VIRTUAL PLATFORM DESIGN WITH HIFSUITE

INTEGRATING REUSED HDL MODELS INTO A VIRTUAL PLAFORM

The adoption of a virtual platform when designing complex embedded systems consists of modeling the whole system as integration of components described at high level of abstraction, using suitable languages like C++ and SystemC TLM.

However, designers usually do not have the description at high level of abstraction of the components making up the virtual platform. HIFSuite tools assist designers to obtain such descriptions automatically from existing models written in traditional HDL languages such as VHDL and Verilog.

The shift to a higher abstraction level allows to simplify the communication protocol of the components and to wrap multiple clock cycles into a single high-level transaction. HIFSuite can automatically perform such operation according to a simple user-provided configuration file, which summarizes the communication protocol to be abstracted.

Once the high-level descriptions of the components have been obtained, they have to be integrated into the virtual platform. In case a corresponding top-level description is not available in VHDL and Verilog, HIFSuite can assist designers in manually integrating the high-level descriptions into the virtual platform.

This whitepaper gives an overview of how HIFSuite can be used to obtain high-level descriptions of existing components, where their original communication protocol can be abstracted into a single high-level transaction. Additionally, this whitepaper shows how manual integration of such high-level descriptions can be facilitated by HIFSuite.

CONTENTS

HIFSuite

SERVICES AND TOOLS FOR VIRTUAL PLATFORMS

HIFSuite is a set of tools to manipulate HDL modules. They are developed and maintained by EDALab, a young Italian company committed in research and development of software for embedded systems.

More info at:
www.hifsuite.com

EDALAB Networked Embedded Systems

www.edalab.it

SEPTEMBER 2015

GENERATING HIGH-LEVEL DESCRIPTIONS

HIFSuite allows to automatically generate high-level descriptions of existing RTL models written in VHDL and Verilog. Please refer to the “Speed up simulations with HIFSuite” white paper available in the Publications section of the HIFSuite website (http://www.hifsuite.com/index.php/publications) for more details about this process.

For the remainder of the paper, we shall refer to Steps 3 and 4 of the aforementioned white paper for the generation of TLM descriptions, and to Steps 5 and 6 for the generation of C++ descriptions. For the reader’s convenience, we provide a visual overview of such steps in Figures 1 and 2.

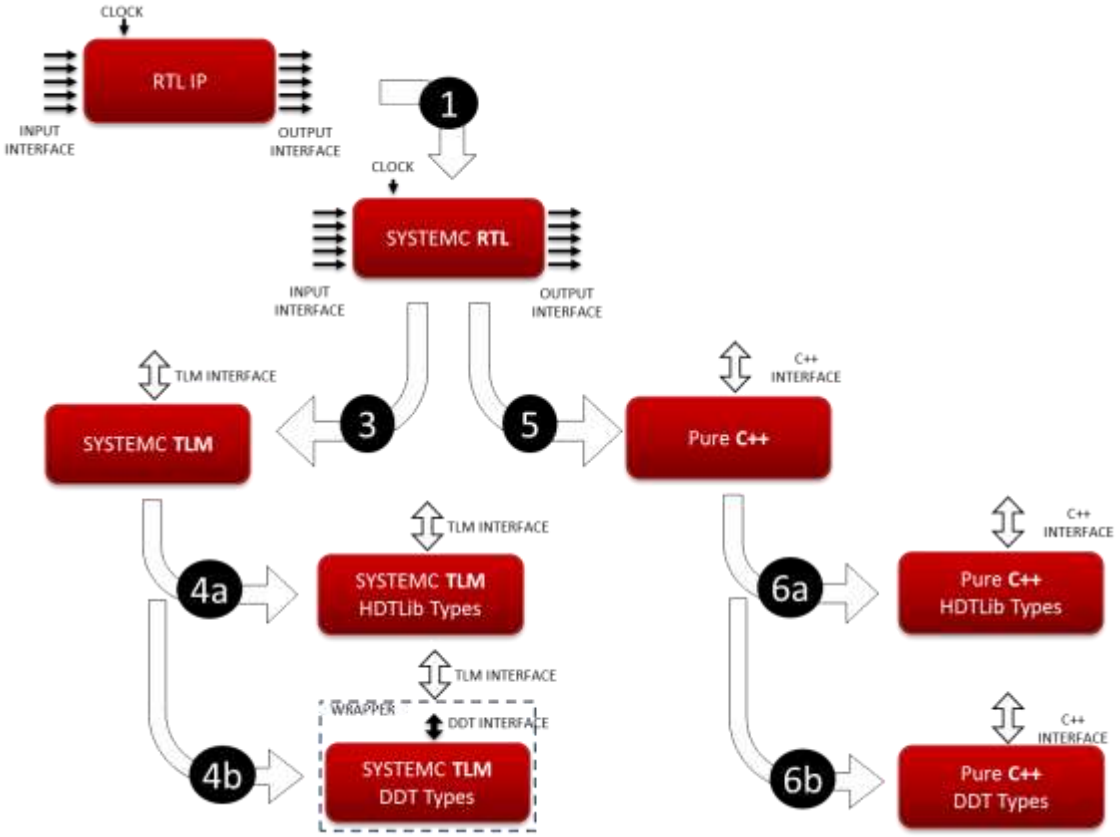

FIGURE 1 – DIAGRAM OF STEPS 3, 4, 5 AND 6

COMMUNICATION PROTOCOL ABSTRACTION

When generating high-level descriptions, HIFSuite is capable of abstracting the original communication protocol of RTL models, so that the elaboration of the design functionality is wrapped up into a single transaction. This allows to compact the execution of a number of multiple clock cycles into a single function call. Additionally, this abstraction process relieves the user from having to worry about the low-level details of the original communication protocol (e.g., setting handshaking signals and timing of signal writes).

HIFSuite is capable of automatically performing the abstraction of the communication protocol according to a simple user-provided configuration file, which briefly summarizes the original communication protocol of the design.

Figure 2 shows an example of the abstraction of the communication protocol operated by HIFSuite. The reference design is the Camellia Verilog cryptographic core from Aoki Laboratory of Tohoku University (http://www.aoki.ecei.tohoku.ac.jp/crypto/).

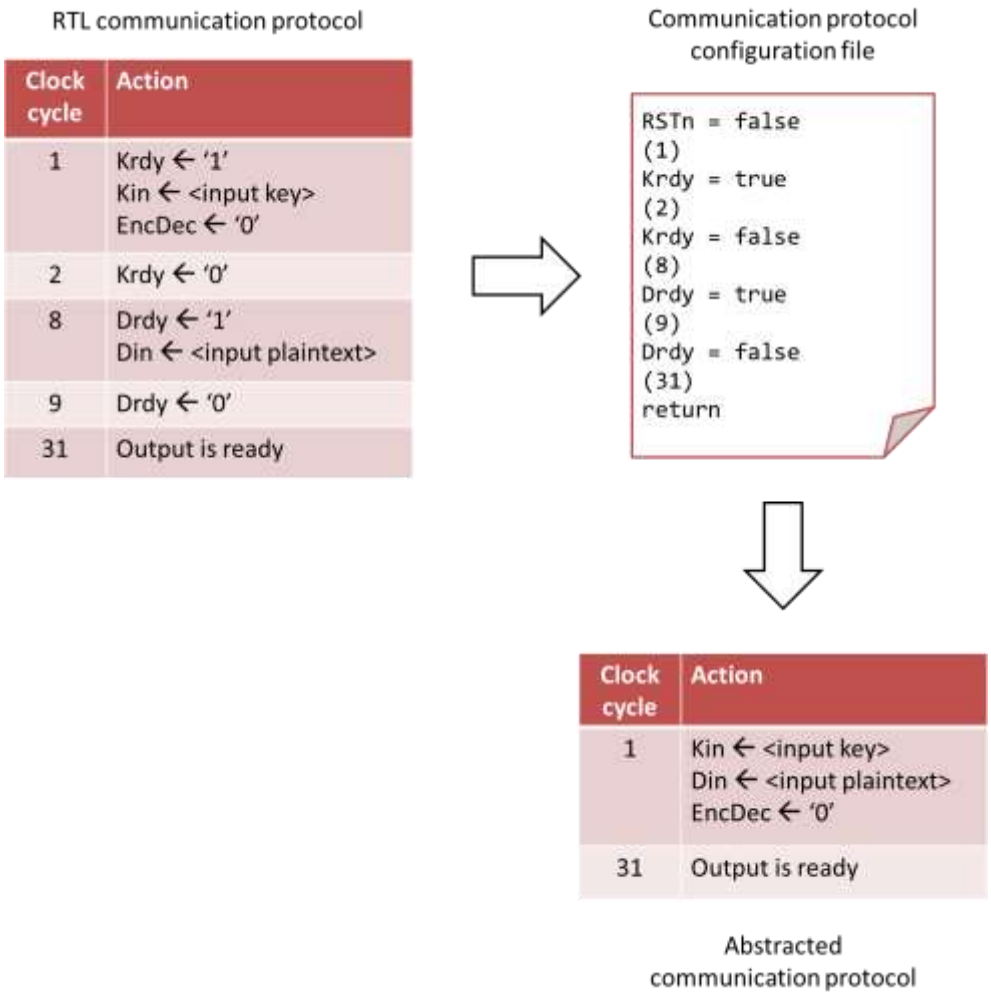

FIGURE 2 – EXAMPLE OF COMMUNICATION PROTOCOL ABSTRACTION

The RTL communication protocol requires to set *Krdy* to '1' and to provide the input key in the first clock cycle. Additionally, *EncDec* must be set to '0' to indicate that the device must operate in encoding mode (instead of decoding). In the next clock cycle, *Krdy* is driven to '0' to indicate that the input key has been provided in the previous cycle. After six clock cycles, the protocol requires to set *Drdy* to '1' and to provide the input plain text to be encoded. In the following cycle, *Drdy* must be set to '0' to indicate that the input plain text has been provided. Finally, the output ciphertext is available after 22 clock cycles.

Such information about the RTL communication protocol is required to properly perform its abstraction. As such, the user must provide a simple configuration file which stores timing and data details of the protocol in a compact way. The configuration file starts by indicating the name of the reset signal and the corresponding active value. Then, it lists the sequence of writes on handshaking signals according to the clock cycle in which they are expected to occur. Finally, the configuration file indicates that elaboration completes after 31 clock cycles.

The high-level description generated by HIFSuite will abstract the communication protocol so that the user is required to provide only the input data, without having to worry about setting handshake signals *Krdy* and *Drdy* and about the timing points requested by the original protocol. Then, the design

functionality is carried out by executing a TLM transaction (in case of a TLM description) or a procedure call (in case of a C++ description). The output ciphertext (or plaintext) will be available after returning from the transaction (or the procedure call), the output ciphertext (or plaintext) will be available.

PROTOCOLS WITH PIPELINED INPUTS/OUTPUTS

HIFSuite is capable of abstracting more complex communication protocols, featuring pipelined inputs and/or outputs. An example of such protocol is illustrated in Figure 3. The reference design in this case is the SHA-512 Verilog cryptographic core from OpenCores (http://opencores.org/project,sha_core).

RTL communication protocol

Clock cycle	Action
2	cmd_i ← 0xa cmd_w_i ← '1'
3	cmd_w_i ← '0' text_i ← <input_block_0>
4	text_i ← <input_block_1>
...	...
34	text_i ← <input_block_31>
102	cmd_i ← 0xe cmd_w_i ← '1'
103	cmd_w_i ← '0' text_i ← <input_block_32>
104	text_i ← <input_block_33>
...	...
134	text_i ← <input_block_63>
201	cmd_i ← 0x9 cmd_w_i ← '1'
202	cmd_w_i ← '0'
203	<output_block_0> is ready
...	...
218	<output_block_15> is ready

Communication protocol configuration file

```
rst_i = true
text_i[64]
text_o[16]
(2)
cmd_i = 0xa
cmd_w_i = true
(3)
cmd_w_i = false
[R] text_i[0]
(4-34)
[R] text_i[-3]
(102)
cmd_i = 0xe
cmd_w_i = true
(103)
cmd_w_i = false
(104-134)
[R] text_i[-71]
(201)
cmd_i = 0x9
cmd_w_i = true
(202)
cmd_w_i = false
(203-217)
[W] text_o[-203]
(218)
[W] text_o[15]
return
```

Clock cycle	Action
1	text_i[0] ← <input_block_0> ... text_i[63] ← <input_block_63>
218	<output_block_0> is ready ... <output_block_15> is ready

Abstracted communication protocol

FIGURE 3 – EXAMPLE OF ABSTRACTION OF A COMMUNICATION PROTOCOL WITH PIPELINED I/O

The RTL communication protocol requires the input text to be subdivided into 64 32-bits blocks. Such blocks are provided as input during clock cycles ranging from 3 to 34 and from 103 to 134, one block per clock cycle. Similarly, the

output hash value is decomposed into 16 32-bits blocks, which are provided one at a time during clock cycles ranging from 203 to 218.

The high-level description generated by HIFSuite will generate arrays corresponding to pipelined input and output ports. In this case, the communication protocol configuration file is enriched with array declaration lines which specify the name of the pipelined ports (*text_i* and *text_o*) and the length of the corresponding array (64 and 16, respectively). Then, the configuration file specifies the range of clock cycles in which the pipelined input port *text_i* is read, together with the negative offset to be added to the index of the current clock cycle in order to obtain the index of the array to be read. Similarly, the configuration file indicates the range of clock cycles in which the pipelined output port *text_o* is written. Finally, the configuration file indicates that elaboration completes after 218 clock cycles.

LIMITATIONS

At the moment, HIFSuite is not capable of abstracting communication protocols featuring variable timing points, i.e., where the clock cycle in which a given signal write must occur is not fixed, but depends on other inputs or on the configuration of the design itself. Furthermore, HIFSuite is not capable of abstracting communication protocols featuring multiple operating modes where the sequence of writes on handshaking signals is different according to the operating mode itself.

In such cases, HIFSuite will generate a high-level description where the original RTL communication protocol is preserved. The abstracted description will thus retain a cycle-accurate communication protocol, where a RTL clock cycle corresponds to a transaction (or a procedure call).

INTEGRATION INTO VIRTUAL PLATFORM

After the high-level descriptions of components have been generated, they usually have to be integrated into a virtual platform. According to the timing accuracy of the virtual platform and the interactions between its components, three different integration scenarios can be identified, as Figure 3 shows.

If the virtual platform is loosely-timed, a high-level description featuring an abstracted communication protocol can be directly plugged into the platform (scenario 1). In this way, such description will be allowed to run ahead of simulation time, in a manner similar to temporal decoupling in SystemC TLM-2.0. In fact, the whole elaboration of the design functionality will be executed in a local time warp, without actually advancing simulation time. Then, control is passed back to the caller (i.e., the main elaboration function of the whole virtual platform), to allow for synchronization with the rest of the system. This allows to greatly speed up simulation, at the expense of timing accuracy.

However, this approach cannot be employed if the high-level description requires to interact with other components during the elaboration of its functionality, i.e., in timing points differing from the call and the return from the elaboration procedure. In such cases, the communication protocol cannot be abstracted, and the original cycle-accurate communication protocol must be retained (scenarios 2 and 3).

In these scenarios, high-level descriptions can be generated so that they expose two procedures into which the elaboration of the design functionality is split. One procedure performs the elaboration of the synchronous processes, while the other one carries out the elaboration of the asynchronous processes.

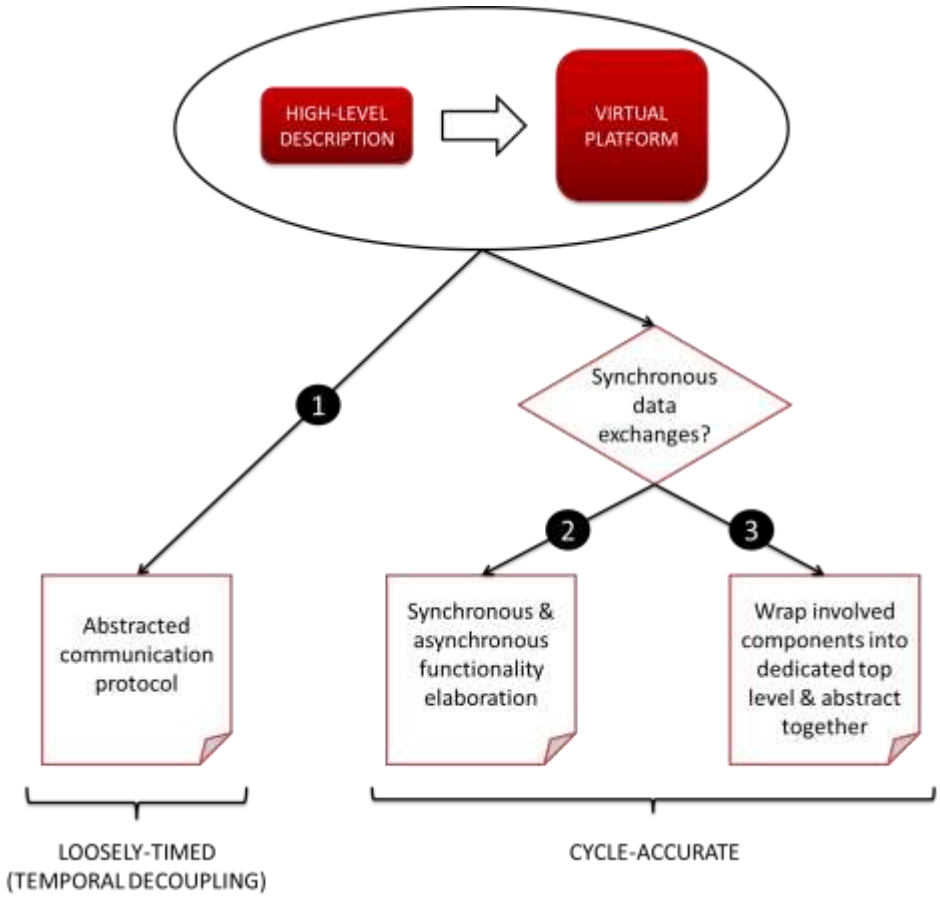

FIGURE 4 – SCENARIOS FOR INTEGRATION INTO VIRTUAL PLATFORM

Manual integration into a virtual platform can be then carried out by invoking these two procedures accordingly in the top-level main elaboration procedure. As long as data exchanges between the components happen synchronously, the order with which elaboration procedures of the different components are invoked does not affect the evolution of the whole platform (scenario 2). Conversely, if two or more components exchange data asynchronously, then the involved components should be wrapped in a dedicated top level and abstracted together, in order to ensure that proper synchronization is achieved in the virtual platform (scenario 3).