Lab report

Digital Design (EDA322)

Group 9, Tuesday AM

Dennis Bennhage
Hampus Lidin

February 12, 2015

Contents

1 Introduction

(max: 1 page)

This part will introduce the reader to the report.

At the beginning, describe what the purpose of this lab report is. Then describe briefly what each section discusses and finally summarise the most important conclusions.

2 Method

The processor was designed in VHDL-code, using software QuestaSim (ModelSim) for simulation and XillinxISE for synthesizing.

2.1 Arithmetic and Logic Unit (ALU)

The ALU can take two 8-bit unsigned data words, and perform either of the following operations: addition, subtraction, bitwise NAND or bitwise NOT. It also has four indicators: *Carry*, *isOutZero*, *Eq* and *NotEq*. The *Carry*-bit sets to 1 when an overflow occurs (note that this indicator is only valid when performing addition). The other indicators are self-explanatory. When choosing what operation to perform on the inputs, you set the two-bit *operation* signal to one of the codes specified in Table 1.

We started out with implementing the data flow architecture for a *full adder*. A full adder takes three bits of input, where two of them are the numbers being added and the third one is a *carry-in*, as shown in Figure 1a. If at least 2 of the inputs are set to 1, a *carry-out* will be set. The sum output is the remainder of the addition of the three inputs. With this implementation, we were able to construct a *ripple carry adder*, using 8 full adders. The ripple carry adder could then be used for addition and subtraction in the final ALU-component. The reader can find the block diagram for the ripple carry adder in Figure 4 in Appendix A.

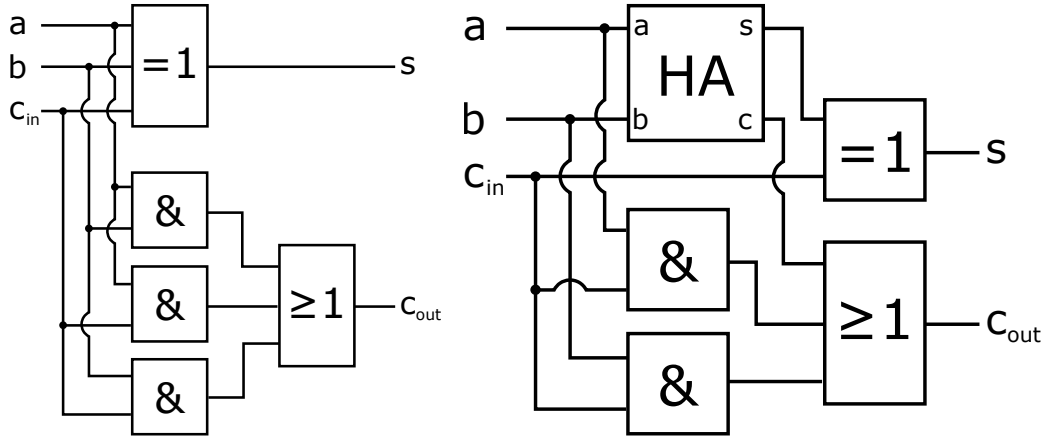
We had to implement additional circuitry for handling the subtraction of the

Operation	**Operation**
00	Addition
01	Subtraction
10	Bitwise NAND
11	Bitwise NOT

Table 1: The operation codes for choosing an ALU-operation.

two inputs. For that we first defined an internal signal *SUB*, that entirely depends on the first bit of the operation signal. We also defined an 8-bit internal signal, that depends on the *exclusive or* between the second ALU-input and the *SUB*-signal. In that way, when the *SUB*-signal is set, the *XOR*-gate behaves like an inverter. With the inverted input and the *SUB*-signal going directly to the carry-in of the first full adder, we have the *2 compliment* of the second input, which is just what is needed for the subtraction to work (see Figure 5 in Appendix A for the block diagram).

Finally, we needed to implement the data flow for a 4-1 *multiplexer*, whose purpose is to select the correct operation for the ALU-output. We did this in VHDL-code by matching the operation signal to it's corresponding operation, and then sending the signal to the output.



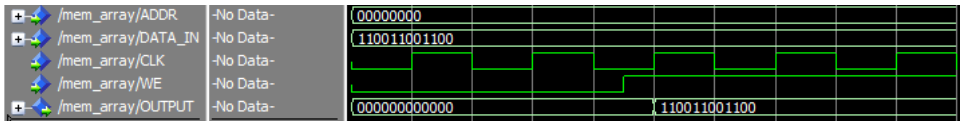
(a) A full adder. **(b)** A full adder using a half adder.

Figure 1: Figure 1a shows the data flow implementation of a full adder, and Figure 1b shows the alternative structural design using a half adder.

2.2 Top-level Design

The top-level design consists of a number of components, such as registers, memory units, ALU's and multiplexers. What we did was to design these components and then in the end, we connected them with each other in the data path.

The register is implemented using a D-flip-flop and a 2-to-1 multiplexer. The *WE* is the select signal for the multiplexer and it chooses what should be stored in the flip-flop, which is either the already stored data or the input data (*DATA_IN*). The register is implemented generically so that words of any size can be stored. The memory unit consists of an arbitrary number of registers, also using generic implementation. In the following picture, we can see what happens when we read and write data to a memory unit:



The memory is initially stored with the value '0' (or "000000000000" in binary) in address '0'. *WE* is also low, which means writing is disabled. When *WE* becomes active, the word "110011001100" is stored at the next rising edge of the clock (*CLK*) signal.

The bus is implemented using a 4-to-1-multiplexer rather than using tri-state buffers. In this way, no signals can overload, but there may be multiple select signals active. For this we have the error signal (*ERR*), that activates when at least two select signals are active. The select signal for the multiplexer and the *ERR*-signal are implemented using data flow, where minimizations of the boolean expressions have been made (see Figure 2).

2.3 Controller

Describe what you did in lab4. More specifically, show the *Finite-state machine* (FSM) of the controller by presenting the diagram you drew. Which design decisions did you make and why? Also include few waveforms, where you show that the controller runs correctly for some particular instructions using the provided testbench. Remember to always explain your design

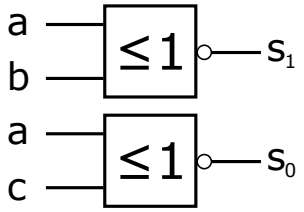


Figure 2: The minimized gate logic for the select signals of the bus multiplexer.

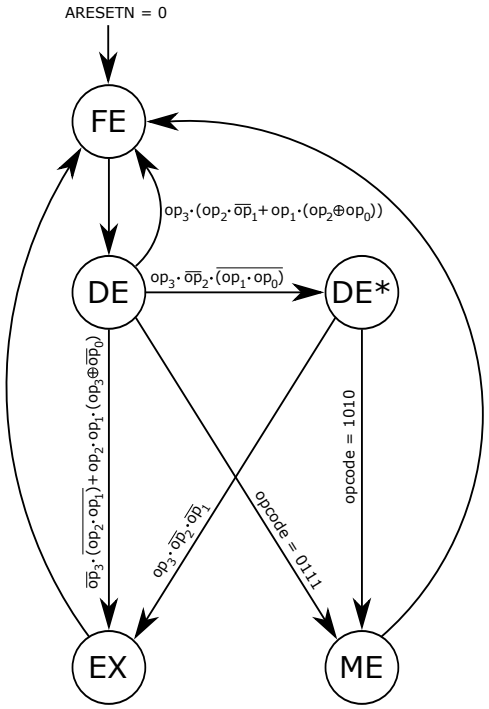


Figure 3: The *Final State Machine* of the processor's controller.

choices and mention any assumptions. Finally, make use of figures and tables.

2.4 Processor's Testbench

(max: 2 pages)

Describe what you did in lab5. More specifically, describe how you made the testbench to verify that your processor design was functionally correct. For example, you can specify how you generated inputs to the processor during the testing, how you were reading the expected outputs and how you compare the expected outputs with the actual outputs. Also mention if your processor design was working correctly from the beginning and if not describe how you backtrack the bugs. Remember to always explain your design choices and mention any assumptions. Finally, make use of figures and tables.

2.5 ChAcc on Nexys 3 board *(Optional)*

(max: 2 pages)

Describe how you verified the correctness of your FPGA implementation. Note that the code that is executed on the implementation is the same code used for testing in Lab 5. You should compare sequences of values on various signals observed on the seven-segment displays to values seen in Modelsim simulation of the design. Please include in the report the sequence of program counter (PC) and display register values you observed during a successful execution on the FPGA.

2.6 Performance, Area and Power Analysis *(Optional)*

(max: 2 pages)

To be announced in the Lab7PM.

3 Analysis

(max: 1 page)

Summarize your results after performing all the labs (2, 3, 4 and 5).

Mention and discuss interesting findings and observations, as well as diffi-

culties in completing some of the tasks of the four last labs.

After looking at your results, draw conclusions and describe briefly the learning outcome, that is what have you learnt by performing these labs?

A Figures

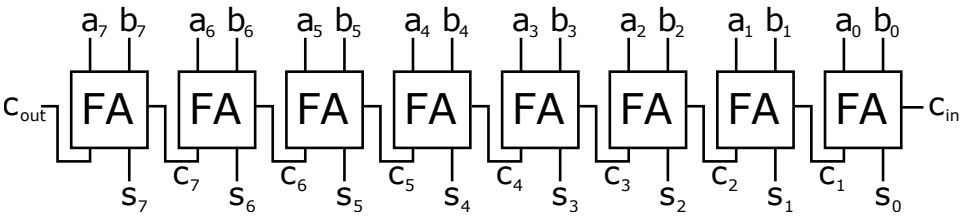


Figure 4: A structural implementation of an 8-bit ripple carry adder using eight full adders.

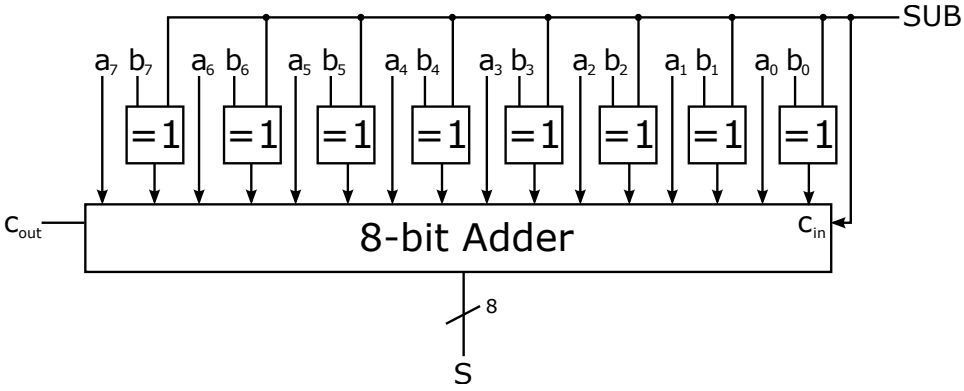


Figure 5: Additional circuitry that is needed in order for the subtraction operation to work.