**ECEGR 2220: Microprocessor Design**

**Spring 2018**

**LAB 3 REPORT**

**Name: Don-Thuan Le**

**Performed by:**

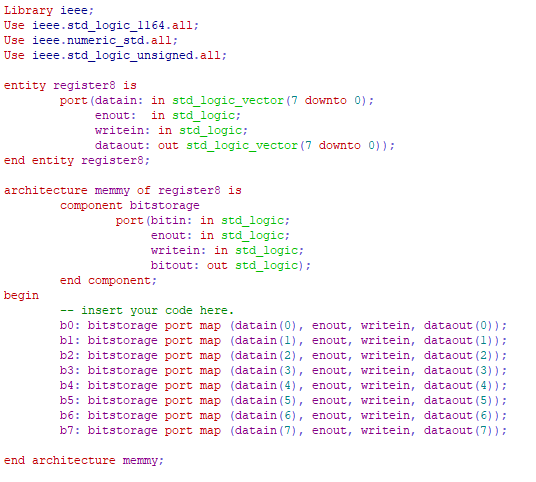
**Don-Thuan Le - Thanh Nguyen – Lauren Molina**

**Date of Report: 05/05/2018**

**SEATTLE UNIVERSITY**

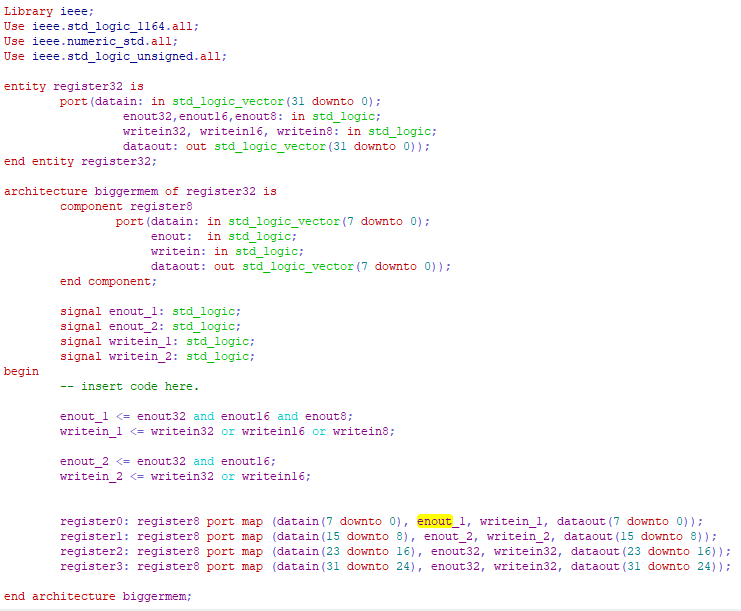
**Department of Electrical and Computer Engineering**

1. **An 8 bit register called register8 from components of the memory cell in register.vhd**



We mapped all the input and output ports of the component **bitstorage** to the register8.

1. **A 32 bit register called register32 built from components of register8.**



We created 4 signals: **enout\_1, enout\_2, writein\_1, writein\_2**.

For enout, we have output only when enout = 0.

We can register 8 bits when one of the enout32, enout16, enout8 is ‘0’. So, we decided enout\_1 = ‘0’ when one of enout32, enout16 and enout8 is ‘0’ by using logic AND between enout32, enout16 and enout8. So, with enout\_1 = ‘0’, we can register 8 bits.

We can register 16 bits when two out of the three enout is ‘0’. So, we decided enout\_2 = ‘0’ using logic AND between enout32 and enout16. So, with enout\_2 = ‘0’, we can register 16 bits.

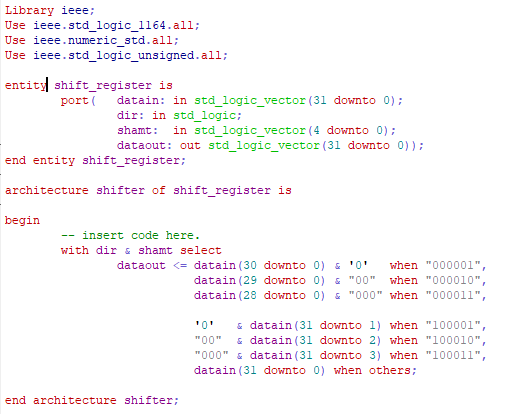
For writein, it will be enable when writein = ‘1’.

We decided writein\_1 = ‘1’ when one of writein32, writein16 and writein8 is ‘1’ by using logic OR between writein32, writein16, writein8.

We decided writein\_2 = ‘1’ when two out of 3 writein is ‘1’ by using logic OR between writein32 and writein16.

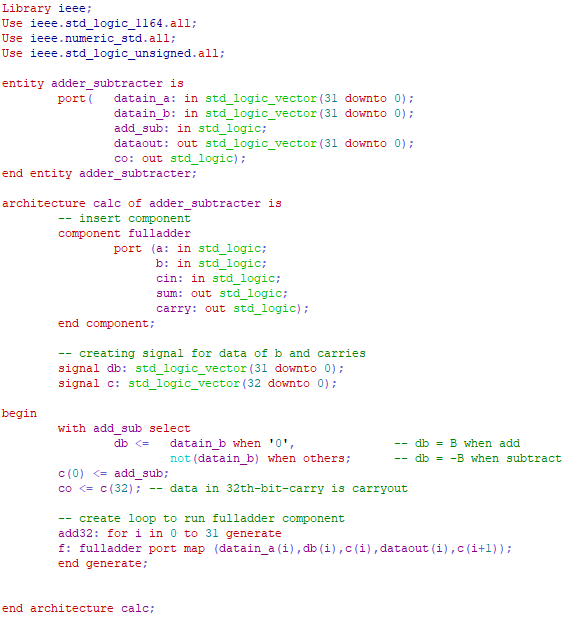
For register0 and register1 we used the signals we created in the port map. But for register2 which account for the 8 bits that would add up to 24 bits, since intervals using binary is not able to account for 24, we used enout32 and writein32 in both register2 and register3.

1. **A 32 bit shift register that can shift up to 3 bits in either direction.**



For the shift register, we concatenated the input for the direction of the shift and the input for the shift amount and utilized a “with-select” statement. When the input for dir is “0,” then the data inputted will be shifted to left and the bits that no longer have values will be filled with zeros. When the input for dir is “1,” then the data inputted will be shifted to the right and the bits that no longer have a value will again be filled with zeros. In any other scenario, there will not be a shift.

1. **A 32 bit adder/subtracter.**



We created two signals **db** and **c**.

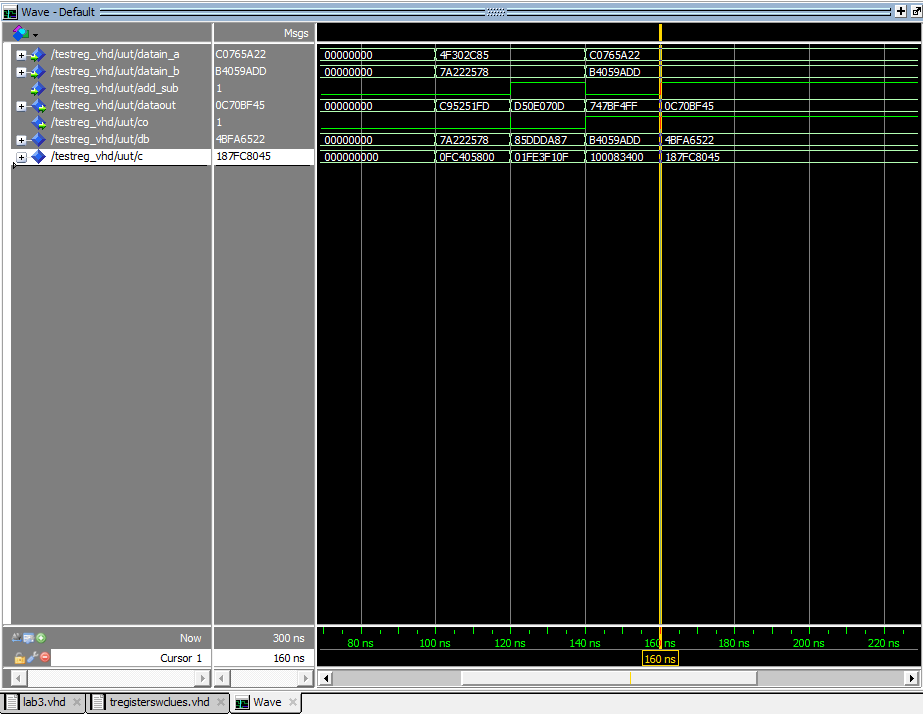
**db** was used for determining the value to add to A. If **add\_sub = 0**, then we have **A+B**, so **db** would be **datain\_b**, if **add\_sub = 1,** then we have **A-B**, so **db** would be **not(datain\_b).** to do this task, we used “with-select” statement.

**c** is used to store the carries. **c** has **33 bits** because we counted an additional bit which was the first bit of c. This takes **add\_sub** value, which means: it is **0** when **add\_sub is 0** (A+B), or **1** when **add\_sub is 1** (A-B).

**co** takes the data in the **32nd bit of c (carryout)**. if it is a “1”, there is overflow; otherwise, there is not overflow.

We created a loop to call the component fulladder 32 times. This would continuously do addition for 32 bits of datain\_a and B and give out the result.

1. **TESTING**

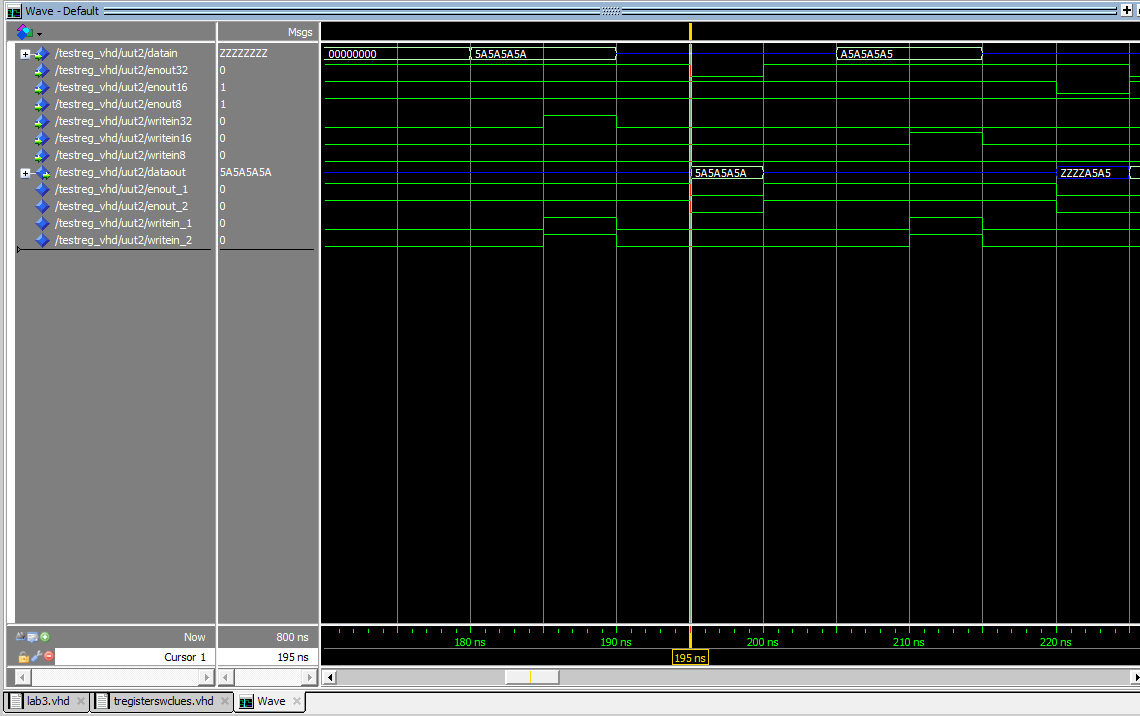
* Test the adder\_subtracter first

From 100nS to 120nS, dataout was 0xC95251FD

From 120nS to 140nS, dataout was 0xD50E070D

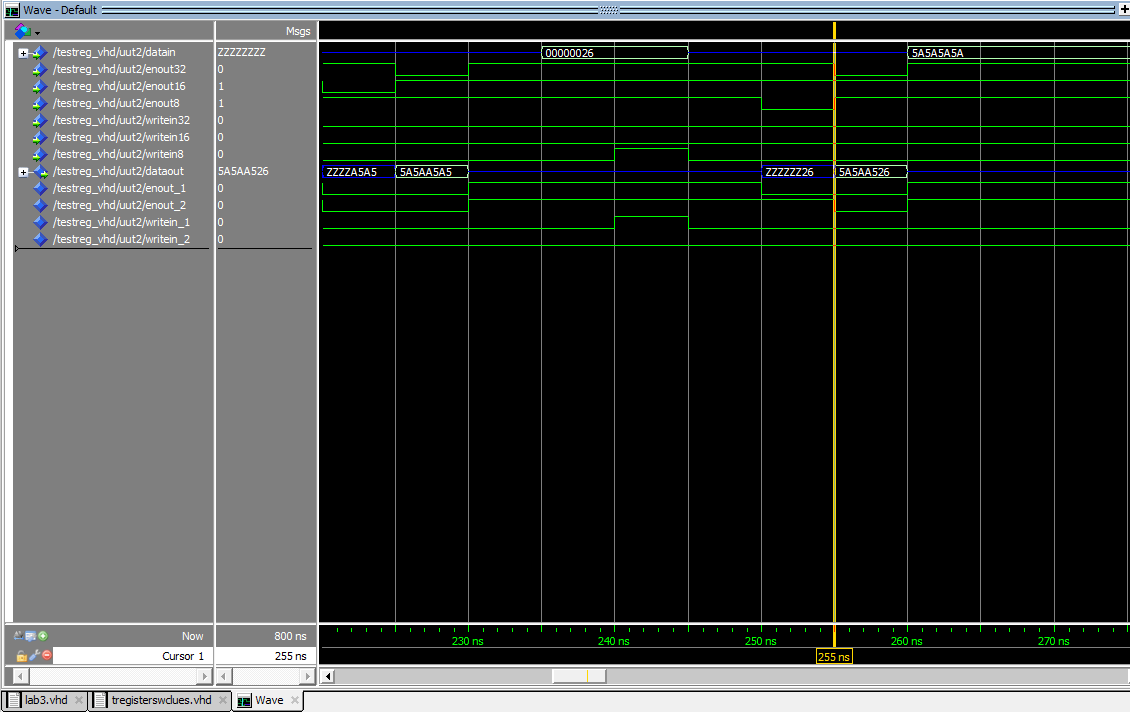
From 140nS to 160nS, dataout was 0x747BF4FF

At 160nS, dataout was 0x0C70BF45

* Test the register32 

From 195nS to 200nS, data was 0x5A5A5A5A

From 220nS to 225nS, data was 0xZZZZA5A5

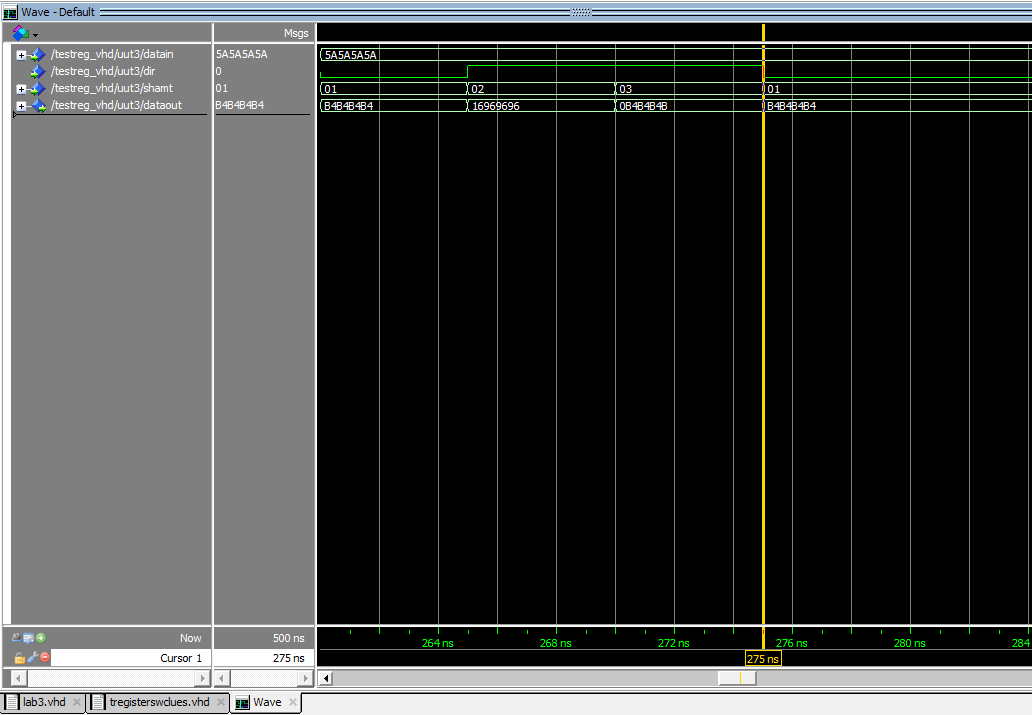


From 225nS to 230nS, data was 0x5A5AA5A5

From 100nS to 120nS, data was 0xZZZZZZZZ

From 250nS to 255nS, data was 0xZZZZZZ26

From 255nS to 260nS, data was 0x5A5AA526

*  Finally test the shift\_register

Shifted left by 1 bits, the data was 0xB4B4B4B4 at 260nS

Shifted right by 2 bits, the data was 0x16969696 at 265nS

Shifted right by 3 bits, the data was 0x0B4B4B4B at 270nS

Shifted left by 1 bits, the data was 0xB4B4B4B4 at 275nS

* Link for our github:
* Link to our master branch:

<https://github.com/SU-ECEGR-2220/AVOCADOS/tree/master/lab%203>

* Link to individual branches:

Thanh: <https://github.com/SU-ECEGR-2220/AVOCADOS/tree/Thanh/lab%203>

Don: <https://github.com/SU-ECEGR-2220/AVOCADOS/tree/Don/lab%203>

Lauren: <https://github.com/SU-ECEGR-2220/AVOCADOS/tree/Lauren/lab%203>

* Note:
* The file that we built from registers.vhd is named lab3.vhd. You can find this file under master branch or using the direct link:

<https://github.com/SU-ECEGR-2220/AVOCADOS/blob/master/lab%203/lab3.vhd>

* Each individual built the program then saved and test on his/her own branch first. Then we came up with the best version of the program and pulled it to the master branch. You can find the contribution of each member towards the project in the individual branches. Thank you!