AIRE/CE : A Revision Towards CAD Tool Integration

Mohammad Hossein Reshadi, Amir Masoud Gharehbaghi, Zainalabedin Navabi
Electrical and Computer Engineering Department
Faculty of Engineering / University of Tehran / Tehran, Iran
{reshadi,amir}@cad.ece.ut.ac.ir, navabi@ece.neu.edu

Abstract

The growth in Micro Electronic industry demands more capabilities from CAD tools. This requires better-integrated environments and more design portability across platforms and tools. The move towards hardware description languages in recent years and interoperability puts a pressure on EDA vendors to come up with a well-defined standard intermediate format. The draft AIRE/CE intermediate format, as distributed publicly on the Web, is one such standard. Although this standard may be better defined and better documented than any other proposed standards for this purpose, it has shortcomings that must be resolved before it is adapted by the EDA vendors and designers. This paper discusses CAD tool integration problems and reviews the AIRE weaknesses in this area and also presents our solutions according to our experiences with the AIRE implementation[1].

Keywords: Object-Oriented Intermediate Format, AIRE/CE, IIR, FIR, HDL, CAD Tool Integration.

•• Introduction

Design process has been changed since the beginning of microelectronic industry. Automating the design process has been the main goal of these changes. The role of hardware description languages, such as VHDL and Verilog in design process has become more vital nowadays. However, manufacturers have developed many CAD tools in various design areas such as simulation, synthesis, test generation, and verification. Design of a real system requires several CAD tools from higher levels of abstraction to lower levels. Since it is very difficult for a developer to fulfill all the designer needs, many designers employ several tools from different manufactures in various stages of their design.

The main problem of designers to use different tools from different companies is feeding in data generated by a tool to another one. The tool developers also face this problem in interfacing with other tools from other vendors. In all the cases sharing the design information between various tools is the main problem of CAD tool integration.

Consider an environment in which several components exist that work with a standard intermediate format. For example a schematic editor, HDL analyzers, software compilers, a waveform editor, digital and analog simulators, synthesis tools, and verification tools share information through intermediate files. Each module works independently and is not worried about the implementation of others, because all of them communicate through a standard known intermediate format. In this way it is possible to build a complete environment in a short time. Tool developers can concentrate on what they are expert in without having to spend extra effort on other modules. It is also possible to have many similar modules, with different functionality, and configure them properly for each design. (Figure 1)

A good intermediate format plays a major role in bringing these ideas to life. In section 2 we discuss the specifications of a suitable intermediate format for CAD tool integration. In section 3 AIRE/CE will be introduced as a good candidate. In section 4 some shortcomings and deficiencies of AIRE/CE and our solutions for them will be proposed. In section 5 we have conclusion and also the future work.

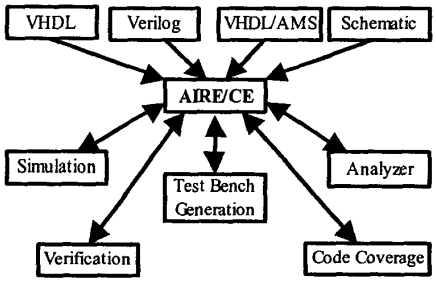

Figure 1. Usage of an Intermediate Format

[1] The version of AIRE/CE being discussed in this paper is the version that is publicly circulated on the Web [1], and several companies commercially use versions if AIRE/CE which may differ from the publicly available draft and which may address issues raised in this paper. AIRE is a trademark of FTL Systems.

2. Specification of a Suitable IF for Integration

An intermediate format must have some specifications to be suitable for CAD tool integration. In this section we discuss briefly about the most important needed specifications.

2.1. Completeness

Designers need to describe their design in various forms, from behavioral to transistor level. It is possible to partition the whole system into subsystems and employ an appropriate specification for each subsystem. It is also possible to purchase subsystems or cores from different vendors and integrate them to make up the system. Most real-world applications consist of analog and digital components. So the designer must be able to test the whole system in a uniform environment. Today, one of the choices of a designer is to specify and test the software of a system simultaneously with its hardware.

In all the above cases the design environment should be able to hold the design information from different levels of specification uniformly.

2.2. Portability

The main feature of a suitable IF for integration is portability. Portability can be defined in two areas: portability over platform, and portability over implementation. Compiled designs must be portable over hardware and/or operating systems. Changing hardware or even operating system of the same hardware can cause portability problems. In addition designs must be reusable over all the tools that comply with the IF standard for their interfaces. Different versions of a tool and also tools from different companies must accept a previously compiled design.

The representation of intermediate format in file and also memory is the main key for portability and also CAD tool integration.

2.3. Documentation

Documentation plays a major role in success of a standard IF. Systematic upgrade and review as well as regular updates are essential for documentation of a standard. A well-documented standard enables both developers and users to concentrate on their specific work without extra effort to make their tool compatible with other ones.

3. AIRE Introduction

Over the past decade, many efforts have been done to develop a suitable IF for hardware description languages. Researches from governmental, academic and EDA vendors have been working on developing an IF. The main effort for standardization of an IF was by the IEEE DASC working group, which was unsuccessful in reaching its goals. Other efforts led to the AIRE/CE (Advanced Intermediate Representation with Extensibility/Common Environment), representation which is an object-oriented IF designed to support VHDL, VHDL-AMS, Verilog, and other languages. Worked into AIRE is extensibility capability, which makes this standard adaptable to new applications. The memory representation (Internal Intermediate Representation, IIR) of AIRE has five layers of hierarchy. Its file counterpart (File Intermediate Representation, FIR) stores the compiled model in the FIR files.

3.1. AIRE/CE Structure

AIRE is an object-oriented intermediate format with five layers of hierarchy, which is shown in figure 2. In AIRE, the collection of objects represents a very generalized abstract syntax tree (AST), while methods associated with the classes (and thus objects) represent an integrated application programming interface (API). AIRE consists of two

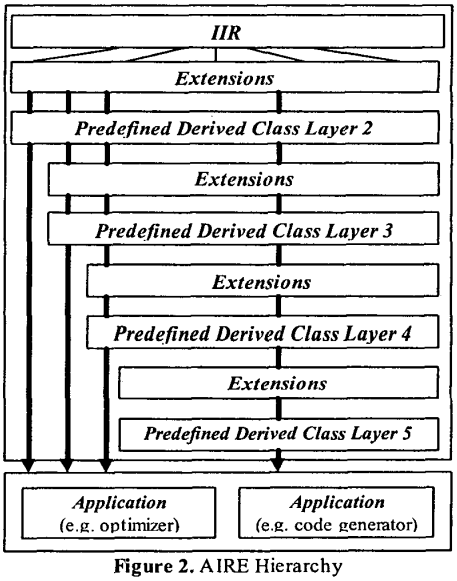

Figure 2. AIRE Hierarchy

parts:

- IIR is memory representation of design information
- FIR is the IIR data, saved in file

AIRE can be implemented by any object-oriented language, while its primary design is based on C++.

Generally speaking, AIRE classes can be divided into two categories:

- Classes which are in one-to-one correspondence with language constructs such as conditional statements, expressions, and signals.
- Classes which hold a collection of the above category elements, such as a list of statements in a block.

3.2. AIRE Extension Layers

As shown in figure 2, every predefined layer has an extension layer. AIRE users only employ predefined AIRE classes, whereas extra functionality can be added by extension classes. So all the AIRE users work with a uniform structure and interface. Each application adds the needed functionality to the AIRE structure by extension classes.

For example a simulator application adds simulation-specific methods to its extension classes, and all the AIRE classes inherit the simulation capability.

4. AIRE problems and solutions

Although AIRE is good enough to be considered as a major candidate for CAD tools integration, There are some obstacles that makes the standardization process difficult. In this section we address some of these weaknesses and our solutions to them.

4.1. Documentation

To ease the use of IF by different CAD tools, each of them should clearly know the exact structure of data, represented by the IF. So a good documentation on data structure and exact mapping from language (source) to IF (target) is a necessity. The current version of AIRE does not have such documentation. Throughout our implementation we gradually gathered the essential information and it would be soon available electronically.

4.2. Memory Management

If AIRE is to be used as the connection node between different applications, besides knowing its structure, it should be possible to consider it as a robust black box. To have a good memory management, and thus robustness, there should be a uniform mechanism to create and destroy objects. In AIRE, there are two kinds of objects: canonical objects that have only one instance in the memory and many references. A technique based on reference-count is used for them. On the other hand, there are non-canonical ordinary objects that are handled by operators like "new" and "delete". There are many examples in AIRE that some parameters can be of either type. Choosing the proper destruction technique by keeping track of creation method is impractical. We expanded the technique used for canonical objects (reference-count based) in a way that it covers all types of objects. In this way, every application can make any number of references to any object; without worrying about other application's references to that object.

4.3. Portability

Portability is in fact the most important property for an IF in order to be used for integration because different CAD tools interact through files. These files must be portable even between different implementations of the IF. In this way, the very first thing to know is the filing structure and how to save and find needed information.

4.3.1. Filing Structure

What should be stored in a single file, how objects in files address each other and how files are configured to present a complete design, are not mentioned in the current version of AIRE. We designed and developed a complete new FIR. This design is based on general concepts of HDLs. In the FIR, a single file includes only the objects of a single library unit (or module in Verilog). A library is a single file that includes the names of library units of that library.

4.3.2. Addressing Techniques

Object addressing in a single file is not as critical in portability as object addressing between files because the former is transferred with the files but the latter is not. In the next two sections, the two addressing techniques that are used in FIR files are presented.

4.3.2.1. LOR

Local Object Referencing (LOR), object addressing in a single file, is done using IDs in the file. Every object has an ID, which is unique in that file. Other objects of this file use this ID to refer to the object. (Figure 3)

```
[1]
kind = IR_ENTITY_DECLARATION
...
identifier = [2]
...

[2]
kind = IR_IDENTIFIER
...
text = "Mux2x1"
length = 6
```

Figure 3. LOR Example

4.3.2.2. GOR

Every design uses other libraries and packages (like the standard package) and hence, there are objects referring other objects in other files. The transfered file must not contain the information of referred file. For example, standard package must not be transferred with other files.

Full names of objects, equivalent to PATH in VHDL, are the basis of the addressing technique, developed for this purpose. Every object referring to another one uses the full name of that object and its file version to refer to it. This is called Global Object Referencing (GOR). For Example consider a desig referring to type "bit" in standard package. As it is using the name, it can use any standard package in which this name exists. Figure 4 shows another example. An entity declararion hasa pointer to its last analyzed architecture. The full name shows that this architecture is in the "work" library and its name is "behavioral". This GOR also shows that if the version of FIR file of the architecture is more than "1", the current file is invalid and should be compiled again.

```
[1]
kind = IR_ENTITY_DECLARATION
last_analyzed_architecture = {1;[work.behavioral(test)]}
...
```

Figure 4. GOR Example

4.3.3. File Formats

Currently the new version of FIR supports two file formats: binary formtat and text format. Text format is usefull for debuging and academic usages, while binary format is more compact and secure.

In this design, Classes in the AIRE have a fixed interface with physical file handling objects. Different implementaions of these objects provid more supported file formats.

5. Conclusion and Future Work

In spite of its shortcomings, AIRE is an intermediate format with a detailed document that is publicly available. There is a limited informal support for this IF, which in many ways is better than that of other intermediate formats. The object-oriented design of AIRE is a significant feature of this IF that is compatible with modern programming styles. A standard intermediate format is essential for today's design environments. AIRE has come a long way towards achieving this goal and we tried to make some more steps toward it.

More new file formats can be added to the support set of AIRE. Security issues and intellectual property are also left open in this IF. These areas are perhaps the last challenging steps to finalize the standardization process.

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. VIFASG 1076 VHDL Procedural Interface and Schema Definition. 1990, Draft version developed by VIFASG.
4. Implementor' s Guide for LEDA VHDL System. 1993, Available only from Leda S.A. Meylan, France.
5. M. H. Reshadi, A. M. Gharehbaghi, Z. Navabi, *"Intermediate Format Standardization: Ambiguities, Deficiencies, Portability Issues, Documentation and Improvements"*, HDLCON, March 2000.
6. M. H. Reshadi, A. M. Gharehbaghi, *"VHDL Intermediate Format Representation"*, CAD Lab. Report 23, University of Tehran, August 1999.[2]
7. S. GhiasiHafezi, M. H. Reshadi, *"FIR: Structure and Implementation Report"*, CAD Lab. Report 27, University of Tehran, November 1999.[3]

[2] Documentation will be made available electronically.

[3] The same as 2.