Specification and Management of Timing Constraints in Behavioral VHDL

Francesco Curatelli, Leonardo Mangeruca, Marco Chirico
DIBE – University of Genova
Microelectronics Lab
Via Opera Pia 11/A, 16145 Genova, Italy
franco@dibe.unige.it

Abstract

In this paper a suitable way to specify and manage timing constraints in behavioral VHDL is described. The problem of timing semantics coherency is addressed and a suitable set of procedures is defined to add timing constraint specification in behavioral VHDL for system synthesis. Then, a proper semantics is described which is able to provide a powerful and flexible management of timing constraints.

1. Introduction

The use of VHDL as behavioral specification language for system synthesis requires the definition of suitable schemes for implementing the correct management of timing constraints.

In VHDL, this management is made difficult by the problem of ensuring the semantics coherency of the specification between simulation and synthesis. So, it is necessary to add specific predefined constructs that make it possible to insert in the original VHDL program the set of real time constraints needed.

To address this problem, in this paper a suitable way to specify and manage timing constraints in behavioral VHDL is described. In Section 2, the problem of timing constraints definition, together with semantics coherency, is addressed. In Section 3, a suitable set of procedures is defined to add timing constraint specification in behavioral VHDL, while in Section 4 a proper semantics is introduced which is able to provide a powerful and flexible management of timing constraints application. In Section 5, some conclusions are outlined.

2. Timing Constraints Definition

The problem of providing timing constraints to a VHDL input description is a fundamental one, because the implementation of any kind of real time system typically requires the definition of minimum and maximum constraints on the execution time of the system; this need is common to any system implementation, either hardware, software, or both. In any case, the presence of timing constraints implies that the code generated during the synthesis task can be guaranteed to have a given execution profile. In this Section, the problem related to the coherency of the timing semantics in VHDL will be addressed. Then, an approach will be described for defining, in a safe way, minimum and maximum timing constraints in a VHDL input specification program.

2.1. Timing Definition Coherency

The general problem of checking the consistency of a specification with timing constraints has been addressed, for instance, in [3] [11] [13]. Instead, concerning the use of VHDL as specification language for synthesis, the problem of defining a synthesizable subset of the whole language has been addressed in [12] [9] [4] [14] [7] [8]

In pure VHDL, the typical algorithmic constructs, composed by high-level sequential statements, can be specified using *processes*, functions and procedures, whereas the timing behavior of a process is determined by *signals* and *wait* statements.

As the language simulation semantics states that all sequential statements of a process are executed in *zero* time until a *wait* statement is reached [16], a strict interpretation of the language timing semantics would prevent any correct synthesis to be done. This is due to the fact that the zero time execution cannot be really achieved. However, the timing behavior of a process is relevant only for the signals which are visible also outside the process, and not for the local *variables* declared inside the process.

Therefore, the functionality of a process is not altered provided that the signals are only read and/or updated at the moments in time specified by the simulation semantics; all other statements that operate on variables can be executed at any time, given that the ordering imposed by the data dependencies is mantained.

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

To ensure that the VHDL simulation timing semantics be satisfied also by the synthesized system, several approaches have been followed [1] [2] [10] [7] [15] [17]. In particular, in [7] two methods are described for the synthesis of concurrent processes, which preserve a partial ordering relation of operations on signals and ports during synthesis. In the first method one is allowed to use freely signals and wait statements in the specification (*unrestricted model*); semantics coherency is achieved by adopting a strict control of process synchronization for signal updating:

- processes update signal values only when all of them are actually executing wait statements;
- synthesis produces systems controlled by either a single FSM or several FSM's working synchronously.

This strict control of process synchronization for signal updating is not needed in the second method (*reduced-synchronization model*), where process interaction does occur only through signal assignment and wait statements, and VHDL processes communicate only through a synchronous message-passing mechanism, in which:

1) communications channels are provided by VHDL signals, and signal assignment is only done with send/receive commands;
2) send/receive commands have the syntax of subprogram calls, and implement a blocking message-passing mechanism, in which the process waits for the actual activation of the exchange operation.

2.2. Timing Definition Problem

Aim of our work has been to define minimum, maximum, and range timing constraints in the VHDL input behavioral description. The input specification is constituted by VHDL processes, communicating internally through a shared memory mechanism and externally through a message-passing communication mechanism. Synchronization between processes is achieved through a blocking communication channel. To meet the semantic coherency of the input description a restriction is stated on the use of signals; i.e., within a process, VHDL signals are only set and used by message-passing procedures [6].

A useful specification requirement concerns the fact that timing constraints should be actually defined with respect to couples of specific points of the input description. This is directly related to the typical approach to the definition of timing constraints, which consists in defining minimum and maximum latency times between two points of the specification. Therefore, we have not chosen to use directly wait statements for the definition of timing constraints. In fact, in VHDL each wait statement states a waiting time with respect to the previous wait statement in the execution sequence, which in case of branches in the flow of control is unknown at compile time.

So, we have adopted another solution in which min/max timing constraints can be specified between specific procedure calls in the input specification program, always known at compile time. Moreover, in this way timing constraints may overlap, while in VHDL timing constraints can overlap only as a result of branches in the flow of control.

This approach has been introduced by Eles et al. in [8], where the main target is the specification of timing constraints in behavioral VHDL for high-level synthesis so that simulation to synthesis correspondence can be ensured for a single process and across processes. This is obtained through the definition of a set of specific procedures which make it possible to assert *exact*, *minimum*, *maximum* and *range* timing constraints between two points of the input VHDL specification. In particular:

1) anchor (anchor_lab): specifies the starting point of the first following timing constraint that actually uses anchor_label;
2) [exact/min/max/range]_time (constr_def, anchor_lab): an [exact/min/max/range] timing constraint of value constr_def is specified between this point of the program and the point where anchor_label was lastly used as a parameter;

More specifically, simulation/synthesis correspondence is automatically ensured for the model with reduced synchronization [7], where process interaction does occur only through signal assignments and wait statements, and VHDL processes communicate only through a synchronous message-passing mechanism.

3. Timing Constraints Procedures

In our work, the definition and use of a set of specific procedures has made it possible to specify *minimum*, *maximum*, and *range* latency constraints, denoted by T_i^m, T_i^M, $T_i^r = (T_i^m, T_i^M)$, respectively. Each timing constraint is defined with respect to a *starting point* $P^s_{T_i}$ and an *end point* $P^e_{T_i}$, so that during the execution, the execution times are ordered, i.e.:

$$t_{ex}(P^s_{T_i}) \leq t_{ex}(P^e_{T_i})$$

It is worth noting that an exact latency constraint is not specifically provided, as it is possible to implement it as a particular range constraint with $T_i^m = T_i^M$.

The actual specification of the timing constraints is obtained through the use of couples of specific procedure calls. Each procedure defines one of the two points (starting, end) in the VHDL input specification with respect to which each timing constraints must be stated.

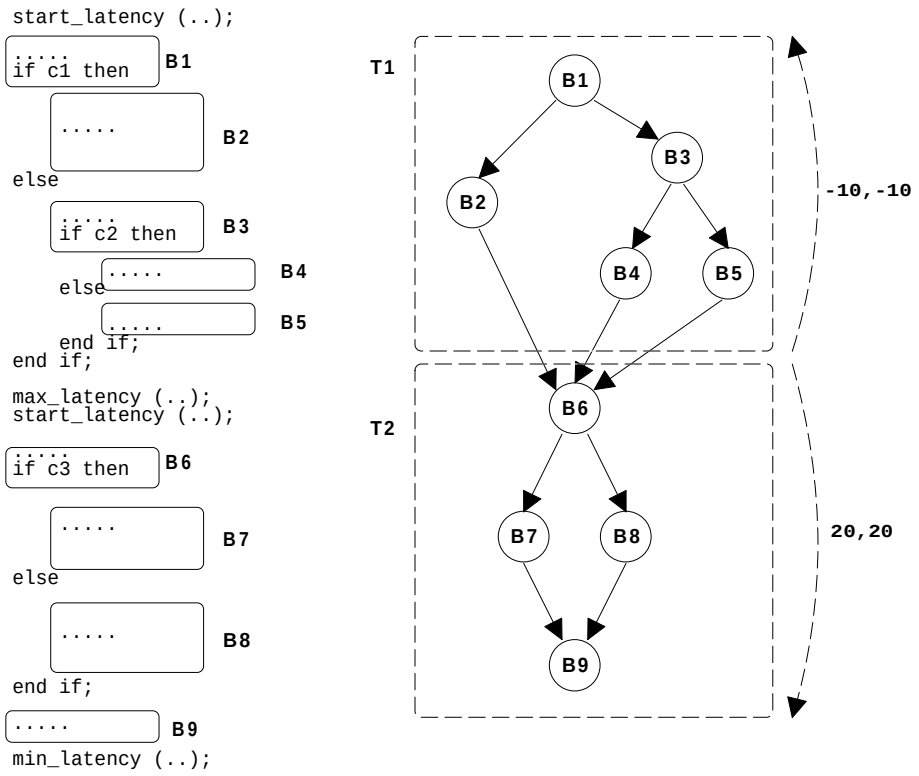

Figure 1. Timing constraints management

So, each timing constraint assertion involves two procedure executions: $(\Pi^{P^s_{T_i}})$, associated to the starting point, and $(\Pi^{P^e_{T_i}})$, associated to the end point; only $(\Pi^{P^e_{T_i}})$ actually contains the information about the kind of timing constraint (min/max/range) defined, while $(\Pi^{P^s_{T_i}})$ acts as an *anchor* point for the definition. This definition makes it possible to characterize situations in which more timing constraints are expressed from different end points to the same starting point; formally:

1) $(\Pi^{P^e_{T_i}}) = (\Pi^{P^e_{T_j}}) \Leftrightarrow P^e_{T_i} = P^e_{T_j} \Leftrightarrow T_i = T_j$

2) $(\Pi^{P^s_{T_i}}) = (\Pi^{P^s_{T_j}}) \Leftrightarrow P^s_{T_i} = P^s_{T_j}$
$\Longrightarrow T_i = T_j \mid T_i \neq T_j$

In other words, it is not possible to define chains of timing constraints starting from a single anchor and using the same label. This restriction has the following advantages:

1) timing constraints are specified completely in the most natural and simplest way (i.e., as a restriction stated between two specific points of the input description);

2) it is possible to provide a more robust definition of timing constraints, not subject to errors of definition due to possible changes (addition and/or deletion) of the timing constraints within the scope of the label.

The procedures used for defining timing constraints are the following ones (Figure 1):

- $(\Pi^{P^s_{T_i}})$: start_latency(label) defined at the point $P^s_{T_i}$ in the program from which the related latency constraints have to be considered (label $\in \mathbf{N}$);
- $(\Pi^{P^e_{T_i}})$: min_latency(label, range) defined at the point $P^e_{T_i}$ in the program where the minimum latency constraint has to be verified, and providing the range of time values in which the latency must stay (range = L^m, L^M; label, L^m, $L^M \in \mathbf{N}$);
- $(\Pi^{P^e_{T_i}})$: max_latency(label, range) defined at the point $P^e_{T_i}$ in the program where the maximum latency constraint has to be verified, and providing the range of time values in which the latency must stay (range = $|U^m|$, $|U^M|$; label, $-U^m$, $-U^M \in \mathbf{N}$);
- $(\Pi^{P^e_{T_i}})$: range_latency(label, range) defined at the point $P^e_{T_i}$ in the program where the range latency constraint has to be verified, and providing the range of time values in which the latency must stay (range = D^m, D^M; label, D^m, $D^M \in \mathbf{N}$);

The VHDL input description is translated (in two steps) into an internal form suitable for the execution of the synthesis tasks (*synthesis representation model*). This internal

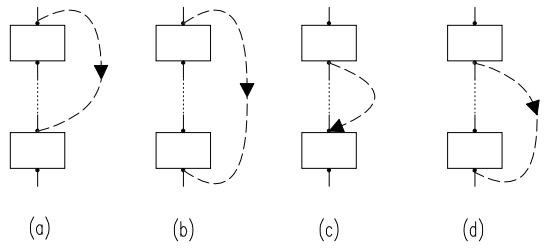

Figure 2. Min latency constraints

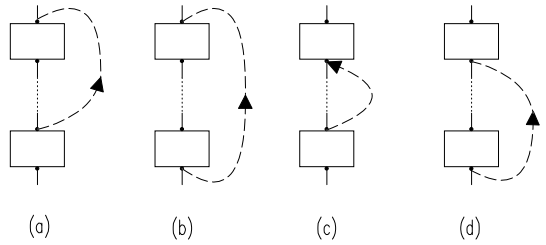

Figure 3. Max latency constraints

form of representation (which will be described in detail in a further paper) is basicly constituted by control graphs in which nodes correspond to deterministic computation entities, such as *basic blocks*, while edges define dependencies between nodes, namely, each kind of dependency is related to a different kind of edge: *control edge*, *data edge*, *timing edge*, the last one related to the definition of timing constraints. The internal form contains at least one *global control-flow graph* (*G-CFG*), in turn composed by a set of partially disjoined *behavioral control-flow graphs H-CFGs*, each one corresponding either to a VHDL process (*P-CFG*), or to a subprogram (*U-CFG*) [5]. In terms of this graph specification our representation scheme makes use of timing edges τ, namely: forward edges, with positive weights, for denoting minimum timing constraints (τ_{T^m}, Figure 2) and backward edges, with negative weights, for denoting maximum timing constraints (τ_{T^M}, Figure 3); a couple of forward and backward edges can be suitably used for denoting range timing constraints ($\tau^f_{T^m}$, $\tau^b_{T^M}$). This means that:

- for a minimum timing constraint T^m_i:
  $head\,(\tau_{T^m_i}) \leftrightarrow P^e_{T^m_i};\ tail\,(\tau_{T^m_i}) \leftrightarrow P^s_{T^m_i}$
- for a maximum timing constraint T^M_j:
  $head\,(\tau_{T^M_j}) \leftrightarrow P^s_{T^M_j};\ tail\,(\tau_{T^M_j}) \leftrightarrow P^e_{T^M_j}$
- for a range timing constraint T^r_k:
  $head\,(\tau^f_{T^r_k}) \leftrightarrow P^e_{T^r_k};\ tail\,(\tau^f_{T^r_k}) \leftrightarrow P^s_{T^r_k}$
  $head\,(\tau^b_{T^r_k}) \leftrightarrow P^s_{T^r_k};\ tail\,(\tau^b_{T^r_k}) \leftrightarrow P^e_{T^r_k}$

A worth noting characteristic of our approach is that a timing constraint can be defined with respect to the *entry* or *exit points* of the nodes, which are in turn put into correspondence with the enabling and termination condition of the nodes (Figures 2,3). This makes it possible to define timing constraints in a very powerful and flexible way.

The following restrictions have been posed to the definition of timing constraints between different points of the VHDL input specification, concerning the points where the starting and end points of the timing constraint can be actually placed.

1) The definition of a timing constraint is only allowed between a couple of points strictly belonging to the same VHDL process or to the same subprogram; in the internal form used in our work, this means that both points must belong to the same H-CFG and at the same hierarchical level (*intra-graph constraint*).

2) The definition of a timing constraint is not allowed between the *then* and *else* scopes of a conditional statement, and crossing the body of a whichever loop.

4. Timing Constraints Semantics

A main problem consists in defining a proper semantics concerning the use of the above procedures to define timing constraints. In particular, we have to decide which basic block at the synthesis level the starting and end point of a given timing constraint specified at VHDL level must be bound to; the problem is complicated by the freedom to define at the synthesis level timing edges with respect to entry or exit point of the basic block [5].

The trivial approach would simply require that each timing constraint put a strict constraint on the execution of VHDL statements; namely: if an operation O_i is defined, in the VHDL input specification, before a timing constraint procedure $P^{e|s}_{T_j}$, this would automatically require that $t_{ex}(O_i) \leq t_{ex}(P^{e|s}_{T_j})$. This simple choice is likely to produce bad results, since only communication through signals and the presence of data dependencies really put constraints on the execution of the system.

Instead, our approach is based on the definition of an *inclusive semantics*; the name derives from the way the timing constraints procedures are bound to entry or exit points of nodes at the synthesis level.

Two basic situations will be treated: 1) a purely operational description, in which no inter-graph communication or synchronization does occur, and 2) a mixed description, in which at least one communication procedure with synchronization is contained in the VHDL description.

In the first case the timing constraint is added to the description only to provide a suitable control on the execution of different parts of the input specification. In this sense no real constraint would really avoid assignment operations

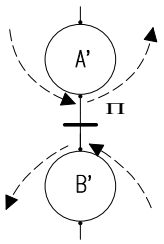

Figure 4. Inclusive semantics without communication

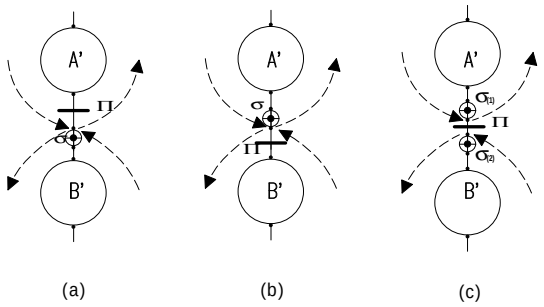

Figure 5. Inclusive semantics with communication

on variables to be freely moved across the points where the timing constraint procedures have been inserted. This is directly related to the semantics of VHDL, which states that operations on variables do not really contribute to the timing of the process, and to the elimination of the use of signals except inside the communication procedures.

The consequence is that the basic block management can potentially perform any kind of code optimization and movement suitable for the improvement of the synthesis task, and in this sense the timing constraint itself is meaningful only at the synthesis level, where, in case of code optimization, the operational composition of the nodes has been changed with respect to that provided as input. So, a timing constraint specification will at this point operates on the actual configuration of the nodes constituting the graph (this will obviously hold only for the system configuration we are considering as a possible solution).

In this case the application of the inclusive semantics can be summarized as follows (see Figure 4).

Let A_V and B_V be two sets of statements at VHDL level suitable to be mapped directly to basic blocks A and B at the synthesis level, and let P_T the starting (or end) point where a timing constraint procedure call has been inserted, provided that the statement calling Π^{P_T} be the only one separating A_V and B_V, i.e.:

$$t_{ex}(A_V) \leq t_{ex}(P_T) \leq t_{ex}(B_V)$$

Moreover, let A' and B' be the basic blocks actually managed at the block level, after code optimization of A and B, A' being executed before B' ($A \rightarrow B$); the timing constraint given by τ_T will then refer to one of these blocks according to the following rules:

1) $P_T = P_T^s,\ A' \rightarrow B' \implies P_T \leftrightarrow \text{entry}_{B'}$

2) $P_T = P_T^e,\ A' \rightarrow B' \implies P_T \leftrightarrow \text{exit}_{A'}$

In other words, according to the inclusive semantics a timing constraint is considered as a condition embracing the set of node operations contained within.

In the second case the timing constraint will actually influence the timing characteristics of the process, since communication procedures are actually constrained not to cross timing constraint procedures.

So, also in this case the basic block management can potentially perform code optimization and movement suitable for the improvement of the synthesis task, provided that after the transformations:

1) the operational semantics is maintained with respect to the data-flow information (including the data dependencies deriving from the presence of receive procedures);

2) the send/receive communication operations do maintain the original relative positions in the control-flow, and do not cross timing constraints specifications.

In this case the application of the inclusive semantics can be summarized as follows.

Let A_V and B_V be two sets of statements at VHDL level suitable to be mapped directly to basic blocks A and B at the synthesis level, let σ_V be a blocking communication operation, and let P_T be the point (starting or end) where a timing constraint procedure call has been inserted, provided that the statement calling Π^{P_T} and σ_V be the only ones separating A_V and B_V, i.e.:

$$t_{ex}(A_V) \leq t_{ex}(P_T \mid \sigma_V) \leq t_{ex}(B_V)$$

Moreover, let A' and B' be the basic blocks actually managed at the block level, after code optimization of A and B, A' being executed before B' ($A \rightarrow B$), and let σ be the *event* or *emit node* related to σ_V in the G-CFG. The timing constraint given by τ_T will then refer to one of these σ nodes according to the following rules (Figure 5(a-b)):

1) $P_T = P_T^s,\ A' \rightarrow \sigma \rightarrow B',$
$t_{ex}(P_T) \leq t_{ex}(\sigma_V) \implies P_T \leftrightarrow \text{entry}_{\sigma}$

2) $P_T = P_T^s,\ A' \rightarrow \sigma \rightarrow B',$
$t_{ex}(P_T) \geq t_{ex}(\sigma_V) \implies P_T \leftrightarrow \text{exit}_{\sigma}$

3) $P_T = P_T^e,\ A' \rightarrow \sigma \rightarrow B',$
$t_{ex}(P_T) \leq t_{ex}(\sigma_V) \implies P_T \leftrightarrow \text{entry}_\sigma$

4) $P_T = P_T^e,\ A' \rightarrow \sigma \rightarrow B',$
$t_{ex}(P_T) \geq t_{ex}(\sigma_V) \implies P_T \leftrightarrow \text{exit}_\sigma$

In other words, according to the inclusive semantics, when a synchronization node is close to a timing constraint specification the related point is anyway bound to that node.

The same scheme is adopted to choose the right bound when the timing specification is placed between two synchronization operations. In this case the application of the inclusive semantics can be summarized as follows (Figure 5(c)).

Let $\sigma_V^{(1)}$ and $\sigma_V^{(2)}$ be two blocking communication operations close to each other and let P_T the point (starting or end) where a timing constraint procedure call has been inserted, provided that $t_{ex}(\sigma_V^{(1)}) \leq t_{ex}(\sigma_V^{(2)})$. The timing constraint given by τ_T will then refer to one of these nodes according to the following rules:

1) $P_T = P_T^s,\ A' \rightarrow \sigma^{(1)} \rightarrow \sigma^{(2)} \rightarrow B',$
$t_{ex}(\sigma_V^{(1)}) \leq t_{ex}(P_T) \leq t_{ex}(\sigma_V^{(2)})$
$\implies P_T \leftrightarrow \text{entry}_{\sigma^{(2)}}$

2) $P_T = P_T^e,\ A' \rightarrow \sigma^{(1)} \rightarrow \sigma^{(2)} \rightarrow B',$
$t_{ex}(\sigma_V^{(1)}) \leq t_{ex}(P_T) \leq t_{ex}(\sigma_V^{(2)})$
$\implies P_T \leftrightarrow \text{exit}_{\sigma^{(1)}}$

5. Conclusions

In this paper we have addressed the problems concerning the specification and management of timing constraints in behavioral VHDL. This has been done by first discussing in detail the need for semantics coherency in the input description, and then introducing a set of procedures able to define a timing constraint specification in behavioral VHDL for system synthesis. The management of each timing constraint is done, within a process, through a couple of procedures which make it possible to set minimum, maximum, and range latency constraints between two points of the VHDL input specification. Finally, a proper semantics has been introduced which is able to provide a powerful and flexible management of timing constraints application; in particular, the adoption of this interpretation of the timing constraints specification makes it possible to explore the potential solution space being constrained only by the communication and data dependencies contained in the input specification.

The approach has been described with a direct reference to the internal form of representation used at the synthesis level. The input specification is supposed to be constituted by a set of processes, communicating internally through a shared memory mechanism and externally through a point-to-point, blocking or non-blocking, message-passing communication mechanism. Synchronization between processes is achieved through a blocking communication channel. A restriction is stated on the use of signals to meet the semantic coherency of the input description.

Acknowledgements

This work has been partially supported by Esprit BRA Project 9138 (CHIPS).

References

[1] R. A. Bergamaschi, A. Kuelhlmann, "A System for Production Use of High-Level Synthesis", *IEEE Proceedings on VLSI Systems*, Vol. 1, No. 3, September 1993.

[2] J. Biesenack et al., "The Siemens High-Level Synthesis System CALLAS", *IEEE Transactions on VLSI Systems*, Vol. 1, No. 3, September 1993.

[3] R. Camposano, A. Kunzmann, "Considering Timing Constraints in Synthesis from a Behavioral Description", *Proceedings of ICCD*, October 1986.

[4] R. Camposano, L. F. Saunders, R. M. Tabet, "VHDL as Input for High- Level Synthesis", *IEEE Design and Test of Computers*, March 1991.

[5] F. Curatelli, L. Mangeruca, M. Chirico, "Specification of a Two-Level Intermediate Form for System Synthesis", *DIBE Technical Report*, June 1996.

[6] F. Curatelli, L. Mangeruca, M. Chirico, "Specification of Data Communication and Timing Constraints in Behavioral VHDL", *DIBE Technical Report*, June 1996.

[7] P. Eles et al., "Synthesis of VHDL Concurrent Processes", *Proceedings of EURO-VHDL*, September 1994.

[8] P. Eles et al., "Timing Constraint Specification and Synthesis in Behavioral VHDL", *Proceedings of EURO-VHDL*, September 1995.

[9] W. Glunz, G. Umbreit, "VHDL for High-Level Synthesis of Digital Systems", *Proceedings of the 1st EURO-VHDL*, 1990.

[10] A. A. Jerraya, I. Park, K. O'Brien, "AMICAL: An Interactive High Level Synthesis Environment", *Proceedings of EDAC*, 1993.

[11] D. Ku, G. De Micheli, "Relative Scheduling under Timing Constraints", *Proceedings of the 27th DAC*, June 1990.

[12] J. Lis, D. Gajski, "Synthesis from VHDL", *Proceedings of ICCD*, October 1988.

[13] J. Nestor, G. Krishnamoorthy, "SALSA: a New Approach to Scheduling with Timing Constraints", *IEEE Transactions on Computer-Aided Design*, Vol. 12, No. 8, August 1993.

[14] A. Postula, "VHDL Specific Issues in High Level Synthesis", *Proceedings of EURO-VHDL 91*, September 1991.

[15] F. Vahid et al., "A Transformation for Integrating VHDL Behavioral Specification with Synthesis and Software Generation", *Proceedings of EURO-VHDL*, September 1994.

[16] "IEEE Standard VHDL Language Reference Manual - IEEE Std 1076-1987", *IEEE*, 1987.

[17] N. Wehn et al., "Scheduling of Behavioral VHDL by Retiming Techniques", *Proceedings of EURO-VHDL*, September 1994.