Designing and verification of Multiply-accumulate (MAC) Unit

A project report submitted by

Avinash Choudhary [EE24E017]
Dadichiluka Shiva Lalith [EE24M089]

Under the guidance of
PURUSHOTHAMAN RAMAKRISHNAN
And
GOPALAKRISHNAN SRINIVASAN

In the partial fulfillment of
CAD for VLSI
subject

ACKNOWLEDGEMENTS

We want to sincerely thank our colleagues from Indian Institute of technology,Madras for their helpful criticism and cooperative attitude during our conversations. Their advice and assistance greatly influenced how we understood and implemented the MAC unit design.
We also like to thank the teaching assistants for helping us with the cocotb framework and giving resources, which made our verification process much easier.
Finally, we recognize that this project requires teamwork. By cooperating, we were able to leverage our own talents and successfully execute the MAC unit.

Avinash Choudhary [EE24E017]
Dadichiluka Shiva Lalith [EE24M089]

Table of contents:

Objective:

A Multiply and Accumulate (MAC) block is a basic hardware unit which is used to perform two fundamental arithmetic operations i.e.multiplication and accumulation.

MAC: Result=Result + (Input_1 * Input_2)

A MAC block plays a very important role when it is about faster computation of results in the application of Deep Learning, Signal Processing, Control systems. We have implemented a MAC block which can handle both Integer and floating point representation type inputs. Inputs and outputs are both in binary format. Since it can handle different types of input formats, its usability range will be increased. The design of MAC block has been implemented in Bluespec Verilog. The design (DUT) is verified using the COroutine based COsimulation TestBench environment (COCOTB) and python. The design implementation contains both pipelined and unpipelined versions to carefully observe the differences in internal data flow and timing requirements.

Scope:

A hardware MAC unit specifically designed for deep learning applications is included in the Multiply-Accumulate (MAC) Unit Design project's scope, which also includes its implementation and verification.

The following essential elements will be upheld by the project:

1. Goals of the design:

Categories of Data In support:

Two particular data kinds will be supported by the MAC unit:

S1: Operations using 32-bit signed integers (int32) and 8-bit signed integers (int8).

S2: Activities with IEEE 754 32-bit floating point (fp32) and bfloat16 (bf16).

Implementation of MAC Operation: The MAC operation, which is defined as MAC=AxB+C, will serve as the foundation for the essential functionality.

2. Details of the Architecture:

Five predefined methods—A, B, C, S1_or_S2, and MAC—will be included in the top-level module of the design. Depending on the operating mode denoted by S1_or_S2, the architecture will use a multiplexer (MUX) for output selection and demultiplexers (DEMUX) for input handling.

3. Design Modifications:

An initial deployment of the MAC unit without pipelining to create baseline capability is known as an unpipelined design.

Pipelined Design: A version that has been altered to improve performance and throughput by implementing pipelining.

4. Requirements for implementation:

In order to rigorously follow project criteria that forbid the usage of ordinary arithmetic operators (+, *) in the implementation, the MAC unit will be developed using Bluespec System Verilog (BSV).

5. Checking:

Verification is done using COCOTB and python. The given test cases contains list of values for a, b, c and mac_output in both integer and floating point binary values. Given the model with input values, the output produced by model and given file both should match.

Introduction:

MAC for Integer type inputs:

- The signed integer inputs given to the MAC block which has been implemented is represented in binary format i.e. the msb representing the sign bit and remaining bits representing the 2’s complement of positive value of given integer.
- First step is to multiply the inputs, say a,b.
- Before directly performing the binary multiplication,store the sign bit of final product by performing bitwise xor on sign bits of both inputs. Then convert both input into positive numbers by 2’s complement method.
- Now zero extend the positive numbers in binary format to 16 bits and find the 32 bit product by performing multiplication with shift and add method.
- Finally if the sign bit of result is to be 1, then return the 2’s complement else return the product as result of multiplication part.
- Next we need to accumulate this with the 3rd input given, say c. Perform bitwise full adder representation on each bit from lsb to msb letting the carry flow in same direction.
- This gives the final result which is 32-bit binary representation.

MAC for Floating point type numbers:

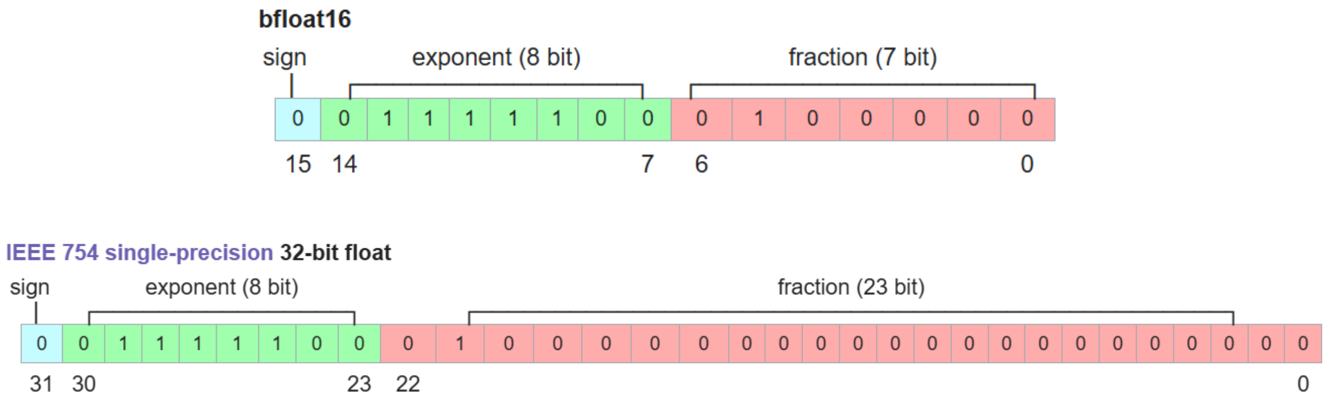


Fig.Binary representation of bf16,fp32 from wikipedia

- The two inputs a,b here are bf16 type and the third input c is for fp32 type. The format for representing both types of representations are shown in the above figures.
- In the first step, we need to multiply the inputs a and b. Considering here only positive values, to multiply a,b their exponents should be added and muntissas with the implicit ‘1’ added at msb should be multiplied in the same way as done for integer representation(shift and add).
- After adding exponents subtract 127 bias from it.
- The product after multiplying the mantissas, say result_mant is a maximum of 16 bit binary value (mantissa with implicit ’1’ will be 8 bits). In this binary value, if the msb is ‘1’, then store the next 15 bits as result_mant and increase the exponent by 1, if not then store the 15 bits except the msb as result_mant.
- Now we have to round the result_mant back to get bf16 representation, containing only 7 mantissa bits. For this we will use

round to nearest rounding scheme. In this rounding scheme, if the bit just after the maximum limit is 1, then increment the mantissa by 1. In this process if the mantissa exceeds the assigned bits, increase the exponent by 1.

- This will give the final rounded result of multiplication block.
- This product is bf16 type.we will convert this into fp32 by simply extendingthe mantissa of the product to 23 bits with ‘0’.
- At this point we have two fp32 type values, and we need to add them.
- To add two fp32 representation numbers,we need to match the exponents of both values and shift the mantissa of lower exponent by delta-expression(initial difference between the exponent of both).
- While right shifting any of mantissa, store the values just atter LSB of 23 bits shifted mantissa, say round_flag.
- Now since exponents are matching, we can simply add the mantissas of both values including implicit ‘1’ at MSB(Total 24 bits).
- If performing addition of mantissas (24 bit) gives over flow, increase the exponent by 1 and store 23 bits without lsb as result-mant. Else simply store as 23 bits without msb as result-mant.
- Now we need to round off the resulting mantissa based on round_flag.If round_flag stores 1,increase mantissa by 1.If the increment of 1 causes overflow,increase the exponent by 1.if round_flag stores 0,let everything remains same.
- Now combining all gives final result.

Design Architecture:

Unpipelined:

- The implemented design contains an interface with five methods named read-A, read-B, read_C, read_S,mac-calc.
- Each of the 4 methods read_A,read_B, read_C, read_S takes input of 16, 16, 32,1 bit binary values.
- These binary values can store both int’s and bf’s representation. We must operate on these values based on S(Select line).
- The method mac_calc contains two functions, mac_flt and mac_int, which calculates and returns the output based on select value.
- After the interface, there are a few functions, which are explained later in this report.
- The function mac_int has 3 inputs Bit #(16)a, Bit #(16)b, Bit #(16)c. Where a, and b are signed extended versions of actual int #(8) inputs.
- Firstly the inputs a, and b are sent into the 16-bit multiplier(bit_16 Multiplier), which returns the product following the procedure explained in theory.
- Then the 32-bit product is sent as input to a bitwise adder function(bit adder).
- Now it generates the final output of the mac_int function.
- The function mac_flt again has 3 inputs Bit #(16)a, Bit #(16)b, Bit #(16)c. Where a, and b are bf16 type and c are fp32 type values.

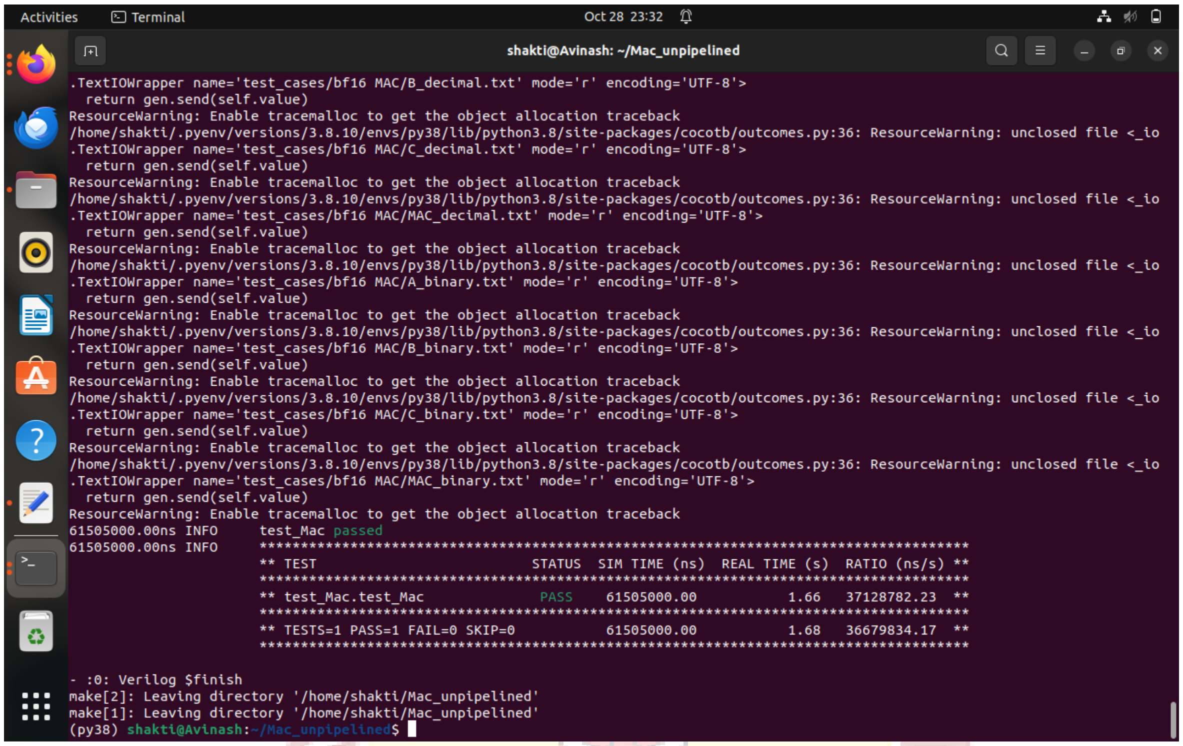
- Perform the multiplication of a, and b as given in theory. Send the mantissa (with implicit '1') into an 8-bit multiplier (bit 8 multiplier).
- We need to perform zero-extended or pad with extra zeros at the end according to size requirement.
- Round this product to the bf16 type using the round to nearest method.
- Then convert the product to fp32 type and add with c(as in theory).
- Carefully following the bit overflow, we get the final result of mac_flt.

Pipelined:

- The pipelined version contains an interface with two methods read_in and mac_calc. Each of these methods is to store input in input FIFO and get output from output FIFO and dequeue it.
- Initially we need to declare structures for taking initial input, inputs for stage 2 of pipelining, and final output.
- Struct multiplier input containing 16-bit val1, which is c, 1 bit s as a select line. All these representations are the same as described in the unpipelined version.
- Struct AdderInput containing 32-bit val1, which is the product of a,b. 32-bit val2, which is c, 1-bit s as select line.
- Struct AdderOutput contains 32-bit val1 as output.
- Instantiate FIFO such structure type using mkPipeline FIFO ( );

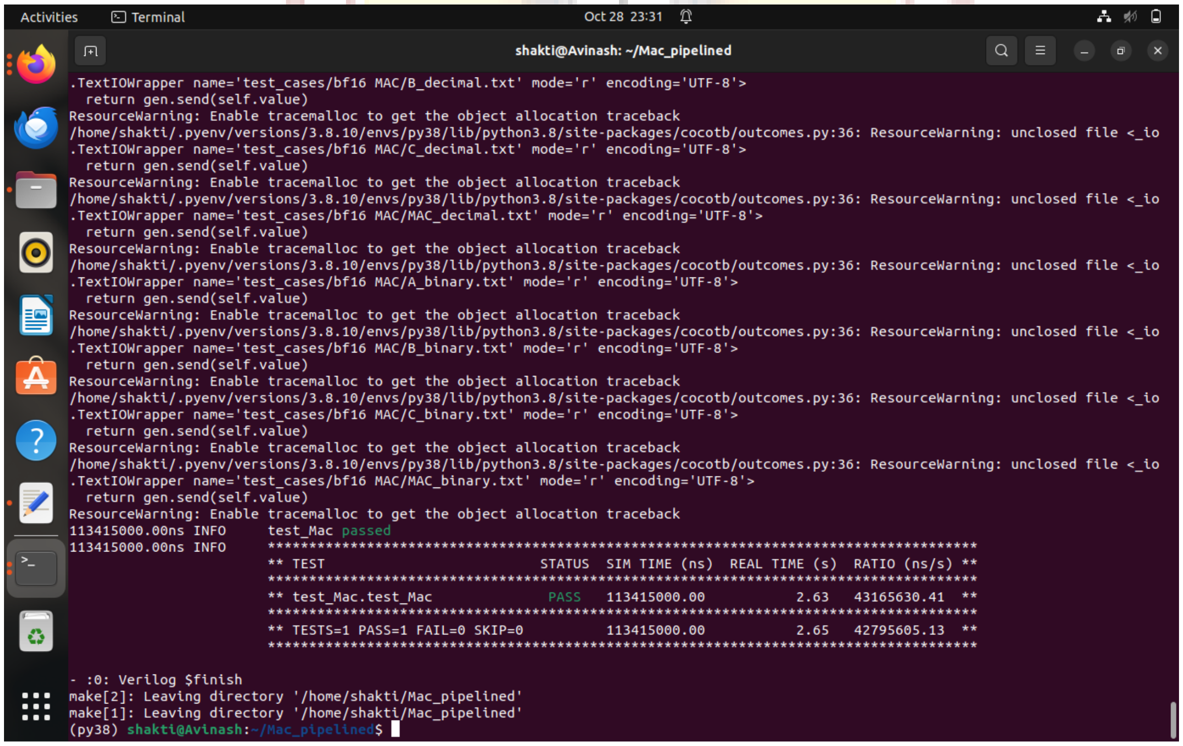
- Then few functions are similar to unpipelined design implementing similar functions. But since we are making two pipeline stages for multiply and add each, there are mac_flt_mul, and mac_flt_add for multiplications and addition each at floating point values.
- After the required functions,we define two rules as rd_pipeline_mulstage, and rd_pipeline_addstage for multiplications and addition stages.
- In rd_pipeline_mulstage, we get the adder inputs from adder_ififo, perform addition as flt or int based on select line. Then enqueue sum into the adder_ofifo, and dequeue the adder_ififo.
- The add, mul operations used above are the same as unpipelined ones.

Outputs:



```
.TextIOWrapper name='test_cases/bf16 MAC/B_decimal.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/C_decimal.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/MAC_decimal.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/A_binary.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/B_binary.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/C_binary.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/MAC_binary.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
61505000.00ns INFO     test_Mac passed
61505000.00ns INFO     ******************************************************************************************
                       ** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                       ******************************************************************************************
                       ** test_Mac.test_Mac              PASS    61505000.00           1.66   37128782.23  **
                       ******************************************************************************************
                       ** TESTS=1 PASS=1 FAIL=0 SKIP=0           61505000.00           1.68   36679834.17  **
                       ******************************************************************************************

- :0: Verilog $finish
make[2]: Leaving directory '/home/shakti/Mac_unpipelined'
make[1]: Leaving directory '/home/shakti/Mac_unpipelined'
(py38) shakti@Avinash:~/Mac_unpipelined$ 
```



```
.TextIOWrapper name='test_cases/bf16 MAC/B_decimal.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/C_decimal.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/MAC_decimal.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/A_binary.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/B_binary.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/C_binary.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/shakti/.pyenv/versions/3.8.10/envs/py38/lib/python3.8/site-packages/cocotb/outcomes.py:36: ResourceWarning: unclosed file <_io
.TextIOWrapper name='test_cases/bf16 MAC/MAC_binary.txt' mode='r' encoding='UTF-8'>
  return gen.send(self.value)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
113415000.00ns INFO     test_Mac passed
113415000.00ns INFO     ******************************************************************************************
                        ** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                        ******************************************************************************************
                        ** test_Mac.test_Mac              PASS   113415000.00           2.63   43165630.41  **
                        ******************************************************************************************
                        ** TESTS=1 PASS=1 FAIL=0 SKIP=0          113415000.00           2.65   42795605.13  **
                        ******************************************************************************************

- :0: Verilog $finish
make[2]: Leaving directory '/home/shakti/Mac_pipelined'
make[1]: Leaving directory '/home/shakti/Mac_pipelined'
(py38) shakti@Avinash:~/Mac_pipelined$ 
```

Fig. Unpiplined and pipelined design outputs(All test cases are passed).

Verification:

Verification is done using COCOTB and python. The given test cases contains list of values for a, b, c and mac_output in both integer and floating point binary values. The results are verified using the output values provided. Also the included model file in python gives output by taking in decimal values of integer and floating point. Since the model file is not using binary values, there is almost no chance of getting an error. Since the floating point binary values are rounded off and this rounding effect on decimal could not be estimated, the verification of floating point values could not be performed using cocotb but is verified using the output file given.