Open Source Formal Verification

SBY

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

July 2025

SBY

- SBY is run through a script defining various information
  - Tasks
  - For each task, the check to apply (bmc, prove, cover)
  - The engines to use (checkers, provers, ...)
  - The script command to execute
  - The source files required for the process

The documentation can be found here:
`https://symbiyosys.readthedocs.io/en/latest/reference.html`

Example

Example.sby

```
[tasks]
...

[options]
...

[engines]
...

[script]
...

[files]
...
```

Example

Tasks

```
[tasks]
cover
bmc
prove
```

- We can define as many tasks as we want

Example

Options

```
[options]
mode bmc
depth 20

cover: mode cover
bmc: mode bmc
prove: mode prove
```

- Here we define the depth of BMC to be 20
- We define, for each task, its mode (cover, bmc or prove)
  - Yes, here the task names are identical to the modes
  - Yes, we can have more than one task per mode

Example

- Maybe we are aware of an assertion that fails
  - During the development process
  - Still interesting to push and use CI
  - But the CI will not be happy with the failure
  - So... We can say that a specific task is supposed to fail

Options - bmc is expected to fail

```
[options]
bmc: expect fail
```

Example

Engines

```
[engines]
cover: smtbmc z3
bmc: abc bmc3
prove: abc pdr
```

- For each task, we define the engine

Example

Script

```
[script]
ghdl --std=08 -fpsl sequencer.vhd psl_sequence.vhd -e psl_sequence
prep -top psl_sequence
```

- `ghdl` compiles (synthesizes) the files
- `-std=08` for VHDL-2008 compatibility
- `-fpsl` to interpret PSL in VHDL source files (not necessary if PSL statements are only in vunits)
- `-e <entity>` identifies the top entity
- `prep -top <entity>` prepares the top entity for formal

Example

Script - generics

```
[script]
ghdl --std=08 -gDATASIZE=8 -fpsl sequencer.vhd psl_sequence.vhd -e psl_sequence
prep -top psl_sequence
```

- `ghdl` accepts generics in the form `-g<GENERICNAME>=value`

Example

Files

```
[files]
../src_vhdl/sequencer.vhd
../src_vhdl/psl_sequence.vhd
```

- All the listed files are copied by SBY prior to running the script commands

Tags

- The tasks can define tags
  - Space-separated items following the task name
- In each section, tags can be used to add some information

Example: Tasks

```
[tasks]
bmc0 ERRNO0 bmc   <-- 2 tags are ERRNO0 and bmc
bmc1 ERRNO1 bmc   <-- 2 tags are ERRNO1 and bmc
prove ERRNO0
```

Tags

Example: options and engines

```
[options]
bmc: depth 30          ← Applied for all bmc
bmc: mode bmc
bmc1: expect fail      ← Applied only to one bmc
~prove: depth 30       ← Another option applied to everything except prove
prove: mode prove

[engines]
bmc: abc bmc3          ← Applied for all bmc
prove: abc pdr
```

Tags

Example: script

```
[script]
# Applies to bmc0 and prove
ERRNO0: ghdl --std=08 -gERRNO=0 fpsl sequencer.vhd psl_sequence.vhd -e psl_sequence
# Applies to bmc1
ERRNO1: ghdl --std=08 -gERRNO=1 -fpsl sequencer.vhd psl_sequence.vhd -e psl_sequence
prep -top psl_sequence
```

gtkwave

- If there is an issue with a model checker or a prover, or if cover is successful
- SBY will generate traces
- We can use gtkwave to display the trace:
  `gtkwave <path_to_trace>/trace.vcd`

Testbench

- If there is an issue with a model checker or a prover, or if cover is successful
- SBY will generate a verilog testbench
- It can obviously be used in any simulator to rerun the case, and is located in: `<path_to_trace>/trace_tb.v`