Marwan Jalaleddine (Student)

420 Project Report

# Algorithm:

The following algorithm was followed for the Vector Norm-2 List Processor:

-We go from each node of memory using only the Next Pointer

The cycles as seen by the memory is as follows:

Cycle1 -> Get the Addresses and save them in a register file

Cycle2 -> Get the Data and pass them to processing unit for calculation.

The benefits of this algorithmic design is that we have more time to do the calculations. We can keep the data for two cycles in the multiplier stage for computations to happen and later pass them to the addition stage for them to be processed.

One disadvantage of this approach is the unequal clock time for each stage of calculation.

The data then passes through the Datapath where it is initially squared and then added and accumulated . Due to the pipelining procedure we can reduce the critical path of our circuit to the maximum delay of our pipelined stages.

The pseudocode is as follows:

Adress<=0;

Sum<=0;

while (adressnext !=1)

adressnext<=data[i] %Cycle 1 %Cycle 3

x1<= data1; %Cycle 2&3 %Cycle 4&5

x2<= data2; %Cycle 2&3 %Cycle 4&5

tempsquare1 <= x1^2; %Cycle 2&3 %Cycle 4&5

tempsquare2 <= x2^2; %Cycle 2&3 %Cycle 4&5

tempadd<= tempsquare1+tempsquare2; %Cycle 4 %Cycle 6

Sum<= tempadd+Sum %Cycle 4 %Cycle 6

Adress<= Adressnext %Cycle 4 %Cycle 6

end

## Data Path:

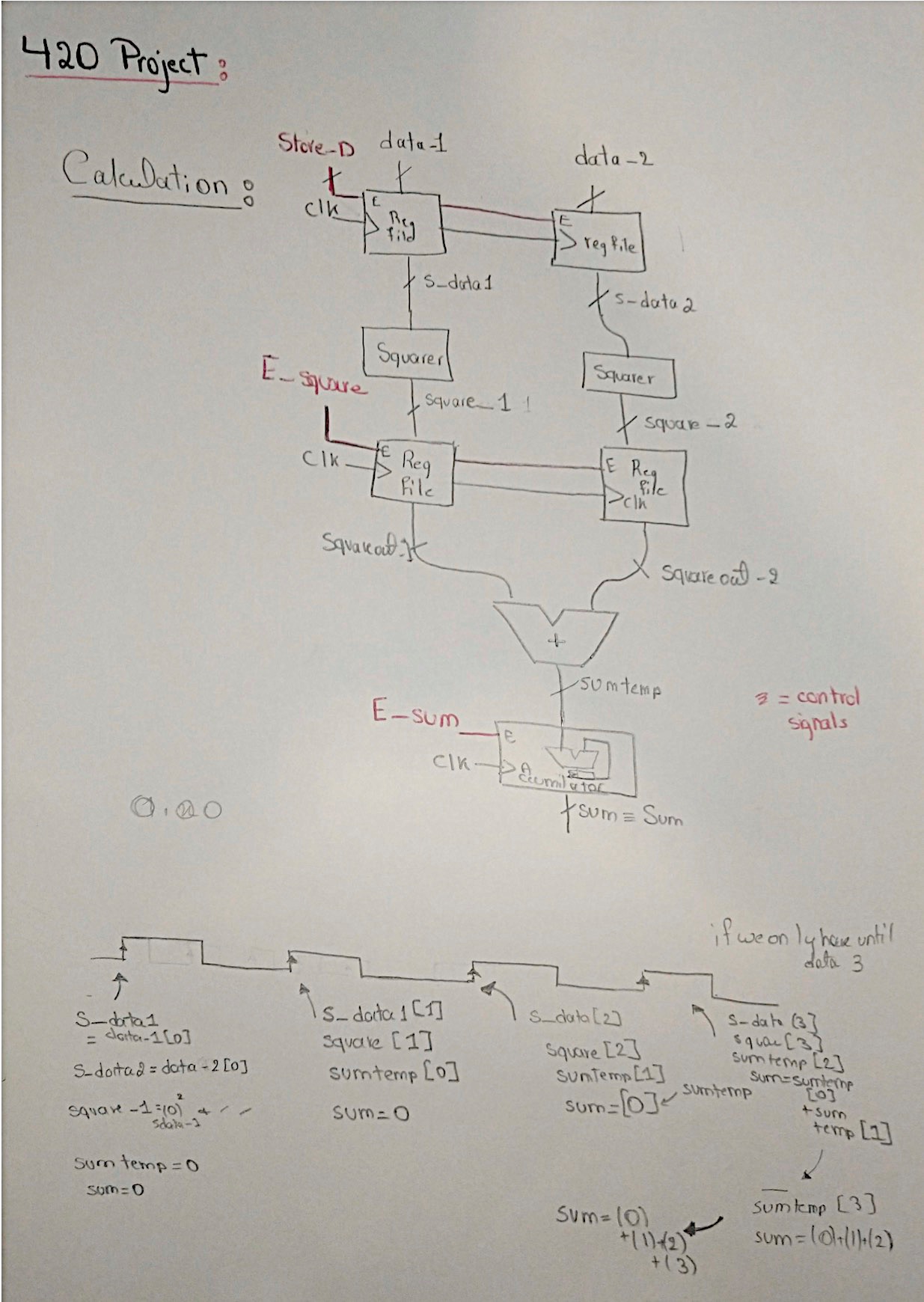
Our Datapath uses 2 Squarers (Multipliers) and two adders . There are three control signals used to control the flow of data in the Datapath.

Store\_D : stores the data coming from memory in the register file and passes them to the Squarers for squaring.

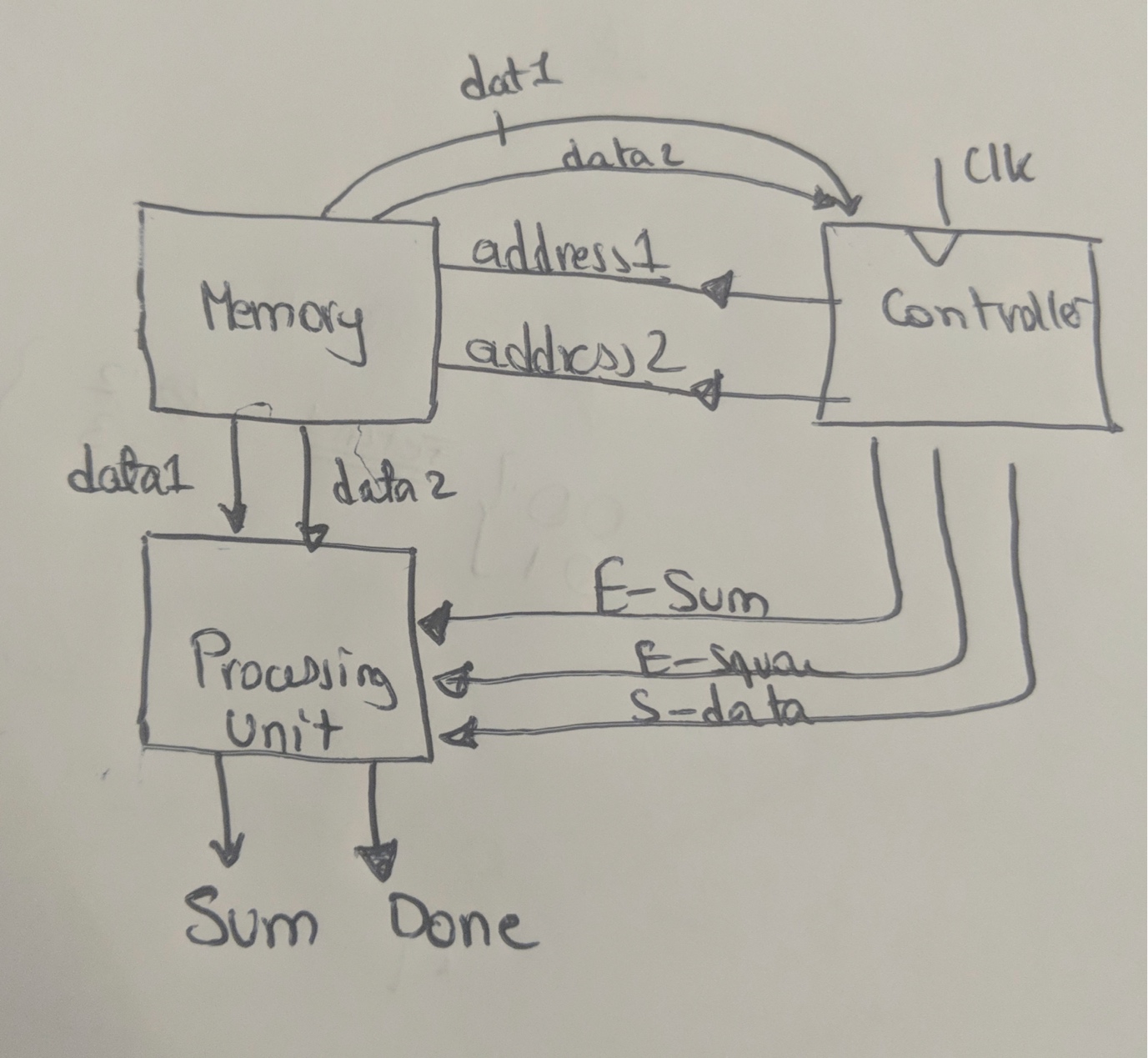
Enable \_Square: Passes the data to the adder/accumulator stage where it will be added together and to the initial sum.

Enable \_Sum: Allows us to add the new sum to the previous one.

The Calculation Unit and an example of its operation is as follows:



The Top Level design is as follows:



# Finite State Machine:

The control Unit’s FSM is as follows:



# Sizes of Data:

We took a fixed size of 24 bits for the entire busses to keep the same data size throughout our Datapath. We need to know the data profile in advance to take into consideration the size of data in and out. The needed information is as follows:

If the data has a similar distribution to the other data following it , we can accept the design choice mentioned here. However, if the data is too large (super passing the 2^7 mark or if it is mixed (large numbers and small numbers) then our method would give us an inaccurate result.

# Addition Example :

always @ (data\_1 or data\_2)

begin

if (exponent\_1<exponent\_2)

begin

shiftedmantissa=mantissa\_1>>exponent\_2-exponent\_1;

mantissa\_final=shiftedmantissa+mantissa\_2;

if (mantissa\_final [15]==1)

begin

mantissa\_final=mantissa\_final>>1;

exponent\_final=exponent\_2+1;

end

else

exponent\_final=exponent\_2;

end

else if (exponent\_2<=exponent\_1)

begin

shiftedmantissa=mantissa\_2>>exponent\_1-exponent\_2;

mantissa\_final=shiftedmantissa+mantissa\_1;

if (mantissa\_final [15]==1)

begin

mantissa\_final=mantissa\_final>>1;

exponent\_final=exponent\_1+1;

end

else

exponent\_final=exponent\_1;

end

sum={1'b0,mantissa\_final[14:0],exponent\_final};

end

We dealt with addition as mentioned above. We shifted the smaller number to the right and we added it to the other mantissa while taking account the size of the exponents.

Multiplication was easier as it only needed unsigned multiplication and addition of exponents.

# Timing Analysis:

The clock cycle needed for our Datapath is the maximum stage delay in our system which is the multiplier time delay.

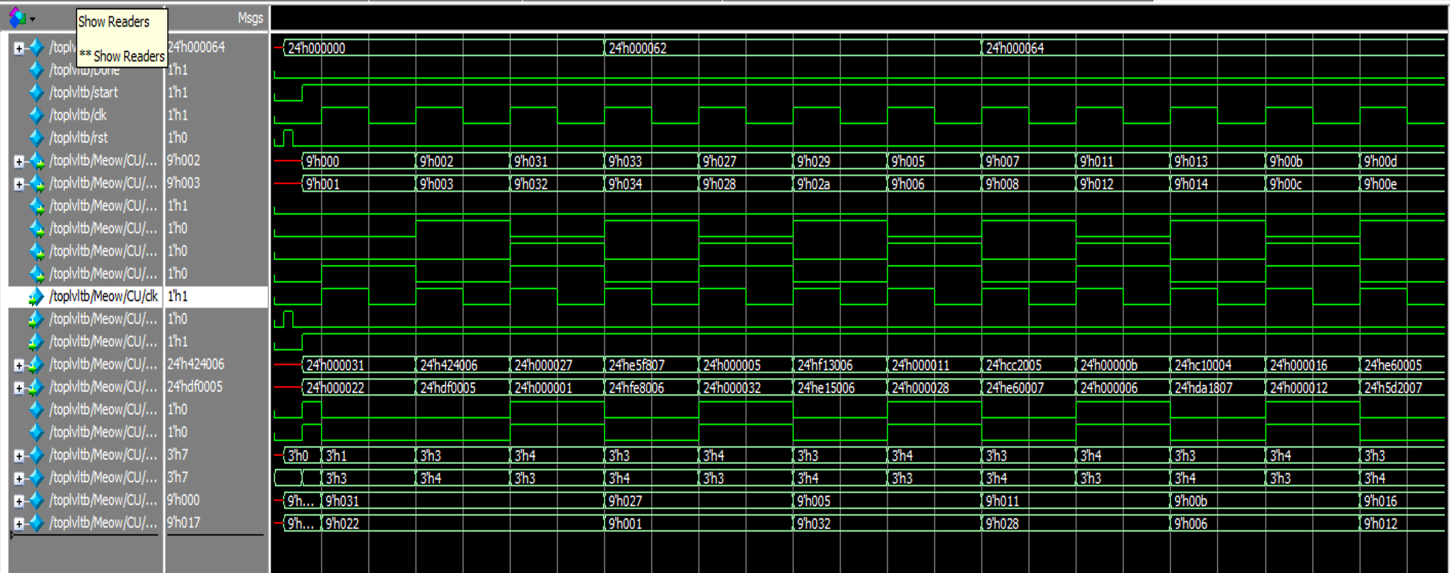
(0.9 n2 + 1.5) ns

Given that we are multiplying 14 bit numbers, the time needed is 177.9 nanoseconds.

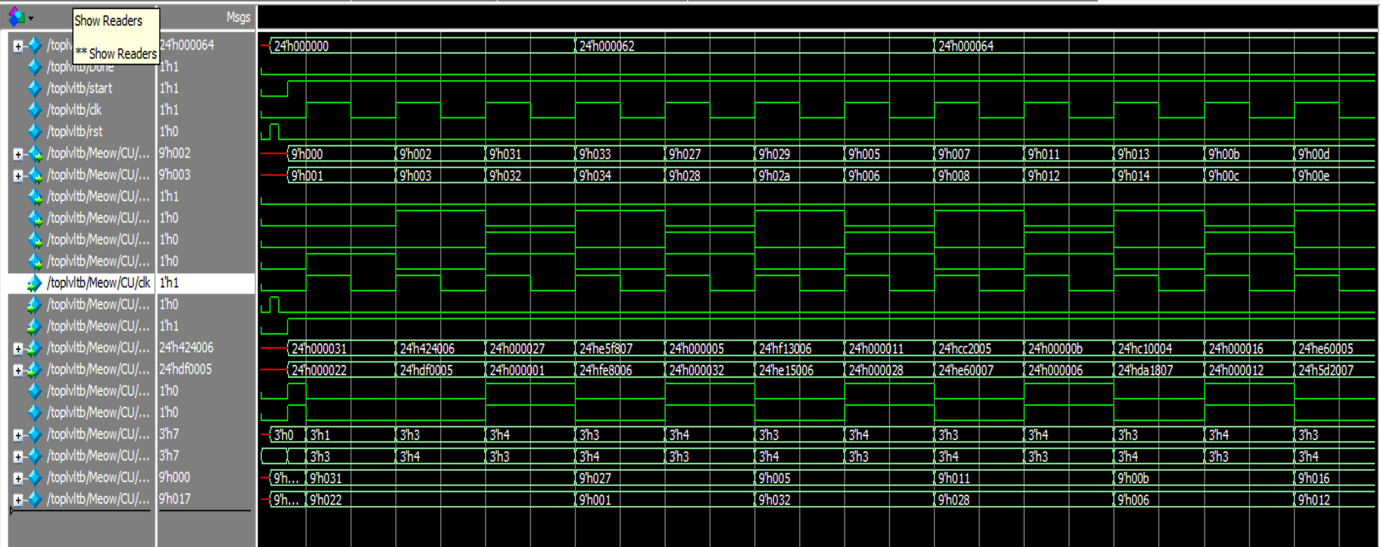
We add to this delay the setup time of the Register files and the Clock to Q delay which gives us a delay of 178.7ns.

We can clock the our Datapath at half the multiplier length , so the maximum clock speed we can have is 89.75 ns. One disadvantage is that the throughput of the design is reduced by half as every two cycles a new sum appears at the extremity of the design. The advantage of this design is that we can fetch 2 data from memory if we use complicated control signals.

# Testing the Design:



The above figure shows the top level data obtained from simulating the device.



In the figure above it shows that we are traversing the entire nodes of the list and fetching data. This verifies our complicated control strategy which requires storing the address for later use.

Initially the design was implemented for Integers in normal notation then transformed to support our mantissa - exponent expression.

The following is our timing diagrams for the individual blocks:



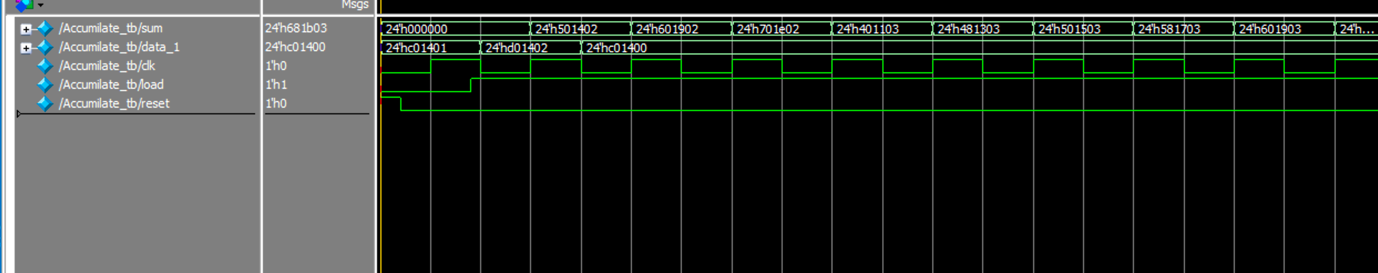
The following is the Control Unit accepting integer data. This was implemented before the mantissa exponent expression.

# Timing Diagrams of the Subcomponents:

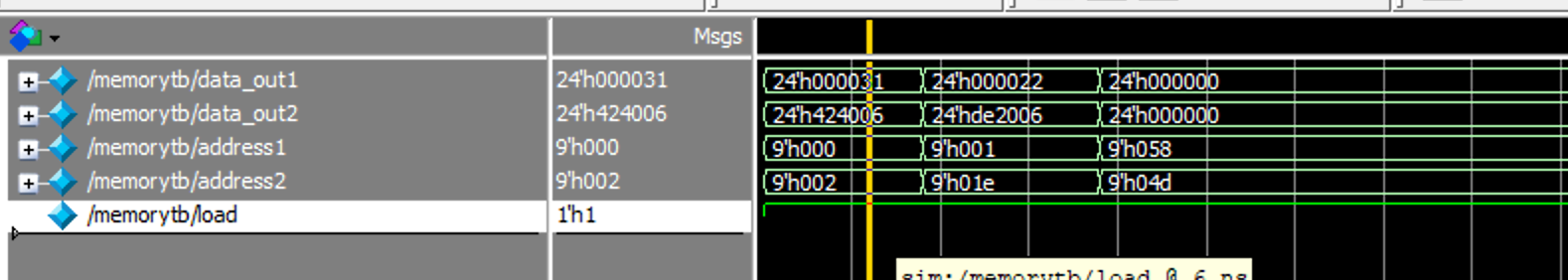
Adder:



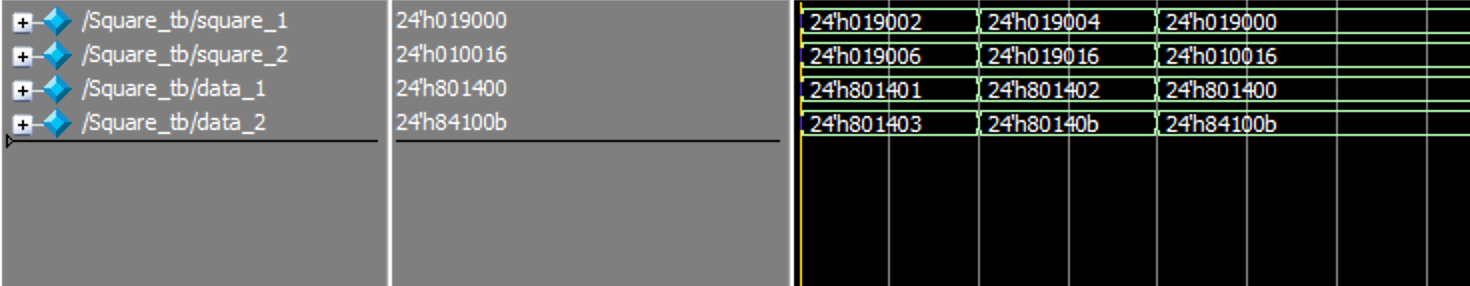
Accumilator:



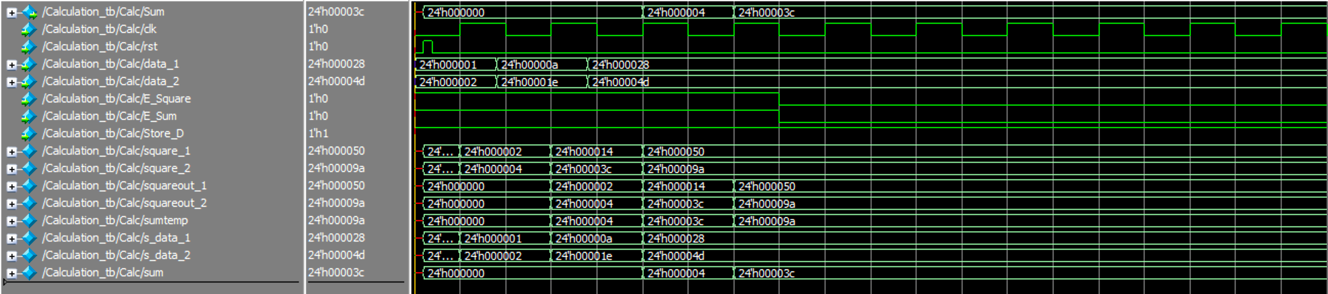
Memory:



Squarer:



Datapath:



# Appendix1:

Matlab Script to transform to our notation:

function [out] = dtobin(number)

%Number to binary representation

if (number>0)

sign=1;

else

sign=0;

end

number=abs(number);

n = 14; % number bits for integer part of your number

m = 14; % number bits for fraction part of your number

exponent=0;

d2b = fix(rem(number\*pow2(-(n-1):m),2));

if (any(d2b(1,1:13)))

while (any(d2b(1,1:13)))

d2b=[0 d2b(1,1:27)];

exponent=exponent+1;

end

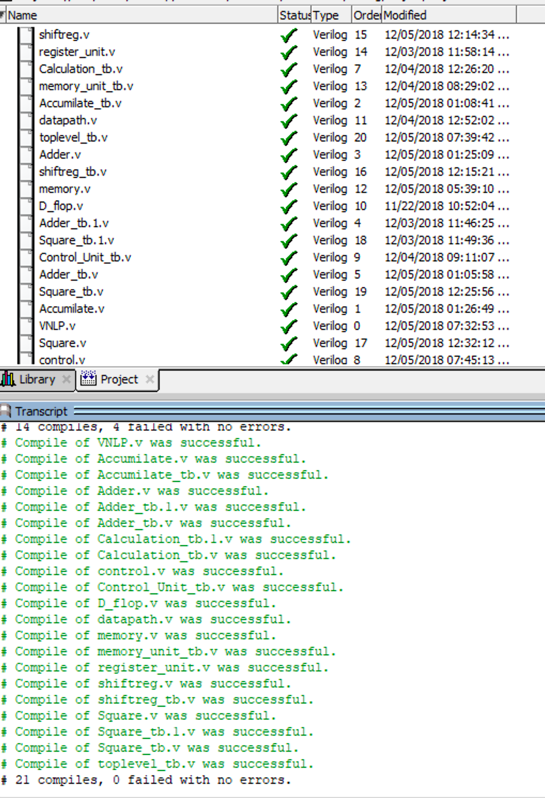
end

expo = fix(rem((exponent+1)\*pow2(-(7):0),2));

out=[sign d2b(1,14:28) expo];

end

# Appendix2:

Compilation proof: