2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

SAWD: Systemverilog Assertions Waveform-based Development Tool

Ahmed Alsawi, QT Technologies Ireland Limited, Cork, Ireland *(aalsawi@qti.qualcomm.com)*

***Abstract*— In this work, Systemverilog Assertions Waveform-based Development tool is implemented to evaluate Systemverilog assertion temporal expressions directly on a value change dump file allowing faster assertion prototyping and shorter turn-around time. The tool is vendor independent and uses fully open-source python packages to parse the assertion and waveform to generate text reports and illustrative diagrams of assertion evaluation attempts.**

Keywords—Systemverilog Assertions; Value Change Dump; Lark; Simulator; Debugger

I. INTRODUCTION

Systemverilog assertions (SVA) became an important part of the modern verification environment because they are used for both functional simulations and formal verification. Beside verifying the functional behavior, they also provide metrics for coverage closure. SVA describe the behavior of the design over several clock cycles. The evaluation model of SVA depends on a sampling event and the evaluation only happens at the sampling event ticks.

Although there are efforts to automate SVA generation, SVA development is usually a manual task and it can be time-consuming as it takes several iterations to modify, run, and analyze to verify the correctness of temporal expressions including sequences and properties. The process usually starts with writing a prototype of temporal expressions and implementing a manual stimulus to verify the behavior of these expressions. When expressions are mature enough, they are integrated into the final verification environment. However, if those expressions need to be modified, it will lead to a long turn-around time.

This work introduces a new tool accelerating the assertion development flow by running assertions directly on a waveform generated from the final verification environment. The advantage is allowing the user to develop, test and tune temporal expressions before integrating them into the verification environment.

A. *Related work*

Assertion development and waveform analysis have been explored by several approaches. In [1], Klemmer and Große introduced a domain specific language to extract patterns and metrics from the waveform. The analysis language allows users to automate the search and analysis of waveforms but there is no native support for SVA. The authors in [2] proposed an approach to extract assertions from natural language specifications. The automation of assertion generation is implemented using semantic analysis of specification documents and processing the parse tree for antecedent and consequent of an implication. In [3], The authors introduced a method to mine assertions from the design directly. The methodology uses simulation traces and design to generate assertions and runs a formal engine to provide a counter example back to the mining engine to evaluate the generated assertion.

II. APPLICATION

Systemverilog assertions waveform-based development (SAWD) combines several open-source python frameworks to parse SVA, read waveform, and generate evaluation attempts report. Figure 1 shows the high-level architecture of SAWD consisting of the following components

- SVA frontend
- Wave frontend
- Evaluation engine

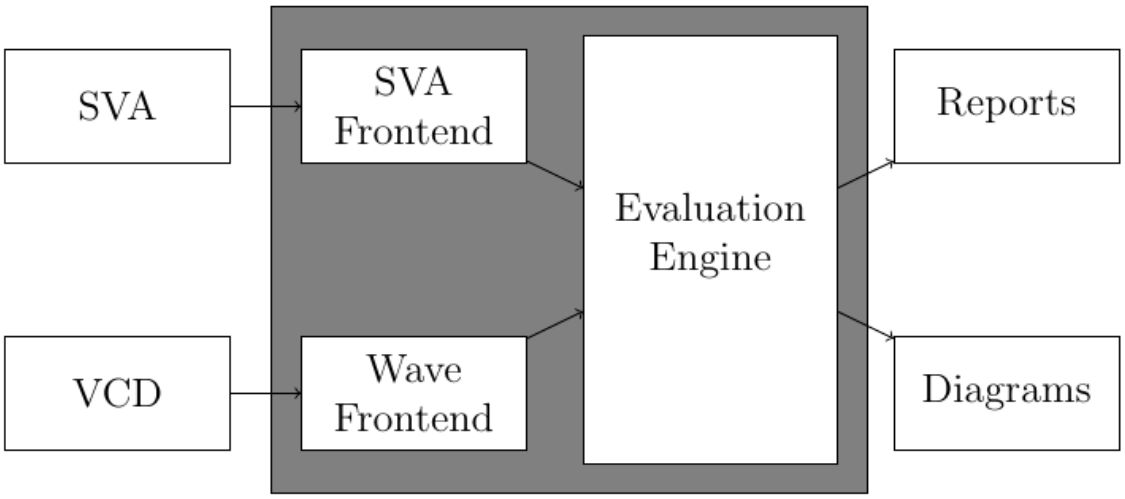

Figure 1. SAWD architecture

A. SVA Frontend

The parser is the main component of the SVA frontend. A grammar for a parser is written in a context free language. A notation like backus-naur form (BNF) is used to describe the context free language. The stages of a typical parser are

- Lexical analysis where the input text is split into tokens based on the language BNF
- The parser consumes the tokens to generate a concrete parse tree which is a representation of the input text containing all the information from original text. For example, whitespace, end of line, brackets are kept in the concrete parse tree
- Transform concrete parse tree to abstract syntax tree (AST) where nodes represent language constructs. whitespace, end of line, brackets are removed from AST as they are not needed by the analysis and evaluation phases

SAWD implements SVA BNF defined by Systemverilog language reference manual [4] using Lark framework [5] extended backus–naur form (EBNF). Lark parser implements the first two stages (lexical analysis and concrete parse tree generation). A custom AST transformations component is implemented to generate SAWD-specific AST for the evaluation engine to process. Figure 2 shows the SVA frontend subcomponents

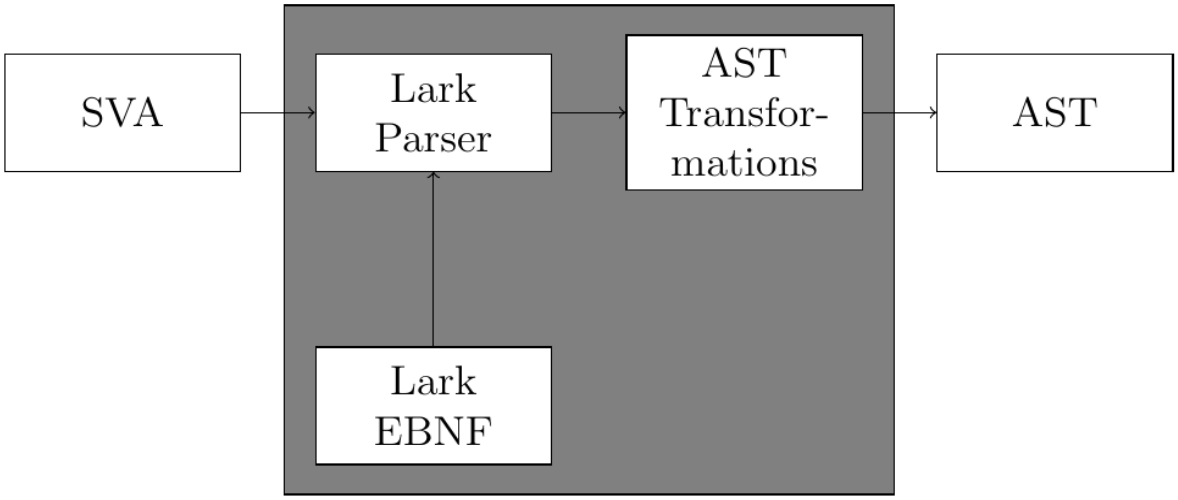

Figure 2. SVA frontend

The assertion in Figure 3 shows a simple temporal expression and the generated AST after transformations is shown in Figure 4.

```
assert_block: assert property (
    @(posedge top.PCLK)
    top.PREADY == 0 ##1 top.PREADY ==  0
    );
```

Figure 3. Example of simple temporal expression

```
NodeType.AST_PROPERTY_SPEC:
  NodeType.AST_CLOCKING_EVENT:
    NodeType.AST_POSEDGE:
      NodeType.AST_IDENTIFIER:
       top.PCLK
 None
  NodeType.AST_PROPERTY:
    NodeType.AST_DELAY:
      NodeType.AST_LITERAL:
       1
      NodeType.AST_EQ:
        NodeType.AST_IDENTIFIER:
         top.PREADY
        NodeType.AST_LITERAL:
         0
      NodeType.AST_EQ:
        NodeType.AST_IDENTIFIER:
         top.PREADY
        NodeType.AST_LITERAL:
         0
```

Figure 4. Generated AST

B. Wave Frontend

Wave frontend generates a generic wave database for the evaluation engine. In order to support value change dump (VCD), vcdvcd python library [6] is used to read VCD file, extract scopes and signal values. VCD is chosen

because it is a vendor-independent waveform database and the availability of open-source parsers. The wave frontend also includes the wave wrapper which is another layer of abstraction for signals and scopes to make it easier to support other waveform formats such as fast signal database (FSDB). Figure 5 shows the wave frontend subcomponents

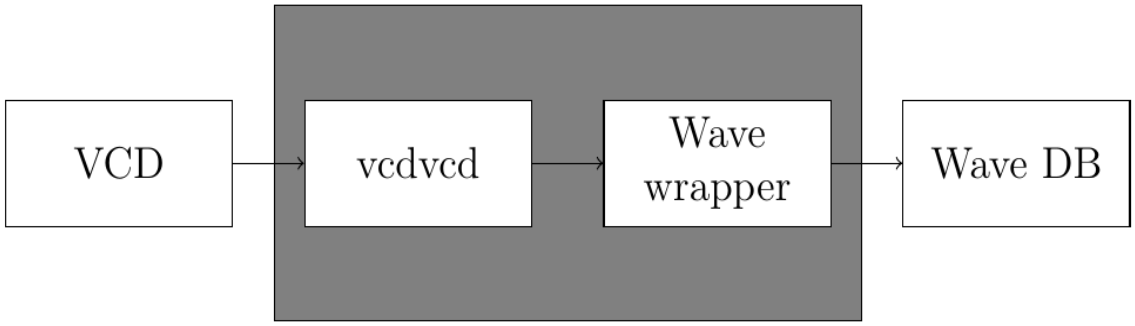

Figure 5. Wave frontend

C. Evaluation Engine

The evaluation engine processes the SAWD-specific AST and the wave database to generate a time-aware expression tree for each evaluation attempt. The time-aware expression tree is a data structure to keep track of start timestamp, end timestamp and expression result. The evaluation engine currently supports sequence operators, property implication, and system tasks ($rose, $fell, $stable, $past).

The evaluation engine generates 2 text reports

- Result of evaluation attempts with the same pass, fail, vacuous and disabled criteria generated by Sytemverilog simulators
- A statistics report with number of attempts in each category

Beside the text reports, the engine uses Graphviz package [7] to generate diagrams of the time-aware expression trees. The diagram contains the assertion tree hierarchy, start timestamp and end timestamp to help understand the evaluation attempt. Figure 6 shows the evaluation engine subcomponents

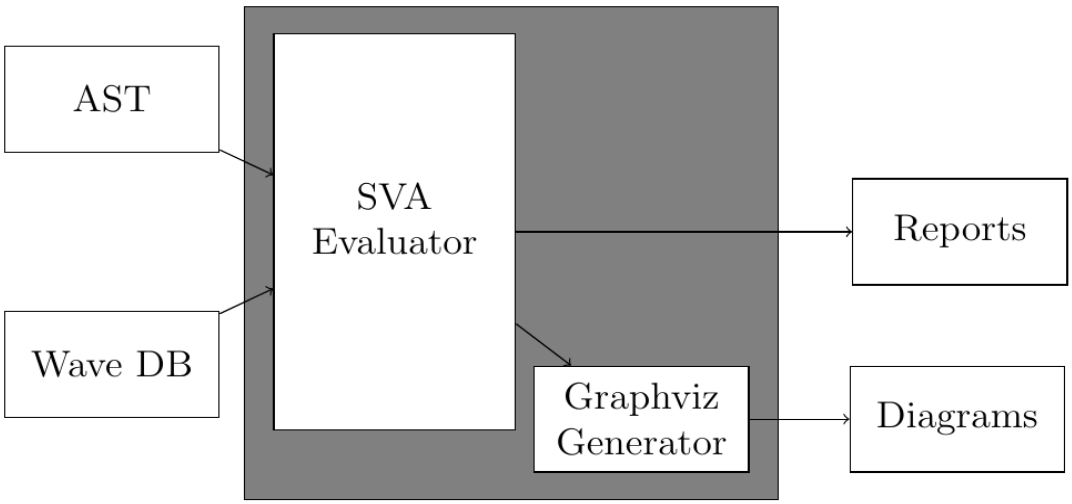

Figure 6. Evaluation Engine

D. Graphical User Interface

The SAWD Graphical User Interface (GUI) is implemented using PyQt5 library [8]. The GUI allows the user to modify, run SAWD flow and view the results all in the same tool. The evaluation attempts list is clickable to open the Graphviz evaluation attempt diagram in a separate window. Figure 7 shows the SAWD GUI with the following fields

- Path to VCD file
- SVA editor
- Evaluation attempts result

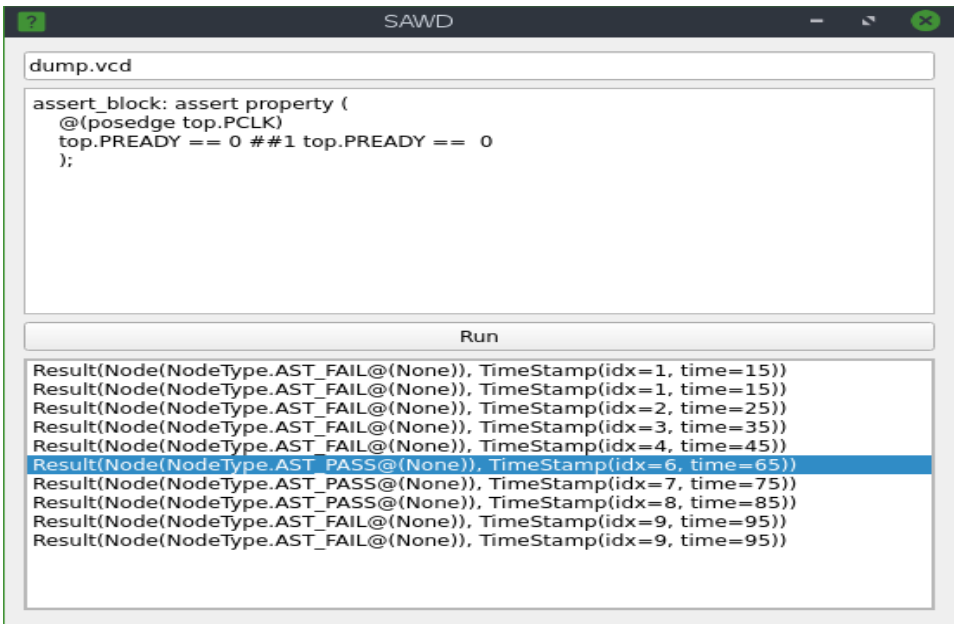

Figure 7. SAWD GUI

III. CASE STUDY: PICORV32 WISBONE ASSERTION

Picorv32 wishbone-based core [9] is used as a case study. Figure 8 shows an assertion to check Wishbone stb/ack behavior. The expression is intentionally using incorrect task $rose to generate a failing evaluation attempt as ack and stb should fall to low at the end of the transaction.

```
assert_block: assert property (
    @(posedge testbench.top.wb_clk)
    disable iff (testbench.top.wb_rst)
    $rose(testbench.top.wb_m2s_stb) |->
          ##1 $rose(testbench.top.wb_s2m_ack)
                ##1 $fell(testbench.top.wb_s2m_ack) &&
     $rose(testbench.top.wb_m2s_stb)
    );
```

Figure 8. Wishbone failing assertion

The tool reports failing attempt failure as shown in Figure [9]. And as expected, the rest of the evaluation attempts are reported as vacuous because of the implication.

```
18:10:21 engine INFO Eval attempt @(TimeStamp(idx=1, time=83080000))
18:10:21 engine INFO Result(Node(NodeType.AST_VACUOUS@(None)), TimeStamp(idx=1, time=83080000))
18:10:21 engine INFO Eval attempt @(TimeStamp(idx=2, time=83090000))
18:10:21 engine ERROR Result(Node(NodeType.AST_FAIL@(None)), TimeStamp(idx=4, time=83110000))
```

Figure 9. SAWD failing assertion report

The SAWD tool also generates a statistics report for the total number of fail/pass/vacuous/disabled evaluation attempts as show in Figure 10.

```
17:16:16 sawd INFO Stats:
    Attempts:6
    Pass:0
    Fail:1
    vacuous:5
    disabled:0
```

Figure 10. SAWD statistics report

To help debug the failing attempt, SAWD generates time-aware expression tree graph shown in Figure 11 to pinpoint the failing expression. The time-aware expression diagram shows that $rose task failed causing the whole assertion to fail.

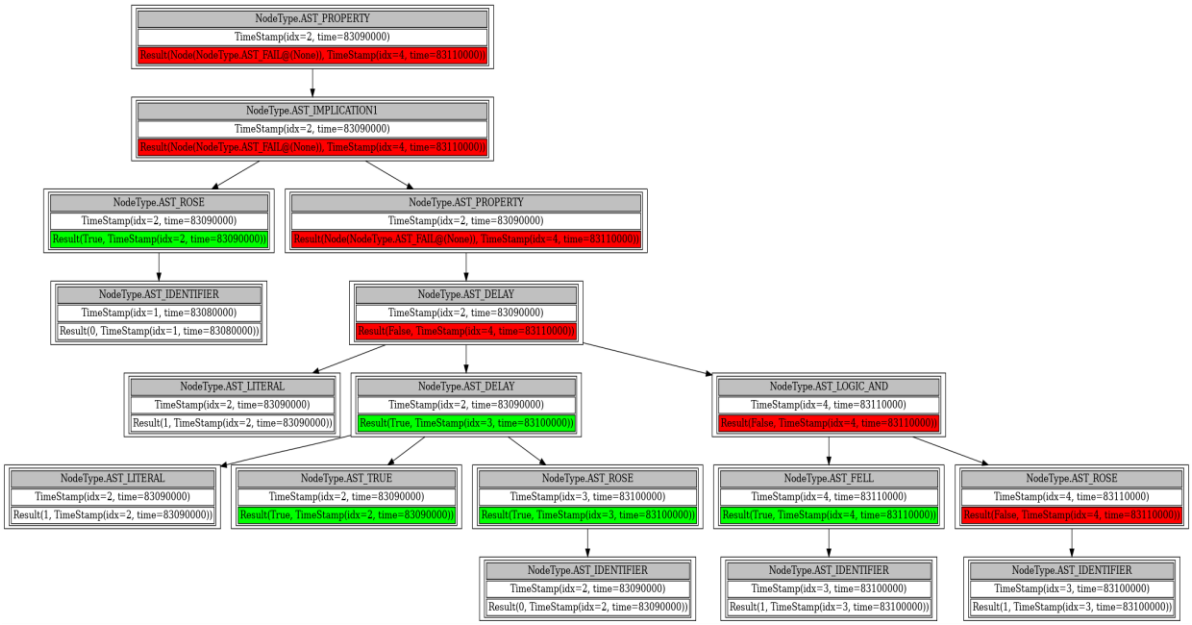

Figure 11. SAWD failing evaluation attempt time-aware expression tree

IV. FUTURE WORK

The following features are planned for the next versions of SAWD

- Support several assertion statements in the same run
- Support standalone sequences and properties to allow reusable temporal expressions
- Support FSDB in the wave frontend

V. CONCLUSIONS

Assertions are indispensable for verification closure, but it is time-consuming to tune and test temporal expressions according to the specifications. In this paper, a tool was developed to test SVA on a waveform saving time to generate manual stimulus or rerun the assertion in the final design. The tool is using open-source packages

for assertion parsing, VCD processing, graphical user interface, and diagram generation. Time-aware expression tree data structure is implemented as a part of the evaluation engine to track evaluation attempts. The results show a failing assertion report and the generated evaluation attempt graph to help debug the failing assertion.

REFERENCES

[1] L. Klemmer and D. Große, "WAL: A Novel Waveform Analysis Language for Advanced Design Understanding and Debugging," 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC), 2022, pp. 358-364, doi: 10.1109/ASP-DAC52403.2022.9712600.

[2] J. Zhao and I. G. Harris, "Automatic assertion generation from natural language specications using subtree analysis," in 2019 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 598-601, 2019.

[3] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy and D. Johnson, "GoldMine: Automatic assertion generation using data mining and static analysis," 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), 2010, pp. 626-629, doi: 10.1109/DATE.2010.5457129.

[4] IEEE Standard for SystemVerilog Unified Hardware Design, Specification, and Verification Language

[5] https://github.com/lark-parser/lark

[6] https://github.com/ZihaoZhao/vcdvcd

[7] https://github.com/graphp/graphviz

[8] https://pypi.org/project/PyQt5

[9] https://github.com/YosysHQ/picorv32