SAWD: Systemverilog Assertions Waveform-based Development tool

Ahmed Alsawi
QT Technologies Ireland
Agenda

• Motivation
• Waveform-based development methodology
• Implementation
  • SVA Frontend
  • Waveform Frontend
  • Evaluation engine
• Graphical user interface

• Example
• Conclusion
Motivation

- SystemVerilog concurrent assertions provide a concise and complicated syntax to define temporal expressions
- The development process can take several iterations to modify, run, and analyze to verify the correctness of an assertion
Waveform-based development methodology

- Evaluating SVA on simulator-agnostic waveform
- Generating failing/passing/vacuous reports
- Generating diagrams for evaluation attempts
Implementation - Architecture

- SVA frontend parses SVA to generate Abstract Syntax Tree (AST)
- Wave Frontend parses VCD to generate Wave DB
- Evaluation engine uses AST and wave DB to generate SVA reports and diagrams.
Implementation - SVA Frontend

• Lark parser provides lexical analysis and parser by reading Lark EBNF to generate a parse tree

• AST transformations are custom transformations implemented to transform the parse tree to AST
Implementation - AST Transformations

assert_block: assert property (  
 \$观察(top.PCLK)  
 top.PREADY == 0  \#1 top.PREADY == 1  
 );

AST

```
NodeType.AST_PROPERTY_SPEC:
  NodeType.AST_CLOCKING_EVENT:
    NodeType.AST_PPOSEDGE:
      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
```
Implementation - Wave Frontend

- Wave frontend uses python package vcdvcd to parse vcd file
- Wave wrapper is an abstraction layer to provide wave DB to the evaluation engine
Implementation - Evaluation Engine

• The SVA evaluator processes AST and wave DB to generate reports for each evaluation attempt

• The Graphviz generator uses time-aware expression and Graphviz utility to generate attempts diagram
Implementation - Reports

Attempts report

Stats report

14:23:31 engine INFO Eval attempt @(TimeStamp(idx=0, time=5))
14:23:31 engine ERROR Result(Node(NodeType.AST_FAIL@(None)), TimeStamp(idx=1, time=15))
14:23:31 engine INFO Eval attempt @(TimeStamp(idx=1, time=15))
14:23:31 engine ERROR Result(Node(NodeType.AST_FAIL@(None)), TimeStamp(idx=1, time=15))
14:23:31 engine INFO Eval attempt @(TimeStamp(idx=2, time=25))
14:23:31 engine ERROR Result(Node(NodeType.AST_FAIL@(None)), TimeStamp(idx=2, time=25))
14:23:31 engine INFO Eval attempt @(TimeStamp(idx=3, time=35))
14:23:31 engine ERROR Result(Node(NodeType.AST_FAIL@(None)), TimeStamp(idx=3, time=35))
14:23:31 engine INFO Eval attempt @(TimeStamp(idx=4, time=45))
14:23:31 engine ERROR Result(Node(NodeType.AST_FAIL@(None)), TimeStamp(idx=4, time=45))
14:23:31 engine INFO Eval attempt @(TimeStamp(idx=5, time=55))
14:23:31 engine INFO Result(Node(NodeType.AST_PASS@(None)), TimeStamp(idx=6, time=65))
14:23:31 engine INFO Eval attempt @(TimeStamp(idx=6, time=65))
14:23:31 engine INFO Result(Node(NodeType.AST_PASS@(None)), TimeStamp(idx=7, time=75))
14:23:31 engine INFO Eval attempt @(TimeStamp(idx=7, time=75))
14:23:31 engine INFO Result(Node(NodeType.AST_PASS@(None)), TimeStamp(idx=8, time=85))
14:23:31 engine INFO Eval attempt @(TimeStamp(idx=8, time=85))
14:23:31 engine ERROR Result(Node(NodeType.AST_FAIL@(None)), TimeStamp(idx=9, time=95))
14:23:31 engine INFO Eval attempt @(TimeStamp(idx=9, time=95))
14:23:31 engine ERROR Result(Node(NodeType.AST_FAIL@(None)), TimeStamp(idx=9, time=95))
14:23:31 sawd INFO Stats:
Attempts:10
Pass:3
Fail:7
vacuous:0
disabled:0
Implementation - Time-Aware Expression Tree

• The time-aware expression tree is a data structure to keep track of start and end timestamps and expression results

```python
class Node:
    def __init__(self, type_, *args, **kwargs):
        self.type_ = type_
        self.children = [Node] = list(args)
        self.ts = Result = None
        self.evaluated = Result = None
```
Graphical user interface

- SAWD Graphical User Interface uses PyQt5
  - Path to VCD file
  - SVA editor
  - Evaluation attempts result
- The evaluation attempts list is clickable to open evaluation attempt diagram in a separate window.
Example – Initial SVA for wishbone stb/ack

VCD

<table>
<thead>
<tr>
<th>Time</th>
<th>wb_clk=1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wb_m2s_stb=0</td>
</tr>
<tr>
<td></td>
<td>wb_s2m_ack=0</td>
</tr>
</tbody>
</table>

SVA

```
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)
};
```

SAWD

Failing evaluation attempt
Example - Failing Attempt review

```java
assert_block: assert property ( 
  @(posedge testbench.top.wx clk) 
  disable iff (testbench.top.wx rst) 
  $rose(testbench.top.wx m2s_stb) \rightarrow
  ($\neg$rose(testbench.top.wx m2s_ack) && $\neg$rose(testbench.top.wx m2s_ack)) 
);
```
Example - After changing $rose to $fell

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) &&  
  $fell(testbench.top.wb_m2s_stb)  
)
Conclusion

• SAWD provides a tool to develop SVA by evaluating SVA on VCD directly without rerunning simulations
• The results show SVA evaluation reports and generated diagrams for passing/failing attempts
• Advantages
  • Simulator-agnostic and using only open-source packages
  • Faster SVA testing and shorter turn-around time
  • Help understand assertion evaluation attempts
Questions?

Contact Ahmed Alsawi at aalsawi@qti.qualcomm.com