FLOATING POINT MULTIPLIER USING FPGA

Felipe Valencia

Mohamed Beltagy

USI

ALaRI

Introduction

Problem statement

Design a floating point multiplier based in the ieee standard 764 using vhdl and implement it in the FPGA Cyclone II

Extras

Management of special cases

- Zero
- Infinities
- Not a Number
- Overflow
- Underflow

Difficulties

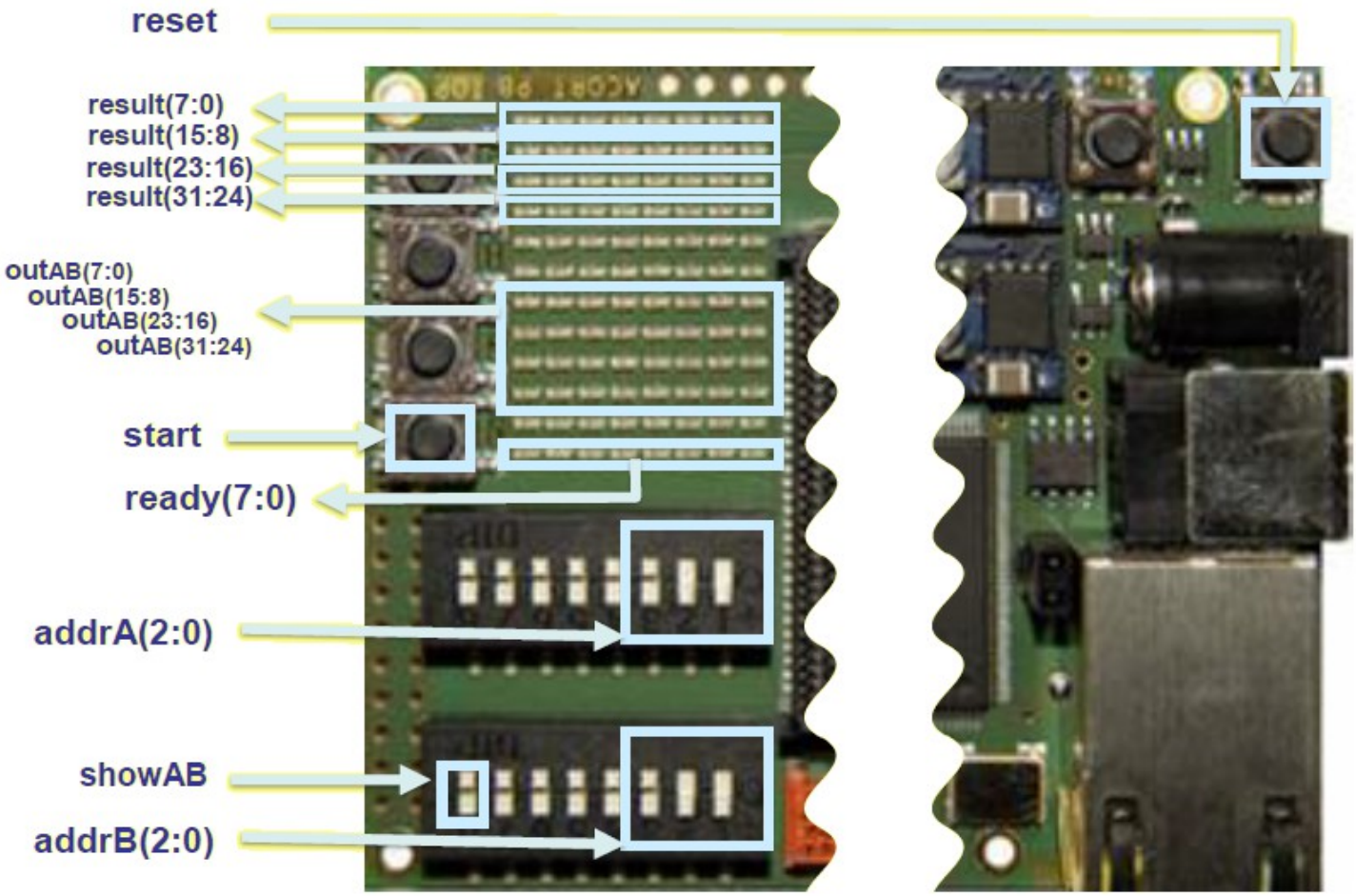
- Clock synchronization
- Conversion between different types of VHDL
- Management of files in different softwares

Manual: How to use the Multiplier Device

Steps

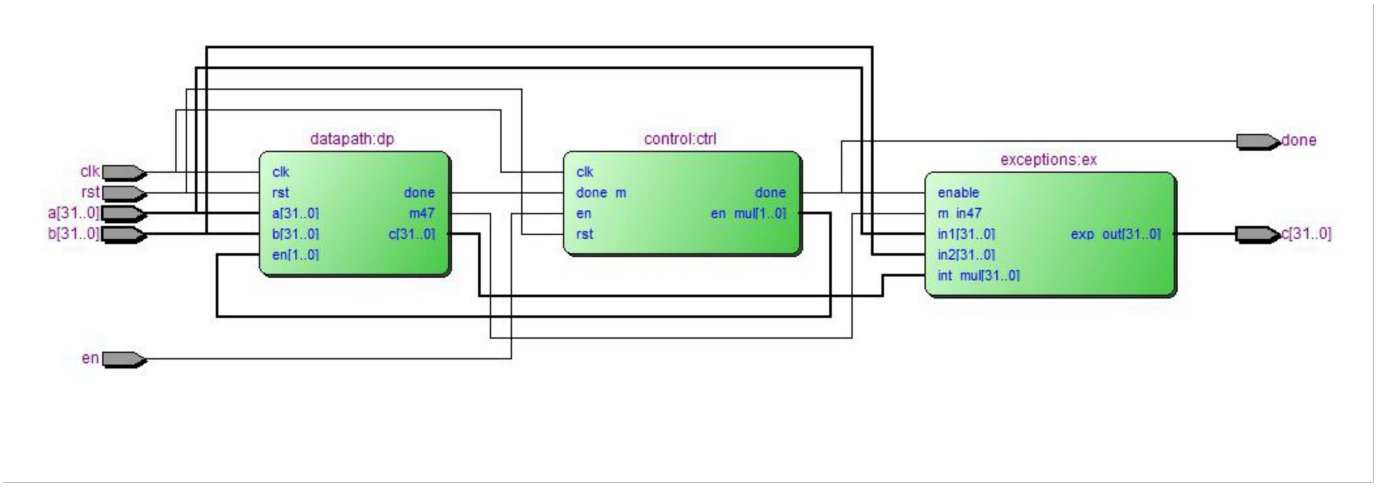
1- Put the numbers you want to multiply in the rom using the romMemOpA and the romMemOpB files, and know at which addresses you put the numbers.
2- Using the address change the value to desired multiplicand and multiplier and check those using the show AB button in the showAB LEDS.
3- Press the start and read the data from the output LEDS.
4- Change the addresses and read the output for new multiplication.

LEDs and Buttons of the Device

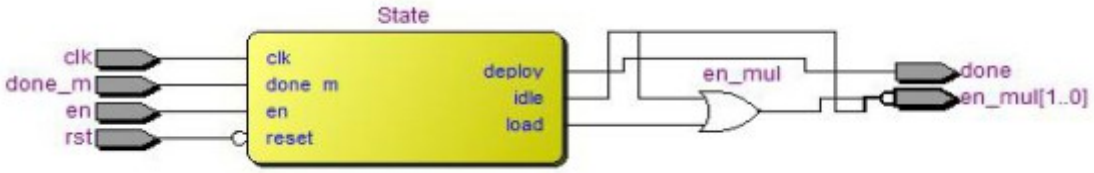


Multiplier

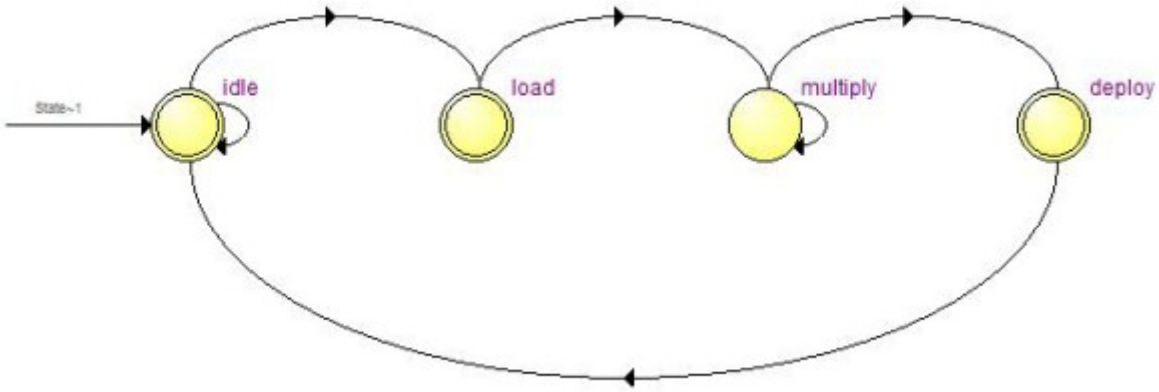
The multiplier is composed by three units: Data path, control and exception unit; the data path and the control unit work together for getting the result of a normal operation, and the exception unit evaluate if there are some special condition, in this case it will change the result by the special result.



Control unit:



This is the unit that manage the data path unit and it is composed of 4 state:



Idle: Meanwhile there are no multiplications the system will be idle and will be here, when the enable signal is set it will go to the next state

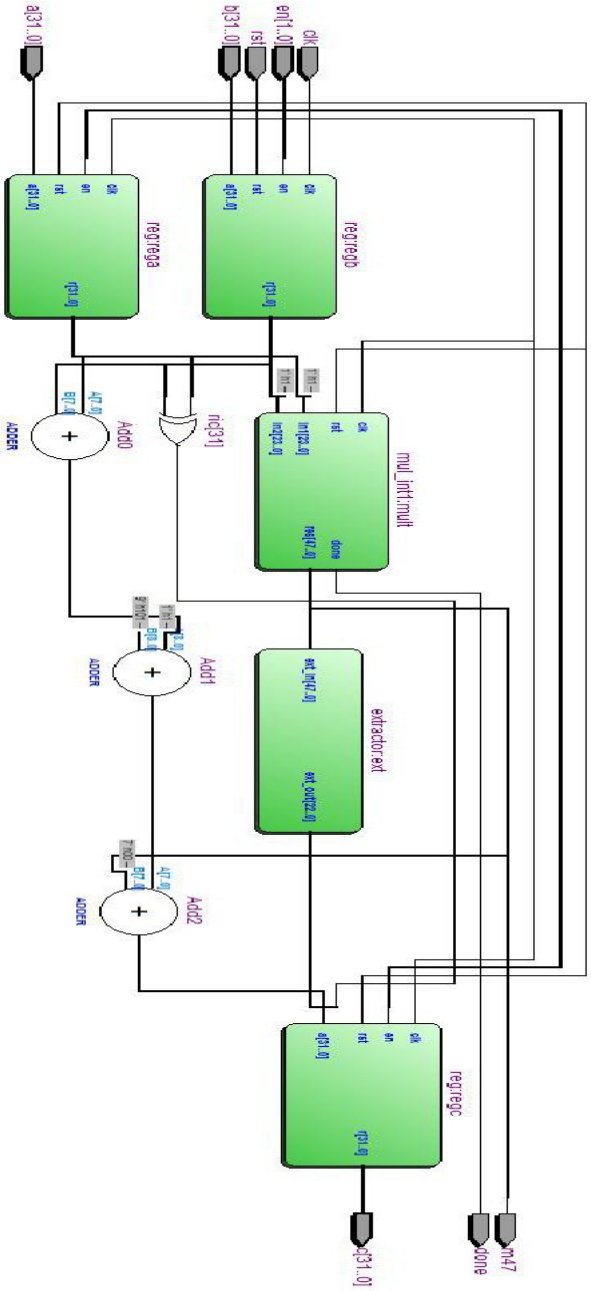
Load: In this state the multiplier loads the values of the multiplication, this task takes one cycle, then it will go to multiply state

Multiply: Here the control unit waits until the datapath finish to do the multiplication.

Deploy: Here the datapath loads the result and the signal are synchronized.

	Source State	Destination State	Condition
1	deploy	idle	
2	idle	idle	(!en)
3	idle	load	(en)
4	load	multiply	
5	multiply	multiply	(!done_m)
6	multiply	deploy	(done_m)

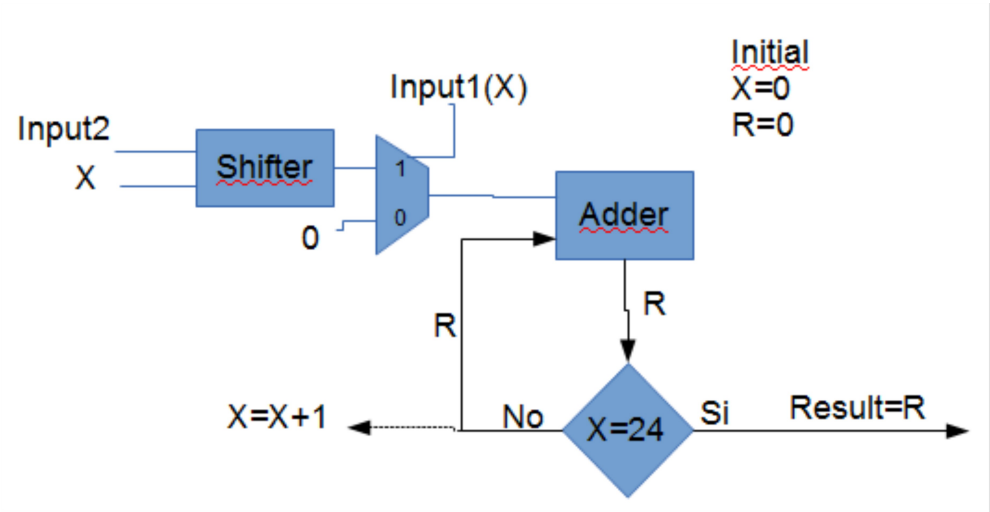
Data Path Unit:



The Multiplication Algorithm

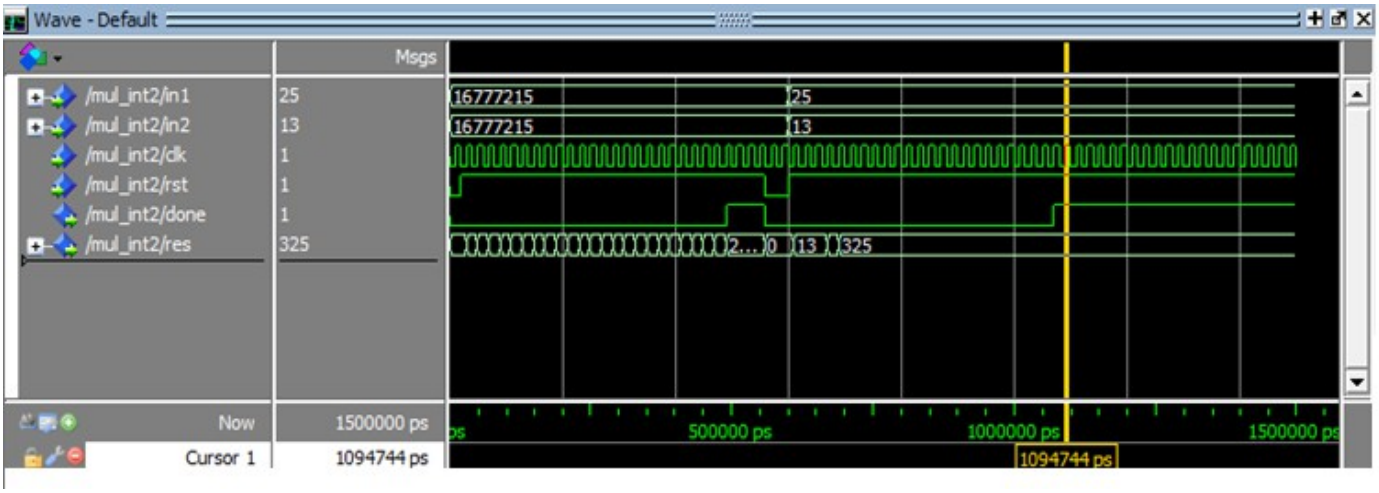
We created two algorithms for multiplication both are fast and correct but one takes 24 cycles and the other takes only 2 cycles.

The 24 cycle multiplication algorithm:

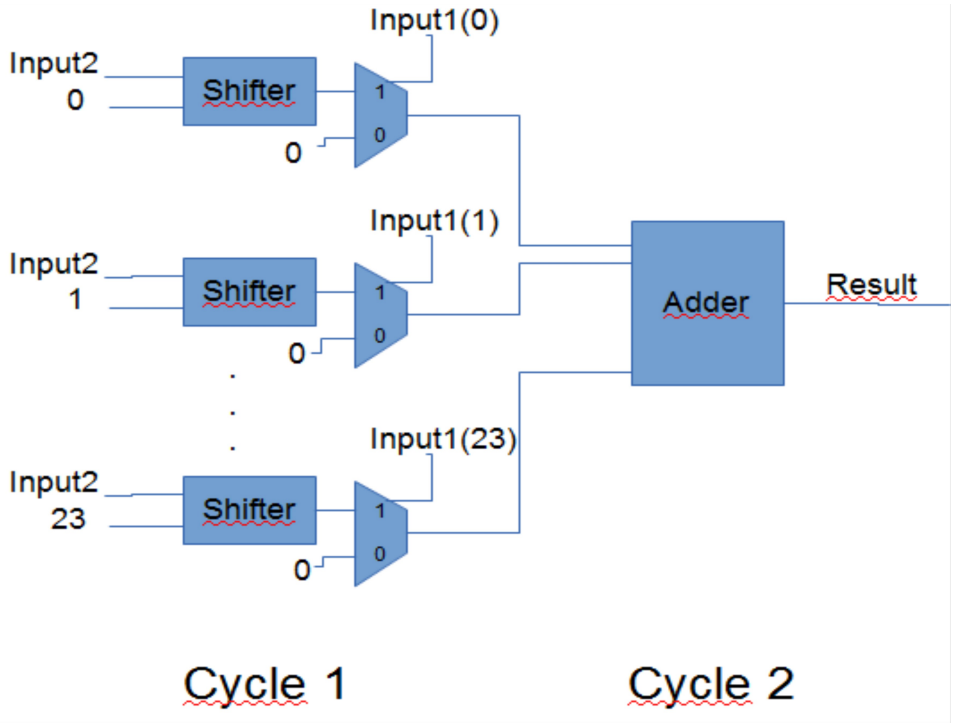


Here the shift left register act as a multiplier of power 2 and checks the bits of the multiplier bit by bit *sequentially* when the multiplier bit is the multiplicand is shifted by the index of this bit and add the result to the previous result (accumulates the 24 results), here the multiplier is the input1 .

For example if the multiplier bit number 5 is one the multiplicand will be shifted to the left by 5 bits.

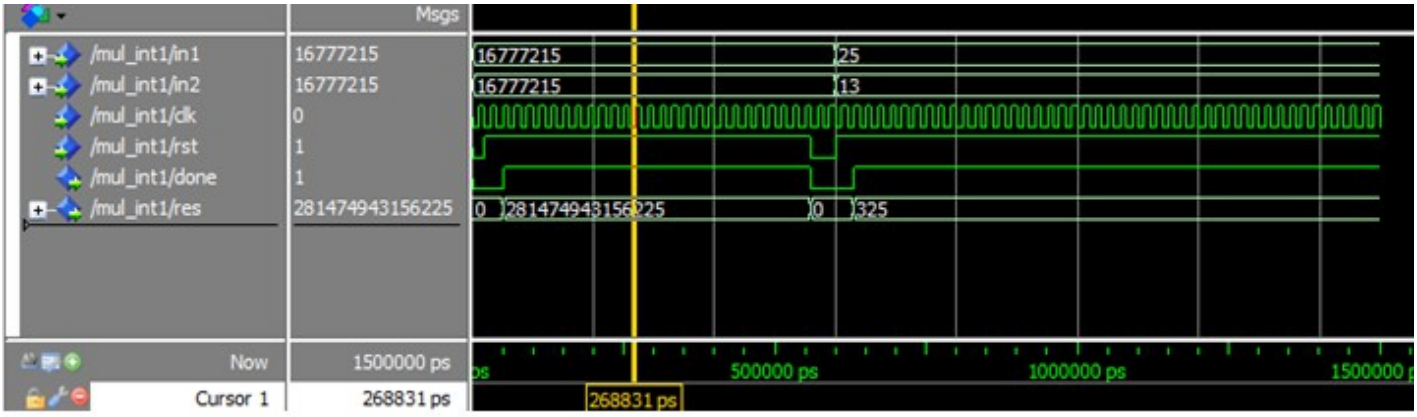


The 2 cycle multiplication algorithm:



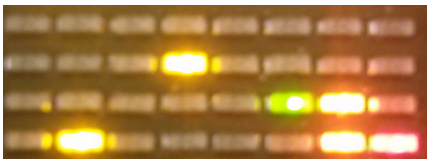
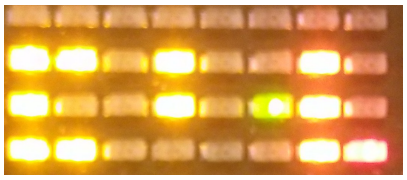
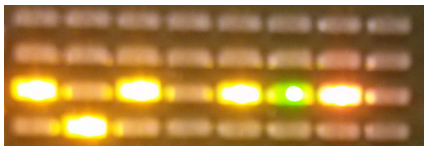
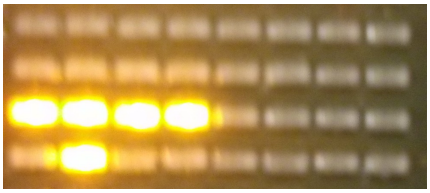
Here the shift left register act as a multiplier of power 2 and checks the bits of the multiplier bit by bit *In parallel* in one clock cycle, when the multiplier bit is 1 the multiplicand is shifted by the index of this bit and add the all the result at the same time in one clock cycle , here the multiplier is the input1 .

For example if the multiplier bit number 5 is one the multiplicand will be shifted to the left by 5 bits

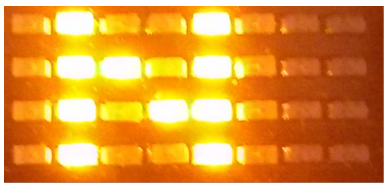
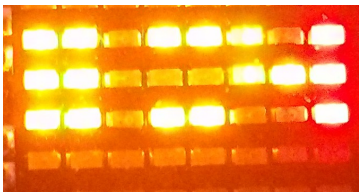
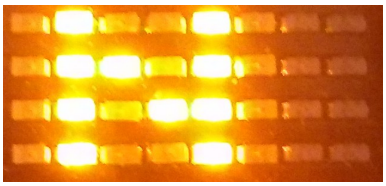
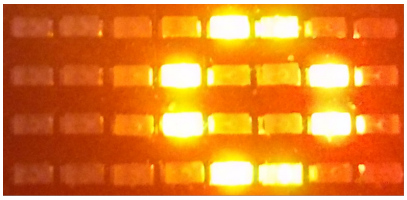
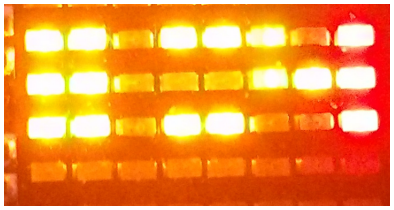
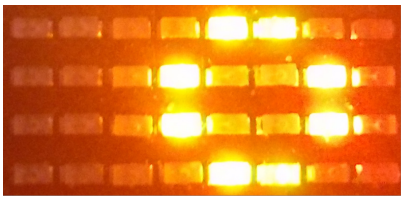
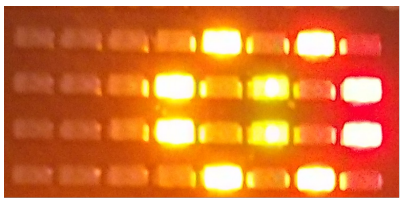
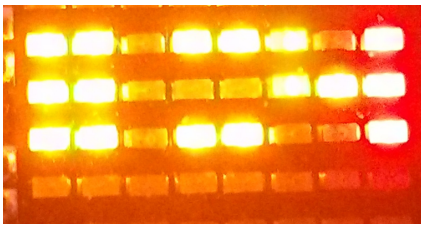
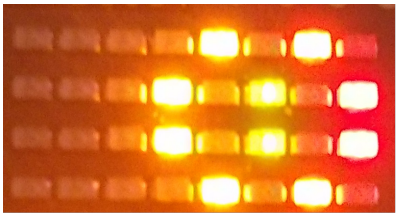


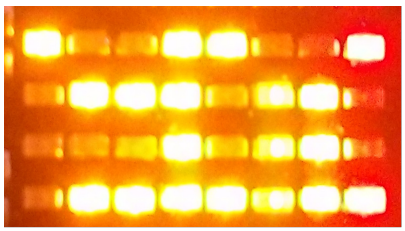
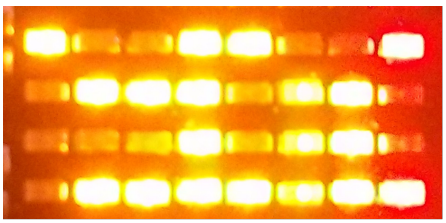
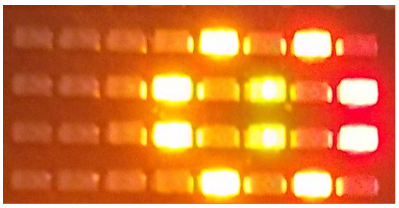
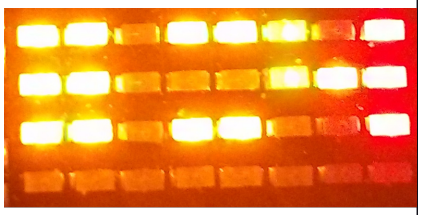
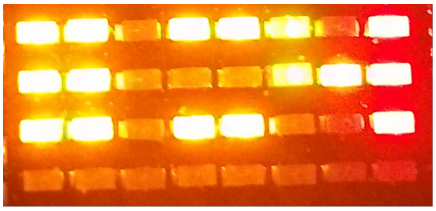
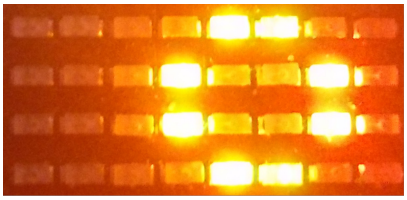
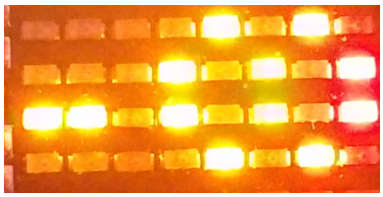
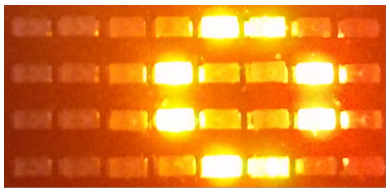
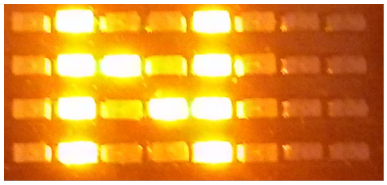
Results

Test Cases:

A	B	Result
134.0625 address 000 "01000011000001100001000000000000" 	-2.25 address 000 "11000000000100000000000000000000" 	-301.640625 “11000011100101101010010 00000000” 
-14.5 address 001 "11000001011010000000000000000000" 	-0.375 address 001 "10111110110000000000000000000000" 	5.4375 “01000000101011100000000000000000” 
7.5 address 010 "01000000111100000000000000000000" 	15.5 address 010 "01000001011110000000000000000000" 	116.25 “01000010111010001000000000000000” 

The exceptions:

A	B	Result
NaN If one operand is NaN the result will be NaN 	Any number b 	NaN 
Zero If one operand is 0 and the other one is not infinity the result will be zero 	B != infinity 	Zero 
Infinity ± ∞ If one operand is infinity and the other one is not 0 the result will be infinity 	B != 0 	Infinity ± ∞ 
Big Number (2e38)	Big Number (2e38)	Infinity ± ∞ If the result of the multiplication is over the range of floating point single precision the result will be

		infinity 
Small number (2e-38) 	Small number(2e-38) 	Zero If the result of the multiplication is over the range of floating point single precision the result will be infinity 
Infinity ± ∞ 	Zero 	NaN 

Appendix A : The Rom Data and addresses

```
-- Memory initialization Memory A
constant romData : romTable := (
  ---------|-------|-------|-------   -- Value     -- Address
  "01000011000001100001000000000000", -- 134.0625 -- 000
  "11000001011010000000000000000000", -- -14.5    -- 001
  "01000000111100000000000000000000", --   7.5    -- 010
  "11000001100100000000000000000000", -- -18.0    -- 011
  "00000000000000000000000000000000", --  0       -- 100
  "01111111100000000000000000000000", -- +inf     -- 101
  "00000000110110011100011111011101", -- 2e-38    -- 110
  "01111111000101100111011010011001"  -- 2e38     -- 111
);


-- Memory initialization Memory B
constant romData : romTable := (
  ---------|-------|-------|-------   -- Value     -- Address
  "11000000000100000000000000000000", -- -2.25    -- 000
  "10111110110000000000000000000000", -- -0.375   -- 001
  "01000001011110000000000000000000", -- 15.5     -- 010
  "01000001000110000000000000000000", --  9.5     -- 011
  "11111111111111111111111111111111", --  NaN     -- 100
  "11111111100000000000000000000000", -- -inf     -- 101
  "00000000110110011100011111011101", -- 2e-38    -- 110
  "01111111000101100111011010011001"  -- 2e38     -- 111
);
```