Research on Verification and Implementation of RTL-based VHDL Simulator

Leng Ming*, Sun Ling-yu

Department of Computer Science, Jinggangshan University, Ji'an 343009, China

Abstract

VHDL simulator based on Register Transfer Level (RTL) is implemented and verified, named RVS. Firstly, we give the implementation of RVS. Secondly, we design the micro program SAP-CPU and logic SAP-CPU based on VHDL language, which includes the format of control instruction, instruction set, addressing method, test program and the architecture of logic SAP-CPU and micro program SAP-CPU. Finally, the experiment and analysis show that the simulator of RVS perform well and produce encouraging solutions correctly on two SAP-CPU designs controlled by combinational logic and micro-program.

© 2011 Published by Elsevier B.V. Selection and/or peer-review under responsibility of International Materials Science Society.

Register Transfer Level; VHDL Language; Simulation; Verification; Implementation.

1. Introduction

The technology of Microelectronic has accelerated the development of modern society. As the core of computer, microprocessor has been applied in various areas. With the development of microprocessor technology, traditional methods of logic design couldn’t meet the needs of design and have been replaced by EDA method step by step. But EDA method is also faced with challenge of high complexity of design. VHDL and Verilog, as the most representative HDL, which is a formalization language used to describe the function of hardware circuit, is a powerful and standardized HDL. In the context of increasing need for LSL design, more and more attention and popularization from the industry make HDL, an international standard language, the primary HDL in hardware design [1]. Essential to HDL design is the ability to simulate HDL programs. Simulation allows an HDL description of a design to pass design verification, an important milestone that validates the design's intended function against the code implementation in the HDL description [2].

In [3], we design a VHDL simulator based on RTL, which inputs the VLSI design described by VHDL language of RTL, generates intermediate code by the corresponding compiler, reads the external excitation signal from the waveform file and applies it on the behavior model. We can decide whether a
VLSI design is achieving the desired function by means of the response of the behavior model, which was imposed the external excitation signal.

In this paper, we give the implementation of RVS. Secondly, we design the micro program SAP-CPU and logic SAP-CPU based on VHDL language, which includes the format of control instruction, instruction set, addressing method, test program and the architecture of logic SAP-CPU and micro program SAP-CPU. Finally, the experiment and analysis show that the simulator of RVS perform well and produce encouraging solutions correctly on two SAP-CPU designs controlled by combinational logic and micro-program.

2. The Implementation of VHDL simulator based on RTL

We design and implement the VHDL simulator based on RTL, which used for the verification and debugging of VLSI design. RVS system mainly consists of two modules which are compilation module and simulation module, as shown in Figure 1 [3]. In the RVS compilation module, we split the flow of compilation into three phases which consist of lexicography, syntax and semantic analysis. In the syntax analysis phase, we propose the top-down syntax analysis method based on recursion [4]. In the semantic analysis phase, we generate intermediate code for the RVS by the rule of syntax analysis method based on recursion. In the RVS simulation module, we implement the simulation scheduler and improve the data structure of the algorithm and disposal of condition, assign expression in the process of simulation. Furthermore, we propose an event-driven schedule algorithm based on process is proposed with debugging functions [3, 5].

![Diagram](image)

Fig. 1. The flow chart of VHDL simulator based on RTL

3. The micro program SAP-CPU and logic SAP-CPU
In this section, we present the format of control instruction, instruction set, addressing method and test program of SAP-CPU. Furthermore, we describe the architecture of logic SAP-CPU and micro program SAP-CPU. Due to limited space, we give the source code of logic SAP-CPU and micro program SAP-CPU described by the VHDL language in the appendix of [6].

3.1. Format of control instruction

We design the format of control instruction of SAP-CPU, which the bits of operation code ranges from 4 to 7 and the bits of address code ranges from 0 to 3, illustrated by Figure 2.

Fig. 2. The format of control instruction of SAP-CPU

3.2. Instruction set and addressing method

Table 1 presents the instruction set and addressing method of SAP-CPU, which includes LAD, ADD, SUB, OUT, HLT instruction. The LAD instruction load the data of specified address to accumulator, whose addressing method is direct addressing and operation code is ‘0000’. The ADD instruction add the data of specified address and the accumulator together and assign the result to the accumulator, whose addressing method is direct addressing and operation code is ‘0001’. The SUB instruction minus accumulator with the data of specified address and assign the result to the accumulator, whose addressing method is direct addressing and operation code is ‘0010’. The OUT instruction displays the value of accumulator on output register, whose addressing method is zero addressing and operation code is ‘1110’. The HLT instruction terminates the execution of CPU, whose addressing method is zero addressing and operation code is ‘1111’.

Table 1. The instruction set and addressing method of SAP-CPU

<table>
<thead>
<tr>
<th>Instruction</th>
<th>operation code</th>
<th>example</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAD(Direct Addressing)</td>
<td>0000</td>
<td>LDA 9H</td>
<td>load the data of memory 9H to accumulator</td>
</tr>
<tr>
<td>ADD(Direct Addressing)</td>
<td>0001</td>
<td>ADD 0BH</td>
<td>add the data of memory 0BH and the accumulator together, assign the result to the accumulator</td>
</tr>
<tr>
<td>SUB(Direct Addressing)</td>
<td>0010</td>
<td>SUB 0EH</td>
<td>accumulator minus the data of memory 0EH, assign the result to the accumulator</td>
</tr>
<tr>
<td>OUT(Zero Addressing)</td>
<td>1110</td>
<td>OUT</td>
<td>display the value of accumulator on output register</td>
</tr>
<tr>
<td>HLT(Zero Addressing)</td>
<td>1111</td>
<td>HLT</td>
<td>terminate the execution of CPU</td>
</tr>
</tbody>
</table>

3.3. Test program of SAP-CPU

Table 2 presents the test program of SAP-CPU according with the format of control instruction, instruction set, addressing method. Firstly, the accumulator loads the data of memory 9H which is 10H. Secondly, the accumulator adds the data of memory 0AH and 0BH which are 14H and 18H separately. Thirdly, the accumulator minus the data of memory 0CH which is 20H. Finally, we display the value of accumulator on output register and terminate the execution of CPU, which is 1CH (10H+14H+18H-20H).
Table 2. The test program of SAP-CPU

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation code</th>
<th>Example</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0H</td>
<td>0000 1001</td>
<td>LDA 9H</td>
<td>load the data of memory 9H to accumulator</td>
</tr>
<tr>
<td>1H</td>
<td>0001 1010</td>
<td>ADD AH</td>
<td>add the data of memory 0AH and the accumulator together, assign the result to the accumulator</td>
</tr>
<tr>
<td>2H</td>
<td>0001 1011</td>
<td>ADD BH</td>
<td>add the data of memory 0BH and the accumulator together, assign the result to the accumulator</td>
</tr>
<tr>
<td>3H</td>
<td>0010 1100</td>
<td>SUB CH</td>
<td>accumulator minus the data of memory 0CH, assign the result to the accumulator</td>
</tr>
<tr>
<td>4H</td>
<td>1110 XXXX</td>
<td>OUT</td>
<td>display the value of accumulator on output register</td>
</tr>
<tr>
<td>5H</td>
<td>1111 XXXX</td>
<td>HLT</td>
<td>terminate the execution of CPU</td>
</tr>
</tbody>
</table>

3.4. The architecture of logic SAP-CPU

The architecture of logic SAP-CPU is illustrated by Figure 2(a), which includes nine components. Component 1 is program counter, which records the address of the next instruction and transfers to the MAR register. Component 2 is MAR register that is address register. MAR has two source inputs, one source is accepted by the external address of the program or data memory, the other source is inputted the address of the next instruction. Component 3 is RAM whose size if 16*8 bits. Component 4 is instruction register (IR), which inputted by the data of RAM. The upper 4 bits of IR is operation code used for the decoder and the lower 4 bits of IR is address code used for MAR. Component 5 is decoder, which decodes the operation code into control signals used for directing other components. Component 6 is accumulator, which stores the real-time data of CPU. Component 7 is adder, which will be responsible for adding or subtracting operation and transfer the result back to accumulator. Component 8 is register, which cooperate with the accumulator and adders. Component 9 is output register, which stores the value of accumulator and drive the display of LED.

3.5. The architecture of micro program SAP-CPU

The architecture of micro program SAP-CPU is illustrated by Figure 2(b), which is same with the logic SAP-CPU in addition to the following three components. Component 1 is 5*11 bits micro-program memory which includes 5 instructions in microcode and the microcode of each instruction is 11 bits in length. Component 2 is buffer register which stores the microcode of each instruction. Component 3 is micro-address register which stores the operation code of instruction register.
4. The Verification of VHDL simulator based on RTL

4.1. Parameter settings of testbed.

The reset signal rst is set "1" in 0-Time, continue to 20-Time and change to "0". The cycle of clock signal Clk is set 15 time, which is "0" with a 12-Time, "1" with a 13 Time.

Fig. 4 shows the simulation waveform of SAP-CPU in Quartus software. We can get the result of computation from output signal, which is 1CH. The results of the simulation show that source code of micro program SAP-CPU and logic SAP-CPU is correct. Furthermore, we get the result of simulation in RVS system and the value of output signal is 00011100. The experimental verification on RVS system and analysis show that the simulator of RVS perform well and produce encouraging solutions correctly on two SAP-CPU designs controlled by combinational logic and micro-program.

Acknowledgements

This work was financially supported by the National Natural Science Foundation of China under Grant No. 61063007/61163062/61106030 and Natural Science Foundation of Jiangxi under Grant No. 2009GQS0060.
References


