VHDL
A Short Review of the Basic Concepts

Paulo Flores

Inesc - Apartado 10105, Lisbon PORTUGAL

Contents

1 Introduction

This report gives a brief review on the use of hardware description languages in a CAD environment. Special attention will be given to the new emerging standard language, the VHSIC[1] Hardware Description Language (VHDL).

Hardware description languages describe the circuit at an higher level of representation allowing an easier description of complex circuits and a direct verification of the desired functionality through simulation. The translation of the circuit to a gate-level description can be done either manually or automatically, using for instance a synthesis tool.

Hardware description languages overcome some of the limitations of algorithmic languages, where the referencing time is not important. While in the first ones the temporal behavior of the circuit is explicit in the language in the second ones this behavior must be represented in some algorithmic way.

The choice of the foundry can, thus, be defered to a later stage in the design flow and design re-use and library sharing is also made easier because of the technology-independet representation and because libraries are more readable and much more pratical to use than netlists.

Hardware description languages are also an important means to interchange hardware information between organizations, an advantage which become more significant if the chosen language enjoys widespread support within the microelectronic community.

In the next sections we summarize VHDL main features, and outline some comments on it's use in a CAD design enviromment.

2 The VHDL Language

The standardization of VHDL began in 1986 with the adoption of VHDL version 7.2. The language definition is now is stable and became a DoD[2]/IEEE standard for electronic designs. It is currently becoming supported in most commercial CAD systems.

---

[1] VHSIC - Very High Speed Integrated Circuits

[2] Department of Defense (US)

2.1 Design Entities

The design entity is one hardware abstraction of VHDL which can represent a whole system, a board, a chip, a cell or just a gate. Each design entity is formed by two main parts: the entity declaration and the architecture body. The entity declaration defines the interface between the device and the environment in which it is used and the architecture body describes the internal architecture of the entity.

Each entity declaration can have associated any number of architecture bodys. The same functionality can be obtained using one of the following descriptions:

Structural Description - the design is represented as an arrangement of interconnected components.

Dataflow Description - represents the behavior but implies some structure. It is similar to an RTL description.

Behavior Description - describes the behavior of the circuit without any structural information. The description is done using a sequence of programming language statements.

A mixed mode description is also possible using a combination of any of the three.

```
entity test_circuit is
    port(I: in integer range 0 to 7;        -- Data in
         D_IN: in bit;                      -- Load data
         X: in bit_vector(1 to 0);          -- Operation code
         O: out integer range 0 to 7;       -- Data out
         OVF,                               -- Overflow condition
         LT, EQ, GT: out boolean;           -- Comparison results
         RESET,                             -- System reset
         CLK: in BIT                        -- Clock
    );
end test_circuit;
```

Figure 1: The interface description

```
architecture behavioral of test_circuit is
begin
    process
        variable reg, acc: integer range 0 to 7;
        variable temp: integer range 0 to 15;
    begin
        if CLK = '1' and not CLK'stable then
            if D_IN = '1' then
                reg := I;
            else
                case X is
                    when "00" =>                -- No Operation
                        null;
                    when "01" =>                -- Load reg into acc
                        acc := reg;
                        ovf <= FALSE;
                    when "10" =>                -- Compares reg with acc
                        LT <= (reg < acc);
                        GT <= (reg > acc);
                        EQ <= (reg = acc);
                    when "11" =>                -- Adds reg and acc to acc
                        temp := reg + acc;
                        acc := temp rem 8;
                        ovf <= (temp > 7);
                end case;
            end if;
        end if;
        if RESET = '1' and RESET'stable then
            reg := 0;
            acc := 0;
        end if;
        O <= acc;
        wait on CLK, RESET;
    end process;
end behavioral;
```

Figure 2: The behavior description

To illustrate the concept of a design entity, we will use as example a 3 bit ALU description[3]. Figure 1 defines a "black box" for a specific function. The VHDL code of figure 2 represents one implementation of this function.

2.2 Objects and Data Types

VHDL objects are the entities that contain a value of a given type. There are three classes of objects: constants, variables and signals. *Constants* have a fixed value that can not be changed and *Variables* have a value which can be changed by assignment. *Signals* can be seen as an abstract description of a wire: they have a past history of values, present value, and a set of projected future values. Only the future values can be changed by assignment.

The type of the object determines the kind of values that it can hold. VHDL permits designers to declare any number of data types to characterize objects.

The four basic scalar types are *integer, floating point, physical* and *enumeration* . Composite types like *arrays* or *records* can be defined from the basic types. Subtypes can also be defined using a constraint on a specific type. Two other important types are available. *Access types* which are similar to pointer variables in other languages, and *file types* which provide a way of communication with an external design environment.

2.3 The Timing Model

The VHDL timing model is quite general and complex. As a discrete event simulation description language, VHDL assumes that all signals propagate in one direction and some delay is always involved.

The time scale can be represented in two different ways, one that measures the real time, and other that represents the time to evaluate one simulation cycle. The latter is called *delta delay* and can be seen as an infinitesimally small but non-zero delay: each time one simulation cycle ends within the same real time one delta delay is incremented. The real time advances for the next value only after all the events for that instant have been handled.

A driver is a container for a projected output waveform. For each driver associated to a signal, through a signal statement assignment, two delay models can be used to control the way that the new pairs, time/value, are placed in the driver:

[3]For a more detailed description see reference [1]

Inertial Delay - This is the default model in VHDL, which is used for model switching circuits, where, unless a value persists for a minimum amount of time in the input, no changes occur on the output.

Transport Delay - This model is analogous to the delay incurred by passing a current through a wire. Any pulse on the input is transmitted, no matter how short its duration is.

2.4 Statements

Statements specify the organization and the operation of the design. As most hardware devices work in parallel, VHDL is a massively concurrent language. Therefore, the designer has two levels to describe the operation of the system: the sequential level and the concurrent level.

2.4.1 Sequential Statements

All sequential elements are encapsulated in processes, subprograms or functions for use in concurrent contexts. The sequential statements specify the algorithms in a step by step fashion. Instruction similar to any high-level language like *if, case, loops, procedures, variable assignment,* etc are allowed with their own syntax. As an hardware description language some constructions are unique to VHDL.

Sequential Signal Assignment Statement is where future values for a signal are "proposed". As will be shown in the following sections, it is not certain that this value will become the actual value of the signal.

Wait Statement suspends the execution of the algorithm, waiting for some set of conditions. These conditions can be just a timeout, a logical condition, a signal sensitivity or a mix of the three.

Sequential Assertion Statement checks if the specified condition is true, otherwise an error is reported.

Sequential statements are used when we want to describe a circuit using just behavior, without any structural information.

2.4.2 Concurrent Statements

Concurrent statements are executed asynchronously, no relative order of execution being assumed. They are used for the structural and dataflow descriptions of the circuit.

- Structural Description

  Block Statement allows concurrent statements to be grouped into one logical unit describing a portion of the design.

  Component Instantiation Statement defines a subcomponent of the design entity and associates signals to the interface ports of that subcomponent.

  Generate Statement provides a mechanism for iterative or conditional elaboration of a portion of a description.

- Dataflow Description

  Concurrent Signal Assignment Statement creates a new driver for each assignment where new values "proposed" for the signal are retained. Each signal may have associated a set of drivers. If for the same instant different drivers try to impose different values, a *resolution function* must be called to decide the actual signal value.

  Process Statement defines an independent sequential way of describing the behavior of some part of the design.

  Concurrent Procedures Calls are executed asynchronously.

  Concurrent Assertions Statements have the same meaning as sequential assertions but used in concurrent statements.

3 Application of VHDL

Although VHDL started as a simulation language, currently other CAD tools accept it as way to describe a circuit. Due to its complexity, some restrictions are usually made, and most of the tools support only a specific subset of the language.

3.1 Simulators

As expected, simulators were the first tools to use VHDL. Today many simulators are available from different CAD vendors but some of them use only a subset of the language.

Two differents approaches have been followed in the design of VHDL simulators. Producing intermediate code, for latter interpretation is one of them, resulting in a slower simulation but with faster turnaround time to correct errors. Others generate C code directly from VHDL, producing a faster executable simulation. Future simulators will compile directly from VHDL to machine executable code.

3.2 Synthesis Tools

Sinthesis tools are starting to support VHDL, although some problems can arise with its use for synthesis [2]. Difficulties such as modeling low-level devices, understanding if timing information is part of specification or if it belongs to the simulation model, defining the equivalence between logic levels and electrical levels are some of the problems.

Thus, synthesizing a circuit from a VHDL description is possible, but some limitations are imposed to that description. Only a subset of the language is used, which means that some of the constructs specific to simulation may be ignored or even not supported. Some tools impose more severe restrictions, requiring the use of some structural information, such as an RTL description.

Moreover the sythesis results depend on the type of description.

3.3 Other CAD Tools

Design capture and test tools have also already started to accept VHDL. Again, just a subset of the language is supported. So for each tool, there is usually a VHDL flavor.

Tools, like analyzers, which just take a VHDL description, verify its correctness with respect to syntax and static semantics, and enter them into a VHDL design library make easier the interface between old and new tools to VHDL. The use of the *attribute* feature of the language, which allows associate arbitrary design data with a variety of items in VHDL, is another way to facilitate tool interface.

4 Conclusions

In the future, electronic designs will start from higher abstraction levels which means that hardware description languages will be routinely used.

Unlike other hardware description languages, that are property languages, VHDL is public, allowing you to be uncommitted to any company polices. Most of all, the design of the language itself involved user participation which "guarantees" that VHDL will become a used standard.

The adoption of VHDL as DoD/IEEE standard for electronic design description brought many benefits to the CAD community. First, from the viewpoint of the designer it is only necessary to learn one language for the whole design task (simulation, synthesis and so on). Second, it provides a standard for sharing information between the design team using different levels of abstraction. Due to the possibility of different styles of description, it is possible to design complex systems on a chip, without any commitment to any particular architecture. Even vendors can distribute the behavior of their devices without giving any proprietary information about construction. Finally, documentation of digital systems is possible in a technology-independent way.

Decide which will be the best set of VHDL constructs and the design style to adopt for a CAD environment depends on the CAD tool involved and is still a matter of current research. The inexistence of a common VHDL subset for the different CAD tools is still a major limitation.

References

[1] José Carlos Monteiro. The use of uncommitted libraries in a design system. QuickChips Deliverable 2, INESC, 1991.

[2] Michel Crastes and Alain Fonkoua. Hdls and logic synthesis. Technical report, INPG/91/ECIP2/WP2/SYN.D, 1991.

[3] R. Lipsett, C. Schaefer, and C. Usserty. *VHDL: Hardware Description and Design*. Kluwer Academic publishers, 1989.

[4] IEEE Std 1076-1987. *IEEE Standard VHDL Language Reference Manual*. IEEE, 1989.

[5] CAD Languages Systems. Inc. *VHDL Training Seminar*, 1990.

[6] Moe Shahdad. An Overview of VHDL Language and Technology. In *Design Automation Conference*, pages 320–326, July 1986.

[7] T. Dillinger, K. McCarthy, T. Mosher, D. Neumann, and R. Schmidt. A Logic Synthesis System for VHDL Design Description. In *IEEE International Conference on Computer-Aided Design*, pages 66–69, 1989.

[8] Larry M. Augustin. Timing Models in VAL/VHDL. In *IEEE International Conference on Computer-Aided Design*, pages 122–125, 1989.

[9] Steve Carlson. *Introduction to HDL-Based Design Using VHDL*. Synopsys, Inc., 1990.