Software for embedded systems

Verification Lab.
(Fault coverage)

Samuele Germiniani
samuele.germiniani@univr.it

Alessandro Danese
alessandro.danese@univr.it

January 25, 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
```

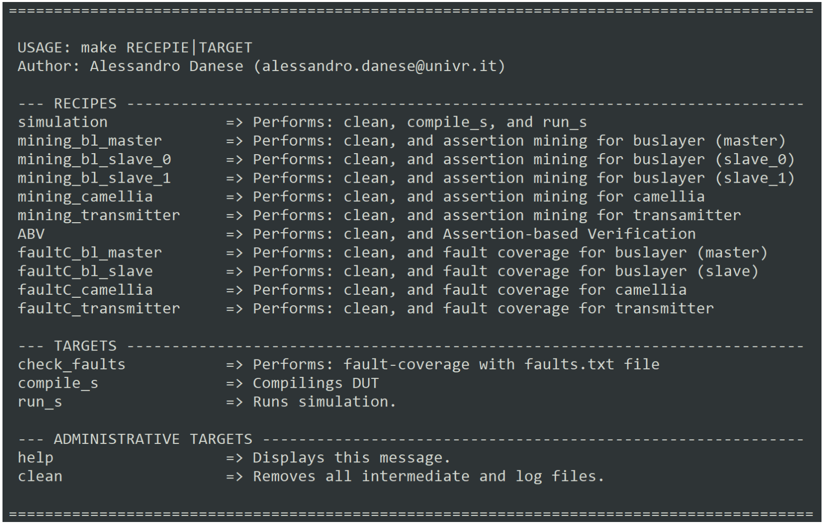
Makefile menu

How to open the Makefile menu (terminal):

1. cd questa.simulation
2. make

Makefile menu

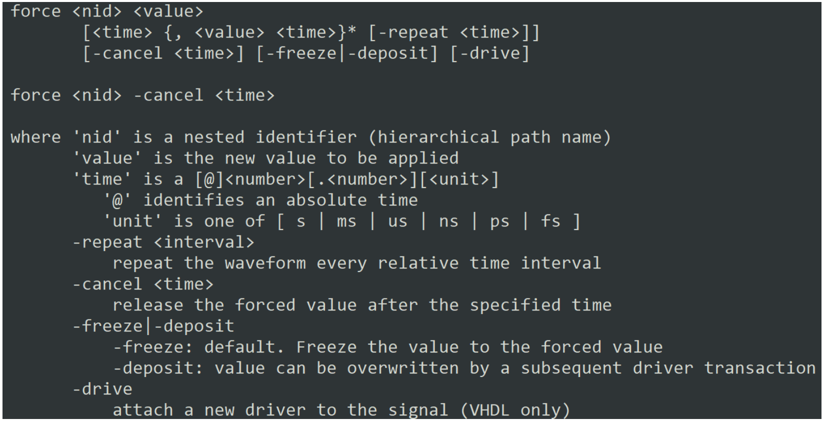
Once started, the following menu should appear on the terminal



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

 USAGE: make RECEPIE|TARGET
 Author: Alessandro Danese (alessandro.danese@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.

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

Force

The command force is a directive requiring the hardware simulator to force a value in a register/signal/port.



```
force <nid> <value>
       [<time> {, <value> <time>}* [-repeat <time>]]
       [-cancel <time>] [-freeze|-deposit] [-drive]

force <nid> -cancel <time>

where 'nid' is a nested identifier (hierarchical path name)
      'value' is the new value to be applied
      'time' is a [@]<number>[.<number>][<unit>]
         '@' identifies an absolute time
         'unit' is one of [ s | ms | us | ns | ps | fs ]
      -repeat <interval>
          repeat the waveform every relative time interval
      -cancel <time>
          release the forced value after the specified time
      -freeze|-deposit
          -freeze: default. Freeze the value to the forced value
          -deposit: value can be overwritten by a subsequent driver transaction
      -drive
          attach a new driver to the signal (VHDL only)
```

Example

Forcing the bit input EN of camellia

- sim1.p.slave_0.camallia_u.EN 1'b0
- sim1.p.slave_0.camallia_u.EN 1'b1

Forcing the third bit of data input array of Transmitter

- sim1.p.slave_1.transmitter.data(2) 1'b0
- sim1.p.slave_1.transmitter.data(2) 1'b1

Fault coverage

How to perform fault coverage with the simple platform.

Example bl_master

1. cd sse_lesson4/questa.simulation
2. make faultC_bl_master

The command *make faultC_bl_master* injects forces to simulate stuck-at faults. The generated file coverage.txt reports the assertions failed for each injected fault.

Exercise - 1

Fault coverage on all assertions generated in the previous lessons.

1. Define a set of fault locations for the components bl_master, wishbone and camellia.
2. Insert the set of fault in the correct fault files "faults/bl_master_faults.txt", "faults/wishbone_faults.txt" and "faults/camellia_faults.txt" (one fault for each line).
3. No need to define new assertions, all the property are already inserted in the assertion files in "simple_platform/simple_platform.srcs/assertions/*.sv"
4. Run a fault coverage analysis for each component to check if all injected faults are covered.
5. Repeat steps 2 and 3 until all injected faults are covered by at least an assertion.