A Run-time Tapered Floating-Point Adder/Subtractor Supporting Vectorization

Ashish Reddy Bommana and Dr. Srinivas Boppu
School of Electrical Sciences
Indian Institute of Technology Bhubaneswar

Overview

- Introduction
- Motivation
- Design’s Traits
- Methodology
- Experimentation and Results
- Conclusion
- Future Scope

Introduction

- Many ML applications relied on Floating points (FP) arithmetic as these provide better accuracy and range.

- Over time, FP arithmetic accelerators have improved computational performance -- using reduced precision, mixed precision, SIMD optimizations, etc..,

- Tapered Floating Points (TFP) allows for variable-sized widths for exponents/mantissas for any precision (8/16/32/64-bit and so on).

- We propose our design -- performs addition and subtraction on four 8-bit TFP operands/two 16-bit TFP operands/one 32-bit TFP operand -- aimed to optimize/accelerate FP arithmetic.

Motivation

- SIMD optimizations, reduced precision training/inference, and mixed precision neural network training have been shown to enhance performance and optimize training area.

- Very Few designs support all these features/characteristics.

- What if we try to provide all these characteristics in one design?.

- Thus, we seek to design that allows mixed-precision, reduced-precision, and SIMD-enabled floating point arithmetic hardware..

Designs' Traits

- Our design takes in two 32-bit Floating points numbers and performs add/sub given any exponent width or the precision (8/16/32-bit).
- Supports vector inputs– four 8-bit, two 16-bit and one 32-bit floating points operands.
- Supports denormal number and NaN/infinite number arithmetic.
- Supports two modes of rounding schemes– tie to even and tie to odd.
- Dynamically changeable exponent width and format (8/16/32).

Methodology

- The basic outline of the designs consists of 5 steps. (based on chapter 16 of Synthesis of Arithmetic Circuits* book). Our design follows a similar way to carry out add/sub operations
  - Extractor – unpacks all operands' exponent, mantissa and sign bits
  - Aligner – Aligns the mantissas of all operands resulted from exponent differences.
  - Adder – Adds/subtracts the aligned mantissas of all operands.
  - Normalizer – Normalizes the mantissas resulted from adder stage. Parallelly calculates the exponent values if needed which is effected from normalization.
  - Optimized Rounder – Rounds the final value after normalization using tie-to-even or tie-to-odd rounding scheme.
- Our rounder is optimized to avoid re-normalization, therefore the tag Optimized.

*Deschamps, J.P., Bioul, G.J., Sutter, G.D.: Synthesis of arithmetic circuits: FPGA, ASIC and embedded systems. John Wiley & Sons (2006)

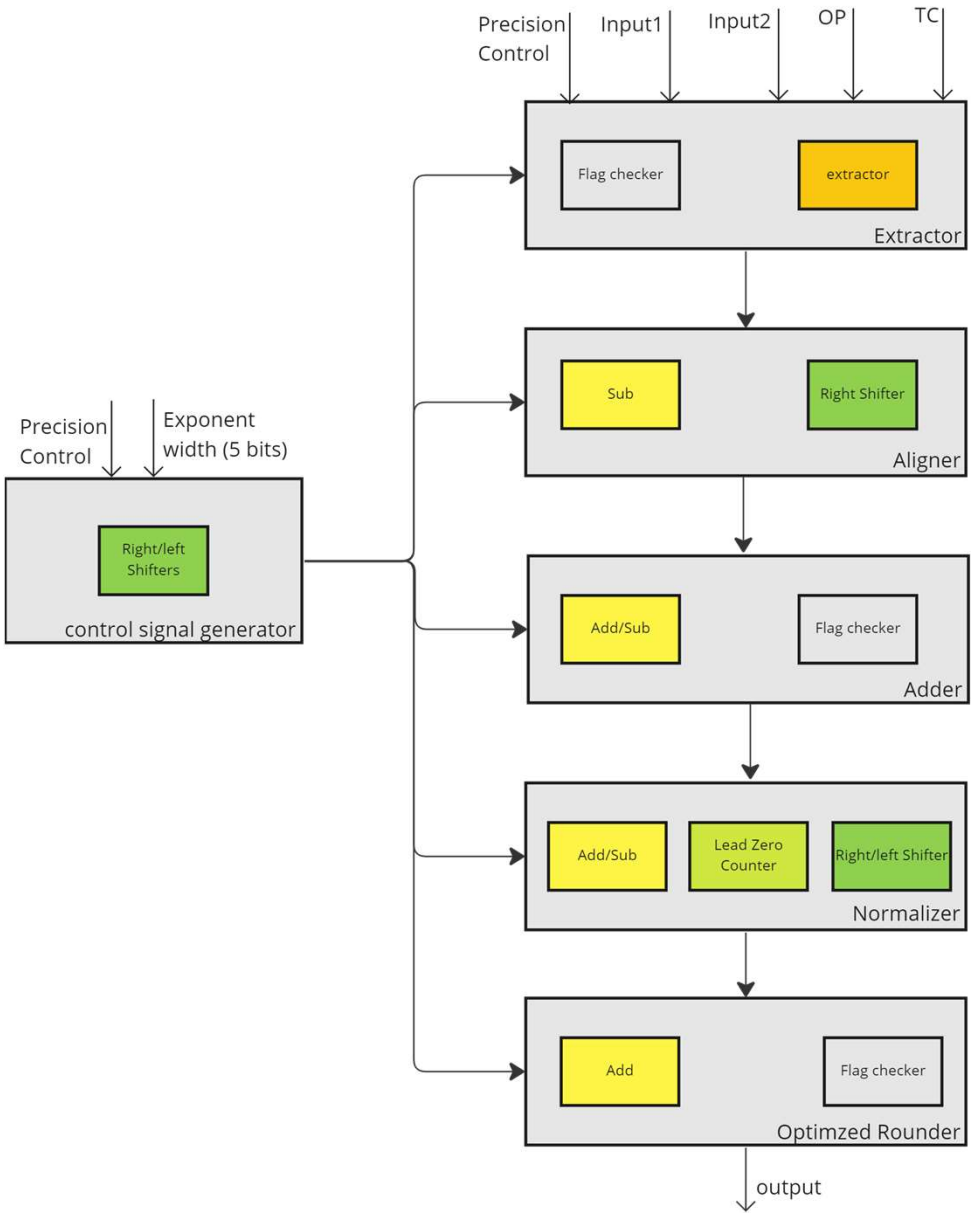
Methodology….Cont’d

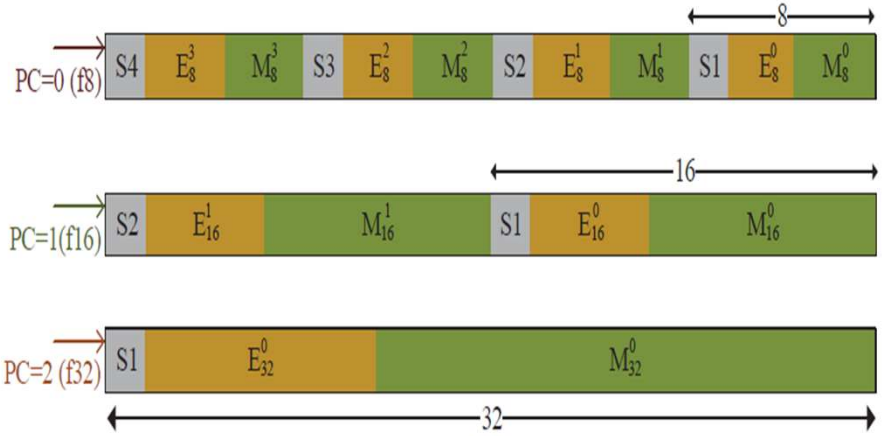
- To fulfill our design's needs - adders, shifters, and combinatorial blocks have been modified/redesigned/added.

- The core of the design is essentially the control signals -- produced within the design based on the exponent width.

- Our design handles infinite, NaN, de-normal instances – solved through Flag Checkers.

Methodology….Cont'd





For the exponent width of 5 -- the control signals resulted from the generator for each precision:

f8:

exponent control – 4{8'b01111100}

Mantissa control – 4{8'b00000011}

f16:

exponent control – 2{16'b0111110000000000}

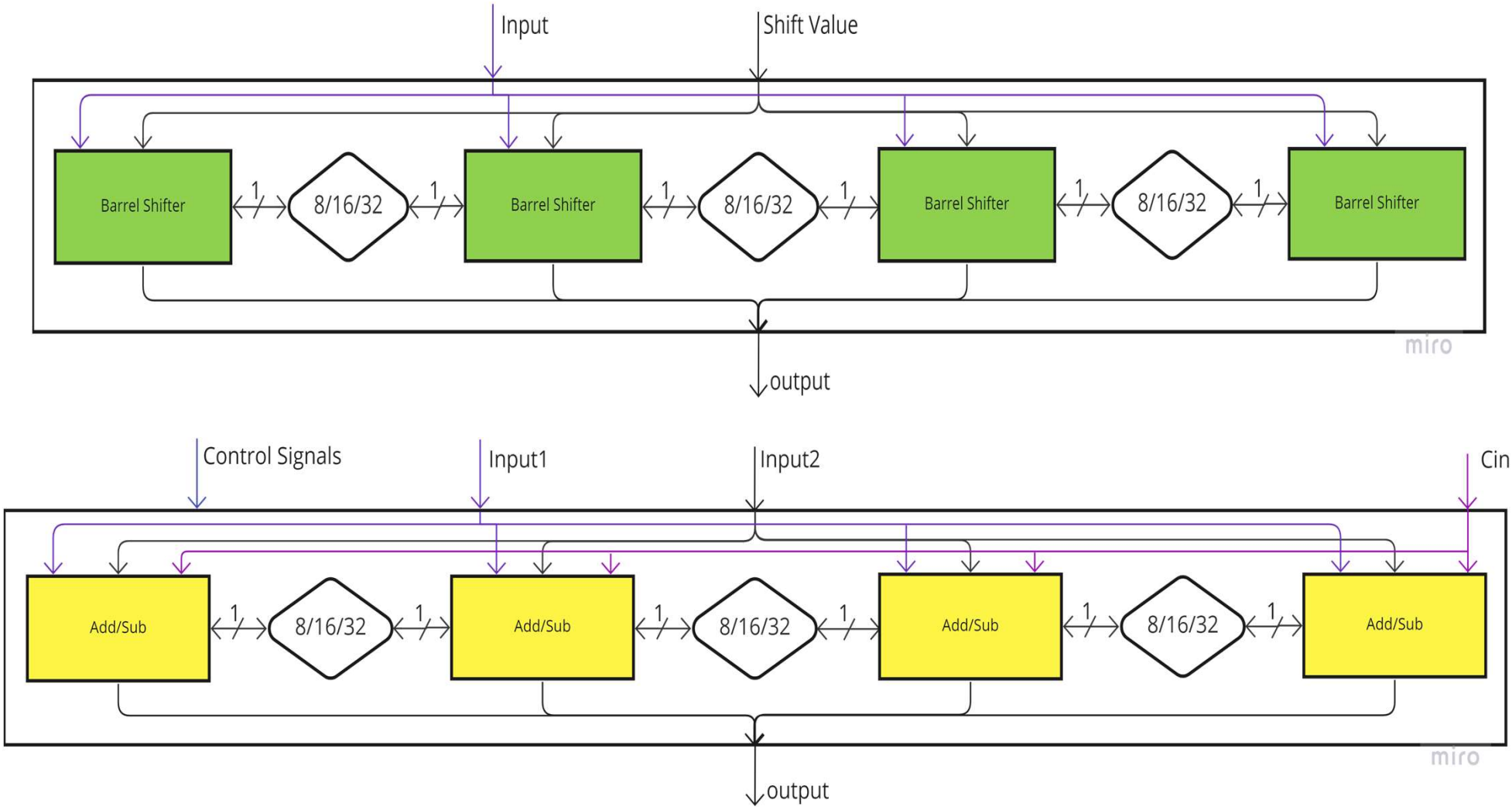
Mantissa control – 2{16'b0000001111111111}

f32:

exponent control – 32'b01111100000000000…0

Mantissa control – 32'b00000011111111111…1

Methodology….Cont'd



Experimentation and Results

- The design is based on the belief that it can accelerate neural network training, as demonstrated by prior research. As a result, we have tested its functioning and left practical testing to the future.

- An automated python framework test the functionality -- from generating inputs to compare the outputs of the design to the expected values.

- Compared our design to FPNew* (scaled down to add/sub operation) on area, frequency, power and resource utilization (LUTs, FFs) -- synthesized and tested on Basys3 board.

- The proposed was able to perform better in three of the parameters.

*Mach, S., Schuiki, F., Zaruba, F., Benini, L.: FPnew: An Open-source Multiformat Floating-point Unit Architecture for Energy-Proportional Transprecision Computing. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 29(4), 774–787 (2020)

Experimentation and Results … Cont'd

FP Format	Resources/Frequency	FPnew	Our Design
FP32S	Max. Frequency (MHz)	47	28
	LUT	1008	2156
	FF	397	514
	Power (mW)	72	86
FP16S	Max. Frequency (MHz)	55	28
	LUT	1020	2156
	FF	474	514
	Power (mw)	72	86
FP8	Max. Frequency(MHz)	65	28
	LUT	1160	2156
	FF	604	514
	Power (mw)	74	86
FP32_FP16S_FP8	Max. Frequency (MHz)	50	28
	LUT	3065	2156
	FF	1473	514
	Power (mw)	139	86

- FPNew offers a slice-based approach to perform arithmetic operations.
- To perform 32-bit SIMD FP operations, it uses 1 slice of FP32, two slices of FP16 and 4 Slices of FP8 hardware.
- If one format is selected the other corresponding slices are gated.

Conclusion

- Through this design we were able to design a reduced precision, mixed precision enabled SIMD hardware for Floating point Arithmetic operations(Add/Sub) at Run-Time.

- Our design has proved to be more area and power efficient compared to FPNew.

- We have developed an automated python framework to validate the functionality of our design.

Future Scope

- We intend to integrate our design to test on actual neural networks training.
- We intend to develop designs for multiplication and division operations.

THE END

THANK YOU !