# Evaluation Criteria of HDLs : VHDL compared to Verilog, UDL/I & M

Serge Maginot

*LEDA* S.A. 35 Avenue du Granier 38240 Meylan France
Phone: (33) 76 41 92 43 Fax: (33) 76 41 92 44

***Abstract***

*VHDL is compared to three other well-known hardware description languages : Verilog (from Cadence Design Systems, now public), UDL/I (new Japanese standard) and M (from Mentor Graphics). This comparative study parallels the fundamental concepts of these languages and highlights the different design processes and methodologies they require.*

## 1. INTRODUCTION

Although VHDL is today considered to be a widespread standard, it is certainly not the only hardware description language (HDL). Before VHDL came to the market, many other languages were (and still are) used by IC designers. The origin of these HDLs is often linked to proprietary simulation products (as it is the case for Verilog and M), and there are even new standards (such as UDL/I) which are emerging after VHDL.

Comparing VHDL with some other well-known hardware description languages has two main interests. First, it may help the designer using one of the three other languages to learn VHDL on the basis of concepts he or she is familiar with. Then, such a comparative study will highlight the most fundamental characteristics of VHDL: generality of the language and thus lack of predefined constructs provided by other specific HDLs, but also implementation of powerful constructs allowing new modeling methodologies.

The motivation of this paper is not to decide whether VHDL is *better* or not than other languages (because it has or does not have some given features...). Such an assertion does not really make sense because it generally forgets that these HDLs have completely different philosophies and must be compared taking into account the differences in their application domains, modeling environments, supported methodologies... Being aware of this context will help the designer to approach VHDL in a more objective, practical and efficient way.

This paper was made possible thanks to the studies undertaken within ESPRIT/ECIP2 project and ELISE contract.

## 2. HISTORY

The history of these languages is very different (**table 1**). The first draft of VHDL was released in August 1985, and designed by Intermetrics, IBM, and Texas Instruments under exclusive contract of the DoD. From then on, IEEE Computer Society decided to standardize VHDL and to promote it in the industry world. The language was reviewed and improved by the VHDL Analysis and Standardisation Group (VASG), working group of an IEEE Design Automation Standard Subcommittee (DASS, a subpart of an IEEE Design Automation Technical Committee (DATC)). In December 1987, IEEE members balloting procedure approved the updated version of VHDL as an IEEE standard under the reference 1076-87.

UDL/I (Unified Design Language for Integrated circuits) is being developed since 1989 by the Japan Electronic Industry Development Association, depending on some major Japanese companies such as NTT. A UDL/I committee is in charged of writing a language reference manual : the draft version of May 14, 1991 (a translation of it) is the support of this paper. This language is the newest one (its design is not achieved yet) and it is almost unknown to IC designers (no UDL/I simulator is available for the time being).

VHDL and UDL/I were (are) thus developed as tool-independent languages. On the other hand, M and Verilog were designed as HDLs for proprietary simulation products.

Verilog is a language originally designed by Gateway Design Automation which has been later merged with Cadence Design Systems, Inc. From then on, Verilog has been known and used as the language of Cadence Verilog-XL simulator. Later faced with the competition of VHDL, Cadence decided in 1990 to transfer Verilog to the public domain, attempting thus to give a "standard label" to its language. The "Verilog" name is still a registered trademark of Cadence, but the language itself can be supported by any CAE tool. The use of the Verilog HDL is promoted by Open Verilog International (OVI) which published in October 1991 the first version of the Verilog Hardware Description Language Reference Manual.

M is the language of the Lsim simulation system developed by Silicon Design Labs, then by Silicon

0-8186-2780-8/92 $3.00 © 1992 IEEE

Compiler Systems, later merged with Mentor Graphics. M is still a proprietary language of Mentor Graphics and there is no sign this company wants to transfer it in the public domain.

Verilog and M languages are thus promoted by two of the most important CAD companies. A lot of models and libraries have already been written in both languages.

## 3. GENERAL CHARACTERISTICS

### 3.1. Syntax and Semantics

Syntactically, VHDL is inspired from the ADA language, Verilog looks like C or Pascal and M is totally based on C (**table 2**). Some tough debates between VHDL defenders and Verilog or M defenders partially recall debates between ADA and C users...

VHDL is certainly not easy to learn because it requires the designer to have or to get some software habits (separate compilation, use of a strongly typed language, overloading, TEXTIO package...). The generality of VHDL is often confusing : signals being of any scalar or composite type (and not only of an implicit bit type (X, 0, 1, Z) like in other HDLs), resolution functions having to be user-defined...

Compared to Verilog and M, it is clear that the level of complexity of VHDL is much higher, and this does not ease the migration to this language. A new (software) culture is often necessary for the designer. The other languages are closer to hardware notions (in Verilog : predefined basic gates and switch devices, in UDL/I : synchronous and asynchronous assignments, latches, registers and RAMs, clock expressions...).

Logical simulation is the first use of these four HDLs, which provide behavioral modeling capabilities, but other applications such as synthesis or formal verification may use these languages as well.

VHDL, Verilog and M are just defined with a simulation semantics (**table 3**). Only UDL/I has a clear and non-ambiguous synthesis semantics (all the constructs in UDL/I are synthesizable). Other languages implement high-level sequential code (such as classical programming languages) that is not always possible to synthesize.

Nevertheless, there are many synthesis tools using VHDL, Verilog or M. For VHDL, a subset must be defined as well as modeling guidelines, leading thus to the definition of tool-dependent "synthesis semantics".

### 3.2. Application Domains

VHDL has the most general purpose and can be used in a wide range of domains: integrated circuits, printed circuit boards and even system modeling (**table 4**). The other languages are more focused on IC design, especially UDL/I which is completely dedicated to integrated circuits ("I" in "UDL/I" stands for Integrated circuits).

### 3.3. Language Reference Manuals

The document defining the syntax and the semantics of a hardware description language is very important. It is useful to the designer, who relies on it to perform correct and efficient modeling, and to the tool builder, who needs clear definitions to develop bug-free simulators or other tools.

The M language is a proprietary language : a very good documentation is available [5], but this is not a tool independent reference manual. This documentation has thus not been included into the comparison.

The other three HDLs are defined by a tool independent Language Reference Manual (**table 5**). The VHDL reference manual [2] is hard to read and almost designer hostile : too much "language" and not enough "hardware", very few examples... On the contrary, it contains many information for the tool builder (a canonical event-driven VHDL simulator is described : driver mechanism, elaboration, initialization, simulation cycles). Definitions, although being difficult to find in the manual, are also very accurate.

The Verilog reference manual [3] has quite opposite characteristics. It is very easy to read and provides a lot of examples having straightforward meaning for the designer. But some definitions are not very accurate, and the tool builder's point of view is not really handled by the manual.

It is more difficult to estimate the UDL/I manual [4] because the used release was not the definitive one (some parts not yet translated from the original Japanese manual).

## 4. MODELING STYLES

### 4.1. Levels of Description

Practically, VHDL is not very usable at low levels of description (**table 6**). Some papers and books illustrate the use of VHDL at switch level, but the efficiency is questionable. Electrical or analog modeling is not implemented at all in VHDL: an extension for mixed-mode simulation is under study and should appear by 1994.

All other languages provide efficient switch level modeling, and M even includes analog modeling. Verilog and M provide predefined gate and switch devices (with delay and/or strength parameters). Constructs implemented by these languages at low levels are very specific to integrated circuit modeling.

On the contrary, VHDL is more powerful than the other HDLs at high levels of description. Actually, it includes software programming capabilities that are close to a subset of ADA. Following is a list of features that make VHDL particularly suitable for high-level modeling and that are not implemented in the other three HDLs :
- strongly typed language (consistency checking)
- separate compilation
- packages and libraries (notion of reusability)
- overloading mechanism
- user-defined I/O operations (TEXTIO package)
- user-defined types: enumeration, physical, array, record...
- user-defined resolution functions
- separation between entity and architecture(s)
- multiple architectures for the same entity
- configuration mechanism (entities and components)

Compared to the others, UDL/I is very weak in functional modeling. Indeed, this language does not provide the designer with the ability of defining his or her own subprograms, and furthermore, it does not implement sequential statements at all (see 4.3).

## 4.2. Design Units

According to the VHDL Language Reference Manual, the design units are the constructs which may be separately analyzed and inserted into a design library (**figure 1**).

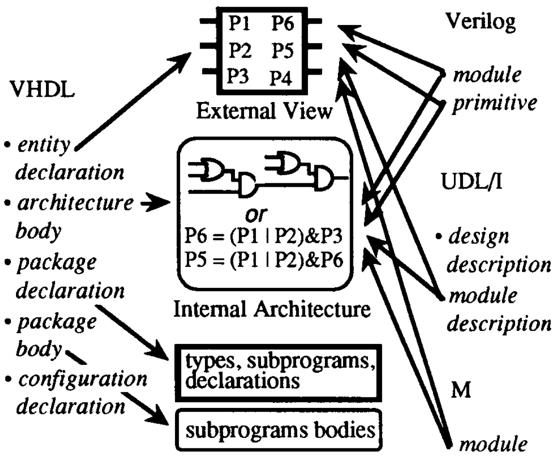


**Figure 1 : Design Units**

VHDL defines five design units : the *entity declaration* and the *architecture body* which may be considered respectively as the external and internal views of a *hardware* unit, the *package declaration* and the *package body* which may be seen as the external and internal views of a *software* unit (but not only, because packages may also declare signals and components!), and the *configuration declaration.* All these five units may be separately analyzed and stored in design libraries.

The other HDLs only provide the *module* which is conceptually equivalent to a pair entity-architecture : there is no separation between external and internal descriptions, and the internal architecture of a module is unique (except for UDL/I which enables to define different descriptions for different purposes: simulation, synthesis...).

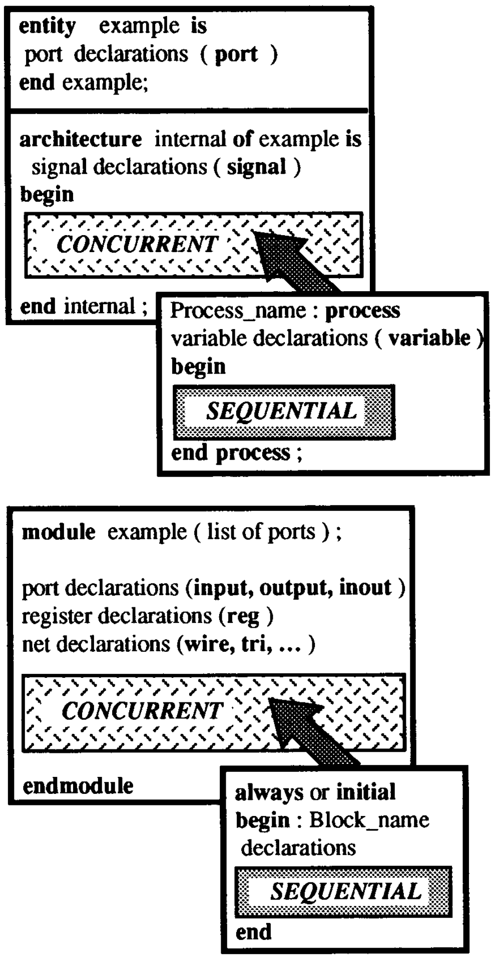
Packages are not implemented in Verilog, UDL/I and M. This VHDL construct is very useful because it increases the factorisation of the modeling effort (separate compilation, reusability of types, subprograms or components).

## 4.3. Concurrent and Sequential Domains

Hardware description languages usually implement a concurrent domain and a sequential domain (**table 7**). The former includes statements that are executed asynchronously, without defined order, and is mainly used for dataflow and structural descriptions. The latter includes statements that are executed in order and is used for algorithmic descriptions (definitions given in the VHDL reference manual).

VHDL and Verilog have similar implementations of sequential and concurrent domains (**figure 2**):

- Verilog modules and VHDL architectures (blocks) include concurrent statements;
- in both languages, concurrent statements may be used for structural descriptions (Verilog module or VHDL component instantiation), dataflow descriptions (Verilog continuous assign or VHDL concurrent signal assignment) and behavioral descriptions (Verilog always and initial statements or VHDL process statements);
- Verilog always (or initial) statements and VHDL process statements both encapsulate sequential algorithms, which can also be embedded inside subprograms in both languages.



entity example is
port declarations ( port )
end example;
architecture internal of example is
signal declarations ( signal )
begin
CONCURRENT
end internal ;
Process_name : process
variable declarations ( variable )
begin
SEQUENTIAL
end process ;
module example ( list of ports ) ;
port declarations (input, output, inout )
register declarations (reg )
net declarations (wire, tri, ... )
CONCURRENT
endmodule
always or initial
begin : Block_name
declarations
SEQUENTIAL
end

**Figure 2 : Concurrent & Sequential Domains in VHDL and Verilog**

The behavioral concurrent domain of M is restricted to the use of kind of UNIX coprocesses : there is no dataflow description in M. In this language, modules mainly include sequential code that is very close to C code.

UDL/I is very different from other HDLs. Indeed, this language does not implement any sequential domain. All UDL/I statements are concurrent statements. This language provides thus powerful dataflow descriptions (including asynchronous assignments as well as synchronous edge-sensitive or level-sensitive assignments), but high level modeling capabilities are very limited.

## 4.4. Objects and Types

The generality of VHDL as well as the specialization of other HDLs in IC modeling are particularly emphasized by the differences between their object and type definitions (**table 8**). Four classes of objects are defined in VHDL: signal, variable, constant and file objects. Each object has a given type: this type may be a scalar type (integer, floating point, physical, enumeration), a composite type (array, record), an access type or a file type (the two last ones only for variables). VHDL signals and variables can be declared with any user-defined bit logic or abstract type.

On the other hand, Verilog and UDL/I only provide a very restrictive set of types, which are very specific to IC modeling. In particular, signals (wires, nets...) are implicitly defined with the classical logic type (X, 0, 1, Z) and associated to one of the predefined resolution functions. Verilog also provide integer, real and time variables for use inside a module. It is thus impossible to define signals of abstract types in Verilog and UDL/I, which is very useful for system modeling.

The M language originally handled only signals with an implicit logic type (levels X, 0, 1 and strengths), but the possibility of defining signals with abstract datatypes and user's resolution functions has been added. Memory variables (equivalent to VHDL variables declared in a process) and C variables (equivalent to VHDL variables declared in a subprogram) can be declared inside modules with any C type.

Even if VHDL has more powerful type definitions, the IC designer is often disappointed when he or she starts modeling with this language. The reason is that the only predefined bit logic type in VHDL is the enumeration type BIT with two values : '0' and '1' (and without resolution function). The implicit logic type provided by other HDLs is more adapted for IC modeling than this BIT type. The VHDL IC designer is therefore almost obliged to define or to reuse another logic type.

This issue has been addressed by the IEEE Model Standards Group which has developped a multi-value logic package STD_LOGIC_1164 [6]. This logic system is based on the following nine-state enumeration type:

```
type STD_ULOGIC is (
    'U',      -- Uninitialized
    'X',      -- Forcing 0 or 1
    '0',      -- Forcing 0
    '1',      -- Forcing 1
    'Z',      -- High Impedance
    'W',      -- Weak 0 or 1
    'L',      -- Weak 0
    'H',      -- Weak 1
    '-'       -- don't care   ) ;
```

Using this standardized VHDL logic system (close to the implicit logic type of other HDLs) significantly increases model interoperability.

## 4.5. Predefined Environment

The necessity shown above to build or reuse an extended bit logic type in order to achieve realistic IC modeling introduces another main characteristic of VHDL: its small number of predefined constructs (**table 9**).

Indeed, the VHDL language has a very poor set of predefined subprograms or types, and does not provide any predefined gate : this is again the counterpart of its general modeling purpose. The other languages have a lot of predefined gates or operators, which are in turn very oriented toward IC modeling.

Actually, VHDL may be seen as a *kit to build its own HDL*. The general philosophy of this language is to let the user define and build (or let build) all the specific constructs he or she needs : multi-value logic type, abstract data types, subprograms and operators, generic and/or technology dependent components and entities...

Therefore, an efficient and necessary methodology for using VHDL in a company is to determine all these constructs that are specific to the target application or domain and to develop (or let develop) the corresponding VHDL libraries. Once such an *environment* is set, then most of the designers will just have to reuse these "predefined" constructs (not predefined in the language, but in the built environment). Due to the language capabilities of VHDL, such a specific environment can be even more powerful than the implicit environment provided by other HDLs.

| | *VHDL* | *Verilog* | *UDL/I* | *M* |
|---|---|---|---|---|
| *Origin* | IEEE Standard | Cadence Design Systems, Inc. | Japanese Standard | Mentor Graphics (originally SCS) |
| *Status* | public domain | proprietary, then public domain | public domain | proprietary |
| *Designers Acceptance* | emerging | widely used | unknown | widely used |

**Table 1 : Origins and Status**

| | *VHDL* | *Verilog* | *UDL/I* | *M* |
|---|---|---|---|---|
| *Syntax close to* | ADA | C, Pascal | Nothing | C |
| *Level of Complexity* | Difficult | Easy | Average | Easy |
| *Straightforward Hardware Meanings* | Not really | Yes | Yes | Yes |
| *Requires Software Background* | Yes | No | No | Yes |

**Table 2 : Main Characteristics**

| ***Semantics of*** | *VHDL* | *Verilog* | *UDL/I* | *M* |
|---|---|---|---|---|
| *Simulation* | Yes | Yes | Yes | Yes |
| *Synthesis* | No | No | Yes | No |

**Table 3 : Language Semantics**

| ***Modeling what ?*** | *VHDL* | *Verilog* | *UDL/I* | *M* |
|---|---|---|---|---|
| *IC* | ◆◆ | ◆◆ | ◆◆ | ◆◆ |
| *PCB* | ◆◆ | ◆ | ◆ | ◆ |
| *System* | ◆ | | | |

**Table 4 : Application Domains**

| ***Language Reference Manual*** | *VHDL* | *Verilog* | *UDL/I* | *M* |
|---|---|---|---|---|
| *Tool independent* | Yes | Yes | Yes | No |
| *Designer friendly* | No | Yes | Yes | - |
| *Tool builder friendly* | Yes | Average | Average | - |
| *Accuracy of definitions* | Good | Average | Average | - |

**Table 5 : Language Reference Manuals**

| ***Level of Description*** | *VHDL* | *Verilog* | *UDL/I* | *M* |
|---|---|---|---|---|
| *Functional Level* | ◆◆ | ◆ | | ◆ |
| *Behavioral Level* | ◆◆ | ◆◆ | ◆ | ◆◆ |
| *Register Transfer Level* | ◆◆ | ◆◆ | ◆◆ | ◆◆ |
| *Logic Level* | ◆◆ | ◆◆ | ◆◆ | ◆◆ |
| *Switch Level* | | ◆◆ | ◆◆ | ◆◆ |
| *Electrical Level* | | | | ◆◆ |

**Table 6 : Levels of Description**

| ***Implementation of*** | *VHDL* | *Verilog* | *UDL/I* | *M* |
|---|---|---|---|---|
| *Concurrent Domain* | ◆◆ | ◆◆ | ◆◆ | ◆ |
| *Sequential Domain* | ◆◆ | ◆◆ | | ◆◆ |

**Table 7 : Concurrent & Sequential Domains**

| ***Objects*** | *VHDL* | *Verilog* | *UDL/I* | *M* |
|---|---|---|---|---|
| *VHDL Variable, Verilog Register, UDL/I Register, M Variable* | any user-type | X,0,1,Z or array, integer,real, time | X,0,1,Z or array | any C-type |
| *VHDL Signal, Verilog Net, UDL/I Terminal, M Terminal* | any user-type except file access | X,0,1,Z or array | X,0,1,Z or array | LOGIC or array, any user-struct |

**Table 8 : Objects and Types**

| ***Predefined Constructs*** | *VHDL* | *Verilog* | *UDL/I* | *M* |
|---|---|---|---|---|
| *Basic Gates* | | ◆◆ | | ◆◆ |
| *Logic Operators* | ◆ | ◆◆ | ◆ | ◆◆ |
| *Arithmetic Operators* | ◆ | ◆ | ◆◆ | ◆ |
| *Resolved Logic Type* | | ◆◆ | ◆◆ | ◆◆ |

**Table 9 : Predefined Constructs**

## 5. LANGUAGE KEY-POINTS

Typical models written in the four hardware description languages are detailed and compared in [1].

### 5.1. Structural Descriptions

A structural description in VHDL looks very different from a structural description written in one of the other HDLs. The main reason is the configuration mechanism which is only implemented by VHDL.

In VHDL, entities are plugged into components (components are configured with couples entity-architecture) which are in turn instantiated into architectures. Components must be declared before their instantiation : these declarations may be embedded in packages in order to allow the reusability of these components, or they may simply appear in the architecture declarative part.

Configuration specifications or declarations should be added to complete the model, unless the default configuration mechanism is used.

In other HDLs, modules are directly instantiated into other modules (no component, no configuration). The advantage is that structural descriptions are often less verbose than in VHDL (no component declarations), but the drawback is the lack of flexibility compared to VHDL. In the latter language, netlist descriptions may be written independently from the used entities, allowing thus to switch easily from one library to the other (from one technology to the other) when choosing the plugged entities. Adaptations can also be performed between components and entities (type conversions, open ports).

All HDLs, except UDL/I, allow generic parameters (timing, bus size...) to be specified for entities or modules.

### 5.2. Dataflow Descriptions

The M language does not implement concurrent assignment statements, which makes dataflow descriptions impossible in this language.

The other three HDLs provide similar dataflow description capabilities: each concurrent signal assignment statement is executed (asynchronously) whenever a signal appearing in the assigned expression changes its value. VHDL, Verilog and UDL/I all implement a delta-delay mechanism to perform zero-delay assignments.

In all four languages, timings can be specified in signal assignments. Verilog and UDL/I only provide inertial delays, whereas VHDL and M implement both inertial and transport delays.

### 5.3. Behavioral Descriptions

VHDL, Verilog and M implement powerful sequential statements. In each of these three languages, portions of sequential code can be encapsulated inside concurrent "boxes" (VHDL processes, Verilog initial or always blocks, M modules) that are executed asynchronously with other concurrent boxes.

Sequential assignment statements can be specified in these three HDLs, using the same kind of delays as those specified for dataflow descriptions.

UDL/I does not implement sequential statements : all UDL/I statements are concurrent. This language does not have constructs equivalent to the VHDL processes.

On the other hand, UDL/I is the only HDL that implement a specific construct, the *automata*, to model finite state machines. Such devices can be described in the other HDLs, but their hardware (synthesis) interpretation is not as straightforward as in UDL/I (explicit clock, reset, states, transistions and commands).

Verilog and UDL/I also provide a convenient and efficient truth-table syntax to model primitive gates.

## 6. CONCLUSION

First of all, modeling integrated circuits is more than possible with all four hardware description languages. Of course, some features are not available in all languages, but there is always a work-around to achieve a given model. Concerning the modeling power, we may say in a few words that VHDL is weak at low levels of description (analog, switch and gate) but very strong at high levels of description (behavioral, functional).

But the main differences between VHDL and the other HDLs are more methodological differences. VHDL is a general purpose modeling language, whereas Verilog, UDL/I and M are more dedicated to IC modeling. The predefined environment of VHDL will look thus very poor compared to the implicit IC environment of other languages. There is thus a necessity to develop (or let develop) a VHDL environment specific to the application domain before starting the real use of VHDL in a design project. Once such a environment is set, then VHDL capabilities may be compared to the capabilities of other specialized HDLs.

## 7. REFERENCES

[1] "VHDL Designer's Reference"
J.M.Bergé, A.Fonkoua, S.Maginot, J.Rouillard
Kluwer Academic Publishers, May 1992

[2] IEEE Standard VHDL Language Reference Manual
IEEE, March 1988

[3] "Verilog Hardware Descript. Lang. Reference Manual"
Version 1.0, October 1991, Open Verilog International

[4] "UDL/I Language Reference", V1.0h4, May 1991
Japan Electronic Industry Development Association

[5] "M Language Users Guide & Reference"
Version 4.0, May 1989, Mentor Graphics

[6] "STD_LOGIC_1164 : Technical Overview"
W.D.Billowitch, Chairman
IEEE VHDL Model Standards Group, May 1991

[7] *(in French)* "VHDL, du langage à la modélisation"
R. Airiau, J. M. Bergé, V. Olive et J. Rouillard
Presses Polytechniques Universitaires Romandes, 1990