Alihan Mahanoglu

2019-81292

Janghoon Han

2008-12407

2019.04.12

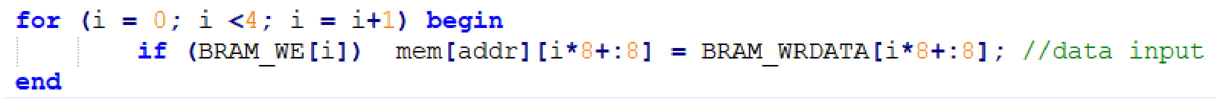
Hardware System Design Lab #5

The purpose of this lab session was to implement a block RAM module and a Processing Element which is basically floating IP for specific use, and to acquire a simulation result by implementing their test benches.

BRAM

The way BRAM module should work is as follows: The user should be able to read (in 2 clock cylces) and wrıte (in 1 clock cycle) data to BRAM. In the lab for the testing part, it was asked from us to initialize 2 BRAM modules, copy the content within input.txt to memory within BRAM1, read the data in BRAM1's memory and then write it to BRAM2's memory. Considering that it should take 2 cycles to read data and 1 cycle to write data, whole process should take 3 cycles, meaning that from the moment an address is assigned to BRAM_RDDATA1 it must take 3 clock cylces to see a data written in internal memory of BRAM2.

There are key input signals for BRAM module to work in the desired way. BRAM module is initiated with BRAM_EN signal meaning that all data reading writing can be achieved only when BRAM_EN is 1. When BRAM_RST signal is 1 BRAM_RDDATA value is assigned 0. In order to read data from BRAM BRAM_WE must be assigned 4'b0 and for each "1" bit (representing the 4 8bit segments in BRAM_WRDATA) in BRAM_WE, the 8bit data in BRAM_WRDATA is coppied to internal memory.



```
for (i = 0; i <4; i = i+1) begin
        if (BRAM_WE[i])  mem[addr][i*8+:8] = BRAM_WRDATA[i*8+:8]; //data input
end
```

i*8+:8 means starting from i*8th adress to next 8 steps. The code provided in the skeleton for this line had errors.

BRAM TestBench

The test bench implementation only included initialization of BRAM modules, assigning BRAM_WRDATA2 to BRAM_RDDATA1 so the data read from first BRAM would be directly copied to data written to second BRAM and finally increasing the input adress values by 4.

Result for BRAM

The code did not work as expected.

Processing Element

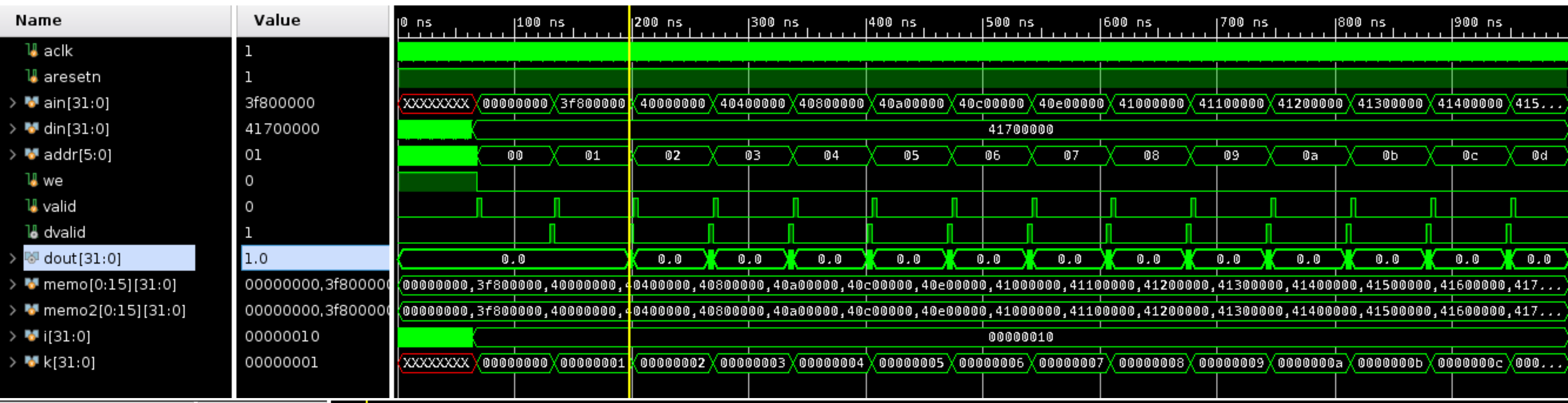
Within implementation of PE floating point IP is initialized. Inputs and outputs of the module is assigned to inputs and outputs of the floating point IP. With a WE signal the local memory is assigned din – the serial input variable – or it is fed to input of the IP. In each positive edge of PE clock, temporary Sum is equated to dout. A reset signal is also used to assign dout 0 when the signal is 0. Another important point is that when dvalid – the validity info of the output of IP – is 0, dout is assigned 0 because the value in dout does not make any sense when dvalid is 0.

Processing Element TestBench

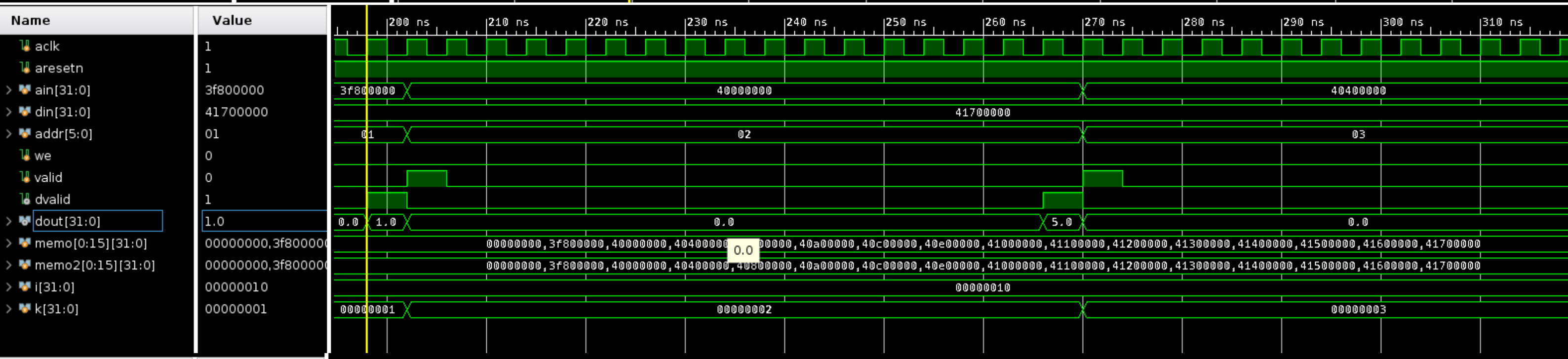
First valid was set to 0 and we was set to 1 in order to store data in din.txt to processing elements internal memory. Then we was set to 0 and valid was set to 1 because writing process ended, and now is the time for calculation to start. The problem in this part that for each step of calculation an accumulation value is required by IP meaning that each step of calculation needed the result of the earlier step. Therefore when serial inputs – ain – was being fed to the PE the test bench needs to wait until making sure that the result of earlier step is valid.

This is implemented by toggling valid signal when dvalid signal is 1 and 0 after an input is given.

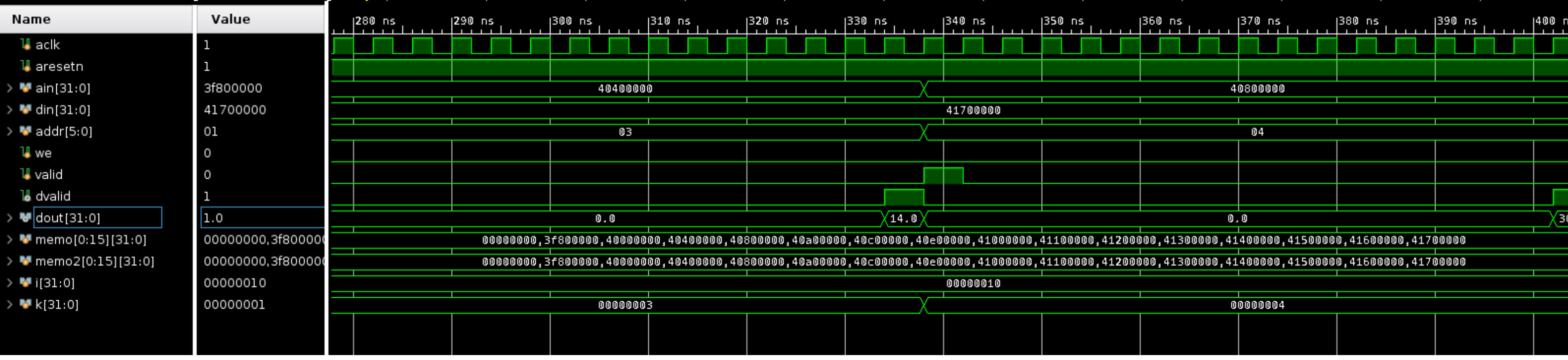
Results for PE



Name	Value
aclk	1
aresetn	1
ain[31:0]	3f800000
din[31:0]	41700000
addr[5:0]	01
we	0
valid	0
dvalid	1
dout[31:0]	1.0
memo[0:15][31:0]	00000000,3f800000
memo2[0:15][31:0]	00000000,3f800000
i[31:0]	00000010
k[31:0]	00000001



Name	Value
aclk	1
aresetn	1
ain[31:0]	3f800000
din[31:0]	41700000
addr[5:0]	01
we	0
valid	0
dvalid	1
dout[31:0]	1.0
memo[0:15][31:0]	00000000,3f800000
memo2[0:15][31:0]	00000000,3f800000
i[31:0]	00000010
k[31:0]	00000001



Name	Value
aclk	1
aresetn	1
ain[31:0]	3f800000
din[31:0]	41700000
addr[5:0]	01
we	0
valid	0
dvalid	1
dout[31:0]	1.0
memo[0:15][31:0]	00000000,3f800000
memo2[0:15][31:0]	00000000,3f800000
i[31:0]	00000010
k[31:0]	00000001