

# Software for embedded systems

Verification Lab.  
(Fault coverage)

Alessandro Danese

University of Verona, Verona, Italy  
alessandro.danese@univr.it

January 15, 2019

# Simple-platform case study

How to download the simple platform:

```
git clone https://AlessandroDanese@bitbucket.org/AlessandroDanese/sse-verifica.git
```

How to open the simple platform-project case study:

- 1 open Vivado
- 2 File -> Open project... ->  
path2/sse-verifica/simple-platform/simple\_platform.xpr

Once started, the following window should appear

The screenshot shows the Vivado IDE interface with the Project Manager window open. The Project Manager window is divided into three main sections: Sources, Project Summary, and Design Runs.

**Sources:** This section shows the project hierarchy. It includes Design Sources (2), Non-module Files (1), and Constraints (1). The Design Sources are further categorized into:

- sys: syscon (syscon.x)
- slave\_0: camelia\_wrapper (camelia\_wrapper.x)
  - slave\_interface: buslayer\_slave (buslayer\_slave.x)
  - camelia\_u: camelia (camelia.x) (3)
- slave\_1: serial\_transmitter\_wrapper (serial\_transmitter\_wrapper.x) (2)
  - slave\_interface: buslayer\_slave (buslayer\_slave.x)
  - transmitter: serial\_transmitter (serial\_transmitter.x)
- core: core\_wrapper (core\_wrapper.x) (1)
  - master\_interface: buslayer\_master (buslayer\_master.x)
  - bus: wishbone\_bus (wishbone\_bus.x) (1)
    - selector: wishbone\_bus\_selector (wishbone\_bus\_selector.x)

The Simulation Sources section shows:

- sim\_1 (2)
  - Non-module Files (1)
  - simt (simt.x) (1)
  - p: simple\_platform (simple\_platform.x) (5)

**Project Summary:** This section provides details about the project settings and board information.

**Settings:**

- Project name: simple\_platform
- Project location: C:/Users/danes/Documents/repository/simple\_platform
- Product family: Zynq-7000
- Project part: PYNQ-Z1 (xc7z020clg400-1)
- Top module name: simple\_platform
- Target language: Verilog
- Simulator language: Verilog

**Board Part:**

- Display name: PYNQ-Z1
- Board part name: www.digilentinc.com/pynq-z1/part0-1-0
- Connectors:
- Repository path: C:/Xilinx/Vivado2017.4/data/boards/board\_files
- URL: http://www.pynq.io
- Board overview: PYNQ-Z1

**Synthesis and Implementation:**

Synthesis		Implementation	
Status:	Not started	Status:	Not started
Messages:	No errors or warnings	Messages:	No errors or warnings
Part:	xc7z020clg400-1	Part:	xc7z020clg400-1
Strategy:	Vivado Synthesis Defaults	Strategy:	Vivado Implementation Defaults

**Design Runs:** This section shows the results of the synthesis and implementation runs.

Name	Constraints	Status	WNS	THS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMS	URAM	DSP	Start	Elapsed	Run Strategy	Report State
synth_1	constraints_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2017)	Vivado Synthesis Defaults
impl_1	constraints_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2017)	Vivado Implementation Defaults

# EDA tools

How to download EDA tools:

- 1 `scp esd-student@esd-srv01.scienze.univr.it:/tmp/esdlab.tar.gz`  
(password: esd-student)
- 2 `tar -xvf esdlab.tar.gz`
- 3 `cd esdlab`
- 4 `source start_eda.bash`

# Makefile menu

How to open the Makefile menu (terminal):

- 1 `cd path2/sse-verifica/simple-platform/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.
=====
```

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 transmitter

- 1 `cd sse_lesson1/vcs.simulation`
- 2 `make faultC_transmitter`

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

# Exercise - 1

Define a set of PSL properties for the transmitter component such that any fault location listed in `questa.simulation/faults/transmitter_faults.txt` is covered by at least a property.

## Exercise - 2

Define a set of fault locations for the components `buslayer_master`, and `buslayer_slave`.

For each component, define a set of PSL properties such that any fault is covered by at least a property.