

# Time prediction in High Level Synthesis

Marcos Luiz Mucheroni<sup>1</sup>, Geraldo Lino de Campos<sup>2</sup> e Renato da Silva Pereira<sup>1</sup>

## Summary

Many tools and environments have been developed to aid hardware designers to carry out the implementation of Application-Specific Integrated Circuits (ASICs). Most of them are aimed at implementing the lower level details of a target circuit with project constraints such as: area, timing, power consumption and so on. At a higher level, these tools have been written in hardware description languages (HDLs) to attend different objectives: algorithms, hardware, communication, specification and environments. High-level simulations are often referred to as behavioral, because only the overall behavior of component devices is made externally visible. On the other hand, structural simulation represents a more precise description of components. The growth of system complexity has made gate-level hardware design increasingly difficult. We discuss the needs of actual system specification with time prediction. Instead of using events in time, also known as reactive model, we use the transformational system technique, where an implicit list with local dependencies of data are defined together with a list-scheduling. We use a hierarchical structure of components, an object-oriented language, to this propose an environment aimed at a transformational model in a simple HDL.

## I - Introduction

An important decision during implementation is at what level of abstraction and which tools will be used to build the system. Tools for software and hardware synthesis are needed at different levels, for example: flowcharts, algorithms, RTL descriptions, Boolean expressions, transistor and timing diagrams. Different tools have different power of expression, but also different levels of complexity during implementation in time and effort.

The question that arises is what features are essential in a simple, fast and realistic system design. Thus, the HDL should be able to reduce drastically the effort required by the designer to express a particular feature of the design (e.g., resources, area and timing) without sacrificing the project itself. The implementation of timing and resources constraints are the most important facilities in many description styles and application domains. There is a difference between time constraints and time prediction; the former is needed as a form of restrictions during the specification phase of design and the latter is a prediction of the time needed for the actual implementation of a system.

An HDL can have a certain level of detail where any feature is monitored. As a general rule, the higher the level of detail the larger is the computing time consumed. This time can be significant if the simulation is structural where it may vary from 10 to 100 million of gates and this time grows in logarithm scale. Some of these languages are HardwareC [Ku and De Micheli (1988)], VHDL [Lipsett et al.(1989)], Statecharts[Drusinsky and Harel (1989)], Silage [Hilfinger and Rabey (1992)],

---

<sup>1</sup> Universidade Federal de São Carlos - São Carlos - SP - BRAZIL

<sup>2</sup> Escola Politécnica da USP- São Paulo - SP - BRAZIL

SpecCharts language [Vahid et al. (1991)], Verilog [Sternheim et al. (1990)] and SDL [Belina and Sarma (1991)].

Although VHDL is considered a good specification language, it fails when support for feature constraints is needed.

Because VHDL's semantics are primarily designed for simulation, high-level synthesis with time prediction is difficult.

High-level synthesis, typically divide the tasks into data-path design and control-path design. Scheduling data-path operations into control steps is the most important task. The scheduling strategy must consider not only time and resource constraints, but also storage and interconnection costs. The approach used here is list scheduling [Davidson et al. (1981)], but other approaches are also presented in the literature [Walker and Chaudhuri (1995)].

By system-level design, we mean reactive or transformational systems [Narayan et al. (1991)]. A reactive system is one which is essentially event-driven and has to respond continuously to external and internal events connection. On the other hand a transformational system is one in which an algorithm performs operations on a set of input data items and produces output data. If we need to support time prediction it would be necessary to analyze the system from a transformational point of view; that is, it is possible to consider time simulations only if we consider actual data producing actual output in time. A recent work, in this direction, uses object-oriented programming and a hierarchical view of time to specify systems [Gillard and Posch (1991)].

Time prediction and its optimization in digital circuits is a recognized key area of high level synthesis, therefore it requires a correct and accurate specification of circuits. Typically, timing prediction is detected using event-driven timing methods. To analyze critical paths in digital design McGeer and Bryton [McGeer and Brayton (1989)] present a logical functions theory for dynamic time analysis. Their approach is adapted here to define a Transformational Model of gates and circuits.

A simple HDL is proposed through the implementation of this Transformational Model and some object-oriented concepts.

## **II - Reactive systems and event-driven systems**

Reactive systems are those whose processes have to respond continuously to events. Examples of reactive systems are telephones, avionics systems and communication networks.

HardwareC is based on the widely popular C language, augmented to support hardware features such as timing and synchronization, however HardwareC suffers from some of the disadvantages found in VHDL.

A specification using HardwareC consists of a set of concurrent processes which communicate with each other. Processes can be enclosed within a hierarchy of blocks. Blocks can be used to define structural relationships between the processes. A process specifies an algorithm as a set of sequential operations described using a subset of programming constructs of the C language. Each process restarts itself upon completion. All operations in a process are assumed to be synchronized by a single-phase system clock and takes a whole number of clock cycles to execute. All I/O operations, message passing and register loading are performed synchronously.

In message passing, explicit send/receive constructs are used for data transfer and synchronization..

One interesting feature of HardwareC is the capability to specify parameterized descriptions or templates for blocks, processes, procedures and functions.

The SpecCharts language is aimed at the requirements capture phase, it, combines the three aspects of system specification (control, behavior and structure) into a single, unified concept. The language possesses abstractions which enable the designer to easily and concisely represent his/her conceptual view of the design. It can be summarized as a combination of hierarchical/concurrent state diagrams. The basic object in SpecCharts is a “behavior” which can be expressed in one of three ways: concurrent sub-behaviors, sequential sub-behaviors and program code (or leaf behaviors).

The Silage language was developed aimed at problems related to digital signal processing (DSP), such as, digital filters, Fourier transform, z-transform and others. DSP systems suffer from typical problems, where an amount of data values are entered, computations are performed on them and result values have to be shown immediately.

The components of this language, named EXUs (EXecution Units), are building components initiated in a set of FBBs (functional building blocks), such as, adders, shifters, buffers and others. The interesting concept in Silage is the modularity present in FBBs.

Statecharts is essentially an event-driven technique, designed for specification of reactive systems. The basic object in Statecharts are states and transitions between states, which occur based on a combination of events and conditions. Transitions in Statecharts are not level-restricted in the sense that transitions between states at different hierarchical levels can be specified.

In contrast, using Verilog is more like programming in any structured high level language, but one single and important difference is its concept of time, and its effect on the execution order of statements in a module. Since in hardware all the elements operate in parallel, a serial model for execution is not appropriate in an HDL, and therefore, in Verilog execution is event-driven. The event scheduler of the Verilog simulator is analogous to the program counter of common languages [Sternheim, E. et al. (1990)]. The order of event execution within the same simulation time, in general, is not known, and one cannot rely on it, since the Verilog simulator may try to optimize execution by ordering the events in a particular way. A very important feature of this language is its concise syntax.

### **III - Transformational View and its problems**

All the HDLs previously shown use common approaches to model the time behavior of systems using an event queue; this performs predicted future changes of the system under consideration in a global queue as data structure; that is, the simulation of the system is accomplished by working through this global queue viewing changes in the states of the system’s variables.

But why should be believed that the solution to achieve a good specification lays in introducing yet another specification language in addition to the meaning crafting solutions that already exist? The answer to this can be influenced by the needed design methodology. For finding new solutions at any level, clearly a variety of languages and methodologies are needed. In the HDL proposed here, unified behavioral and structural specifications may be carried out and simulated; with some restrictions however, because problem partitioning is not presented here. To solve

this problem we make a clear cut between hardware and software, then code generator and control are performed using PLA.

To incorporate timing information in a design specification, a more precise description of circuits is needed, that is, how many times the information pass through the circuit. There are two important problems in time specification:

- a) electronic devices do not respond immediately, and,
- b) different inputs can respond at different times.

Concerning the first problem, VHDL and SpecCharts use time specification in design, which can be of two types: 1) specification of the maximum time that the information can spend waiting for the delay signal, (e.g. VHDL: **wait** on Signal\_1 **until** (Signal\_2='1') **for** 10 ns), and 2) specification of time in the future, when it will assume a new value (e.g.VHDL: Signal\_1 <- 1 **after** 10 ns).

To solve the first problem VHDL-92, introduces the reserved words **inertial** and **transport**. The keyword **inertial** corresponds to the wait for a signal to become active, while **transport** is concerned with simple delay, that is, its behavior corresponds to, for example, the delay of the transmission-line.

The second problem is more subtle and complex. It is complex as been shown by the many works [Narayan et al. (1991)], [De Micheli (1994)] and [Gillard and Posh (1995)] that have been written to discuss the importance of structural hierarchy and object-oriented programming in timing problems. Narayan et al. present a survey of HDLs and point out need for hierarchical methods to specify systems. De Micheli analyzes the problem from the scheduling point of view; and the work by Gillard and Posch shows the importance of using object-oriented languages to solve hierarchical and timing problems. The problem is subtle because sometimes, models of simulation hide some path variations that appear in actual implementations.

In order to implement the transformational model we will analyze some formal definitions.

Let us assume that there is a logic function that has one output  $f$  and  $n$  inputs  $x_1, x_2, \dots, x_n$ . If one of the inputs to the logic function suffers a hazard, say input  $x_i$ , then the output would be  $f(x_1, \dots, \bar{x}_i, \dots, x_n)$ . To analyze when an error occurs, it would be desirable to know what will be the actual output, and for this, we define the function  $df(X)/dx_i$ , called *Boolean difference* of  $f(X)$ :

$$\frac{df(X)}{dx_i} = f(x_1, \dots, x_i, \dots, x_n) \oplus f(x_1, \dots, \bar{x}_i, \dots, x_n) \quad (1)$$

The *cofactor* of a function  $f(x)$ , written  $f(\bar{x})$  is a function  $f$  obtained by evaluating  $f$  at  $x = 1$  and  $x = 0$ . We simplify them by writing  $f_x$  and  $f_{\bar{x}}$ .

The work by [Sellers et al. (1968)] is an example of the analysis of circuit errors using the boolean difference.

A *path* through a combinational circuit is a *sequence of nodes*,  $\{ f_0, \dots, f_m \}$ , such that the output of  $f_i$  is an input of  $f_{i+1}$ .

Each node  $f_i$  in a combinational circuit has a *propagation time*  $\{ f_0, \dots, f_m \}$ . The value of node  $f$  at time  $t$  is that determined by the evaluation of the node using the values of its inputs at time  $t - \Delta(f)$ .

We define *delay* in a path  $P = \{ f_0, \dots, f_m \}$  as:

m

$$d(P) = \sum_{i=0} \Delta(f_i) \quad (2)$$

An *event* is the transition of a node from a value of 0 to 1, or vice-versa.

If a gate has different inputs at different arrival times, the *static transformation delay* of a node  $f$  is computed as:

$$d(f) = \Delta(f) + \max\{d(i) | i \in \text{inputs}(f)\} \quad (3)$$

A *transformation* is a particular transition from 0 to 1 or 1 to 0 at a time  $t_0$  or  $t_1$ . A *transformational path* is a sequence of nodes  $\{f_{t_0}, \dots, f_{t_m}\}$  such that the output of  $f_{t_j}$  is an input of  $f_{t_{j+1}}$  taking input specific values in a delay  $d(f_{t_j})$ , for  $0 \leq i \leq m$ .

When specific values of a sequence of the functions  $c = f_{t_0}, \dots, f_{t_m}$  are taken, the *dynamic transformation delay* of a node  $f$  is computed as:

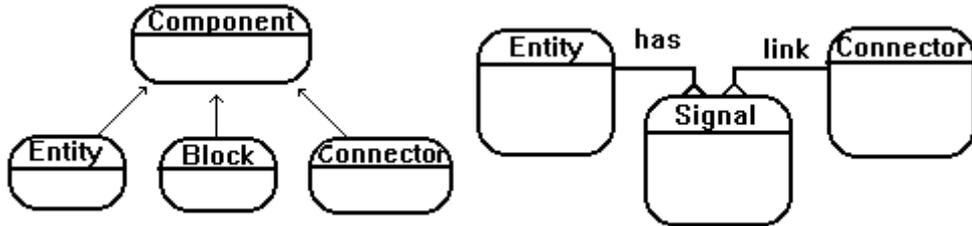
$$d_c(f_{t_k}) = \Delta(f_{t_k}) + \{d(f_{t_j}), 0 \leq j \leq k\} \quad (4)$$

These models are used to implement a simple HDL.

#### IV -A simple HDL with Time Prediction.

The reserved-words used in the proposed HDL follow roughly the same meaning as in VHDL, but there is not any compromise with it. A hierarchical model of parallel components are simulated using list-scheduling of general structure of ENTITY. Entities contain structural definitions in terms of time and they are interconnected through Connectors. Signals are derived from Connectors and are passage points for any data types and perform communication between Entities. Entities are specific implementations of Components.

Component is equivalent to a class in C++; that is, it defines a set of objects that have common characteristics. Despite Signal being very similar to Entity and Connector, it is not a specialization of them or any other type of components, presenting only an association relationship with them. In Figure 1, inheritance is depicted by an arrow and the association relationship is represented by a line.



**Figure 1** - Classes specialization or relationship between Entity, Connector and Signal.

SIGNAL present in logical circuits can be: HIGH, LOW, Z(high-impedance), X (indifferent), U (unused), and its types are: IN(input), OUT (output), INOUT (bi-directional), OPEN-COLLECTOR and PASSIVE. The operators used to implement gates are: NOT, OR, AND, XOR, and the relational operator EQU.

All statements are conditional because they depend on input or clock signal, the scope is limited like in C using  $\{ \}$ , the reserved-words used are IF, ENDIF, DELAY.

The RET reserved-word is mandatory to finish a program. The reserved-word INSTANCE is an object that is defined by a component. The reserved-word

ACCESSIBLE is analogous to the PUBLIC, feature used in C++, which is used by SIGNAL and means that other component can have access to it.

## V - Conclusion and Future Work

Hardware modeling, by means of HDLs, has changed the way computer architects think of circuits.

Decisions made during the implementation have to be taken concerning the specification level will be used to build the system. Different tools have different power of expression, but also different levels of difficulty of implementation in terms of time and effort. The two levels are often referred to as behavioral and structural. Other important question is what features are essential in a simple and fast system design. The effort required by the designer to insert a particular feature, without sacrificing the whole schedule, is important in order to consider time prediction.

The behavioral level is simpler than the structural but represents only the behavior of component devices, therefore structural simulation represents a more precise description of components.

Structural simulations have different level of detail, but the consumed computing time can be significant depending whether the simulation involves million of gates.

HDL can have a specific level of detail where any feature is monitored; some of these languages are HardwareC, VHDL, Statecharts, Silage, SpecCharts and Verilog.

By using these languages it would be possible to describe the design behavior of a system at several abstraction levels, however, they do not explicitly support time prediction because they are essentially event-driven and have to respond continuously to conditional events. Their semantics have been designed for events simulation; so, high-level synthesis with time prediction using events is difficult to accomplish.

The HDL presented here uses object-oriented programming concepts to implement hierarchical levels in order to provide time prediction.

## VI - Bibliography

- [Belina and Sarma (1991)] - Belina, D.H.F. and Sarma, A. - "SDL with Applications from Protocol Specifications" - Prentice Hall, 1991.
- [Davidson et al. (1981)] - Davidson, S. and others - "Some Experiments in Local Microcode Compaction for Horizontal Machines", IEEE Trans. Computers, Vol. C-30, N. 7, 1981, p. 460-477.
- [De Micheli (1994)] - De Micheli, G. - Synthesis and Optimization of Digital Circuits, McGraw-Hill Int. Ed., NY-USA, 1994.
- [Drusinsky and Harel (1989)] - Drusinsky, D. and Harel, D. - "Using Statecharts for Hardware Description and Synthesis", IEEE Transactions on Computer-Aided Design, 1989, p. 798-807.
- [Gillard and Posch (1991)] - A Hierarchical View of Time, Proceedings of the 34th Midwest Symposium on Circuits and Systems, Monterey, USA, 1991.
- [Hilfinger and Rabey (1992)] - Hilfinger, P. and Rabey, J. - Anatomy of a Silicon Compiler, Kluwer Academic Publishers, 1992.
- [Ku and De Micheli (1988)] - Ku, D. and De Micheli, G. "HardwareC - A Language for Hardware Design" - Stanford University, Technical Report CSL-TR-90-419, 1988.

- [Lipsett et al. (1989)] - Lipsett, R.; Schaefer, C. and Ussery,C. - VHDL: Hardware Description and Design, Kluwer Academic , 1989.
- [McGeer and Brayton (1989)] - McGeer,P.C. and Brayton,R.K. - Provably correct Critical Paths - in Advanced Research in VLSI - Proc. of Decennial Caltech Conference on VLSI, ed. Seitz, C.H., MIT Press, USA, 1989.
- [Narayan et al. (1991)] - Narayan, S. ; Vahid, F. and Gajski,D - “System Specification and Synthesis with the SpecCharts Language”, in Proc. of the Int. Conference on Computer-Aided Design, 1991.
- [Sellers et al. (1968)] - Sellers Jr., F.F., Hsiao,M.Y. and Bearnson,L.W. Analyzing Errors with the Boolean Difference - IEEE Trans. on Comp, V. C17, n. 7, 676-683.
- [Sternheim, E. et al. (1990)] - Sternheim, E., Singh, R. and Trivedi, Y. - Digital Design with Verilog HDL, Automata Pub., CA-USA, 1990, 215 p.
- [Vahid et al. (1991)] - Vahid, F.; Narayan, S. and Gajski,D. . - “SpecCharts: A Language for System Level Synthesis”, Proceedings of the Int. Symp. on Comp. HDL an their Applications,1991.
- [Walker and Chaudhuri (1995)] - Walker, R.A. and Chaudhuri, S. - “Introduction to the Scheduling Problem” , IEEE Design & Test of Comp., 1995,p. 60-69.