VHDL subsets in the SDEV environment : A Case Study : The Synopsys subset

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

May 15, 1995

Abstract

This paper presents two new tools integrated in the Syntax Driven Editor for VHDL: SDEV[1]. The first tool allows the user to build his own VHDL subset. The second is a tool which allows to verify if a VHDL source belongs to a given VHDL subset. This paper is illustrated through the example of the subset used for the SYNOPSYS synthesis tool.

1 Introduction

VHDL is today the most widespread hardware description language supported by CAD tool vendors, although some implementations do not yet fully comply with its standard definition. This difference is bigger in the synthesis domain [3], each synthesis tool imposes its own synthesis methodology on the user. This fact has many disadvantages which are more relevant in a context, like synthesis, which today represents one of the most important applications of the language with a high user demand.

This paper focuses on two new tools included in SDEV. But before presenting these two points, remind us what is SDEV:

1. The ability of writing VHDL with a weak knowledge of the syntax,
2. The opportunity to choose between different subsets of VHDL, focusing on RTL Synthesis tools (Synopsys, Compass, ...),
3. The generation of the **AST** (Abstract Syntax Tree) of VHDL for Front-End tool developments,
4. The integration of such a tool in a Framework of Architectural Synthesis Tools: ASAR[4, 5],

2 VHDL subsets tailored by the designer

From the VHDL grammar included in SDEV, the user can build his own subset VHDL. This possibility is offered by the tool called TransForm, included in CENTAUR [6]. The main characteristics are the ability for the user to delete constructs, or to add new ones. Moreover, the user can use different subsets in a same session of development (Figure 1).

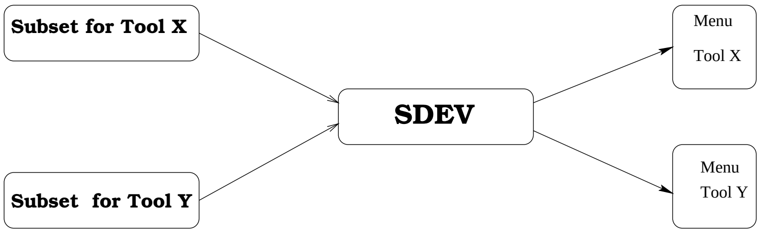


Fig. 1: *The editors generated by the user with* SDEV

The SDEV environment enables the user to generate a skeleton of VHDL program from the VHDL abstract syntax specification, according to a TransForm specification. A TransForm program is a set of rules describing transformations to be performed on a fragment of abstract syntax tree (a VHDL AST in our case). A rule constains *a rule_name*, a *typed_pattern* and a *transformation* where:

- *rule_name* is the name of the rule (default : the concerned operator name)
- *typed_pattern* consists of a pattern (default: a single unconstrained metavariable named **any* matching any tree) and of a phyla constraint [1] restricting the rule application (the rule can't be applied only if the current tree matches the pattern and its operator belongs to the named phyla)
- *transformation* is the pattern of a subtree that will replace the current when the rule is applied (by default transform reflects the operator's signature with metavariables [2] for each descendant of the operator)

Each rule may be specialized interactively by changing the pattern or transformation as desired: the designer may wish to add several rules for a given operator with different parts of the pattern more or less instantiated.

For example, let's consider the possibility for the user to build the VHDL editor for SYNOPSYS (Figure 2).

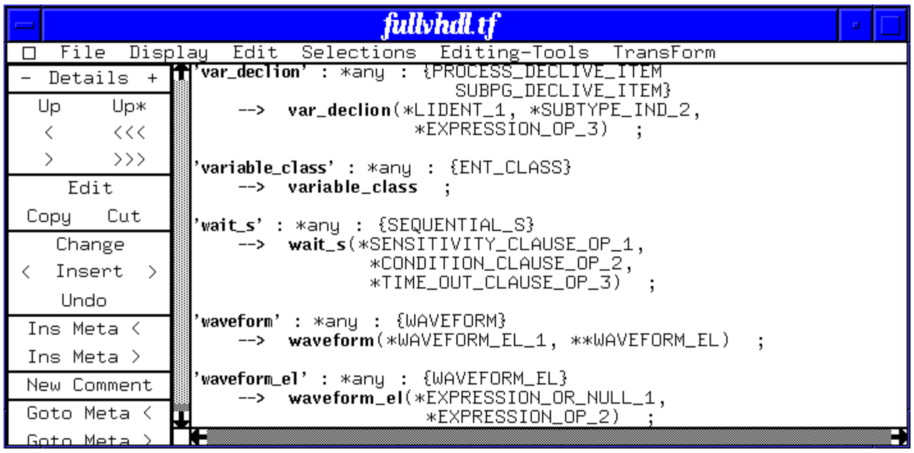


Fig. 2: *The VHDL skeleton generated by* SDEV

In the SYNOPSYS compiler, the *wait statement* can take three forms (Figure 3):

1. *wait until clock = value,*

[1] A phylum constraint means "any operator belonging to the phylum".

[2] a metavariable corresponds to an hole in an incomplete abstract syntax tree

2. *wait until clock'event and clock = value,*

3. *wait until not(clock'stable) and clock = value.*

where *clock* and *value* are identifiers. Note that in full VHDL, a *value* can be any expression.

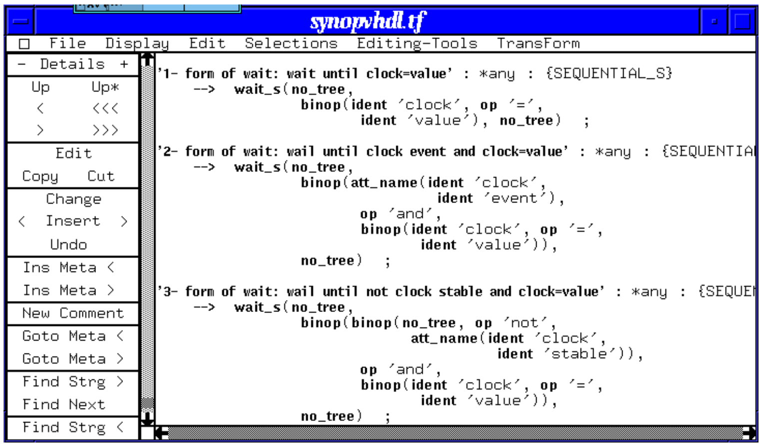


Fig. 3: *VHDL Subset for* SYNOPSYS*: example of the wait statement*

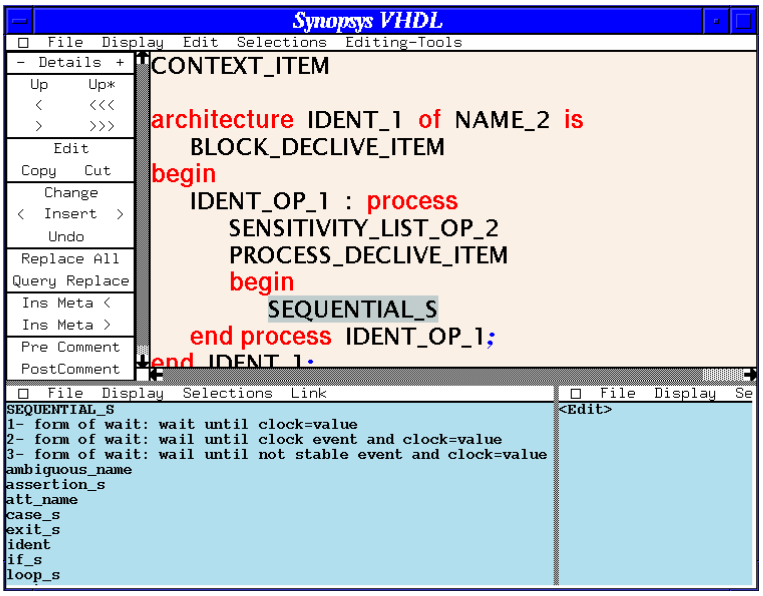


Fig. 4: *SDEV for* SYNOPSYS

By compiling this set of TransForm rules see (Figure 3), the designer creates a new editor for editing SYNOPSYS subset programs illustrated in Figure 4. Then, with the menu illustrated in the lower part of the figure 4, the end-user clicks on a pattern and instantiates the editor current subtree with the corresponding transformation subtree.

3 Pattern Matching

Syntactical differences between tools are closely related with the different VHDL subsets and packages they support. The problem is non trivial due to the fact that a subset always be necessary for synthesis. SDEV allows to parse any VHDL program and checks if this VHDL source belong to a subset or not. The methodology used here is to use *Pattern Matching* recognition on Abstract Syntax Trees (AST). When a VHDL source is loaded, it is represented by its AST. The SYNOPSYS editor performs a tree traversal on this AST and tries to *match* the encountered subtrees with the subtrees allowed by the Abstract Syntax Specification of the SYNOPSYS subset. We illustrate this with the example of a *wait statement* which is one of the particularity of the SYNOPSYS subset.

Consider a VHDL source where appears :

```
wait until clock = value * 2;
```

The corresponding AST is :

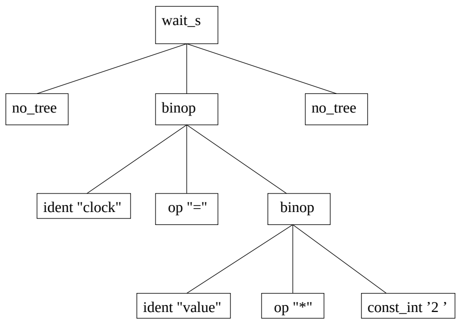
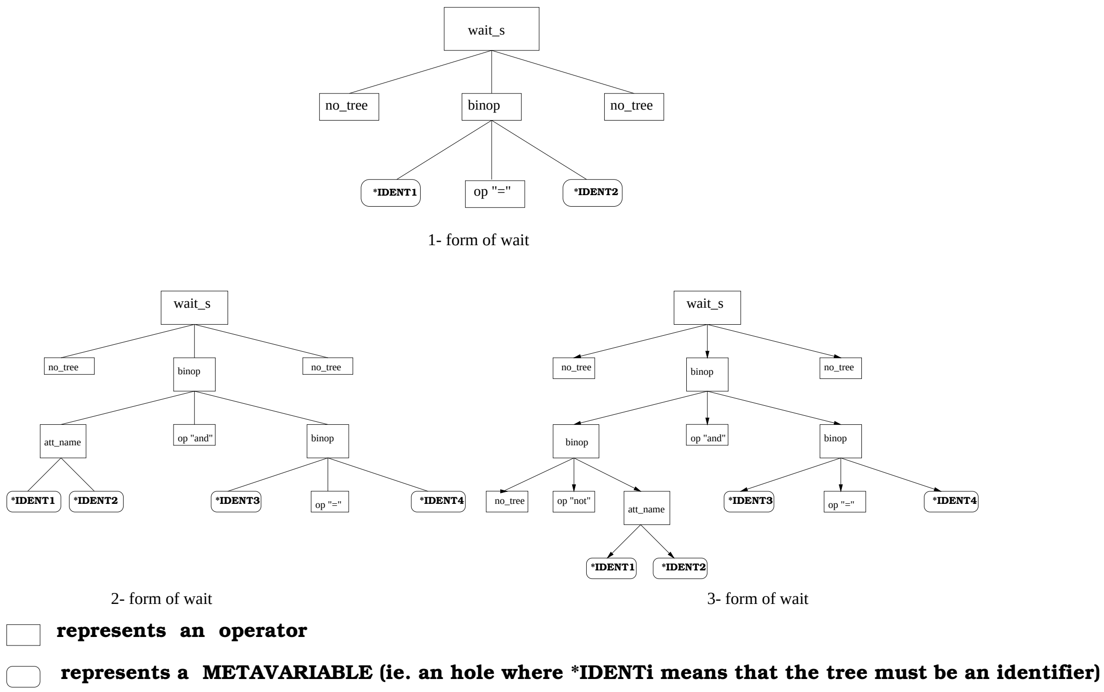


Fig. 5: *AST for wait until clock = value*2*

The editor compares this AST to the three forms of *wait statement* authorized by SYNOPSYS:



In our example of *wait statment* the typechecker generates a mistake :

1. In the form 1, the subtree “binop(ident “value”,op “*”,const_int ’2’) of the Figure 5 can’t be match with the metavariable *IDENT2 of the Figure 6.
2. In the form 2, the subtree “binop(ident “clock”, op “=”, binop(ident “value”,op “*”,const_int ’2’)) of Figure 5 can’t be match with binop(att_name(*IDENT1,*IDENT2),op “and”, binop(*IDENT3, op “=”,*IDENT4)) of the Figure 6.
3. In the form 3, the subtree “binop(ident “clock”, op “=”, binop(ident “value”,op “*”,const_int ’2’)) of Figure 5 can’t be match with binop(binop(no_tree, op “not”, att_name(*IDENT1, *IDENT2)), op “and”, binop(*IDENT3,op “=”,*IDENT4)) of the Figure 6.

When the user clicks in the Error window, the corresponding error appears highlighted in the VHDL editor. Then he clicks on the bad construction in source code, and SDEV proposes the good possible constructions.

4 Conclusion

In this paper, we have presented two new applications, included in the Syntax Driven Editor for VHDL: SDEV. Theses tools give to the user the opportunity to build his own subsets : for SYNOPSYS, COMPASS, or other synthesis tools. Moreover, the user can switch between different editors, for a target VHDL in a same session. These tools provide a unified solution to write VHDL oriented synthesis tools, SDEV can be good interface between different tools using different VHDL subsets.

References

[1] T.Bouguerba, J.Benzakki,M.Israël, L.Rideau *A front-end VHDL Editor for Synthesis tools* VIUF’95, San Diego, USA, AVRIL 1995.

[2] *VHDL Synopsys compiler Reference, V3.1*

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

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

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

[6] 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.

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