VHDL Circuits Hardware Description Language: Notes

Hachour Ouarda

?

Abstract—Field Programmable Gate Array (FPGAs) are usually programmed using languages and methods inherited from the domain of VLSI (Very Large Scale Integration) synthesis. These methods, however, have not always been adapted to the new possibilities opened by FPGA, nor to the new constraints do they impose on a design. For FPGA circuit we can use the VHDL language as hardware description (acronym for Very High Speed Integrated Circuits Hardware Description Language). The key of the art design is focused around high level synthesis which is a top down design methodology that transforms an abstract level using VHDL description. the synthesis tools allow designers to realize the mainly reasons: the need to get a correctly working systems at the first time, technology independent design, design reusability, the ability to experiment with several alternatives of the design, and economic factors such as time to market. VHDL allows for the description of hardware behavior from system to gate levels . To fit this level of description, the language has to offer lager degrees of abstraction, powerful algorithmic, wide capabilities for merging different description levels, and an easiness expression of causality, and also the possibility of introducing non determinism, which may be an interesting feature. To date, this level of description has not been synthesizable: no explicit architecture is described and no tool on the market offers a real and an efficient architectural synthesis (except for some specific target architecture. In this paper we present some useful notes of VHDL: main hardware concept of VLSI.

Keywords— Field Programmable Devices (FPD), (FPGAs) Field Programmable Gate Arrays, ASICs (Application Specific Integrated Circuits), VLSI (Very Large Scale Integration), VHDL (acronym for Very High Speed Integrated Circuits Hardware Description Language).

I. INTRODUCTION

Field Programmable Gate Array (FPGAs) are usually programmed using languages and methods inherited from the domain of VLSI (Very Large Scale Integration) synthesis [7,8,9,10]. These methods, however, have not always been adapted to the new possibilities opened by FPGA, nor to the new constraints do they impose on a design. For FPGA circuit we can use the VHDL language as hardware description (acronym for Very High Speed Integrated Circuits Hardware Description Language) [1, 2,3,4,5].

The key of the art design is focused around high level synthesis which is a top down design methodology that transforms an abstract level using VHDL description. the synthesis tools allow designers to realize the mainly reasons: the need to get a correctly working systems at the first time, technology independent design, design reusability, the ability to experiment with several alternatives of the design, and economic factors such as time to market.

VHDL allows for the description of hardware behavior from system to gate levels. The system level focuses on the description of the functionalities of the system (what is does) and tries to avoid it implementation description (how it is constituted).

The notion of time is essentially a notion of causality: one action implies another. A constant is to forge useless details, which would imply architectural choices too early in the design methodology. Too detailed a system description is a drawback for it restricts further architectural choices or implies a given technology. Therefore, hiding the information structure is desirable and the notion of concurrency may not be necessary at this phase.

A VHDL description is never monolithic, modularity is everywhere. The first structuring level is the design unit. When compiling a VHDL source file, this file notion does not exit after compilation: each contained design unit once successfully analyzed is independently stored within a VHDL library.

The only structuration of the original file which exists after compilation in the VHDL world is the notion of design unit. A VHDL source file is seen a collection of design units. A design unit link between design units written in the same source files, and there is no implicit link between design units written in the same source file.

The kinds of design unit can be roughly split into two parts: those that describe the hardware hierarchy (i.e., the structure of the model) and those that are more of software oriented. This operation is very important. An orthogonal classification shows that three deign unit are the external world. A secondary unit is the implementation (internal view) of its primary. A varying number of design unit, possible of different kinds, constitute a library each design unit is self –compilable but may use objects of other design unit, possibly stored in other libraries. In this present work we present some useful notes of VHDL concepts.

In this paper we present some useful notes of VHDL: main hardware concept of VLSI. We present the main description and the advantages of using this language. We discuss some examples of structure of the language and we clarify why the importance of using this VLSI describe language.

II. VHDL CONCEPT

VHDL language as hardware description **(**acronym for Very High Speed Integrated Circuits Hardware Description Language). The key of the art design is focused around high level synthesis which is a top down design methodology that transforms an abstract level using VHDL description. the synthesis tools allow designers to realize the mainly reasons: the need to get a correctly working systems at the first time, technology independent design, design reusability, the ability to experiment with several alternatives of the design, and economic factors such as time to market.

VHDL allows for the description of hardware behavior from system to gate levels. The system level focuses on the description of the functionalities of the system (what is does) and tries to avoid it implementation description (how it is constituted). The notion of time is essentially a notion of causality: one action implies another. A constant is to forge useless details, which would imply architectural choices too early in the design methodology. Too detailed a system description is a drawback for it restricts further architectural choices or implies a given technology. Therefore, hiding the information structure is desirable and the notion of concurrency may not be necessary at this phase.

To fit this level of description, the language has to offer lager degrees of abstraction, powerful algorithmic, wide capabilities for merging different description levels, and an easiness expression of causality, and also the possibility of introducing non determinism, which may be an interesting feature. To date, this level of description has not been synthesizable: no explicit architecture is described and no tool on the market offers a real and an efficient architectural synthesis (except for some specific target architecture).

A VHDL design begins with an ***ENTITY*** block that describes the interface for the design. The interface defines the input and output 1ogic signals of the circuit being designed. The ***ARCHITECTURE*** block describes the internal operation of the design, see the figure 1. Within these blocks are numerous other functional blocks used to build the design elements of the logic circuit being created.

The ***IEEE STD_LOGIC_1164*** standard includes additional definitions for VHDL data types. For the bit type, the IEEE type is ***STD_LOGIC*** and for a bit_vector it is ***STD_LOGIC_VECTOR***. The use of the IEEE standard types assures that your VHDL code will be **portable**. That is it can be used by any vendors implementation software. Bit and bit-vector are types which are not universally accepted and may not be recognized by some application programs.

After the design is created, it can be simulated and synthesized to check its logical operation. **SIMULATION** is a bare bones type of test to see if the basic logic works according to design and concept. **SYNTHESIS** allows timing factors and other influences of actual **FPGA** devices to effect the simulation thereby doing a more thorough type of check before the design is committed to the **FPGA** or similar device.

Many software packages used for VHDL design also support schematic capture which takes a logic schematic or state diagram and translates it into VHDL code. This, in turn, makes the design process a lot easier. However, to fine tune any design, it helps to be familiar with the actual VHDL code. VHDL is a very **strongly** typed language. It does not allow a lot of intermixing of data types. The idea here is that since we are describing a piece of hardware, you need to keep things like signals and numbers separate. We shall start by looking at the different types of data that can be used with VHDL which include bits, buses, Boolean, strings, real and integer number types, physical, and user defined enumerated types.

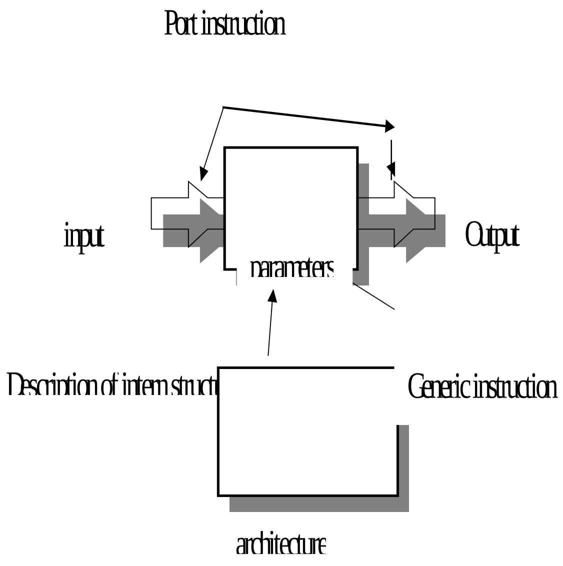

Fig. 2 A VHDL language description

A. Entity block

An **entity block** is the beginning building block of a VHDL design. Each design has only one entity block which describes the interface signals into and out of the design unit. An ***entity*** is the external view of a model: ports (inputs/outputs) and parameter (named generic), as well as static check on parameter values (such a range or minimal value verification) and dynamic check on port (such a set –up or hold time verification) are described in this design unit

The syntax for an entity declaration is:

```
    entity entity_name is

    port (signal_name, signal_name  : mode type;
              signal_name,signal_name : mode type);
  end entity_name;
```

An entity block starts with the reserve word **entity** followed by entity_name. Names and identifiers can contain letters, numbers, and the underscore character, but must begin with an alpha character. Next is the reserved word **is** and then the **port** declarations. The indenting shown in the entity block syntax is used for documentation purposes only and is not required since VHDL is insensitive to white spaces.

A single **PORT** declaration is used to declare the interface signals for the entity and to assign **MODE** and data **TYPE** to them. If more than one signal of the same type is declared, each identifier name is separated by a **comma**. Identifiers are followed by a **colon (:)**, mode and data type selections.

In **VHDL,** ALL STATEMENTS are terminated by a **semicolon.** The entity declaration is completed by using an **end** operator and the entity name. Optionally, you can also use an **end entity** statement. In general, there are five types of modes, but only three are frequently used. These three will be addressed here, they are **in, out,** and **inout** setting the signal flow direction for the ports as input, output, or bidirectional. Signal declarations of different mode or type are listed individually and separated by **semicolons (;)**. The last signal declaration in a port statement and the port statement itself are terminated by a semicolon on the outside of the port's closing parenthesis.

We can define a **literal constant** to be used within an entity with the ***generic*** declaration, which is placed before the port declaration within the entity block. Generic literals than can be used in port and other declarations. This makes it easier to modify or update designs. For instance, if we declare a number of bit vector bus signals, each eight bits in length, and at some future time you want to change them all to 16-bits, you would have to change each of the bit vector range. However, by using a generic to define the range value, all you have to do is change the generic's value and the change will be reflected in each of the bit vectors defined by that generic. The syntax to define a generic is presented as follow:

generic *(name : type := value);*

The reserved word ***generic*** defines the declaration statement. This is followed by an identifier name for the generic and a colon. Next is the data type and a literal assignment value for the identifier. **: =** is the assignment operator that allows a literal value to be assigned to the generic identifier name. This operator is used for other assignment functions. For example, here is the code to define a bus width size using a generic literal.

entity my processor is generic (bus Width: integer := 7);

B. Architecture block

The structure of a model, its behavior, or any mixture of both structure and behavior are described within it architecture. The syntax of such a design unit is

Architecture A ***of*** entity-b **is**
Architecture _Declarative_part
Begin
….
End

The **architecture block** defines how the entity operates. This may be described in many ways, two of which are most prevalent: ***STRUCTURE*** and ***DATA FLOW* or *BEHAVIOR*** formats. The ***BEHAVIOR*** approach describes the actual logic behavior of the circuit. This is generally in the form of a Boolean expression or process. The **STRUCTURE** approach defines how the entity is structured - what logic devices make up the circuit or design. The general syntax for the architecture block is:

architecture *arch_name* ***of*** *entity_name* ***is***
declarations;
begin
statements defining operation ***;***
end *arch_name****;***

III. LEVEL DESCRIPTIONS

The language must allow a description of the model at this level with a sufficient level of abstraction toward the physical level. Clock, sequentiality, dataflows, and combinational art have to be easily expressed. A large degree of parameterization is also required. The figure 2 shows the different possible description levels presented as follow:

A. The system level

This system focuses on the description of the functionality of the functionality of the system (what is does) and tries to avoid its implementation description (how it is implies another). The notion of time is essentially a notion of causality: one action implies another.
A constant concern is to forget useless details, which would imply architectural choices too early in the design methodology.
Too detailed a system description is a drawback for it restricts further architectural choices or implies a given technology.
Therefore, hiding the information structure is desirable and the notion of concurrency may not be necessary at this phase.

B. The synthesizable level

This level is the potential input for synthesis tools. Here , some implementation choices have already been made
-an architecture, or at least an architecture family is targeted and is implied by the code structure.
-the widths of data paths are known.
Time can be expressed in terms of clock or sequentiality is executed after the previous one).

C. The netlist level

The netlist level is the potential output of synthesis output of synthesis tool. It is structural view appearing as a collection of model instantiation. This kind of description involves the existence of model libraries.

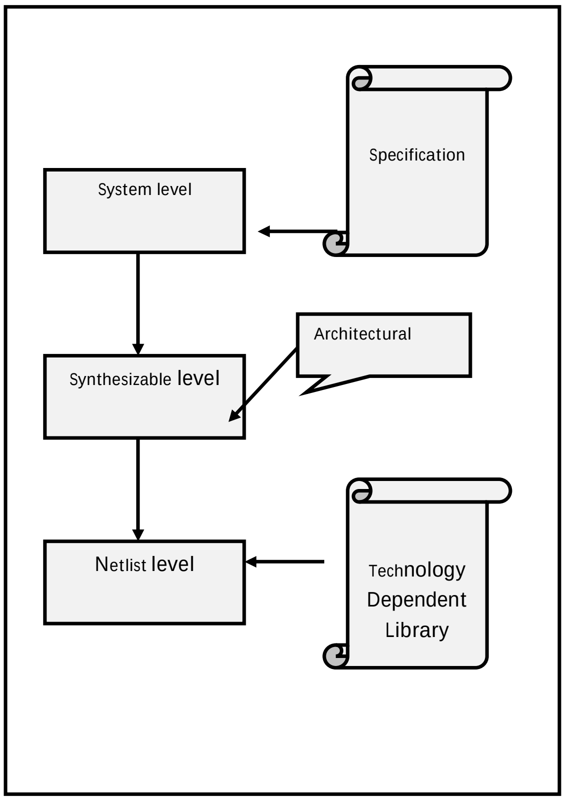

Fig. 2 Different level of Description

IV. VHDL STRUCTURAL

A. Describing structural

A digital electronic system can be described as a module with inputs and/ or outputs. The figure 5.7.a shows an example of this view of a digital system. The module F has two inputs *a* and *b* and an output Y using VHDL terminology. We call the module F the design entity and the inputs and outputs are called ports.

One way of describing the function of a module is to describe how it is composed of sub-modules. Each of the sub-modules is an instance of some entity, and the ports of instances are connected using signals. Figure 3 shows how the entity F might be composed of instances of entities G, H and I. This kind of description called a structural description. Not that the entities each G, H and I might also have a structural description.

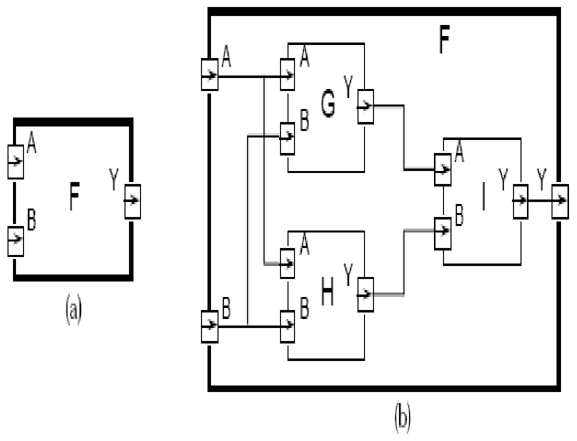

Fig. 3 Example of a structural description

B. Description behavior

In many cases, it is not appropriate to describe a module structurally. One such case is a module which is at the bottom of the hierarchy of some other structural description.

For example if you have describing a system using IC package bought from an IC shop we do not need to describe the internal structure of IC. In such cases, a description of the function performed by the module is required, without reference to its actual internal structure. Such a description is called a functional or behavioral description.

To illustrate this, supposed that the function of the entity in Figure 3 is the exclusive or function. Then the a behavioral description of F could be the Boolean function

$$Y = A.\overline{B} + \overline{A}.B$$

More complex behaviors cannot be described purely as a function of inputs. In such system with feedback, the outputs are also a function of time. VHDL solves this problem by allowing description of behavior in the form of an executable program.

More than one architecture can be associated with a given entity but only one is selected for each model instantiation at simulation time. Allowing multiple architecture for the same entity is of great interest.

For example, after synthesis, it is possible to compare the two architectures (before and after synthesis) by instantiating them (taking a copy of them) in two different instances. Certain synthesis tools even propose different architectures as output of the synthesis process. Each one is optimized for a given purpose: accurate simulation, quick simulation, or hardware acceleration.

An architecture is implicitly dependent on its associated entity: all objects defined in the entity are known within the architecture. Therefore, as shown in figure 4 ports (here NQ and Q) are seen as signals and can be assigned within the architecture.

To fit this level of description, the language has to offer lager degrees of abstraction, a powerful algorithmic, wide capability for merging different description levels, an easy expression of causality, and also the possibility of introducing non determinism, which may be an interesting feature

To date, this level of description has not been synthesizable: no explicit architecture is described and no tool on the market offer a real and efficient architectural synthesis (except for some specific target architecture).

The synthesizable level is the potential input for synthesis tools.

Architecture, or at least an architecture family is targeted and is implied by the code structure.

The widths of datapath are known.

Time can be expressed in term of clock or sequentially (one statement is executed after the previous one).

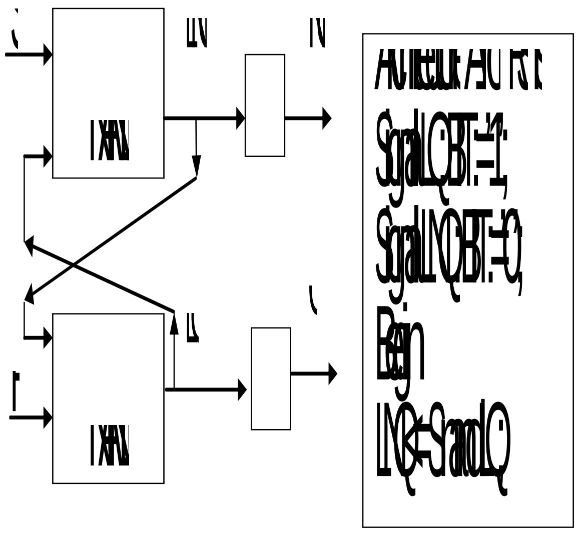

Fig. 4 . An example of Structural architecture at Gate-level

V. VHDL NOTES

We note that in the VHDL, the notion of time is often present in the description of these models, from the notion of propagation delay trough a gate to very sophisticate delays (using slopes, temperature, etc.). These delays are either provided by the library and are therefore only indicative, or are relevant to a given technology and therefore more accurate. They can even be deduced from the enlist using a back notation mechanism probably outside the VHDL world). At this step, the language has to offer an optimal flexibility in terms of timing configuration or technology. These two aspects are most often linked. The following sections describe more.

A. *The notion of time*

The notation of time, which is carefully described in the LRM, is only related to simulation. This time is discrete: Only event have a date. Indeed, the notion of time does not exist between two of these date. The simulator is event driven: it kip from one event to the following one without exploring what happen in between.

No synthesis semantic is defined in the VHDL LRM. Therefore, it is not possible to directly express implementation timing constraints using the language. The only exiting notion is the delay: "this output takes this value after this exact delay". Moreover, no MIN / MAX simulation mechanism is deduced in the language.

B. *Modularity*

A VHDL description is never monolithic, modularity is everywhere. The first structuring level is the design unit. When compiling a VHDL source file, this file notion does not exit after compilation: each contained design unit once successfully analyzed is independently stored within a VHDL library [6, 11].

The only structuration of the original file which exists after compilation in the VHDL world is the notion of design unit. A VHDL source file is seen a collection of design units. A design unit link between design units written in the same source files, and there is no implicit link between design units written in the same source file .The kinds of design unit can be roughly split into two parts: those that describe the hardware hierarchy (i.e. , the structure of the model) and those that are more of software oriented. This operation is very important.

An orthogonal classification shows that three deign unit are the external world. A secondary unit is the implementation (internal view) of its primary. A varying number of design unit, possible of different kinds, constitute a library each design unit is self –compilable but may use objects of other design unit, possibly stored in other libraries.

C. *Portability*

Portability was one of the main guidelines during the VHDL language design phase. The widespread uses of this language is mainly due to the fact that it is a standard. A standard is the only way for users to be free of the potential precariousness of a proprietary language.

New prospect in the next new years; new synthesis techniques will have more and more impact on system design methodology. The synthesis process by itself is not the source of such a modification. Synthesis is only of the potential targets of a hardware description.

Modeling (“modelware”, that is the act to describe the behavior of a system as a whole) , is really be changing design methods. The remarkable points is that a single language. VHDL, is now able to cover the entire design cycle from functional specification to low—level structural design

D. *Notion of component*

To be more general, VHDL offers a general and flexible mechanism: the instantiation is applied to the idea we have of a model in the case of direct instantiation to the model itself.

It is therefore possible, either to directly instantiated a pair entity/ architecture or to initiate an intermediate object called component. Indeed, this notion of component represents the external view of a desired model. This desire may be quite different from the actual external view of the model we will finally use, and adaptation mechanisms are provided.

There is no behavior attached to the notion of component, but it has on be great advantage: it allow compilation. So, many static checks may be performed even if the entity/ architecture pair that will finally be used is not known (or does not even exist yet).

This mechanism allows for a top-down methodology with real decoupling between the component library (what is desired) and the model library (what we have).

Using the notion of component implies three fundamental operations: declaration, instantiation, and configuration of the component. Fortunately, and especially in the logic synthesis domain, any tool generate this source code automatically.

Nevertheless, considering the component declaration as a simple redundancy of the entity declaration error. When using already exiting libraries or design units written by somebody else, the power of the component notion appears obvious adaptation is possible.

Continuing our analogy, this operation can be seen a plugging a circuit into a socket. Each socket corresponds to a component instantiation. Adapting the socket to the circuit is possible during this operation.

The flexibility provided by the notion of component is very powerful. Selecting an entity / architecture pair is possible very late in the design cycle (just before simulation or synthesis) and switching from one library to another to change one model into another is a straightforward operation.

E. *Library*

Libraries are a convenient way to store and retrieve designs, functions, procedures, and other commonly used items. The ***IEEE STD_LOGIC_1164*** *LIBRARY* contains definitions for many of the VHDL IEEE standard data types and logic functions. However, the use this library to create a logic design. VHDL has a built in library that is automatically accessed without a library or use statement. This library is specified by IEEE as **VHDL 1076** and contains some of the basic definitions of VHDL reserve words and operators.To access an existing library, we must declare the location of the library and which parts of it you want to use. The syntax for access the entire contents of the IEEE.STD_LOGIC_1164 is:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

The **library** statement denotes the **parent library** of files which contains defined functions, procedures, operators, and processes.

This declaration must precede your design. The **use** statement selects which groups of files from the parent library you want to use in your design. Each entity in your design must be preceded by the **use** clause if that section of the design file is to use the contents of the 1164 library. When you compile your design, the requested files from the library are accessed and added to your design.

In turn, at the completion of the compile function, the design is added to a working library, aptly referenced to by the reserved word *WORK*. The contents of this library are accessible by any design in the current open project as long as you indicate it use with the statement:

```
use work.package_name.all
```

PACKAGES are units that are defined at the beginning of the design to hold commonly used functions, procedures, constants, etc. Additionally, designs in the current file can be accessed from the library that the compile process stores completed designs into when they are successfully compiled.

As briefly mentioned earlier, **packages** are a way of storing commonly used items in a library file. If the package is to hold basic items like *constants* and *type* declarations then it is only required to define the package at the beginning of a design.

The package will be added to that design's library when it is compiled. To create a library strictly for holding packages, then do not include any designs (entity declarations). When it is compiled, the defined packages will be saved in the created library file. The syntax for a simple package declaration is:

```
package package_name is
package_contents;
end package_name
```

F. *Consistency between simulation and synthesis*

One of the main objectives of all methodology is to make the transformation between two steps as safe as possible for this. Synthesis consists of the transformation of an initial description, as abstract as possible, into a structural description using well identified hardware resources.

The power of description of VHDL is able, without any constraints, to support the two levels of description.

Indeed, there is no real problem at this stage, but much more trouble is related to the transformation technology

G. *Synthesis Design Cycle*

The synthesis modeling style is mainly characterized by reducing the VHDL possibilities to a subset in which:
-Delay expressions (after clauses, wait for statements) are ignored
-Certain restrictions on the writing of process statement occur
-Only a few types are allowed
-Description is oriented towards synchronous styles by explicating clocks.

Three phases may be distinguished in this design cycle: modeling for synthesis, synthesis process, and final validation

When performed well, these steps lead to an efficient design, they may be divided into design steps and validation steps. Validation steps are essential to ensure that the inferred hardware has the desirable properties whatever the quality of the synthesis tools. Indeed, differences may appear between the before-synthesis and after –synthesis tools. Indeed, differences may appear between the before-synthesis and after-synthesis behaviors.

The figure 5 illustrates the design cycle. Such a cycle may be used for ASIC design as well as for programmable circuit (FPGA, EPLD) design.

Three phases may be distinguished in this design: modeling for synthesis, synthesis process, and final validation

H. *Modeling for synthesis*

Modeling fir synthesis is the first phase at this level, the specifications have to be clear and are coded in VHDL. The modeling style used is important, and this modeling task consists mainly in finding a good tradeoff between the abstraction level and the accuracy of the description.

The choice of data types (integers, enumerated types, composite types, and so on) and the choice of VHDL constructs (subprograms, loop statements, and so on) are essential.

The most abstract is the description, and the easiest are its validation, its main trainability, and its understanding by somebody else. A more accurate description involves more details. These details usually have a negative effect on the reach ability and maintainability of the description but may significantly increase the control of the inferred hardware.

I. *Final validation*

This validation takes place after the place - and- route operation has been performed. Its goal is to check that the original functionality has been respect ted and that the timing characteristics of the generated hardware meet the timing constraints. Two methods of achieving this goal are possible:

-Performing the simulation within the proprietary environment. In this case, the simulator uses the values of the net loads computed after the place - and- route operation.

-Back annotating the VHDL netlist resulting from the synthesis process the characteristics deduced from the place - and- route operation. The great advantage of this second method is to be able to remain in the VHDL simulation framework.

It is interesting to note that information provided by place - and- route tools may be useful for synthesis tools. For example, more accurate information on the goal of a given port may allow an extra optimization.

To avoid synthesis routing cycles, which may possibly not coverage, a new approach is now proposed by the latest synthesis tools.

Their strategy mainly consists in heavily coupling synthesis and place - and- route tools. Place - and- route evaluations are available throughout the synthesis process and are taken into account during synthesis tool optimizations

The figure 5 illustrates the design cycle. Such a cycle may be used for ASIC design as well as for programmable circuit (FPGA, EPLD) design.

Three phases may be distinguished in this design: modeling for synthesis, synthesis process, and final validation
VHDL allows for the description of hardware behavior from system to gate levels. The system level focuses on the description of the functionalities of the system (what is does) and tries to avoid it implementation description (how it is constituted).

The notion of time is essentially a notion of causality: one action implies another. A constant is to forge useless details, which would imply architectural choices too early in the design methodology.

Too detailed a system description is a drawback for it restricts further architectural choices or implies a given technology. Therefore, hiding the information structure is desirable and the notion of concurrency may not be necessary at this phase.

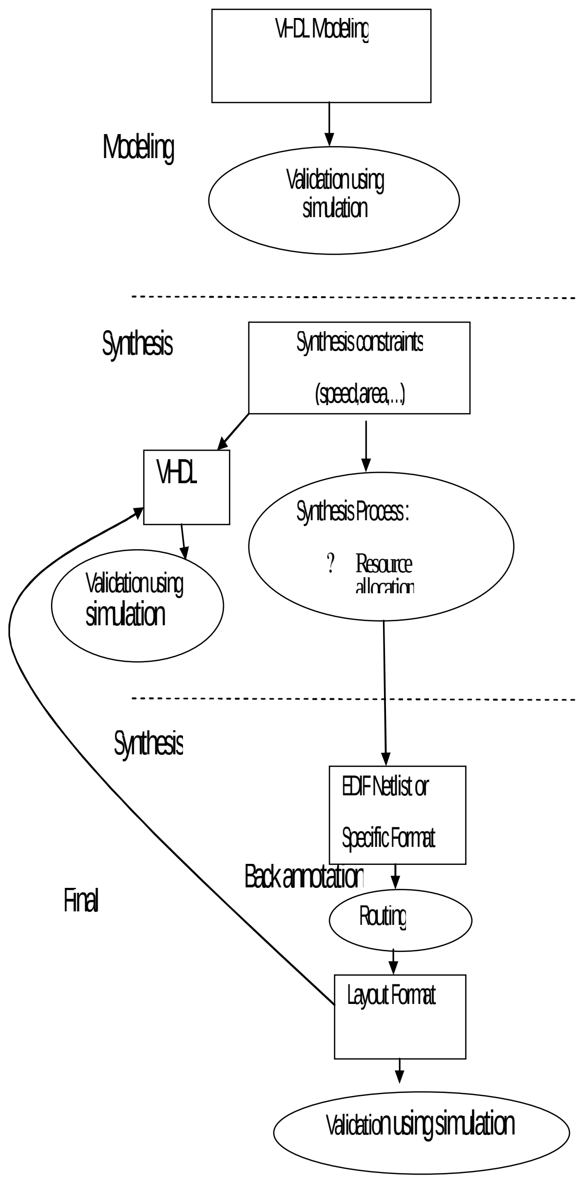
VHDL Modeling
Modeling
Validation using simulation
Synthesis
Synthesis constraints (speed, area,...)
VHDL
Synthesis Process :
? Resource allocation
Validation using simulation
Synthesis
EDIF Netlist or Specific Format
Back annotation
Final
Routing
Layout Format
Validation using simulation

Fig. 5 Design cycle flow chart

J. Conclusion

In this present work we have presented the main concept of VHDL acronym for Very High Speed Integrated Circuits Hardware Description Language. The key of the art design is focused around high level synthesis which is a top down design methodology that transforms an abstract level using VHDL description. the synthesis tools allow designers to realize the mainly reasons: the need to get a correctly working systems at the first time, technology independent design, design reusability, the ability to experiment with several alternatives of the design, and economic factors such as time to market.

VHDL allows for the description of hardware behavior from system to gate levels. The system level focuses on the description of the functionalities of the system (what is does) and tries to avoid it implementation description (how it is constituted). The notion of time is essentially a notion of causality: one action implies another. A constant is to forge useless details, which would imply architectural choices too early in the design methodology.

One of the main objectives of all methodology is to make the transformation between two steps as safe as possible for this. Synthesis consists of the transformation of an initial description, as abstract as possible, into a structural description using well identified hardware resources. The power of description of VHDL is able, without any constraints, to support the two levels of description.

This mechanism allows for a top-down methodology with real decoupling between the component library (what is desired) and the model library (what we have).

References

[1] O.Hachour, "The Proposed Genetic FPGA Implementation For Path Planning of Autonomous Mobile Robot", *International Journal of Circuits , Systems and Signal Processing,* Issue 2, vol2 ,2008,pp151-167.

[2] O. Hachour AND N. Mastorakis, Avoiding obstacles using FPGA –a new solution and application ,*5th WSEAS international conference on automation & information (ICAI 2004) , WSEAS transaction on systems* , issue9 ,vol 3 , Venice , Italy , , ISSN 1109-2777, November 2004, pp2827-2834

[3] G.R.Goslin, "using Xilinx field programmable Gate Array's (FPGA's) for Application –Specific Digital Signal Processing Performance", Xilinx Coprporate Application Group,pp150-153

[4] S.K.Knapp, INC & the ASM group, "using Programmable Logic to accelerate DSP functions" ,Xilinx, 1995,pp1-8

[5] XAPP 057 july7, 1996(version1.0)

[6] R. Airiau, J.M Berger, V. Olive, *Circuit synthesis with VHD*, Kluwer Academic Publishers, 1994.

[7] S.D.Brown, R.J., J.Francis Rose, and Z.G.Vranesic : *Field-Programmable Gate Array*, *Kluwer Academic Publishers*, 1997.

[8] M.Cummings and S.Haruyama, FPGA in the soft radio, *IEEE Communication Magazine*, 0163-1999, pp.108-112.

[9] *GALILEO HDL Synthesis Manual,* Exemplar Logic,1995.

[10] A.Gonzalez and R.Perez. : SLAVE : A genetic learning System Based on an Iterative Approach, *IEEE, Transaction on Fuzzy systems*, Vol 7, N.2, April 1999, pp.176-191.

[11] J.Legenhausen, R.Wade, C.Wilner, and B. Wilson,: *VHDL for programmable logic*, Addison- Wesley, 1996