WHERE VHDL FITS WITHIN THE CAD ENVIRONMENT

DR. JOHN HINES

AFWAL/AAD VHSIC PO, Wright-Patterson AFB OH 45433-6543

ABSTRACT

This paper discusses the relationship of the VHSIC Hardware Description Language (VHDL), the Electronic Design Interchange Format (EDIF), and the Initial Graphics Exchange Specification (IGES) in a Computer Aided Design (CAD) environment.

INTRODUCTION

There has been much discussion in the community on the positioning of the emerging standards. At one end of the spectrum there is an opinion that the standards of the VHSIC Hardware Description Language IEEE P1076[1], the Electronic Data Interchange Format[2], and the Initial Graphics Exchange Specification[3], are redundant and heavily overlapping in function and purpose. At the opposite end of the spectrum there are those who hold the opinion that they are orthogonal with little or no overlap. Then there is the middle of the road position that they are complimentary, each having its unique niche and purpose, as well as having appropriate areas of overlap.

It is the view of this author that most of the perceptions by the community have evolved from a misunderstanding of the underlying technology and rational for which the individual standards were based. It is human nature to criticize that which we do not understand. We feel threatened by it, hence criticism is the natural defense mechanism. Once we do understand the particular technology, it is generally the case that we made "much ado about nothing." It is the purpose of this paper to put the issues involved in perspective and to show that the emerging standards are actually complimentary in purpose.

The approach taken in this paper in analyzing the standards will be to briefly trace the historical origins and purpose for each standard. Their individual position within the CAD environment will also be discussed.

HISTORY

For the purposes of this paper, CAD is defined to have a scope encompassing design, modeling, documentation, logistics support, and test. The utilization of hardware description languages (HDLs) for design and documentation is a relatively new concept. Historically, they evolved from the university community and were not commercialized until 1982 or 1983. Today what most engineers have in mind when the word "CAD" is spoken is schematic entry and layout - a much narrower scope. In some sectors where mechanical design is also performed, the interpretation that is given to the word "CAD" is software for mechanical drawings.

The use of formal languages for design is still in an infant stage in industry. Taken as a group, only about 20% of engineers are currently using workstations, and of that 20% only about 10% have been exposed to non-structural HDLs. Evidence of this can be readily observed in the type of documentation that is predominant - schematics.

As in most technologies, HDLs more or less evolved to fill a need - modeling, evaluation, and testing through simulation of digital systems. In the early days of digital design it was adequate to do designs on paper and simulations by drawing timing diagrams on rolls of graph paper. In those early days all digital design was at a gate level. It did not take long, however, for designs to grow larger than a piece of paper. Using graph paper simulators was tedious for small designs and unmanagable for large designs. There was a need for a more formal way to describe the logic design and simulate the logic. To fill this need two basic tools had to be created. The first was a language that could describe the structure or interconnect the logic. The second was a logic simulator. Since design was at the gate level in those days, simulators were constructed along the lines of the paper methodology. That is, they were basically a set of fixed gate level models that could be connected using the structural language. This style of design was strictly bottom up from the gate level, and other views of the design object were impossible, eg: functional abstractions. In the mid 70's computing systems were viewed as classic von Neuman architectures. System engineers who were designing these programmable machines found the gate level view of the world too limiting. They were viewing the machine as an assembly language programmer would, at an Instruction Set Architecture view of the world. To allow computer architects to view and simulate this view, languages such as the ISPS language[4] at Carnegie-Mellon University were created. These

were niche languages in that their range of application was narrow. From a hardware modeling point of view, the model had to be developed as a whole. There was no notion of heirarchy in the sense of a structural language. Since the language viewed the object most conveniently as an aggregation of behavior and structure, other views were difficult or cumbersome to implement.

ISPS worked reasonably well at describing a machine as long as it was viewed as a single object. As components got more complex, and designers wanted to describe separate components and connect them, languages of this type were difficult to use. As a result, a structural interface was added to allow components to be described separately and connected for simulation. ISP', a dialect of ISPS, from CASE Western University and commercialized by ENDOT, is an example of this type language. This class of language worked well in its niche of modeling computing systems at a programmer's view, but efforts of extending these languages for broader applications have been difficult.

As components became more complex, hardware designers also wanted to view components at higher levels of abstraction. The concept that evolved was to view the components not as fixed function gates, but as behaviors. Thus, the fixed model simulation was replaced by a programmable simulator. The view was that the component behavior or function was like a computer program written to describe the function of the object. The structural language was then used to connect these behavioral models. The language that the behaviors were written in was actually a programming language extended with additional features to better support hardware descriptions and timing. Thus, these HDLs were actually dual-language systems comprised of a separate structural language and behavioral language. An example of this class of language is ADLIB/SABLE from Stanford University and commercialized by SILVAR LISCO as SDL/HELIX.

Since these languages were based on the philosophy that structure and behavior were separable, they had some difficutly handling the mixed views, such as the view that ISPS handled well.

USE OF VHDL IN THE DESIGN PROCESS

The next rung in the evolution of HDLs is represented by the VHDL (IEEE P1076). VHDL did not evolve, it was designed from a set of well formulated requirements. It also represents a departure from the evolutionary path that makes a distinction between behavior and structure. Unlike the two language systems previously discussed, VHDL is a unified language that makes no distinction between behavior and structure. This allows the language to describe an object with virtually any degree of structure or behavior with equal ease, thus overcoming the limitations of previous languages.

Because these languages are for description and simulation of hardware, they had to have a relatively well developed concept of time. However, most languages more or less left timing issues to the simulator rather than as a part of the formal language. VHDL diverges from that philosophy in that it has a precisely defined timing model with accompanying semantics as a part of the language definition. Having a well defined timing model insures that all simulators built for the language will give the same results for the same description.

Since HDLs are primarily used during the process, they interact directly with the designer. During this phase of development an engineer develops a design and then simulates it to evaluate its performance against some set of requirements. He will then iterate on that design until there will be many activities in parallel. There will be system engineers developing global models of a system and running system simulations. There may be an instruction set architecture programmers view. There may be hardware designers who require a functional block view. If components are designed as a part of the project, there will be several component views that need to be generated as a part of the design effort. One of the major problems in the past has been that different design groups had to use a different language depending on his area of design. One of the major goals of VHDL was that a single language would support all levels of design from system to gate, and at the same time have features in the language to help the designer organize and manage his design, thus promoting the human interface. In addition, powerful attribution capabilities are a part of the language to allow other types of data to be inserted into design descriptions.

Thus, HDLs are for direct communication with the designer and as such must be human readable. The nearest analogy to the relationship of HDLs to the designer is the relationship of a Higher Order Language (HOL) to a computer programmer. The HOL was created to bridge the gap between the machine, but for humans to communicate with each other and understand what the machine is to do. Similarly, HDLs are for humans to communicate with one another about hardware.

MANAGEMENT OF DESIGN DATA WITH VHDL

VHDL was specifically designed to assist a designer in viewing hardware in terms that are easily understood. In addition to supporting a broad range of design requirements, VHDL also helps a designer organize and manage a design. The basic abstraction in the langauge is the "design entity" which gaves a designer a "black box" model to visualize. A design entity has an interface that essentially defines the input/output ports of the object. The "Architectural Body" for the entity describes the combination of structure and/or function for the entity. There can be multiple architectural bodies for an entity. This allows a designer to have any number of alternate views of the same object. In addition, an entity interface can

contain a specification that can be used to insure that each architectural body meets the requirements. To help the designer manage large designs, the language supports the concept of libraries from which a designer can select precompiled designs and reference them in his total design. Because engineers desire to view their designs as they are created, the emphasis in a HDL is human readability. VHDL takes this concept to the point of providing an "inverse" compiler or reverse analyzer so that the engineer can even view the precompiled design objects referenced in his design as source if he so desires. All of these features are to aid the designer to easily conceptualize and manage his hardware designs.

To summarize, attributes of a HDL that stand out are: they support the design phase of development; they interact directly with the human and must be human readable; modern HDLs such as VHDL have features that aid the designer in managing his design; they communicate design information from human to human; support the whole spectrum of design and test, from system through chip. This VHDL system feature also helps in the transmission of design data, including structural and functional design intent within the design community.

TRANSFER OF DESIGN DATA TO MANUFACTURING

The HDLs evolved within the design community. The evolution of EDIF stems from the need in the semiconductor manufacturing industry to have a stable standard to transfer electronic data resulting from circuit design to manufacturing environment. Historically, this type of information was geometric and needed in order to produce a mask for the manufacture of an integrated circuit. While there are no formal standards in this arena, over the past several years there have been two more or less defacto standards prevalent in the industry to accomplish this function.

The CAL-TECH Interchange Format (CIF) was created by and is most popular in the university community. Being university based, it is perhaps the least controlled of the formats in that there are many variations of CIF existing in the academic community and industry. The other popular format came from the GE-CALMA Company and is the Graphic Design System II (GDSII) exchange format. In discussing these formats, a careful distinction must be made between the data formats described above and the graphics languages that create the formats. There is a tendancy to mix the two and to use them interchangeably, although they are not the same thing. The CIF and GDSII formats are just that - formats.

For both CIF and GDSII there are accompanying graphics languages that are used to create the format. The fact that the CALMA Graphics Description Language (GDL) was proprietary made standardization difficult. However, since the GDSII format is specified, companies could convert any internal data format they used to GDSII format. A person does not have to use the CALMA GDL to create the data file for this type of data. It is not uncommon for companies to create the geometric data by use of multiple languages, and then to translate the data stored in a local database to the mask making format.

EDIF - A STANDARD TO CONVEY DESIGN DATA TO MANUFACTURING

As time has progressed, both CAD/CAE vendors and semiconductor manufacturers recognized that a stable standard was needed for conveying physical data from design to manufacturing. Thus, the EDIF effort was born. It was also long recognized that other types of data such as test, schematics, and printed circuit board layout information, was also desirable for the manufacturing process.

The EDIF working group is addressing a broad range of electronic data, and a brief discussion will be given on these. As with all previous types of formats, EDIF is intended to be machine readable. This format was created to make data easily readable and interpretable by computer. EDIF diverges from past formats in its structure. The previous formats were basically file formats. EDIF is structured around a format of a left parenthesis followed by a keyword followed by a sequence of "symbolic constants," "identifiers," "primitive data," or EDIF statements, followed by a right parenthesis. EDIF is thus more flexible than the previous file formats because it has no fixed structure other than that described. It has a "free form." Formats such as EDIF are still in the class of formats, they are not executable languages, but are based on a dictionary concept. The machine knows what to do because it knows the dictionary.

EDIF has its data organized into classes or "views" so that the data about a part can be placed into the appropriate bin for future access. EDIF currently defines the "views" of MASKLAYOUT, DOCUMENT, BEHAVIOR, SCHEMATIC, NETLIST, and STRANGER. The current version of EDIF can represent: Hierarchical Design Data; Schematic Symbol Libraries; Physical Design Libraries for Printed Circuit Boards (PCBs); Gate Arrays; Standard Cells; and Full Custom VLSI; Schematics and Netlists; Physical Cell Placement and Interconnect Routing for PCBs; Gate Arrays; Standard Cells; and Custom VLSI; VLSI Mask Layout; Simulator Stimulus and Response Data; Simulator Logic and Timing Models.

EDIF is not intended to be a design tool, but a bridge between the design system and manufacturing. EDIF is intended to compliment design tools by specifying a "neutral" format such that data created by design tools may be transported to the manufacturing environment.

IGES/PDES

The Initial Graphics Exchange Specification (IGES) grew out of the mechanical community. It is appropriate at this time to give some

perspective to what that community means by CAD. When mechanical designers speak of design they are referring to a mechanical drawing or set of drawings and specifications accompanying a set of drawings. CAD to the mechanical designer is a computer program that helps him do mechanical drawing on a graphics device instead of on a drafting table using paper. IGES is not the design tool itself, but is used in a way analogous to how EDIF is used in the electronic CAD arena. IGES is a data exchange specification to transport manufacturing data to the manufacturing environment.

The structure of IGES is a file format. The model for the data storage is a simple Entity Attribute database. Data is stored in the file referenced by entity type. Entities that have a dependency relation have a pointer in the data record that establishes the dependency relation between entities.

The basic file format concept relies on a dictionary. The entity is a predefined model that is referenced by name or number. Each data record associated with an entity is stored with the parametric data associated with the entity type. For example, a "point entity" is characterized by its assigned number, 116, and its xyz coordinates, as well as a pointer to a symbol to display the point. Most of the entities are geometric in nature, including both two and three dimensional shapes, annotation structures and fonts. There are also special entity types for various types of physical data about mechanical objects, eg: the modulus of elasticity of a steel beam. All data is stored in a record structure in the form: Entity Number, Parameter 1.....Parameter n.

There is also a MACRO language for defining new entity types. The language is a subset of FORTRAN. From a modeling standpoint, IGES makes FORTRAN function or subroutine calls. In that sense, IGES is slightly analogous to the fixed model logic simulators.

Because IGES is used for mechanical drawings, and there is conceptually little difference between lines and shapes on paper and a PC board or integrated circuit, the IGES Committee has defined electrical entity descriptions for making PCB and component layouts and schematics. In that respect both EDIF and IGES could be used as an exchange specification for that class of data.

The Product Design Exchance Specification (PDES) is an effort to define an information model that spans the spectrum of product information from mechanical through electrical. The electrical portion of the model is jointly sponsored by the IGES Electrical Applications Committee and the IEEE DASS. The goal of the effort is to define a standard conceptual model for electrical products applicable over the product lifecycle.

CONCLUSIONS

After reviewing the three standards, what conclusions can be drawn? In reality there is little conflict. VHDL is a design tool; EDIF and IGES are data exchange specifications. The typical relationship between design and the data exchange specification is that appropriate design information (eg: netlist information) is translated from the design data space to the desired format in the exchange space. In the final analysis, having stable standards in design and exchange data will greatly enhance the engineer's ability to produce reliable, maintainable systems.

REFERENCES

[1] VHDL Language Reference Manual Draft Standard 1076/A, 31 December 1986.

[2] EDIF Steering Committee Electronic Design Interchange Format Version 2.0.

[3] Initial Graphics Exchange Specification Version 3.0.

[4] IEEE Design and Test of Computers, Vol. 3, No. 2, April 1986.