 [Translated from Greek to English - www.onlinedoctranslator.com](https://www.onlinedoctranslator.com/en/?utm_source=onlinedoctranslator&utm_medium=docx&utm_campaign=attribution)

8/3/2018

# 1st Laboratory Exercise

**DESIGN AND SIMULATION OF ARITHMETIC AND LOGIC UNIT (ALU) AND REGISTER FILE**

**27/02/2018**

Group LAB31235515

|  |
| --- |
| *CHRISTOS ZISKAS 2014030191* |
| *ANTONIS ANTONAKAKIS 2014030160* |

## Purpose of laboratory exercise

It is dealing with the hardware description language VHDL (VHSIC Hardware Description Language) for the construction of a unit of numerical and logical calculations, as well as the design of a complete register file according to the behavioral or structural knowledge of VHDL. In addition, this design is a trigger for the implementation of an important part of a processor - integrated functionality - digital circuits.

## Preparation

We present the correlations for the first circuit which concerns calculations between operands (arithmetic-logical operations) of the computing unit but also the second circuit which concerns the implementation of the register file - We present the ALU unit but also the schematic illustration of a register element as well as the overall registry file entity.

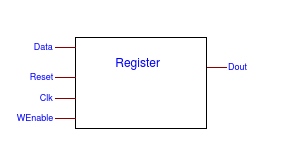
Circuit 1

Arithmetic and logical operation unit (ALU unit)

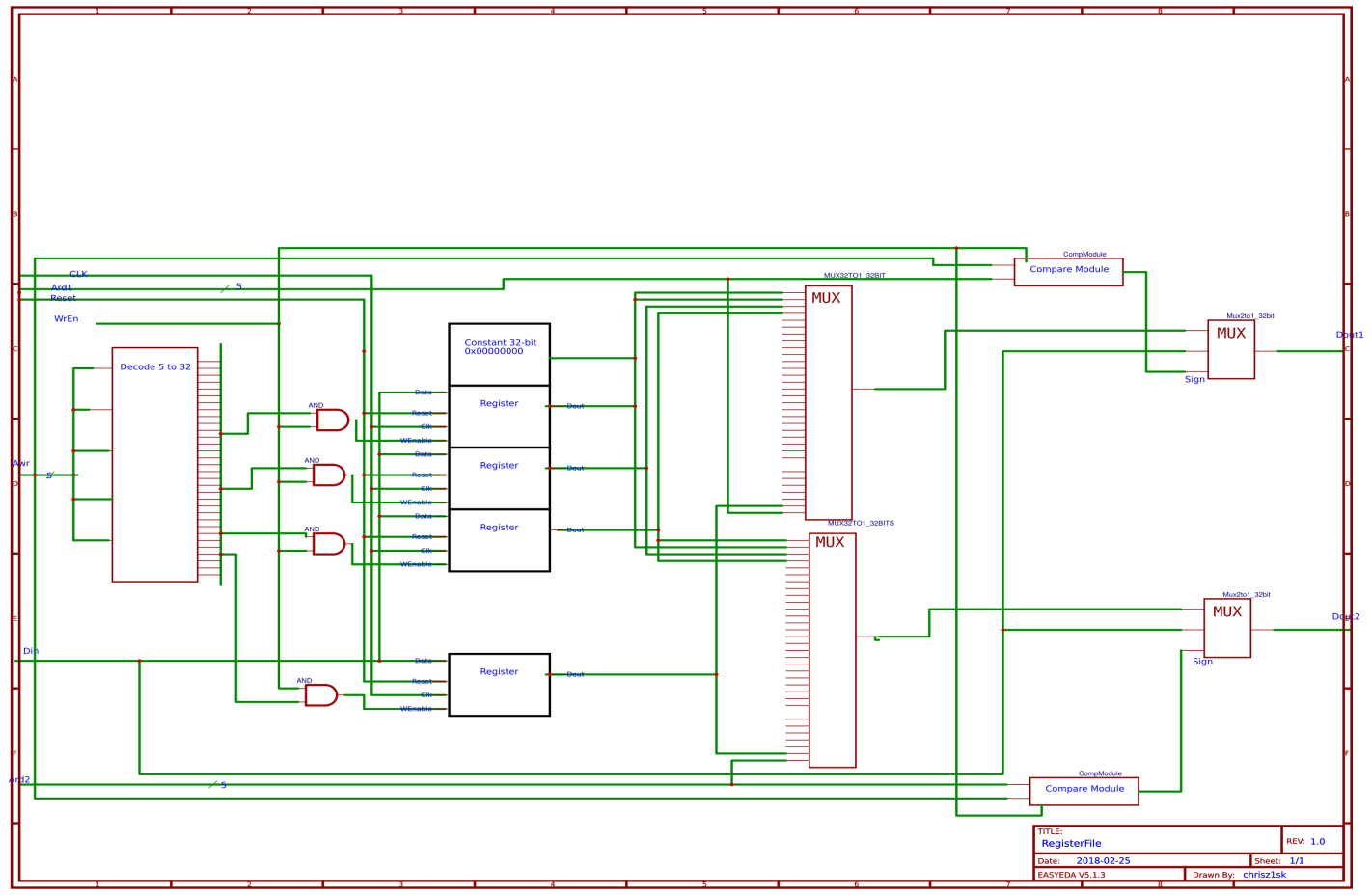
|  |  |  |
| --- | --- | --- |
| Code | Act | Result |
| Op=0000 | Addition | Out=A+B |
| Op=0001 | Removal | Out=AB |
| Op=0010 | Logical “NO And” | Out=!(A&B) |
| Op=0011 | Logical “H” | Out=A|B |
| Op=0100 | Inversion of A | Out=!A |
| Op=1000 | Numeric shift right by 1 place | Out=(int)A>>1 |
| Op=1001 | Logical shift right by 1 position | Out=(unsigned int)A>>1 |
| Op=1010 | Logical shift left by 1 position | Out=A<<1 |
| Op=1100 | Circular slide to the left of A by 1 position | Out={A(30),,,,,,A(31)} |
| Op=1101 | Circular shift to the right of A by 1 position | Out={A(0),,,,,,,A(1)} |

## Circuit2

2.1 Registrar & File of Registrars



2.2 File of registrars



## Description

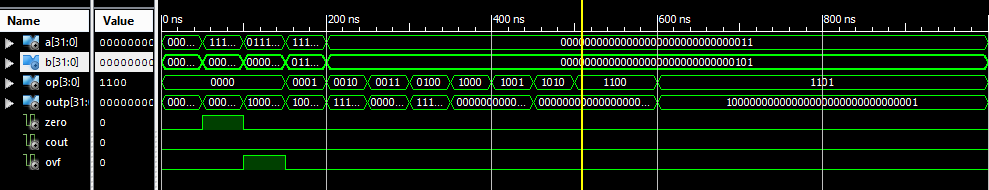
For the first circuit, the presentation of the ALU unit is requested for representing mathematical, logical and sliding operations for inputs A, B and outputs depending on the type of operations. Mathematical commands infer a result of either zero, carry, or overflow. The logic operations affect the result through logic gates (NAND,OR,NOT) while the shift operations act on the operator A. For the second circuit, a prerequisite is the implementation of the design of a register. Its construction consists of a D-flipflop with a write-enable input and a 2-to-1 multiplexer. Completion of this design leads to its completion by producing a register file that includes the number of 32 registers with three ports (two read and one write). . The design needs a 5 to 32 decoder to match the read addresses to the corresponding register as well as the asynchronous Compare module circuit that checks if the two registers are the same when the enable signal of the registers is activated. The input is routed to the file via the 32bit Din variable. For enabled enable, the input is written to one of the registers except the register in the first position whose value remains constant. The file simultaneously reads the data of the registers through the read addresses and displays their contents. . The register value is driven to the output when enable is given in the same clock cycle while the storage in the register is performed one clock cycle later than the time enable is given (understandable because if the write-read addresses coincide then Din is output at the dout output after the RF it is asynchronous- The compare module gives 1 to the select of the multiplexer). The register file is considered an asynchronous circuit (the clock element is used to operate the register)

## Waveforms-Simulation

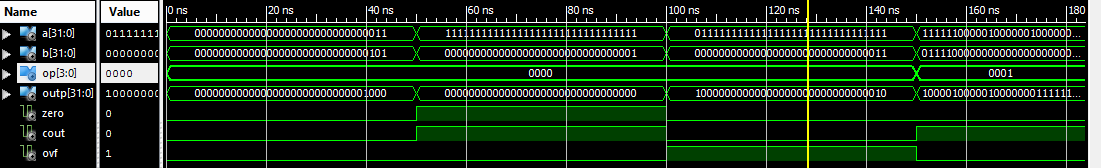
We present the waveforms of the 2 circuits

Circuit 1

Arithmetic and logical operation unit (ALU unit)



The codes are used to analyze the output of each calculation. Inputs are represented by 32bit numbers.



According to the operation of the addition, we place as inputs numbers that show, per case, the various outputs of held, overflow and zero result (011+101=1000,11...1+0......1=0000 and is maintained prisoner).

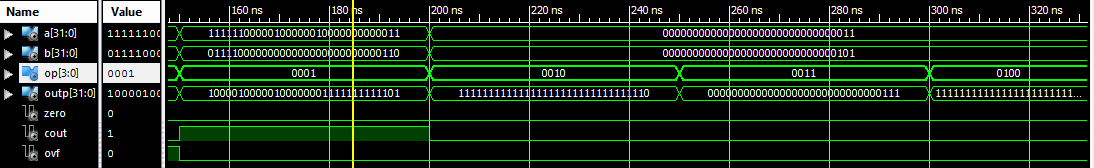
The overflow phenomenon is demonstrated when the addition of numbers is performed



The finding of the phenomenon is verified by observing the MSBs (must be identical) of the operands by comparing the MSB of the result. Their difference highlights the phenomenon.

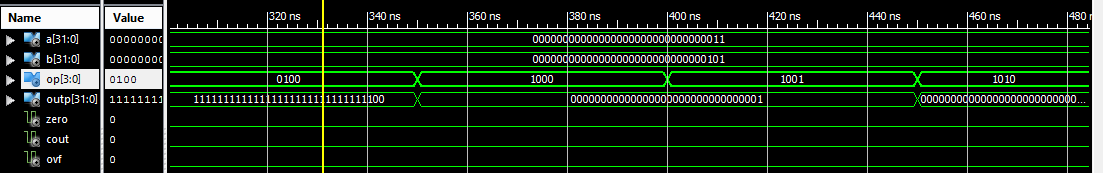
The act of subtraction is useful in the example to capture the conditions in which the prisoner is viewed. In this particular example the second operand accepts a borrowed bit so the result has a prisoner contribution.

Verification of logical operations needs boolean logic to confirm correct execution of operations (NAND,OR,NOT). Apart from the final result, there are no outputs for results (holding etc.) beyond arithmetic operations.

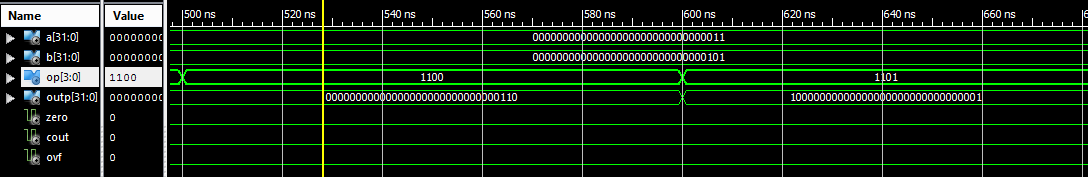


As input for the sliding and logic operations are kept:

A=..0011 , B=..0101



|  |  |  |
| --- | --- | --- |
| Shift | 000000000000000000000000000000011 | New MSB |
| Arithmetic right | 000000000000000000000000000000001 | Old A[31] |
| Logical right | 000000000000000000000000000000001 | 0 |
| Logic left | 000000000000000000000000000000110 | A[30] |



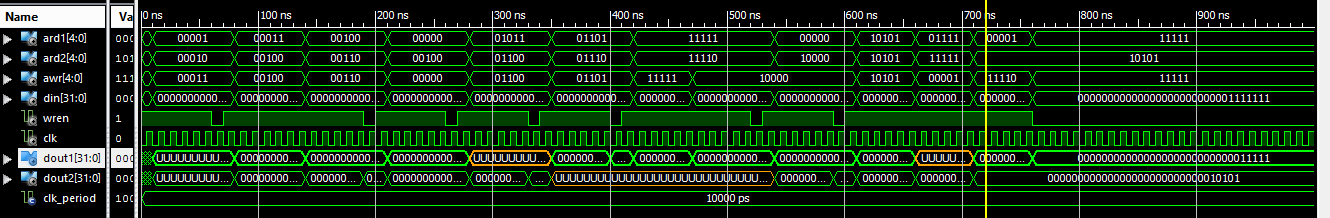
The results of the calculations are as expected in the operations of sliding (we observe the circular sliding that operator A undergoes for both right and left sliding)

|  |  |
| --- | --- |
| Rotate shift | 000000000000000000000000000000011 |
| Left | 000000000000000000000000000000110 |
| Right | 100000000000000000000000000000001 |

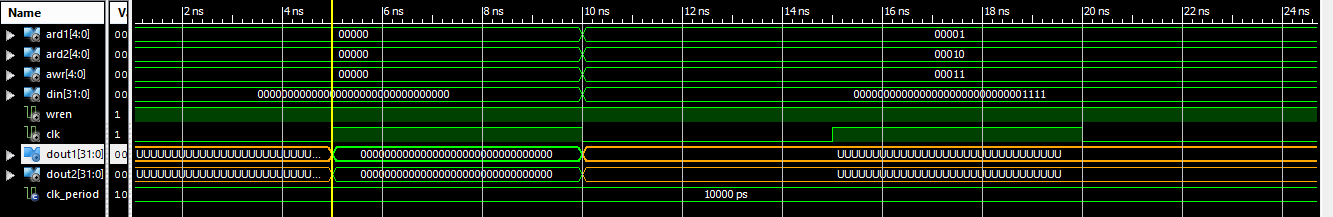
## Circuit 2

Register File

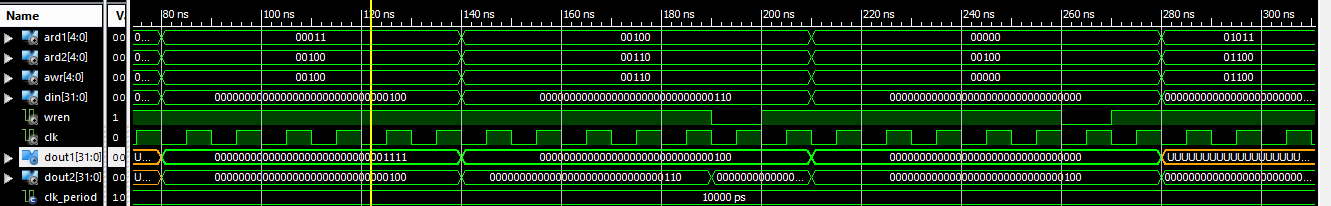
According to the results of the waveforms as well as the results of the design it is found that the diagram fully illustrates the functionality of a register file.



Assuming the read addresses point to register 0, the file outputs the constant value of the register. By varying the read addresses, the contents of the registers are determined to be uninitialized by any possible write. Continuing, the read and write addresses are alternated to expose the characteristics of the file. Registers initially have no content. The register set to write is 00011 with entry number 15.

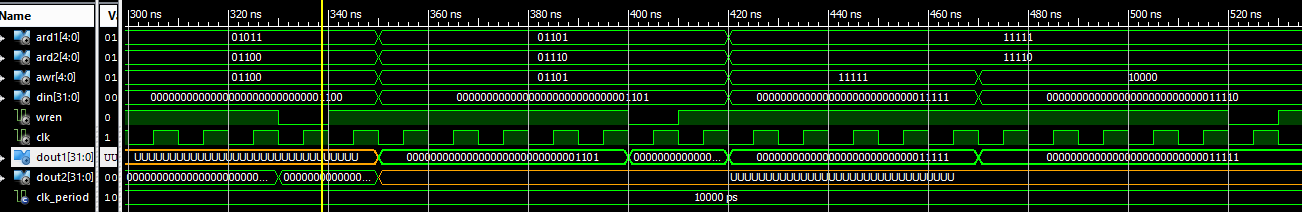


Addresses are being changed to verify registration. The observation in the waveform is distinct as the request to store the number 15 in register 3 succeeded while simultaneously writing register 4 with the corresponding number. At this point the characteristics of the specific design can be distinguished as the functionality of the Compare Module is worth noting (the equality between ard,awr leads to read the stored input). The address of the fixed register is reset for reading so that it can be compared with the results of the other registers, therefore the correct management of the decoder is also confirmed (The 4th register has the value 4, the register 0 the value 0)

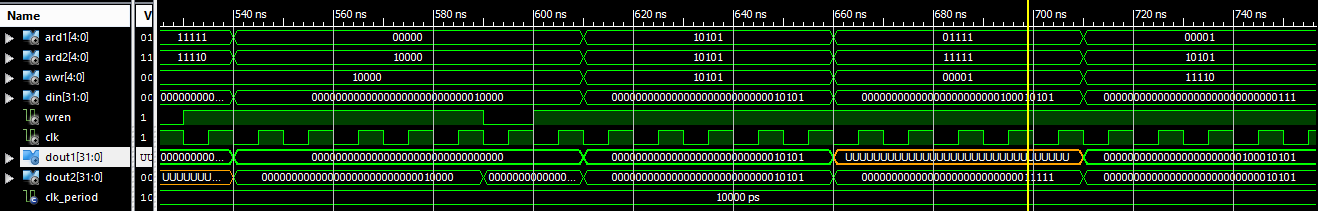


By repeating similar changes in the script for reading new registers, their empty content is detected while at the same time the writing of one of the two registers follows. Continuing follows reading addresses other than write address (ard1=11111 , ard2=11110, awr=11110)

The recording data is different each time.



Arriving at the end of the script, matching the two read addresses to the same register, the write is realized as the result is presented at the respective outputs.



To detect additional functions, the record data was varied to ..00001111111 while enable was disabled. The conclusion is captured in the waveform as the register has its previous entry (the value 000....0011111 was written to the register).

## Conclusions/Problems

Upon completion of this lab exercise, some notable takeaways are realized, such as familiarity and practice with hardware programming commands, an introduction to complex circuit design, and an understanding of the basic principles of the standard digital circuit unit (ALU) as well as the registry file.