WSVA: A SystemVerilog Assertion to WAL Compiler

Lucas Klemmer Daniel Große
Institute for Complex Systems, Johannes Kepler University Linz, Austria
lucas.klemmer@jku.at daniel.grosse@jku.at

***Abstract*—*SystemVerilog Assertions* (SVA) is an industry standard for specifying properties that describe the correct behavior of a system. Compared to SystemVerilog's immediate assertions, they provide a much more powerful syntax including the ability to specify properties spanning over multiple clock cycles. However, to the best of our knowledge, SVA is not supported by any available open-source *Electronic Design Automation* (EDA) tool. In this paper, we present WSVA, a compiler from SVA to the *Waveform Analysis Language* (WAL).**

I. Introduction

In this paper, we present WSVA[1], a compiler from SVA to the *Waveform Analysis Language* (WAL) [1]. WSVA can compile SVA properties to WAL programs, leveraging WAL's waveform analysis features and its ability to easily act as a backend for other languages. By using WAL as a backend, SVA properties can exploit all features available in WAL. As a consequence, SVA properties can be checked on a waveform. The other way round, WAL programs can also use compiled SVA properties and thus can leverage SVA for example to specify complex signal behaviors.

Currently, WSVA is a proof-of-concept and in an experimental stage. Therefore, it is not intended to be used for proper verification of a design. However, even in its current form, WSVA allows some interesting use cases: First, it gives people without access to costly commercial tools the possibility to get to know and try out SVA. Second, because it operates on waveforms, it allows to quickly develop properties without having to simulate the same design again. And last, because of its current experimental nature, WSVA can serve as a test bed for new ideas in open-source *Electronic Design Automation* (EDA) tools. For example, we plan to integrate WSVA into waveform viewers such as Surfer [2] to provide users a very direct and rich debug and verification environment.

II. Related Work

SystemVerilog Assertions (SVA) is an industry standard for specifying properties that describe the correct behavior of a system. However, to the best of our knowledge, SVA is not supported by any available open-source EDA tools. Tools such as Yosys [3] and Verilator [4] either support only immediate assertions, a limited subset of SVA, or require commercial extensions to unlock SVA support.

Checking SVA properties on waveforms was first presented by SAWD in [5]. However, SAWD was not made available, which prompted us to start developing WSVA.

[1]https://gitlab.com/lklemmer/wsva

```
1 set_cnt: assert property(
2   @(posedge clk)
3   disable iff(rst)
4   start |=> (cnt == value));
```

Listing 1: Simple SVA property example

```
1  (defun sva_set_cnt_single []
2    (define was-disabled? #f)
3    (define started-at TS)
4    (define property-status #f)
5    (defun check-disable [] (set (was-disabled? reset)))
6    (timeframe
7      (set (property-status
8            (do (check-disable)
9                (cond [start (do (step 1)
10                                (check-disable)
11                                (= cnt value))]
12                      [#t #t]))))
13     (unless (|| property-status was-disabled?)
14             (list started-at TS))))
15
```

Listing 2: WAL code generated for simple property

A related research domain is the synthesis of PSL and SVA properties to hardware which has been studied before [6]–[8].

III. SVA to WAL Compilation

In this section, we provide a small example of how SVA properties are compiled into WAL programs. Consider the SVA property shown in Listing 1. This property checks, that on every rising clock edge an asserted *start* signal results in the counter *cnt* being set correctly to the value of the *value* signal in the following clock cycle.

This SVA property is compiled to the WAL function shown in Listing 2. This function checks if the property holds at the current timestamp. Further, another function is compiled which checks if the property holds on the whole waveform, however, it is much simpler, and we omit it due to space limitations.

The variables defined on Lines 2–4 track if the disable condition was true some time, when the property started being evaluated, and the result of the property, i.e., if it is violated or if it holds. Next, the `check-disable` function is defined which is used inside the property to determine if the disable condition is true. Finally, the generated code of the property is on Line 8–12. First, the code checks if the disable condition is true on Line 8. The implication is compiled into a WAL `cond` expression, shown between Line 9 and Line 11. If the antecedent is true (i.e., `start` is high), the time is advanced to the next clock edge (using `step 1`), the disable condition is checked, and the result of the consequent is returned as the result of the implication. If the antecedent is false, the result of the implication is true (we plan to support vacuous results in the future). Finally, if the result was not true and the disable

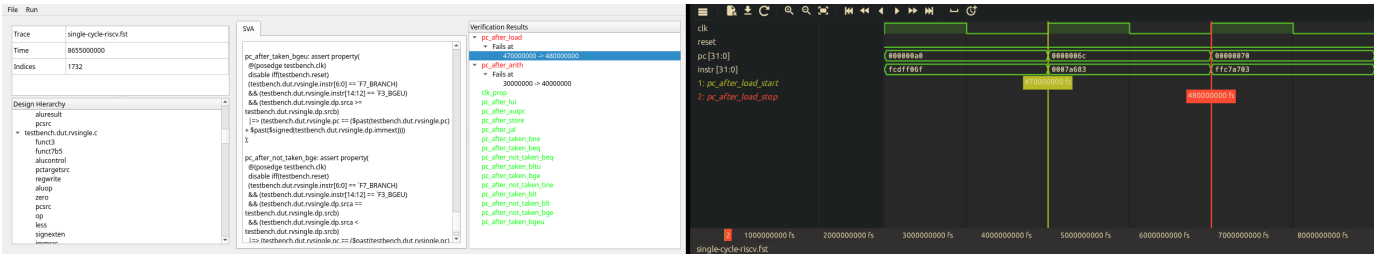

Fig. 1: A screenshot of the GUI and a failing property opened in Surfer.

```
$ wsva run single-cycle-riscv.fst --sv instr.sv regs.sv
Trace: single-cycle-riscv.fst
SVA : instr.sv
Property                    Result
reg_0_is_always_0       : PASS
pc_after_lui            : PASS
pc_after_auipc          : PASS

Trace: single-cycle-riscv.fst
SVA : regs.sv
Property                          Result
reg_1_update_on_write           : FAIL @ 8610000000
reg_1_no_change_on_other_write  : PASS
```

Listing 3: Output of the WSVA CLI showing passing and failing properties.

condition was not activated, a list containing the start and end matching times of the property are returned. If the property was true, or if it was disabled, the function returns a null value.

This is a simple example. However, it shows how WAL code for complex properties can be constructed by following a set of rules for translating SVA operators to WAL.

IV. WSVA Interfaces

The *Command Line Interface* (CLI) allows checking a number of SVA files against a waveform. After finding failing properties, WSVA CLI allows the user to select and open one of the failing properties in the Surfer waveform viewer (see Fig. 1). An exemplary output of the WSVA CLI is shown in Listing 3. Since WSVA requires no licenses, the CLI can check properties in parallel, utilizing as many threads as are available.

The WSVA GUI allows developing and checking properties on a loaded waveform (see Fig. 1). To aid developers in writing properties, information about the waveform (e.g., signals or length) is also shown in a sidebar.

The WSVA compiler does not check properties but emits a WAL file which can be used by other WAL programs. This WAL file contains all required utility functions and the single-time and full-waveform functions of each property.

An example application for using compiled SVA properties is shown in Listing 4. This example, taken from [9], shows how compiled SVA properties can work together with other WAL features such as virtual signals. In this case, a compiled SVA property (`check_forwarding`) is used to detect a bug occurring in a waveform. Using WAL's virtual signals, the faulty signal is shadowed with a virtual signal implementing a bug fix. Finally, the new implementation is checked again using the same compiled SVA property.

```
1  >-> (load "bug.vcd")
2  >-> (require properties)
3  >-> (check_forwarding)
4  ((90000000 90000000))
5  >-> (step-to-ts 90000000)
6  >-> forwardae
7  0
8  >-> (wire forwardae/new
9        (cond [(&& (= rs1e rdm) regwritem rs1e) 2]
10             [(&& (= rs1e rdm) regwritew rs1e) 1]
11             [else 0]))
12 >-> (alias forwardae forwardae/new)
13 >-> forwardae
14 2
15 >-> (check_forwarding)
16 ((90000000 90000000))
17 >-> (wire srca/new
18       (case forwardae
19             [0 rd1e]
20             [1 resultw]
21             [2 aluresultm]))
22 >-> (alias srca srca/new)
23 >-> (check_forwarding)
24 ()
```

Listing 4: WAL shell session documenting the analysis and repair of the forwarding logic.

V. Conclusions

In conclusion, WSVA allows checking if SVA properties hold on a given waveform. The tool is experimental, however, it can be used in various use cases, including quickly developing SVA properties without having to simulate again after updating properties.

VI. Acknowledgments

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

References

[1] L. Klemmer and D. Große, "WAL: a novel waveform analysis language for advanced design understanding and debugging," in *ASP Design Automation Conf.*, 2022, pp. 358–364.

[2] F. Skarman, O. Gustafsson, and L. Klemmer, "Surfer 0.1.0," Feb. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.10653541

[3] C. Wolf and Yosys Contributors, "Yosys open synthesis suite," https://yosyshq.net/yosys/.

[4] W. Snyder and Verilator Contributors, "Verilator," https://www.veripool.org/verilator/.

[5] A. Alsawi, "Sawd: Systemverilog assertions waveformbased development tool," in *Design and Verification Conference and Exhibition Europe*, 2022.

[6] S. Das, R. Mohanty, P. Dasgupta, and P. Chakrabarti, "Synthesis of system verilog assertions," in *Proceedings of the Design Automation & Test in Europe Conference*, vol. 2, 2006, pp. 1–6.

[7] O. Amin, Y. Ramzy, O. Ibrahem, A. Fouad, K. Mohamed, and M. Abdelsalam, "System verilog assertions synthesis based compiler," in *2016 17th International Workshop on Microprocessor and SOC Test and Verification (MTV)*, 2016, pp. 65–70.

[8] M. Boule and Z. Zilic, "Automata-based assertion-checker synthesis of psl properties," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 13, no. 1, pp. 1–21, 2008.

[9] L. Klemmer and D. Große, "Towards a highly interactive design-debug-verification cycle," in *ASP Design Automation Conf.*, 2024, pp. 692–697.