Hardware implementations of fixed-point Atan2

Florent de Dinechin, Matei Istoan
Université de Lyon, INRIA,
INSA-Lyon, CITI, F-69621 Villeurbanne, France
florent.de-dinechin@insa-lyon.fr matei.istoan@inria.fr

***Abstract*—The atan2 function computes the polar angle arctan(y/x) of a point given by its cartesian coordinates. It is widely used in digital signal processing to recover the phase of a signal. This article studies for this context the implementation of atan2 with fixed-point inputs and outputs. It compares the prevalent CORDIC shift-and-add algorithm to two multiplier-based techniques. The first one computes the bivariate atan2 function as the composition of two univariate functions: the reciprocal, and the arctangent, each evaluated using bipartite or polynomial approximation methods. The second technique directly uses piecewise bivariate polynomial approximations of degree 1 or 2. Each of these approaches requires a relevant argument reduction, which is also discussed. All the algorithms are last-bit accurate, and implemented with similar care in the open-source FloPoCo framework. Based on synthesis results on FPGAs, their relevance domains are discussed.**

***Index Terms*—arctangent, atan2, hardware, FPGA arithmetic**

I. Introduction

A. Definitions and notations

In all this article we implement the function $\operatorname{atan2}(y,x) = \arctan(y/x)$. This function (part of the standard mathematical library) returns an angle in $[-\pi,\pi]$. Compared to a plain composition of division and the arctan function (that returns an angle in $[-\pi/2,\pi/2]$), $\operatorname{atan2}(y,x)$ keeps track of the respective signs of x and y. It is used to compute the phase of a complex number $x+iy$.

We are interested in implementations of this function for fixed-point input and outputs, each on w bits. As $\arctan(\frac{ky}{kx}) = \arctan(\frac{y}{x})$, the range of the inputs is not really relevant, as long as both x and y are in the same format. We choose to have both inputs as fixed-point numbers in the range $[-1,1)$, represented classically in two's complement.

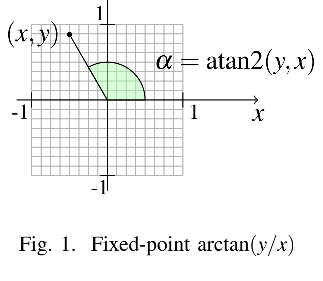

Fig. 1. Fixed-point $\arctan(y/x)$

There are two sensible choices for the output.

- It may be expressed in radian, from $-\pi$ to π;
- It may be expressed in $[-1,1)$, in which case the function being computed is $\frac{1}{\pi}\operatorname{atan2}(y,x)$. We will refer to this option as "binary angles".

There are several reasons for preferring binary angles in a fixed-point context. The first is that it fully uses the representation space. An output on $[-\pi,\pi]$ will be coded on a fixed-point format that may represent the interval $[-4,4)$. Therefore, all the codes between π and 4 will never be used.

The second is that it slightly simplifies implementation: the modulo-2π periodicity of radian angles becomes a modulo-2 periodicity that comes for free in two's complement binary arithmetic. For instance, the fact that the two's complement output range $[-1,1)$ is slightly asymmetrical is not a problem, as modulo-2 arithmetic will map angle $1\times\pi$ to angle $-1\times\pi$ as it should. This will save some computations in the range reduction, and allow it to be exact, where radian angles require computations with the constant $\pi/2$ which cannot be represented exactly in binary.

The third reason for preferring binary angles is that it generally simplifies the application context as well. For instance, in QPSK decoding, the atan2 of incoming samples are averaged to estimate a phase shift. Computing this average is simpler with binary angles than with radian angles, again because it is a modulo operation.

For all these reasons, the focus of this article is the function (sometimes called atan2Pi)

$$f(x,y) = \frac{1}{\pi}\operatorname{atan2}(y,x) \quad .$$

However, all the algorithms in this paper may be straightforwardly adapted to radian angles.

B. Overview of hardware implementation methods

This paper compares several hardware implementation techniques, some classical, some new. To ensure a meaningful comparison, each technique is implemented with the same care, and with the same accuracy objective: $f(x,y)$ is computed with last-bit accuracy (LBA), *i.e.* the error (the difference between the returned result and the infinitely accurate one) is less than the weight of the least significant bit (LSB) of the result. This is sometimes called *faithful rounding* in the literature. LBA takes full advantage of the output format, but is slightly less accurate than correct rounding. However, it can be obtained at a much smaller cost: correct rounding may require twice the internal precision [1], which is not justified in a hardware context. Last-bit accurate architectures are also almost bit-for-bit compatible with each other (differing by at most one bit in the last place) which enables a fair comparison of methods.

The most classical technique for fixed-point hardware implementation of atan2 is CORDIC. It performs a series of micro-rotations of the point (x,y) to send it on the x axis. The angle of each micro-rotation is chosen so that it can be implemented by shift and add. CORDIC is studied in Section III, which contributes a fine error and range analysis that guarantees last-bit accuracy at the smallest possible datapath width.

One could consider tabulating all the values of atan2 in a table addressed by a concatenation of x and y. However, this table would have 2^{2w} entries, which is impractical even for small values of w. Besides, classical table-compression techniques (bipartite, etc) do not apply here, since they address functions of one variable, whereas $f(x,y)$ is a function of two variables.

The next technique, studied in Section IV and illustrated in Figure 5, therefore reduces atan2 to two functions of one variable: the reciprocal $1/x$, used to compute the division $z = y/x$, and the arctangent, used to compute $\arctan(z)$. There is a wide body of techniques that can be used to compute functions of one variable. However, we now have the problem that z may become very large when x comes near to zero. This would require either a floating-point format, or a very large fixed-point format for z. This can be avoided by a scaling-based range reduction that will be detailed in Section II.

The last and most original technique, studied in Section V, is the use of a piecewise approximation by bivariate polynomials. We have limited experiments to polynomials of first and second degree because the number of multipliers in bivariate polynomials explodes with the degree. Thus, $\text{atan2}(y,x)$ will be approximated by $P_1(y,x) = ax+by+c$, respectively $P_2 = ax+by+c+dx^2+ey^2+fxy$. The corresponding architectures are depicted by Figures 7 and 8.

It is interesting to compare rough estimations of the asymptotic complexity of each approach with respect to the precision. It is well known that CORDIC area and delay are quadratic in the precision. The other methods are table-based and will therefore be exponential in area. However their delay should be sub-quadratic, with the bivariate approach having shorter latency, but larger tables. The main objective of this article is therefore to study the relevance domain of each method in Section VI.

This work essentially focuses on FPGAs. An unexpected result is that, even on modern FPGAs enhanced with DSP blocks and memories, CORDIC is a clear winner. This will be analyzed in SectionVI. However, this work also contributes an open-source VHDL generator that covers all the presented methods, so that comparisons can be performed in other contexts.

$$\begin{cases} x < 0 \\ y < 0 \\ y < x \end{cases}$$
$$\Rightarrow \text{atan2}(y,x) = -\frac{\pi}{2} - \text{atan2}(|x|,|y|)$$

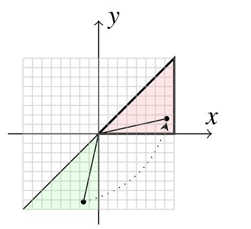

Fig. 2. One case of symmetry-based range reduction. The 7 other cases are similar.

C. Notations for fixed-point formats

In all the article, we will describe a signed fixed-point format by $\text{sfix}(m,l)$ and an unsigned fixed-point format by $\text{ufix}(m,l)$. The two integers m and l denote respectively the weights of the most significant bit (MSB) and LSB of the format. For instance our inputs and outputs in $[-1,1)$ on w bits will be on the format $\text{sfix}(0,-w+1)$: the sign bit has weight 2^0 and the LSB has weight 2^{-w+1}.

II. Range reductions

A. Using parity and symmetries

As arctan is an odd function, inputs may be straightforwardly reduced to the positive quadrant. Besides, there is a symmetry between x and y:

$$\arctan(y/x) = \pi/2 - \arctan(x/y).$$

If $y > x$, we may therefore swap x and y, so the computation is reduced to the first octant ($x \geq 0,\ y \geq 0,\ y < x$).

Similar range reduction formulae may be used on other quadrants, one example being given by Figure 2.

Implementing such argument reduction formulae is much easier with binary angles. Indeed, the final addition of $\pi/2$ becomes an addition of $1/2$ to a number in $[-1,1)$ (an exact operation with only a one-bit carry propagation). Conversely, adding $\pi/2$ would require a full-width carry propagation, while entailing a systematic error due to the rounding of the irrational $\pi/2$.

The absolute values $|x|$ and $|y|$ require negating negative input values (the value -1, which has no opposite in two's complement arithmetic, must be saturated). This may be implemented by a full-size subtraction if we want them to be exact, or by bitwise complement (which is smaller and has lower latency) at the cost of an error on the last bit. The two options will be relevant, depending on the subsequent algorithm.

B. Scaling range reduction

Due to the division by x, the value of $z = y/x$ can become very large. On the plot of atan2 over a quadrant (Figure 3, left), this translates to much larger slopes when y is small.

For an approach that explicits the computation of z, or for a bivariate polynomials approximation, it is therefore interesting to avoid this region. This can be easily performed by the architecture depicted on Figure 4 [2]. This architecture first

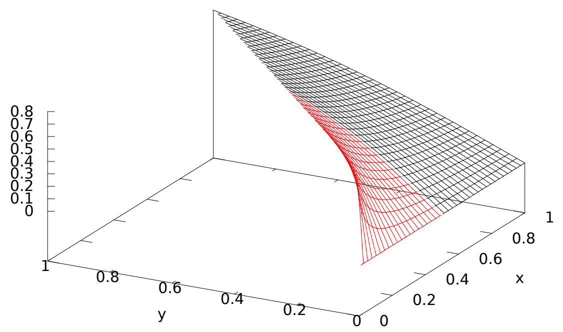

Fig. 3. 3D plot of atan2 over the first octant.

counts how many leading zeroes are common to both x and y. This leading zero count is called s. Then, thanks to the trivial identity

$$\arctan\left(\frac{2^s y}{2^s x}\right) = \arctan\left(\frac{y}{x}\right)$$

both x and y may be scaled up by a shift left of s bits, resulting in $x \geq \frac{1}{2}$.

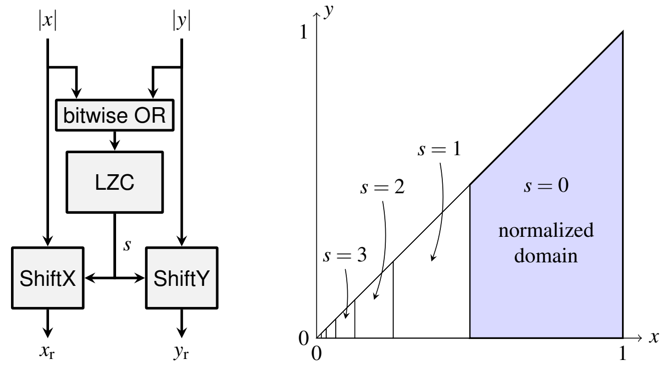

Fig. 4. Scaling range reduction

III. CORDIC

CORDIC computation of the arctangent is classically [3], [1] described as

$$\begin{cases} x_1 &= x \\ y_1 &= y \\ \alpha_0 &= 0 \end{cases} \quad \text{and} \quad \begin{cases} x_{i+1} &= x_i - 2^{-i} s_i y_i \\ y_{i+1} &= y_i + 2^{-i} s_i x_i \\ \alpha_{i+1} &= \alpha_i - s_i \arctan 2^{-i} \end{cases}$$

where $s_i = -\mathrm{sgn}(y_i)$ (vectoring mode). This iteration converges as follows:

$$\begin{cases} x_i & \longrightarrow \quad K\sqrt{x^2+y^2} \\ & \qquad \text{with } K = \prod_{i=1}^{\infty} \sqrt{1+2^{-2i}} \approx 1.1644... \\ y_i & \longrightarrow \quad 0 \\ \alpha_i & \longrightarrow \quad \arctan \frac{y_r}{x_r} \end{cases}$$

It may produce a binary angle if the constants $\arctan 2^{-i}$ are expressed in this format. Note that thanks to the initial argument reduction to the $[0, \pi/4]$ octant, we may start from iteration 1 instead of iteration 0, which leads to a smaller value of K than found in most textbooks .

In FPGA technology, addition carry propagation is very fast compared to generic logic. Therefore we choose to ignore CORDIC variants (reviewed in [1], [4] and [5]) that accelerate the iteration by using carry-save or other forms of redundant arithmetic. This choice will be validated by the results in Section VI.

Also, we are interested in an unrolled implementation that, once pipelined, will produce one result each cycle.

A. Error analysis and datapath sizing

If the iteration is computed with infinite accuracy, the error on α_i after i iterations is smaller than 2^{-i} radian [1]. To obtain an approximation error on the binary angle smaller than 2^{-w} (i.e. smaller than one half-ulp of our result), we may therefore stop after iteration $w-1$.

Let us now discuss rounding errors, and more generally fixed-point implementation issues. Let us call p the weight of the LSB on the x_i and y_i datapath. It is easy to see that $0 \leq x_i < K\sqrt{2} \approx 1.646$. This defines the MSB of x_i: its format will be ufix$(1, -p)$.

We also have $y_i < K \sin \alpha_i$. Since $\alpha_i \leq 2^{-i}$, we have $|y_i| < 2^{-i+1}$. This defines the MSB of y_i: its format will be sfix$(-i+2, -p)$.

Let us now study the error of implementing the CORDIC iteration. Formally, if we note $\tilde{x}_i$ and $\tilde{y}_i$ the computed values we may define $\varepsilon_i^x = \tilde{x}_i - x_i$, and similarly $\varepsilon_i^y = \tilde{y}_i - y_i$. The only source of error in each iteration is the error u_i^x and u_i^y due to the loss of the bits discarded in the shifts. This is written

$$\begin{aligned} \tilde{x}_{i+1} &= \tilde{x}_i - s_i 2^{-i} \tilde{y}_i + u_i^x \\ &= x_i + \varepsilon_i^x - s_i 2^{-i}(y_i + \varepsilon_i^y) + u_i^x \\ &= x_{i+1} + \varepsilon_i^x - s_i 2^{-i} \varepsilon_i^y + u_i^x \end{aligned}$$

Hence the recurrence defining the accumulation of rounding errors:

$$\varepsilon_{i+1}^x = \varepsilon_i^x - s_i 2^{-i} \varepsilon_i^y + u_i^x$$

If we consider a common bound $\overline{\varepsilon}_i$ of ε_i^x and ε_i^y, and the bound 2^{-p} on u_i^x, we get

$$\overline{\varepsilon}_{i+1} = \overline{\varepsilon}_i(1+2^{-i}) + 2^{-p}$$

with $\overline{\varepsilon}_1 = 0$, unless the opposite in the range reduction is computed by simply complementing the bits, in which case $\overline{\varepsilon}_1 = 2^{-p}$. This only marginally adds to the overall error.

To build an accurate architecture, the recurrence $\overline{\varepsilon}_{i+1} = \overline{\varepsilon}_i(1+2^{-i}) + 2^{-p}$ is computed up to $i = w-1$. We then define $p = w - 1 - \lceil log_2 \overline{\varepsilon}_{w-1} \rceil$ as the precision of the x_i and y_i datapaths. It ensures that y_i is computed with enough accuracy to ensure that rounding errors do not change its sign in a way that cannot be corrected by CORDIC.

Finally, considering that $|y_i| < 2^{-i+1}$, the term $2^{-i} y_i$ added or subtracted to x_i is smaller than 2^{-2i+1}. Therefore, we may stop updating x_i as soon as $2i - 1 > p$.

On the α_i datapath, the error analysis is much simpler: each $\arctan 2^{-i}$, correctly rounded to the precision p_α, contributes at most one half-ulp to the error. The final rounding will contribute 2^{-w}. We therefore need $g_\alpha = 1 + \lceil log_2((w-1)\times 0.5)\rceil$ guard bits to absorb all these errors.

Here the only trick is that the final rounding may be performed by truncation, provided 2^{-w} has been added to one of the $\arctan 2^{-i}$ constants.

IV. Reciprocal-mult-arctan

This variant, illustrated by Figure 5 (where x_r and y_r are the reduced inputs), is the most natural but requires the scaling argument reduction of Figure 4. After this scaling, we have $0.5 \le x \le 1$. Since we are on the first octant where $y \le x$, we also have $z = y/x \in [0,1]$. On such intervals, both $1/x$ and $\arctan(z)$ are regular (see Figure 6) and can be fully tabulated, or well approximated by polynomials. The main difficulty will be to define the bitwidths of r and z that will enable last-bit accuracy at the minimal cost.

Here again, the situation is much simpler with binary angles. The output of the arctangent box belongs to $[0,1/4]$ and is needed in the $\text{ufix}(-2,-w+1)$ format. Its two leading bits will be added, error-free, by the reconstruction.

A. Related work

This technique is classically used in software [6], [7]. The division $z = y/x$ is performed in the working precision, then a high-degree odd polynomial is used for an accurate approximation of $\arctan(z)$.

In hardware, early works [8], [9] use piecewise linear approximations of unspecified accuracy for the computation of $\arctan(z)$ to some precision. On FPGAs, this technique is used in [2] with multipartite tables [10] to save multipliers. Comparisons to CORDIC are only given for 12-bit precision (the main claim is a near halving of power consumption). Similarly, the accuracy of the architecture is only measured for 12-bit precision (10.3 bits in the worst case). In a follow-up article [11] the division is transformed to a subtraction in the logarithmic domain. However the architecture now requires three function evaluations (two log conversions on the input, and one $\arctan(2^z)$). A non-uniform decomposition is used for the evaluation of $\arctan(2^z)$.

The present section generalizes [2] to more precisions, and with more rigorous rules of the game (last-bit accuracy) and error analysis. It also uses polynomials of larger degrees, as in [12], to approximate the unary functions when needed.

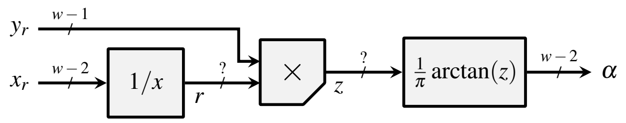

Fig. 5. Architecture based on two functions of one variable

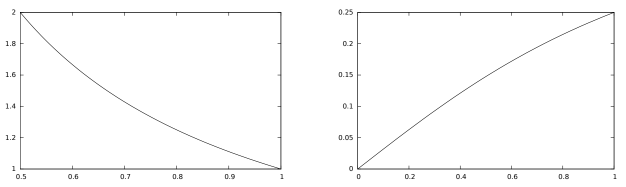

Fig. 6. Plots of $1/x$ on [0.5,1] (left) and $\frac{1}{\pi}\arctan(z)$ on [0,1] (right)

B. Error analysis

Let us define formally the errors of the reciprocal and arctangent boxes:

$$\varepsilon_{\text{recip}} = r - 1/x$$
$$\varepsilon_{\text{atan}} = \alpha - \frac{1}{\pi}\arctan(z)$$

We have to truncate or round the product to keep the number of input bits to the arctangent box small. Therefore, the multiplier will not be exact: let us define its error

$$\varepsilon_{\text{mult}} = z - yr$$

We may now define the total error on z:

$$\begin{aligned}\varepsilon_z &= z - \frac{y}{x}\\ &= z - yr \quad + \quad yr - \frac{y}{x}\\ &= \varepsilon_{\text{mult}} + y\varepsilon_{\text{recip}}\end{aligned}$$

As $0 \le y \le 1$ we will have

$$|\varepsilon_z| \le |\varepsilon_{\text{mult}}| + |\varepsilon_{\text{recip}}|$$

For last-bit accuracy, the total error must remain smaller than $u = 2^{-w+1}$. The total error is defined as

$$\begin{aligned}\varepsilon_{\text{total}} &= \alpha - f(x,y)\\ &= \left(\alpha - \frac{1}{\pi}\arctan(z)\right) + \left(\frac{1}{\pi}\arctan(z) - f(x,y)\right)\\ &= \quad \varepsilon_{\text{atan}} \quad + \quad \varepsilon_2\end{aligned}$$

From $\arctan'(x) = \frac{1}{1+x^2}$ we have

$$\arctan(z+\varepsilon) - \arctan(z) \approx \varepsilon\frac{1}{1+x^2}.$$

From this we deduce that, for $z \in [0,1]$,

$$\left|\frac{1}{\pi}\arctan(z+\varepsilon) - \frac{1}{\pi}\arctan(z)\right| < \frac{1}{3}\varepsilon$$

where the rounding up of $\frac{1}{\pi}$ to $\frac{1}{3}$ accounts for the higher order error terms. The previous inequality may also be observed on Figure 6. Applying it to $z = y/x + \varepsilon_z$, we get:

$$|\varepsilon_2| = \left|\frac{1}{\pi}\arctan(z) - \frac{1}{\pi}\arctan(y/x)\right| < \frac{1}{3}|\varepsilon_z|$$

and finally

$$|\varepsilon_{\text{total}}| < \frac{1}{3}|\varepsilon_{\text{recip}}| + \frac{1}{3}|\varepsilon_{\text{mult}}| + \varepsilon_{\text{atan}} \tag{1}$$

C. Datapath dimensioning

Let us first consider the simpler case when the two functions are purely tabulated. The arctangent table may be correctly rounded to its output format. We therefore have $|\varepsilon_{\text{atan}}| \leq u/2$, and we need $|\varepsilon_{\text{total}}| < u = 2^{-w+1}$ (last-bit accuracy). Unfortunately, we cannot use a correctly-rounded reciprocal table, because we want its output to be in $[1,2)$ and not in $(1,2]$ when the input is in $[\frac{1}{2},1)$. What we tabulate for r is therefore the correct rounding of $1/x - 2^{-w}$ to the format ufix$(1,-w)$. This entails $|\varepsilon_{\text{recip}}| \leq u$.

The multiplier itself does not need to be correctly rounded. A truncated multiplier, faithful to an output format ufix$(0,-w)$ for z will entail $|\varepsilon_{\text{mult}}| \leq u/2$. This combination of errors satisfies (1). Interstingly, however, this error bound is not reached: exhaustive tests (up to $w = 12$) show that LBA is still achieved with a truncated multiplier faithful to ufix$(0,-w+1)$.

A correctly rounded multiplier adds to the cost of the multiplier (by requiring to compute and sum all its partial products), but it provides the same accuracy for z using one less bit. Saving one bit on z will reduce the size of the arctangent box, possibly halving it. This trade-off between the multiplier and the arctangent box has not been fully studied yet.

Using a black-box polynomial approximator [12] or the multipartite method [10] entails that the two functions are themselves last-bit accurate. In this case, we may no longer ensure $|\varepsilon_{\text{atan}}| \leq u/2$ at a reasonable cost, due to the Table Maker's dilemma [1]. What is possible is a more accurate implementation that will ensure $|\varepsilon_{\text{atan}}| \leq 3u/4$. We now have a $u/4$ error budget to distribute among the multiplier and the reciprocal table. With the same reasoning, using a ufix$(1,-w-1)$ format for r and a ufix$(0,-w)$ for z will satisfy (1).

This analysis has been implemented in the FloPoCo generator (`FixAtan2` operator). This program builds an architectures for any size w and degree d. For $d = 1$, the bipartite method is currently used, and the multipartite method should be used in the near future.

D. FPGA-specific considerations

For small precisions, the following optimizations may apply on some FPGAs that embed hard memory and multiplier blocks.

The hard multipliers will compute the full product, so truncation will be faithful, and correct rounding will come at the expense of only one addition which can be mapped in the post-adder included in all recent DSP blocks. The latter is relevant if it saves one hard memory block, for instance in the situations depicted below.

Memory blocks are of fixed size (20Kb and 36Kb on current Altera and Xilinx chips respectively). They are dual-ported, which means that we can store two tables in one single block. The largest architecture that fits in a single 20Kb block is for $w = 11$, with a reciprocal table of $2^9 \times 11$ bits and an arctangent table of $2^{10} \times 9$ bits, both packed in a dual-port block configured as $2^{10} \times 20$. In a 36Kb block, the largest architecture that fits is for $w = 12$. Then we may have a $2^{10} \times 12$ reciprocal table and a $2^{10} \times 10$ arctangent table. In both cases the multiplier must be correctly rounded.

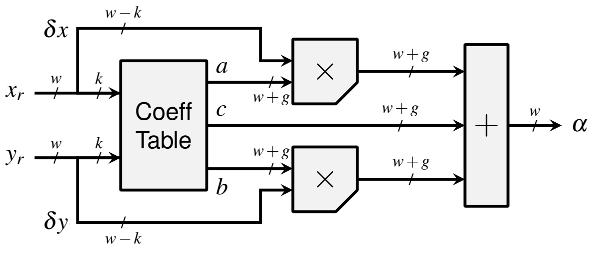

Fig. 7. Architecture based on a first order bivariate polynomial

V. Piecewise bivariate approximation

In this section, we evaluate atan2(y,x) using a piecewise bivariate polynomial approximation of degree 1 or 2. We are not aware of comparable work in the literature.

The idea is to decompose the domain of Figure 4 into a grid of square subdomains indexed by the first few bits of x and y. On each subdomain, we evaluate a bivariate polynomial in the remaing bits. In other words, let x_h (resp. y_h) the number formed of the k (resp. $k+1$) MSBs of x_r (resp. y_r). Let δx and δy the numbers formed of their $w-k-1$ respective LSBs, such as

$$\begin{cases} x_r = x_h + \delta x \\ y_r = y_h + \delta y \end{cases}$$

The bits (x_h, y_h) are used to index a table of the coefficients of bivariate polynomials approximating atan2(y,x) on each subdomain (see Figures 7 and 8):

$$\text{atan2}(y,x) \approx P_{x_h y_h}(\delta x, \delta y) \ .$$

As illustrated by Figure 3, such approximation benefits from the scaling range reduction of Figure 4.

A. First order bivariate approximation

Here

$$P_{x_h y_h}(\delta x, \delta y) = a \cdot \delta x + b \cdot \delta y + c$$

A first way of obtaining a, b and c is the Taylor series around point (x_0, y_0):

$$\begin{aligned} T_1(x,y) &\approx f(x_0,y_0) \\ &+ \tfrac{\partial}{\partial x} f(x_0,y_0)(x-x_0) + \tfrac{\partial}{\partial y} f(x_0,y_0)(y-y_0) \end{aligned}$$

Taking for (x_0, y_0) the centre of a square subdomain indexed by (x_h, y_h), we may obtain the coefficients a, b and c.

The multiplications $a \cdot \delta x$ and $b \cdot \delta y$ are of reduced size, as $\delta x = x - x_h < 2^{-k}$ (idem for δy).

The coefficients a, b and c can also be obtained using a different approach. A degree-1 bivariate polynomial is a plane. Three points from the surface of f define a unique plane that goes through these three points. This plane is a bivariate degree-1 approximation of the function. Therefore, if we chose three points of the form $(x, y, f(x,y))$ with (x,y) inside the square subdomain, the equation of the plane defined by these

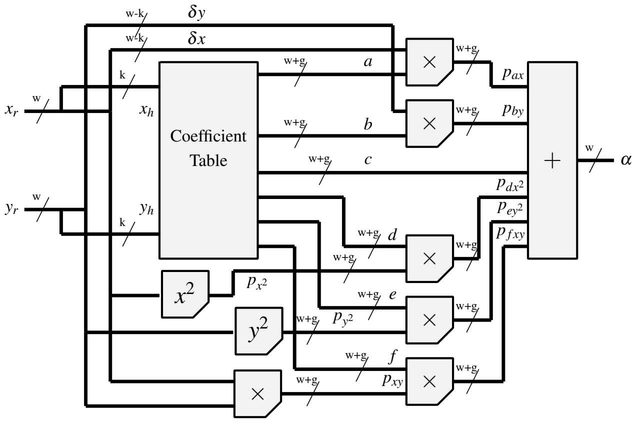

Fig. 8. Architecture based on a second order bivariate polynomial

three points will provide candidate coefficients a, b and c. If the three points are very close to (x_0, y_0) this methods gives the Taylor coefficients. Chosing a larger triangle inside the square subdomain actually provides a better approximation to the function.

B. Second order bivariate approximation

The approach is very similar to the previous one. The motivation for using the second order is to provide a better approximation, which will enable a smaller value of k, hence a smaller coefficient table. We approximate $\operatorname{atan2}(y,x)$ as:

$$\begin{aligned} T_2(x_r, y_r) &\approx P_{x_h y_h}(\delta x, \delta y) \\ &\approx\ a.\delta x + b.\delta y + c + d(\delta x)^2 + e(\delta y)^2 + f\delta x.\delta y \end{aligned}$$

The coefficients a through f are again obtained by using the Taylor series for $\operatorname{atan2}(y_r, x_r)$, around a point (x_0, y_0):

$$\begin{aligned} T_2(x_r, y_r) &= f(x_0, y_0) \\ &+ \tfrac{\partial}{\partial x} f(x_0, y_0)(x_r - x_0) + \tfrac{\partial}{\partial y} f(x_0, y_0)(y_r - y_0) \\ &+ \tfrac{1}{2}\tfrac{\partial^2}{\partial x^2} f(x_0, y_0)(x_r - x_0)^2 \\ &+ \tfrac{1}{2}\tfrac{\partial^2}{\partial y^2} f(x_0, y_0)(y_r - y_0)^2 \\ &+ \tfrac{\partial^2}{\partial x \partial y} f(x_0, y_0)(x_r - x_0)(y_r - y_0) \end{aligned}$$

Replacing f by $\operatorname{atan2}(y_r, x_r)$, and performing the variable change $x_r = x_h + \delta x$ and $y_r = y_h + \delta y$, we get:

$$\begin{aligned} T_2(x_r, y_r) &= \operatorname{atan2}(y_h, x_h) + \tfrac{-y_h}{x_h^2 + y_h^2}\delta x + \tfrac{x_h}{x_h^2 + y_h^2}\delta y \\ &+ \tfrac{x_h y_h}{(x_h^2 + y_h^2)^2}\delta x^2 + \tfrac{-x_h y_h}{(x_h^2 + y_h^2)^2}\delta y^2 + \tfrac{y_h^2 - x_h^2}{(x_h^2 + y_h^2)^2}\delta x^2 \delta y^2 \end{aligned}$$

C. Error analysis

We limit the discussion to the approximation of $\operatorname{atan2}(x_r, y_r)$ by the second order Taylor polynomial, as the case of the first order method is simpler. Our goal is to produce a result which is within 1 unit in the last place (*ulp*) from the exact mathematical result, meaning:

$$\varepsilon_{total} < 2^{-w}$$

where ε_{total} is the total error and:

$$\varepsilon_{total} = \varepsilon_{method} + \varepsilon_{final\_round} + \varepsilon_{round}$$

The magnitude of ε_{method}, the method error, is determined by the parameter k. Limiting the approximation to the second order terms results in:

$$\varepsilon_{method} = \sum_{i=3}^{\infty} \left(c_{x\_i} x^i + c_{y\_i} y^i + \sum_{k,l,\ k+l=i} c_{x\_ky\_l} x^k y^l \right)$$

where the c_{x_i}'s and c_{y_i} are obtained by expanding the Taylor series. ε_{method} is bounded by:

$$\begin{aligned} \varepsilon_{method\_Max} &= c_{x\_max} \sum_{i=3}^{\infty} (2^{-k})^i + c_{y\_max} \sum_{i=3}^{\infty} (2^{-k})^i \\ &+ c_{xy\_max} \sum_{i=3}^{\infty} (2^{-k})^i \\ &< (c_{x\_max} + c_{y\_max} + c_{xy\_max}) \tfrac{2^{-3k}}{1 - 2^{-k}} \end{aligned} \tag{2}$$

c_{x_max}, c_{y_max} and c_{xy_max} being the maximum values of c_x, c_y, and c_{xy}. Equation 2 determines a bound for ε_{method}, which, in turn, determines the value of k, that must satisfy $k > \lceil \frac{w}{3} \rceil$. But this is a pessimistic bound. In practice, we set the value of k more optimistically, then fill the table, and compute ε_{method} in each point of the domain using the actual coefficient values. If the error is above budget, k is increased. Using the actual values for the coefficients provides a tighter bound on ε_{method}.

The error due to the final rounding $\varepsilon_{final\_round}$ can be limited to $1/2$ulps by rounding the final result to the output accuracy.

Each term of $T_2(x_r, y_r)$, except for c, brings to the final sum an error due to the truncation/rounding of the multiplications/squares, and an error due to storing the real coefficients in the table. The expressions for the error terms are given after the variable change. For $a \cdot \delta x$, the error is:

$$\begin{aligned} \varepsilon_{a \cdot \delta x} &= p_{a \cdot \delta x} - a \cdot \delta x \\ &= (\delta x \cdot \tilde{a} + \varepsilon_{mult_{ax_r}}) - a \cdot \delta x \\ &= (\delta x (a + \varepsilon_{a\_table}) + \varepsilon_{mult_{a \cdot \delta x}}) - a \cdot \delta x \\ &= \varepsilon_{mult_{a \cdot \delta x}} + \delta x \cdot \varepsilon_{a\_table} \end{aligned}$$

The same holds for $b \cdot \delta y$ term. The error due to c in the final sum comes from storing the coefficient in the table:

$$\varepsilon_c = \varepsilon_{c\_table}$$

For the second order terms, the error can be found in a similar manner:

$$\begin{aligned} \varepsilon_{d \cdot \delta x^2} &= p_{d \cdot \delta x^2} - d \cdot \delta x^2 \\ &= (p_{\delta x^2} \tilde{d} + \varepsilon_{mult_{d \cdot \delta x^2}}) - d \cdot \delta x^2 \\ &= ((\delta x^2 + \varepsilon_{sqr_{\delta x^2}})(d + \varepsilon_{d\_table}) + \varepsilon_{mult_{d \cdot \delta x^2}}) \\ &- d \cdot \delta x^2 \\ &= \delta x^2 \cdot \varepsilon_{d\_table} + d \cdot \varepsilon_{sqr_{\delta x^2}} + \varepsilon_{sqr_{\delta x^2}} \cdot \varepsilon_{mult_{d \cdot \delta x^2}} \end{aligned}$$

The same holds for $e \cdot \delta y^2$. Lastly, for $f \cdot \delta x \cdot \delta y$ the error can be expressed as:

$$\begin{aligned} \varepsilon_{f \cdot \delta x \cdot \delta y} &= p_{f \cdot \delta x \cdot \delta y} - f \cdot \delta x \cdot \delta y \\ &= (p_{\delta x \cdot \delta y} \tilde{f} + \varepsilon_{mult_{f \cdot \delta x \cdot \delta y}}) - f \cdot \delta x \cdot \delta y \\ &= ((\delta x \cdot \delta y + \varepsilon_{mult_{\delta x \cdot \delta y}})(f + \varepsilon_{f\_table}) \\ &+ \varepsilon_{mult_{f \cdot \delta x \cdot \delta y}}) - f \cdot \delta x \cdot \delta y \\ &= \delta x \cdot \delta y \varepsilon_{f\_table} + f \cdot \varepsilon_{mult_{\delta x \cdot \delta y}} \\ &+ \varepsilon_{mult_{\delta x \cdot \delta y}} \cdot \varepsilon_{mult_{f \cdot \delta x \cdot \delta y}} \end{aligned}$$

This gives the expression for ε_{round}:

$$\varepsilon_{round} = \varepsilon_{a\delta x} + \varepsilon_{b\delta y} + \varepsilon_c + \varepsilon_{d\delta x^2} + \varepsilon_{e\delta y^2} + \varepsilon_{f\delta x\delta y}$$

Let us assume that the errors due to storing the coefficients in the tables are all equal to ε_{table} and that the errors due truncation/rounding of multiplications and squares are ε_{mult} and ε_{sqr} respectively. Thus, ε_{round} becomes:

$$\begin{aligned} \varepsilon_{round} &= \varepsilon_{mult}(2 + f + 2\varepsilon_{sqr} + \varepsilon_{mult}) \\ &+ \varepsilon_{table}(1 + \delta x + \delta y + \delta x^2 + \delta y^2 + \delta x \cdot \delta y) \\ &+ \varepsilon_{sqr}(d + e) \end{aligned} \quad (3)$$

D. Datapath dimensioning

As in the case of the previous sections, the error analysis ensures that the final result is within less than 1 ulp of the mathematical result. It also allows us to create architectures that compute with the minimal amount of resources.

Thus, let us analyze the required number of additional guard bits g due to $\varepsilon_{final\_round}$ and ε_{round}. Due to $\varepsilon_{final\_round}$ we must extend the precision of our computations to $-w$ (from $-w+1$). The contribution due to ε_{round} is worth the discussion. From equation 3, ε_{round} is influenced by the precision of the multiplications and squares. It is also influenced by precision of the stored coefficients, which depends on our choice for the k parameter. This is due to the dependence of ε_{round} to δx and δy, which satisfy $\delta x < 2^{-k}$ and $\delta y < 2^{-k}$.

When deciding the value of g, we attempt try to strike a balance between the size of the multiplications and that of the coefficient tables.

VI. Results and discussion

This section presents synthesis results obtained for Virtex 6 (6vhx380tff1923) using ISE 14.7.

A. Logic-only synthesis

Tables I and II show results when the multipliers as well as the tables are synthesized purely in logic (in FPGA Look-Up Tables, or LUT). In this case, we use faithful truncated multipliers to save resources and latency. Degree 0 uses plain tabulation, degree 1 uses a bipartite approximation [10], and degree 2 and above use a Horner scheme inspired by [12]. The squarers of Figure 8 also use bipartite approximation (which saves resources over a dedicated squarer for the small precisions needed here).

The objective of these tables is to compare the methods in absolute terms. A first observation is that CORDIC behaves extremely well on modern FPGAs. The multiplier- and table-based methods never perform better in area, and only rarely beat its latency. This is especially disappointing as the architecture of Figure 8 exhibits a lot of parallelism. The comparison between CORDIC and the RecipMultAtan method is consistent with the findings of [2].

The implementation of combined sine and cosine (which can be viewed as the inverse function of atan2) was studied in [13]. Interestingly, it lead to the opposite conclusion: for sine/cosine, it was shown that CORDIC had longer latency than multiplier-based methods even when the multipliers were implemented in logic.

The bivariate approximation methods could be compared to a bivariate interpolation technique used in [14]. The technique used there resorts to two sequential interpolation processes (on x, then on y). The results reported in that work (extrapolated from a different FPGA family, Altera Stratix II), tend to show that this bivariate interpolation technique has longer latency and consumes even more resources, mainly due to the high memory requirements.

It is interesting to re-assess the complexity asumption made in the introduction in the light of these results. In principle, classical (i.e. non-redundant) CORDIC is quadratic both in area and delay: The CORDIC algorithm in precision w consists of about w iterations, each consisting of three add/sub operations of about w bits, hence the quadratic area. Besides, there is a carry propagation in each iteration, whose LSB input depends on the MSB of the previous iteration: there is a critical path of size w^2.

On the one hand, Table I indeed exhibit this quadratic area. However, the constant is very small. Specifically, we can observe that the area of CORDIC is roughly $3w^2$. This means that each CORDIC iteration is indeed implemented in 3 LUT per bit. In other words, a multiplexed addition and subtraction is mapped to a row of LUTs, and therefore consumes no more than a simple addition.

On the other hand, the latency of CORDIC does not seem quadratic, it seems linear in w. This is explained by the fact that the carry propagation delay is about 30 times faster than the standard routing used between two iterations. It justifies *a posteriori* the choice of ignoring redundant versions of CORDIC.

TABLE I
Logic-only synthesis results

Bitwidth	Method	LUT	Latency (ns)
CORDIC			
8		173	9.3
12		435	14.6
16		734	19.7
24		1504	31.0
32		2606	43.1
Taylor degree 1 / Plane			
8		207	12.64
12		1258	14.74
16		37744	20.20
Taylor degree 2			
8		356	13.72
12		469	14.75
16		1509	17.90
RecipMultAtan			
8	degree 0	175	11.8
12	degree 0	683	16.2
12	degree 1	443	19.0
16	degree 1	1049	19.1
24	degree 2	2583	35.2
32	degree 2	6190	40.7
32	degree 3	5423	50.8

TABLE II
BREAKDOWN OF AREA AND LATENCY FOR RECIPMULTATAN METHOD

		LZC	Shift	Recip	Mult	Atan
12 bits	area	12	2x36	149	159	220
	delay	1.63	1.72	2.23	4.22	2.54
16 bits	area	16	2x47		294	
	delay	2.5	1.76		5.9	

A second observation is that the degree 1 bivariate approximation is never interesting. Its size explodes too fast, and this even impacts the latency.

The scaling-based argument reduction is quite small and fast, as illustrated by Table II. In [2], the shifts were implemented as one-hot encoding then multiplication in a DSP block. This is probably overkill.

B. Pipelining, DSP- and table-based results

Small multipliers (or DSP blocks) and memories are embedded in all recent FPGAs. Table III shows some synthesis results for pipelined operators at various frequencies, exploiting these resources when possible. The multiplier-based methods do indeed improve logic count, but bring no clear advantage in terms of latency nor frequency compared to a pipelined unrolled CORDIC. This probably reflects the current weakness of FloPoCo at managing the pipelining of such entities.

Still, we again observe the excellent match of CORDIC to FPGA logic: as the third line of TableIII shows, it is possible to pack two 16-bit iterations in one pipeline stage operating at nearly 400MHz.

Finally, these results should not be extrapolated to ASIC. There, without carry propagation and large LUTs, CORDIC will appear much less favorably.

VII. CONCLUSION AND FUTURE WORK

This article compares several methods for the evaluation of the atan2 function. The most novel method, based on piecewise bivariate polynomials, does reduce the latency as much as expected, at least on FPGAs. There are many improvements to bring to this method, the first being to reduce k using degree-2 approximation technique that are more accurate than Taylor. Unfortunately we are not aware of a Remez or minimax polynomial approximation algorithm for functions of two inputs. The problem is that the alternation property on which Remez is based [1] is difficult to transpose in two dimensions.

TABLE III
RESULTS FOR PIPELINED 16-BIT ARCHITECTURES

Method	LUT + Reg.	BRAM + DSP	Speed cycles@freq.
CORDIC	816 + 44		2@191
	799 + 202		5@274
	796 + 336		8@389
RecipMultAtan 1	320 + 51	2+1	2@112
	315 + 68	2+1	3@199
RecipMultAtan 2	425 + 199	0+5	10@130
	432 +250	0+5	14@253
Taylor 2	331 + 53	4+6	1@135
	327 + 103	4+6	3@144
	329 + 140	4+6	5@220

Current work also focuses on improving the pipelining of the multiplier-based methods to make the best use of the FPGA embedded resources.

To make things even better for CORDIC, it should be noted that it may also compute the module $\sqrt{x^2+y^2}$ along with the angle [1]. This costs only one additional constant multiplication by $1/K$.

Among the possibilities to explore around atan2, [15] decomposes the computation into two successive rotation, a coarse one and a finer one. Both first approximate $z = y/x$, then compute atan2(z). This provides a blend between CORDIC and multiplicative methods.

A table- and multiplier- based combined range reduction and scaling method could enable the bivariate techniques to scale better, but again at the expense of latency.

ACKNOWLEDGEMENTS

This work was supported by the French National Research Agency through the INS program *MetaLibm*.

REFERENCES

[1] J.-M. Muller, *Elementary Functions, Algorithms and Implementation*, 2nd ed. Birkhäuser, 2006.

[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, 2008.

[3] J. E. Volder, "The CORDIC trigonometric computing technique," *Electronic Computers, IRE Transactions on*, pp. 330–334, 1959.

[4] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *International Symposium on Field Programmable Gate Arrays*, 1998.

[5] P. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 2009.

[6] S. Gal, "An accurate elementary mathematical library for the IEEE floating point standard," *ACM Transactions on Mathematical Software*, vol. 17, 1991.

[7] S. Story and P. T. P. Tang, "New algorithms for improved transcendental functions on IA-64," in *14th IEEE Symposium on Computer Arithmetic*. IEEE Comput. Soc, 1999.

[8] S. Rajan, S. Wang, and R. Inkol, "Efficient Approximations for the Four-Quadrant Arctangent Function," in *Canadian Conference on Electrical and Computer Engineering*. IEEE, 2006.

[9] M. Saber, Y. Jitsumatsu, and T. Kohda, "A low-power implementation of arctangent function for communication applications using FPGA," in *Signal Design and its Applications in Communications*. IEEE, 2009.

[10] F. de Dinechin and A. Tisserand, "Multipartite table methods," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 319–330, 2005.

[11] R. Gutierrez, V. Torres, and J. Valls, "FPGA-implementation of atan(Y/X) based on logarithmic transformation and LUT-based techniques," *Journal of Systems Architecture*, vol. 56, 2010.

[12] F. de Dinechin, M. Joldes, and B. Pasca, "Automatic generation of polynomial-based hardware architectures for function evaluation," in *Application-specific Systems, Architectures and Processors*. IEEE, 2010.

[13] F. de Dinechin, M. Istoan, and G. Sergent, "Fixed-point trigonometric functions on FPGAs," *SIGARCH Computer Architecture News*, vol. 41, no. 5, pp. 83–88, 2013.

[14] D. Wang, M. Ercegovac, and Y. Xiao, "Complex function approximation using two-dimensional interpolation," *IEEE Transactions on Computers*, vol. 63, no. 12, pp. 2948–2960, Dec 2014.

[15] D. Hwang and A. Willson, "A 400-MHz processor for the conversion of rectangular to polar coordinates in 0.25-μm cmos," *IEEE Journal of Solid-State Circuits*, vol. 38, 2003.