51 single-chip microcomputer verification platform with interrupt controller based on verilog2systemc

Bo Huang1,*, 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: hb265419@126.com

Abstract. Today, there is no interrupt debugger that can be used for free during development and learn with 51 microcontroller. except for integrated IDEs such as paid keil. Based on the advantages of network virtualization and independent of hardware conditions, a 51-chip virtual verification platform with interrupt controller based on verilog2systemc was completed. The effect achieved by this platform is that the same piece of code with interrupts can run verification on the actual 51 single-chip microcomputer, and can also run verification on the 51 virtual verification platform, and the running results of both are consistent and correct.

1. Introduction
As we all know, 51 microcontroller is an ancient and classic product, which has a wide range of applications and important influences. At present, under the windows operating system, the 51 microcontroller development and learning is still inseparable from keil_c51, which is very simple and easy to use, integrating coding, compilation, and debugging, but there is a disadvantage that it is a fee-based software. Under the Linux operating system, the most commonly used 51 MCU development and learning is sdcc, which can perform simple compilation work, can not achieve simulation, generally at most using sdcc to do a simple Compiled. In addition, a development board is still a waste of resources, but the software for online simulation has no choice other than keil_c51. So, there is a certain demand for the development of 51 single-chip microcomputers in other ways.

This software platform solution is based on such requirements and constraints, and proposes a 51-chip virtual verification platform with interrupt controller based on verilog2systemc. In short, the expected effect of this platform is that the same piece of bare metal code can run on both the physical 51 microcontroller and the 51 virtual verification platform, and the results of the two operations are consistent and correct.

2. Platform framework
Based on verilog2systemc, this software system implements a 51 single-chip microcomputer virtual verification platform with an interrupt controller. It is mainly used in the field of teaching and auxiliary development verification. 51 single-chip microcomputer virtual verification platform with interrupt controller, which is implemented by verilog HDL language 51 single-chip microcomputer CPU module [1], verilog is converted into systemc's 51 single-chip microcomputer virtual core module [2], 51 single-chip microcomputer interrupt controller module, and verification module. The framework is shown in Figure 1.
Software implementation

This platform is finally completed in a Linux environment. The software implementation process includes: 51CPU implemented by verilog; 51CPU implemented by verilog is converted into a virtual core of systemc; An interrupt controller module is added to the virtual core of systemc to implement interrupts. 51 single-chip microcomputer virtual core and verification. The specific implementation process is as follows:

3.1. Realize 51 single chip CPU of verilog

Verilog can be found in the open source project to implement 51 single-chip microcomputer project, standing on the shoulders of predecessors can climb higher peaks. Therefore, in this part, we will directly use the 51 microcontroller project implemented by verilog in the open source project to quickly implement the 51 microcontroller CPU of verilog.

3.2. Realize the virtual core converted from verilog to systemc

In the Linux environment, install systemc-2.3.2, open the Linux terminal, and execute the prepared makefile file in the terminal make. Then you can compile and complete the verilog to systemc implementation of the 51 single chip virtual core. After the above execution of make, the model from verilog to systemc has been completed, the basic functions of 51 single-chip microcomputers have been implemented, 51 single-chip bare metal codes can be run, and the correctness of bare metal codes can be verified. However, the model completed by this conversion does not have an interrupt function. In order to implement a 51 microcontroller single-chip virtual verification platform with an interrupt controller, our next difficult task is to implement the 51 microcontroller interrupt controller.

3.3. to achieve 51 single-chip interrupt controller module

Based on the modeling idea of systemc, in the existing systemc project of 51 single-chip microcomputer cores, create a class that implements the interrupt controller to implement the interrupt controller module.

First of all, understand that 51 single-chip microcomputers have 5 interrupt sources and 2 priority levels, and can achieve two-level interrupt nesting [3].

Next, it is known that the interface of the interrupt controller module systemc refers to the interface between the interrupt controller module and the 51 virtual core. The status information of the 51 virtual core can be obtained through these ports, so as to determine whether the interrupt is enabled, whether it can be triggered, whether it has been executed, etc. As shown in Table 1:
Table 1. Ports of the interrupt controller module systemc

| Number | Port definition | Port description |
|--------|-----------------|------------------|
| 1      | sc_in<bool> clk; | System clock, CPU clock, cpu to the module. |
| 2      | sc_in<bool> next_flag; | High to low can generate interrupt indicators, cpu to the module. |
| 3      | sc_in<bool> cmd32_flag; | interrupt flag, the cpu gives the module. |
| 4      | sc_in<uint32_t> rom_addr; | Memory address when interrupt is triggered, cpu is given to module. |
| 5      | sc_out<bool> inter_flag; | IT trigger flag, the module gives the cpu. |
| 6      | sc_out<uint32_t> inter_cmd; | Interrupt entry address of the interrupt request, the module gives the cpu. |
| 7      | sc_out<uint32_t> inter_addr; | Current memory address, the module gives the cpu. |

The specific sc_in indicates that the 51 virtual core is output to the interrupt controller module, and sc_out indicates that the interrupt controller module is output to the 51 virtual core.

Then, the interrupt trigger timing of the 51 virtual core is shown in Figure 2 below:

Figure 2. Virtual kernel interrupt trigger mode

The timing in Figure 2 above illustrates how the 51 virtual core interrupts are triggered. The specific description is as follows: When a low-to-high square wave appears in the 51 virtual core next_flag signal, it indicates that the virtual core is suitable for receiving an interrupt request at this time, then the interrupt controller module needs to do is: at the next clock cycle, set the inter_flag signal to high, save the current PC pointer to inter_addr, and at the same time set the inter_cmd value to 0x12. In the next two clock cycles, assign the interrupt vector entry address to the inter_cmd, Finally pull inter_flag low for the next clock cycle. Taking the serial port as an example, after assigning inter_cmd to 0x12, assign 0 to inter_cmd in the next clock cycle, and then assign 0x23 to inter_cmd in the next clock cycle (these three clock cycles cannot be hit (Broken), and finally pull inter_flag low in the next clock cycle, at this point you can trigger a serial port interrupt. Through the sc_in and sc_out interrupt controller modules, a communication channel is established with the virtual core, and both parties can obtain the data of the other party to know the current state, which is the basis for implementing the interrupt controller. systemc is based on coroutines. All the sensitive events of coroutines are clk mentioned above, which corresponds to the physical hardware, so it can achieve the same effect as hardware [4].

Later, conceive the relevant interface functions of the interrupt controller module. Through the function design of the interrupt controller module, the nesting of 51 single-chip microcomputer interrupts is realized [5-6]. The main interface functions are shown in Table 2:
Table 2. Main interface functions of the interrupt controller module

| Number | Interface function definition | Interface Description |
|--------|-------------------------------|-----------------------|
| 1      | bool uart_request_res(void);  | Determine UART request |
| 2      | bool t1_request_res(void);    | Determine Timer 1      |
| 3      | bool ex1_request_res(void);   | Determine External    |
| 4      | bool t0_request_res(void);    | Determine Timer 0      |
| 5      | bool ex0_request_res(void);   | Determine External    |
| 6      | uint8_t uart_int_deal(void);  | UART interrupt request |
| 7      | uint8_t timer1_int_deal(void);| TIMER1 interrupt      |
| 8      | int8_t ex1_int_deal(void);    | EX1 interrupt request  |
| 9      | uint8_t timer0_int_deal(void);| TIMER0 interrupt      |
| 10     | uint8_t ex0_int_deal(void);   | EX0 interrupt request  |
| 11     | void uart_int_flag_clear(void);| Serial port interrupt |
| 12     | void t1_int_flag_clear(void); | TIMER1 interrupt flag  |
| 13     | void ex1_int_flag_clear(void);| EX1 interrupt flag     |
| 14     | void t0_int_flag_clear(void); | TIMER0 interrupt flag  |
| 15     | void ex0_int_flag_clear(void);| EX0 interrupt flag     |

Finally, the signal connection of the 51 virtual kernel module and the 51 interrupt controller module is shown in Figure 3 below:

Figure 3. Signal connection diagram of 8051 virtual core and 51 interrupt controller module

3.4. Implement the verification module.

In the bare metal code, set external interrupt 1 to high priority, then the priority order of the interrupts is external interrupt 1, external interrupt 0, timer 0, serial port interrupt. It runs on the 51 single-chip microcomputer. After testing, the bare metal code with interrupts is correct.

Load the bin file generated by the bare metal test code with the interrupt in the previous step to the virtual kernel with the interrupt controller, and verify the interrupt controller module by comparing the effect of the actual operation. If the interrupt execution order is consistent with the actual object, it indicates that the virtual verification platform of the 51 microcontroller interrupt controller is consistent with the expected effect; otherwise, it is necessary to locate which interrupt is not executed, or the interrupt execution order is inconsistent with the actual object, and then continuously modify the debug interrupt control Module, until the interrupt controller of 51 single-chip microcomputers is realized.

The overall implementation process is shown in Figure 4 below:
4. Validation result and conclusion

After the above software implementation, we need to verify the results of our design. Open the .vcd file generated in the simulation core running process in gdkwave. An excerpt is shown in Figure 5 below:

![Gdkwave screenshot]

**Figure 5.** Screenshot of interrupt controller test gdkwave

An enlarged screenshot of the first three interruptions in Figure 5 is shown in Figure 6:

![Gdkwave screenshot]

**Figure 6.** Interrupt controller test gdkwave enlarged screenshot

Combining the information in Figures 5 and 6 above with the experimental verification module, the description is as follows: In Figure 5, there are 4 waveforms on cmd32_flag, which means that the execution of 4 interrupts is completed. The last waveform on cmd32_flag corresponds to 0223 of inter_cmd [31: 0], which is the interrupt entry address of the serial port, indicating that the serial port interrupt execution is complete. Figure 6 is an enlarged part of the first three interrupts in Figure 5. The 000B, 0003, and 0013 of inter_cmd [31: 0] correspond to the interrupt entry addresses of timer 0, external interrupt 0, and external interrupt 1, respectively, and then sent to the interrupt of the virtual core. To apply, execute first. It can be seen that external interrupt 1 is executed first, then external interrupt 0, and then timer 0, so before the serial port interrupt application, there are 3 waveforms on cmd32_flag, which indicates external interrupt 1, external interrupt 0, and timer 0 interrupt. After all
three interrupts have been executed, the serial port interruption is performed. The results shown in Fig. 5 and Fig. 6 verify that the interrupt priority order is: external interrupt 1, external interrupt 0, timer 0, serial port interrupt, and thus also verify the interrupt controller of the 51 virtual core.

After verification, the 51 single chip microcomputer virtual verification platform with an interrupt controller implemented by this method has the same effect as the real hardware 51 single chip microcomputer development board and achieves the expected effect. Provides a debugging and verification method other than keil_c51, which can be used by 51 microcontroller developers without a physical 51 development board and unable to experiment with keil_c51 on this method.

5. Acknowledgments

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

References

[1] Wang Lei, Liao Fan, Ji Hongyang. Research on Header File Nesting Relationship in Verilog HDL to SystemC Compilation, Computer and Modernization, 2011 (10): 183-186.
[2] Lu Yanjun. Design of Joint Simulation Platform Based on SystemC and SystemVerilog, Science and Technology Innovation, 2017 (27): 16-18.
[3] Zhang Youlin. Single Chip Microcomputer Interrupt Control System, Information and Computer (Theoretical Edition), 2017 (23): 91-92 + 95.
[4] Bao Zhizhong. SystemC-based precise model of configurable dedicated processing core period, Nanjing University, 2015.
[5] Zhou Hongyue. Design and Research of SoC System-level Modeling and Simulation Platform, Tianjin University, 2012.
[6] Cheng Chunlei. Design and development of functional model of ARM core SoC platform based on SystemC, North China Electric Power University (Beijing), 2008.