Floating Point Tangent Implementation for FPGAs

(invited paper)

Martin Langhammer
Intel
High Wycombe, UK

Bogdan Pasca
Intel
France

***Abstract*—This paper presents an implementation of the floating-point (FP) tangent function, optimized for an FPGA containing hard floating point (HFP) DSP Blocks. This function inputs values in the interval $[-\pi/2, \pi/2]$, uses the IEEE-754 single-precision (SP) format, and has an accuracy conforming to OpenCL requirements. The presented architecture is based on a combination of mathematical identities and properties of the tangent function in FP. The resultant design outperforms generic polynomial approximation methods targeting the same resource utilization spectrum, and provides better resource trade-offs than classical CORDIC-based implementations. The presented work is widely available as part of the Intel DSP Builder Advanced Blockset.**

I. Introduction

Many hardware implementations of trigonometric functions use the CORDIC family of algorithms [1], [2]. Iterative implementations of CORDIC consume low resources, and are preferred when implemented as part of the FP unit of embedded processors. Higher throughput applications require unrolled versions, but these are recognized to be very stressful to support in FPGAs because of the multiple, deep arithmetic structures, with each level containing a wide adder. Chip-filling designs using such structures are usually unable to close timing at high frequencies [3].

Architectures based on polynomial approximations can be used to implement the sine, cosine and division operators, but additionally require the inverse function, with the attendent area and latency costs. [4]. These approaches map better to the recent FPGAs containing thousands of multipliers and embedded memory blocks, but can be quite wasteful when implementing the tangent function by means of operator assembly [5], [6].

In [7] we implemented a SP FP tangent as a *fused operator*, targeting fixed-point DSP Block resources. Here we extend our previous work to make use of of embedded HFP DSP Blocks available in newer devices. We show that significant resource savings can be achieved by making use of these new functions.

II. Algorithm

We have presented an algorithm for computing the single precision FP tangent in [7]. As the proposed architecture presented here adapts the previous ideas for use with HFP DSP Blocks, we will first summarize the key elements of the algorithm.

We focus on computing tangent over the $[-\pi/2, +\pi/2]$ interval. If the full FP range is required then a range-reduction phase can be implemented [8], [9]. We use symmetry to further reduce the input interval: $\tan(-x) = -\tan(x)$. Furthermore, we use the fact that for very small inputs ($< 2^{-w_F/2}$), a good approximation of $tan(x)$ is x. With this restricted dynamic range, a 36-bit fixed-point number can be used to represent the input SP value error-free.

We first recursively expand the identity of the tangent of the sum of two numbers into the tangent of the sum of three numbers:

$$\tan(a+b+c) = \frac{\frac{\tan(a)+\tan(b)}{1-\tan(a)\tan(b)} + \tan(c)}{1 - \frac{\tan(a)+\tan(b)}{1-\tan(a)\tan(b)}\tan(c)} \quad (1)$$

This can be approximated to:

$$\tan(a+b) = \frac{\tan(a)+b+\tan(c)}{1-(\tan(a)+b)\tan(c)} \quad (2)$$

provided that a, b and c, the components of the fixed-point input are selected as:

The reader is invited to consult [7] for a more detailed explanation of the approximation that studies the cancellation in the denominator.

III. Implementation targeting HFP FPGAs

The proposed architecture is presented in Figure 1 and is based on Equation 2. The input is split into the 3 segments, a, b and c. The first segments c and a are obtained by casting the input into fixed-point, and aligning the work towards the right, starting from weight 1, and using the exponent. The main difference compared to [7] is the smaller shifter size; for obtaining a and c we only use $9+9$ bits (including the hidden 1) when performing the alignment, and the right shifter discards bits with weight smaller than 2^{-17}. From the respective 9 bits of a and c, the values $\tan(a)$ and $\tan(c)$ are tabulated in SP from the 9 bits of a and c, respectively.

The objective is to obtain b in FP, as this will allow for an efficient implementation of both numerator and denominator.

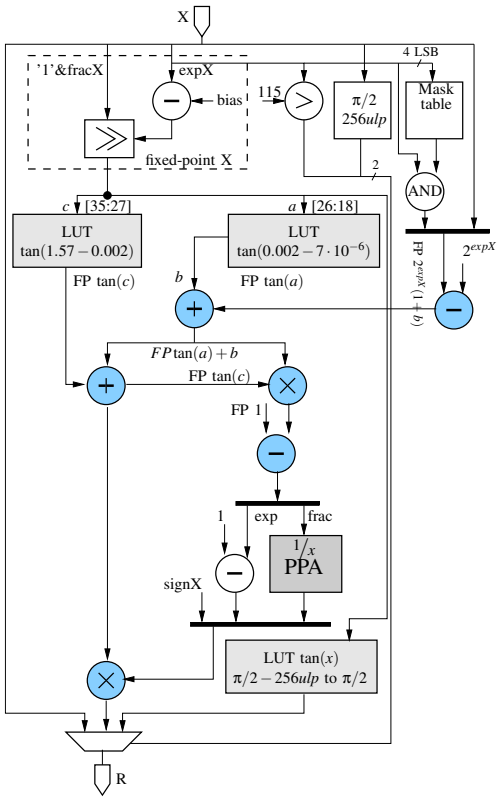

Figure 1. SP FP tangent utilizing FPGA FP DSP Blocks

We first apply a mask on $fracX$, the 23 fractional bits of our input. The mask value is read from a table indexed by the 4 least significant exponent bits. The table entries are 23-bit values with the following format (from MSB to LSB): (2+i zeros, 21-i ones). By applying this mask to the fraction of X we obtain the constituent components of b, but a second step is required on the FP value formed from (0, expX, fracPostMask) to extract b. This second step requires creating the variable (0, expX, 0) and pernforming the operation:

$$b = (0, expX, fracPostMask) - (0, expX, 0)$$

The numerator is obtained by adding $\tan(a)+b$, then adding this value to $\tan(c)$, all in FP SP. The right-hand side term in the denominator, $\tan(c)(\tan(a)+b)$, is also obtained by performing the multiplication in FP, and reusing the result $\tan(a)+b$. Finally, the subtraction is also performed in FP to calculate the denominator.

The piecewise-polynomial approximation-based inverse calculation requires a function $f(x) = 1/1+x$ with $x \in [0,1)$ and $IM(f) \in [0.5,1)$. The function f is simply obtained by making $x = fracDenom$. The image of the function might be 1 in two situations: (i) if the input is 1, or (ii) if the rounding of the inverse overflows to 1. Both cases are captured by checking bit weight $1/2$, which forces the output to 1 if not set.

This has implications on setting the output exponent. The exponent of the inverse denominator would normally be calculated as $2bias - expDenom$, followed by a possible decrement if the fraction inverse is < 1. This latter update is automated in one single step, by changing the $2bias$ constant to newCst (note that u is bit weight $1/2$)

$$newCst = 111111\widetilde{uu}$$

Table I
SYNTHESIS RESULTS FOR RECENT FPGA ARCHITECTURES

Target	Architecture	Lat	Freq.	Resources
Arria 10	[7]	36	520MHz	8 DSPs, 6 M20K, 990ALMs
Arria 10	ours FP	32	487MHz	8 DSPs, 6 M20K, 393ALMs
Arria 10	$\tan(\pi x)$ [10]	37	545MHz	10DSPs, 3 M20K, 1935ALMs

IV. RESULTS

Table I presents the synthesis results for our proposed implementations obtained using Quartus Prime 17.0, fastest speed-grade. Compared to our previous work in [7] the proposed implementation consumes significantly fewer ALMs, and has a shorter latency. Compared to an FPGA specific piecewise-polynomial implementation of $\tan(\pi x)$ in [10] (which is expected to be 100-200LUT smaller for a limited input range) our proposed architecture requires significantly fewer ALMs.

V. CONCLUSION

The HFP resources available in the DSP block provide new levels of utility. Significant savings can be achieved by restructuring fixed-point computations to make use of the new HFP resources. Compared to our previous work, we have migrated almost all computations into DSPs and memory blocks. This improves performance in chip-filling designs by having a more predictable place-and-route.

REFERENCES

[1] Y. Shang, "Implementation of ip core of fast sine and cosine operation through FPGA," *Energy Procedia*, vol. 16, Part B, no. 0, pp. 1253 – 1258, 2012, 2012 ICFEEM. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187661021200210X

[2] E. Garcia, R. Cumplido, and M. Arias, "Pipelined CORDIC design on FPGA for a digital sine and cosine waves generator," in *Electrical and Electronics Engineering, 2006 3rd International Conference on*, sept. 2006, pp. 1 –4.

[3] "An Independent Analysis of Altera's FPGA Floating-point DSP Design Flow," 2011. [Online]. Available: http://www.bdti.com/MyBDTI/pubs/2012_Altera_FloatingPoint_Design.pdf

[4] B. Pasca, "Correctly rounded floating-point division for DSP-enabled FPGAs," in *22th International Conference on Field Programmable Logic and Applications (FPL'12)*. Oslo, Norway: IEEE, Aug. 2012.

[5] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design and Test*, 2011.

[6] M. Langhammer and T. VanCourt, "FPGA floating point datapath compiler," *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, vol. 17, pp. 259–262, 2009.

[7] M. Langhammer and B. Pasca, "Faithful single-precision floating-point tangent for FPGAs," in *Proceedings of the 2013 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2013, pp. 39–42.

[8] J. Detrey and F. de Dinechin, "Floating-point trigonometric functions for FPGAs," in *International Conference on Field Programmable Logic and Applications*. Amsterdam, Netherlands: IEEE, aug 2007, pp. 29–34.

[9] M. H. Payne and R. N. Hanek, "Radian reduction for trigonometric functions," *ACM SIGNUM Newsletter*, vol. 18, no. 1, pp. 19–24, Jan. 1983.

[10] "DSP Builder Advanced Blockset," https://www.altera.com/products/design-software/model---simulation/dsp-builder/overview.html.