**Design Report**

**Final Project of Logic and Computer Design Fundamentals**

**Wu Jianan, Yao Peiyong, Zhao Bingqian**

**Date: 2011-1-23**

Catalogue

[Chapter 1: Objective 3](#_Toc283577595)

[Chapter 2: Detailed Design 3](#_Toc283577596)

[2.1 Specification and Requirement 3](#_Toc283577597)

[2.2 Analysis 3](#_Toc283577598)

[2.3 The Divider Module 4](#_Toc283577599)

[2.4 Synthesize and Debugging 4](#_Toc283577600)

[2.5 Simulation 5](#_Toc283577601)

[Chapter 3: Discussion 5](#_Toc283577602)

[3.1 Possible Solutions 5](#_Toc283577603)

[3.2 Algorithm Specification and Proving 6](#_Toc283577604)

[3.3 Improvement and Comments 7](#_Toc283577605)

[Chapter 4: Conclusion 7](#_Toc283577606)

[Reference 7](#_Toc283577607)

[Contributions 7](#_Toc283577608)

# Chapter 1: Objective

In this experiment, we are to design a divider. In detail, given a 16-bit unsigned number and an 8-bit unsigned number, the divider calculates the quotient and the remainder.

The quotient and remainder are limited as two 8-bit unsigned numbers. So, when the quotient is longer than 8-bit, it should give a signal that the quotient is overflow. What’s more, the divisor is illegal to be 0, and a signal should be given when it’s illegal.

# Chapter 2: Detailed Design

## 2.1 Specification and Requirement

Use the switches and buttons to input the dividend and divisor. The dividend is a 16-bit unsigned number, and the divisor is an 8-bit unsigned number.

Use the switches to choose the quotient or the remainder. The quotient and the remainder are both 8-bit unsigned numbers.

A LED lights to indicate the quotient is longer than 8-bit or the divisor is 0.

## 2.2 Analysis

A 16-bit unsigned number can be displayed by 4 seven-segments in hexadecimal number, and an 8-bit can be displayed by 2 seven-segments. Use 2 switches to choose which number to display, and two other switches and one button to input the dividend and divisor. Use two 1-bit numbers DZ and DO to be the flags of the 0 divisor and the quotient overflow.

We use a data\_input module to input the data, which calls the debounce module. The divider module calculates the quotient and remainder. Then, the display\_driver module handles the information to be output. Finally, the seg7\_display module displays the numbers and the led\_lamp module lights the LED. Of course, the clock module is necessary for the whole project.

Anti\_jitter

Data\_input

divider

Display\_driver

Led\_lamp

Seg7\_display

Clk\_divider

Totally, there 7 modules: anti\_jitter, data\_input, divider, display\_driver, led\_lamp, seg7\_display, clk\_divider. As 6 modules except the divider module, have be provided by Wang Donghui teacher. We only discuss the detail of the divider module here.

## 2.3 The Divider Module

The divider module gets the 1ms clock signal, dividend and divisor. Output two 8-bit unsigned number quotient, remainder and two flags DZ, DO. Every 1 ms, this module calculate the answers of the current inputs.

First, the quotient, remainder, DZ and DO have all be initialized to 0. We use a 24-bit unsigned number divid to store the result of divider, whose first 8-bit are 0 and last 16-bit equal to the dividend.

Then, we do the following steps 16 times. Shift the divid left 1 bit. If the first 8-bit of divid is no less than the divisor, subtract its first 8-bit by the divisor and add 1 to the last bit.

Finally, the last 16-bit of divid is the quotient, but we just get the last 8-bit. The first 8-bit of divid is the remainder. If divisor equals to 0, quotient and remainder assign 0, and DZ assigns 1.If the last 16-bit of divid is larger than 255, which means the quotient is overflow, the flag DO assigns 1.

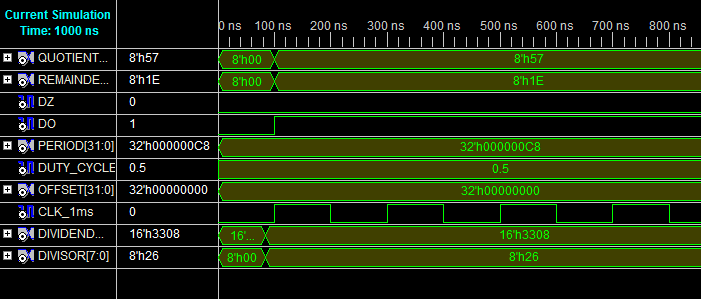
## 2.4 Synthesize and Debugging

When all the modules have been designed, we call them all in the top module, using wires to connect them.

We tested our program by Xilinx ISE Design Suite 10.1 on Xilinx Spartan-3 Boards & Kits in the CS lab of ZJG. We inputted some test data and compared their answers to the right answer calculated by the Calculator of Windows. At first, we found the divider couldn’t get the correct answer. We analyzed the algorithm, calculated it by ourselves on the paper, and found no problem. Then, we examined the program and found the increase of the last bit was written in the wrong place. After we corrected the error, the divider worked well and passed all the test data.

## 2.5 Simulation

The last four digits of our Group Leader’s student id are 3308, and the last two digits of one Group Member’s student id are 26. So, we divide 3308 by 26 in hexadecimal number. The simulation result of the divider module is provided as follows.



The quotient is overflow, so DO is 1. The quotient is 57 and the remainder is 1E. The result is correct.

# Chapter 3: Discussion

## 3.1 Possible Solutions

We have thought about two solutions to construct a divider:

1. By subtraction.
2. The most intuitionistic way is using dividend minus divisor until the dividend is not enough. Though this design is simple (all we need are two subtracters ), this way is too primitive and takes a lot of time.
3. Use resorting or non-resorting way to construct a divider. The main idea is shifting, subtraction and comparison.
4. Use SRT algorithm.
5. By multiplication.
6. Newton-Raphson algorithm and Goldschmidt algorithm change “a/b” into “a\*(1/b)” and through functional iteration to calculate 1/b.

There are many other solutions like very high radix, table look-up, variable latency and so on.

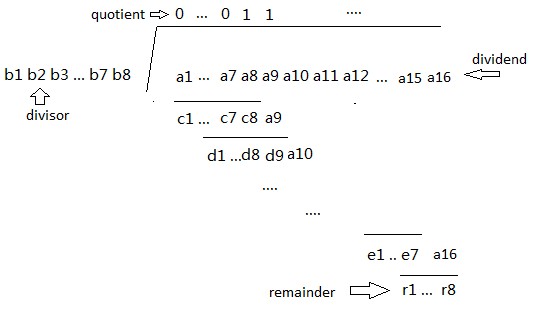
The goal of this project is to design a 16-bits unsigned integer divide 8-bit unsigned integer divider and we need to get the remainder. So finally we choose a method like resorting to construct the divider, it’s easy to implement and its speed is acceptable. The reason why we don’t choose other methods is that they are hard to get the remainder and they are too complicated, we shouldn’t take a musket to kill a butterfly.

## 3.2 Algorithm Specification and Proving

In resorting method, we minus the divisor and if the result is positive, add the quotient by 1, otherwise, add the quotient by 0 and add back the divisor to the dividend, finally shift the dividend.

In our algorithm, we compare the divisor and the dividend, if dividend is larger, then add the quotient by 1, otherwise add the quotient by 0. Then shift. The only difference is that we use a comparison takes the place of addition (add back the divisor). The comparison can be implemented through a subtractor, so actually these these two ways are the same.

Now let’s see the algorithm is correct. The method is very similar with the computation on the paper with hands. The comparsion is the process we decide quotient is 0 or 1, the shiftint is the borrow. For the quotient is either 0 or 1, so the subtraction can be considered as divison. After 32 times’ comparison, we can get the quotient and remainder. Like the fllowing figure:



## 3.3 Improvement and Comments

Non-resorting algorithm is an improvement of resorting algorithm, though we don’t implement it in our design.

In non-resorting algorithm, we don’t add back the divisor if the divisor is larger than the dividend, but according the last quotient. If the last quotient is 1, we use dividend minus divisor as usual. If the last quotient is 0, we use dividend add divior. Can be proved, this method is correct also and it can save one addtion every iteration.

# Chapter 4: Conclusion

In this project, we understand that different implementations of the divider and implement a kind of divider by ourself. Deepened the understanding of time and space requirements. Strengthen the use of the verilog language.

Also, we Improve the ability to search literature and develop a sense of teamwork.

# Reference

1. 《Design and Implementation of a Fixed-Point Arithmetic Unit》 Ge Liang(Computer Architecture)

— We get to know the difference of dividers through this article.

1. 《基于FPGA的32位除法器设计》 周殿风，王俊华

— We design our implementation according this article.

1. 《基于加减交替法除法器的FPGA设计》 许勇

— We get to know about non-resorting method by this article

1. 《Verilog数字系统设计教程》 夏闻宇

— We take this book as verilog grammar reference.

# Contributions