Software for embedded systems

Verification Lab.
(Assertion-based verification)

Samuele Germiniani
samuele.germiniani@univr.it

Alessandro Danese
alessandro.danese@univr.it

January 13, 2020

Simple-platform case study

How to download the simple platform:

```
git clone https://github.com/SamueleGerminiani/simple_platform.git
```

Environment set-up:

1. cd simple_platform
2. source env_setup.sh

Makefile menu

How to open the Makefile menu (terminal):

1. cd questa.simulation
2. make

Makefile menu

Once started, the following menu should appear on the terminal

```
================================================================================

 USAGE: make RECEPIE|TARGET
 Authors: Alessandro Danese (alessandro.danese@univr.it)
          Samuele Germiniani (samuele.germiniani@univr.it)

 --- RECIPES ------------------------------------------------------------------
 simulation              => Performs: clean, compile_s, and run_s
 mining_bl_master        => Performs: clean, and assertion mining for buslayer (master)
 mining_bl_slave_0       => Performs: clean, and assertion mining for buslayer (slave_0)
 mining_bl_slave_1       => Performs: clean, and assertion mining for buslayer (slave_1)
 mining_camellia         => Performs: clean, and assertion mining for camellia
 mining_transmitter      => Performs: clean, and assertion mining for transamitter
 ABV                     => Performs: clean, and Assertion-based Verification
 faultC_bl_master        => Performs: clean, and fault coverage for buslayer (master)
 faultC_bl_slave         => Performs: clean, and fault coverage for buslayer (slave)
 faultC_camellia         => Performs: clean, and fault coverage for camellia
 faultC_transmitter      => Performs: clean, and fault coverage for transmitter

 --- TARGETS ------------------------------------------------------------------
 check_faults            => Performs: fault-coverage with faults.txt file
 compile_s               => Compilings DUT
 run_s                   => Runs simulation.

 --- ADMINISTRATIVE TARGETS ---------------------------------------------------
 help                    => Displays this message.
 clean                   => Removes all intermediate and log files.

================================================================================
```

Simulating the simple platform

How to simulate the simple platform

1. make simulation

The command *make simulation* compiles the platform and the firmware source code (directory firmware).

Afterwards, it runs a simulation.
The sim.vcd file records the values of any register/wire/port of the platform. The transactor_log.txt file records any read_transaction and write_transaction performed by the firmware.

Assertion-based verification (ABV)

How to perform ABV with the simple platform

1. make ABV

The command *make ABV* compiles the source code of the platform, the firmware source code, and the verification unit in the files: buslayer_master.psl, buslayer_slave.psl, camellia.psl and transmitter.psl (sse_lesson1/vcs.simulation/psl).

Verification unit

How add assertions

1. open simple-platform/simple_platform.srcs/assertions/module_name.sv
2. add a checker and bind it to a target module as shown in the SVA lesson.

Exercise 1 - Capture a behavior with an assertion, Part 1

In the file simple-platform/doc/buslayer_spec.pdf you can find the specifications for the buslayer. Both read and write transactions(master) present a similar behavior, can you write an assertion that capture this behavior?

Steps:

1. Read the buslayer(master) specifications to find the common behaviour between a read and a write transaction.
2. Write an assertions capturing the behavior.
3. Execute an assertion based verification to check if your assertion capture exactly the intended behavior.
   N.B. In this example a correct assertion must never fail.

Exercise 1 - Capture a behavior with an assertion, Part 2

Now you should have an assertion that never fails when performing ABV. It doesn't mean that your assertion is correct! The assertion could be vacuous (trivially true) or never tested: this is not formal verification, we are verifying an assertion only on few inputs of the design. How can you "prove" that the assertions capture exactly the intended behavior?
Hint: try to count the occurrence of the expected behavior and compare it with the number of times the assertion is satisfied.