IIT Madras
Department of Computer Science and Engineering

CS6230: July-Nov '24
Project 1: Multiply-Accumulate (MAC) Unit Design
Due: Friday, October 18 at 11:59 PM

1. Ground rules

1. Students must work in groups of 2 for this project.
2. Sharing of code between student groups and/or copying from public git repositories are considered cheating. The TAs will scan source code through various tools available to us for detecting cheating. Source code that is flagged by these tools will receive the following actions:
   - Zero credits for the project.
   - Final grade will be two grades lower than the actual grade obtained by the student.
3. Students are encouraged to engage in white board discussions, which is an essential aspect of the project.
4. Students must design the module using Bluespec System Verilog (BSV), and subsequently, compile and verify the design on the provided Shakti Distro.

2. Project Description

In deep learning computations, the MAC operation is the most important operation. It helps us realize a major part of the matrix related computations required in any Deep Neural Network (DNN). Building hardware support on the chip for doing these operations is a popular way of speeding up these computations.

A MAC operation, performed on 3 numbers A, B, and C, is formulated as shown below.

- `MAC = A*B + C`

In this project, you will need to design a Multiply-Accumulate (MAC) module that supports the MAC operations for the following data types.

- S1. (A: int8 , B: int8 , C: int32) -> (MAC: int32)
- S2. (A: bf16, B: bf16, C: fp32) -> (MAC: fp32)

The data types are described below.

- int8 : Signed 8-bit integer
- int32: Signed 32-bit integer
- bf16 : bfloat16 format (https://en.wikipedia.org/wiki/Bfloat16_floating-point_format)
- fp32 : IEEE 754 32-bit floating point

3. Specification of MAC Unit

The architecture of the MAC unit is illustrated in Fig. 1 and described below.

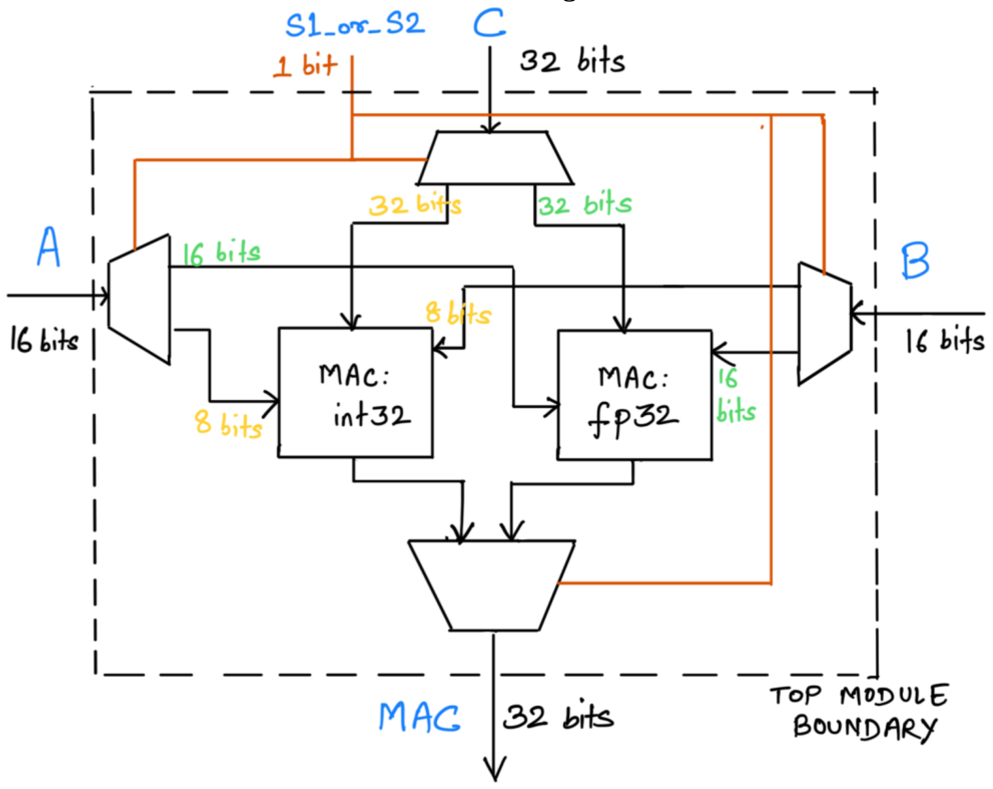

Figure 1. Architectural specification of the MAC unit

- The dashed line shows the top module boundary.
- There are 5 predefined methods in the top-level interface to the module, namely, `A`, `B`, `C`, `S1_or_S2`, and `MAC`. No additional methods are required. Please feel free to rename the methods to make your BSV code more readable. For instance, `method get_A(…)` can be used to load the value of 'A' into the design.
- Every wire shown in the Fig. 1 is, in general, a bus, with the number of bits mentioned closer to that wire. For example, '`A`' is a method through which a 16-bit bus enters the top module. This bus can either carry 8 bits or 16 bits depending on which type of MAC operation is being done (configured based on `S1_or_S2`). You can decide which 8 bits to use out of the 16 bits for transmitting the 8-bit value in the former case.
- The isosceles trapezoid structures near the methods `A`, `B` , `C` are demultiplexers (DEMUX), and the one near `MAC` is a multiplexer (MUX). The MUX/DEMUX behavior is controlled by the wire coming from the '`S1_or_S2`' method.
- Please use the **'ROUND_TO_NEAREST'** rounding mode for the fp32 MAC computation. There is no need to take care of floating point exceptions.

4. Design Requirements

Implement the MAC module using Bluespec System Verilog (BSV), as mentioned in the beginning of this specification, **without using the + or * operators**. As a part of this assignment, you would have to implement the following design variants.

a) Implement the MAC module as an **Unpipelined** design.
b) Modify the implementation into a **Pipelined** design.

5. Verification Requirements

The verification of the top module has to be done using the **cocotb** framework, as described during the demo session. Sample test cases have been provided via 4 files per supported format (int8 and bf16). Each file contains 1000 different values, which are specified in binary format and detailed below.

- **S1 (int8):** *A_binary.txt* and *B_binary.txt* contain 8-bit signed integer values. *C_binary.txt* contains 32-bit signed integer values. *MAC_binary.txt* contains the 32-bit signed integer values that are expected to be produced by the S1 MAC operation.
- **S2 (bf16):** *A_binary.txt* and *B_binary.txt* contain bfloat16 values. *C_binary.txt* contains the fp32 values. *MAC_binary.txt* contains the fp32 values that are expected to be produced by the S2 MAC operation.

Please note that you will be given the leniency of 2 Least Significant Bits (LSBs) when comparing the expected values with the ones produced by your implementation. This means that even in the case that the 2 LSBs of the 32-bit MAC output produced by your module mismatches with that provided in the reference, the test case will be considered to be passed.

6. Submission Requirements

Students are requested to maintain a github repository where the source code and a detailed report, with description of microarchitecture and design/verification methodologies used, are uploaded. Kindly go through the demo github repository for reference.

7. Evaluation Criteria

- This design assignment would account for 10% of your final grade.
- Equal weightage would be given for evaluating both the design and the verification components.
  - For design evaluation, adhering to the specifications given and design/ implementation methodologies would hold significant importance.
  - On the other hand, reference model, coverage, and test cases used would be considered for evaluating your verification methodology.