

**EEE373 – Digital System Design**

Department of Electrical Engineering & Electronics

**Assignment 4\_Report**

-NIOS-II Custom Instructions-

**Abstract**

Assignment 4 is mainly to use QSYS to build a basic NIOS-II system, then add a custom instruction to it, and write a program to test this custom instruction. Among them, it needs to understand the use of ASM design methods to design digital systems, the use of Verilog hardware description language to implement digital systems and the use of Quartus NIOS-II to implement SOPC systems.

**Table of Contents**

[**Abstract**](#_Toc512788800)

[Table of Contents](#_Toc512788801)

[**1** **Introduction 1**](#_Toc512788802)

[**2** **ASM Chart and Verilog code 2**](#_Toc512788802)

[**3** **Results 4**](#_Toc512788802)

[**4** **Discussion 5**](#_Toc512788802)

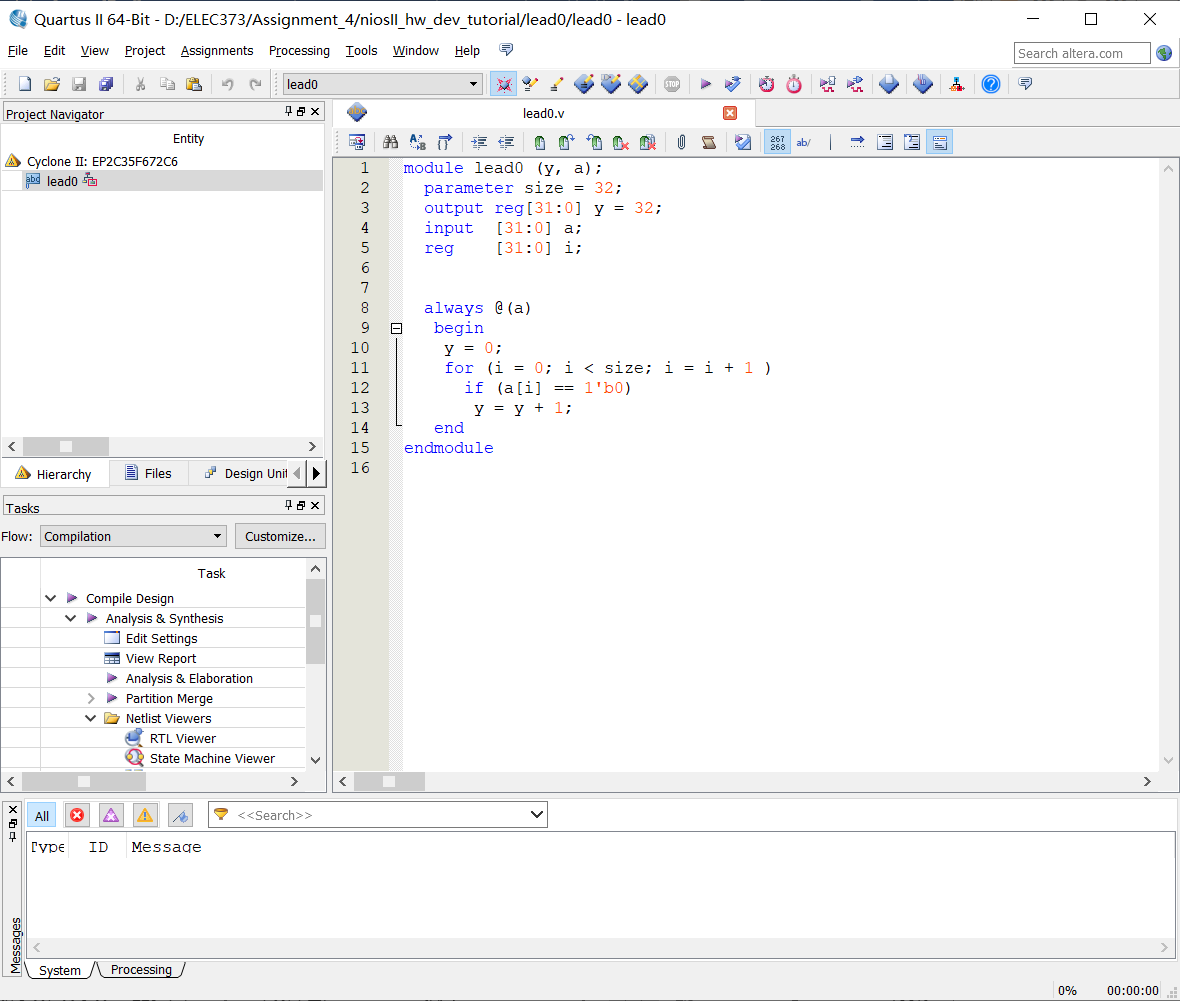
[**Appendix 6**](#_Toc512788802)

**1. Introduction**

First, the basic NIOS-II system is built according to the NIOS-II tutorial manual and video, in which a custom instruction is developed and its operation in the NIOS processor is simulated. The custom instruction is required to calculate the number of leading zeros in the 32-bit number passed to the instruction. Then, use ModelSim to test and simulate the custom instructions on the C/C++ assembler of the NIOS processor. Finally, the speed of the custom instruction is compared with your software implementation. In addition, an ASM diagram of the custom instructions compiled in Verilog is drawn and the simulation and test results for the NIOS II processor are discussed.

**2. ASM Chart and Verilog code**

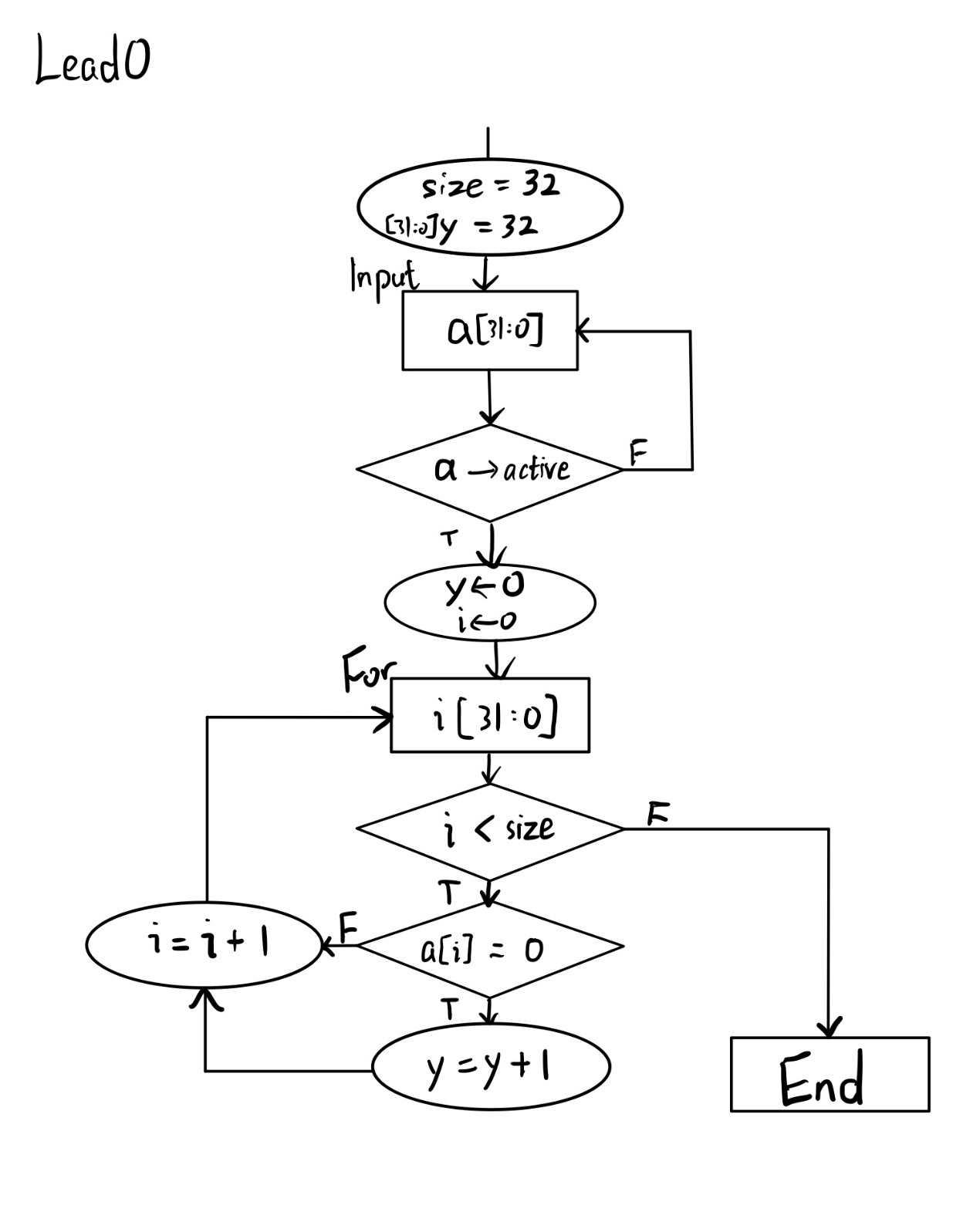
Figure 2.1 is a custom instruction compiled with Verilog.



**Figure 2.1:** The Verilog code of the custom instruction

The code is to calculate the number of leading ‘0’ in the 32-bit number, by using for () to loop each of the 32-bits, and then use if () to determine whether each of the 32-bits is ‘0’. If input **a** = 0xFF00F000, output **y** = 0x14 (Decimal is 20).

Figure 2.2 is the ASM chart is drawn based on the Verilog code in Figure 2.1.

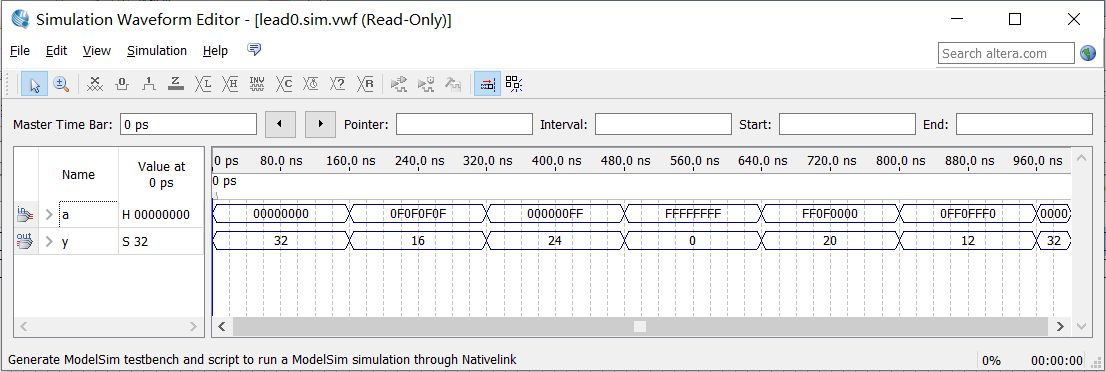
****

**Figure 2.2:** The ASM chart of the custom instruction

When input **a** is not active, it is 0x0000000 and has 32 zeros, output **y** = 32. When input **a** is activated, it enters a cycle of 32 times while **y** and **i** are equal to 0. If **i** is less than **size** = 32, the loop continues, otherwise the program ends. When the cycle continues, **i** and output **y** will be **+1** if the current bit is 0, otherwise only **i** + 1. Note the number of cycles **i** corresponding to the number of bits judged.

**3. Results**

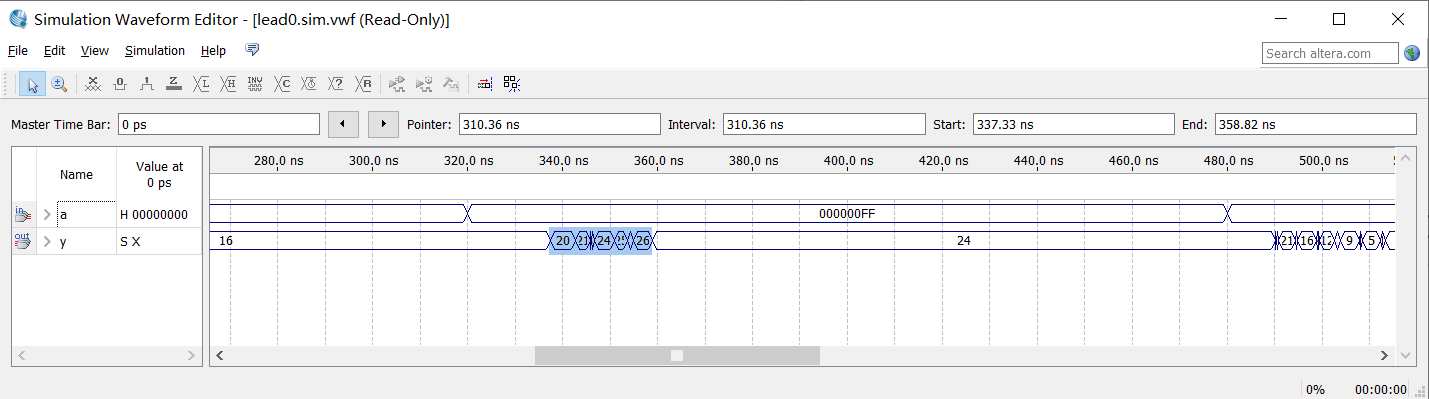
Figure 3.1 is the result of testing the Verilog code.

****

**Figure 3.1:** The waveform of the custom instruction

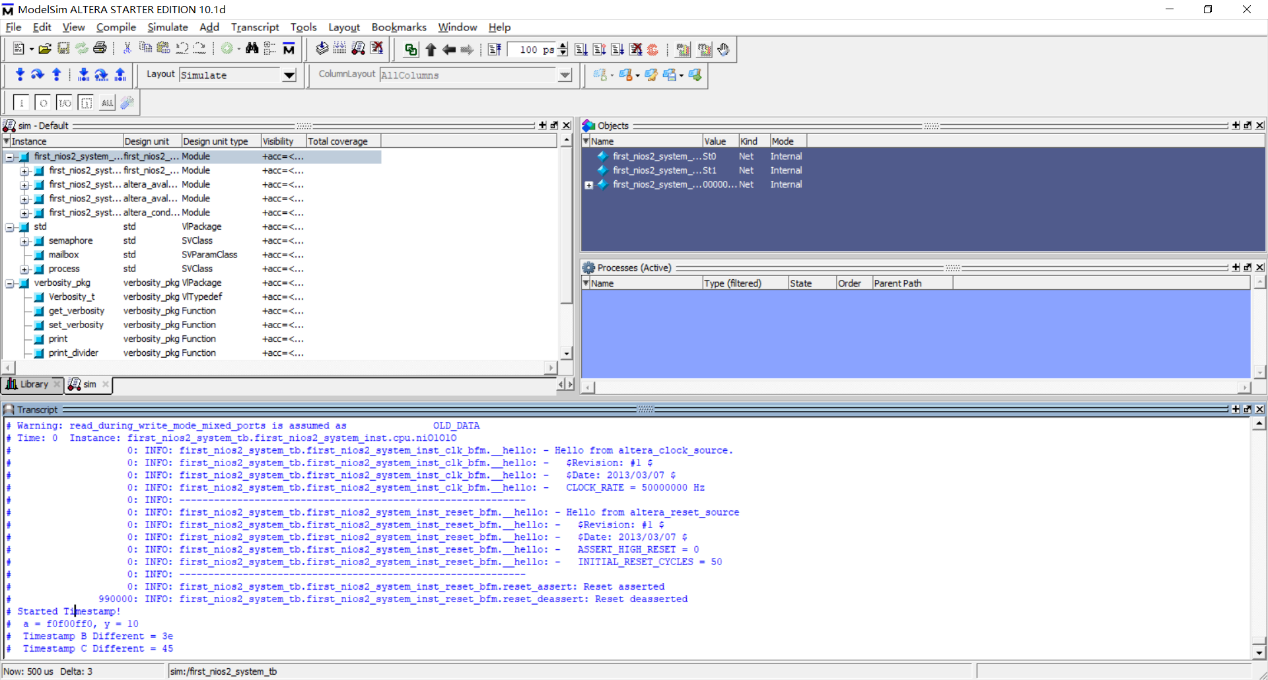
The input **a** is a 32-bit number, and the output **y** is the result of calculating the number of zeros.

Figure 3.2 is the time delay of simulating custom instructions on Verilog (Software).



**Figure 3.2:** The Simulation of the custom instruction in the Verilog (Software)

The time taken for the custom instruction in the software is 358.82 ns – 337.33 ns = 21.49 ns.

Figure 3.3 shows the results of simulating the Verilog code in the NIOS processor (Hardware) and testing it with C/C++ assembler. (run 500us)****

**Figure 3.3:** The Simulation of the custom instruction in the NIOS-II (Hardware)

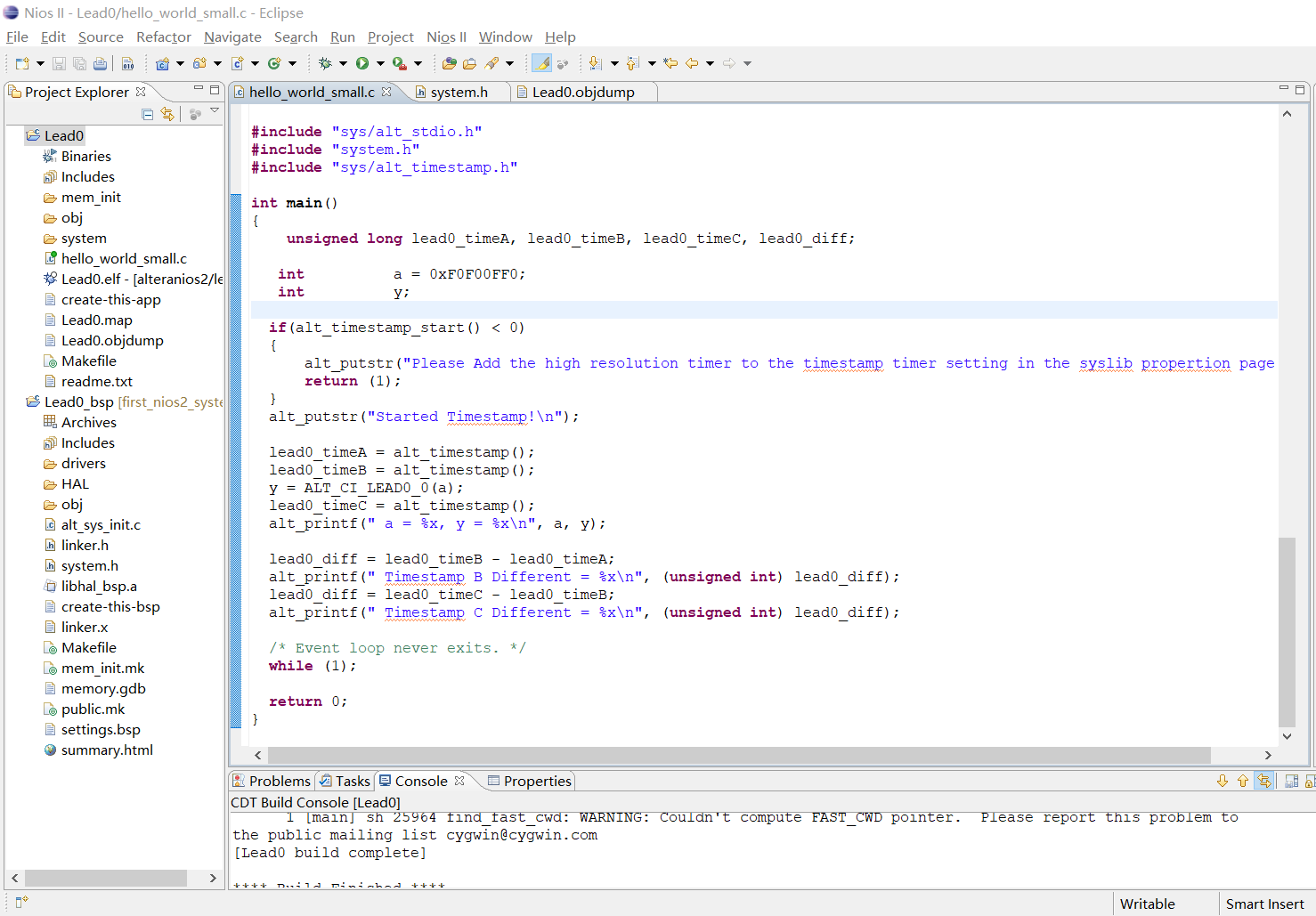
In the C/C++ assembler, set the input **a** = 0xf0f00ff0, the simulated output **y** = 0x10 (Decimal is 16, there are 16 zeros). The time taken for the custom instruction in the hardware is 0x45 – 0x3e = 0x7 ns. (f = 50 MHz)

**4. Discussion and Conclusion**

It can be analyzed from the results that the execution speed of software is lower than that of custom instructions. Therefore, if the core functions implemented by the standard instruction sequence are implemented with custom instructions, the execution efficiency of the system can be significantly improved.

**Appendix**

**C/C++ for test program**

****