Portability and Security, All in CHIRE File System

Mohammad Hossein Reshadi, Bita Gorji-Ara, *Zainalabedin Navabi
Electrical and Computer Engineering Department
Faculty of Engineering / University of Tehran / Tehran, Iran
{reshadi, bita}@cad.ece.ut.ac.ir
*Northeastern University / Boston, MA 02115
Tel: 617-373-3034; Fax: 617-373-8970
navabi@ece.neu.edu

Abstract

Demand for faster, larger and more complex systems requires that designers partition their large designs, reuse available cores and cells and use advanced design tools. This way, design parties must be able to exchange their designs, vendors' intellectual property must be protected and tool developers should be able to focus on specific areas and integrate and combine other tools to make up a complete product. All these depend on how exchange of information takes place.

A good intermediate format can be a very good connection point for all these requirements. While technical properties of the intermediate format affect its efficiency, portability and security affect its success as an intermediate format. The former needs clear definition and the latter needs open areas in the standard for intuitions. CHIRE file system introduces a very modular architecture that covers all aspects of these two properties.

Key Words: intermediate format, VHDL, object oriented, portability, security, intellectual property, core and cell reuse.

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. AIRE/CE [1], **Error! Reference source not found.** was one such standard.

In [3] it is discussed that portability and security are two very important properties of standard intermediate files and that the requirements of these two issues are contradictory. Portability means implementation and platform independency while security means to control the access to some parts of the design. To support the former, the standard should clearly define every thing about files and their structures and to support the latter, there should be enough flexibility in the standard to provide means of access control for the developers and companies. On the other hand, different applications may need different file formats and the standard should not limit them to some specific formats.

By introducing CHIRE[1], as a revision to AIRE/CE, we have tried to solve almost all the discussed problems. A VDHL compiler uses this data structure and compiles language constructs into binary representations in the memory. These data are then stored in files so that other applications can use them. Here we will focus on portability and a completely new file system that not only considers portability requirements but also is very flexible for security aspects. This file system has two separate parts. One part is the general algorithm and strategy of loading and saving the class contents using fixed interfaces with some buffers. The second part is the implementation of interfaces in the buffers. The standard will define the algorithms and what each interface in the FR (File Representation) classes is supposed to do. Following the standard, new FR classes can support any file format and any coding and decoding algorithm. The implementation of FR classes is not a part of intermediate format itself and hence they can be used as plug-ins provided by companies and designers who sell or share their components.

The requirements and the definition of the algorithm provides portability and the open implementation keeps security issues hidden and unlimited. XML, Binary and Text are the three currently supported file formats that their structure, advantages and disadvantages are discussed.

[1] - **C**ompiled **H**DL **I**ntermediate **R**epresentation with **E**xtensibility

In Sections 2 and 3 portability and security issues are discussed respectively and in Section 4 the contradictory requirements are explained. In Section 5 the architecture of CHIRE file system is shown and Section 6 explains save and load algorithms. Section 7 briefly discusses the implemented file formats in CHIRE. Finally Section 8 discusses some possible secure applications of this file system.

2. Portability

A standard is supporting portability if files, generated under that standard, are always usable irrespective of the operating system and the machine by which the file is produced. This requires the platform and implementation independency features in the standard.

2.1. Platform Independency

Every machine has its own architecture and hence size and structure of standard types, such as integer, may differ on different machines. Although theorically good, the solution of AIRE/CE for this independency is not practically efficient because it adds a lot of overhead to all files.

In CHIRE, a complete design is stored in different files and every file points to a single text file known as the basic data file. This file contains the information about the structure of types that are used in the data files. This way, there is only one basic data file for every implementation of the standard on a machine and under its operating system. A typical basic data file contains the following information:

- Bit structure of integer and floating point numbers.
- The binary codes assigned to every KIND in the implementation.
- Types used in CHIRE and their description. For example the *MRT_Boolean* and codes assigned to *false* and *true*.
- Byte lengths of character data to show whether characters are Unicode or not.

Comparing the basic data file of the transferred files with the original basic data file of the intermediate format, the loader can decide what kinds of conversions are necessary.

2.2. Implementation Independency

The standard specification of an intermediate format should be such that files written by one implementation (vendor) can be used by any other implementation. For this, filing structure and addressing techniques should be clearly defined.

2.2.1. **Filing structure.** The filing structure specifies how a complete design is represented on hard disk, how files are configured and how data are stored in a file.

AIRE/CE suggested a single file per design strategy, which is extremely inefficient especially for large designs that use many different libraries [3].

In CHIRE, every library is saved in a separate file in which the names of primary and secondary units, their relative path to the file and the list of source codes compiled in the corresponding library are stored.

Every library unit is stored in a single file and the file has three parts:

- The basic data file name.
- The file version. This version is increased on some very special cases and can be used in detecting the validity of files at load time.
- The objects of the corresponding library unit. All objects accessible from a library init are considered its objects. Every object is stored in a distinct area and the mechanism is so that extensions can simply store their information in the same area of an object.
  File names of library unit files are generated automatically to ease the process of loading and saving. The file names are constructed by concatenation of type descriptor and full name of the library unit. The extension shows the file format.

2.2.2. **Addressing techniques.** When multiple files are used to represent a complete design, addressing techniques for referencing objects must be clearly defined.

In CHIRE, there are two kinds of object referencing. Local Object Referencing (LOR) points to other objects in the same file, while Global Object Referencing (GOR) points to objects in other files. *FIR_ProxyRef* and *FIR_ProxyIndicator* of the proposed FIR definition in AIRE/CE do not present a solution for GOR and are inefficient for LOR [3].

LOR addressing is simply implemented using unique IDs given to objects of a file. In a Library Unit, every object that points to another object uses its ID to maintain its connectivity. The saving mechanism promises the uniqueness of IDs in a single file. Using IDs instead of physical positions in the file is a necessity when supporting different file formats is intended. The sequence of IDs assigned to objects depends on the implementation considerations, but since these IDs are not used outside a file, it does not affect portability.

However, GOR is more complicated and is based on the fact that unique names extracted from source code are

always uniquely extractable in any implementation by just following an algorithm.

In GOR, objects in other files are addressed through a version and a sequence of names. The version confirms that the destination file is not compiled lately and that this part of the design is still valid. The sequence of names is needed to solve the problems of visibility and scope that is not available (and is not needed to be) after compilation. This very important property enables the designs to be ported easily even if they have many references to the standard packages. This sequence of names is called an explicit name and is somehow equivalent to the 'PATH attribute in the VHDL.

Using the explicit name, the loader can locate the filename where the object is stored and find the object in the located file.

3. Security

Portability is utilized well by designers and vendors only if their intellectual property is protected properly. In the other words, the standard should provide a means for controlling the access to some particular parts of the design. For example the standard should allow the vendors to produce and purchase coded data files that are decoded and loaded when specific conditions are satisfied. Even it should be possible to partially decode the file using any coding and decoding algorithm.

4. Contradictory goals

While portability is based on clarifying every thing about structure of files, organization of data inside files and load and save algorithms, security depends on hiding some of such information from users.

CHIRE tries to cover both topics by defining a standard and fixed mechanism and flow of data and leaving the file formats unlimited and open to the users. In this way a file can be used only if the proper file handler is available and the small handler performs reading and writing in any desired format.

5. Block diagram of CHIRE file system

As shown in Figure 5-1 the data transmission between classes in the memory and files on the hard disk is done through Object and File Buffers. In this design, every object in the memory stores/retrieves its data to/from an Object buffer by calling a fixed and standard interface. The Object buffers in turn, send/receive data to/from a File Buffer and finally, by calling operating system routins, the File buffer interacts with physical file on the hard disk using any coding and decoding algorithm and in any desired format.

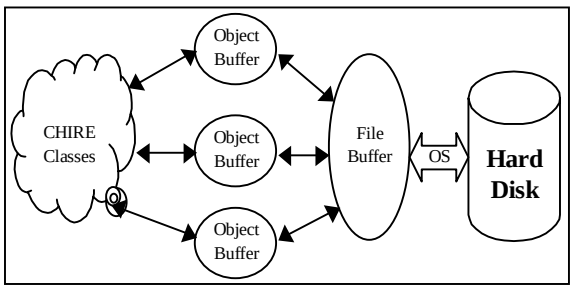

Figure 5-1- Levels of file access in CHIRE

5.1. Object Buffers

A single Object Buffer corresponds to a single object in the memory and has at least three parameters: kind of object, ID and explicit name of the object. All kinds of Object Buffers have a method for saving and loading a parameter whose name, value, size and extension name, to which the parameter belongs, is known. The order of calls to this method indicated the order in which fields of an object are stored in a distinct area in the file. Parameters of different extensions are stored separately so that a complete extension's information can be updated regardless of other information in the file.

ID			
KIND			
Explicit name			
Extension 1	Extension 2		Extension n
Data of extensiotn 1	Data of extensiotn 2		Data of extensiotn n

Figure 5-2- Logical Structure of Object Buffer

Figure 5-2 shows the logical structure of an Object Buffer on the file.

5.2. File Buffers

A File Buffer corresponds to a physical file and controls the format, the coding and decoding algorithm and the sequence in which Object Buffers are stored on the hard disk. File buffers do this by calling operating system routines.

6. Data flow in the system

In this section we explain how data flow from classes to the hard disk in save and vise versa in load.

6.1. Save algorithm

Save always starts from a *Library Declaration* by calling the save method for all library units in that library. Since every library unit is stored in a separate file, it creates a file buffer and passes it to all its objects. When all object buffers are dumped to the file buffer it

organizes the buffers and then dump them on the hard disk.

On the other hand, when save method is called for an object, it creates an object buffer and passes it to all layers of hierarchy so that parameters of every class in the hierarchy are stored in the object buffer. Finally when all parameters are written, the object buffer is dumped to the corresponding file buffer.

6.2. Load algorithm

A library unit is loaded accompanied with all its dependencies. In fact load also links the files used in a complete design. Since this linking occurs every time a file is loaded, the standard and common files are not transferred with the main files of a design.

However, VHDL relays heavily on its libraries and the use clauses. To increase the performance of loading and handling the objects, objects are fist virtually loaded and then when they are referenced, they are actually loaded.

6.2.1. **Virtual Load.** Most of the time, users use ALL keyword in use clauses. The process of loading the whole library is extremely slow in such cases.

In virtual loading technique, when a library is asked to load some library unit, it merely creates the corresponding object and sets a *not_loaded* flag. Later when a GOR addressing happens and the library unit of the referenced object is not loaded yet, actual loading is invoked.

6.2.2. **Actual Load.** Complete loading has two major steps. In the fist step a list of objects of the file is created and in the second step the relations are reconstructed.

In the first step, an item containing two data members is added to the list for every object area in the file. These items are shown in Figure 6-1. The buffer data member is filed with the information of the object area then an object is created based on the KIND variable of the buffer.

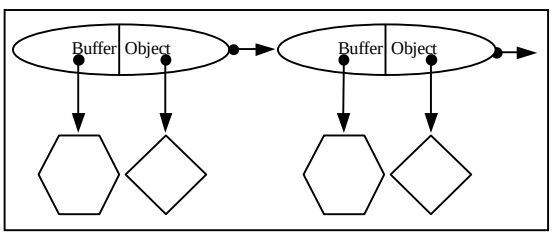

Figure 6-1- Objects & Buffers

Creating and filling the buffers (first step) are handled differently in file format related classes while the operations of the second step are unique for all file formats. In this step, processing the buffer contents restores object's parameters. LOR and GOR addresses are resolved using ID and explicit name respectively.

7. File formats

Supporting different file formats is not only necessary for security but also required for some applications. For example, text formats can be very useful in debugging and academic uses. Currently the following file formats are implemented in CHIRE.

7.1. Binary

A binary file in CHIRE has three major parts. The Header contains the version number and basic data file name. The Body includes the information of objects and the Trailer contains a check sum of the whole file. Binary object buffers are stored in separate areas in the Body. Every area starts with the size of the buffer, the unique ID of the object in the file, the kind of the object and the explicit name. Other parameters of the object are stored from this point on.

The length of strings and other dynamic length records of data is stored before the main data of that record. For example the string "TEXT" is stored as 4|T|E|X|T. The binary format is very machine and operating system dependent and relies heavily on the contents of basic data file. Figure 7-1 shows the structure of binary format.

This format is very compact and has very fast load and save routines. It is also very suitable for applying security algorithms. But this format is not as portable as text formats specially when transferred on different machines and by means of some transfer protocols on the network. On the hand it is not readable enough to be used in tracing and debugging of the tools developed in CHIRE. The data must be read in the same order they have been written.

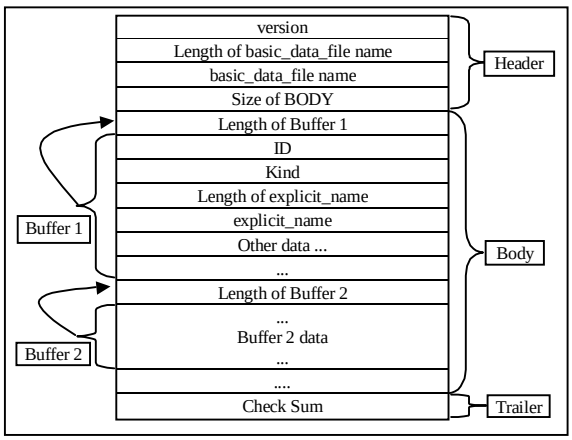

Figure 7-1- The binary file format structure

7.2. ini-text

The structure of this format is very similar to the ini files in Windows. There is again a header that includes the version number and basic data file name. For every object buffer there is a different section identified by the unique ID of the object. Parameters of the object are stored in a "*key=value*" format. "[]" symbol is used for LOR addresses and "{}" symbol is used for GOR addresses (**Error! Reference source not found.**).

Besides being very portable, this format is so readable that without having any compiler it is possible to write or modify such files. The other important advantage is that reading and writing orders can be different. Of course it is obvious that the size of this format is much more than binary format.

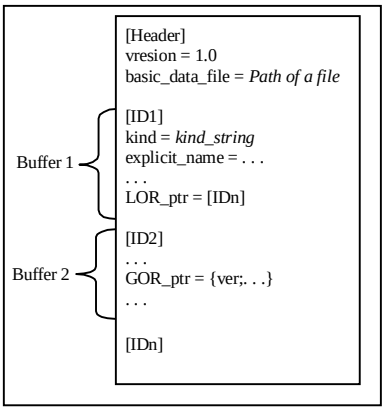

Figure 7-2- The ini-text file format structure

7.3. XML

XML is now a very well known format on the web and there are a plenty of free tools available for this format. Even browsers such as Microsoft Internet Explorer has some facilities to browse XML file nicely. To store the data of CHIRE in The XML format the HDML[2] was designed. This format is described in detail in [4]. In this format, data are stored in tags and attributes. The only disadvantage of this format is its large file size and slow read and write algorithms. But it is very suitable for rule checking and debugging.

8. Security aspects

File handlers in CHIRE are designed so that they do not need to be aware of the nature of data and hence they can be separate modules. This enables vendors and designers to use specific formats that need specific handlers that check for example a license and then decodes the file.

Handlers can also partially decode the data so that for example a behavioral design is sold to the user while structural information is also available (but coded) in it. In this way when the user returns back his/her design and the purchased code, structural information can be decoded.

On the other hand, in a multi file structure, replacing files (e.g. behavioral architecture with structural one or simulatable with synthesizable) is very easy. Consider how complex it would be if the purchased core was mixed with the user's design.

CHIRE file system defines all portability issues clearly and facilitates using unlimited formats and file handlers that can help developers and vendors distribute their compiled cells and libraries easily.

9. Conclusion

Security and portability are two major requirements for developers using a standard intermediate format. These two need contradictory properties, i.e. the former needs unlimited and not known and standard techniques while the latter needs every thing to be clearly defined.

Implementation and platform independency are two aspects of portability. The CHIRE file system covers all these requirements by introducing a very modular and reconfigurable design.

Currently three file formats are implemented in CHIRE. The Binary format is fast, compact and secure but is not very readable. The ini-text format is readable enough for debugging and tracing but its files are slower and larger than binary files. XML format in CHIRE is implemented based on HDML and is very suitable for XML based applications.

10. References

[1] AIRE document Version 4.6 at 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] M.H.Reshadi, B.Gorji-Ara, Z.Navabi, *"HDML: Compiled VHDL in XML"*, VIUF 2000, October 2000.

[2] - **H**ardware **D**escription **M**arkup **L**anguage