Using VHDL for Link to Synthesis Tools

Mohammed Belhadj

HAL Id: hal-00545948

https://hal.science/hal-00545948

Submitted on 13 Dec 2010

Using VHDL for Link to Synthesis Tools

Mohammed BELHADJ

IRISA, Campus de Beaulieu
35042 Rennes, FRANCE

Abstract

This paper presents the work done to use industry and academic synthesis tools for the hardware-software codesign of reactive systems. It emphasizes the hardware synthesis and design part by linking SIGNAL and VHDL. The SIGNAL language is used for system specification and VHDL for the link to synthesis tools. To permit a maximum of flexibility, different strategies for linking are described.

1 Introduction

VHDL is now used in different synthesis tools as an input and/or output description language. This presents an ideal opportunity to develop tools with an easy access to various synthesis and CAD/CAE tools using VHDL as common platform.

The flexibility of VHDL allows description of architecture at different levels of abstraction (behavioral, register transfer, or gate levels) and with various styles [4]. Synthesis tools depending on the input abstraction level use subsets of VHDL that can be mapped automatically to architecture. An architecture (in a given abstraction level) can be described with different styles. Not all description styles will lead to an optimized design.

This paper describes the work done to develop linking methodologies between the SIGNAL [8] environment for specification, simulation and design of reactive, signal processing, and real-time applications, and synthesis tools and CAD/CAE environments using VHDL.

We describe first our motivations to use VHDL as an access language to various synthesis tools. Then, a brief description of SIGNAL environment is given. The fourth and fifth sections deal with the VHDL generation method and the experimental environment.

2 Motivations

In our view, SIGNAL is used as a high level design language (front-end) and VHDL as an access medium for CAD/CAE tools. This fits in the general stream of Top-Down design environment and methodology for reactive systems, where parts of a system (or the whole) may be implemented in hardware. SIGNAL environment will definitely offer tools for hardware-software codesign including: specification, simulation, formal verification, hardware synthesis, software compilation, etc. Use of VHDL offers the possibility to use a wide number of synthesis tools without loss of independence regarding the technology and CAD/CAE tools, and increasing design reuse.

The flexibility of VHDL permits to translate a SIGNAL program at different levels of abstraction (see Fig. 1). A SIGNAL program can be translated in behavioral level VHDL. A program with more implementation details (Register Transfer "RTL" or gate level) is translated into RTL or structural VHDL. This flexibility can be used for the development of specific high level synthesis tools around SIGNAL and translating the result into RTL or structural VHDL (structural can be understood as gate level or any *library* based design).

We will purpose a framework for SIGNAL environment using a multi-level VHDL generator:

- A behavioral translator converts a SIGNAL program into a VHDL program that can be used by high level synthesis tools.

- An RTL translator generates VHDL programs suitable for register transfer (RTL) synthesis.

- A structural translator.

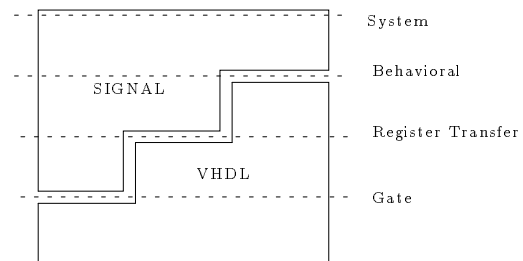In the next section we will give an insight of the SIGNAL environment.

Figure 1: SIGNAL to VHDL translation

3 SIGNAL Environment

The SIGNAL language [8] was developed at IRISA Rennes to design and implement reactive systems. The

basic object handled by SIGNAL is a possibly infinite sequence (or a stream) of values called a *signal*.

A SIGNAL process is a set of relations (equations) between signals, specifying both constraints on their values and timing. SIGNAL contains the following basic operators:

- **Functions:** Usual functions AND, OR, +, *, etc from instantaneous domains (boolean, integer,...) are extended to signals.

- **Delay:** The delay operator gives access to past values of a signal. The SIGNAL statement corresponding to delay is: ZX := X \$ 1 , with ZX init x_0. For every occurrence of the signal X, ZX carries the previous value of X. We can extend this operator to a delay of k (k > 1).

- **When:** This operator allows one to conditionally extract values from a given signal. The expression: Y := X **when** C, where X and Y are signals of the same type and C is a boolean signal describing the under sampling of X. Y is present when both X and C are present and C is TRUE ("$\perp$" represents the absence of value).

$$
\begin{array}{ccccccccc}
X: & x_1 & x_2 & \perp & x_3 & \perp & x_4 & x_5 & x_6 & ... \\
C: & F & T & F & F & T & T & \perp & T & ... \\
Y: & \perp & x_2 & \perp & \perp & \perp & x_4 & \perp & x_6 & ...
\end{array}
$$

- **Default:** This operator allows the merge of signals of the same type: Z := X **default** Y, Z merges X and Y, with priority to X when both signals are simultaneously present.

$$
\begin{array}{cccccccc}
X: & x_1 & x_2 & \perp & x_3 & \perp & ... \\
Y: & y_1 & \perp & y_2 & \perp & y_3 & ... \\
Z: & x_1 & x_2 & y_2 & x_3 & y_3 & ...
\end{array}
$$

We can build elementary processes as equations between signals " X := *exp*", where X is a signal and *exp* is any valid SIGNAL expression built using the kernel operators and other signals (e.g X:= (A when B) default (C+D)). More complex processes can be built using the commutative and associative composition operator "|". The process

$$(| \ P1 \ | \ P2 \ ....| \ Pn \ |)$$

simply denotes the union of the equations expressed by P1,P2,...,Pn, where Pi is an elementary process or a composed process. Consider the following example:

(| Y := (0 *when* R) *default* ((ZY+1) *when* C)
| ZY := Y\$ 1 |)

The behavior of the process is simple: when the signal R (reset) is true Y is equal to 0, otherwise it is equal to the previous value of Y noted ZY plus one when the signal C (clock) is true. A possible execution is shown hereafter, ZY is initialized with 0:

$$
\begin{array}{ccccccccc}
R: & T & F & \perp & F & \perp & F & \perp & T & ... \\
C: & T & T & T & T & T & T & T & T & ... \\
Y: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 0 & ... \\
ZY: & 0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & ...
\end{array}
$$

Other operators have been defined from the kernel that permit the reduction of programming effort, they can be found in [8].

The SIGNAL compiler transforms a program into the kernel language, and checks for its consistency. It is in fact a formal calculus system [8]. A valid executable program is then translated into an intermediate form, used in different tools in the SIGNAL environment.

The SIGNAL environment consists of: a mixed schematic/textual program entry, a compiler, and generation tools (see Fig. 2). The compiler transforms SIGNAL programs into an intermediate graph format. This format is used for sequential code generation (C, Fortran77). It can be also used for parallel execution on general purpose machines (e.g. iPSC, T-node) or specialized parallel machines. The intermediate format is also used for generation of compact representation for formal verification and for hardware synthesis.
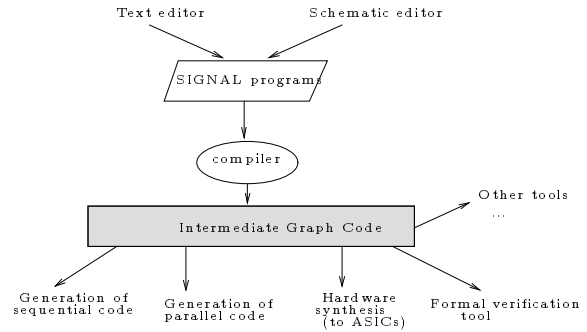

Figure 2 : SIGNAL environment

4 Link to VHDL synthesis tools

To map SIGNAL programs to VHDL some important points must be taken into account. SIGNAL is an abstract language that can describe behavior of a system or properties (constraints). The following SIGNAL statement: " X$\hat{=}$ *when*(X)", means that X is always true (it is a property or a constraint, and not a function that computes outputs given inputs and internal states). For synthesis purpose we require that SIGNAL programs to be translatable to VHDL must be executable (this can be checked using the compiler).

SIGNAL programs can be implemented as synchronous, asynchronous or mixed (synchronous and asynchronous) circuits. As most of the existing tools use synchronous circuits, we will present in this section how we achieve the translation to VHDL for synchronous implementation. Starting from SIGNAL executable programs, three translations can be performed: behavioral, Register Transfer, and structural.

4.1 Behavioral

Most of synthesis tools translate VHDL into a representation called CDFG (control data flow graph). It repre-

sents the data parallelism and sequentiality and the control flow operations [6].

The internal representation of SIGNAL programs (Dynamic graph or DG) is close to CDFG. However, as VHDL is the standard description language for hardware design we will translate the DG into a sequential VHDL process.

The DG of SIGNAL programs are generated during the compilation process. It describes the dependency of data and control.
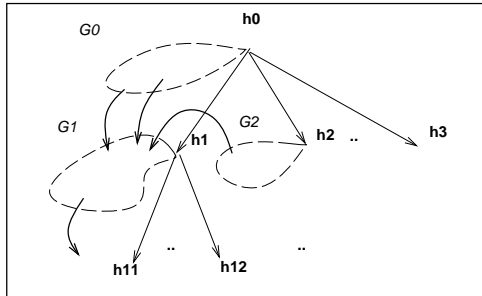

Figure 3: SIGNAL internal graph representation

In Figure 3 h_0, h_1,.. represent control instructions (condition, ..) and G_0, G_1, .. represent data dependency graphs where vertices represent operation (input, output, $+$, $*$,etc) and arcs data transfer.

The translation from a SIGNAL program to a behavioral VHDL program is quite obvious. We generate a VHDL process representing the DG. There are only two difficult points:

1. synchronization: As a SIGNAL program is a reactive process it waits for events on inputs and executes all corresponding computations (it reacts to input stimuli). The computation takes "0" logical time. A possible interpretation of SIGNAL's Dynamic Graph, would be:

   ```
   wait until H;
   ..   sequential representation of
   ..   control and data flow graph (variables :=)
   .. signal assignments(<=)
   ```

   where H is true if there is a transaction in any one of the inputs. We use signal assignments ($<=$) once for each signal assigned in the process at the end of the process. We calculate the values to be assigned using variables.

2. High Level Synthesis limitations: One can translate easily SIGNAL into VHDL , but to be useful for synthesis the generated code have to follow some guidelines:

   - restrictions on accepted data types.
   - some synthesis tools take into account only a restricted form of the wait (e.g. wait until CONDITION, where CONDITION is a boolean expression on signals). Attributes like Transaction and Delayed are prohibited. The translation of

the synchronization described above is not suitable. We replace it by a condition set to true before the wait.

- memory management: in SIGNAL the delay operator ($ k) is optimized by the compiler in order to generate efficient code for simulation and execution of SIGNAL programs. This is done by a specific representation of past value of a signal and an efficient access to the delayed signal. This is misleading for synthesis, and for delay greater than 1, we have to change the representation of the delay in the intermediate graph generated by the compiler.

4.2   Register Transfer

In an RTL description the time is divided into intervals or steps of control. An RTL description specifies for each step the transfers to perform and the next control step to enter. To be synthesizable, an RTL VHDL description has to meet some restrictions. As different synthesis tools have different restrictions, we choose (for the moment) the most restrictive subset.

To translate a SIGNAL program into the VHDL RTL description, we have to check if the program verifies those restrictions (data type, one clock, etc). For register transfer translation two cases can be distinguished:

1. Combinatorial logic: The SIGNAL program will ultimately correspond to a combinatorial circuit described as a VHDL process with an explicit sensitivity list. The following SIGNAL statement:

   ```
   (| (D := A+B when C) default (A*B when not C) |)
   ```

   will be translated as:

   ```
   process(A,B,C)
   begin
     if C then  D <= A+B;
         else  D <= A*B;
     end if;
   end process;
   ```

2. Clocked logic: For clocked logic we restrict ourselves to one clock with either rising or falling edge. A SIGNAL program will correspond to a sequential process with a wait statement at the beginning of the form:

   ```
   begin
   wait until clock=VALUE;
   .. -- statements
   ..
   end
   ```

   This is not the only possible scheme for translation and not the most efficient for all synthesis tools. The main advantage of this approach is that almost all RTL synthesis tools can handle such a description.

Initial values: Synthesis tools like Synopsys[2] do not accept signals with initial values. To permit the accurate translation of SIGNAL programs we add an input RESET that will be used to describe the behavior corresponding to the setting of initial values.

```
if RESET then
    ....
    set signals to initial values
else
    .... description used previously
    ....
end if
```

Control extraction: Another possible translation scheme will be to convert the initial SIGNAL specification into two processes: control and data path. The difference with the previous scheme is that the control extraction is done in the SIGNAL environment and not in the RTL synthesis tool. The user can choose either the control extraction associated with each synthesis tool or use a unique one, implemented in the SIGNAL environment. The control extraction algorithm uses the set of boolean used for memory operator (the delay $ in SIGNAL). Given this set, We can use a simple constructive algorithm that computes the transition function, control function (from control to the data path), and status function (from data path to control).

4.3 Structural

The structural translation converts a net of SIGNAL processes into a net of VHDL entities. Therefore, an obvious one to one mapping, but useful when dealing with already implemented design (the corresponding VHDL was generated), using different synthesis environments for different parts of a design, or for a bottom-up methodology.

5 Experimentation

The methodology described above is subject of ongoing development. We use three high level synthesis tools: Gaut, VSS, and AMICAL (see Fig. 4). Gaut[5] is a pipeline architecture synthesis tool, dedicated to signal processing applications under real-time execution constraints. AMICAL[7] and VSS[3] are more general high level synthesis tools. Two commercial tools are used for RTL synthesis: Synopsys[2] and Compass[1].

The user can choose to translate a SIGNAL program into behavioral, RTL or structural VHDL. The default is behavioral. For behavioral and RTL we check for the possibility of translation of SIGNAL program to VHDL suitable for synthesis (data, clock, ..).
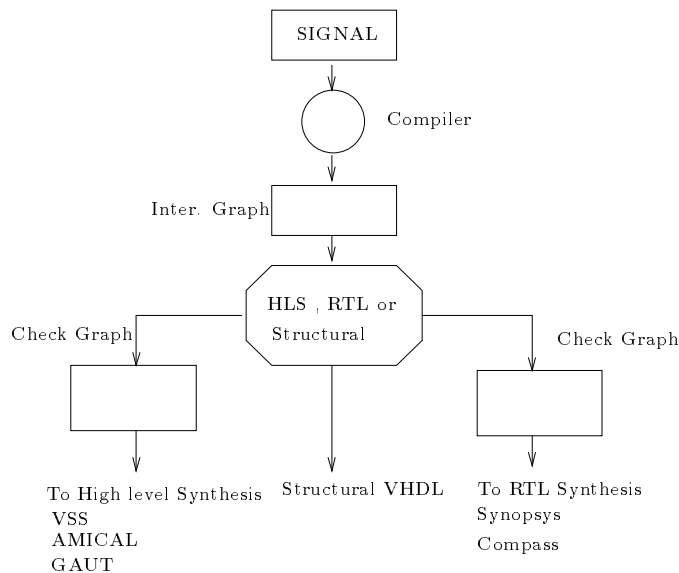

Figure 4: SIGNAL synthesis environment

6 Conclusion

To meet today technological challenge design methodologies for complex system are required. These methodologies have to give flexibility, accuracy and achieve good performances. SIGNAL environment permits the design of reactive systems and their implementation on software and hardware. The link to hardware is done using VHDL. This translation is still in early step and is restrictive, as no standard subset for synthesis of VHDL exists. However, the interface is flexible enough to access differents tools and by adapting the generated VHDL to specific synthesis environments, we can expect to achieve efficiency and accuracy.

References

[1] *ASIC Synthesizer for VHDL Design.* COMPASS design automation, V8R4.0 edition, Nov. 1992.

[2] *Synopsys VHDL Compiler Reference Manual.*

[3] *VHDK Synthesis System (VSS), User's Manual.* University of California, Irvine, version 5.0 edition, Jun. 1992.

[4] Allen Wu Daniel Gajski, Nikil Dutt and Steve Lin. *High-Level Synthesis.* Kluwer Academic Publishers, 1992.

[5] H. Dubois J. Philippe E. Martin, O. Sentieys. GAUT: an architectural synthesis tool for dedicated signal processors. In *Euro-DAC*, pages 14–19, Sep. 1993.

[6] L. Stock J. van Eijndhoven, G. de Jong. *The ASICS Data Flow Graph.* Technical Report 91-E-251, EINDHOVEN UNIVERSITY OF TECHNOLOGY, Jun. 1991.

[7] A. Jerraya, I. Park, and K. O'Brien. AMICAL: interactive high-level synthesis environment. In *EDAC 93 Paris*, Feb. 1993.

[8] Paul Le Guernic, Thierry Gautier, Michel Le Borgne, and Claude Le Maire. Programming real-time applications with SIGNAL. *Proceedings of the IEEE*, 79(9):1321–1336, sep. 1991.