MAC Design and Verification Report

Design Methodology

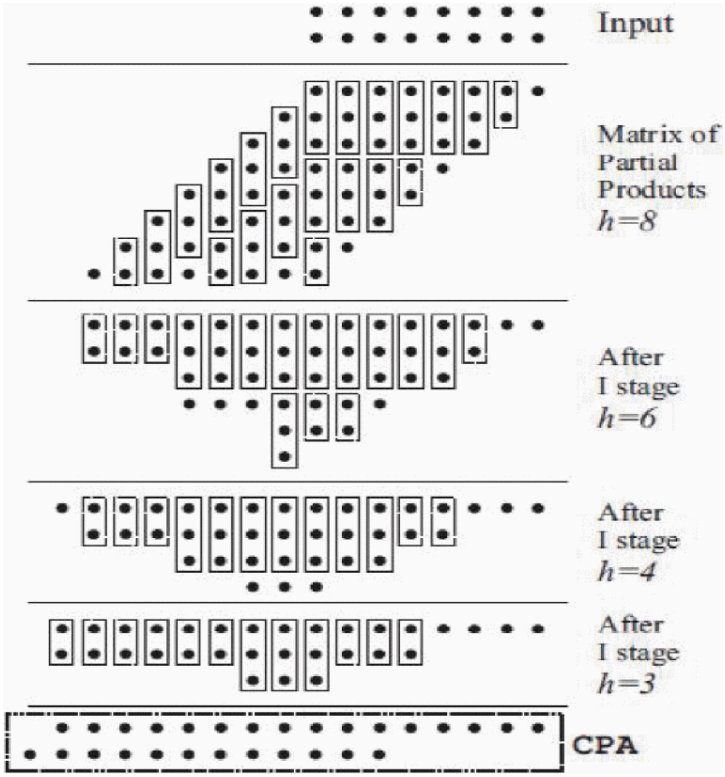
The objective is to design a Multiply-Accumulate (MAC) unit that supports the following operations:

i. **Integer Mode**: (A: int8, B: int8, C: int32) → (MAC: int32)
ii. **Floating-Point Mode**: (A: bf16, B: bf16, C: fp32) → (MAC: fp32)

The design includes two MAC modules, each dedicated to performing one of the operations above.

- **Design of Mac: Int Module**
  a. **Multiplication:** Implemented using a Wallace Multiplier with a Carry-Save Adder (CSA).
  b. **Accumulation:** Performed using a Ripple Carry Adder (RCA).

  Wallace Multiplier: The Wallace Multiplier is a hardware-efficient implementation designed to speed up binary multiplication. A naive approach to adding partial products results in a time complexity of O(log^2(n)), while the Wallace Multiplier reduces this to O(log(n)). Our implementation further optimizes the delay compared to the standard design.



  Accumulation: The accumulation phase uses a standard Ripple Carry Adder (RCA) to sum the result of the multiplication with the input accumulator value (C).

- **Design of MAC: Float Module**

a. **Multiplication:**
   - The bf16 floating point has 1 sign bit, 8 exponent bits and 8 mantissa bits with 1 bit being implicit
   - To obtain the multiplication output the sign bits are xor-ed, the exponents are added and the mantissas are multiplied using an unsigned version of the Wallace multiplier.
   - If the product of the multiplication is greater than 1 then the mantissa is normalized and the exponent is incremented by 1.
   - To convert the result to type bf16 the 16 bit product is converted to 8 bit mantissa
   - The first rounding is done here where rounding is done using round to nearest. This can be achieved by checking if the bits to be removed add up to 0.5 or not.

b. **Accumulation:**
   - The bf16 is converted to fp32 format by padding zeros to get a 24 bit mantissa with the implicit bit included.
   - Now the mantissas are aligned by taking the difference of the exponents and bit shifting the lower valued mantissa to the right.
   - Now the mantissas are added using a carry save adder and thereafter rounded using round to nearest.
   - While rounding we need to check if the rounding of the mantissa causes an overflow and if it does the exponent needs to be incremented by 1 and mantissa shifted to the right.
   - The final result is obtained by taking the the larger exponent and the resultant mantissa.

Verification Methodology

The verification involves a variety of inputs to check the validate the design these are

1. Walking ones and zeros.
2. Alternating ones and zeros.
3. Random values taken uniformly.
4. All zeros.
5. All ones.
6. Maximum and Minimum values.
7. Maximum signed value and Minimum signed value