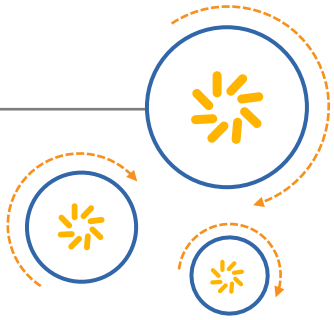

Qualcomm Technologies, Inc.

Qualcomm® Math (qmath) Library

80-VB419-105 A

December 7, 2017

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Contents

Figures

Tables

1 Introduction

The Qualcomm® math (qmath) library provides assistance with performing common math primitives on Qualcomm Hexagon™ Vector eXtensions (HVX). Initially, it provides HVX vector functions for emulated floating point operations.

1.1 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, `<number>`.

Commands to be entered appear in a different font, for example, **`copy a:*.* b:`**.

Button and key names appear in bold font, for example, click **Save** or press **Enter**.

1.2 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://createpoint.qti.qualcomm.com/.

If you do not have access to the CDMATech Support website, register for access or send email to support.cdmatech@qti.qualcomm.com.

2 HVX vectorized alternative to floating point

HVX is a fixed-point wide-vector engine for the Hexagon DSP. While HVX is designed for fixed-point processing, floating-point processing on the Hexagon remains non-vectorized and comparatively slow. For Hexagon workloads that are not feasible to convert to fixed-point arithmetic, this library offers a vectorized non-IEEE-compliant floating-point alternative to the scalar Hexagon floating point unit, which is significantly faster.

2.1 Pseudo-float numbers

The qmath library uses a different representation (that is not IEEE-compliant) for floating point numbers. A pseudo-float N-bit representation is comprised of two N-bit numbers:

- One number expresses the exponent as a 2's complement N-bit signed integer
- The other number expresses the mantissa as an S1.(N-2) number with no implied bits, i.e., a signed fixed-point number with 1 bit for the integer and N-2 bits for the fractional.

One pseudo float format is currently supported: pseudo32, which uses two 32-bit numbers to represent a value.

3 Contents of the library

The qmath library contains conversion functions between IEEE and pseudo-float formats, and HVX functions to perform vectorized math upon pseudo-float types.

There are some caveats:

- When IEEE floating point types (e.g. float, double) are converted to pseudo-float, all de-normalized numbers are converted to 0, and all infinities and NaNs are converted to a very large-magnitude number.
- When converting back to IEEE floating point types, all numbers larger than can be expressed in the destination format are assigned to +/- infinity. All numbers in the denormalized range are converted to 0.
- None of the math operations are fully IEEE-compliant.

The pseudo-float functions are available in both C-callable intrinsics functions and inlined intrinsics. The latter might perform better in loops, but use the inlined versions sparingly, because HVX register pressure can lead to degradation in performance and, ultimately, stack overflow.

- `inc/qmath.h` contains type definitions and function declarations, both for compiled and inlined forms.
- `src/qmath.c` contains the function implementations.
- `src/qmath_ref.c` contains the C-callable intrinsics functions. It also contains (in some cases only) plain-C reference versions, which are compiled out.

3.1 Accuracy and precision

The pseudo-float operations are constrained by the number of bits available for the mantissa. The pseudo-float format includes the sign bit and implied integer bit, which IEEE formats do not. Thus, single-precision IEEE float, which effectively has 23 fraction bits, has 25 bits of precision, compared to the 32 bits in pseudo-float.

Pseudo-float is not IEEE-compliant, which means it does not guarantee that every individual operation will produce the output equivalent to the infinite-precision result upon the same inputs, with proper rounding to single precision. Thus, pseudo-float can perform better than IEEE over a series of operations, due to the extra bits it has for intermediate operation. This is observed in the following experiments.

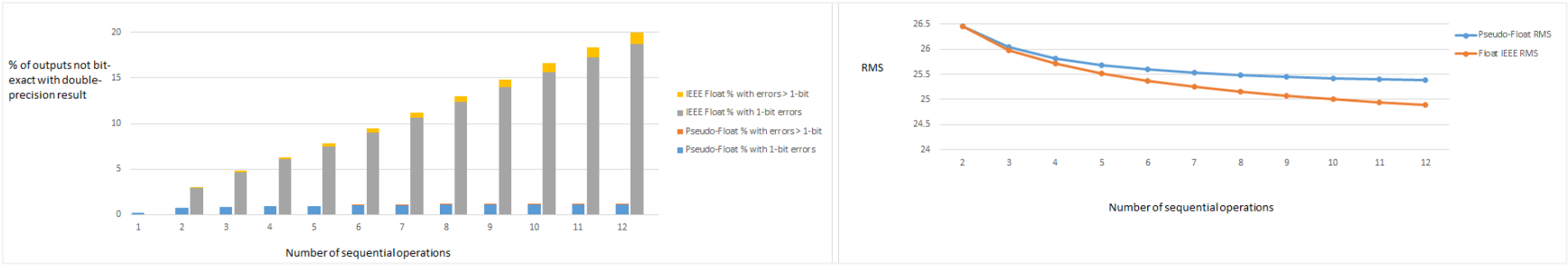

Figure 3-1 Accuracy of addition operations

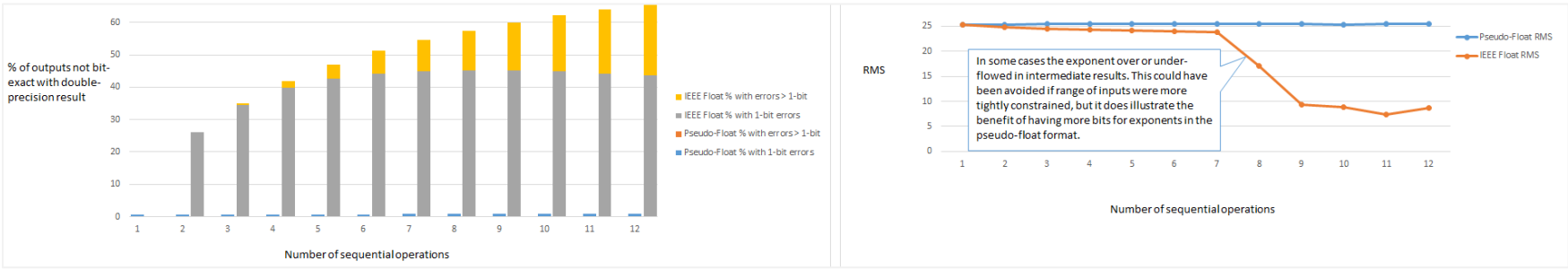

Figure 3-2 Accuracy of multiply operations

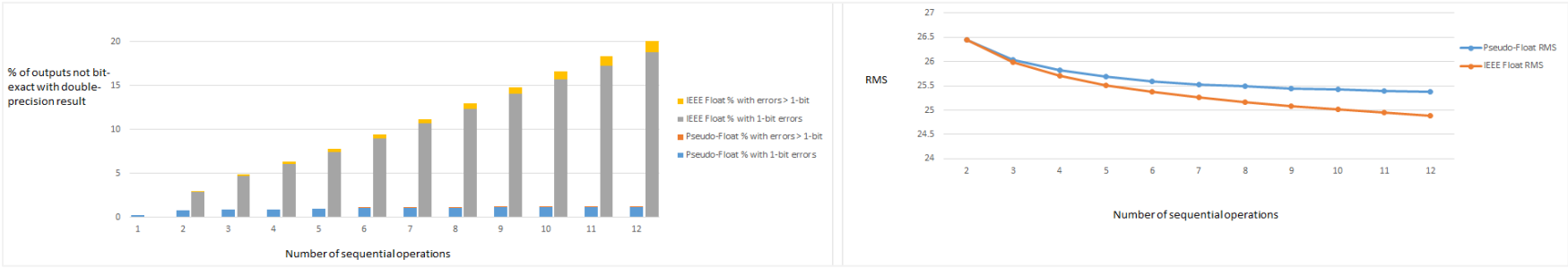

Figure 3-3 Accuracy of subtract operations

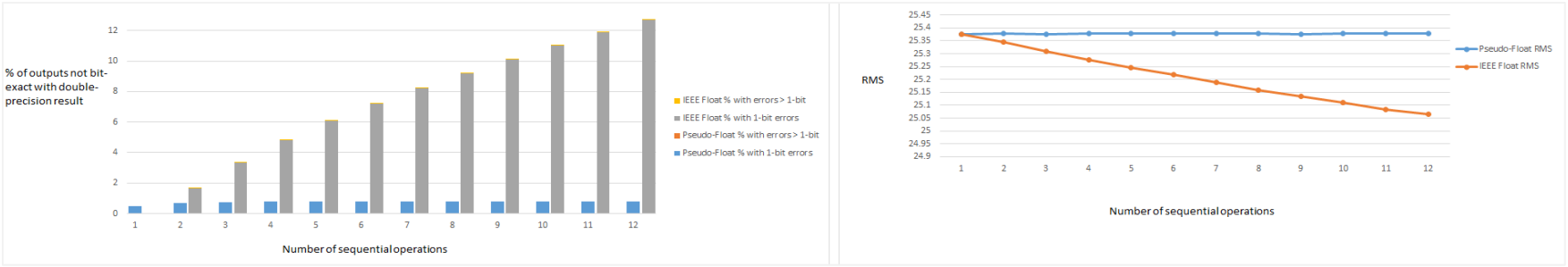

Figure 3-4 Accuracy of MAC operations

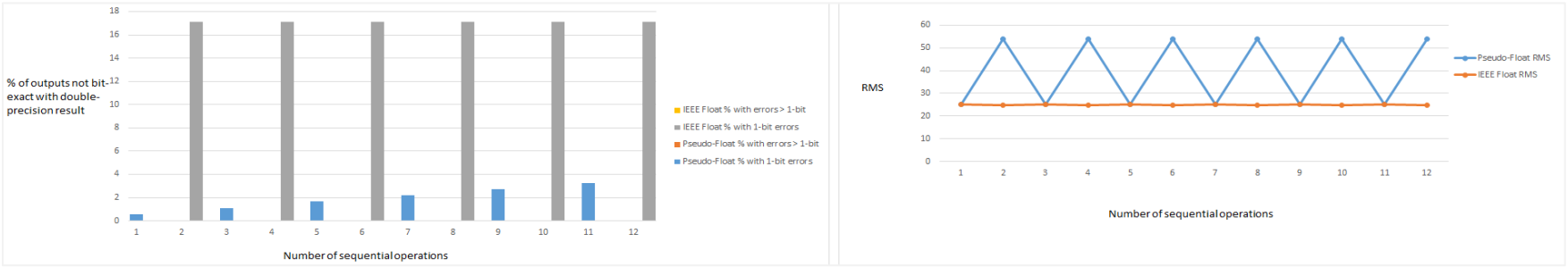

Figure 3-5 Accuracy of inverse (1/x) operations

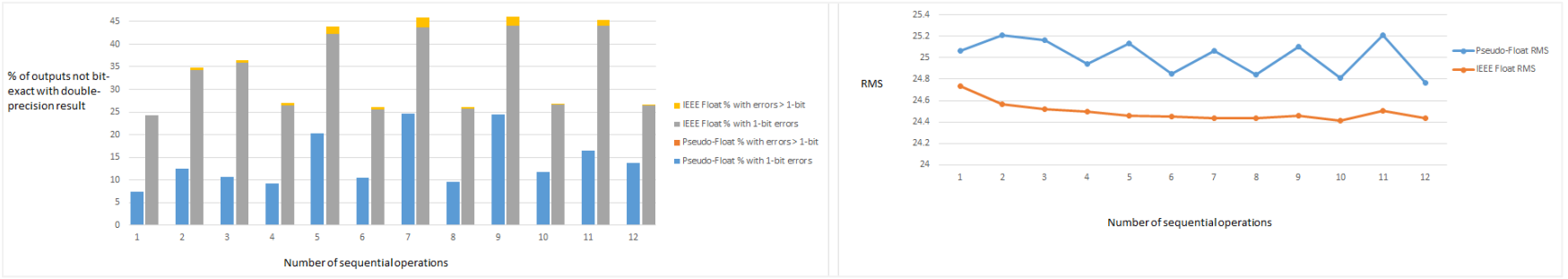

Figure 3-6 Accuracy of inverse square root operations

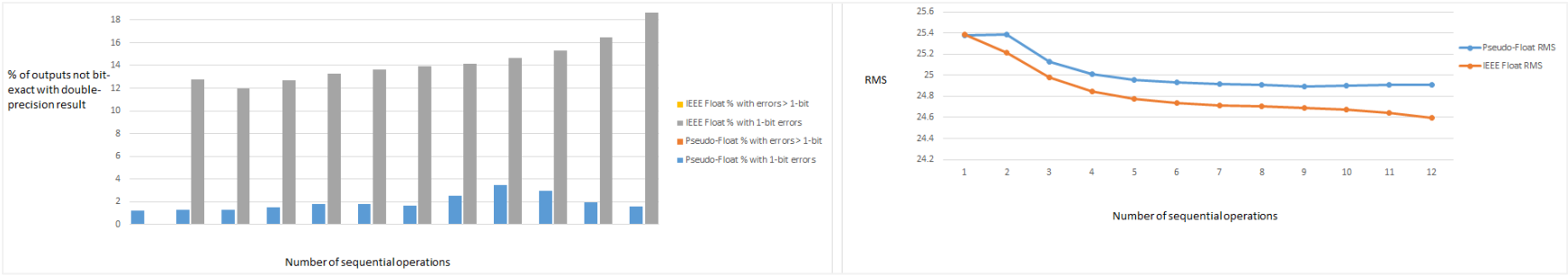

Figure 3-7 Accuracy of square root operations

3.2 Performance

Following is a comparison in the number of processor cycles consumed per float/pseudo-float operation, on SDM835 hardware, compared to the best possible native floating point performance. Conversions to and from pseudo-float are not included in the profiling, but they are roughly comparable in complexity to an add or sub operation. the 2x128 HVX configuration significantly out-performs even the 4-thread FP configuration, and it only uses 2 out of the 4 hardware threads.

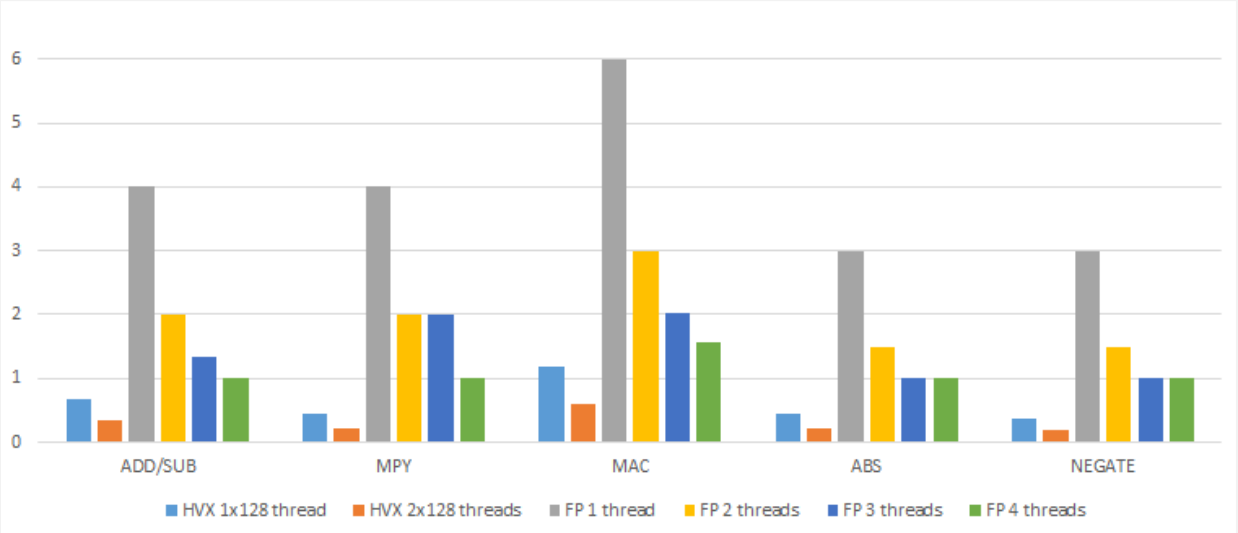

Figure 3-8 Processor cycles per floating point operation

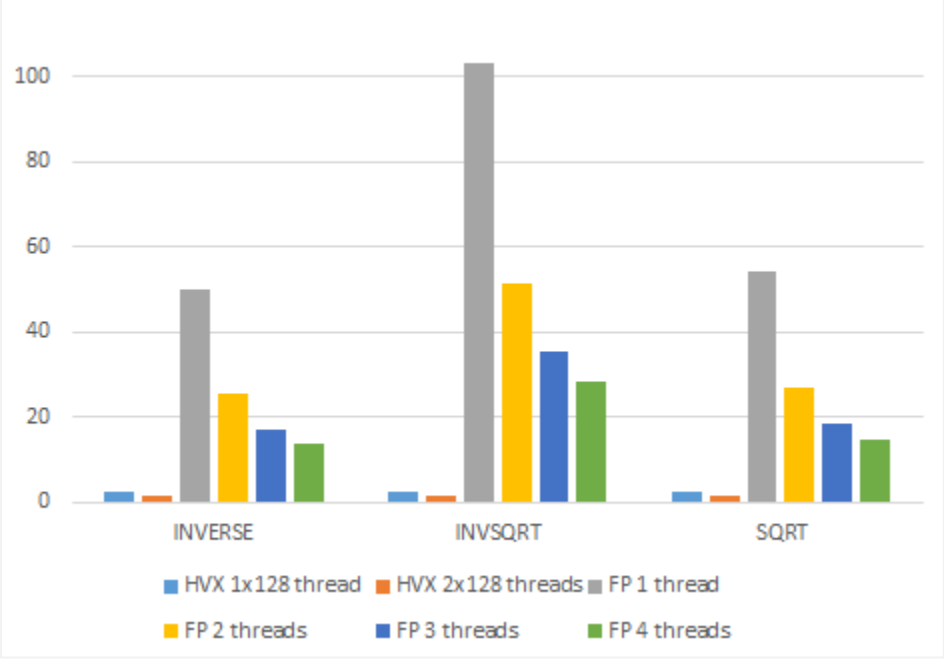

Figure 3-9 Processor cycles per floating point operation

Table 3-1 Performance

	ADD/SUB	MPY	MAC	ABS	NEGATE	INVERSE	INVSQRT	SQRT
HVX 1x128 thread	0.688065	0.437532	1.188294	0.438688	0.374981	2.563739	2.438853	2.438917
HVX 2x128 threads	0.343921	0.218751	0.594012	0.219284	0.18742	1.282122	1.21934	1.219387
FP 1 thread	4.000606	4.000144	6.000186	3.000142	3.000139	50.06125	103.0142	54.00867
FP 2 threads	2.000549	2.000097	3.000089	1.5001	1.500101	25.26934	51.62336	27.05094
FP 3 threads	1.336622	2.000387	2.024418	1.000237	1.000205	16.96311	35.46683	18.60039
FP 4 threads	1.005197	1.00045	1.572417	1.000201	1.000365	13.82259	28.11046	14.68771