HDML: Compiled VHDL in XML

Mohammad Hossein Reshadi, Bita Gorji-Ara, and *Zainalabedin Navabi

University of Tehran
Electrical and Computer Engineering Department
14399 Tehran, IRAN
Phone: 98-21-800-9215; Fax 98-21-877-8690
reshadi@cad.ece.ut.ac.ir, bita@cad.ece.ut.ac.ir

*Northeastern University
Electrical and Computer Engineering Department
Boston, Massachusetts 02115
Phone: 617-373-3034; 617-373-8970
navabi@ece.neu.edu

Abstract

Communication between designers and tools is a necessity and plays a significant role in productivity and time to market. XML, as an emerging standard, has proved to be a very suitable format for exchanging data on network and between applications. In addition, there are many public-domain tools for manipulating data in XML format regardless of what that data represents. HDML (Hardware Description Markup Language) is a new model for representing compiled data from original VHDL code in XML format. In addition to benefiting from XML advantages, HDML also develops a powerful mechanism for rule checking on a compiled design model. Although, the model does not depend on a specific data structure, we have used a revised version of AIRE/CE[1] for developing an analyzer for converting VHDL to HDML. To illustrate the capabilities of HDML, this paper discusses an example application of our XML based intermediate format, which is typical of many other potential applications. This specific application checks syntax and semantics of an input VHDL code for synthesis and we will refer to is as synthesizability rule checker. ***Key words:*** *VHDL, XML, hardware, CAD tool, system design, rule checking, DTD, XSL, AIRE/CE.*

1. Introduction

Facing the increasing need for more and more complex systems and to win quality and time to market competition, companies and designers demand faster and better CAD tools. This requires design partitioning and consequently converging designs that are done on different platforms by different designers using different tools.

Combining design components into a complete design requires people involved to communicate and exchange data and results. The critical part of this communication is a standard intermediate format in which design data are represented.

The efforts for designing an intermediate format for HDLs have a history as long as the HDLs themselves. AIRE/CE [1, 2] was the last partially successful one but its weaknesses in portability and documentation [3] prevented it from becoming a usable standard.

The first XML standard was released in 1998 and since then XML has been widely used in many different areas. Some of XML advantages that made it so public are:

- The standard is based on SGML, a very powerful and well-structured language.
- Although it is intended for machine readability and processing, its structure is also human readable and can easily be analyzed. This in fact means that the data represented in XML are also documented by this structure.
- XML files are text files and therefore it is quite portable and readable by different platforms.
- There are many public domain tools available for XML. Therefore it is not necessary to develop tools for reading well-formed XML files or for checking the validity based on DTD and making transformation by using XSL.

These and many other advantages of XML have made it an ideal basis for data storage and communication.

In contrast to XML, the VHDL grammar is very complex and hard to process. By compiling VHDL and

[1] Advanced Intermediate Representation with Extensibility/Common Environment

0-7695-0890-1/00 $10.00 © 2000 IEEE

storing the results in an XML file, one can easily process the design data without being concerned with language processing problems. The final goal in this way is to have a standard tagging with which the design can be represented. Before this can be done, we need to know what properties the structure should have.

HDML introduces a general model to present every object oriented data structure in XML format so that some important needs of hardware designers and CAD tool designers will be satisfied regardless of what data structure is used.

In this article, we first discuss the model and its advantages (Section 2). In Section 3, we discuss a sample data structure to represent VHDL in XML. In section 4, we explain a sample application of this model and in Section 5 we will address the limitations of the model and make suggestions for improvements.

2. General Model

2.1. Model Outline

Every object oriented data structure has four important properties: the hierarchy, names of classes, names of parameters in every class, and set of classes that can be instantiated. Three name spaces in HDML represent the last three properties and the hierarchy is implemented by the order in which these name spaces are used. The name spaces are:

- **obj**: concatenation of "obj" and the type name of the class that is instantiated represents an object in the memory and the class from which it is instantiated.

```
class A
{
  int iA, iID;
  Kind kK;
  A( Kind k) { kK = k; };
};
class B : public A
{
  char cB;
  B( Kind k) : A( k ) {};
};
class C : public B
{
  float fC;
  C ( Kind k ) : B ( C_Kind) { } ;
};
...
C x;
x . iA = 10; x . cB = 'a'; x . fC = 0.48;
```

Figure 1- sample C++ code

```
<obj:C-Kind>
   <cls:A>
        < p:iA type="int">10</p:A>
   </cls:A>
   <cls:B>
        < p:cB type="char">a</p:cB>
   </cls:B>
   <cls:C>
        < p:fC type="float">0.48</p:fC>
   </cls:C>
</obj:C-Kind>
```

Figure 2-HDML of Figure 1

- **cls**: "cls" appears before the actual names of classes. This tag encloses the parameter of a specified class.
- **p**: this shows the name of a parameter in a class.

An example based on example C++ code of Figure 1 illustrates these issues further. As shown in this figure there is a hierarchy of three classes and an object instantiates from class C. As shown in Figure 2 <obj:C-KIND> illustrates that an object of class C type is being presented. This tag contains three "cls" tags corresponding to the three classes in the hierarchy. Note that the name of last class (cls:C) is equivalent to the type name C-KIND. Each class tag contains a list of "p" tags corresponding to the class parameters. The "type" attribute of "p" tags shows the type of information that the parameter holds. It is clear that parameter's access type; i.e. "public", "protected" and "private", does not provide any useful information and is not saved in HDML.

2.2. Advantages of HDML

2.2.1. Extensibility. HDML is extensible from two points of view. First, it can represent any kind of object oriented data structure. Second, if a data structure is extended by adding some classes in the hierarchy, it can still be represented in HDML. The former implies that even if a standard data structure is used for other languages such as Verilog, HDML can still be useful.

2.2.2. Ease of applying rules. Consider that class D inherits from B (Figure 3). An object of this type contains information of classes A, B and D. In this way, if we are going to apply rules to all classes of type B, i.e. C and D, we can just apply that rule to <cls:B> and hence the rule will be applied to all objects containing this tag. This is in fact the concept of inheritance in rules.

2.2.3. Utilizing XML free tools. HDML is still an XML file and therefore every tool working on XML files can

also be used for HDML. Particularly XSL processor and DTD checkers can still be used widely and easily.

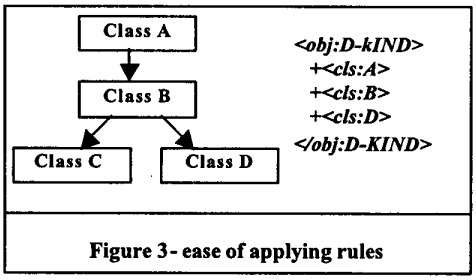

Figure 3- ease of applying rules

2.2.4. Extractable data structure. It is clear that data structure in Figure 1 can easily and automatically be extracted from information in Figure 2.

2.2.5. Compiled code. HXML [4] is another attempt for using XML for design representation, but it is merely a reformatting of VHDL source code. It requires some processing for example to understand which similarly named signals are referring to the same declaration. In addition, a large amount of type checking is necessary in HXML to realize which overloading function is actually used. On the other hand, HDML relies on the results of VHDL compilation that is also useful for extensions and algorithms. This means that despite knowing the relations, for example, one can add a security extension to the classes and store security information (like code/decode keys).

2.2.6. Representing complete designs. MoML [5] is another attempt in utilizing XML for representing designs. MoML describes a netlist of components and blocks independent of their function. On the other hand, HDML can represent all parts of an HDL. Therefore, not only the graph of components but also their functionality can be stored in the HDML model.

3. VHDL / XML structures and utilities

In this section we first describe basic concepts of utilizing XML and then will explain data structure used for HDML.

3.1. Xpath [7, 11]

XPath is a search language for addressing specific elements in an XML file. The XPath data model views a document as a tree of nodes. XPath leans heavily on familiar description of a document and uses genealogical taxonomy to describe the hierarchical makeup of an XML document. It refers to children, descendents, parents and ancestors. Of course the first ancestor is called root. Figure 4 shows a simple XML example and its corresponding tree.

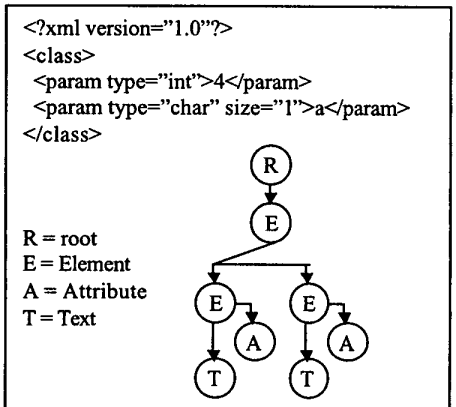

Figure 4- Familial Tree of an XML code

In XPath the fist "/" selects the root and stepping down the tree is done using element names. For example "*/class/param*" selects the *param* element whose parent is *class*. Symbols "@", "*", and "." select *attributes*, *all* and *current* element(s) respectively. A very important part of XPath is its predicates, which is useful in conditional selection of tree nodes. These predicates are XPath expressions enclosed in brackets ([ ]). For example */class/*[@type="int"]* selects every child of *class* that its *type* attribute's value is *int*. Also */class/param[2]* selects the second *param* of every *class* child of the root. Predicates are also cascadable, for example */class/param[2][@size]* selects second *param* children of any *class* that the selected *param* element has an attribute named *size*.

3.1.1. XSL (Extensible Style sheet Language) [8,11]

XSL is one of the very powerful tools that operates on XML and that uses XPath expressions for selecting elements and applying rules. It suffices here to show a very simple example in XSL showing the value of any *param* child whose parent name is *class* and whose *type* attribute is *char*.

```
<xsl:template match="/class/param[@type="char]>
  <xsl:value-of select = "."/>
</xsl:template>
```

We can use XSL and XPath to find structure of an XML file and produce messages or apply rules.

3.2. Programming APIs

Every XML structure must be processed using a program. The W3C (Word Wide Web Consortium) that supports XML and related standards has provided a standard API for XML programmers. This helps the programmers to use any implementation of this API for loading and saving XML files. This API is called DOM (Document Object Model) and provides a library of functions and classes to create and access XML data structures in the memory. Compared to XPath and XSL, it is more flexible but requires more programming efforts. Figure 5 shows how to select //param[2], i..e., every second *param* child in a document, using DOM.

```
nodes=doc->GetElementByTagName("param");
if (nodes->Item[1])
//do whatever
```

Figure 5- DOM finding 2nd param

3.3. DTD (Document Type Definition)

If an XML file follows all XML rules, DTD defines extra rules to limit the file to a very particular structure. A valid XML file is one that is well formed and follows the rules of a DTD. Here is a simple example:

```
<!ELEMENT class(param *)>
<!ELEMENT param (#PCDATA)>
<!ATTLIST param type ( char | int | float) #REQUIRED
                    size CDATA #IMPLIED>
```

In this example a *class* element can have every number of *param* elements. A *param* element can contain only text, must have a type attribute and may have *size* attribute. The type attribute can be *char, int* or *float* and the *size* attribute value may be anything. If a data structure is to be used for representing designs in HDML then a corresponding DTD can be designed to exactly show the structure. By changing such a DTD, one can expand or limit the structure.

3.4. Proposed data structure

The previous sections described basis upon which HDML is defined. The data structure used for storing compilation results will be discussed here. Our first candidate for this purpose was AIRE/CE, but it proved to suffer from certain deficiencies and weaknesses [3]. We have developed our revised version in which the problems are solved and some new properties are added. These changes are extensive and we will only highlight some of the key issues here. Our format is similar to AIRE/CE in which there are five layers of hierarchy and 11 categories of classes [1, 2].

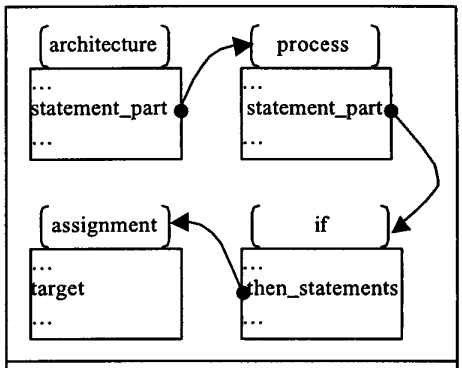

Figure 6- relation of objects in AIRE/CE

To use Xpath expressions we must know the relations in the data structure. Figure 6 shows a simple example of the relations in the data structure.

Objects pointed by others in the memory are contained by the representation of pointing object in the HDML. When an object is pointed by two other objects, two strategies are possible: one is to repeat the object in every pointing object's parameter scope in the HDML, the other is to use an ID for every object and use that ID to point to the object. The strategy used depends on the way objects are handled in the data structure but it does not affect the HDML properties.

4. A sample application

The examples given in this section are synthesis oriented and show how we can apply rules to HDML.

4.1. Time expression in waveforms

AIRE/CE stores waveforms in IIR_WaveformElement with two parameters: *value* and *time*. In the synthesis subset the time expression is ignored. Using XSL and DOM, we show how to find waveforms with a time expression.

Figure 7 shows the VHDL code of a simple assignment and its representation in HDML. It is not necessary in HDML to follow path to waveforms to check their time expression. It is enough to find every cls:WaveformElement tag and check it regardless of the class that contains it; i..e., conditional waveforms, sequential assignments, ...

Using an XSL processor, warning messages can be generated when the following expression returns a non-null value:

```
cls:WaveformElement[p:time[*]]
```

This Xpath expression will select all waveform elements that their p:time child has at least one child. Figure 8 shows the equivalent DOM pseudo code.

```
architecture t1 of test is
begin
  s <= '0' after 10ns;
end;                                        (a)
```

```
<obj:ARCHITECTURE>
...
<cls:ArchitectureDeclaration>
<p:statement_part>
...
<obj:SIGNAL_ASSIGNMENT>
...
<p:waveforms>
<cls:WaveformElement>
<p:value>...</p:value>
<p:time>...</p:time>
</cls:WaveformElement>
</p:waveforms>
...
</obj:SIGNAL_ASSIGNMENT>
</p:statement_part>
</cls:architecture_decl>
</obj:ARCHITECTURE>                         (b)
```

Figure 7- VHDL (a) and HDML (b) sample

```
nodes=doc->GetElementByTagName("cls:WaveformElement");
current = nodes->Item[0];
while (current != NULL){
  pTime = current-> GetElementByTagName("p:time");
  if (pTime->hasChildNodes())
    printf(" wanrning: waveform with time expression");
  current = nodes-> nextMatchingElemtAfter(current);
}
```

Figure 8- DOM code to find time expressions

4.2. Checking synthesizable subset

Synthesizable subset of VHDL differs from full VHDL both in syntax and in semantics. Syntactical exclusions, like *Block* and *guard*, can be checked via XSL and DTD. A DTD can be defined if the data structure is fixed in HDML. By removing some parts of DTD it will be possible to check syntactical exclusions.

In this paper we focus on XSL based methods and semantic checks. Almost all synthesizablity rules can be expressed with Xpath expressions. The complexity of these expressions depends on the rule and the data structure.

For example, consider the rule of clocks in process statements. A process is stored as in Figure 9. To be synthesizable, *If* statements that have a *'EVENT* in their condition must have no *else* part. Also *process* statements must contain only one wait statement [6].

```
<obj: PROCESS>
+<cls:Statement>
+<cls:ConcurrentStatement>
<cls:ProcessStatement>
<p:statement_part> ... </p:statement_part>
</cls:ProcessStatement>
</obj: PROCESS>
```

Figure 9- Process Statement

A process object that has more than one *wait* statement object in its *statement_part* is not synthesizable. Such a process is selected by the following Xpath expression:

```
obj: PROCESS_STATEMENT
[cls:ProcessStatement//obj:WAIT_STATEMENT[2]]
```

This expression finds all processes that their cls:ProcessStatement child has more than one obj:WAIT_STATEMENT in its successors. Figure 10 shows the equivalent DOM code.

```
nodes=doc->GetElementByTagName("obj:PROCESS");
current = nodes->Item[0];
while (current != NULL){
if (current->
GetElementByTagName("obj:WAIT_STATEMENT")-> Item[1]
)
  printf(" error: 2 wait in a process");
current = nodes-> nextMatchingElemtAfter(current);
}
```

Figure 10- DOM finding 2nd wait in process

Figure 11 shows the structure of an *if* statement. The following Xpath expression finds all if statements that have a 'EVENT in their condition part and also their else part is not empty.

```
obj: IF_STATEMENT [// p:condition //
obj:EVENT_ATTRIBUTE][//p:els e_sequence//p: element]
```

Figure 12 shows the equivalent DOM code.

```
<obj:IF_STATEMENT>
+<cls:Statement>
+<cls:SequentialStatement>
<cls:IfStatement>
<p:condition> ... </p:condition >
<p:else_sequence >
<obj:SEQUENTIAL_STATEMENT_LIST>
<p:element>... </p:element>
<p:element>... </p:element>
</obj:SEQUENTIAL_STATEMENT_LIST>
</obj:IF_STATEMENT>
```

Figure 11- If Statement

```
nodes=doc-> GetElementByTagName("obj:IF_STATEMENT");
current = nodes->Item[0];
while (current != NULL){
if (current-> GetElementByTagName("p:condition"))->
GetElementByTagName("obj:EVENT_ATTRIBUTE"))
if (current-> GetElementByTagName("p:else_sequence"))->
GetElementByTagName("p:element"))
  printf(" error: bad clocked if");
  current = nodes-> nextMatchingElemtAfter(current);
}
```

Figure 12- DOM finding clocked if with else part

5. Limitations and suggestions

Large file size and slow processing algorithms are two inherent weaknesses of text files, and hence XML files and special techniques must be used for solving them.

For example our experiences showed that removing tab and new line characters reduced the file size to less than a half of the original size. Also using short tags for objects, classes and parameter names can have similar affects. Standard techniques of compression are also applicable.

Processor execution speed is not only dependent on file size but also on the organization of the data in the file. This is more important when DOM solutions are considered and a large number of accesses are done to different parts of design.

6. Conclusion and future work

Communication between parties and tools is vital for today's design needs. XML, has proved its capabilities on the web and in the e-commerce. HDML is a representation for description of hardware in XML format. Using HDML strategy and tools and libraries like XSL processors and DOM, one can utilize the whole power of object oriented programming in hardware.

Our HDML definition does not include a data structure but uses one. We have used our fully implemented intermediate representation format for this purpose.

7. References

[1]- AIRE v.4.6 document (http://www.eda.org/aire/)

[2]- J.C. Willis, G.D. Peterson, S.L. Gregor, *"The advanced Intermediate Representation with Extensibility / Common Environment (AIRE/CE)"*, IEEE Transaction on Computer, 1998.

[3]- M.H. Reshadi, A.M.Gharehbaghi, Z.Navabi, *"Intermediate Format Standardization: Ambiguities, Deficiencies, Portability issues, Documentation and Improvements"*, HDLCon 2000, March 2000.

[4]- A. Zamfirescu, Z. Zhao, "HXML, A new approach to managing hardware information", VIUF, 1999.

[5]- E.A. Lee, S. Neuendorffer, "MoML, A Modeling Markup Language in XML-Version 4.0", ICCAD, March 14, 2000.

[6]- IEEE P1076.6/D1.12 Draft Standard for VHDL Register Transfer Level Synthesis, The Institute of Electrical and Electronic Engineers, New York, NY 10017, USA.

[7]- C.F. Goldfarb, P. Prescod, *The XML Handbook*, ch. 59, second edition, Prentice Hall, 2000.

[8]- C.F. Goldfarb, P. Prescod, *The XML Handbook*, ch. 60, second edition, Prentice Hall, 2000.

[9]- C.F. Goldfarb, P. Prescod, *The XML Handbook*, ch. 2, second edition, Prentice Hall, 2000.

[10]- C.F. Goldfarb, P. Prescod, *The XML Handbook*, ch. 54, second edition, Prentice Hall, 2000.

[11]- E.R. Harold, *XML Bible*, 1999, XSL(http://metalab.unc.edu/xml/books/bible/14.html), XPointer(http://metalab.unc.edu/xml/books/bible/17.html).

[12]- XSL Transformations (XSLT) version 1.0, (http://www.w3.org/TR/XSLT)