FPGA virtual platform based on systemc and verilog

Chunwei Song¹, Xianghu Wu² and Yongchao Tao³
¹Shenzhen Academy of Aerospace Technology, Shenzhen, China
²Shenzhen Academy of Aerospace Technology, Shenzhen, China
³Shenzhen Academy of Aerospace Technology, Shenzhen, China

*Corresponding author e-mail: scw13473519956@163.com

Abstract. With the development of science and technology, FPGA is widely used in many fields of society, such as communication, algorithm implementation, and embedded field, with its flexible programming, high-performance parallel computing capabilities, and short development cycles. There are already many types of FPGA hardware products on the market, but the tedious problems of the hardware system itself and the cost problems of the hardware system bring a lot of inconvenience to users' learning, which is inconvenient for users to learn, develop, and debug FPGA jobs. This article is based on such a problem. From the software level, a scheme for implementing an FPGA virtual platform based on systemc and verilog is proposed. This solution uses software to simulate the implementation of FPGA hardware, TCP communication technology, systemc and verilog applications to meet the needs of users to learn, develop and debug FPGA without a hardware platform.

1. Introduction

FPGA (Field-Programmable Gate Array) is also field programmable gate array. FPGA is a product developed on the basis of GAL, PAL, CPLD and other programmable devices. It appears as a semi-custom circuit in the field of integrated circuits (ASIC). It not only addresses the shortcomings of custom circuits, but also overcomes them. The disadvantage of the limited number of gate circuits of the original editable device is eliminated. From the perspective of the chip, FPGA is a programmable digital logic chip. We can program the FPGA through HDL (Hardware Description Language) language to achieve the digital functions we want. In short, FPGAs can change their internal structure through software programming.

Verilog is the abbreviation of Verilog HDL. Verilog HDL is a hardware description language (HDL: Hardware Description Language). The hardware description language is a language for describing the hardware behavior, structure, and data flow of electronic systems. Using this language, the designer of digital circuit systems can describe their design ideas layer by layer from the top to the bottom (from abstract to concrete), and use a series of hierarchical modules to represent extremely complex digital systems. There are currently two popular HDL languages: VHDL and verilog. VHDL developed earlier, with strict grammar, and was difficult to learn. Verilog is similar to the C language syntax style, easy to learn and master, suitable for beginners to FPGA.

SystemC is a software / hardware cooperative, system-level modeling and design language. In simple terms, systemC is a C++ class library developed using the C++ programming language and provides an event-driven Simulation Kernel to schedule the execution order of various processes. SystemC's scheduling algorithm idea is to divide continuous simulation time into multiple discrete simulation mom
ents, and then divide a simulation moment into multiple delta-cycles, and update the channel value after a delta-cycle or multiple delta-cycle cycles (sistema's channel design is very clever, similar to the role of shadow variables), so you can use sequential execution programming language in these delta-cycles to simulate the parallel behavior of hardware. It enables designers of simulation systems to use C++ syntax rules to simulate parallel processes, especially in SoC systems.

There are already many types of FPGA hardware products on the market, but the tedious problems of the hardware system itself and the cost problems of the hardware system bring a lot of inconvenience to users, which is inconvenient for users to learn, develop, and debug FPGA jobs.

Products with an FPGA virtual platform can replace the actual hardware platform, and need to provide custom components of a certain size, such as FPGA core components, FPGA resource components, TCP server components, verilog adapter components, and other commonly used equipment components. On this basis, the product can automatically generate an FPGA virtual platform containing verilog functional components on the premise that users independently design verilog functional components. This platform is convenient for users to learn, develop and debug FPGA. Using this product, users not only reduce the learning and development costs, but also shorten the FPGA development cycle.

Este producto está desarrollado en base al lenguaje de diseño systemc y verilog. Los usuarios pueden usar este producto para desarrollar y depurar el lenguaje HDL (verilog) y aprender a dominar el proceso de desarrollo FPGA. Hoy en día, la popularidad de los FPGA en los colegios y universidades es cada vez mayor. En la enseñanza tradicional, los recursos de hardware se desperdiciarán y los inconvenientes de llevarlos inevitablemente se causarán. La operación engorrosa del hardware dificultará el uso de los estudiantes. Disminución del deseo de los estudiantes de aprender y explorar. Este producto es más conveniente de usar y es muy adecuado para la enseñanza en colegios y universidades.

2. System framework

This software system is mainly used to simulate an FPGA. Its network topology is shown in Figure 1. The entire software structure includes several constituent components: TCP server component, verilog adapter component, FPGA core component, FPGA resource component, FPGA debug component, verilog component.

![FPGA virtual platform architecture](image-url)
2.1. Component introduction

The following is an introduction to the functions of each component of the system.

2.1.1. TCP server component. This component provides the function of interaction between the FPGA virtual platform and the outside world. This component has the following functions: parsing or encapsulating TCP packets, calling the verilog adapter component to convert the verilog interface component into a systemc interface component, calling the FPGA debugging component to set the FPGA excitation signal or viewing the waveform, and calling the FPGA core component to obtain the FPGA status value.

2.1.2. verilog adapter component. This component analyzes the user's verilog component configuration file, generates a verilog interface adaptation file, and finally converts the verilog interface component into a systemc interface component.

2.1.3. FPGA core component. This component initializes the FPGA virtual peripheral resources, provides the FPGA virtual core clock, implements the FPGA pin mapping function, and encapsulates the FPGA status value and sends it to the TCP server component.

2.1.4. FPGA resource component. This component defines the virtual peripheral functions of the FPGA virtual platform, including FPGA virtual pins (led peripherals, SEG peripherals, RGB peripherals, DIP switch peripherals, key peripherals and other resources), virtual pull-up or pull-down resistor.

2.1.5. FPGA debugging component. This component is mainly used for user debugging, including parsing or packaging TCP data packets, setting FPGA excitation signals, and setting waveform tracking.

2.1.6. Verilog component. The verilog interface component provided by the user is both a part of the FPGA virtual platform and a user code block being debugged.

3. Software implementation

To build an FPGA virtual platform based on systemc and verilog, you need to complete the following steps:

First, implement verilog functional components and use physical hardware to verify the verilog component's correctness. If the verilog component is correct, proceed to the next step.

Second, implement systemc components, including FPGA core components, FPGA resource components, tcp server components, verilog adapter components, and FPGA debugging components.

After that, verify the correctness of systemc components using verified verilog components. The functions that FPGA core components need to verify include initialization of FPGA virtual peripheral resources, FPGA virtual core clock, FPGA pin mapping function, and FPGA status values sent to the TCP server. Components; the functions that FPGA resource components need to verify include the definition of FPGA virtual pins, the implementation of virtual pull-up or pull-down resistors; the functions that need to be verified by TCP server components include the interaction function between the FPGA virtual platform and the outside world; the functions that need to be verified by FPGA debugging components contain the processing of TCP data packets, the setting of FPGA excitation signals, and waveform tracing; the functions that verilog adapter components need to verify include the analysis of the user's verilog component configuration files, the generation of verilog interface adaptation files, and the conversion of verilog components into systemc interface components. If all functions of the systemc components are correct, proceed to the next step.

Finally, the verified systemc component and the verilog component provided by the user are used to build the FPGA virtual platform. The user uses this virtual platform to learn, develop, and debug the FPGA.

The various functions of the FPGA virtual platform are mentioned above, and the interface specifications are summarized as shown in Table 1, Table 2, Table 3, and Table 4.
Table 1. TCP server component interface specifications.

| Function interface definition                      | Function interface description                                                                 |
|----------------------------------------------------|------------------------------------------------------------------------------------------------|
| void host_thread(void)                             | Create TCP communication thread.                                                               |
| void process_recv_data(char *buffer, int count)    | TCP data receiving function, wakes the sc_thread coroutine, and forwards the data to sc_thread.|
| virtual void sc_thread(void)                       | Systemc coroutine, responsible for the TCP data forwarding function, forwards the data to the deal_ json_packet function for processing. |
| void deal_json_packet(Json::Value &json_obj)       | Transform the received json data into a json object and process it according to the "type" attribute.|
| void fpga_json_decode(Json::Value &fpga_root)      | Parse the json data received by tcp, and execute different branch codes based on the data ID.    |
| uint32_t json_len(const char* json_buffer)          | Calculate the length of TCP sent data.                                                        |
| int send_data(const char *buffer, int count)        | Send the data through TCP.                                                                     |

Table 2. Verilog adapter component interface specifications.

| Function interface definition                      | Function interface description                                                                 |
|----------------------------------------------------|------------------------------------------------------------------------------------------------|
| void fpga_pinmap_decode(Json::Value &fpga)         | Parse JSON data packet and create verilog adaptation layer file.                              |

Table 3. FPGA core component interface specifications.

| Function interface definition                      | Function interface description                                                                 |
|----------------------------------------------------|------------------------------------------------------------------------------------------------|
| void tcp_get_fpgapins_value();                     | Send the current pin value to the client when the TCP connection is successful.              |
| void fpgapins_realtime_value();                    | Send the changed pin status to the client immediately.                                       |
| void fpga_clk_source();                            | Systemc coroutine, analog fpga virtual platform clock source.                                |
| inline void sc_vlog_bind(void);                    | Systemc and verilog port binding functions.                                                 |

Table 4. FPGA debug component interface specifications.

| Function interface definition                      | Function interface description                                                                 |
|----------------------------------------------------|------------------------------------------------------------------------------------------------|
| void fpga_listen_wave(Json::Value &fpga)           | Enable waveform tracking.                                                                     |
| void fpga_set_event(Json::Value &fpga)             | Set the stimulus signal and debug the function of the verilog component.                     |
| void run_wave(sc_event& tb_clk)                    | Systemc coroutine sends the traced signal waveform to the client.                           |
| void wave_flag_reset()                             | Disable waveform tracking.                                                                   |

As shown in the above Table 1, Table 2, Table 3, and Table 4, it is the specification constraint of the component interface function of the FPGA virtual platform. These interface specification constraints are the core content of this solution.

The overall process is shown in Figure 2 below:
**4. Software verification and Conclusion**

To verify the FPGA virtual platform, complete the following steps in order:

First, build the FPGA virtual platform. Then, run the FPGA virtual platform. The source code of the verilog component is shown in Figure 3. The client is used to set the stimulus signal and enable the waveform tracking function. Click the DIP switch or the button to input the excitation signal, and observe the signal status of the pin as shown in Figure 4.
Second, use the waveform viewing software to view the waveform file (example gtkwave), as shown in Figure 5.

Finally, the tester used the physical hardware platform to load the above verilog components, and observed the peripheral status and timing status of the physical hardware. After testing and comparison, the results were consistent with the FPGA virtual platform and verified the correctness of the virtual platform.

After the above verification steps, users can use the FPGA virtual platform based on SystemC and verilog to develop and debug FPGA.

5. Acknowledgments
This work was financially supported by Shenzhen Special Fund for Strategic Emerging Industry Development JCYJ20170816151055158.

References
[1] Chen Xi, Xu Ningyi, SystemC System-on-Chip Design [M], Science Press, 2003.
[2] Wang Jinming, Verilog HDL Programming Tutorial [M], People Post Press, 2004.
[3] Qiao Lufeng, Verilog HDL Digital System Design and Verification [M], Electronics Industry Press, 2009.
[4] Jiang Guoqiang, New digital logic circuit [M], Beijing University of Posts and Telecommunication Press, 2013.