Power-Aware Architectural Exploration of the CORDIC Algorithm

Jennifer Manica, Roberto Passerone and Luca Rizzon
Dipartimento di Ingegneria e Scienza dell'Informazione – Università degli Studi di Trento
Povo di Trento, Italy 38123

***Abstract*—Mobile applications require the use of specialized design techniques to reduce power consumption and maximize battery life. New implementation technologies are able to reduce energy at the device level. At the same time, architectural choices can lead to significantly different power consumption profiles for equivalent implementations of the same function. In this paper we focus on the architectural level, and analyze various implementation alternatives for the CORDIC function. Our results provide insight into the trade-offs between area, performance and power consumption, and give designers directions for their architectural choices.**

I. Introduction

In the last decade, the widespread adoption of battery-powered mobile devices has made power consumption become one of the most critical performance metric in the design of digital electronic systems. In addition to the obvious lifetime improvement, reducing power consumption simplifies problems related to thermal management and battery size, which significantly affect the reliability and cost of a device [1]. At the same time, constraints on area and performance limit the range of optimal implementations, which must be searched through a careful balance of conflicting requirements.

Several factors influence the power consumption of a design. The implementation technology, for instance, determines the overall characteristics of the devices and their power profile. At the design level, techniques such as dynamic voltage and frequency scaling are used to selectively activate and modulate the performance of the computational blocks to adapt to the varying requirements of the applications [2], [1]. Likewise, different implementation alternatives for the core functions of the system exhibit wildly different performance characteristics which are not always easy to predict using intuition and back-of-the-envelope calculations.

In this work, we focus on this last aspect and analyze the performance of the CORDIC algorithm [3], [4], whose flexibility and simplicity make it ideal in diverse applications such as direct frequency synthesis, digital modulation and coding or direct and inverse kinematic computation for robot manipulation [4]. We chose three different implementation architectures, which we have compared on the basis of area, throughput and power consumption. The results, based on an FPGA technology, demonstrate how the specifics of each architecture contribute to the determination of the final performance metrics, and can be used to quickly choose a solution given the system constraints.

A. Related work

We target an FPGA CORDIC implementation, and take as reference the architectures reviewed by Andraka [3]. Other studies have targeted low power implementations of CORDIC. Sarrigeorgidis and Rabaey study methods to reduce the number of iterations and make use of redundant number representations to improve performance [5]. Kim et al. employ hard wired operations and interpolation to achieve the same [6]. Our work is orthogonal, as we wish to study the impact of different architectural choices on the overall performance. Indeed, the above techniques can be applied to all the architectures we consider. Studies similar to ours have been conducted for *other* algorithms in widespread use. Rhee et al. consider the H.264/AVC encoder and use various techniques to trade-off power consumption with compression efficiency [7]. Sherazi et al. focus on digital filters and evaluate several structures in the sub-V_T domain [8]. Roth et al. discuss the trade-off in LDPC decoders [9], while Lin et al. study ways to improve the power consumption of Viterbi decoders [10]. Our results complement the existing literature with a characterization of the CORDIC architectures.

In the following, we first describe the principles of the CORDIC algorithm, and describe the chosen architectural solutions in Section II. Then, Section III discusses the results of the architectural analysis. Section IV concludes the paper outlining directions for future work.

II. CORDIC algorithm and architectures

The CORDIC algorithm is used in a variety of applications that require the computation of trigonometric and hyperbolic functions. CORDIC, short for Coordinate Rotation Digital Computer, is an iterative algorithm used to efficiently compute vector rotations. In its basic form, the algorithm computes the rotation of a vector (x, y) by an angle φ into a sequence of elementary rotations for angles whose trigonometric tangent is a power of 2. This way, a single elementary rotation can be computed as [3]

$$x_{i+1} = x_i - y_i d_i 2^{-i}, \qquad y_{i+1} = y_i + x_i d_i 2^{-i}$$
$$z_{i+1} = z_i - d_i \tan^{-1}(2^{-i}).$$

Variable z_i, initialized with the desired angle φ, represents the residual rotation and is updated at each iteration with the effective rotation. The coefficient d_i determines the direction of rotation, and is equal to 1 if $z_i \geq 0$, and -1 otherwise. Hence, the algorithm successively approximates the desired rotation with incrementally smaller elementary rotations, until z_i converges to 0. The advantage of this formulation is that it does not need a multiplier, since powers of 2 can be implemented with shifters. It requires a look-up table to compute $\tan^{-1}$, which is nonetheless restricted to a small number of constant values. This algorithm alters the length of the vector while it rotates it by a factor that can be considered constant for a sufficient number of iterations, and will therefore be ignored in this study.

A. CORDIC architectures

The CORDIC iteration can be implemented using several architectures [3]. The iterative nature of the algorithm lends itself to an iterative implementation, as shown in Figure 1. The intermediate values for the vector coordinates are stored

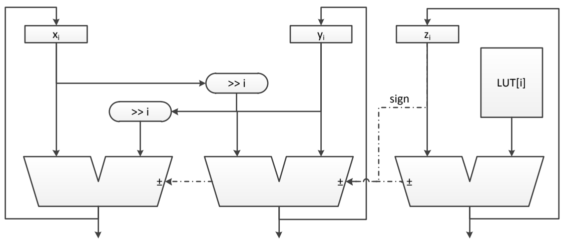


Fig. 1. Iterative CORDIC architecture

in the x_i and y_i registers, while the residual angle in z_i. The shifts are implemented as barrel shifters, shown in the figure as rounded rectangles. A counter, not shown, indexes the iterations and selects the appropriate shift value. The $\tan^{-1}$ operator is implemented as a look-up table (LUT) which stores the required values, also selected by the iteration counter. The three adders execute the main update operation. These adders perform either an addition or a subtraction according to the formula and the sign of z_i.

The values are stored using a fixed point representation, with one bit for the sign (in two-complement), two bits for the integer part, and the remaining bits for the fractional part. All angles are measured in radians. The number of iterations n required to compute the results depends on the total number of bits chosen for the operands. The higher the number of bits, the higher the number of iterations required to exploit the extra representation. In our experiments, we stop the iterations once the $\tan^{-1}$ operator returns a value below the chosen precision (i.e., zero). In particular, we have $n = 6, 9, 13, 18, 21, 29$ iterations for $8, 12, 16, 20, 24$ and 32 bits.

The performance of the iterative architecture depends on the clock frequency f_c and on the number n of iterations. The throughput is equal to $th = f_c/n$, while the latency is $l = n/f_c$. The area increases with the number of bits, because the registers, the shifters, the adder and the LUT become larger. It is instead only marginally affected by the number of iterations, since only the LUT increases in size.

An alternative architecture consists in unrolling the iteration. In this case, one could either construct a *feed-through* architecture, where each adder is connected directly to the following adder, or a *pipeline* architecture, where the stages are separated by registers. The pipeline is shown in Figure 2. The number of stages equals the number of iterations. The advantage of a pipeline is that more than one computation is active at the same time, increasing the throughput which is

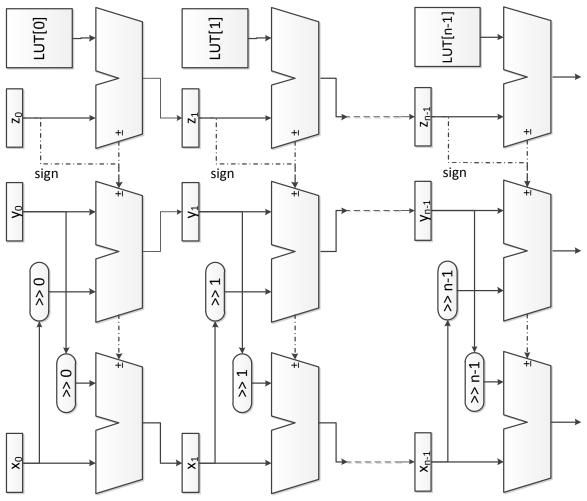


Fig. 2. Pipeline CORDIC architecture

now equal to the clock frequency. The latency, on the other hand, is the same as the iterative solution, since the data has to traverse the entire structure.

The maximum clock frequency of the pipeline architecture is higher than that of the iterative solution. This is because for each stage the amount of shift is known beforehand, so one has simply to re-route wires, instead of using a barrel shifter. Hence, shifting does not incur in any extra delay, and one component is removed from the critical path. For the same reason, the area is not n times as large as that of the iterative architecture, because the shifters are removed.

The feed-through solution does not have pipeline registers. In this case, the clock frequency will be significantly lower, since the critical path now encompasses all the stages of the architecture, but not n times lower, since the delay introduced by the registers is removed. The throughput is equal to the clock frequency, while the latency is equal to the clock period.

For these architectures, the area increases both with the number of bits, as well as with the number of iterations, since every new iteration requires an additional stage.

III. Architecture analysis and results

We have developed the above architectures in VHDL, for word sizes ranging from 8 to 32 bits. The code was synthesized for the Altera DE1 Development Board, which employs a 90nm Cyclone II FPGA, using the Quartus tool with default settings. Our architectural analysis is in particular focused on analyzing the power consumption of the different architectures. In this study, we concentrate on dynamic power and neglect the contribution of static power (e.g., leakage) since its contribution is not significant in the chosen FPGA technology. In general, the dynamic power consumption P depends on the number of times that signals switch their state, and is proportional to several factors [1]:

$$P = \alpha \cdot f_{ck} \cdot C_L \cdot V_{DD}^2,$$

where f_{ck} is the clock frequency, V_{DD} is the power supply, C_L is proportional to the total capacitance in the circuit, and α

is a factor that accounts for the switching activity. The various architectures will differ in terms of their size (and thus C_L), of the frequency at which they need to run to achieve a certain throughput, and the activity α in the internal nodes.

Determining the internal node activities may be difficult to do in general. In our study, we have simulated the various architectures driving them with the same test-bench requesting the computation of the trigonometric functions for a series of 32 angles. The results are used by the Quartus tool to precisely determine the internal activity, which is therefore averaged out on thousands of transitions. The tool is then able to provide us with accurate estimates of the power consumption based on the models of the FPGA components. In this study, we have considered the power consumed by the core logic only, and excluded the contribution of the I/Os, which is of comparable size and may therefore obscure the actual dependency of the power from the architectural parameters. For the same reason, it was not possible to compare the results to measurements, since there is no access to the internal circuits. Nevertheless, our estimates were obtained using detailed simulation (after place and route) with the manufacturer provided models.

A. *Circuit size*

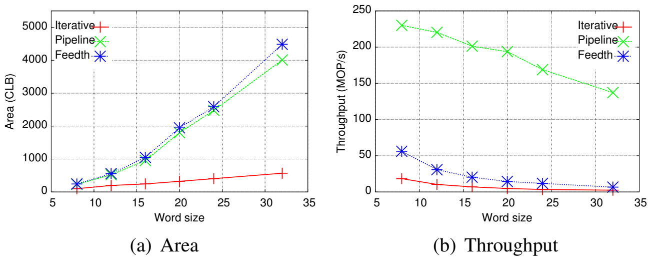
Figure 3(a) shows the dependency of the area of each architectural solution as a function of the word size. The iterative architecture shows a linear area increase with the number of bits, due to the larger size of the registers, adders, the shifters and the look-up table. The size of the look-up table also increases, since the architecture works with more bits.

The size of the pipeline and feed-through architectures also increases with the number of bits in a close to linear fashion. However, the *rate* of increase is higher than that of the iterative architecture, because the algorithm requires a larger number of iterations to converge (and make use of the extra bits), and therefore *more stages*. As expected, the size of the pipeline and feed-through architectures is not n times as large as that of the iterative solution, where n is the number of iterations. This is because, as discussed, the shifters in these architectures can be implemented by simply moving wires, rather than using a barrel shifter. In addition, the iterative architecture requires a small control circuit to keep track of the number of iterations, which is not present in the other two.

In the FPGA implementation, the feed-through architecture requires slightly more area than the pipeline architecture, although it obviously uses fewer components (it does not have intermediate pipeline registers). This result is specific to FPGAs, whose fabric consists of logic blocks that already incorporate a register which is conveniently used by the pipeline implementation, while it is essentially wasted in the feed-through solution. The longer combinational chain also makes it more difficult for the tool to optimize the area, resulting in some overhead. Nonetheless, automatic design tools might be able to reclaim these extra registered for use with the logic required by the surrounding system. This effect would not exist in a standard cell implementation where every function has a dedicated component.

B. *Performance*

Figure 3(b) shows the maximum throughput achievable by each architecture as a function of the number of bits for our chosen FPGA. Throughput is computed as the number of computations per unit time. Hence, for the pipeline and feed-through it corresponds to the clock frequency, while for the iterative architecture it must be scaled down to account for the n sequential iterations.



(a) Area (b) Throughput

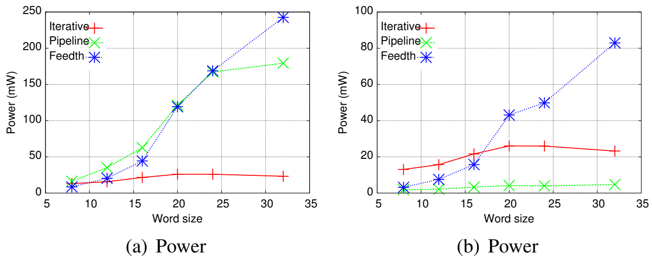
Fig. 3. CORDIC size (a) and throughput (b) for different architectures

As expected, the pipeline architecture has the largest throughput, since it takes advantage of parallelism. The throughput decreases linearly with increasing precision, because the larger adders incur larger delays. The feed-through architecture performs better than the iterative, since it eliminates the overhead due to intermediate registers, while at the same time eliminates the delay penalty due to the shifters.

Notice that the decrease in maximum performance for the feed-through and the iterative architectures is more than linear with the number of bits. This is because not only do the adders (and the shifters, for the iterative) become slower, but also because the higher precision requires a higher number of iterations for the iterative architecture, and more stages for the feed-through architecture, increasing the length of the critical path. Also the pipeline solution has more stages. However, the pipeline registers break the critical cycles at every stage, making the decrease linear.

C. *Power consumption*

Figure 4(a) shows the power consumption of each CORDIC architecture as a function of the number of bits, running the computation at its maximum throughput. Power consumption, in general, increases with the number of bits due to the larger size of the circuit. However, higher precision also implies lower throughput, i.e., a lower clock frequency which tends to decrease the power consumption. For this reason, power tends to level off with precision, and in fact eventually decreases in the case of the iterative architecture, and to some extent for



(a) Power (b) Power

Fig. 4. CORDIC power consumption at maximum (a) and iterative throughput (b) for different architectures

the pipeline. The feed-through architecture is not significantly influenced by this effect since clocked registers account for a much lower percentage of the circuit than in the other cases.

A better comparison can be made by running all architectures so that they achieve the same throughput. Figure 4(b) shows how power consumption changes when, for each word size, all architectures are run at the throughput of the iterative solution. We notice how the power consumption of the pipeline architecture is well below that of the iterative solution. This is because the *same* throughput is obtained with a clock frequency which is n *times lower*, while the increase in area and activity factor is not as large. The feed-through version, instead, shows a much more marked increase of power consumption with the increased number of bits. This is due to an increase in the activity factor α. This effect can be explained as follows. The activity factor depends on the amount of switching in the circuit. The adders update their outputs in sequence, starting typically with the least significant bits, and adjusting the higher bits as the carry propagates through the carry chain. The result is that the value at the output is updated several times during the computation. In the feed-through architecture, these updates are immediately propagated to the adder downstream, which compounds its own updates. This produces an elevated amount of switching, which is reflected by the increased power consumption. This cascading effect is instead blocked by the pipeline registers and by the iteration registers in the other two architectures. The graph also shows that this effect is more pronounced as the adders become larger, and as the number of stages increases, so that the feed-through solution quickly becomes the worst performing architecture in terms of power, while it is comparable to the other two for lower precisions.

D. Architectural trade-offs

The data collected in the experiments can be used to construct trade-off graphs where one metric is compared to another. One example is shown in Figure 5, where power consumption and throughput are plotted against each other, on log scale axis, for each of the different architectures and word size. The best solutions are those closer to the left (lower power) and to the top (higher throughput). These plots make the trade-off between the different objectives explicit. In particular, they show when a solution dominates another in more than one metric, making it preferable for the implementation. For instance, the pipelined architecture is to be preferred over the feed-through, especially for large word size, and also over the iterative, when area is not an issue. The plot can be populated with additional data points, not shown here to avoid clutter, where each of the architecture is run at intermediate speeds to achieve different trade-offs, to construct a Pareto front. The data is also useful for higher level design exploration algorithms to help them choose the best implementation.

IV. Conclusions

We have presented the architectural analysis of various CORDIC architectures. The data collected through accurate simulation allows a designer to choose the architecture with the best performance in terms of area, throughput and power, relative to the system requirements. There are several directions for improving this study. One may consider the effect of lowering

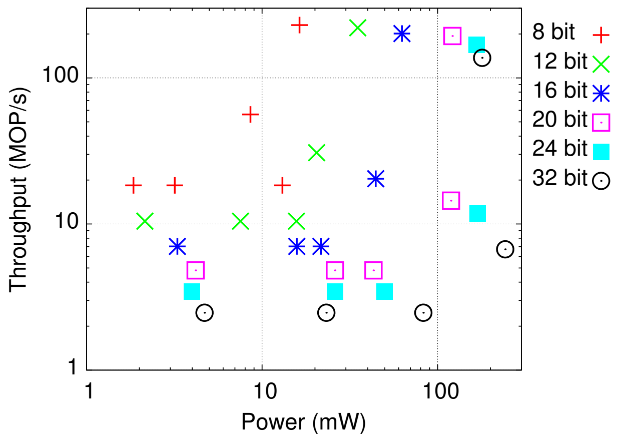


Fig. 5. Power vs. throughput trade-off

the supply voltage when running at clock frequencies below the maximum (voltage scaling is not possible in our targeted FPGA). From an architectural point of view, one may consider intermediate solutions where, for instance, the iteration is only partially unrolled into a pipeline to strike the desired trade-off between area and performance. In an orthogonal direction, one may consider different implementations for the components used in the architectures, such as different kinds of adders whose carry propagation has a large influence on the feed-through architecture. These considerations are part of our future work.

References

[1] J. Rabaey, *Low Power Design Essentials*. New York, NY, USA: Springer, 2009.

[2] P. J. M. Havinga and G. J. M. Smit, "Design techniques for low-power systems," *J. Syst. Archit.*, vol. 46, no. 1, pp. 1–21, January 2000.

[3] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Sixth International Symposium on Field Programmable Gate Arrays*, Monterey, CA, February 22-24, 1998.

[4] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *Trans. Cir. Sys. Part I*, vol. 56, no. 9, pp. 1893–1907, September 2009.

[5] K. Sarrigeorgidis and J. Rabaey, "Ultra low power CORDIC processor for wireless communication algorithms," *Journal of VLSI Signal Processing 38*, pp. 115–130, 2004.

[6] Y. B. Kim, Y.-B. Kim, and J. T. Doyle, "A low power CMOS CORDIC processor design for wireless telecommunication," in *50th Midwest Symp. on Circuits and Systems*, Montreal, Québec, August 5-8, 2007.

[7] H. Kim, C. E. Rhee, J.-S. Kim, S. Kim, and H.-J. Lee, "Power-aware design with various low-power algorithms for an H.264/AVC encoder," in *IEEE International Symposium on Circuits and Systems (ISCAS11)*, Rio de Janerio, Brazil, May 15-18, 2011, pp. 571–574.

[8] S. Sherazi, J. Rodrigues, O. Akgun, H. Sjöland, and P. Nilsson, "Ultra low energy vs throughput design exploration of 65 nm sub-VT CMOS digital filters," in *NORCHIP*, Tampere, Finland, November 15-16, 2010.

[9] C. Roth, A. Cevrero, C. Studer, Y. Leblebici, and A. Burg, "Area, throughput, and energy-efficiency trade-offs in the VLSI implementation of LDPC decoders," in *IEEE International Symposium on Circuits and Systems (ISCAS11)*, Rio de Janerio, Brazil, May 15-18, 2011.

[10] Y.-M. Lin, W.-C. Liu, L.-Y. Chang, C.-Y. Lien, P.-Y. Chen, and S.-C. Chen, "A low-power IP design of Viterbi decoder with dynamic threshold setting," in *IEEE International Symposium on Circuits and Systems (ISCAS10)*, Paris, France, May 30-June 2, 2010, pp. 585–588.