# Assignment 2. Digital electronics

By Nicolai Rode & Jesper Bertelsen

Indholdsfortegnelse

[Assignment 2. Digital electronics 1](#_Toc156832760)

[Problem 2](#_Toc156832761)

[Declarations 2](#_Toc156832762)

[To do 2](#_Toc156832763)

[The making 2](#_Toc156832764)

[Full Adder 2](#_Toc156832765)

[The counter 5](#_Toc156832766)

[Issue 6](#_Toc156832767)

[The end 6](#_Toc156832768)

[Revisiting the code 7](#_Toc156832769)

[8-1 multiplexer 8](#_Toc156832770)

[New entity structures 9](#_Toc156832771)

[Discussion 14](#_Toc156832772)

[Conclusion 15](#_Toc156832773)

[Author’s contributions 16](#_Toc156832774)

## Problem

We want to make a 4-bit synchronous up / down counter with reset, load and enable features.   
During the making we want to document our progressions and issues that we stumbled upon.

## Declarations

As me and Nicolai both uses Mac and are simulating vivado using Windows VM and Linux interpreter, not every feature works as intended. As simulations doesn’t work, we will use the board to test our code on.

## To do

Make the

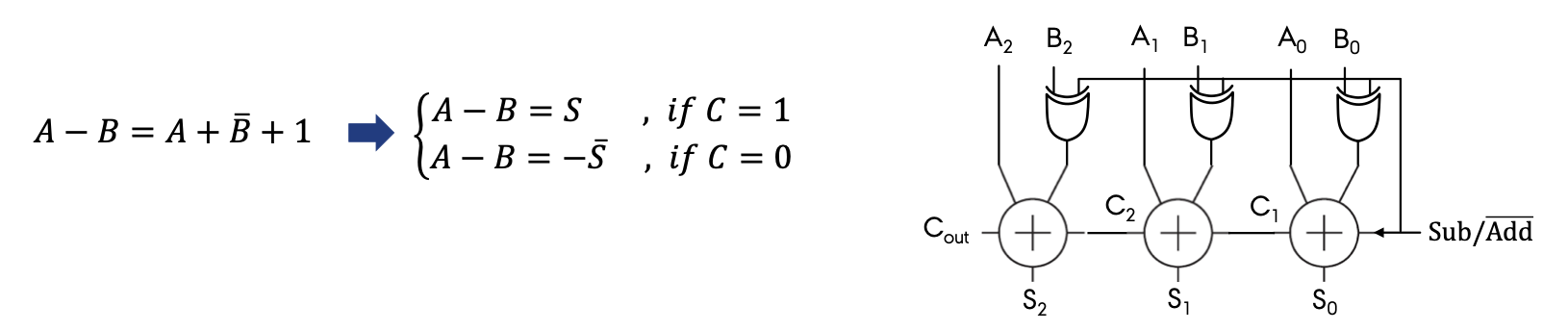
* Full adder
  + Make the up / down feature
* Clock gate
  + Make enabler
  + Make reset
* Load
  + Q\_n-1 / Load data

## The making

### Full Adder

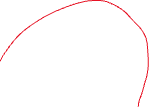
In the full adder we want to check whether it’s supposed to count up or count down.

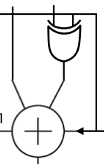
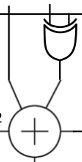
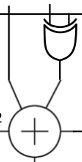
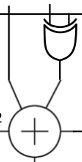
The design we follow are:



C indicating whether it’s a count up or count down.

For the program we need to make some signals







So the signals is the ones inbetween gates:

As vectors.



For our matter, we want to add a increaser value for every clk.

For the first adder:

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, lilla/violet

Automatisk genereret beskrivelseFor the rest we do:

So we can a for loop the rest.

We are using structural architecture to reuse the full adder again and again. The code of the full adder is like so, following the design above.

The sum and carry can be derived from our knowledge about gates and the knowledge of how we do addition and subtraction.

## The counter

This counter works with us sending a signal for it to update, which we say is the clk. The clk should be either rising or falling edge triggered, resulting in one update pr. click. Both edge triggered results in two updates pr. click, which we aren’t interested in. Which edge the signal is triggered is for us redundant, so we just choose to make it rising edge triggered.

We’ve made the structure of the full adder, but the clock gate and load we just implement by code and makes VHDL do the trick for us:

Our code in psudo:

Is reset = True?   
 Do we have a load? Then reset to load.

No load -> Then reset to either 0000 or 1111 depended on which way we count.

No reset ->

Is the counter enabled & the clk triggered? If yes, then:

full\_adder\_logic(Q, increase value, carry, Sum, Carry out)

For starters we need to declare all the variables needed:

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede

Automatisk genereret beskrivelse

And the behavior of the up down counter:

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Automatisk genereret beskrivelse

### Issue

We have done a lot of trial and error with the up down counter, but haven’t gotten familiar enough with VHDL to make the code function.

Trying to trouble shoot it, we didn’t really get much wiser, it just seemed like whatever we did an error would be there, even though we were pretty sure it had no reason to give an error. Even old code, that has worked in other documents failed in this design.

It’s frustrating because it should work.

### The end

The code still doesn’t work syntax wise, which might just be a bug in VIVADO. The code looks now like this:

Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, Syren/lyslilla

Automatisk genereret beskrivelse

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype

Automatisk genereret beskrivelse



Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Automatisk genereret beskrivelse



Switching away from the carry as a vector style as I realised that the carry is just a temporary value, only needed for the next bit. So c is just a std\_logic value that is mutated a lot.

In the end I make the

The code should be fine, but somehow the entity work just doesn’t work.

The full adder design introduced the opportunity to make the increase value more than 1, so we introduced the inc variable in the up\_down counter. This just gives flexibility. In our constraints we have set the JD pmod ports to be the increasement value.

### Revisiting the code

A thought that came to mind is, that in our architecture we use both behavioural- and structural modelling. Instead of nesting the enable and clk signal, we use behavioural logic inside the full adder and uses a multiplexer to select between Q, Load and reset values.

We drew the block circuit for visualization:

Et billede, der indeholder tekst, whiteboard, håndskrift, indendørs

Automatisk genereret beskrivelse

### 8-1 multiplexer

To select what signal to be extracted to the next count we use an 8 to 1 multiplexer with 4 possible input signals:

Fa = the output of the full adder

LD = load data

“0000” = the reset signal for up count

“1111” = the reset signal for down count

For selection we use a 3 bit selector with

|  |  |  |  |
| --- | --- | --- | --- |
| Reset | Load | Up-Down | output |
| 0 | 0 | 0 | Fa |
| 0 | 0 | 1 | Fa |
| 0 | 1 | 0 | LD |
| 0 | 1 | 1 | LD |
| 1 | 0 | 0 | “0000” |
| 1 | 0 | 1 | “1111” |
| 1 | 1 | 0 | LD |
| 1 | 1 | 1 | LD |

Et billede, der indeholder diagram, tekst, skærmbillede, linje/række

Automatisk genereret beskrivelse

### New entity structures

#### Full adder

(New old full\_adder) For the full addder we wanted to implement behavioural code for enable and the rising edge of clk, our new architecture of this is.

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype

Automatisk genereret beskrivelse

This is made in a behavioral modelling way, assigning the sum and carry.

Our reason to revisit the full\_adder again was to be able to make a four-bit adder using this structure. This still got an error, it might have to do with mixing behavioral and dataflow. We took another approach.

( New new full\_adder) This approach uses only behavioral, where we describe all the different combinations.

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Automatisk genereret beskrivelse

This worked so this is what we ended up using for the structure of the full adder.

#### 4 bit full adder

There isn’t anything new about the 4-bit full adder. This still doesn’t work.

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype

Automatisk genereret beskrivelse

#### Multiplexer

Code:

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Automatisk genereret beskrivelse

the code functions as a 8 to 1 multiplexer where we only have 4 inputs that share the 8 inputs.

We used select as a logic vector of 3 bits where we ofc have the bits representing reset load and up/down count. If we look at the simulations and compare it to the truth table we can see that they have the same output

The simulation of it:

Et billede, der indeholder skærmbillede, Multimediesoftware, Grafiksoftware, software

Automatisk genereret beskrivelse

We encountered some issues regarding simulations of the parts we coded. The biggest issue was that Vivado upon simulation wouldn’t respond 99% of the time. It’s a miracle that we could run this 1 simulation of the multiplexer we implemented.

Below you can see The simulation of the a code that wouldn't work. the program kept not responding after we pressed "run simulation." As we didn't know whether it was the code or the computer, we used a code from the internet that we knew would work. However, it did not work, and the program didn't respond.

Et billede, der indeholder skærmbillede, tekst, software, Webside

Automatisk genereret beskrivelse

#### The counter

The final code is then:

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, algebra

Automatisk genereret beskrivelse

So what the counter does is call the four bit full adder, with Q being one of the 4 bit numbers added, the inc (Increaser value) is then other 4 bit number. U\_D is being pointed, so that the entity knows whether to count down or to count up. An enable and clk then tells the entity when to operate.

The signal fa is then being assigned the new value.

For the mux entity it receives the input(0) value, fa, the increaser input(1) value, the reset input(2) value, the reset input(3) value. A selector of ( Reset, load, u\_d ) has then been given, and the output is the new Q value.

## Discussion

We first got into this project not thinking that structure were an important element of VHDL. We now know better, that VHDL is an unforgiving code language, which needs order for it to work.

We didn’t think much about mixing architecture styles from withing one architecture, which could have given us a better overview, if we just new how strict our styles should be, more structure should have been made.  
If we were just to get the entity calls to function, we could probably have gotten the project to work.

We could have simplified this project a lot, by just putting our structure into one behavioral block, but we took upon the challenge of making different components, so that our code would act more like the hardware level.

In terms of the simulation, we still haven’t figured out exactly why the program wouldn’t respond most times. As far as we know, macOS isn’t a supported OS, so we used a VM with Windows 11 to download Vivado and try to simulate the different coded parts. However, Vivado, as mentioned previously in the assignment, would tend to not respond and crash upon starting the simulation. Therefore, we did some research and found the system requirements for Vivado. Vivado needed 16 GB of allocated RAM to the VM to run optimally. As the computer has a total of 16 GB of RAM, and the VM runs in parallel with the computer, you can only allocate half. Therefore, our thought is that the program would tend to crash and freeze 99% of the time because the CM only could get 8GB. This made it especially hard to debug the code and make progress in the creation of sufficient code that runs.

## Conclusion

We’ve shown the problem being resolved whilst writing this assignment. We’ve made the code, stumbled upon issues and have had success with correcting some issues and failed with correcting other issues, which still stands. We’ve switched from the belief of putting a full adder as a component inside a behavioral block, mixing between behavioral and structural architecture modelling, to making a full adder, 4bit\_full\_adder and a mux choosing between Q, Load data and reset values. The new way of designing our counter corresponded more to what we would have seen on the hardware level. Even though we didn’t get the program to run, we found out some things about VHDL in Vivado, that we previously didn’t know about.

Furthermore, our journey through the simulation phase revealed challenges that hindered the smooth execution of our program. The persistent issue of the program unresponsiveness remained an problem throughout the assignment.   
Our initial attempt to run Vivado on macOS where we used a Windows 11 virtual machine to run Vivado and run simulations. However, using a VM introduced its own set of complications which in the end contributed to hindering our success.

## Author’s contributions

Nicolai:

* Multiplexer
  + Code
  + design
* Simulations
  + Multiplexer simulation
* Discussion
* Conclusion

Jesper:

* Multiplexer
  + Code
  + design
* Full adder
* Combining code to create counter.
* Discussion
* Conclusion