DEVELOPMENT OF A SEMANTIC AND SYNTACTIC ERROR DETECTION ENGINE FOR THE VHDL LANGUAGE: A DIDACTIC TOOL FOR FPGA DESIGN EDUCATION

Ricardo Francisco Martinez-Gonzalez and Fernando Huerta Mota

Department of Electrics and Electronics, Tecnologico Nacional de Mexico – IT de Veracruz, Veracruz, Mexico

ABSTRACT

This paper presents the development and validation of the VHDL Syntax Checker (VSC), a lightweight static analysis tool designed to accelerate the debugging cycle and enhance formative feedback in academic environments. Unlike industrial synthesis tools such as Intel Quartus or Xilinx Vivado, which require long execution times, the VSC performs lexical, syntactic, and semantic analysis in milliseconds. The tool integrates a rule-based engine with 10 verification rules (RULE-1 to RULE-10) and 15 automatic correction functions (FIX). A test vehicle with 10 induced errors was used to validate the detector. Results show high accuracy for punctuation and semantic errors, but reveal a cascading effect when structural keywords like "is" are missing, as well as a false negative in identifier validation. Beyond its technical performance, the paper discusses the pedagogical implications of the cascade effect as a teachable moment about VHDL hierarchy. The VSC proves to be a viable didactic platform that reduces student frustration, reinforces correct syntax through immediate feedback, and prepares future engineers for professional design workflows.

KEYWORDS

VHDL, static analysis, syntax checking, semantic error detection, FPGA education, innovative educational practices.

1. INTRODUCTION

In electronics engineering education, the use of Hardware Description Languages (HDLs) is a necessary skill for digital system design. Among these languages, VHDL (VHSIC Hardware Description Language) stands out for its strict syntax and strong typing, defined in the IEEE 1076 standard [1]. This rigor guarantees design integrity but also imposes a steep learning curve on students, who often face the complexity of the grammar before understanding the underlying hardware [2]. From a pedagogical perspective, this creates a barrier where syntactic details overshadow structural and logical concepts.

The FPGA design flow follows a linear cycle: coding, analysis, synthesis, and implementation [3]. The problem is that industrial tools such as Intel Quartus or Xilinx Vivado have long execution times due to the complexity of synthesis algorithms. The debugging cycle becomes a bottleneck when the designer must wait for complete compilations only to discover simple errors, such as a missing semicolon or an undeclared signal [4]. In the classroom, this delay interrupts the learning flow and can lead to student frustration.

This project proposes a specialized software tool that goes beyond grammatical validation. By integrating a lexical-semantic analysis engine developed in Python, the VSC (VHDL Syntax Checker) allows immediate detection of common errors. The tool not only points out the error but also offers automatic corrections (FIX), turning debugging into an active learning experience [5]. This aligns with current research on formative feedback, which emphasizes that timely and actionable error messages improve learning outcomes.

Unlike industrial tools, the VSC prioritizes response speed and didactic feedback. To test the tool in the laboratory, a "Test Vehicle" with induced errors was designed to compare the software's detection capability against the most frequent errors in the student environment. This paper presents both the technical validation of the engine and a discussion of its pedagogical value as an innovative educational practice.

2. SYSTEM DEVELOPMENT

2.1. General Description Of The VSC

The VHDL Syntax Checker (VSC) is a cross-platform desktop application developed in Python 3.8+. The architecture follows a simplified view-controller model. The graphical user interface (GUI), built with Tkinter, acts as a lightweight development environment. It allows direct code editing with syntax highlighting and management of multi-file projects. The design is portable: it can run both in Python environments and as standalone executables for Windows.

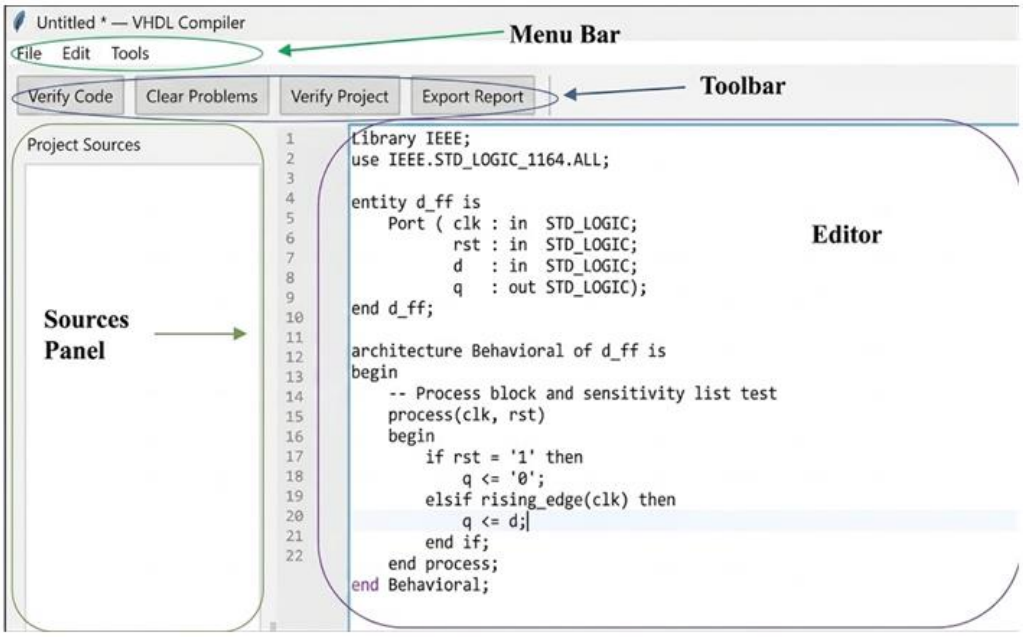

Figure 1. VSC User Interface. The code editor with reserved word highlighting is shown, along with the bottom problem panel, which indicates the line, column, and severity of errors.

2.2. Architecture Of The Error Detection Engine

The core of the tool is a static analysis engine that processes source code without the need for logic synthesis. It operates in two phases:

- Lexical and structural analysis: The analyzer decomposes the .vhd file into tokens and logical lines, validating the mandatory hierarchy of VHDL (Library, Entity, Architecture) [1]. The engine ignores comments and text strings. If keywords such as "is" are missing in the entity, the object is not registered in the symbol table, generating cascading warnings.

- Semantic and heuristic rules: Once the base structure is validated, the code is processed by a specialized rule engine applying 10 verifications (RULE-1 to RULE-10). These rules look for grammatical errors and logical inconsistencies that would typically only appear after minutes of synthesis [4].

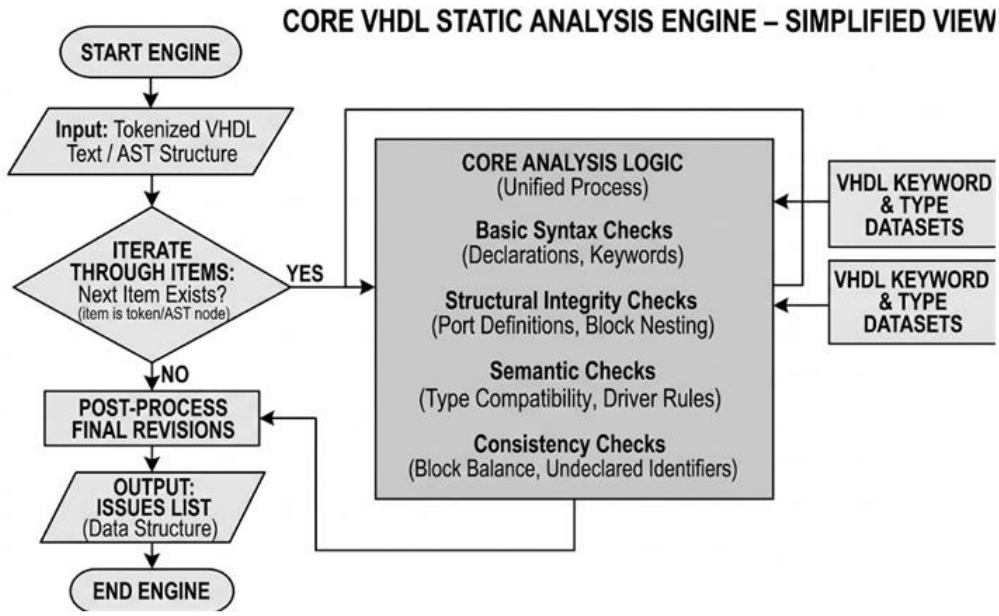

Figure 2. Flowchart of the static analysis engine. It shows the process from code input to error reporting, including the shielding phase and the sequential application of semantic verification rules.

2.3. Implementation Of Technical Rules

The didactic utility of the VSC lies in translating the IEEE 1076 standard into clear, actionable error messages, classified into Error, Warning, and Note [1]. Among the implemented rules:

- Sensitivity management (RULE-2): Detects incomplete sensitivity lists in processes, preventing discrepancies between simulation and hardware.

- Implicit latches (RULE-4): Searches if and case statements for uncovered paths that could infer latches.
- Signal integrity (RULE-9 and RULE-10): Tracks identifiers to detect multiple drivers or undeclared signals.

Table 1. Selected verification rules.

Rule	Level	Technical description	Design impact
RULE-4	Error	Implicit latch in if/case	Prevents instability in combinational logic
RULE-2	Warnin	Incomplete sensitivity list	Prevents simulation/hardware

	g		mismatches
RULE-10	Error	Used identifier not declared	Ensures IEEE 1076 compliance
RULE-9	Error	Multiple drivers on unresolved signal	Prevents electrical collisions

2.4. Automatic Correction System (Auto-Fix)

The module of 15 automatic correction functions (FIX) uses heuristic algorithms to suggest immediate changes, such as correcting the assignment operator (= vs <=) based on context. By offering the solution at the moment of detection, corrective learning is reinforced [6]. This feature transforms the tool from a mere error indicator into an active pedagogical agent.

3. RESULTS AND DISCUSSION

3.1. Validation Protocol: Test Vehicle

To evaluate the VSC, a functional module (modulo_test.vhd) was designed with 10 induced errors (lexical, syntactic, and semantic). This allows stressing the engine and validating the rules in laboratory cases.

Code 1. Test Vehicle With Induced Errors.

```
vhdl
-- Test module with intentional errors
library IEEE;
use IEEE.STD_LOGIC_1164.ALL  -- Error 1: missing ';'
entity modulo_test  -- Error 2: missing 'is'
port (
    clk : in STD_LOGIC;
    1_bad_signal : in STD_LOGIC;   -- Error 3: identifier starts with
number
    data_out : out STD_LOGIC_VECTOR (7 downto 0)   -- Error 4: missing
';'
);
end entity;

architecture behavioral of modulo_test is
begin
    process(clk)
    begin
        if rising_edge(clk) then
            out_signal <= data_in;  -- Error 5: undeclared signal
            data_out <= internal_reg(7 downto 0;   -- Error 6: unclosed
parenthesis
        end if
    end process;
end architecture behavior;  -- Error 7: architecture name mismatch
```

3.2. Detection Analysis and Results Obtained

After processing the test vehicle, the VSC generated a diagnostic report. The results are shown in Table 2.

Table 2. VSC performance against the test vehicle.

Category	Induced error	Result	Technical observation
Syntactic	Missing ; (lines 3, 11)	Detected	Precise identification by the parser
Semantic	= instead of <=	Detected	Sequential assignment rule applied correctly
Semantic	Undeclared signal	Detected	RULE-10: absence in symbol table
Lexical	Identifier starts with number (1_bad_signal)	Missed	False negative: tokenizer accepted initial digit
Structural	Missing "is" in entity	Missed	Caused a cascade effect
Contextual	Architecture closure mismatch	Detected	Warning issued for name inconsistency

3.3. Discussion of Findings: Pedagogical Implications

The validation showed the "cascade effect": when the parser fails to detect a structural keyword (such as is), it does not register the entity, and then multiple "entity not found" warnings appear. This reflects the hierarchical dependency of the engine.

From a pedagogical perspective, this cascade effect is not merely a technical limitation. It represents a valuable teachable moment. When a student omits the is keyword, the subsequent avalanche of warnings visually demonstrates how VHDL's hierarchical structure operates: the entity is the foundation upon which the architecture depends. Instructors can use this behavior to explain why certain errors appear more serious than others and how to prioritize debugging by fixing higher-level issues first.

The missed error in 1_bad_signal indicates that the lexer's regular expression must be adjusted to strictly comply with the IEEE 1076 standard. This is a technical refinement that will be addressed in a future version.

4. CONCLUSIONS

The development and validation of the VHDL Syntax Checker (VSC) lead to the following conclusions:

- Impact on academic productivity and learning flow: The tool reduces debugging times by processing code in milliseconds, eliminating the bottleneck of industrial tools for trivial errors. This keeps students engaged in the design process and reduces frustration, which is critical for maintaining motivation in introductory courses.
- Effectiveness of the semantic engine: Tests with the test vehicle confirmed the robustness of the engine in detecting critical logic and assignment errors (RULE-2, RULE-4, RULE-10), providing timely formative feedback.

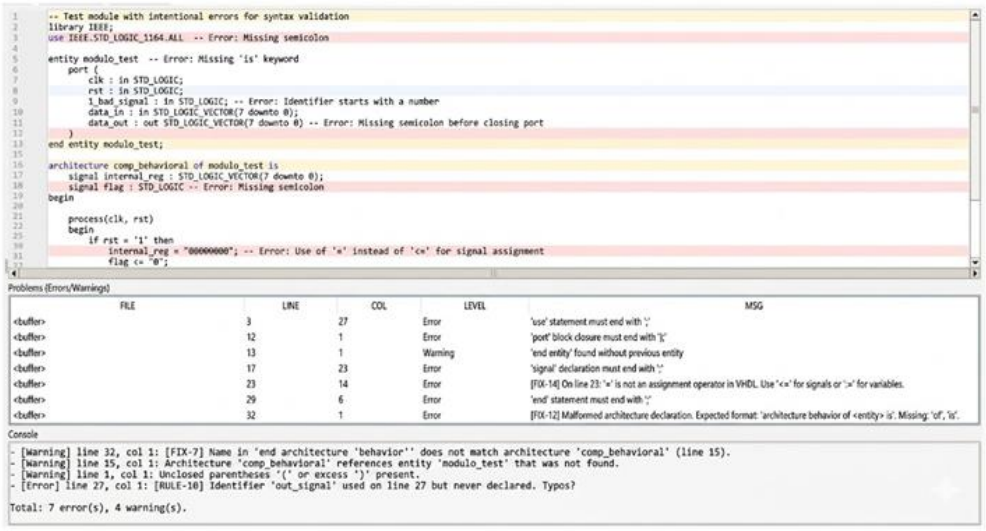

Figure 3. Report exported by the VSC in .csv format, with location and severity of each error.

- Identified limitations as learning opportunities: The cascade effect, while a technical limitation, offers a pedagogical opportunity to teach VHDL hierarchy and debugging prioritization. The false negative in identifier validation will be corrected by refining the regular expressions according to IEEE 1076.

- Pedagogical value of the FIX module: Automatic correction acts as a passive tutor, reinforcing correct syntax at the moment of error. This immediate, actionable feedback is aligned with best practices in educational technology.

As future work, a lexer reengineering is proposed to prevent the propagation of false positives, and a deeper data flow analysis would allow expanding the rule catalog. More importantly, a formal study with students is planned to measure the tool's actual impact on learning outcomes, comparing debugging time and error rates between a control group and an experimental group using the VSC. In summary, the VSC is a viable didactic platform that prepares students for professional design environments while embodying innovative educational practices.

REFERENCES

[1] IEEE Computer Society, IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-2008, 2009.
[2] P. J. Ashenden, The Designer's Guide to VHDL, 3rd ed. Morgan Kaufmann, 2008.
[3] P. P. Chu, FPGA Prototyping by VHDL Examples. Wiley-Interscience, 2008.
[4] S. Brown and Z. Vranesic, Fundamentals of Digital Logic with VHDL Design, 3rd ed. McGraw-Hill, 2009.
[5] J. Bhasker, A VHDL Primer, 3rd ed. Prentice Hall, 1999.
[6] A.V.Aho, M. S. Lam, R. Sethi, and J. D. Ullman, Compilers: Principles, Techniques, and Tools, 2nd ed. Addison-Wesley, 2006.