A fast and low-complexity operator for the computation of the arctangent of a complex number

Vicente Torres[1], Javier Valls[2]

[1,2]Departamento de Ingenieria Electronica, Universitat Politecnica de Valencia, Valencia, Spain

Abstract

The computation of the arctangent of a complex number, *i.e.* the $atan2$ function, is frequently needed in hardware systems that could profit from an optimized operator. In the present work we present a novel method to compute the $atan2$ function and a hardware architecture for its implementation. The method is based on a First Stage that performs a coarse approximation of the $atan2$ function and a Second Stage that improves the output accuracy by means of a look-up table. We present results for fixed-point implementations in an FPGA device, all of them guaranteeing last-bit-accuracy, which provide an advantage in latency, speed and use of resources, when compared with well-established fixed-point options.

I. Introduction

THE computation of the arctangent function $atan2(a,b)$ (see Fig. 1), *i.e.* obtaining the angle of a complex number $c{=}b{+}ja$, has been the subject of extensive study because this computation is required in many applications.

In hardware approximations for the $atan2(a,b)$ there is often a trade-off between the use of resources and the computation speed and/or latency. For example, the fastest option for the computation of any function may always be the direct implementation with a look-up-table (LUT), but, since the $atan2(a,b)$ is a function of two input variables, in such a case, if the precision of the input data increases by one bit, the amount of memory needed increases by a factor of four. On the other hand, iterative algorithms such as the COordinated Rotation DIgital Computer (CORDIC) can be implemented with a minimal use of resources [1], but at the cost of a low processing speed. If more parallelism is

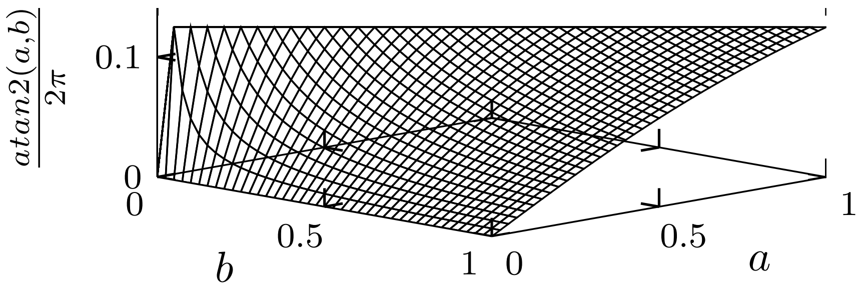

Fig. 1. 3D plot of $atan2(a,b)/2\pi$ for the first octant.

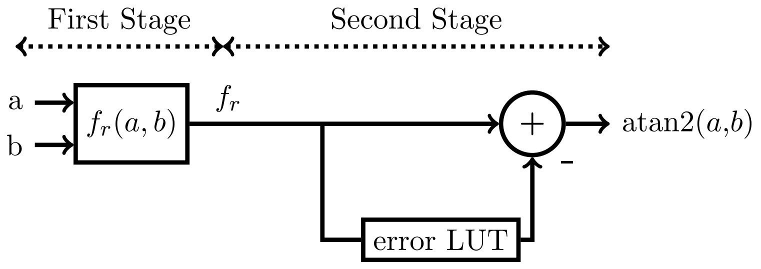

Fig. 2. Simplified scheme of the proposed approximation for $atan2(a,b)/2\pi$

introduced in the implementation, much higher throughputs can be achieved, but the use of resources and the computational delay are increased.

Smaller LUTs can be achieved by using the recip-mult-atan method (RMAM) [2], [3]: first, $z{=}a/b$ is calculated computing $1/b$ with a LUT (avoiding thus the large LUT needed for a two-variable function) and multiplying the result by a, and finally, the one-variable function $atan(z)$ is computed using another LUT.

Another option is to use high-order algebraic polynomials, like Chebyshev polynomials or Taylor series [1]. These methods offer good precision, but since the arctangent is highly non-linear they lead to long polynomials and intensive computations. In other cases, approximations based on rational functions are used [4], [5], [6], as they may provide good enough results with a few elementary operations. As a general rule, in this kind of approximations the division operation is the main contributor to their computational cost, but in addition to that division they usually require one or more multiplication operations.

The architecture we propose is essentially a two-stage method, as shown in Fig. 2. The First Stage uses a low-complexity coarse approximation for the two-input $atan2(a,b)/2\pi$. The Second Stage improves the accuracy by means of a small LUT that stores precomputed error values, as a function of the output of the First Stage. This table does not depend on the two inputs a and b of the $atan2$ operator and is, therefore, comparatively much smaller. As will be shown, the resulting operator is small and can compute the arctangent faster than other popular options, for the same output accuracy.

The organization of this paper is as follows: in Section II we present the algorithm used for the $atan2$ approximation. Section III details the error analysis. The proposed hardware architecture and relevant implementation details are discussed in Section IV. In Section V we present implementation results in an FPGA and we compare our results with those from other $atan2$ operators that use the same normalization.

II. The approximation for the atan2 function

A. *First Stage: coarse approximation of atan2(a,b)*$/2\pi$

The proposed approximation of $atan2(a,b)/2\pi$ of a complex number $c{=}b{+}ja{=}\left|c\right|e^{j\theta}$ is performed in two stages. The First Stage computes a coarse approximation for $atan2(a,b)/2\pi$ using a first-order Lagrange interpolation (see [7]). In the range $[-\pi/4, \pi/4]$ this approximation is:

$$\frac{atan2(a,b)}{2\pi} = \frac{atan(a/b)}{2\pi} \approx \frac{a}{8b}. \qquad (1)$$

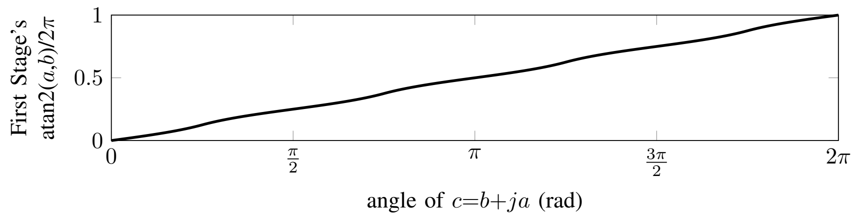

Fig. 3. Coarse approximation for $atan2(a,b)$ computed with eq. 2–5.

Using basic trigonometrical identities (see [8]), this approximation can be extended to the full $[0, 2\pi)$ range:

$$atan2(a,b)/2\pi = 0/4 + atan(a/b)/2\pi \approx$$
$$\approx \left(0+\frac{a}{2b}\right)/4 \quad \text{if } \theta \in \left[\frac{7\pi}{4}, \frac{\pi}{4}\right), \tag{2}$$
$$atan2(a,b)/2\pi = 1/4 - atan(b/a)/2\pi \approx$$
$$\approx \left(1-\frac{b}{2a}\right)/4 \quad \text{if } \theta \in \left[\frac{\pi}{4}, \frac{3\pi}{4}\right), \tag{3}$$
$$atan2(a,b)/2\pi = 2/4 + atan(a/b)/2\pi \approx$$
$$\approx \left(2+\frac{a}{2b}\right)/4 \quad \text{if } \theta \in \left[\frac{3\pi}{4}, \frac{5\pi}{4}\right), \tag{4}$$
$$atan2(a,b)/2\pi = 3/4 - atan(b/a)/2\pi \approx$$
$$\approx \left(3-\frac{b}{2a}\right)/4 \quad \text{if } \theta \in \left[\frac{5\pi}{4}, \frac{7\pi}{4}\right). \tag{5}$$

According to eq. 2–5, $atan2(a,b)/2\pi$ is approximated as (*offset*+f_r)/4, where *offset* is 0, 1, 2 or 3 and f_r is either $a/2b$ or $-b/2a$, depending of the angle range. In Table I we summarize the values of *offset* and f_r for the 4 different angle ranges used in Eq. 2–5. As it is made explicit in Table I, the angle range can be identified from the signs of $a+b$ and $a-b$. This coarse approximation is shown for the full $[0, 2\pi)$ angle range in Fig. 3.

TABLE I
f_r AND *offset* FOR THE FOUR POSSIBLE QUADRANTS

θ range	$a+b>0$	$a-b>0$	*offset*	f_r
$[7\pi/4, \pi/4)$	1	0	0	$a/2b$
$[\pi/4, 3\pi/4)$	1	1	1	$-b/2a$
$[3\pi/4, 5\pi/4)$	0	1	2	$a/2b$
$[5\pi/4, 7\pi/4)$	0	0	3	$-b/2a$

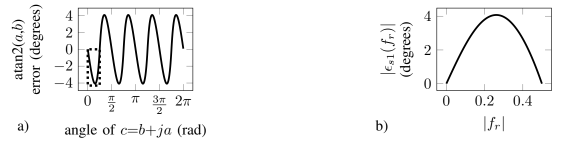

Fig. 4. a) Error in the approximation of the arctangent in degrees after the First Stage and b) contents of the error LUT used in the Second Stage.

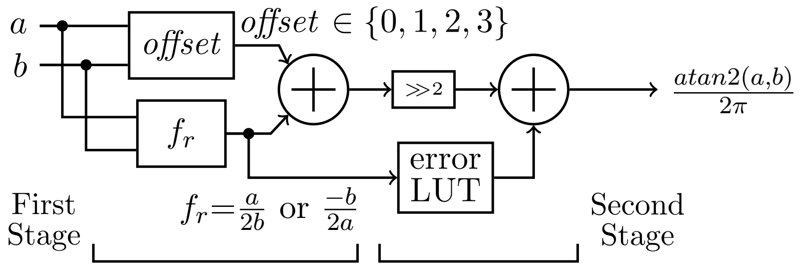

Fig. 5. Basic blocks for the proposed approximation for $atan2(a,b)/2\pi$.

B. Second Stage: error reduction step for the First Stage

The First Stage described in Section II-A approximates $atan2(a,b)$, normalized to the range $[0,1)$, as follows:

$$atan2(a,b)/2\pi \approx (\mathit{offset} + f_r)/4 \ mod \ 1. \tag{6}$$

The error in that approximation is shown in Fig. 4a as a function of the angle of the complex number $c{=}b{+}ja$. If this error function were stored in a LUT it would require a large amount of storage resources, since $atan2(a,b)$ is a two-variable function, and the LUT would have to be addressed by both variables. However, this error can be transformed into a new error function that only depends on f_r, the coarse approximation calculated in the First Stage. This can be easily seen if, without loss of generality, we express this error for the first octant:

$$\begin{aligned}\epsilon_{s1}(f_r) &= ((\mathit{offset} + f_r)/4 \ mod \ 1) - atan2(a,b)/2\pi \\ &= ((\mathit{offset} + f_r)/4 \ mod \ 1) - atan(2f_r)/2\pi. \end{aligned} \tag{7}$$

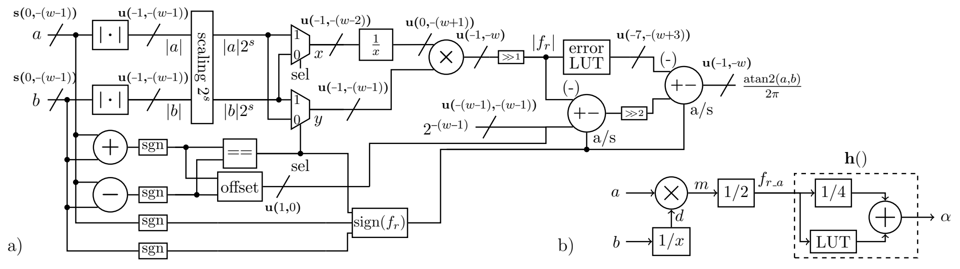

Fig. 6. a) Implementation scheme for the proposed approximation for the $atan2(a,b)/2\pi$ function. b) Simplified model for error analysis.

Therefore, in our proposal we add a Second Stage that improves the accuracy of the First Stage using a tabulated version of $|\epsilon_{s1}(f_r)|$ addressed by $|f_r|$. The contents of that LUT is shown in Fig. 4b. Since this error is periodic, only 1/8 of the values need to be stored. The schematic diagram of the whole system is shown in Fig. 5, which is analogous to the Kmetz/Maenner method of improving Mitchell's method [9].

III. Error analysis

Since implementation results strongly depend on the target accuracy, to ensure a fair comparison of our results with those from other authors, all the implementation results we give in the present work guarantee the same accuracy objective: for the computation of $atan2(a,b)/2\pi$, where a and b are signed values represented with w bits, the output has also w bits with last-bit accuracy (LBA): the absolute value of the difference between the output and the $atan2$ computed with infinite accuracy is lower than the weight of the least significant bit of the output. Since we normalize the output to the $[0,1)$ range, corresponding to $[0,2\pi)$ rad, LBA means that the absolute value of the error is lower than 2^{-w}.

Fig. 6b shows the scheme used to analyze the error. According to this figure, the total error is:

$$\begin{aligned}\epsilon_{total} &= \alpha - atan2(a,b)/2\pi = \alpha - h(f_{r\_a}) + \\ &+ h(f_{r\_a}) - atan2(a,b)/2\pi = \epsilon_h + \epsilon_c, \end{aligned} \tag{8}$$

where α is the actual output (with errors) of the operator, $h()$ is the ideal function performed by the blocks inside of the dashed box of Fig. 6b and ϵ_h and ϵ_c are the contributions to the total error from the operators in $h()$ and from the errors at the input of $h()$, respectively:

$$\epsilon_h \equiv \alpha - h(f_{r\_a}), \tag{9}$$

$$\epsilon_c \equiv h(f_{r\_a}) - atan2(a,b)/2\pi = h(f_{r\_a}) - h(f_r). \tag{10}$$

$h'(f_r)$ can be approximated around f_r, in order to obtain a bound for ϵ_c:

$$h'(f_r) \approx \frac{h(f_r+\epsilon_f) - h(f_r)}{\epsilon_f} = \frac{\epsilon_c}{\epsilon_f}, \tag{11}$$

where $\epsilon_f \equiv f_{r\_a} - f_r$. From eq. 11 we get $\epsilon_c \approx h'(f_r)\epsilon_f$. We obtained empirically that 0.32 is the maximum value for $|h'(f_r)|$, which we round to 1/3 to account for higher order error terms, so:

$$|\epsilon_c| \le |\epsilon_f|/3. \tag{12}$$

On the other hand, the error in m, the output of the multiplier, can be expressed as:

$$\begin{aligned}\epsilon_m &= m - a/b = m - ad + ad - a/b = \\ &= m - ad + a(d - 1/b) = \epsilon_{mult} + a\epsilon_{recip}. \end{aligned} \tag{13}$$

where $\epsilon_{mult} \equiv m - ad$ and $\epsilon_{recip} \equiv d - 1/b$ are the errors introduced by the multiplier and by the reciprocal table, respectively. Since $|a|<1$ and $\epsilon_f = \epsilon_m/2$,

$$|\epsilon_f| < \left(|\epsilon_{mult}| + |\epsilon_{recip}|\right)/2. \tag{14}$$

TABLE II
PROPOSED $atan2(a,b)/2\pi$ IMPLEMENTATION RESULTS

w=8 bits			w=12 bits			w=12 bits			w=16 bits		
LUTs+reg	BS+DSP	cyc@f	LUTs+reg	BS+DSP	cyc@f	LUTs+reg	BS+DSP	cyc@f	LUTs+reg	BS+DSP	cyc@f
127+0	0+0	1@138	387+0	0+0	1@93	414+93	0+0	5@239	891+0	0+0	1@65
93+0	0+1	1@119	243+0	0+1	1@89	237+41	0+1	5@235	697+0	0+1	1@69
101+0	0+1	1@135	362+42	0+0	2@160	384+95	1+0	5@254	896+54	0+0	2@121
132+24	0+0	2@230	243+16	0+1	2@157	214+49	1+1	5@238	685+20	0+1	2@131
91+12	0+1	2@203	360+27	1+0	2@106	430+150	0+0	7@261	893+81	0+0	3@173
131+38	0+0	3@341	187+16	1+1	2@106	227+103	0+1	7@385	537+70	1+1	3@146
87+29	0+1	3@268	371+64	0+0	3@212	396+112	1+0	7@251	261+45	2+1	4@228
132+57	0+0	4@361	234+64	0+1	3@195	214+86	1+1	7@327	261+78	2+1	5@250
91+41	0+1	4@277	371+44	1+0	3@155	234+118	0+1	8@440	240+92	2+1	7@273
91+57	0+1	5@356	198+20	1+1	3@185	216+86	1+1	8@350	250+128	2+1	8@312
89+93	0+1	7@369	361+84	0+0	4@251	230+147	0+1	9@549	244+174	2+1	10@367
91+90	0+1	8@589	234+45	0+1	4@209	215+113	1+1	9@421	267+185	2+1	12@399
			388+70	1+0	4@247	216+139	1+1	10@511	244+262	2+1	13@504
			204+20	1+1	4@217	216+154	1+1	11@561	285+295	2+1	14@556

And the total error is bounded by:

$$|\epsilon_{total}| < |\epsilon_h| + (|\epsilon_{mult}| + |\epsilon_{recip}|)/6. \tag{15}$$

The output α is rounded to w bits, adding an error whose absolute value could be as big as $2^{-(w+1)}$. Since LBA means that the total error is lower than 2^{-w}, in the worst case it should be satisfied:

$$|\epsilon_{total}| < 2^{-(w+1)}. \tag{16}$$

IV. HARDWARE ARCHITECTURE

Fig. 6a shows the scheme of the proposed hardware architecture. It works with the absolute values of the inputs a and b and their signs are used later in the scheme. The "sel" signal selects the operands for the division operation according to the signs of $a+b$ and $a-b$, as detailed in Table I. The division required for the computation of $|f_r|$ is implemented with a table that stores $1/x$ and a multiplier that completes the division. A scaling stage is added in order to reduce the size of the reciprocal table. The most relevant implementation details are commented upon in the following subsections. Specifically, we give details for three accuracies: $w=\{8,12,16\}$ bits.

A. Datapath dimensioning

Fig. 6a shows the binary formats used in different buses of the system, where $s(q,t)$ and $u(q,t)$ denote signed and unsigned fixed-point formats, respectively. 2^q is the weight of the most significant bit (MSB) and 2^t is the weight of the least significant bit (LSB).

If we consider a simple case where the only errors present in both LUTs are from the rounding of the stored values, the worst case errors would be $\epsilon_{recip} \leq 2^{-(w+2)}/6$, $\epsilon_h = 2^{-(w+4)}$ and $\epsilon_{mult}=0$. Note that truncating the output of the multiplier doesn't introduce an additional error unless the truncated bits are used in the following processing steps. It can be easily checked that under these simplified conditions, eq. 16 is satisfied even for the upper bound defined by eq. 15. In a more realistic scenario, LBA can still be achieved with smaller LUTs that either don't use all the available bits as their address word or that are implemented as bipartite tables [10]. Although in these cases the tables introduce bigger errors (as explained below), LBA could still be achieved, since eq. 15–16 represent an upper bound that could not be reached. For this reason, we performed exhaustive tests for different LUT sizes looking for optimized implementations. Fig. 7 is an example of the error pattern obtained in one of the w=12 operators.

B. Range reduction

Obtaining $|f_r|$ requires the computation of y/x (see Fig. 6a), which involves the computation using a LUT of $1/x$. Since $1/x$ can be extremely large for small values of x, a scaling operation is performed on $|a|$ and $|b|$. This block detects the leading-zeros in both $|a|$ and $|b|$, scaling both by the same factor (2^s) so the MSB of x, the biggest one of both outputs, is always 1. Details of this block are found in [2], [3]. Thanks to this block x is always in the $[0.5, 1)$ range and the biggest possible value of $1/x$ is 2.

C. Computation of the reciprocal

For the computation of $1/x$ two different strategies, both table-based, are used: direct tabulation for the w=8 bit operator and bipartite tables for w=$\{12, 16\}$ bits. In both cases all the bits from the input word are used. Therefore, for the direct tabulation the only errors are those created by rounding the words stored in the table, and for the bipartite tables the maximum absolute value of the error can be estimated from the second derivative of the stored funcion and also from the rounding errors [11]. The value stored in the first address of this table should be 2, but $2{-}2^{-n+1}$ is stored for a table with n-bit words, so the MSB of all the stored words is the same and, therefore, it doesn't need to be stored.

The size of this table is 64×6 for w=8, 128×14+128×7 for w=12 and $1k$×18+512×8 for w=16 bits.

D. The error LUT

The error LUT stores the values for $|\epsilon_{s1}(f_r)|$. Since only the absolute value is stored, the proper sign is applied later taking advantage of an adder-subtractor. As is the case for the $1/x$ table, two different strategies are used: direct tabulation for the w=$\{8, 12\}$ bit operators and bipartite tables for w=16. Irrespectively of the option selected, not all the available input bits are used to address the table and this fact introduces an additional error term related to the first derivative of the stored function [12]. Since the address input of the table is created truncating the input word, this table is always filled using values halfway in the segments [12].

The size of this table is 64×5 for w=8, 256×9 for w=12 and 512×13+512×7 for w=16.

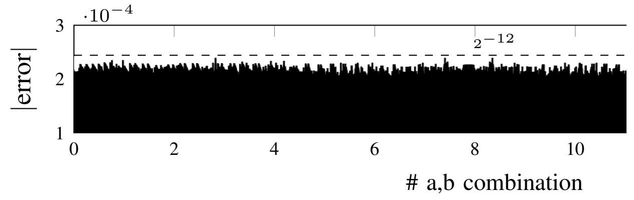

Fig. 7. Absolute error for one of our w=12 $atan2(a, b)/2\pi$ operators

TABLE III
CORDIC $atan2(a,b)/2\pi$ IMPLEMENTATION RESULTS

w=8 bits		w=12 bits		w=16 bits	
LUTs+reg	cyc@f	LUTs+reg	cyc@f	LUTs+reg	cyc@f
197+0	1@109	471+0	1@68	794+0	1@46
199+43	2@190	471+66	2@127	801+86	2@91
203+65	3@286	471+111	3@174	803+152	3@117
209+98	4@355	475+158	4@217	803+209	4@163
210+225	9@537	473+243	6@302	799+443	8@333
		501+512	13@482	836+919	17@567

TABLE IV
RECIP-MULT-ATAN DEG-1 FROM FLOPOCO IMPLEMENTATION RESULTS

w=8 bits			w=16 bits		
LUTs+reg	BS+DSP	cyc@f	LUTs+reg	BS+DSP	cyc@f
209+0	0+0	1@79	1042+0	0+0	1@51
155+0	0+1	1@77	755+0	0+1	1@51
216+9	0+0	2@86	1064+36	0+0	2@73
156+9	0+1	2@84	794+36	0+1	2@77
202+41	0+0	3@182	1007+81	0+0	3@108
143+25	0+1	3@171	745+49	0+1	3@130
204+54	0+0	4@196	729+79	0+1	4@130

E. The multiplier

The size of this multiplier is 6×7, 15×11 and 19×15 for w=$\{8,12,16\}$ bits, respectively. Although the natural implementation option for w=$\{12,16\}$ bit operators is to use a DSP48 block, for comparison purposes we implemented logic-only versions of the required multipliers. Those multipliers were truncated by computing only the most significant bits of the product and an offset value was added to round the output and compensate for the truncation. We did not explore the possibility of using an internal pipeline in those multipliers.

F. The final rounding

To achieve LBA, *i.e* an absolute error smaller than 2^{-w} for the selected output format, the output value has necessarily to be rounded. In the worst case the absolute value of the rounding error is 2^{-w-1}, and this error added to the combination of all the other error terms should be lower than 2^{-w} to guarantee our accuracy goal. In order to avoid the use of a final rounding step, half LSB (2^{-w-1}) is added to the offset value (see Fig. 6a) and the output of the $atan2$ operator is simply truncated to w fractional bits.

V. IMPLEMENTATION RESULTS

Our proposal was modelled using VHDL language and was implemented for w=$\{8,12,16\}$ bits in a Kintex7 xc7k325tffg900-2 FPGA device from Xilinx. Table II summarizes the implementation results. As can be seen, we report results for the case when only inputs and output are registered (1 cycle) and also for several degrees of pipelining. It should be noted that when embedded dual-port RAMs, called Block-Select RAMs (BSRAM), are used in the designs, the lowest latency that can be achieved by the operator depends on the amount of BSRAMs, since they are synchronous memories that add at least one latency cycle. We also report results for different implementation options for the required tables and multipliers. When those operators are not implemented with embedded BSRAM/DSP48 blocks, they are implemented with the LUTs available in the target device. Abbreviations used in Tables II–IV are: LUTs: 6-input look-up-tables, reg: pipelining registers, BS: Block Select RAM, DSP: DSP48, cyc: total latency in clock cycles, f: maximum clock frequency (MHz).

Next, we analyze the best implementation results from other authors. Gutierrez et al. [2], [13] reported implementation results for LUT-based approximations with 12-bit inputs, but their results are not comparable with ours, since their achieved accuracy is only around 10 bits. Dinechin and Istoan [3] have recently reported the implementation results for different last-bit-accurate fixed-point $atan2(a, b)$ options, including the method proposed in [2]. Their results suggest that CORDIC should be the main reference for performance comparisons.

For comparison purposes we implemented in the same device an $atan2$ operator based on an unrolled CORDIC architecture designed to guarantee LBA. It was optimized by using the minimum amount of guard bits, by stopping updating the x_i path in the last stages, and also by progressively cutting down the width of the y_i path (see [3]). The initial z_i value used was set to half output LSB (2^{-w-1} for the same normalization criterion as in our proposal) with the purpose that a truncation of the output performs a rounding operation. Table III shows the implementation results. When comparing the computational speed from our operator (see Table II) with that of CORDIC's, as a general conclusion, our proposal can run faster than CORDIC for the same latency. For all the three input widths we reach the fastest clock frequency supported by the FPGA device with smaller latency than CORDIC.

Although the use of resources can not be easily compared with CORDIC's because our proposal requires a different kind of resources (BSRAM and DSP48 blocks), we also implemented only-logic versions of our operator so that comparisons could be made. The results show that our $w=\{8, 12\}$ bit only-logic implementations need fewer LUTs and registers than CORDIC. For example, for $w=8$ bits and 2 cycles of latency our proposal is 21% faster while requiring 33% and 44% fewer LUTs and registers, respectively.

We have also implemented in our target device the RMAM degree 1 from [5], using the FloPoCo framework provided by the authors. Their RMAM operators also produce a normalized output. The results are shown in Table IV. It should be noted that when we tested the operators provided by the FloPoCo tool (using its exhaustive TestBench) for $w = \{8, 9, 10, 11\}$ bits, they were not 100% LBA-compliant, but only above 99.9%. These results confirm [5]'s conclusion that CORDIC is generally faster while using fewer resources than RMAM. When Tables II and IV are compared, it is seen that our proposal outperforms RMAM both in resources and speed. This result is as expected, since our error LUT stores values that are smaller and have a lower first derivative than the values stored in the RMAM. A lower first derivative means that the same error caused by truncating the addressing word can be achieved with fewer stored words.

VI. Conclusions

In this article we propose a novel method and a hardware architecture to approximate the arctangent of a complex number. We report implementation results in an FPGA device for 8, 12 and 16 bits of accuracy. Thanks to its simplicity, the proposed method has demonstrated a reduced use of resources and also good speed when compared with other approximations of akin accuracies. This operator may have practical application for systems that require either a simple approximation with minimum use of resources or improved latency and speed.

Acknowledgments

This work is funded by the Spanish Ministerio de Economía y Competitividad and FEDER under the grant TEC2015-70858-C2-2-R.

References

[1] J.-M. Muller, *Elementary functions: algorithms and implementation*. Cambridge, MA, USA; Berlin, Germany; Basel, Switzerland: Birkhäuser, 1997.

[2] R. Gutierrez and J. Valls, "Low-power FPGA-implementation of atan(y/x) using look-up table methods for communication applications," *Journal of Signal Processing Systems*, vol. 56, no. 1, pp. 25–33, 2009.

[3] F. de Dinechin and M. Istoan, "Hardware implementations of fixed-point atan2," in *22nd Symposium on Computer Arithmetic*, 2015, pp. 34–40.

[4] R. G. Lyons, "Another contender in the arctangent race," *IEEE Signal Processing Magazine*, vol. 21, no. 1, pp. 109–110, January 2004.

[5] R. I. S. Rajan, S. Wang and A. Joyal, "Efficient approximations for the arctangent function," *IEEE Signal Processing Magazine*, vol. 23, no. 3, pp. 108 – 111, May 2006.

[6] C. J. X. Girones and D. Puig, "Full quadrant approximations for the arctangent function," *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 130–135, January 2013.

[7] J. M. Shima, "FM demodulation using a digital radio and digital signal processing," Master's thesis, University of Florida, Gainesville, 1995.

[8] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, 10th ed. New York: Dover, 1964.

[9] M. Arnold, T. Bailey, and J. Cowles, "Error analysis of the kmetz/maenner algorithm," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 33, no. 1, pp. 37–53, 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1021189701352

[10] D. D. Sarma and D. W. Matula, "Faithful bipartite rom reciprocal tables," in *Computer Arithmetic, 1995., Proceedings of the 12th Symposium on*, Jul 1995, pp. 17–28.

[11] M. J. Schulte and J. E. Stine, "Symmetric bipartite tables for accurate function approximation," in *Computer Arithmetic, 1997. Proceedings., 13th IEEE Symposium on*, Jul 1997, pp. 175–183.

[12] M. J. Schulte and J. E. Stine, "Approximating elementary functions with symmetric bipartite tables," *IEEE Transactions on Computers*, vol. 48, no. 842–847, 1999.

[13] R. Gutierrez, V. Torres, and J. Valls, "Fpga-implementation of atan(y/x) based on logarithmic transformation and lut-based techniques," *J. Syst. Archit.*, vol. 56, no. 11, pp. 588–596, Nov. 2010.