible for the communication with other STG–based synthesis packages.

These graphical front–end was implemented using the universal C++ class library proposed in [BBH94]. One of the new features of this library is object representation in Adobe's Postscript

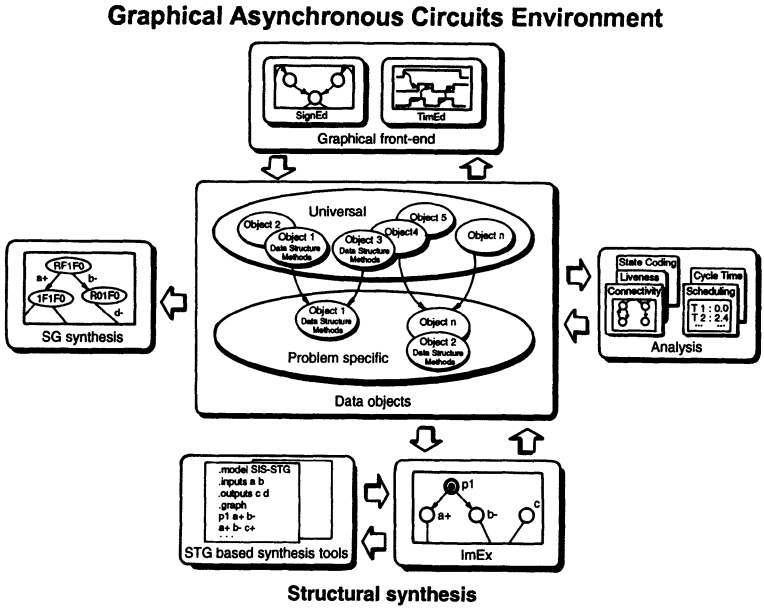


Figure 2: Components of GRACE

language. Thus, each imported STG specification and its corresponding timing diagram can not only be visualized, but also be represented in Postscript, a feature, which is important for documentation and visual validation.

The analysis component hosts a library of algorithms for the assessment of the problem specification in STG terms. Since initial specifications of application specific protocols are not necessarily hazard-free or even consistent [LKS91], powerful analysis means are mandatory to unveil several problems, such as non-feasible initial conditions or wrong schedules during protocol operation [BH94]. Finally, the synthesis component produces the structural description of the logic circuits (netlist) according to the validated STG representation of the communication protocol.

4 The graphical front-end

The graphical front-end of GRACE includes two main components: an STG editor SignEd outlined in Figure 3, and the wavefront postprocessor TimEd, as depicted in Figure 4.

4.1 SignEd

The editor SignEd represents an interactive X–Window–based graphic tool allowing for user–friendly specification of STGs in order to obtain a clearly arranged representation of the communication protocol. Some of its features are menu–based command entry, user-defined configuration of input devices, and a set of visualization and manipulation commands, such as node

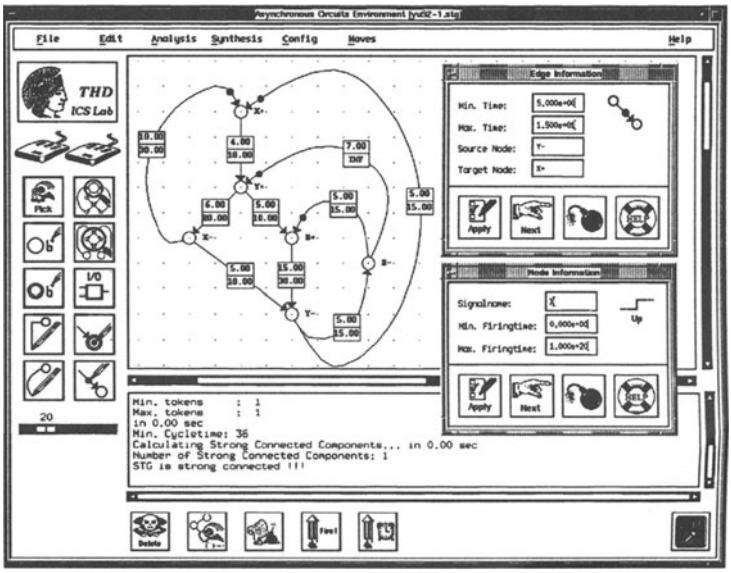


Figure 3: The editor SignEd

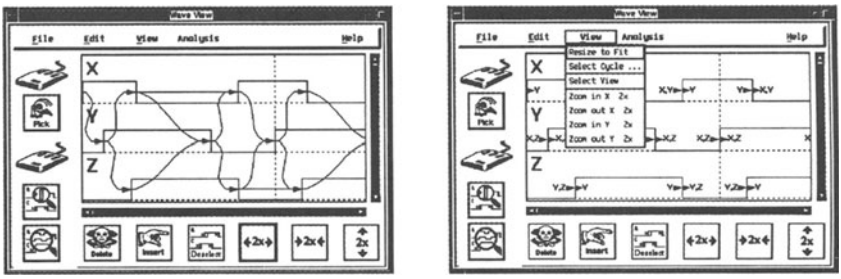


Figure 4: Selecting different views in the wavefront postprocessor TimEd

and arc editing or parameterization of objects..

SignEd is integrated into the application shell of GRACE, such that graph and net specific characteristics, like token flow and strong connected components, are first calculated by the analysis component as outlined in Section 4 and then are visualized using the SignEd back–end.

4.2 TimEd

The waveform postprocessing tool TimEd supports timed specifications of asynchronous interface circuits. Performing a minimum scheduling of the signal transitions in order to derive timing relations between any pair of transitions as well as the minimum pipelining rate of the specified circuit, TimEd supports an appropriate presentation of the results.

By using TimEd a designer is able to select signals in order to check maximum separations between particular signal transitions for an assessment of possible overlapping zones between these transitions. Separations between concurrently enabled transitions are important for, e.g., logic reduction during the synthesis process [MM93].

5 Analysis and synthesis

Many algorithms have been proposed for various stages of asynchronous circuit synthesis. General methods for analysing STG specifications corresponding to their initial marking [BH93] and their state coding properties [YS92] as well as specific methods for analysing timed STGs (e.g. pipelining rate calculation, interval based schedules, etc.) are among these proposals. Some of these analysis algorithms are integrated into the graphical front–end of GRACE in order to arrange for an appropriate presentation of possible errors resulting from STG specification. Based on this feedback a specification may be refined by an user, resulting eventually in an error-free STG representation of the communication protocol to be implemented.

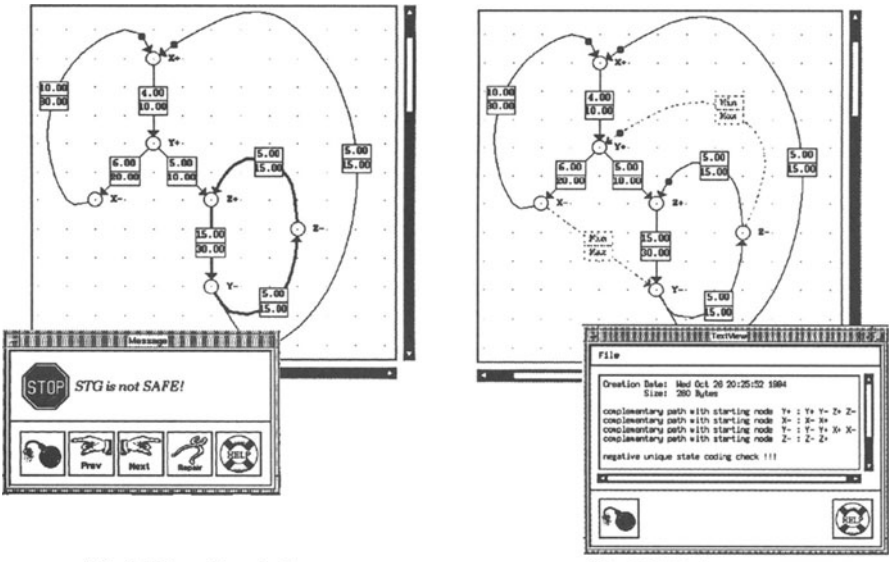
Another group of algorithms includes synthesis methods for speed independent and timed specifications, respectively. In the current version of GRACE state graph synthesis and the derivation of Boolean equations is supported by built–in algorithms. The synthesis tool SIS from UCB may be used for minimization and technology mapping as well.

5.1 Analysis component

The initial marking of a strongly connected STG has to be checked in order to guarantee that every simple cycle contains more than one token and every arc belongs to at least one simple cycle that contains exactly one token. Figure 5a shows the visualization of such a check within the framework. Referring to the error message box, GRACE supports a visual browsing between the detected errors as well as an automatic refinement of the incorrect marking, while preserving this marking as far as possible. For a detailed description of the algorithms we refer to [BH93][BBH94].

Other essential conditions for deriving correct SGs are the USC (Unique State Coding) or CSC (Correct State Coding) properties [YS92][LMBS92]. In the following we give an example for a USC check performed by GRACE. An STG has the USC property if two distinct states in the SG derived from the STG do not have identical binary codes. It is clearly possible to have an algorithm that first generates an SG and then checks the SG for this property. However, working directly at STG level with a substantial lower graph complexity, a designer can more easily suggest ways to fix the problem. Different algorithms for USC are presented in the literature. In GRACE a path-oriented method based on complementary path detection is integrated. For a detailed algorithm description and more information about complementary paths we refer to [YS92]. This method has the advantage of an easier visualization of the USC property. Due to its intuitive correlation of USC to an STG specification an user may derive possible solutions to modify the STG in order to ensure the USC property.

The message box in Figure 5b shows the results of a USC check performed by GRACE on our



(a) Initial marking check (b) USC check

Figure 5: STG analysis

running example. The check is negative since complementary paths are detected. A possible modification of the STG in order to guarantee the USC property is outlined in the graphic window of Figure 5b.

5.2 Synthesis component

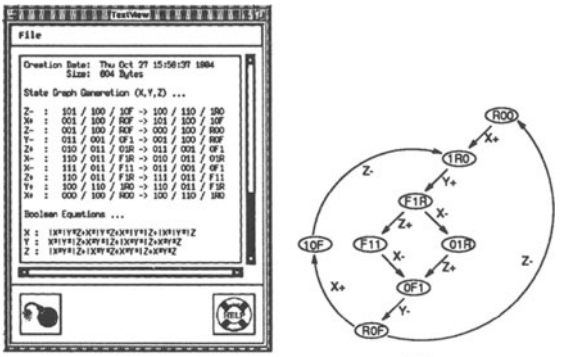


Figure 6: (a) GRACE synthesis results (b) SG representation

Performing a complete state coding on the analyzed and correctly modified running example depicted in Figure 5b leads to the results given in Figure 6. In addition, the corresponding Boolean equations are generated. These equations may then the input for SIS as described in the example outlined in Section 7. New algorithms for generating reduced SGs corresponding to timed STG specifications [MM93] are also part of this component.

6 Import / export

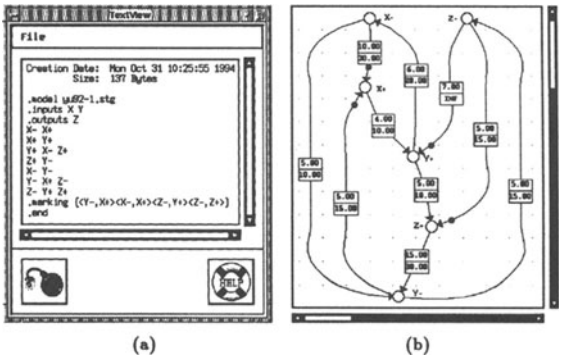


Figure 7: (a) SIS format (b) Graphical representation

In this section the import / export component of GRACE, called ImEx, is presented. Using ImEx textual STG specifications generated for or by other synthesis tools may be imported into GRACE in order to provide the utilization of the enhanced graphical analysis features as outlined above. For an appropriate representation the universal object–oriented STG model [BH93] has been extended by new algorithms for constructing geometric representations from textual specifications. Therefore the vertices of the graph are assigned to different layers and then the crossings of arcs between this layers are minimized. Finally, the vertices of each layer have to be moved in order to beautify the total layout. Quadratic programming methods and heuristics have been implemented in order to solve this problem. The final graph may be visualized using polygon and spline arcs as well. For a detailed description of the algorithms and their results we refer to [Bau94]. Figure 7 shows the the import of our running example from a textual description in Berkeley SIS format.

Another component of ImEx is the export of STG models generated within GRACE to textual representations. This interface may be easily extended to utilize new STG specification formats.

7 An example

An example of a complete design flow as processed by the framework GRACE, is outlined in Figure 8. We start therefore from the behavioural description of a simple handshake circuit given in [HM90] that is, e.g., imported from SIS format, then we analyse and modify with GRACE. Finally the Boolean equations for the corresponding output signals Aout and Rout are generated. For minimization and technology mapping, e.g., SIS may be used.

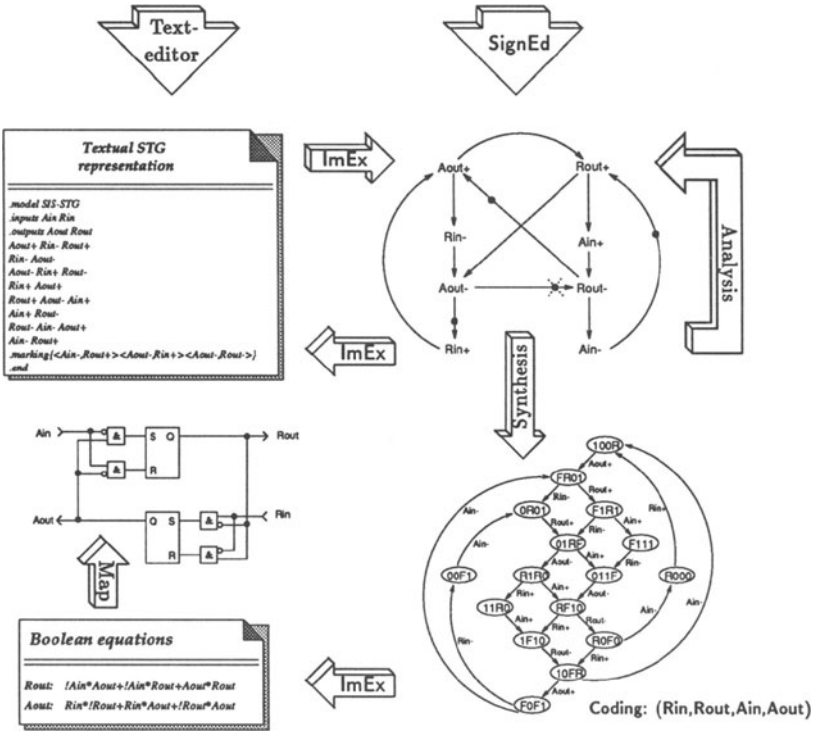


Figure 8: Design Flow

8 Conclusions

Many algorithms have been proposed for analysis and synthesis of asynchronous control circuits as outlined in this paper. However, no comprehensive visualization of the results and no complete graphical specification, analysis, and synthesis system, in which all of these algorithms are employed, has been reported to date. Therefore, we propose GRACE, a graphical asynchronous circuit environment that is both, a framework for implementing and evaluating new algorithms

and a tool for automatic synthesis of asynchronous circuits using external state of the art synthesis tools by ImEx. Past experience has shown that in comparison to alphanumeric environments a designer may gain a lot of time during the specification and analysis process by using GRACE.

References

[Bau94] Jens Baumgart. Object-oriented implementation and visualisation of a graph-based data model. Master's thesis, Dept. of Computer Science, Technical University of Darmstadt, 1994.

[BBH94] Uwe Baake, Werner Bachmann, and Sorin A. Huss. High-level specification and validation of asynchronous communication protocols in an object-oriented graphical design environment. In Proc. of the International Conference on Concurrent Engineering and Electronic Design Automation, pages 349–354, April 1994.

[BH93] Uwe Baake and Sorin A. Huss. Object-oriented representation, analysis and scheduling of signal transition graphs. In IEEE Proc. of the International Symposium on Circuits and Systems, pages 2737–2740, May 1993.

[BH94] Uwe Baake and Sorin A. Huss. Scheduling of signal transition graphs under timing constraints. In IEEE Proc. of the International Symposium on Circuits and Systems, pages 205–208, June 1994.

[BLM+92] R. K. Brayton, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, E. M. Sentovich, K. J. Singh, P. R. Stephan, and A. Sangiovanni-Vincentelli. Sis: A System for Sequential Circuit Synthesis. Memorandum UCB/ERL M92/41, Electronics Research Laboratory, University of California, Berkeley 94720, May 1992.

[Bor89] Geatano Borriello. Synthesis of mixed synchronous/asynchronous control logic. In IEEE Proc. of the International Symposium on Circuits and Systems, pages 762–765, 1989.

[Chu87] Tam-Anh Chu. Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications. PhD thesis, M.I.T. Laboratory for Computer Science, June 1987.

[CL93] T.-A. Chu and C. K. C. Leung. An efficient critical race-free state assignment technique for asynchronous finite state machines. In ACM/IEEE Proc. of the Design Automation Conference, pages 2–6, 1993.

[HM90] Andy Hung and Teresa H.-Y. Meng. Asynchronous self-timed circuit synthesis with timing constraints. In IEEE Proc. of the International Symposium on Circuits and Systems, pages 1126–1130, 1990.

[LKS91] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Algorithms for synthesis of hasard free asynchronous circuits. In ACM/IEEE Proc. of the Design Automation Conference, pages 302–308, 1991.

[LL91] Kuan-Jen Lin and Chen-Shang Lin. Automatic synthesis of asynchronous circuits. In ACM/IEEE Proc. of the Design Automation Conference, pages 296–301, 1991.

[LMBS92] Luciano Lavagno, Cho W. Moon, Robert K. Brayton, and A. Sangiovanni-Vincentelli. Solving the state assignment problem for signal transition graphs. In ACM/IEEE Proc. of the Design Automation Conference, pages 568–572, 1992.

[LS93] L. Lavagno and A. Sangiovanni-Vincentelli. Algorithms for Synthesis and Testing of Asynchronous Circuits. Kluwer Academic Publishers, 1993.

[Mar90] Alain J. Martin. Formal program transformations for vlsi circuit synthesis. In Edsger W. Dijkstra, editor, Formal Development of Programs and Proofs. Addison Wesley, 1990.

[Men90] Teresa H. Meng. Synchronization Design for Digital Systems. Kluwer Academic Publishers, 1990.

[MM93] Chris J. Myers and Teresa H.-Y. Meng. Synthesis of timed asynchronous circuits. IEEE Transactions on VLSI Systems, 1(2):106–119, June 1993.

[MM94] Teresa H. Meng and S. Malik. Asynchronous Circuit Design for VLSI Signal Processing. Kluwer Academic Publishers, 1994.

[RY85] L. Y. Rosenblum and A. V. Yakolev. Signal graphs: From self-timed to timed ones. In Proc. of the International Workshop on Timed Petri Nets, pages 199–206, 1985.

[Van93] Peter Vanbekbergen. Synthesis of Asynchronous Controllers from Graph-theoretic Specifications. PhD thesis, Katholieke Universiteit Leuven, September 1993.

[VGD92] P. Vanbekbergen, G. Goossens, and H. De Man. Specification and analysis of timing constraints in signal transition graphs. In IEEE Proc. of the European Design Automation Conference, pages 302–306, 1992.

[VGL94] P. Vanbekbergen, G. Goossens, and B. Lin. Modeling and synthesis of timed asynchronous circuits. In IEEE Proc. of the European Design Automation Conference, 1994.

[YDN93] Kenneth Y. Yun, David L. Dill, and Steven M. Nowick. Practical generalisations of asynchronous state machines. In IEEE Proc. of the European Design Automation Conference, 1993.

[YS92] Meng-Lin Yu and P. A. Subrahmanyam. A new approach for checking the unique state coding property of signal transition graphs. In IEEE Proc. of the European Design Automation Conference, pages 312–321, 1992.

[YVL93] C. Ykman, P. Vanbekbergen, and B. Lin. Alpha release of the ASSASSIN asynchronous compiler. Technical report, IMEC, 1993.