Faithful Single-Precision Floating-Point Tangent for FPGAs

Martin Langhammer, Bogdan Pasca
Altera European Technology Centre
High Wycombe, UK

***Abstract*—This paper presents an FPGA-specific implementation of the floating-point tangent function. The implementation inputs values in the interval $[-\pi/2, \pi/2]$, targets the IEEE-754 single-precision format and has an accuracy of 1 *ulp*. The proposed work is based on a combination of mathematical identities and properties of the tangent function in floating point. The architecture was designed having the Stratix-IV DSP and memory blocks in mind but should map well on any contemporary FPGA featuring embedded multiplier and memory blocks. It outperforms generic polynomial approximation targeting the same resource spectrum and provides better resources trade-offs than classical CORDIC-based implementations. The presented work is widely available as being part of the Altera DSP Builder Advanced Blockset.**

I. Introduction

Many hardware implementations of trigonometric functions use the CORDIC family of algorithms [16], [11]. Iterative implementations consume low resources and are preferred when implemented in the floating-point unit of embedded processors. Unrolled implementations are often encountered in computational datapaths targeting high throughputs. These are recognized to be very stressful to support in FPGAs due to 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 fmax [3].

Architectures based on polynomial approximations can be used to implement the sine, cosine and division based on the inverse [14]. These approaches map better to the recent FPGAs containing thousands of multiplier and embedded memory blocks but can be quite wasteful when implementing the tangent operation by operator assembly [7], [12]

In this work the floating-point tangent function is implemented as a *fused operator*. We present a step-by-step error analysis which allows optimizing the internal operations by computing just-right for obtaining a faithfully rounded implementation. The results presented in Section V show the superiority of this approach compared to the best polynomial approximation implementations.

II. Background

The IEEE-754 standard for floating-point arithmetic (revised in 2008) [1] represents binary floating-point numbers using a three element tuple: sign (1 bit), exponent ($w_E$bits) and fraction ($w_F$bits) - (s, e, f) such that: $x = (-1)^s 2^e 1.f$. The widths of the fields defines the supported formats: (1, 8, 23) for single precision and (1, 11, 52) for double precision.

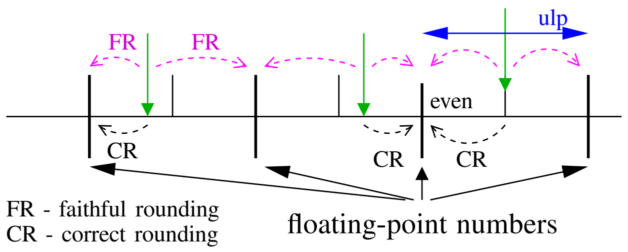

Fig. 1. IEEE-754 correct rounding for round to nearest and the non-standard and faithful rounding

Let x, y and q be floating-point numbers such that $q = \circ(x \text{ op } y)$ where $(\circ)$ denotes rounding an infinitely accurate result to the target format. Rounding to nearest is desired as it provides the best accuracy (maximum error of 1/2 *ulp* where the *ulp* denotes the unit in the last place). For elementary functions round to nearest is very difficult to obtain - problem called the Table Maker's Dilemma [13]. For these functions implementations diverge from the standard and implement the relaxed *faithful rounding* (Figure 1) with a maximum error of $1ulp$.

In this work we present a multiplier and memory based architecture of the floating-point tangent function having an accuracy of 1 ulp.

Recent Altera FPGAs such as Stratix-III/-IV [2], [4] contain thousands of such embedded resources. One Stratix-IV half-DSP block (each half can be configured independently) can be configured to implement either 2 18×18 multipliers or one 36×36 multiplier. Embedded memory blocks can also have various configurations, most relevant for this work being the 256×36 and 512×18 modes of the M9K block.

III. Algorithm

We present here the algorithm used to compute a faithfully accurate floating-point tangent in single precision, where the input interval is restricted to $[-\pi/2, +\pi/2]$. This type of operator can be directly used in datapaths where we can bound beforehand the range of the the input variable. In order to obtain an operator for the full floating-point input range a supplementary range-reduction step is required [9], [15] which is currently outside the scope of this work.

Tangent is symmetrical to the origin: $\tan(-x) = -\tan(x)$; this allows restricting the computation to positive arguments $[0, \pi/2)$.

The Taylor expansion for the tangent is

$$\tan(x) = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + ... \; x \in \left(\frac{-\pi}{2}, \frac{\pi}{2}\right) \quad (1)$$

If x is very small ($< 2^{-\mathrm{w_F}/2}$) a good approximation for $tan(x)$ is x. This is due to the fact that the higher order terms in Equation 1 have weights lower than the LSB of x and are shifted-out in the final summation. This is a well known property of the floating-point sine and tangent functions. The dynamic range of the input is therefore limited to $[2^{-\mathrm{w_F}/2}, +\pi/2]$, allowing the input to the function to be represented in an error-free fixed-point format with $1 + \mathrm{w_F} + \lceil \mathrm{w_F}/2 \rceil$ bits (24+12=36 bits for single precision).

The following mathematical identity holds true for tangent:

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

this can be expanded to:

$$\tan(a+b+c) = \frac{\dfrac{\tan(a)+\tan(b)}{1-\tan(a)\tan(b)} + \tan(c)}{1 - \dfrac{\tan(a)+\tan(b)}{1-\tan(a)\tan(b)}\tan(c)} \quad (3)$$

Building a faithfully accurate floating-point tangent requires the final error to be less than $1ulp$. The final error has two components: $E_{\text{total}} = E_{\text{approx}} + E_{\text{round}}$. The rounding error ($E_{\text{round}}$) is a bound on the maximal error done when packing a possibly infinitely accurate result into the target format. Rounding to nearest will yield a maximal error of $1/2ulp$. The approximation error (E_{approx}) sums the both the method errors and the errors due to implementation optimizations (datapath trimmings). The objective is to keep the sum of these errors smaller than $1/2ulp$.

The proposed architecture is based on Equation 3, which is implemented as a floating-point multiplication between the numerator (n) and the inverse of the denominator (id): $p = n \times id$. In the following equation, tilde variables are approximations. The approximation error is computed as $E_{\text{approx}} = |(\widetilde{p} - p)/p|$.

$$\begin{aligned}
p &= n \times id; \\
\widetilde{p} &= \widetilde{n} \times \widetilde{id} \\
&= n(1+\epsilon) \times id(1+\epsilon) \\
&= n \cdot id + 2 \cdot n \cdot id \cdot \epsilon + n \cdot id \cdot \epsilon^2. \\
E_{\text{approx}} &= |(p - \widetilde{p})/p| \\
&= |(2 \cdot n \cdot id \cdot \epsilon + n \cdot id \cdot \epsilon^2)/(n \cdot id)| \\
&= |2\epsilon + \epsilon^2| \leq 1/2 \cdot 2^{-p} \quad (4)
\end{aligned}$$

From inequality 4, where $p = 24$ for single-precision results that the value of ϵ should be slightly smaller than 2^{-26} which translates into slightly better than $1/4ulp$ error bound for both the numerator and denominator.

The complexity of computing the numerator in Equation 3 to the required accuracy can be reduced, for single-precision, by using the fixed-point decomposition presented in Figure 2. This decomposition favours tabulating the tangent computations for $\tan(a)$ and $\tan(c)$ using embedded memory blocks.

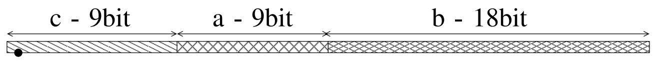

Fig. 2. The fixed-point decomposition of the input argument x

In addition, as both $\tan(a)$ and $\tan(b)$ are small, $\tan(a)\tan(b)$ is also very small. Moreover, as $b < 2^{-17}$ it is safe to use $\tan(b) \approx b$. Therefore, we will use the follwing approximation for computing the numerator:

$$\mathrm{n} = \tan(c) + \tan(a) + b \quad (5)$$

We need to compute a bound on the error of this approximation. We do this for two cases. First, $\tan(c) = 0$ and $\tan(a)\tan(b)$ maximal:

```
a =   .         111111111
b =   .                   111111111111111000
```

In this case the relative error is slightly less than 2^{-25}, and should be 2^{-26}. However, in this case the denominator value is 1 and carries no error, so the accuracy is reached. Second, for $\tan(c)$ minimal but greater than zero and $\tan(a)\tan(b)$ maximal, the value for $\tan(a)$ has a lower weight than $\tan(c)$, which pushes the relative error in the summation to 2^{-26}. Consequently, computing both $\tan(a)$ and $\tan(b)$ with $1 + \mathrm{w_F} + 2$ bits of accuracy suffices.

The denominator will be computed using a similar approximation as for the numerator.

$$\mathrm{d} = 1 - (\tan(a) + b)\tan(c) \quad (6)$$

This computation involves a possible cancellation which can amplify an existing error by an amount equal to the cancellation size. This would require computing the subtracted term with additional accuracy. As the cancellation occurs when the input is close to $\pi/2$ we use an additional table for the final $256ulp$ before $\pi/2$. Hence, the largest cancellation can now be produced by the following input:

```
c = 1.10010010;
a =   .         000111001;
b =   .                   010000;
```

The cancellation size is 3 bits, and therefore we require 3 additional bits of accuracy in computing the right term of the subtraction. This requires computing both $\tan(a)$ and $\tan(c)$ with $1 + \mathrm{w_F} + 2 + 3$ bits of precision and $0.5ulp$ of accuracy.

IV. Implementation

The implemented architecture is presented in Figure 3. As a and c are small subsections (9 bits) of the fixed point input, the tangents for all possible bit combinations can be stored in 36-bit wide data tables:

- $\tan(c)$ has a dynamic range between 2^{-8} and 2^{11}. It will be stored in a normalized floating-point format with an exponent on 5 bits and a fraction on $\mathrm{w_F} + 5$ bits. The hidden '1' is also stored explicitly in order to save on the decoding logic for 0. The total data width in this table is therefore 34 bits (M9K have a width of 36-bits).

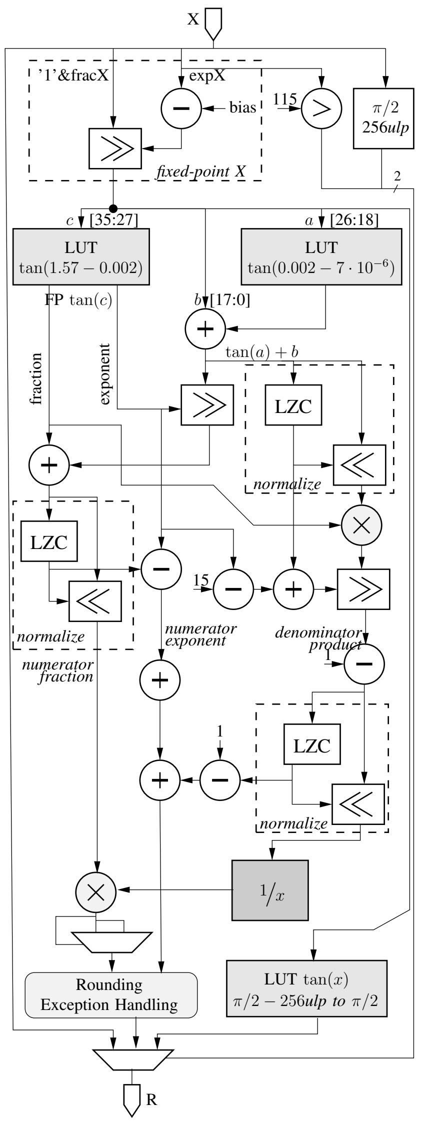

Fig. 3. The architecture for the faithful single-precision floating-point tangent

- $\tan(a)$ has a dynamic range of just 9 positions. We store this value directly in fixed-point for a total width of $9 + 23 + 5 = 36 + 1$ which one bit wider than the M9K block in Stratix-IV/-III devices.

The problem has now been reduced to a 36 bit fixed-point multiply and a 36 bit fixed-point divide. There are also some additions required: although they will be a form of floating point, explained below. The subtraction will be a simple fixed-point subtraction.

The input number is first converted to a 36 bit fixed-point number by shifting the difference between the number and the maximum biased input exponent. The fixed point input is then split into three numbers: c - bits (35 downto 27), a - bits(26 downto 18), and b, the least significant 18 bits.

The numerator is calculated largely on the left side of Figure 3. The $\tan(a)$ and b numbers are in fixed-point format, and can immediately be added together. The $\tan(a) + b$ sum may grow by one bit when both a and b approach their maximums. The sum must then be aligned to the exponent of c, which can range from 0 to 19 (127 to 146 in single precision offset equivalent).

After right shifting the sum $\tan(a) + b$ it can finally be added to the mantissa of c. Next, $\tan(c) + \tan(a) + b$ sum is then normalized. The numerator now exists in a floating point format.

The denominator is calculated largely on the right side of the diagram. The $\tan(a) + b$ sum is normalized before the multiplication by the mantissa of $\tan(c)$. The local exponent is used to denormalize the product to a fixed-point number again. The now fixed-point value of $(\tan(a) + b) \cdot \tan(c)$ is subtracted from 1, and is normalized again before the division. The maximum cancellation size is 3 positions which simplifies the normalizer implementation.

As presented in the error analysis section, the final division is implemented as an inverse of the denominator, which is then multiplied by the numerator. The division uses a normalized denominator, and the multiplier input is a normalized numerator. The product therefore requires only a single bit normalization, which is implemented as a 2-1 multiplexer. Finally, a rounding stage, along with the application of special or signalling conditions, is performed.

If the biased input exponent is less than 115, the output and the input are considered the same, as this is the point where $\tan(x) = x$. This is implemented as an input in the final multiplexer in Figure 3. If the input is within the $256ulp$ prior to $\pi/2$ a tabulated value is used, which is also an input in the final multiplexer.

V. Results

Table I presents the synthesis results for our proposed implementation on a Stratix-IV C2 speedgrade FPGA. The *Resources* numbers are given as returned by QuartusII: number of DSP blocks is given in terms of 18-bit multipliers (4 18-bit multipliers compose a 36 x 36 block). The 18 18-bit multipliers also comprise the units needed for implementing the inverse calculation.

We first compared our proposed solution against the $\tan(\pi x)$ implementation available in Altera DSP Builder Advanced [5] block set. This implementation also performs a simple range reduction, so it is expected to be 100-200LUT smaller when the input range is limited to $-\pi/2$ to $\pi/2$. Nevertheless, the proposed fine-tuned implementation has a significantly shorter latency, consumes fewer multipliers, roughly the same number of memory blocks and less than half the logic.

We have also compared this implementation against the $\sin\cos$ operator presented in [10]. This is a combinatorial design, targeting a VirtexII-Pro and uses a fused $\sin\cos$ operator. Similar to [5], it inputs an argument of πx which slightly penalizes the implementation compared to ours. In the bottom part of Table I we give two top divider implementations, each targeting different resources spectrum: the FloPoCo [7] divider uses a digit recurrence method whereas the DSPBuilder Advanced version [14] uses polynomial approximation. The full tangent implementation based on [10] would require an extra division. Additionally, pipelining the implementation is

TABLE I
SYNTHESIS RESULTS FOR STRATIX-IV C2. MUL = 18-BIT MULTIPLIERS

Architecture	Lat @ Freq.	Resources
ours	30 @ 314MHz	18MUL, 8M9K, 1172LUT, 1078Reg
$\tan(\pi x)$ [5]	48 @ 360MHz	28MUL, 7M9K, 2633LUT, 4099Reg
$\sin\cos(\pi x)$ [10]	85ns	10 MUL, 2*1365 LUTs
div [7]	16 @ 233MHz	1210LUT, 1308REG
div [14]	11 @ 400MHz	8MUL, 4M9K, 274LUT, 291Reg

also expected to increase resource usage. All this considered, the proposed architecture manages to outperform this implementation.

VI. CONCLUSION

We have presented the architecture of a faithfully accurate single-precision tangent. Unlike previous works, the tangent is viewed as a fused operator which, in combination with a careful error analysis, allows significantly reducing implementation cost. The implementation targets recent Stratix-III/-IV devices and resizes the internal datapath in order to: 1/ use the larger multiplier granularity – one 36-bit multiplier is equivalent in cost to two 18-bit ones 2/ make good use of the available block memory size.

Future work includes generalizing these techniques for larger precision. In such a case $\tan(c)$ and $\tan(a)$ will be too large for direct tabulation. Using the HOTBM approach by Detrey and de Dinechin [8] would allow computing these to higher precisions with little cost. The larger multipliers needed in these operators would benefit from the truncated multiplier techniques presented by Banescu et. al. [6].

REFERENCES

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1–58, 29 2008.

[2] *StratixIII Device Handbook*, 2010. http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf.

[3] An Independent Analysis of Altera's FPGA Floating-point DSP Design Flow, 2011.

[4] *StratixIV Device Handbook*, 2011. http://www.altera.com/literature/hb/stratix-iv/stx4_5v1.pdf.

[5] DSP Builder Advanced Blockset, 2017. https://www.altera.com/products/design-software/model---simulation/dsp-builder/overview.html.

[6] S. Banescu, F. de Dinechin, B. Pasca, and R. Tudoran. Multipliers for floating-point double precision and beyond on FPGAs. In *International Workshop on Higly-Efficient Accelerators and Reconfigurable Technologies (HEART)*. ACM, jun 2010.

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

[8] J. Detrey and F. de Dinechin. Table-based polynomials for fast hardware function evaluation. In *Application-Specific Systems, Architectures, and Processors (ASAP'05)*, pages 328–333, Samos, Greece, July 2005. IEEE Computer Society.

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

[10] J. Detrey and F. de Dinechin. Floating-point trigonometric functions for FPGAs. In K. Bertels, W. Najjar, A. van Genderen, and S. Vassiliadis, editors, *17th International Conference on Field Programmable Logic and Applications (FPL'07)*, pages 29–34, Amsterdam, Netherlands, Aug. 2007. IEEE.

[11] 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*, pages 1 –4, sept. 2006.

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

[13] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhauser Boston, 2010.

[14] 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, Aug. 2012. IEEE.

[15] M. H. Payne and R. N. Hanek. Radian reduction for trigonometric functions. *ACM SIGNUM Newsletter*, 18(1):19–24, Jan. 1983.

[16] Y. Shang. Implementation of ip core of fast sine and cosine operation through FPGA. *Energy Procedia*, 16, Part B(0):1253 – 1258, 2012. 2012 ICFEEM.