A Front-End VHDL Editor for Synthesis tools.

T. Bouguerba, J. Benzakki, M. Israël
LaMI
Université d'Évry-Val-d'Essonne,
Bd des coquibus
F-91025 Évry cedex
FRANCE
{bouguerb,judith,israel} @univ-evry.fr

L. Rideau
CROAP
INRIA-SOPHIA,
2004 route des Lucioles B.P 93
F-06902 SOPHIA ANTIPOLIS cedex
FRANCE
lrg@sophia.inria.fr

Keywords: ***VHDL*** grammar, Framework, Interchange Formats, Multiple CAD tools integration.

Abstract

This paper describes a new tool box for editing ***VHDL*** and generating ***AST***. It is integrated in the ***ASAR*** multi-formalism framework oriented towards Architectural Synthesis and generates a common intermediate data-flow format GC, for the different formalisms and tools available in that framework.

1 Introduction

High Level Synthesis users need to describe design views for the capture, optimization and verification of different design aspects. These languages may be graphical, tabular, or textual. Nowadays, we have a standard description language, VHDL, which is widely used in the world of CAD tools. In ***OSYS*** [4] and other synthesis tools, the designer can describe its architecture writing VHDL files if one is an expert; but also using what we called ***SDEV*** (**S**yntax **D**riven **E**ditor for **V**HDL) which is a "guided editor" to write VHDL without knowledge of the syntax.

This paper focuses on the different uses of the Syntactic Editor SDEV :

1. The ability of writing VHDL without knowledge of the syntax,
2. The opportunity to choose between different subsets of VHDL focusing on RTL Synthesis tools (Synopsys, Compass, ...),
3. The integration of such a tool in a Framework of Architectural Synthesis Tools : ***ASAR***,
4. The generation of the **AST** (Abstract Syntax Tree) of VHDL for Front-End tool developments.

2 The Syntax Driven Editor for VHDL : SDEV

- The ***VHDL***'92 [1] grammar as defined by IEEE has been included in ***CENTAUR***[5]. For that purpose, we have described in Centaur, VHDL concrete and abstract syntaxes using the METAL (for syntax) and the PPML (for the pretty printing) formalisms. According to the grammar and pretty-print specification, Centaur produces a tree generator and decompiler allowing the manipulation of VHDL files.

- The designer can describe its architecture writing VHDL files if one is an expert; but also using what we called SDEV which is a "guided editor" to write VHDL without any knowledge of the syntax.

  The edition is guided by the syntax. As the grammar we have developed is full VHDL, SDEV is a full VHDL editor. SDEV proposes the different statements available in VHDL, and the designer just has to click to build his description.This editor reduces programming errors and increases editing speed. Several menus are available for common tasks such as template editing and program transformations. The designer may toggle between "beginner" and "expert" modes, where menus present more or less detailed VHDL entries respectively.

- The designer can write full VHDL or synthetizable VHDL subsets according to the syntax and to specific synthesis tools. He disposes of pull-down menus where he can choose the level of description (Behavioral, Data flow, Structural) or VHDL synthetizable subset, Figure 1. Today the available synthesis tools work on different VHDL subsets and the difficulty is to write VHDL according to the target subset. SDEV gives opportunity to choose between different subsets, for High Level Synthesis or for RTL synthesis tools (Compass, Synopsys or View Logic...).

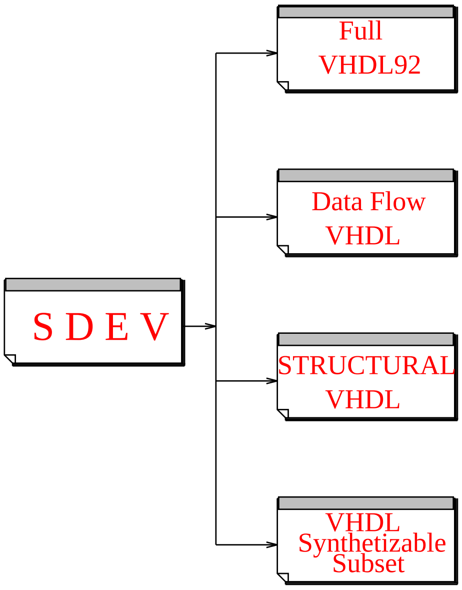

Fig. 1: *SDEV*

- SDEV seems to be a good support for teaching and learning VHDL for new CAD designers. In the "beginner" mode, the new VHDL user learns simple constructs of the syntax, and progressively turns on the "expert" mode to check what he just has learned. If he is an "expert", using SDEV, gives about 40% time gains in VHDL code writing.

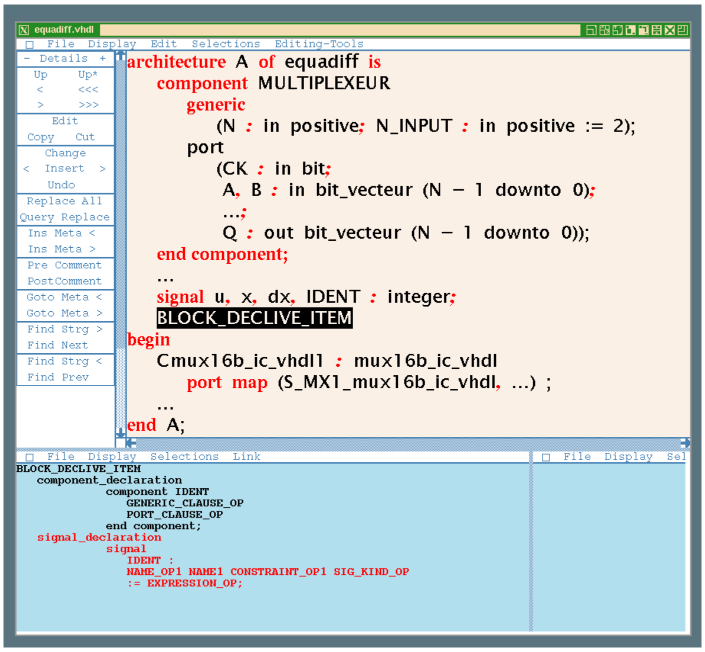

Fig. 2: *SDEV in ASAR : The example of the Differential Equation (Structural VHDL Subset)*

- In Figure 2, the **structural** VHDL mode is displayed ; we have already chosen an Architecture with it's declarative part and concurrent statement. The designer clicks on "BLOCK_DECLIVE_ITEM", SDEV proposes different constructs (COMPONENT, SIGNAL) according to the subset chosen.
- The most important application of SDEV is :
  - The generation of the ***AST*** :
    This structure, illustrated Figure 3, gives the opportunity to develop tools for VHDL transformations : VHDL optimizations, Architectural Synthesis tools developments, VHDL compilations, etc. The VHDL Abstract Syntax Tree can be saved in an ASCII file, from which we can build tools for target applications, like we have done in OSYS [4, 6].
  - The customization VHDL subset :
    The designer has the possibility of including his new VHDL subset in SDEV. For example from the full VHDL grammar, he reduces the grammar according to the target subset. And SDEV automatically, produces an editor for this new subset.

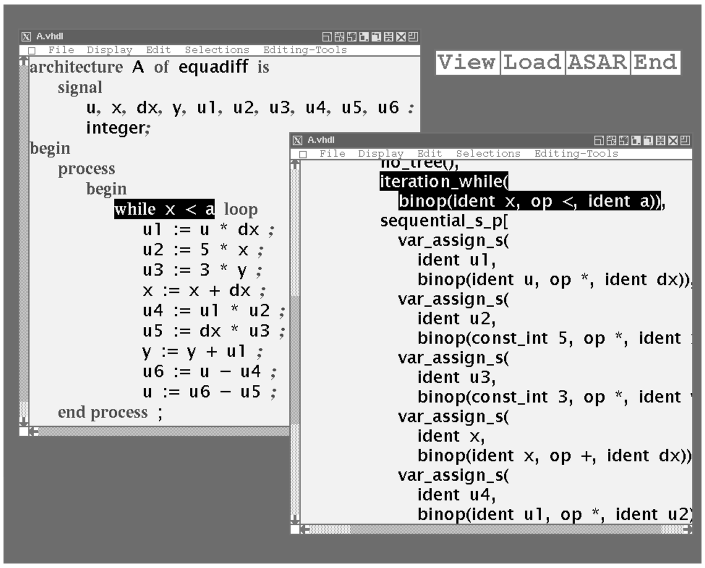

Fig. 3: *AST in SDEV*

Differences between SDEV and GnuEmacs

Structured editing is based on an abstract syntax specification. A structured editor displays an abstract syntax tree and allows it to be modified interactively. Structured editing ensures syntactic correctness whereas unstructured editing allows more freedom. Just as unstructured editors such as GnuEmacs, structured editors allow you to define special modes for various file types (lisp mode, TEX mode, etc.); moreover they offer the menu mode technique that allows the customization of a language environment through menus that reflect common tasks.

Let's give here some advantages of structured edition:

- Structured editing ensures syntactic correctness; only complete syntactic elements can be removed from or added to a program.
- It allows the use of colors and fonts according to syntactic or semantics properties, even on incomplete programs.
- The mouse clicking in a SDEV window is constrained by the underlying abstract structure: for example, when you click on the *if* token of an *if* statement, you select the entire *if* node; this is very usefull in particular in the case of fitted in *if* nodes.
- A print level mechanism allows you to control the detail of a program displayed in a structured editor according to the depth of a node in the abstract syntax tree: a low level of detail is used to emphasize global structure of a program.

– The menu mode consists of a number of graphic tools that together facilitate structural development and program transformations. Menu mode enables the rapid construction of a program by selecting and combining templates. A template is a fragment of abstract syntax interspersed with metavariables. The current selection in the main program establishes a context to which menus react, displaying the appropriate templates for that context. Contexts are usually syntactic, but may also be defined according to semantic concerns. The primary application of menu mode is the guided editing mode. In this mode, clicking on an applicable template replaces the currently selected fragment with the menu entry. This accelerates editing and reveals essential aspects of the language.

– Menu entries are generated from abstract syntax specifications, but these may be altered and extended by the user, and saved for future use: in particular it can be used to define sub-sets of a given language, and guide the edition following the corresponding restrictions.

3 SDEV in the Framework ASAR

The overall structure of the ASAR framework is sketched in Figure 4. The ASAR framework provides a set of languages and tools oriented towards particular target architectures or domains.

A more complete presentation of these tools can be found in [2, 3] .

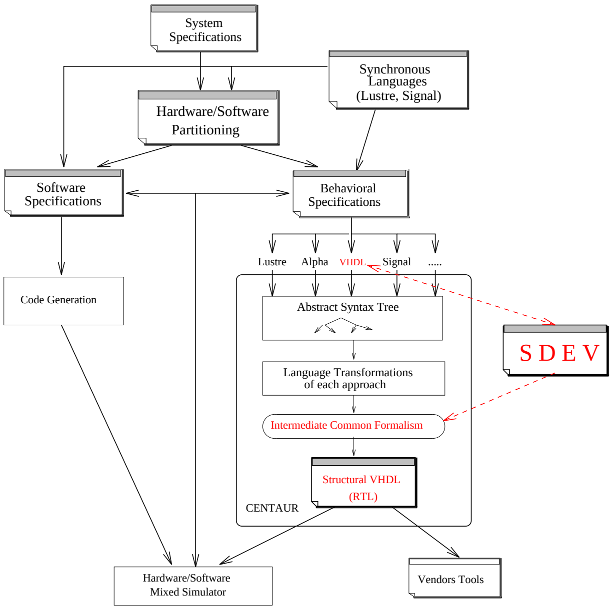

Fig. 4: *ASAR Framework*

Three components of ASAR are of main importance:

Centaur The Centaur generic environment is the basis on which we build this common framework, allowing to manipulate different syntaxes (including abstract, concrete syntaxes) for different languages and user interfaces with menu-guided edition in an uniform way.

An Internal Common Formalism (ICF) In order to federate our different approaches based on distinct user-formalisms, a sound semantic basis is needed, into which each formalism can be expressed. In other words, there is an underlying hidden common semantic to our multiple user-formalisms, to be conveyed by this ICF. This common formalism, could serve as an interchange format between different tools.

As our different approaches are all synchronous, a synchronous data-flow semantic seems to be the most appropriate. It is the reason why the *GC* formalism is under investigation for the ICF. GC[7] is a common format for synchronous data-flow languages, developed in the french *projet SYNCHRONE*.

SDEV in ASAR The SDEV editor for VHDL is integrated in ASAR system. The designer uses SDEV for writing VHDL code. Then he can choose between different approches of synthesis through ASAR by generating the ICF common formalism GC. In Figure 5, the ICF common formalism GC for Elliptic filter is illustrated.

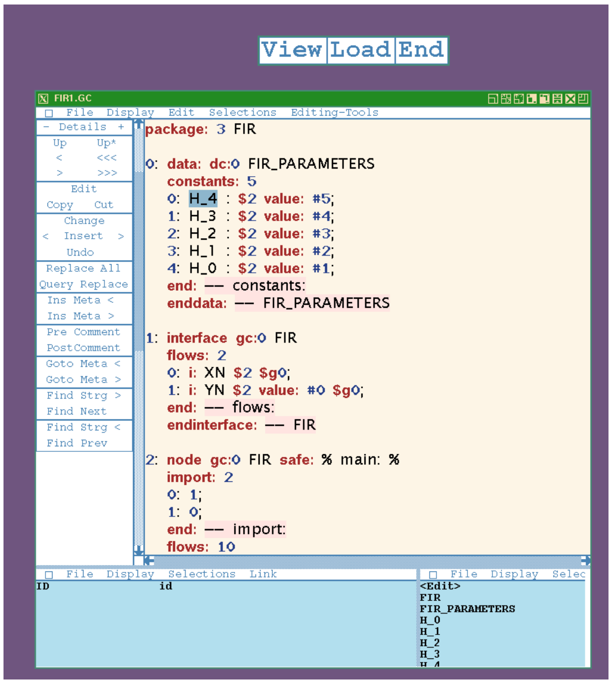

Fig. 5: *GC Editor in ASAR on the example of the Elliptic Filter*

4 Conclusion

The two first objectives have been obtained with the implementation of the VHDL editor (SDEV), including VHDL subsets for synthesis applications.

For the third point, the data-flow graph format GC, with possible evolutions, seems well suited to play the important role of inter-communication between formalisms in our framework. Moreover, it will be used as a link to external tools such as generators of distributed code, formal verification tools, etc. It has now to be integrated as a possible input or output format in each one of the tools available in the ASAR project.

The last point, generation of AST, is already available through SDEV for Front-End VHDL developments.

References

[1] Roland Airiau, Jean-Michel Bergé, Vincent Olive : *Circuit Synthesis with VHDL* Kluwer Academic Publishers 1994.

[2] P. ASAR. *Towards a Multi-formalism Framework for Architectural Synthesis : the ASAR Project* Codes/Cashes September 94, Grenoble, France.

[3] P. ASAR. *Framework and Multi-Formalism: the ASAR Project* EDAF'94 November 94 BRASIL.

[4] Judith Benzakki : *Outil de Synthèse et de spécification* Phd thesis, Paris 1993.

[5] P.Borras, D. Clement, T. Despeyroux, J. Incerpi, G. Kahn, B. Lang, V. Pascual, *Centaur the system*, Proceedings of the Third ACM SIGSOFT Symposium on Software Development Environments, Boston, Dec. 1988.

[6] M. Israël and J. Benzakki. *Osys: synthesis tool and hardware/software codesign.* IFIP Codes/cashes93, IFIP, Austria, May 1993.

[7] J.-P. Paris, G. Berry, F. Mignard, Ph. Couronné, P. Caspi, N. Halbwachs, Y. Sorel, A. Benveniste, Th. Gautier, P. Le Guernic, F. Dupont, and C. Le Maire. *Projet SYNCHRONE – Les formats communs des langages synchrones.* Technical Report 157, INRIA, June 1993.