Published in IEEE Proceedings of the European Design Automation Conference (EURO-DAC)
with EURO-VHDL '93, 494-499, 1993
which should be used for any reference to this work 

# High-Level Modeling using Extended Timing Diagrams

## A formalism for the behavioral specification of digital hardware

Philippe Moeschler, Hans Peter Amann, Fausto Pellandini

Institute for Microtechnology, University of Neuchâtel, Rue de Tivoli 28, 2003 Neuchâtel, Switzerland

### Abstract

*This paper describes the principles of high level modeling of digital hardware circuits using the Extended Timing Diagrams (ETD) formalism which adds conditions, events, action expressions and particular constraints to traditional timing diagrams. Hierarchy and concurrency are integrated too such that a full top-down design becomes possible, enhancing in the same time the readability. While for simulation purposes, the implementation of the formalism generates behavioral VHDL (VHSIC Hardware Description Language) models, a dedicated high-level translator generates VHDL code for synthesis. Both the ETD formalism and its implementation are part of MODES, a more complex modeling expert system including complementary editors.*

*Keywords: VHDL-generation from high-level specifications, VHDL and CAD framework developments*

## 1: Introduction

ETD is part of the MODES Modeling Expert System [1] project aimed at supporting hardware engineers in their efforts to create behavioral models of digital devices using VHDL [7]. The MODES CASE tool provides various editors, for truth-tables and finite state machines, etc. It also includes a selection and a specification tool for the reuse of existing models or parts of them and the instanciation of existing generic models respectively. Furthermore, a rule based system, under development, supports the user through a guided man/machine dialogue for the selection of convenient generic models and performs on-line verification of completeness and consistency of specifications introduced.

The aim of this paper is to present a method that allows the creation of bus oriented models (e.g., in VHDL) from annotated timing diagrams. Since hardware engineers are familiar with such circuit representations, the ETD formalism aims to stay close to this way of thinking.

Traditional timing diagrams do not effectively describe the behavior of a circuit, or its functionality. They only represent signal wave-forms at the I/O pins. Therefore we introduce the notion of *extended timing diagrams. ETD's* include wave-form descriptions and action expressions that may be attached through constraints to events or conditions.

It is not necessary to use the whole wave-form to describe the functionality of the circuit: the entry point of the timing diagram is arbitrary and only some parts of the diagram contain useful information. Obviously the wave-forms show the response of the circuit to some stimuli applied to its inputs. This fact is illustrated in the example of figure 1 which represents the reset and the load processes of a counter.

At a first glance, the two processes are independent, their relative positions in the diagram have no importance. The useful information is attached to the events CLR'rising and LOAD'falling. Nevertheless, the information is not complete, e.g., CLR has priority on LOAD, implicit information the user has to consider.

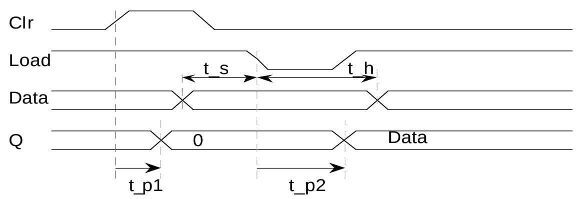


**Figure 1: Datasheet example**

With the ETD approach, we resolve this ambiguity through the introduction of particular constraints that are used to link events and conditions. Furthermore, we add hierarchy and concurrency, allowing for more readable representations.

### 1.1: Related previous work

Harel's papers [9,10] and the article of Vahid, Narayan and Gajski [11] introduce to the field of reactive system design, using hierarchy and concurrency. For the specific domain of timing diagrams, the present work has been based on the following four papers. Borriello described the possibility to model hardware using timing diagrams [2].

Amon, Borriello and Séquin introduce a similar approach to timing diagram modeling [5]: behavior and timing constraints are described separately. Another approach has been developed by Dufresne, Khordoc and Cerny [3], introducing the notion of hierarchy and concurrency in timing diagrams. The timing diagrams were described with dynamic structures in VHDL. Finally a major source of information for this paper is Schlör [12] who introduces a possible approach of system design and verification using timing diagrams, but does not support a full graphical representation of the behavior of the reactive system.

### 1.2: Organization of the paper

The paper first presents the important definitions and terms of the formalism. Then, the notions of concurrency and hierarchy are introduced. In paragraph 3, we develop the semantics of ETD's. Next, the evaluation scheme (§4) and the principles of high-level simulation are introduced (§5), followed by timing constraint checks (§6). In paragraph 7, the VHDL code generation is illustrated with an example, presenting the path from ETD representation over the VHDL code generated to a net-list created with a synthesis tool. Finally, the results are discussed.

## 2. Terms and definitions

A *design* represents the top hierarchical level of the system: it is composed of various *diagrams* at different levels of hierarchy. Each diagram includes one (no concurrency) or more (concurrency) *sub-diagrams*, standing each for a graphical representation of an ETD. An ETD contains one or more *events* and *conditions* that trigger *action expression* in association with different classes of *constraints*. If an action expression is not linked to any event or condition through a constraint, it is considered to be a *default action expression* for the wave-form. This feature can be used to initialize or to define implicit behavior of a signal when a particular sub-diagram is activated.

With the help of constraints, conditions and events are kept in relation. We distinguish the following types of constraints: *simultaneous constraints* define a condition of simultaneity between two events or conditions. *Conflict constraints* disallow two events or conditions to be active at the same time. Forward constraints are used to build causal relations between signals as well as hold-time or minimum/maximum pulse width checks. *Backward constraints* can be used to define timing constraints such as setup-time. Finally, it is possible to define loops with a particular kind of backward constraint, the *loop constraint*. For example, this feature will be useful when waiting for a sequence of n falling edges of the clock. The different classes of constraints are shown in figure 2.

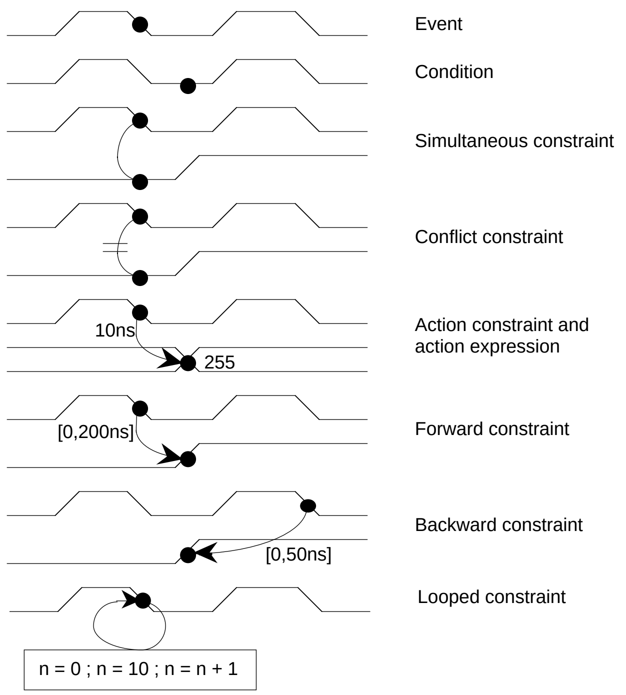


**Figure 2: Event and constraint classes**

Hierarchy and concurrency are important in hardware modeling: for simulation, synthesis and test. These two concepts enable a better readability of the design and therefore decrease significantly the debugging time.

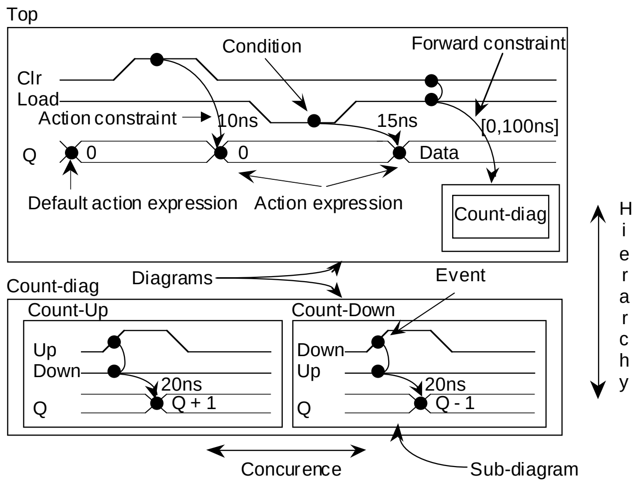


**Figure 3: Terms and definitions**

### 2.1: Concurrency

This feature reflects the fact that electronic devices are composed of coupled modules that together perform a given functionality. In the ETD formalism, concurrency is implemented at two levels. First, sub-diagrams contained in the same diagram are by definition concurrent. Second, each wave-form of a sub-diagram may contain several independent events, that are considered to be concurrent too.

### 2.2: Hierarchy

Through this concept, the condition for an event at a given level of hierarchy can be refined at a lower level of hierarchy in a *hierarchical diagram*, enabling top-down modeling and a better readability of the design. The example of figure 3 above shows the different ETD concepts defined before. The hierarchy concept implies the *override children* notion for the evaluation scheme.

This feature enables to bypass the evaluation of a child diagram if, at the higher level of hierarchy, the simultaneous constraint or conflict constraint linked to the diagram is not active. In the case of forward constraint, the child diagram becomes active as soon as the corresponding constraint has been activated.

### 2.3: Chain concept

For the evaluation of the activity scheme, we must introduce a new term: a *chain*. This object represents a chain of event/conditions linked with constraints.

A starting event/condition is the first occurring event/condition in a chain: a notion introduced to solve the dependencies between two levels of hierarchy. The forward constraint in the higher level of hierarchy is implicitly in relation with every starting events/conditions of every sub-diagrams at the lower hierarchical level. In the special case of a simultaneous constraint or conflict constraint, the constraint at the higher level is implicitly linked to every events/conditions at the lower level.

An ending event/condition in a chain is the event/ condition of a chain that has no forward constraints starting from it.

## 3: Semantics of ETD

For the semantic analysis of the design, we first define the following sets of objects.

| | |
|---|---|
| *I:* | *set of input ports* |
| *O:* | *set of output ports* |
| *Event:* | *set of events* |
| *Cond:* | *set of conditions* |
| *Const:* | *set of constraints* |
| *Action:* | *set of action expressions* |
| *Chain:* | *set of chains* |
| *Diag:* | *set of diagrams* |
| *Sub-diag:* | *set of sub diagrams* |

A design *des* can be represented with the following semantic:

*des = (I, O, top_diagram), top_diagram* $\in$ *Diag*

Similarly, we can define a diagram, a sub-diagram and a chain.

*diag = (Sub-diag)*

*sub-diag = (Chain)*

*chain = (Events, Cond, Const, Action, Diag)*

An event *e* is defined either as a couple <*x*,*value*> where $x \in I$ and *value* is the new value of signal *x* or as a single value <*x*> to test any changes of the signal *x*. A condition *c* can also be represented with a couple <*x*,*value*> where *value* represents here the current value of signal *x*. An action expression *a* is defined with two values <*x*,*expr*>, where $x \in O$ and *expr* is an expression that will determine the new value of signal *x*.

Simultaneous constraints *sim(u,v)* and conflict constraints *conf(u,v)* define simultaneity relations between *u* and *v* where $u \in$ *Events* $\cup$ *Cond* and $v \in$ *Events* $\cup$ *Cond* $\cup$ *Diag*. A forward constraint *for(e,e',t_min,t_max)* can only be defined between two events or an event and a diagram (hierarchy) and implies that the event *e'* (or any starting event of the diagram) occurs at time $t \in [t(e)+t\_min, t(e)+t\_max]$.

Similarly, a backward constraint *back(e,e',t_min,t_max)* means that event *e* has occurred at time $t \in [t(e')-t\_max, t(e')-t\_min]$. An action constraint *act(u,a,t_prop)* is used to define a propagation delay *t_prop* between a triggering event/condition *u* and an action expression *a*.

Finally, a loop constraint *loop(e,e',init,exit,incr)*, can be considered as a special case of the backward constraint and is used to implement counters, where *init* is the initialization value of the counter, *exit* is the exit condition and *incr* is the increment expression.

Forward constraints can be chained to define sequences of events/conditions to be observed before an action is initiated. On the contrary, it is not possible to do the same with backward constraints because the system is supposed to be causal.

## 4: Evaluation scheme

A verification phase precedes the evaluation of the design: consistency checks, syntax and semantic checks, uniqueness of default action expressions, etc. Obviously, erroneous specifications cannot be corrected and lead to malfunctions of the models.

During the evaluation of the ETD, priority is given first to simultaneous/conflict constraints, next to loop constraints and finally to forward sequential constraints.

### 4.2: Activity

The notion of *activity* can be applied to diagrams, sub-diagrams, events, conditions, actions and constraints. A diagram or a sub-diagram is said to be *active* if its related constraint is active. By default, the top diagrams and sub-diagrams of the design are always active.

During simulation, only active diagrams - and by extension sub-diagrams - are evaluated, improving the speed of simulation in an important way (see overriding children). Formally, the activity of the different objects used in ETD formalism can be defined as follows:

act(des) = true
act(top_diag) = true
act(diag) = (act(diag) + Σact(for(e,diag))) *Πact(sim(u,diag))*Πact(conf(u,diag))
act(sub_diag) = act(diag)
act(chain) = (act(start_event) + act(chain)) * not act(end_event) * act(sub_diag)
act(e<x,value>) = act(sub_diag) * Πact(sim(u,e)) * Πact(conf(w,e)) * (Σact(for(v,e)) +Σact(loop(e',e)) + not act(chain)) * (x = value) * (x'event)
act(e<x>) = act(sub_diag) * Πact(sim(u,e)) * Πact(conf(w,e)) * (Σact(for(v,e)) + Σact(loop(e',e)) + not act(chain)) * (x'event)
act(c<x,value>) = act(sub_diag) * (Πact(sim(u,e)) + Πact(conf(v,e)) + not act(chain)) * (x = value)
act(a<x,value>) = act(sub_diag) * Σact(action(u,a))
act(action(u,a,t_p)) = act(u) after t_p
act(for(e,e') = act(sub_diag) * [act(for(e,e') + act(e)] * not act(e') * not(Σact(loop(e,e")))
act(loop(e,e') = act(sub_diag) * [act(loop(e,e') + act(e)] * not act(e') * not exit
act(back(e,e') = act(sub_diag) * [act(back(e,e') + act(e)] * not act(e')
act(sim(u,v)) = act(sub_diag) * act(u) * act(v)
act(conf(u,v)) = act(sub_diag) * (act(u) xor act(v))

Finally, the simultaneous and conflict constraints have the following property:

act(sim(u,v)) = act(sim(v,u))
act(conf(u,v)) = act(conf(u,v))

## 5: High-Level simulation

In order to verify as soon as possible the correct functionality of a design, a simulation must be possible at the level of abstraction used for specification. Thus a simulation tool will be provided at the ETD level.

The simulation process runs top-down through the hierarchy. At each level, all "next" events of active chains are evaluated. If they are true, the respective constraints and action expressions are executed. For the case where an active constraint is linked to a hierarchical diagram, the diagram is activated. Default action expressions are executed upon entry into a sub-diagram prior to the evaluation of the events. Figure 4 shows a possible representation of this recursive evaluation process.

The horizontal axis represents the simulation time. A *time step* represents the time between two events. The *resolution* is the minimum amount of time required between two events. While the *concurrency stepping* axis illustrates the evaluation of the concurrent events at a

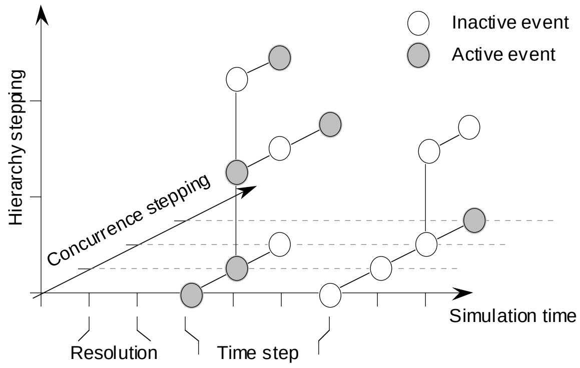


**Figure 4: Simulation process**

given simulation time, *hierarchy stepping* stands for iterative evaluation of hierarchical levels.

ETDs have been introduced for the specification of digital devices. Nevertheless, they can also be used for the representation of the simulation results. Obviously, the structuring feature of hierarchy disappears: the ETDs are flattened.

## 6: Timing constraint checks

As introduced in the paragraph dealing with the semantics, the ETD formalism includes several timing constraint checks: setup, hold and minimum pulse width.

Note that if the timing constraints of forward and backward constraints are violated, the corresponding constraint is not deactivated, but a warning message is displayed during the simulation process.

According to Liu and Pawlak [4], the implementation of the timing constraint checks in VHDL can be realized with concurrent procedure calls, described in a VHDL package called ETD_Standard. This method offers two major advantages: the same procedure can be used to define different timing constraints and the VHDL code is hidden to the programmer.

## 7: Translation into VHDL

The translation into simulatable and synthesisable VHDL is done according to the evaluation scheme described before. The different activity algorithms are translated into a finite state machine structure, preserving hierarchy and concurrency of the ETD design.

Basically, the timing diagrams are asynchronous. Therefore, an explicit clock signal is not necessary for the evaluation of the design. For simulation purposes, this asynchronous mode is preserved. For synthesis in turn, the asynchronous structure cannot always be used due to the limitations of the currently available commercial synthesizers: usually, one single signal per VHDL process can be tested for a triggering edge. Hence, if multiple edges have to be checked, the circuit must be transformed into a synchronous finite state machine. Nevertheless, the functionality can be preserved under the condition that between two triggering events, the evaluation of the complete circuit can be performed: the clock period must be sufficiently small with respect to the input event rate.

An explicit synchronous running mode can be selected by specifying the clock signal of the circuit for both simulation and synthesis VHDL code generation: implicitly, each signal is evaluated at each active edge of the clock.

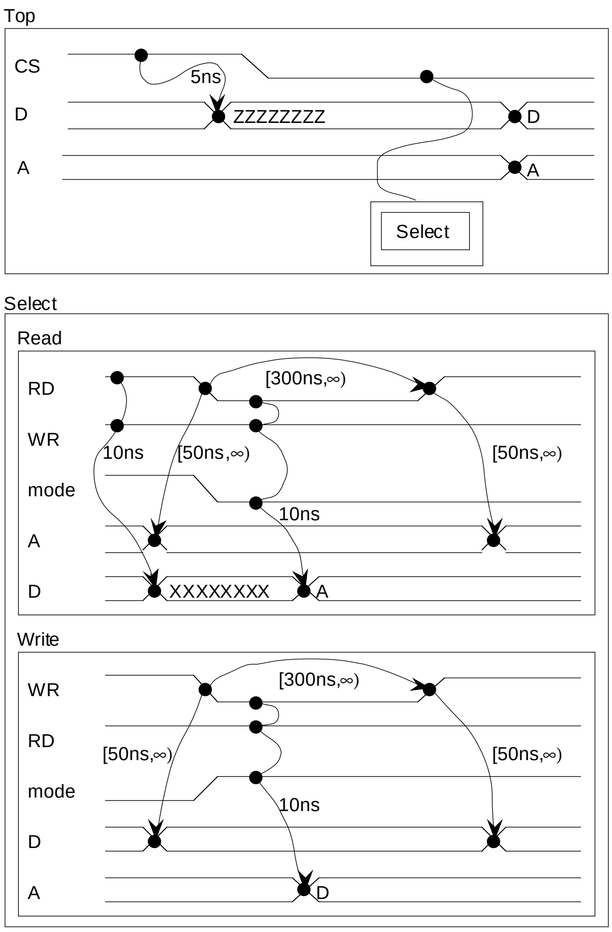


**Figure 5: PPI design**

## 8: Discussions

The ETD formalism is well suited for asynchronous bus functional models, since it is easy to define timing constraints between asynchronous signals. Nevertheless, it remains possible to design other types of digital reactive systems. However, the relative simplicity of the action expression authorized by the formalism may cause some difficulties to implement complex behavior of output signals.

### 8.1: Example

In the following example, we show the description of a programmable peripheral interface PPI. The *mode* signal determines the direction of the I/O port A. Figure 5 shows the ETD graphical representation, table 6 the corresponding simulatable VHDL code.

```
Library work;
Use work.ETD_Standard.all;
Entity PPI is
  Port (
   CS      : in STD_ULOGIC;
   RD      : in STD_ULOGIC;
   WR      : in STD_ULOGIC;
   mode  : in STD_ULOGIC;
   D         : inout STD_LOGIC_VECTOR(7 downto 0);
   A         : inout STD_LOGIC_VECTOR(7 downto 0);
  );
End PPI;
Architecture Behavioral of PPI is
Begin

   Check_backward(A'changing, RD'falling,
  50 ns,Time'High,WARNING,"Backward check RD->A");
   Check_forward(RD'falling,RD'rising,
  300 ns,Time'High,WARNING,"Forward check RD->RD");
   Check_forward(A'changing,RD'rising,
  50 ns,Time'High,WARNING,"Forward check RD->A");
   Check_backward(D'changing, WR'falling,
  50 ns,Time'High,WARNING,"Backward check WR->D");
   Check_forward(WR'falling,WR'rising,
  300ns,Time'High,WARNING,"Forward check WR->WR");
   Check_forward(D'changing,WR'rising,
  50 ns,Time'High,WARNING,"Forward check WR->D");

  Process(CS,RD,WR,mode,D,A)
  Begin
   A<=A;
   D<=D;
   If ( CS = '1' ) Then
     D <= "ZZZZZZZZ" after 5 ns;
   End if;
   If ( CS = '0' ) Then
     If ( RD = '1' and WR = '1' ) Then
       D <= "XXXXXXXX" after 10 ns;
     End if;
     If ( RD = '0' and WR = '1' and mode = '0' ) Then
       D <= A after 10 ns;
     End if;
     If ( WR = '0' and RD = '1' and mode = '1' ) Then
       A <= D after 10 ns;
     End if;
   End if;
  End Process;
End Behavioral;
```

**Table 6: PPI - Simulatable VHDL code**

With a second, more specific code translator, VHDL code for synthesis can be generated. Unlike the code example of table 6, the synthesisable VHDL code does not contain anymore "after" clauses, "wait" clauses and timing constraint checks. Figure 7 shows the result of the synthesis of the VHDL code realized with the SYNOPSYS Synthesizer.

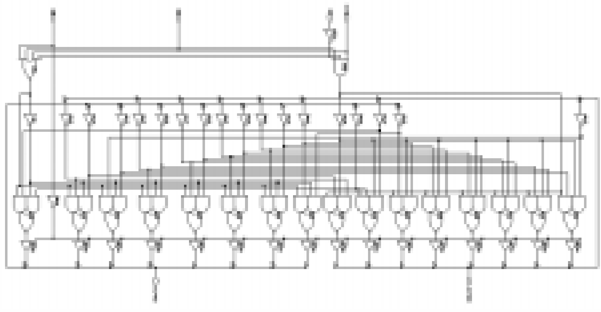

**Figure 7: PPI - Net-list**

## 8.2: Future work

In a next phase, ETD will be integrated into a user-friendly tool, allowing for more thorough testing of the formalism. Therefore, the present results should be considered as preliminary. Hopefully, ETD will finally be integrated into MODES.

## 9: Conclusions

The aim of the ETD (Extended Timing Diagram) formalism is the convenient high-level specification of behavioral digital models through the use of annotated hierarchical and concurrent timing diagrams. Since the ETD form of representation is close to the one used in conventional wave-form diagrams used in data sheets, ETDs are well accepted by hardware engineers.

The internal representation of the ETDs has been implemented in the form of FSMs. They are then used both for high-level simulation and translation into VHDL according to a common evaluation scheme. Furthermore, they can be translated into synthesisable VHDL for fast prototyping.

While ETDs can be used independently, they have been conceived as a complement to the existing high-level specification tools of the MODES[1] project. The user should finally be free to specify each part of a model with the most convenient behavioral editor of MODES. The system then integrates the respective elements into a single simulatable model, which can be translated into standard HDLs.

Many of the elements of MODES already exists, i.e. the BEMCharts formalism[8] for the high-level specification of FSMs, commercially available under the name of SPeeDCHART™[6]. Our current interest is focused on the integration of design elements originating from different specification sources.

## 10: Acknowledgments

We thank our colleagues in the MODES project - Khaled Bettaieb, Sean Dart, Dirk van den Heuvel, Charles Munk, Pierre Ukelo, Alain Vachoux and Roger Zinszner - for the fruitful collaboration.

MODES is a joint project of the University of Neuchâtel, the Swiss Federal Institute of Technology, Lausanne and Speed Electronic SA, Neuchâtel. It has been supported by the Swiss Commission for the encouragement of scientific research CERS/KWF under project numbers 2081.2, 2260.2, 2499.1 and 2609.1 .

## References

1. H.P. Amann et al., "The MODES modelling expert system", Proc. EURO-VHDL'91, Stockholm, Sept 1991, pp 192-195

2. G. Borriello, "Specification and Synthesis of Interface Logic", in "High-Level VLSI Synthesis", Editors R. Camposano and W. Wolf, Kluwer, 1991, pp 153-176

3. M. Dufresne et al., "Using Formalized Timing Diagrams in VHDL Simulation", Proc. EURO-VHDL'91, Stockholm, Sept 1991, pp 24-31

4. F. Liu et al., "Timing Constraint Checks in VHDL - a comparative study", Proc. EURO-VHDL'91, Stockholm, Sept 1991, pp 39-45

5. T. Amon et al., "Operation/Event Graphs: A Design Representation for Timing Behavior", Proc. IFIP WG 10.2 $10^{th}$ Int. Symp. on CHDL and their Applications, Marseille, France, 1991, in "Computer Hardware Description Languages and their Applications", Editors D. Borrione and R. Waxman, Elsevier, 1991, pp 261-280

6. S. Dart, "SPeeDCHART-VHDL reference manual V2.0", Speed Electronic SA, Neuchâtel (Switzerland), October 1992

7. "VHSIC Hardware Description Language", Language Reference Manual, IEEE-1076/87, New York, 1987

8. D.O. van den Heuvel et al., "High-Level behavioral Modelling using BEMCharts", to be published in ECCDT'93, Davos (Switzerland), Aug/Sept 1993

9. D.Harel, "StateCharts: A Visual Formalism for Complex Systems", Science of Computer Programming 8, Elseiver Science Publishers, 1987, pp 231-274

10. D.Harel et al., "StateMate: A Working Environment for the Development of Complex Reactive Systems", IEEE Transactions on Software Engineering, vol. 16, no. 4, April 1990, pp 403-414

11. F. Vahid et al., "SpecCharts: A Language for System Level Synthesis", in "Computer Hardware Description Languages and their Applications", Editors D. Borrione and R. Waxman, Elsevier, 1991, pp 165-174

12. R. Schlör et al., "Specification and Verification of System-Level Hardware Designs using Timing Diagrams", Proc. EDAC-EUROASIC 1993, Paris, Feb. 22-25, pp 518-524