Open Source Formal Verification

OSFV - Continuous integration and scripting

Yann Thoma

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

July 2025

Introduction

1. Integration into a CI pipeline
   - As everything is scripted, easy to integrate in a pipeline
2. Scripting for generic parameters
   - Useful to select what combination of parameters should be verified

CI Integration

- If the `.sby` files are available, then quite straightforward
- Requires a docker image prepared with oss-cad-suite
- The following example works for github
- It runs all the `.sby` files it finds in a specific directory when someone pushes on the server

DockerFile

```
FROM ubuntu:22.04
ARG USERID=500
ARG GROUPID=${USERID}
ENV USERNAME=cern
ENV GROUPNAME=${USERNAME}
ENV USERHOME=/home/${USERNAME}/
ENV WORKDIR=${USERHOME}/formal_verif

# Install base
RUN apt update && \
apt upgrade -y
RUN DEBIAN_FRONTEND=noninteractive apt install -y git python3-full build-essential perl
    rsync file wget cpio unzip bc

# Setup environment
RUN echo '%sudo ALL=(ALL) NOPASSWD: ALL' >> /etc/sudoers && \
    addgroup --gid ${GROUPID} ${GROUPNAME} && \
    adduser --disabled-password --shell /bin/bash --uid ${USERID} --ingroup ${GROUPNAME}
        --home ${USERHOME} ${USERNAME} && \
    adduser ${USERNAME} sudo && \
    mkdir ${WORKDIR} && \
    chown ${USERNAME}:${GROUPNAME} -R ${WORKDIR}
```

DockerFile

```
WORKDIR ${WORKDIR}

# Install OSS CAD Suite
RUN wget https://github.com/YosysHQ/oss-cad-suite-build/releases/download/2025-06-27/oss-
    cad-suite-linux-x64-20250627.tgz &&\
tar -xzf oss-cad-suite-linux-x64-20250627.tgz -C /opt/ &&\
rm -rf oss-cad-suite-linux-x64-20250627.tgz

# Instead of doing:
#     source oss-cad-suite/environment
# do it by hand, else it fails
ENV VIRTUAL_ENV=/opt/oss_cad_suite
ENV VIRTUAL_ENV_PROMPT='OSS CAD Suite'
ENV PATH="/opt/oss-cad-suite/bin:/opt/oss-cad-suite/py3bin:$PATH"
ENV VERILATOR_ROOT="/opt/oss-cad-suite/share/verilator"
ENV GHDL_PREFIX="/opt/oss-cad-suite/lib/ghdl"
```

yml file

```yaml
name: Formal Verification

on:
    # Launch when modifications to the code folder are pushed
    push:
        paths:
        - code/**
    # Allows you to run this workflow manually from the Actions tab
    workflow_dispatch:
```

yml file

```yaml
jobs:
    formal-verif:
        name: Formal Verification
        runs-on: ubuntu-latest
        container: redscalculator/cern-formal
        env:
            SRC_FOLDER: "code"
        steps:
            # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access
                it
            - uses: actions/checkout@v4
            # Find all .sby scripts in all subfolders of code/ and run them in the parent
                folder of the script
            - name: Run Formal Verif
              run: |
                cd ${SRC_FOLDER}
                for sby_path in $(find . -iname "*.sby"); do
                    sby_script=${sby_path##*/}
                    cd ${sby_path%/*}
                    sby --prefix ../verif_out/ --yosys "yosys -m ghdl" -f $sby_script
                    cd -
                done
```

Better option

Only run the .sby files where code has changed

```
- name: Run Formal Verif
  if: steps.changed-files.outputs.any_changed == 'true'
  env:
    ALL_CHANGED_FILES: ${{ steps.changed-files.outputs.all_changed_files }}
  working-directory: ${{ github.repository }}
  run: |
    all_changed_dirs=$(for f in ${ALL_CHANGED_FILES}; do dirname "$f"; done |
        sort -u)
    for cdir in $all_changed_dirs; do
      rcdir="${cdir}/.."
      echo "verify folder ${rcdir} ..."
      for sby_path in $(find $rcdir -iname "*.sby"); do
          sby_script=${sby_path##*/}
          cd ${sby_path%/*}
          sby --prefix ../verif_out/ --yosys "yosys -m ghdl" -f $sby_script
          cd -
      done
    done
```

Handling generics

- Three options:
  1. Hand-writing all options
  2. Use python to help with the process
  3. Use python and use tasks tags to help with the process

Hand-writing

- Example: ALU with a generic parameter `SIZE`

run_generic.sby

```
[tasks]
cover
prove1
prove2
prove4
prove16

[options]
depth 20

cover: mode cover
prove1: mode prove
prove2: mode prove
prove4: mode prove
prove16: mode prove
```

Different versions for the proof

Different versions for the proof

Hand-writing

run_generic.sby

```
[engines]
cover: smtbmc z3
prove1: abc pdr
prove2: abc pdr
prove4: abc pdr
prove16: abc pdr

[script]
cover:   ghdl --std=08 -gSIZE=16 -fpsl alu.vhd alu.psl -e alu
prove1:  ghdl --std=08 -gSIZE=1  -fpsl alu.vhd alu.psl -e alu
prove2:  ghdl --std=08 -gSIZE=2  -fpsl alu.vhd alu.psl -e alu
prove4:  ghdl --std=08 -gSIZE=4  -fpsl alu.vhd alu.psl -e alu
prove16: ghdl --std=08 -gSIZE=16 -fpsl alu.vhd alu.psl -e alu
prep -top alu

[files]
../src_vhdl/alu.vhd
../src_vhdl/alu.psl
```

Different versions for the proof

Idem

Python's help

- Python commands can be added in the `.sby` file, surrounded by

```
--pycode-begin--
# Some code here
--pycode-end--
```

- However each code snippet is independant, so no sharing of data

Python's help

run_generic_python.sby

```
[tasks]
cover SIZE=16
--pycode-begin--
for t in "1 2 4 16".split():
  output("prove{} SIZE={}".format(t, t))
--pycode-end--

[options]
depth 20

cover: mode cover
--pycode-begin--
for t in "1 2 4 16".split():
  output("prove{}: mode prove".format(t))
--pycode-end--
```

Python's help

run_generic_python.sby

```
[engines]
cover: smtbmc z3
--pycode-begin--
for t in "1 2 4 16".split():
  output("prove{}: abc pdr".format(t))
--pycode-end--

[script]
--pycode-begin--
for t in "1 2 4 16".split():
      output("ghdl --std=08 -gSIZE={} -fpsl alu.vhd alu.psl -e alu".format(t))
--pycode-end--
prep -top alu

[files]
../src_vhdl/alu.vhd
../src_vhdl/alu.psl
```

Python's help - tags

- Tasks can have a list of tags

```
[tasks]
task1 tag1 tag2
task2 tag3 tag4
```

- These tags can be accessed in python code

```
--pycode-begin--
for t in tags:
  # do something
```

- Warning: the task name is also a tag

Python's help - tags

run_generic_python_tags.sby

```
[tasks]
cover SIZE=16
--pycode-begin--
for t in "1 2 4 16".split():
  output("prove{} SIZE={}".format(t, t))
--pycode-end--

[options]
depth 20

cover: mode cover
--pycode-begin--
for t in tags:
  if "SIZE=" in t:
    output("prove{}: mode prove".format(t.replace("SIZE=", "")))
--pycode-end--
```

Python's help - tags

run_generic_python_tags.sby

```
[engines]
cover: smtbmc z3
--pycode-begin--
for t in tags:
  if "SIZE=" in t:
  output("prove{}: abc pdr".format(t.replace("SIZE=", "")))
--pycode-end--

[script]
--pycode-begin--
for t in tags:
  if "SIZE=" in t:
      output("ghdl --std=08 -g{} -fpsl alu.vhd alu.psl -e alu".format(t))
--pycode-end--
prep -top alu

[files]
../src_vhdl/alu.vhd
../src_vhdl/alu.psl
```

Python's help

- Both approaches can be extended to multiple generic parameters

Configs per task

- Each task ends up with a specific config that will be run
- We can dump the config of a task with the command

```
sby --dumpcfg <sbyFile> <taskName>
```

- Valid with or without python snippets

Example

- For instance, for `prove4`:

```
sby --dumpcfg run_generic_python_tags.sby prove4

[options]
depth 20

mode prove

[engines]
abc pdr

[script]
ghdl --std=08 -gSIZE=4 -fpsl alu.vhd alu.psl -e alu
prep -top alu

[files]
../src_vhdl/alu.vhd
../src_vhdl/alu.psl
```