

# Software for embedded systems

Verification Lab.  
(simple platform - overview)

Alessandro Danese

`alessandro.danese@univr.it`

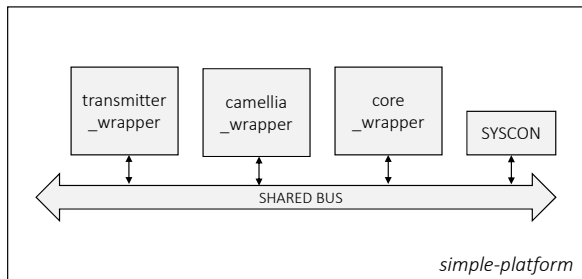
Samuele Germiniani

`samuele.germiniani@univr.it`

January 19, 2020

# Case study: the simple platform

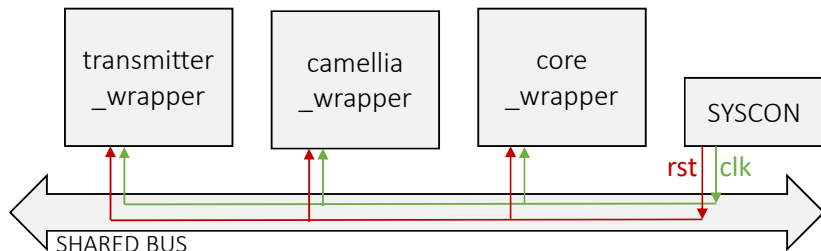
## Overview of the case study used in SSE Verification Lab



- **SYSCON**  
clock and reset signal generator
- **Core\_wrapper**  
platform controller
- **Camellia\_wrapper**  
cipher
- **Transmitter\_wrapper**  
parallel to serial transmitter

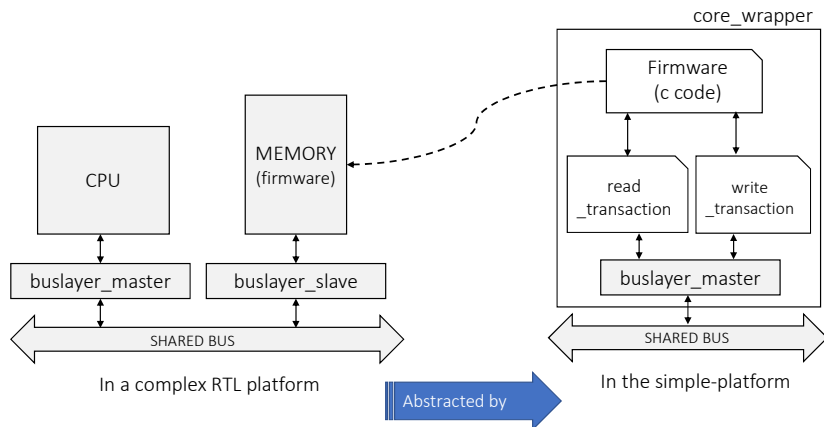
# SYSCON

The SYSCON component generates the clock signal and the reset signal, for all platform's complements.



# core\_wrapper

The core\_wrapper component is an abstraction of a RTL CPU executing a firmware. The firmware code, which is usually stored in memory, is compiled and executed together with the platform (see next slides).



# read\_transaction

The `core_wrapper`, which behaves as a transactor, provides the following two Verilog tasks to interact with the platform's components:

The `read_transaction` provides the reading functionality.

```
task read_transaction (input int addressIn, input int byteSelIn, output int
dataOut, output byte errorOut);
```

- **addressIn**: the address of the component in the platform
- **byteSelIn**: in which byte the component has to provide valid data (considering a 4-byte word)
- **dataOut**: the data returned by the addressed component
- **errorOut**: 1 if the write transaction fails, 0 otherwise

# write\_transaction

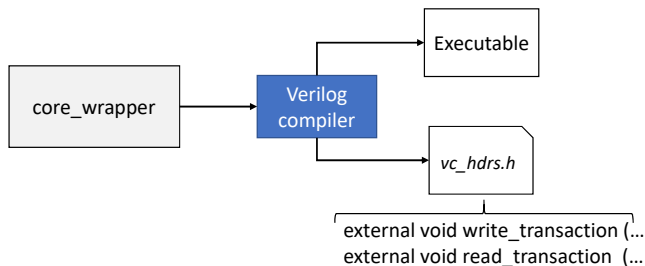
The write\_transaction provides the writing functionality.

```
task write_transaction (input int addressIn, input int dataIn, input int  
byteSellIn, output byte errorOut);
```

- **addressIn**: the address of the component
- **dataIn**: the data sent to the addressed component
- **byteSellIn**: which byte of dataIn is valid
- **errorOut**: 1 if the read transaction fails, 0 otherwise

# Exporting Verilog tasks as C methods

Both Verilog tasks are exported as **DPI-C** task. A header file (*vc\_hdrs.h*) including a C method declaration of *write\_transaction* and *read\_transaction* is generated by the Verilog compiler.



# Read/write a memory-mapped component

The same firmware running in a CPU can be executed in the *simple-platform* by replacing any operation involving a memory-mapped component with a C-method call, which corresponds to a Verilog task invocation.

```
...  
int *camellia_adr = 0x12341234;  
*camellia_adr = 5;  
int res = *(camellia_adr + 1);  
...
```

original firmware



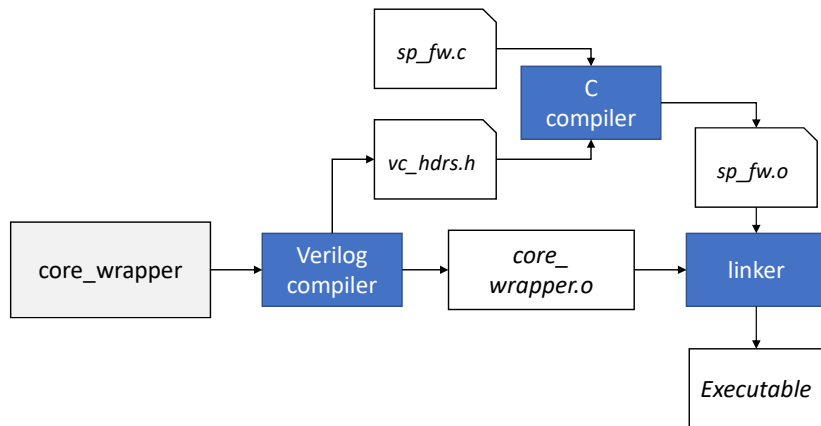
```
...  
write_transaction(0x12341234, 5, ...);  
int res;  
read_transaction(0x12341238, &res, ...);  
...
```

Firmware executed in the  
*simple-platform*



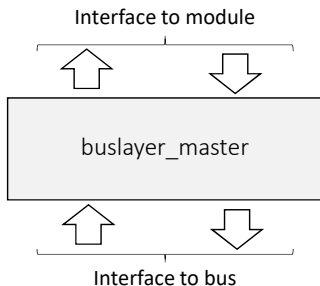
# Combining C and Verilog code

Overview of the C-Verilog compilation flow in the *simple-platform*



# buslayer\_master

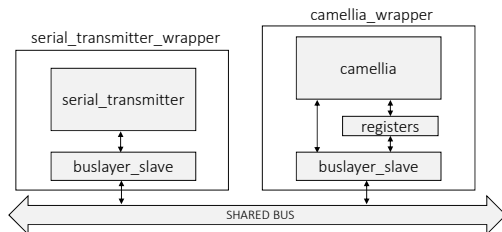
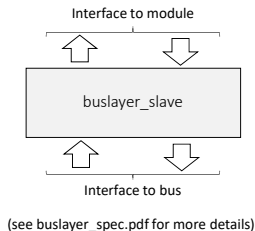
The buslayer\_master is the component in charge of forwarding any read/write request coming from the firmware to the shared bus.  
Any transaction follows the wishbone protocol rules in the shared bus.



(see buslayer\_spec.pdf for more details)

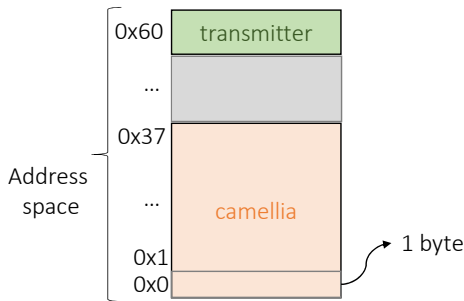
# buslayer\_slave

The `buslayer_slave` is the component in charge of forwarding any read/write request coming from the shared bus to a component. The `camellia_wrapper` and the `serial_transmitter_wrapper` use a `buslayer_slave` to forward read/write transactions from the shared bus to registers and ports.



# Memory-mapped components

The *simple-platform* has a cipher (camellia) and a serial-transmitter. The two components are memory mapped according to the following address-space map:



The input port of the serial-transmitter is memory-mapped at the write-only address 0x60.

# Memory-mapped Camellia

The input/output ports of Camellia are memory-mapped according to the address-space map below reported. The memory addresses from 0x0 to 0x23 can be read and written. Meanwhile, the addresses from 0x24 to 0x37 are read-only. The memory space of Camellia has granularity a word.

