Software for embedded systems

Verification Lab.
(Assertion-based verification)

Samuele Germiniani
samuele.germiniani@univr.it

Alessandro Danese
alessandro.danese@univr.it

January 17, 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 - Find a bug with an assertion

The current implementation of the simple-platform contains a major fault making a certain signal stuck at a constant value.
The fault is so severe that the execution can hardly progress.
Read the bus_layer specifications and write down assertions that should hold in an correct implementation in order to reach a better understanding of the fault (what signal is stuck at a constant value)?.
Once the bugged signal is found, inspect the code to find the cause of the bug and correct it (It is a very easy fix).
Add the assertions to
simple_platform/simple_platform.srcs/assertions/bl_master_assertions.sv
See the next page for an hint.

Exercise 1 - HINT

Start from this specifications:

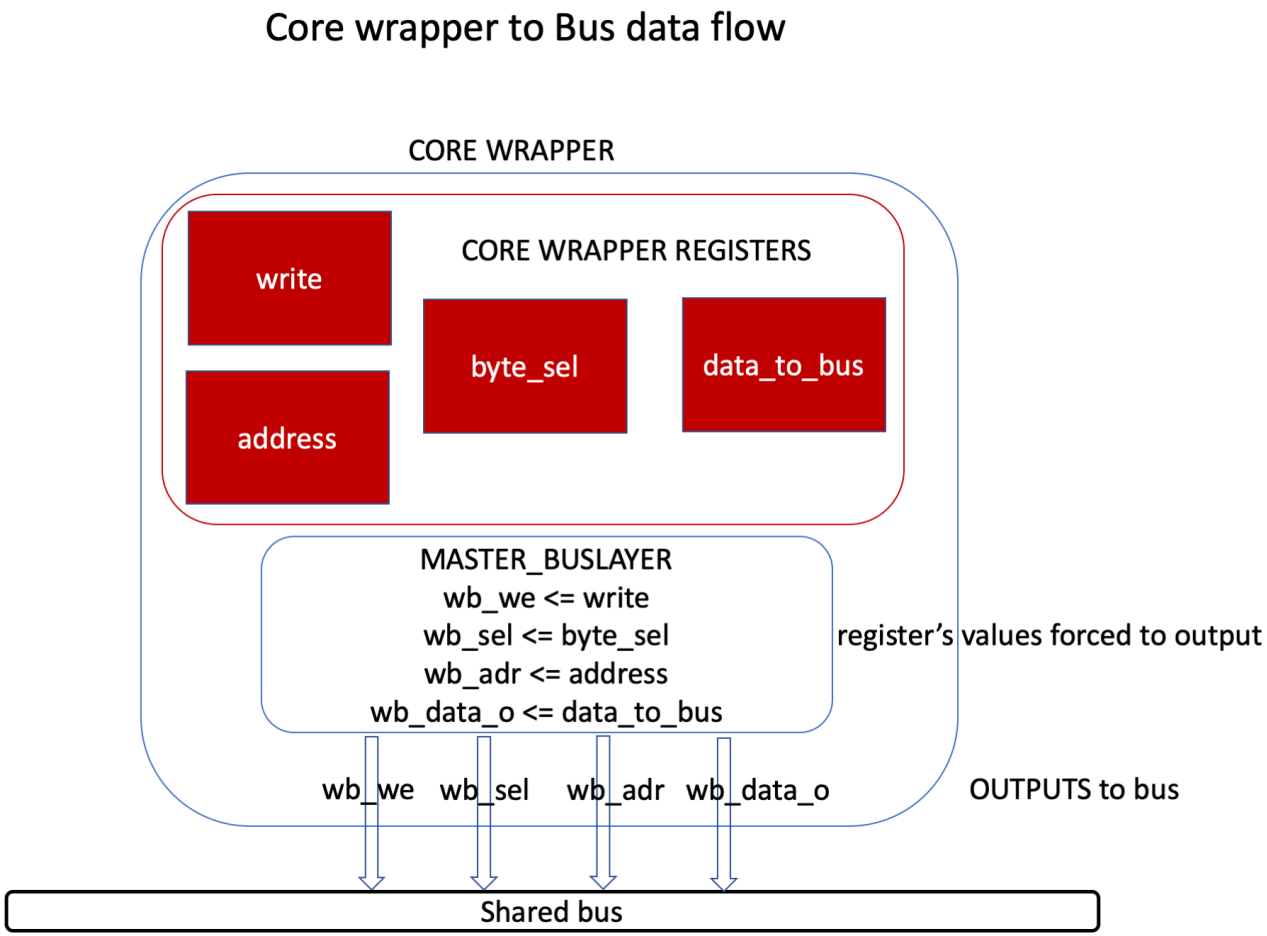
Write transaction (master) The module-interface for a write transaction works as follows:
CLOCK EDGE 0

- MODULE presents a valid address on [ADDRESS_I]
- MODULE presents a valid data on [DATA_TO_BUS_I]
- MODULE asserts [WRITE_I]
- MODULE presents bank select [BYTE_SEL_I]
- MODULE asserts [REQUEST_I]
- BUSY_O is negated

CLOCK EDGE 1

1. BUSLAYER asserts BUSY_O in response to asserted [REQUEST_I], and starts a new bus WRITE CYCLE

Exercise 2 - Write assertions to check the data-flow



Exercise 2 - Write assertions to check the data-flow

When the firmware needs to perform a read/write operation, it forwards the task to the core wrapper (read_transaction, write_transaction). The core wrappers interfaces with the bus with the bus_layer which contains the logic (FSM + DataPath) to implement the wishbone protocol to perform read/write cycles. When a new write cycle begins (request is asserted), the registers in the core wrappers retain the values that must be forwarded to the bus by the bus_layer. Write assertions checking that the values assumed by the output ports to the shared_bus are the same values assumed by the core_wrapper registers one clock tick before. You can use the $past operator to achieve this goal (see the manual). See the figure in the previous page to find out what ports/registers must be checked in the assertions. Add the assertions to simple_platform/simple_platform.srcs/assertions/bl_master_assertions.sv

Exercise 3 - Write a “complex” assertion

Read the specifications for the wishbone protocol (in bus_layer.pdf).
Write an assertion highlighting the behavior of a write cycle (the master forward a value to a slave).
Add the assertion to simple_platform/simple_platform.srcs/assertions/wishbone_assertions.sv
Hint: see the specifications in the next pages.

Exercise 3 - Write a “complex” assertion

WISHBONE BUS WRITE CYCLE

CLOCK EDGE 0

- MASTER presents a valid address on [ADR_O]
- MASTER presents a valid data on [DAT_O]
- MASTER asserts [WE_O]
- MASTER presents bank select [SEL_O]
- MASTER asserts [CYC_O] and [STB_O]

SLAVE consumes data immediately

CLOCK EDGE 1

- responding SLAVE asserts [ACK_I]

CLOCK EDGE 2

- MASTER negates [STB_O] and [CYC_O] to indicate the end of the cycle
- Responding SLAVE negates [ACK_I] in response to negated [STB _O]

Exercise 3 - Write a “complex” assertion

SLAVE does not consume data immediately

CLOCK EDGE 1

- responding SLAVE asserts [STALL_I]
- MASTER negates [STB_O] in response to asserted [STALL_I]

CLOCK EDGE N

- responding SLAVE negates [STALL_I] and asserts [ACK_I] to indicate data was accepted

CLOCK EDGE N+1

- MASTER negates [CYC_O] to indicate the end of the cycle
- responding SLAVE negates [ACK_I] in response to negated [CYC_O]