Toolset for SystemC Code Generation of Heterogeneous Platforms

Pavel Ivanov, Evgeny Gavrin
Intel Labs
Saint-Petersburg, Russia
{Pavel.Ivanov, Evgeny.Gavrin}@intel.com

Abstract

This paper considers the toolset for code generation of system model of the platform and automatic code generation test bench for generated components of the platform. Description of platform architecture as input data for tools and a described list of possible component typesare provided. The work-flow description, SystemC platform opportunities and SystemC test bench code generator are demonstrated.

Index Terms: SoC, System level design, SystemC, Code generation.

I. Introduction

Designing of Systems-on-Chip (SoC) is one of the main fields in the development of modern electronics. The modern design of SoC is a multi-dimensional problem. Development of hardware and software, in general, has the following flow:

1) The conceptual design of the system (description of the whole system);
2) Development of a model system-level (SystemC / System Verilog);
3) Development of register transfer level RTL (Verilog / VHDL);
4) Hardware verification;
5) Physical prototyping;
6) Software development;
   a) Drivers development;
   b) Embedded software development;
1) Software testing;
2) Testing of entire system.

By increasing the functionality of the systems (functionality increases with the quantity of gates on a chip and with the functionality of executing software), the time to develop the system grows at all stages of design. As a consequence, the cost of the project increases. The diagram of relation of time costs from technology process (introduced by Cadence Design Systems [1]) is shown in Fig.1.

Hardware and software system-level verification, i.e. the presence of a system model, by means of early functional testing allows to save time and costs of development. The system model includes a specification of all system functions and implementation of an executable system model (HW / SW). This model facilitates the next stage of development, because a specification for the development of RTL model and software developing for the system can be taken from it. To reduce the system development time costs, namely the cost of development of system-level models, the author proposes an approach of development a system model by using tools called SystemC platform code generator and SystemC test bench code generator that automatically generate the code of system-level models based on

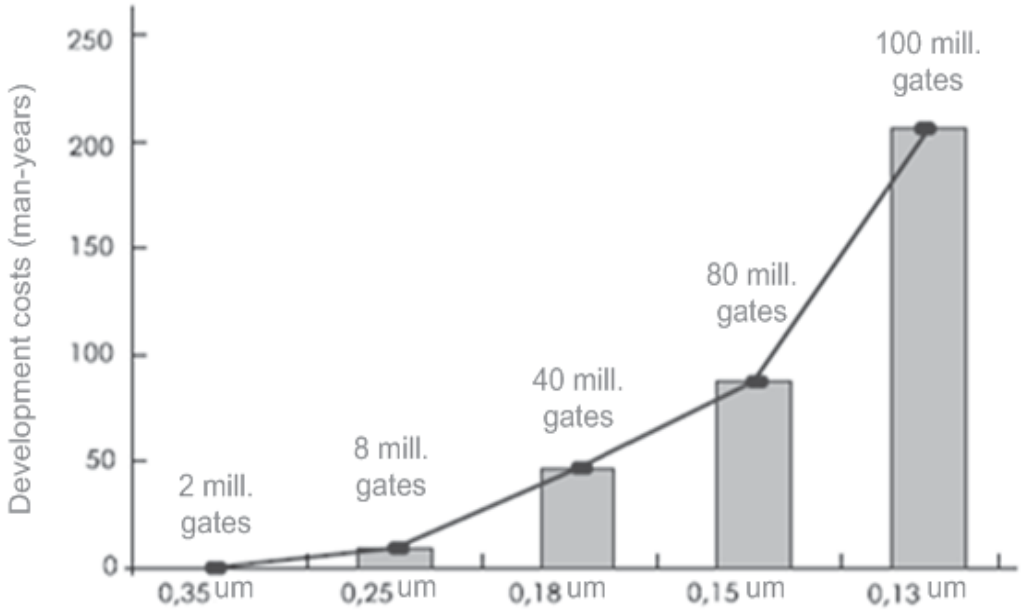

Fig. 1. The diagram of relation of time costs from technology process

the input data (description of the platform architecture and properties of architectural components).

II. Main part

A. Description of input data. Architecture component types

As an input for the SystemC platform code generator and for SystemC test bench code generator uses the intermediate representation of a computer program (SW IR - software intermediate representation) and intermediate representation of a hardware platform (HW IR - hardware intermediate representation), in the form of XML-file. The topology of the hardware platform can be different. The generator does not check the correctness of the connections between the blocks. As well, the generator does not check the correctness of computer programs.

As an example, the intermediate representation of computer software and hardware platform, we assume that we have a full mesh platform with network topology, where each processing element can directly accept and receive data from any other processing element. The example at Fig. 2 illustrates the description of computer program (the program graph) that contains all the necessary data.

The graph of the program contains information about the name of the process, its unique identification number, the type of process, the run-time and the name of the custom function as an optional parameter. Also, it contains the relationship between processes, the maximum amount of data transmitted and the quantity of ports.

Fig. 3 shows the description of the hardware platform, which has the data about the type of processing elements, maximum bandwidth and relations between processing elements. Also, at Fig. 4 presented graphical representation of hardware platform.

```
- <program comment="" id="p0" mainSection="i0">
  - <section comment="" id="i0">
    - <nodes>
      - <Functional comment="Input operator" id="i10027" isPermanent="false" function="">
          <port link="i10042" portNum="10" id="i10033" />
        </Functional>
      - <Functional comment="Final process & output" id="i10044" isPermanent="false"
          function="">
          <port link="i10043" portNum="4" id="i10047" />
        </Functional>
      - <Functional comment="Initial proces-sing" id="i10059" isPermanent="false" function="">
          <port link="i10042" portNum="4" id="i10062" />
          <port link="i10043" portNum="10" id="i10065" />
        </Functional>
      </nodes>
    - <links>
        <link comment="" source="i10027" type="read-erase" id="i10042" target="i10059" />
        <link comment="" source="i10059" type="read-erase" id="i10043" target="i10044" />
      </links>
    </section>
  </program>
```

Fig. 2. Program graph file example

```
<?xml version="1.0" encoding="UTF-8" ?>
- <platform name="Test platfrom" dataRate="1">
    <pe name="pe00" type="peMain" />
    <pe name="pe01" type="peMain" />
    <pe name="pe10" type="peMain" />
    <pe name="pe11" type="peMain" />
  </platform>
```

Fig. 3. Hardware description file example

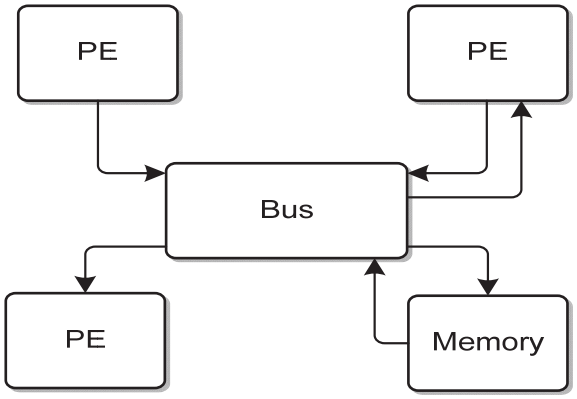

Fig. 4. Hardware platform graphical representation

In the process of the construction of the hardware platform can be used the following types of components:

1) Processing element;
2) Control unit;
3) Memory;
4) Bus;
5) Switch.

To connect to the above components channels that are connected to the ports of base components are used. All blocks are parameterized: clocked / not clocked, bandwidth, buffer size on the port, the duration of the function on the block, etc. Using a set of these components various architectural solutions can be described: microprocessors, DSP, FPGA chips, etc. Schematic representation of the input data stream is shown in Fig. 5.

Files of SW IR and HW IR go to the input of mapper. Mapper is responsible for two functions:

1) Scheduling;
2) Allocation.

Scheduler maps the start order to each component during the execution of a computing program on the platform. Allocation functions are mapped to each hardware component, which will be performed on each of them. A platform architecture usually has a control unit, which gives the start and stop commands to the processor elements. Control code for the control unit is also compiled by a mapper.

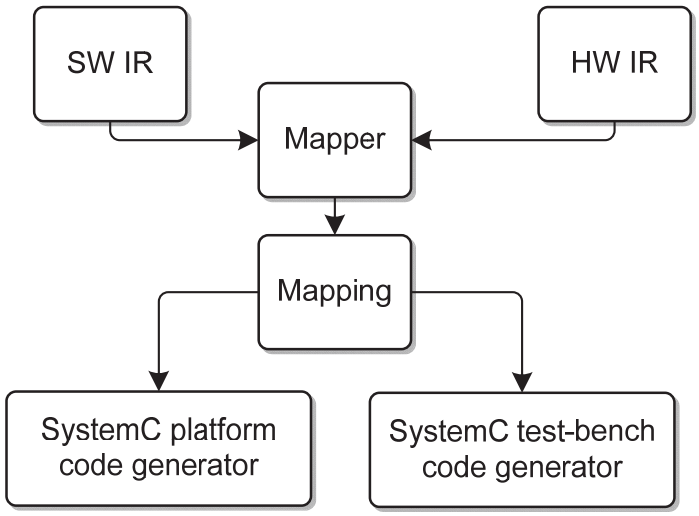

Fig. 5. Input data flow for the SystemC platform code generator and SystemC test bench code generator

B. SystemC platform code generator description

After mapping SW IR and HW IR at each other, based architecture goes to the input of the SystemC platform code generator. Before starting, a user must configure the types of logs in the tool which he wants to see after or during code generation, simulation time and timing unit (ns, us, etc.), take folder path, where there are the Reference Lib and components templates. The process is shown schematically in the Fig. 6.

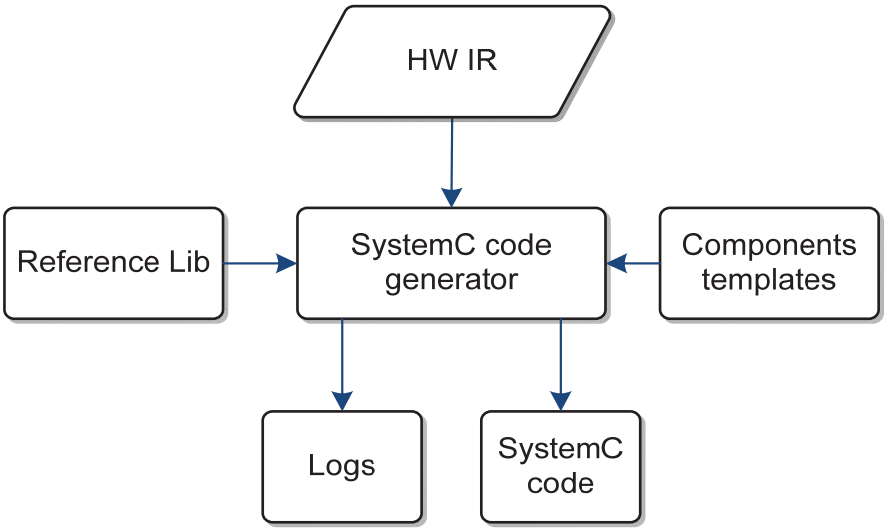

Fig. 6. The work process of SystemC platform code generator

SystemC platform code generator supports the following types of logs:

1) Text log;
2) Console log;
3) VCD file (waveforms).

During execution, the code generator, the element by element, begins to generate SystemC code for each component. At the start of the next code generation component, SystemC platform code generator selects the template for the type of the generated element. To generate the architectural elements described in Chapter II.A, it is enough to use three templates:

1) Processing element template;
2) Interconnect template;
3) Memory template.

For example, Fig. 7 illustrates a template for processing element generating. The template contains tags that are enclosed in "%tag name%" to easily find a place to insert the generated code, the correct order of ports declarations, function and design. All files with the description of the components, at the end of generation are named [component_name.h].

Process of code generation consists of the following stages:

1) Content of the template is copied to the new file;
2) Inserting the name of the generated component;
3) Ports declaration;
4) Declaration of local memory or flags (if any);
5) Executable function of component is taken from Reference Lib;
6) The name of the function, which is executed on the component, is substituted to the constructor.

After completion of the generation of all the elements of architecture, a file with the entry point of the simulator is generated. There, on the basis of input data, generated blocks are initialized and bound.

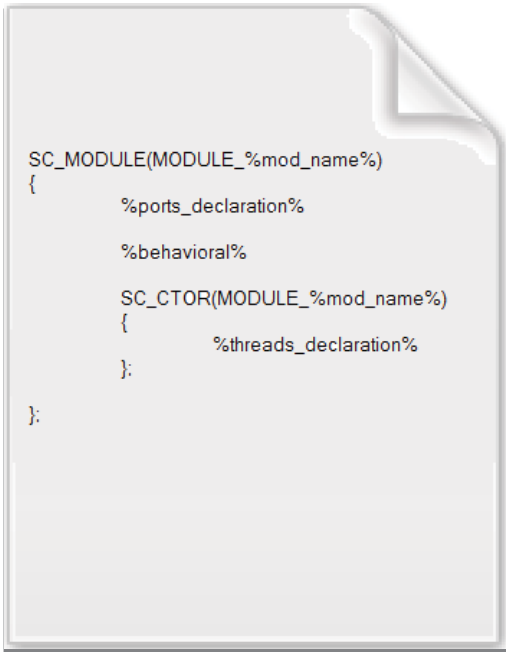

Fig. 7. Example of processing element file template

C. SystemC test bench code generator description

As well as the SystemC platform code generator, SystemC test-bench generator receives a description of the platform architecture as the input, which is obtained from the process of mapping SW IR and HW IR. Before running the SystemC test bench code generator one configures the types of logs for viewing the results and the process of testing and a timing unit. Types of logs are the same as in SystemC platform code generator. Fig. 8 schematically illustrates the flow of input and output data for SystemC test bench generator.

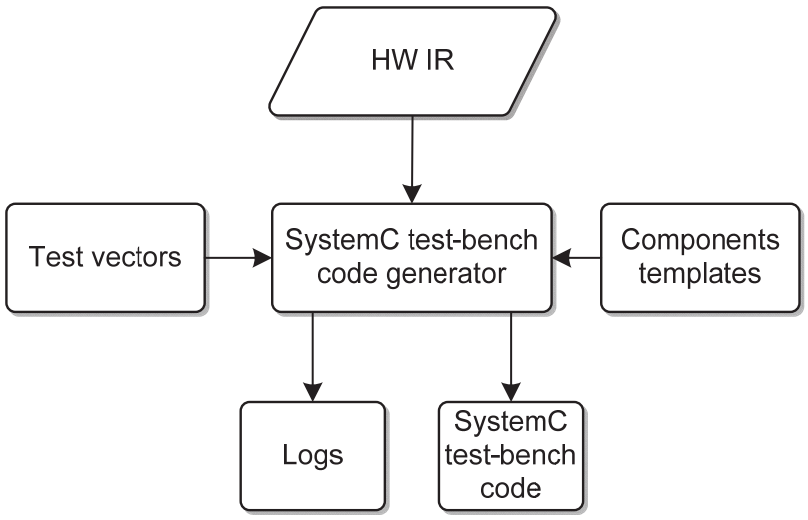

Fig. 8. Work process of SystemC test bench code generator

Functionally, SystemC test bench generator has two modes of testing:

1) Testing each component separately;
2) Testing of the entire system.

As of writing of this paper, the test mode of the entire system has not been developed. In the test mode of each separate component, for each test component of the input platform its own test bench is generated. The components of the test environment, as well as in the SystemC platform code generator, are generated with the help of templates. For testing of any architectural component there is a set of test vectors (i.e. functions for components of the test environment), that are used during code generation. Test vectors have the following scenarios:

1) Testing of processing element or memory. The component that is connected directly to a testing component is generated (Fig. 9).

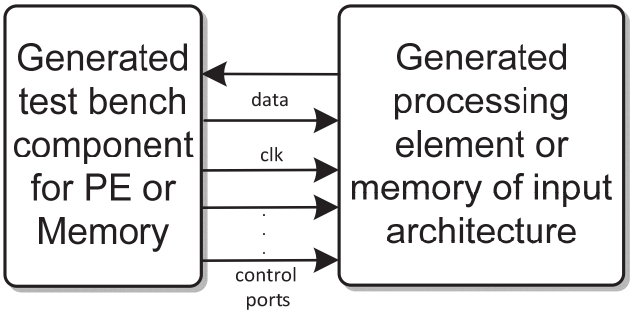

Fig. 9. Wiring diagram of testing and tested components

The correctness of the data after the transmission to a tested component or after receiving the data from it is tested. The correctness of the transmitted data delivery time is checked (considering temporal characteristics of the tested components.) When testing the memory, the correctness of work is tested if the combinations of control signals are wrong;

2) Testing of interconnect. The components in an amount equal to the number of input and output ports of the component under test architecture are generated. Generated components from the test environment have their own functions (transmitter or receiver). The example of test component “bus”, which transfers data with “packet” structure (Fig. 10).

SystemC test bench code generator creates a table of input and output ports. Test vector tests the bus on data transferring from each input port to each output port. For each interconnect component the correctness of input data on the receiver from transmitter via bus is checked. The delivery time of the data, taking into account the time characteristics of interconnector, is checked. If any port has a buffer, its capacity and feedback with data transmitter (for example, buffer overflow) is checked.

For automatic compilation of test bench application files and its execution, it generates *bat file that will be called from SystemC test bench generator. One test application is the process of checking of a single component. Test is not interrupted if there is an error at any stage of testing.

At the completion of the test environment generation for all of the test components, SystemC test bench code generator generates an entry point of test application for each of the tested components.

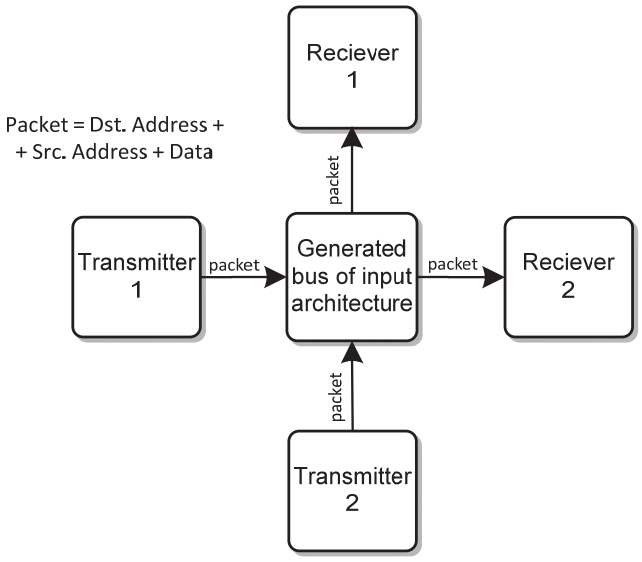

Fig. 10. Wiring diagram of testing transceivers with the bus component under test

SystemC test bench code generator creates a table of input and output ports. Test vector tests the bus on data transferring from each input port to each output port. For each interconnect component the correctness of input data on the receiver from transmitter via bus is checked. The delivery time of the data, taking into account the time characteristics of interconnector, is checked. If any port has a buffer, its capacity and feedback with data transmitter (for example, buffer overflow) is checked.

For automatic compilation of test bench application files and its execution, it generates *bat file that will be called from SystemC test bench generator. One test application is the process of checking of a single component. Test is not interrupted if there is an error at any stage of testing.

At the completion of the test environment generation for all of the test components, SystemC test bench code generator generates an entry point of test application for each of the tested components.

III. Conclusion

In this paper two tools were presented: SystemC platform code generator and SystemC test bench code generator. This toolset is developed to simplify the process of development of a system level model. The existing versions of SystemC platform code generator and SystemC test bench code generator already have a good functionality of log service. Reference lib does not limit functionality of components, which were generated by SystemC platform code generator. The use of these tools can greatly reduce the time of system level model development.

References

[1] A.A. Ivanov, A.P. Rizhov, “Development SoC by Cadence tools”, Cadence Design Systems pp. 64, 2003.
[2] E.A. Gavrin, “Software simulator of parallel computing communicated via messages,” Bachelor thesis, pp 28-29,2009.
[3] D.C. Black, “*SystemC: From the Ground Up*” second edition, Springer 2010.
[4] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew McNelly, Lee Todd, “*Surviving the SoC revolution*”, Kluwer Academic publishers, 1999.