TSEA26 Tutorial 3. MAC design

Frans Skarman [1]

November 24, 2021

[1] based on material by Dake Liu, Olle Seger, Andreas Ehliar, and Jian Wang

MAC introduction

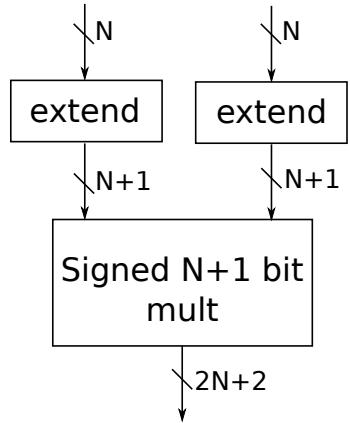
- Most important HW module in DP of any DSP Processor
- Supports Algorithms like
  - Convolution (most frequently used in DSP Algorithms)
    - Filtering, FIR, IIR, Auto-Correlation, Cross-Correlation
  - Transforms (FFT, DCT etc)
  - Double Precision Arithmetic Operations

MAC Building blocks

- Multiplier
- Long adder
- Accumulator Registers (ACR)
- Multiplexers
- Functions (e.g. Rounding, Scaling, Saturation, Flags etc)

Multipliers

We only use signed mulitpliers. Signed values are sign extended, unsigned values are zero extended



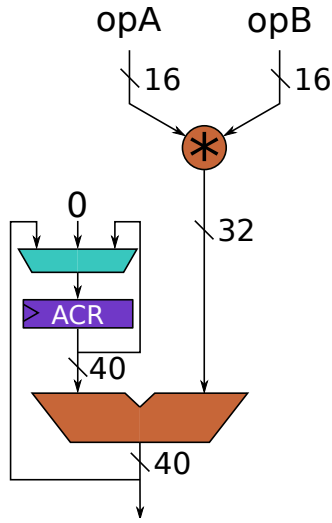
MAC Design: A case study

Design a MAC unit with the following operations

- Integer / Fractional multiplication (16 x 16)
- Signed / Unsigned multiplication
- Convolution with 8 guards and initialization
- Round
- Saturation
- 32 bits Long Plus and Minus
- 32 bits Long operation: ABS
- **There are plenty of ways to do this. This is one example**

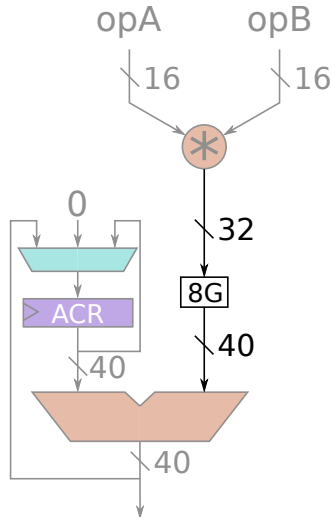
Basic design

- ACR is the accumulation register
- 0 for clear
- Feedback for nop



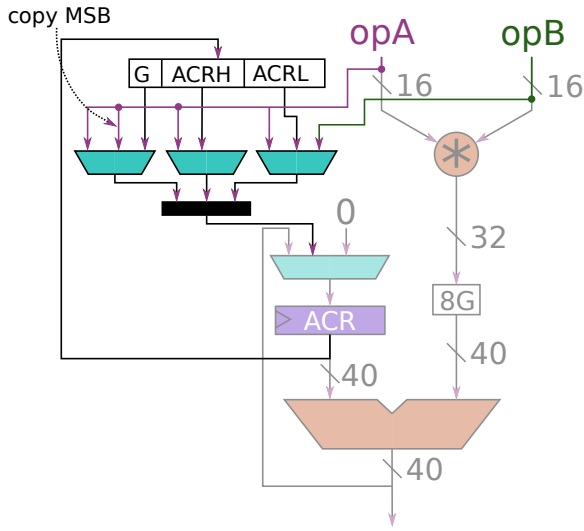
With guard bits

- G8 block adds 8 guard bits to input using sign extension



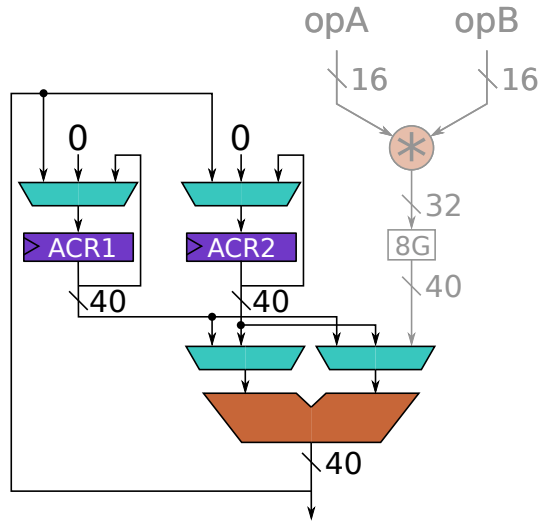
Load to ACR

- Load guard, ACRH or ACRL from RF
- Also allows load of OPA $\rightarrow$ ACRH, OPB $\rightarrow$ ACRL
- Auto-set G when loading ACRH
- **For clarity these changes are implicit in the rest of the slides**



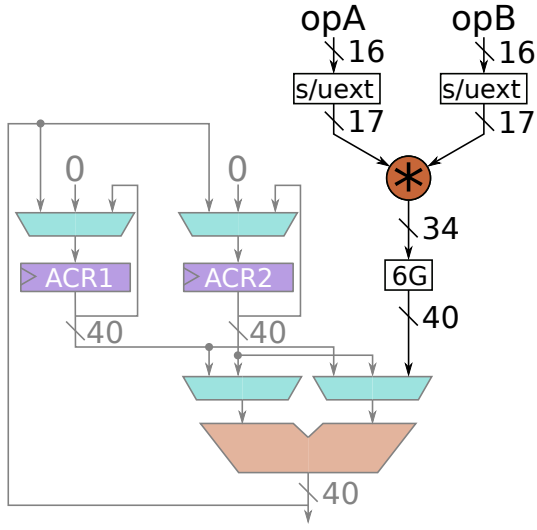
ACR1 and ACR2 to support long addition

- Two or more ACRs allows for long addition



Signed and unsigned multiplication

- Add sign extension discussed in the beginning
- Requires fewer “additional” guard bits



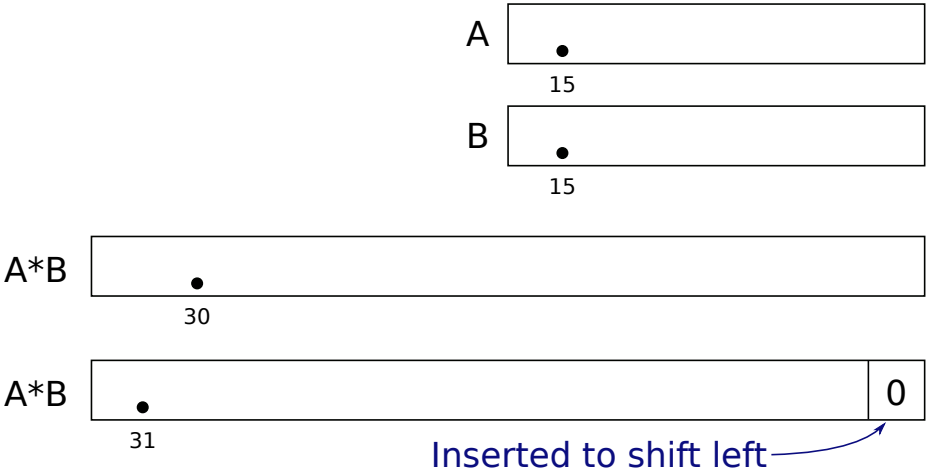
Fractional multiplication

Fractional multiplication "moves" fractional point
16 bit signed fractional multiplication:

$$x \cdot 2^{15} \cdot y \cdot 2^{15} = (x \cdot y) \cdot 2^{15+15} = (x \cdot y) \cdot 2^{30}$$

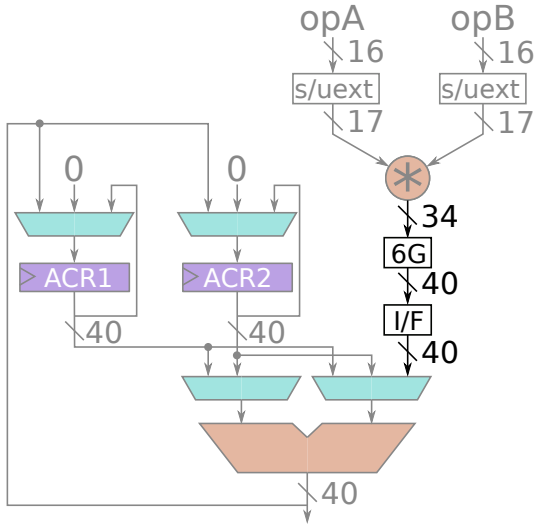
Normally we assume the fixed point in ACR to be at bit 31
$\Rightarrow$ We need to shift the result

Fractional multiplication



We'll build this this into a `I/F` block

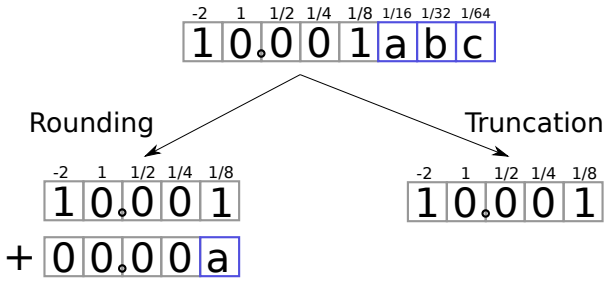
Fractional multiplication



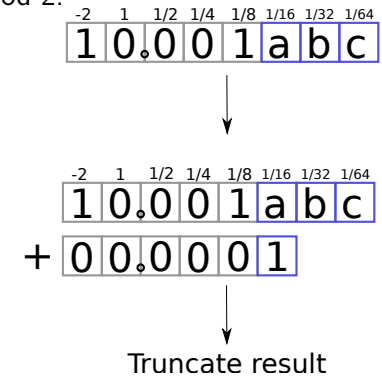
Rounding

Recall tutorial 1. Using method 2 here

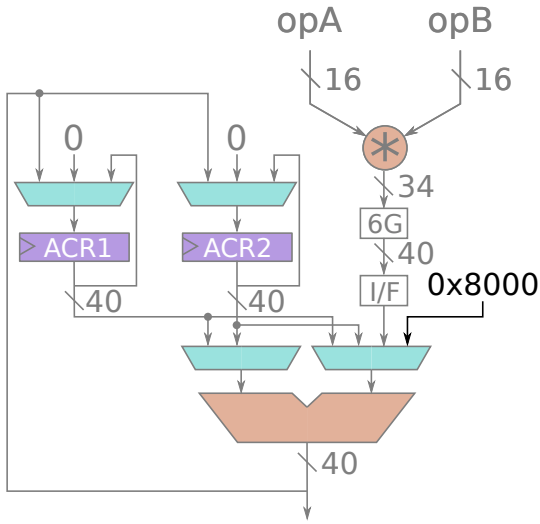
Method 1:



Method 2:

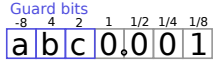


Rounding

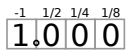


Saturation

- From tutorial 1
- Check guard bits for overflow

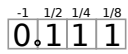


a=1, bc!=11



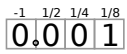
min

a=0, bc!=00

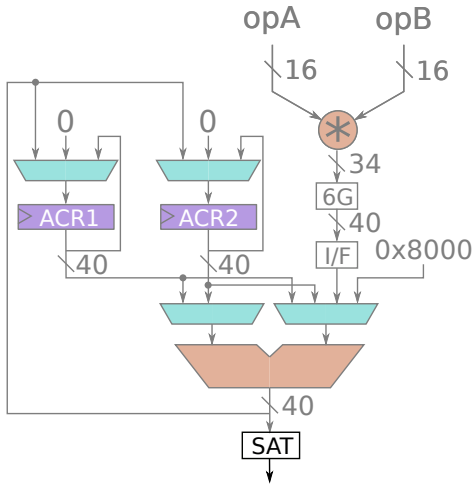


max

otherwise



Rounding



Subtraction, absolute value etc.

Same as for ALUs, See tutorial 2 for details.

Exercises

- Exercise 3.1 and 3.3 for “normal mac design”.
- Exercise 3.4, 3.2 and 3.5 for complex valued MAC