CSC3050 Project 3 Report: Design of an ALU

杨宇凡 117010349

**Summary:**

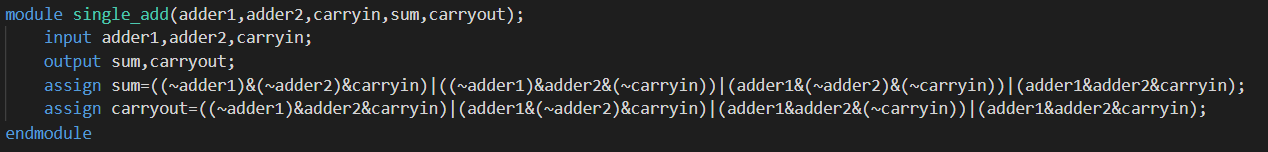
The method I do this project is a little bit complicated. I don’t use any grammar skills (like for loop, if adjustment) but build all the electricity modules, and use these basic modules to make up the whole ALU. The reason why I do this is because hardware design requires us to think like hardware instead of software. If you just use these grammar skills, it’s no difference of writing C languages program instead of a hardware. This project requires us to support about 20 MIPS functions, and I write each function as a Verilog module. Besides, I also write a lot of basic modules, like single adder, comparator, single divisor and so on. These basic modules help a lot for building these complicated modules.

Since the electricity map is very complicated, the running of testbench will take some time (about 30 seconds for each module’s test, the whole test methods about 10 minutes).

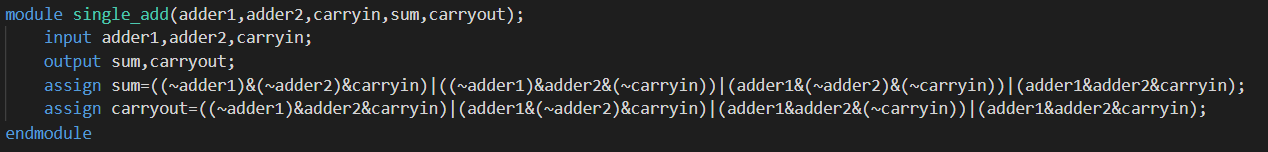
**Program Logic:**

Write different MIPS’s functions in different modules, and finally combine these modules together in the ALU. Therefore, these modules can get the input from the ALU, and transfer the output to the ALU. Then ALU needs to select the proper output according to the opcode. To do this, the multiplexers are designed as well. After the selection, ALU is responsible for transferring the data to the corresponding registers or other data structures.

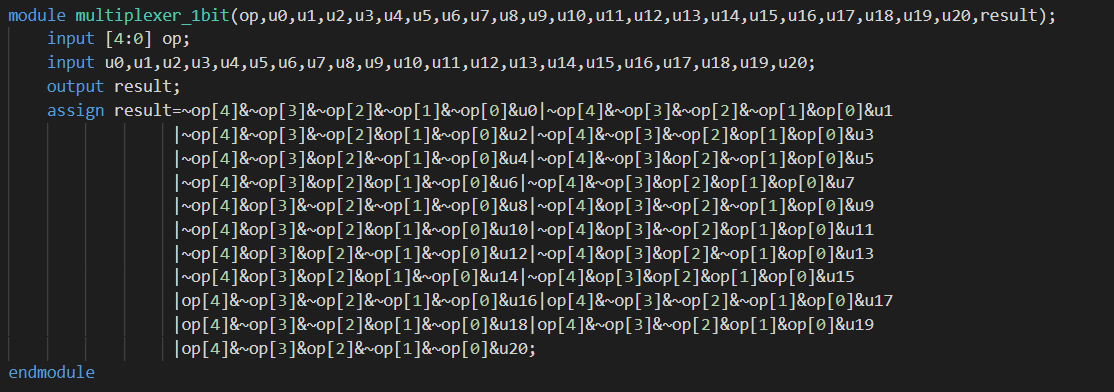
**Basic Module Instructions:**



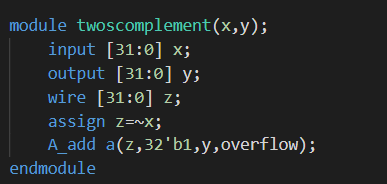
This module is used for doing one bit add operation and output the carry out.



This module is used for doing one bit comparing, which can be applied into making up more complicated comparator. The comparator is very use for the make up of some function modules. DIV is the representative.



This module is called multiplexer and it is responsible for select the proper data according to the opcode, which is very important for the ALU.



Two’s complement module is for doing the two’s complement, which is very useful in the subtraction operation.

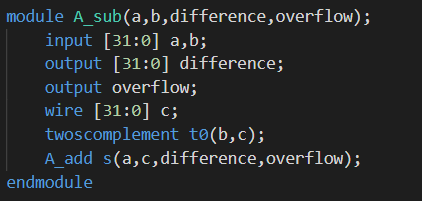
**Function module instructions:**

In this part I will choose some representative modules to introduce.

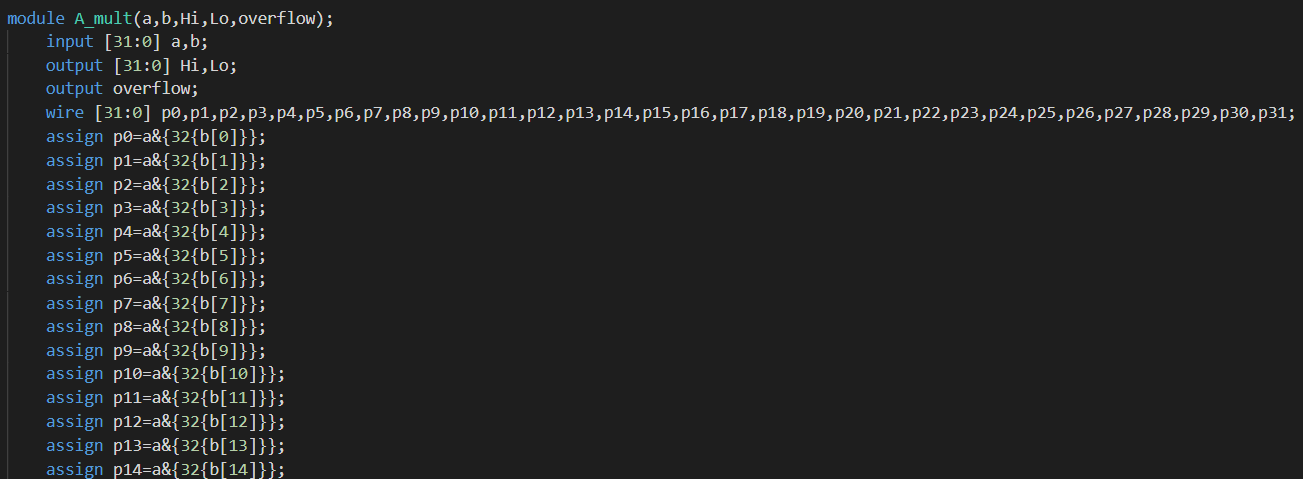


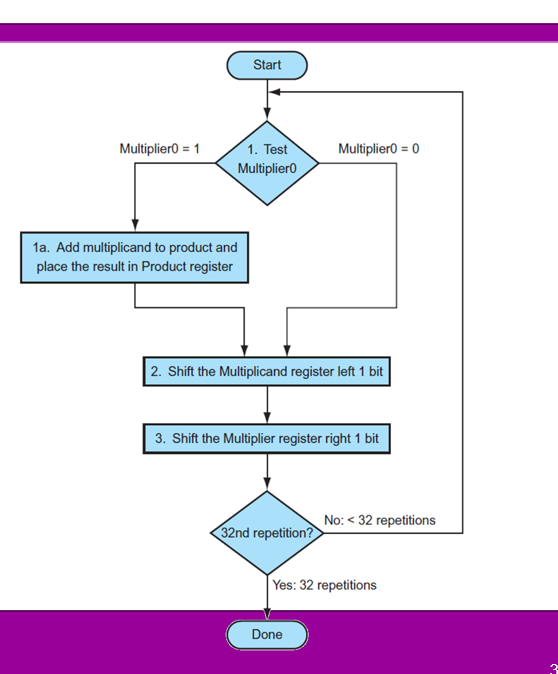


Add module is made up of the single adders one by one. The overflow check method is shown above.

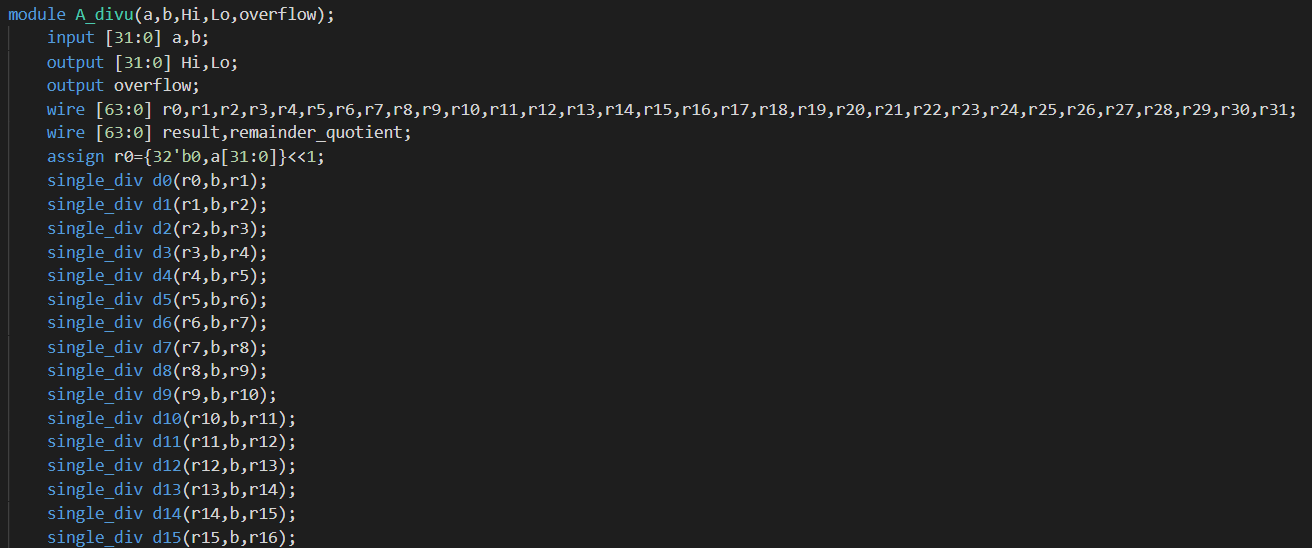


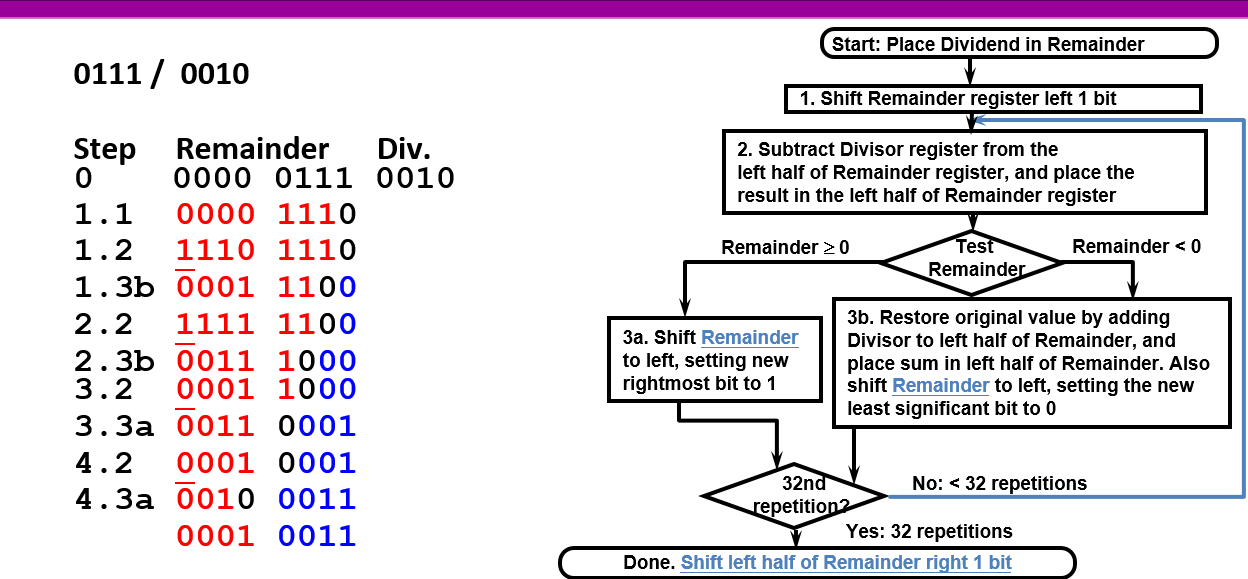
The sub operation consist of two’s complement module and the add module.





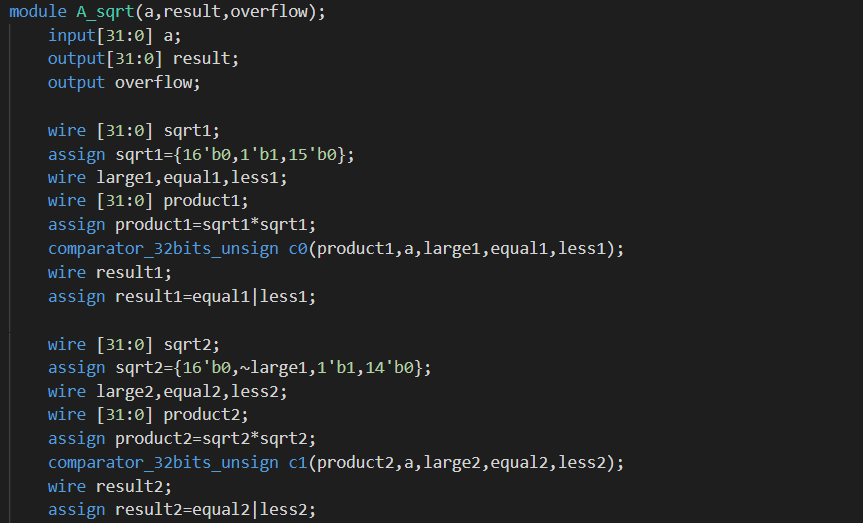
This is the multiple unsigned operation module. To build it, I use the method showed above. It is kind of complicated.





This is the division unsigned module. The method is shown above which uses shifting, comparison and loop operations. Therefore it is kind of complicated to realize with the electricity map. Before the whole module, the single division module is built to simply conducted the loop.

Besides, the unsigned operation of add and sub ignore the overflow operation. The signed operation of division and multiply uses the sign checking.



The design of sqrt is very complicated. The basic logic is to assume a correct answer and then checking. This method is easy to realized by software programming, but very difficult for hardware design. However, this time I still use this logic.

**Three flags:**

* Overflow

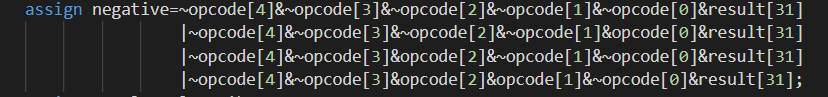
Overflow flag is check by each module. The method for each module is different. Many functions ignore the overflow, like most unsigned functions.

* Zero:



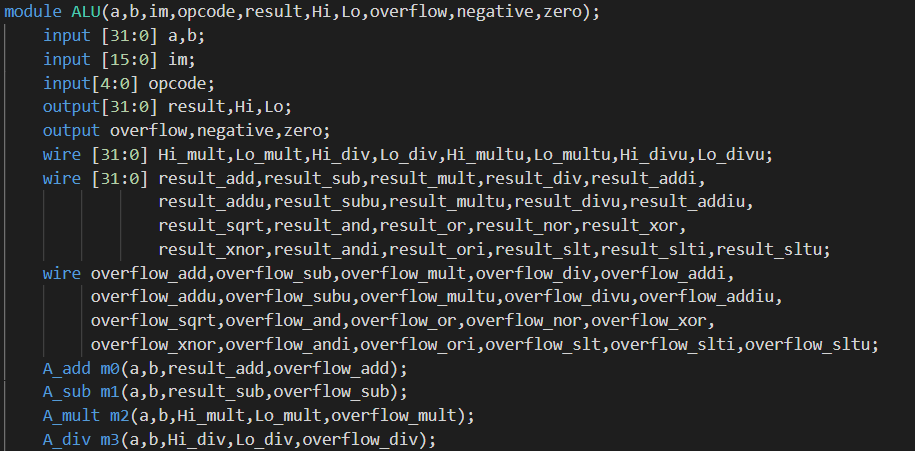
Zero flag is checked in ALU by the above logic, doing the or operation inside the result.

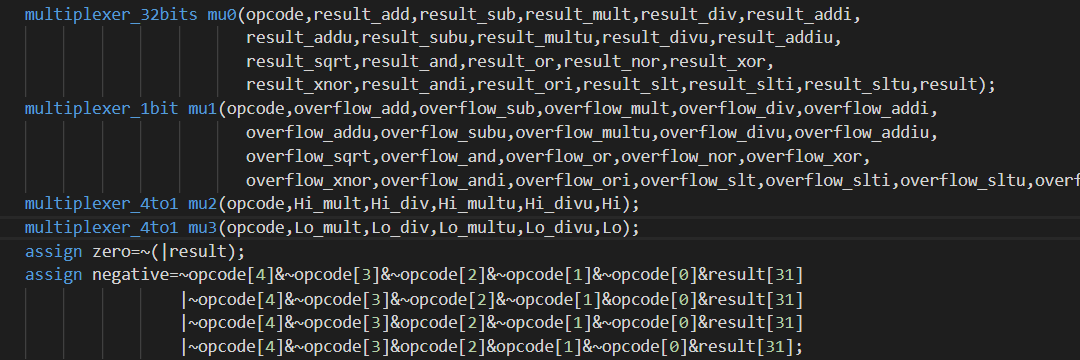
* Negative:



Negative flag is done by the above logic, which will be only activated in some specific functions (opcodes) like signed adder, unsigned sub.

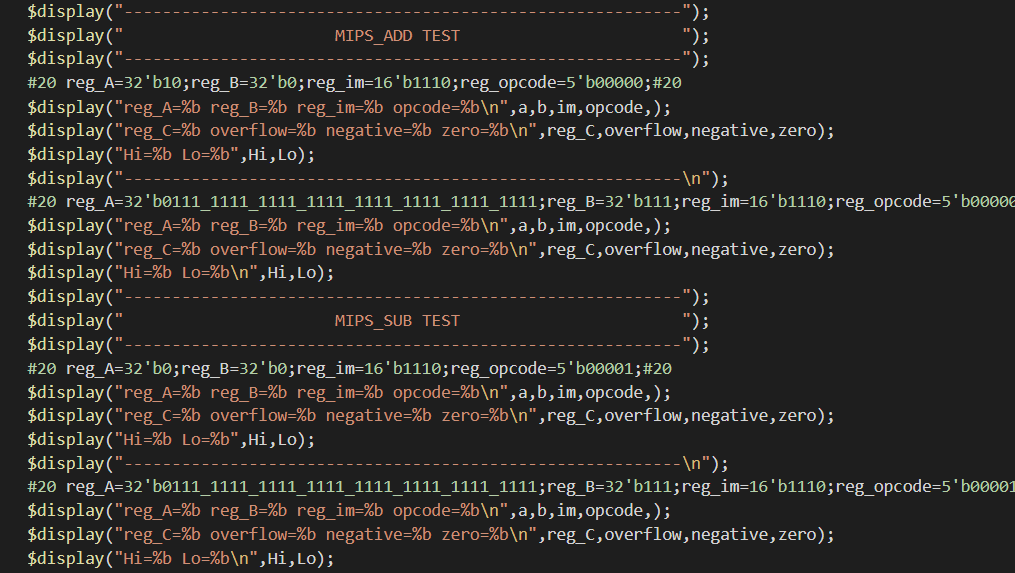
**ALU:**





ALU is shown above. It uses the multiplexer to select the right output and transfer it to the corresponding register or other data structures.

**Test**

****

The test module is shown above. Each instruction consists of two instances. All the related register, opcode, three flags, Hi and Lo will be shown on the test result. Following are the example output:

