FORSCIENCE
ISSN 2318-6356

DOI: 10.29069/forscience.2020v8n1.e685
Recebido em: 03/05/2019
Aprovado em: 18/10/2019
Publicado em: 16/04/2020

IMPLEMENTAÇÃO DO ALGORITMO CORDIC PARA CÁLCULO DE SENO E COSSENO EM FPGA[1]

Ricardo Gonçalves de Aguiar[2]
Valfride Wallace do Nascimento
Fernando Lessa de Oliveira Magalhães
Hugo Daniel Hernandez Herrera

RESUMO

O presente trabalho apresenta a implementação em *hardware* para cálculo das funções trigonométricas seno e cosseno por meio de rotação vetorial utilizando o algoritmo CORDIC. O código foi sintetizado no FPGA DE10-Lite da Terasic Inc. com as ferramentas de desenvolvimento da Altera/Intel para família MAX 10. Com o objetivo de avaliar o desempenho da implementação em *hardware*, é realizado uma comparação com o algoritmo desenvolvido em Python e seus resultados são apresentados. Tais resultados demonstram a precisão numérica da arquitetura proposta para a implementação do CORDIC no FPGA versus Python, não considerando o tempo de execução.

Palavras-chave: CORDIC. Verilog. FPGA.

IMPLEMENTATION OF CORDIC ALGORITHM FOR SINE AND COSINE CALCULATION IN FPGA

ABSTRACT

The present work presents the hardware implementation for calculating the trigonometric functions sine and cosine by means of vector rotation using the CORDIC algorithm. The code was synthetized in the DE10-Lite Board from Terasic Inc. with Altera/Intel development tools for MAX Family 10. In order to evaluate the performance of the hardware implementation, a comparison with the algorithm developed in Python is performed and its results are presented. These results demonstrate the numerical precision of the proposed architecture for the implementation of CORDIC in FPGA versus Python, not considering the execution time.

Keywords: CORDIC. Verilog. FPGA. Python.

[1] **Como citar este artigo:**
AGUIAR, R. G.; NASCIMENTO, V. W.; MAGALHÃES, F. L. O.; HERRERA, H. D. H. Implementação do algoritmo CORDIC para cálculo de seno e cosseno em FPGA. **ForScience**, Formiga, v. 8, n. 1, e00685, jan./jun. 2020. DOI: 10.29069/forscience.2020v8n1.e685.

[2] **Autor para correspondência:** Ricardo Gonçalves de Aguiar, rigoaguia@gmail.com

1 INTRODUÇÃO

CORDIC (Coordinate Rotation Digital Computer) (VOLDER, 1959a), também conhecido como algoritmo de Volder, em homenagem ao seu criador Jack E. Volder (VOLDER, 1959b), é um método simples para calcular funções trigonométricas, hiperbólicas, logarítmicas e exponenciais (SWARTZLANDER JR., 1990). Por basear-se em operações simples como soma, subtração, deslocamento de bits e *look-up tables* (LUT), o CORDIC é empregado em CPUs de baixa complexidade, nas quais geralmente não há um multiplicador implementado em *hardware* (microcontroladores e FPGA's) (MEHER et al., 2009).

A premissa fundamental do CORDIC é que o valor de uma função pode ser calculada, um bit por vez, como uma série alternada (VOLDER, 1959a). A cada bit gerado pelo CORDIC, um valor de correção é somado a partir de uma LUT de modo que, a cada iteração, o valor final da função ganha maior precisão. Entretanto, a quantidade de bits ofertados pelo *hardware* influencia diretamente na precisão do valor final a ser calculado. Caso a série seja restrita, o valor é equivalente a um arredondamento para n casas decimais (VOLDER, 1959b).

O CORDIC é utilizado principalmente nas áreas de análise de sinais, processamento de imagem, sistemas de comunicação, robótica e computação gráfica (MEHER et al., 2009). Deve-se isso ao fato de sua performance no cálculo de funções trigonométricas e hiperbólicas, multiplicação de números complexos e reais, divisões, autovalores e autovetores, solução de sistemas lineares, decomposição QR, entre outras várias atividades (MASRAM; KARULE, 2014). Recentemente, o CORDIC tem sido amplamente utilizado em aplicações em biomedicina e especialmente em conjunto com FPGAs (BISWAL; BANERJEE, 2010).

Quando um multiplicador está presente em *hardware*, o uso do CORDIC não é necessário, já que o cálculo por meio de séries de potência e LUT é mais eficiente. O uso do CORDIC atualmente via *software* é desnecessário devido a presença de registradores de ponto flutuante nas CPUs (VOLDER, 1959a).

O principal objetivo do presente trabalho é demonstrar o algoritmo CORDIC sob a implementação de rotação circular, quando empregado em FPGA utilizando arquitetura paralela, no cálculo de funções trigonométricas (seno e cosseno). A implementação descrita neste trabalho é amplamente utilizada em processamento de sinais digitais com FPGA, como descrito no trabalho de Aimei Tang em 2016 (TANG et al., 2016).

O trabalho está organizado da seguinte forma: a seção 2 apresenta o referencial teórico descrevendo o algoritmo CORDIC. A seção 3 apresenta o projeto e a arquitetura adotada. A seção 4 apresenta a implementação na FPGA. A seção 5 apresenta os resultados e discussões. E, por fim, as conclusões principais a partir do estudo realizado no presente trabalho na seção 6.

2 REFERENCIAL TEÓRICO

2.1 Algoritmo CORDIC - modo rotação

O presente trabalho tem como principal objetivo calcular as funções trigonométricas Seno e Cosseno de um ângulo z_0. Para isso, o CORDIC foi implementado utilizando o modo rotação para encontrar as coordenadas x e y dentro do círculo trigonométrico unitário do ângulo z_0 considerado (MULLER, 1961). Para começar, o algoritmo considera um vetor inicial v_0 (Equação 1) com as seguintes coordenadas:

$$v_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{1}$$

Em cada iteração do algoritmo, o vetor v_0 é rotacionado em sentido anti-horário ou horário em passos decrescentes a fim de obter uma aproximação das magnitudes de x e y (Equação 2). Os passos são definidos segundo a função trigonométrica $\arctan(2^{-i})$ para i um número natural iniciando em 0.

$$\begin{cases} x_0 = \frac{1}{K} = 0.6072 \\ y_0 = 0 \\ z_0 = \theta \end{cases} \tag{2}$$

Para a Equação 2, a constante K é calculada, como será mostrado adiante, com base nas Equações 10 e 11. Formalmente, a cada iteração, o algoritmo calcula uma rotação por meio da multiplicação do vetor v_i com a matriz de rotação R_i:

$$v_i = R_i v_{i-1} \tag{3}$$

Dada a matriz R_i da Equação 4:

$$R_i = \begin{bmatrix} \cos(\gamma_i) & -\sin(\gamma_i) \\ \sin(\gamma_i) & \cos(\gamma_i) \end{bmatrix} \tag{4}$$

Com as seguintes propriedades trigonométricas das Equações 5 e 6:

$$\cos(\gamma_i) = \frac{1}{\sqrt{1+\tan^2(\gamma_i)}} \tag{5}$$

$$\sin(\gamma_i) = \frac{\tan(\gamma_i)}{\sqrt{1+\tan^2(\gamma_i)}} \tag{6}$$

É possível reduzir a Equação 3 para o seguinte formato:

$$v_i = \frac{1}{\sqrt{1+\tan^2(\gamma_i)}} \begin{bmatrix} 1 & -\tan(\gamma_i) \\ \tan(\gamma_i) & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \tag{7}$$

Para que seja possível realizar as operações de aproximação das funções trigonométricas de cosseno e seno, é necessário limitar o ângulo γ_i de tal forma que ele assuma valores $\tan(y_i) = \mp 2^{-i}$. Desta maneira, a multiplicação com a tangente pode ser simplificada por uma divisão de potência de dois, sendo assim de fácil cálculo com uma simples operação de deslocamento de bits. Com isso, podemos simplificar a expressão para a Equação 8 abaixo:

$$v_i = \frac{1}{\sqrt{1+2^{-2i}}} \begin{bmatrix} 1 & -\sigma_{ii}2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \tag{8}$$

A partir deste resultado é possível estabelecer as seguintes relações matemáticas da Equação 9:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \frac{1}{\sqrt{1+2^{-2i}}} \begin{bmatrix} x_{i-1} - \sigma_i 2^{-i} y_{i-1} \\ \sigma_i 2^{-i} x_{i-1} + y_{i-1} \end{bmatrix}$$

Omitindo a constante de escala $\frac{1}{\sqrt{1+2^{-2i}}}$, obtemos:

$$\begin{cases} x_i = & x_{i-1} - \sigma_i 2^{-i} y_{i-1} \\ y_i = & y_{i-1} - \sigma_i 2^{-i} x_{i-1} \end{cases} \rightarrow \begin{cases} x_{i+1} = & x_i - \sigma_i 2^{-i} y_i \\ y_{i+1} = & y_i - \sigma_i 2^{-i} x_i \end{cases} \rightarrow \begin{cases} x_{i+1} = & x_i - \sigma_i 2^{-i} y_i \\ y_{i+1} = & y_i - \sigma_i 2^{-i} x_i \\ z_{i+1} = & z_i - \sigma_i \arctan(2^{-i}) \end{cases} \tag{9}$$

Da Equação 9 o termo $z_{i+1} = z_i - \sigma_i \arctan(2^{-i})$ tem como fução acumular o valor de todas as rotações e comparar com seu valor z_0, de maneira que seja possível identificar o sentido de rotação a ser realizado na próxima iteração (sentidos horário ou anti-horário). O valor de σ (Equações 8 e 9) alterna entre -1 e 1 e serve para indicar o sentido de rotação do vetor. Durante as iterações do algoritmo é possível ignorar a seguinte expressão e aplicá-la ao final como um fator de escala, como sugere a Equação 10:

$$K = \frac{1}{\sqrt{1+2^{-2i}}} \tag{10}$$

Como fator de escala, a expressão anterior necessita ser trabalhada de acordo com a Equação 11:

$$K(n) = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}} \tag{11}$$

Se o número de iterações for limitado, o valor de K (Equação 12) pode ser aproximado a uma constante:

$$K = \lim_{n\to\infty} K(n) \approx 0.607252935008881256 1694 \tag{12}$$

Em alguns casos, para reduzir a complexidade do algoritmo, o K não é corrigido. Por este motivo, há um ganho de processamento A, calculado de acordo com a Equação 13:

$$A = \frac{1}{K} = \lim_{n\to\infty} \prod_{i=0}^{n-1} \sqrt{1+2^{-2i}} \approx 1.64676025812107 \tag{13}$$

Ao final do algoritmo os valores do seno e cosseno de z_0 serão, respectivamente, as coordenadas y e x.

2.2 Algoritmo CORDIC - modo vetorização

Neste modo assume-se que a coordenada x_0 é um valor positivo e que a coordenada y_0 é escolhida arbitrariamente. Assim como o modo de rotação, a vetorização é baseada em rotações sucessivas do vetor, com o objetivo de rotaciona-lo para o eixo x e anular o valor de y_i, desta maneira o ângulo z_0 conterá o total de rotações (em unidades de ângulos) e o valor final de x_i será a magnitude original do vetor escalada por K.

3 PROJETO E ARQUITETURA

3.1 Convergência

Umas das considerações de projeto para um processador CORDIC é a região de convergência. Na forma básica, o algoritmo CORDIC não converge para todas as coordenadas de entrada. Para o modo rotação, o algoritmo CORDIC converge desde que o valor absoluto do ângulo de rotação não seja maior que 1,7433 radianos, ou aproximadamente 99,88°. A abordagem mais direta para lidar com a questão da convergência é primeiro notar que o gama natural de convergência se estende além do ângulo $\frac{\pi}{2}$. Ou seja, o conjunto básico de equações converge para o intervalo de $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Para aumentar a faixa de convergência (de $[-\frac{\pi}{2}, \frac{\pi}{2}]$ para $[-\pi, \pi]$), é necessário uma rotação inicial do vetor, de forma a colocá-lo nos quadrantes I e IV (HAUCK, 2008).

3.2 Projeto da unidade de micro-rotação

Para implementar o algoritmo CORDIC em *hardware*, é necessário uma unidade de micro-rotação que tem a finalidade de girar o vetor, em cada iteração, em um valor determinado por θ_i. A Tabela (1) contem os ângulos $\arctan(2^{-i})$ que são armazenados em uma ROM criada no dispositivo.

A unidade de micro-rotação é composta por duas unidades de deslocamento aritmético, três somadores/subtratores, e uma LUT (Tabela 1) que contém os valores $\arctan(2^{-i})$, conforme mostrado na Figura 1. Para determinar se a operação a ser realizada é de soma ou de subtração, considera-se o bit mais significativo do sinal $\theta_i(Z)$.

Tabela 1 – Valores para o arctan(2^{-i})

Iteração	arctan(2^{-i})	Ponto Fixo
0	0.785400390625000	00100000000000000000000000000000
1	0.463653564453125	00010010111001000000010100011101
2	0.244979858398438	00001001111110110011100001011011
3	0.124359130859375	00000101000100010001000111010100
4	0.062423706054688	00000010100010110000110101000011
5	0.031234741210938	00000001010001011101011111100001
6	0.015625000000000	00000000101000101111011000011110
7	0.007812500000000	00000000010100010111110001010101
8	0.003906250000000	00000000001010001011111001010011
...	...	...

Fonte: Desenvolvido pelo autor (2019).

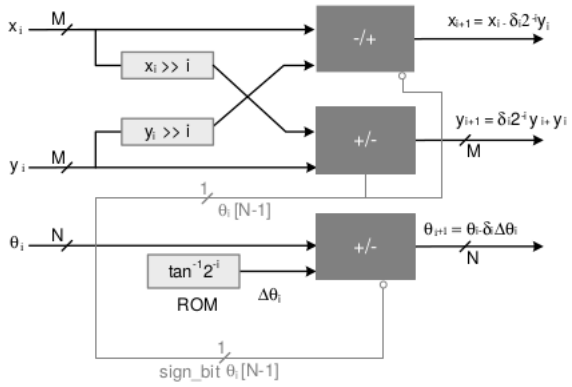


Figura 1 – Unidade de micro-rotação
Fonte: Khan (2011).

3.3 Arquitetura paralela desenvolvida

O CORDIC opera em dois modos: paralelo e *pipelined*. No modo paralelo, todas as operações de rotações são realizadas em apenas um ciclo de *clock*. O algoritmo CORDIC separa em etapas cada iteração. Cada etapa é composta dos mesmos componentes: três somadores/subtratores, duas unidades de deslocamento aritmético e uma LUT. Portanto, a saída de uma etapa corresponde à entrada da etapa seguinte. O diagrama da arquitetura paralela é apresentado na Figura 2.

A arquitetura paralela, mostrada na Figura 2, resulta em duas vantagens: As unidades de deslocamento e as constantes correspondente ao arctan(2^{-i}) podem ser interconectadas por trilhas, sem a necessidade de registradores adicionais; O circuito é constituído de circuito combinacional tornando a arquitetura mais simples. A principal desvantagem é a grande quantidade de área que é necessária para sua implementação e isto implica em um maior consumo energético. A arquitetura paralela é facilmente

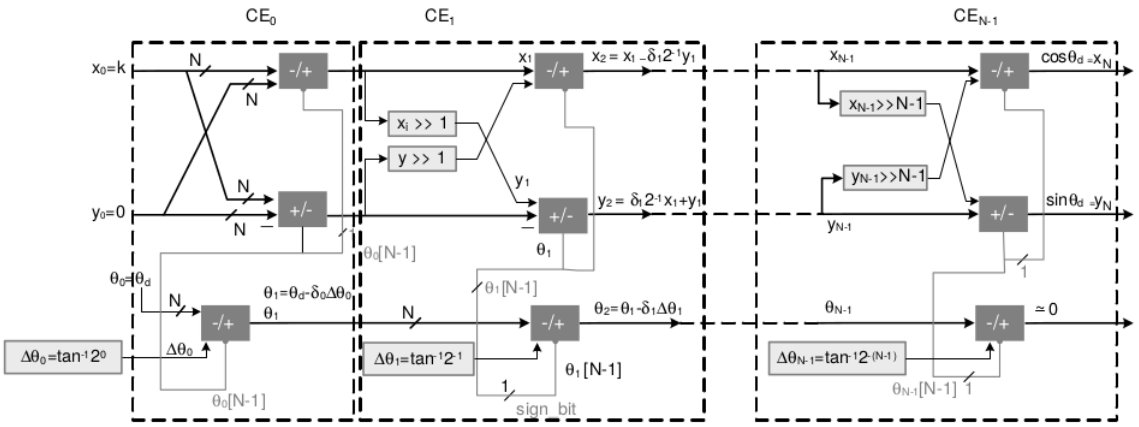


Figura 2 – Arquitetura paralela para o CORDIC
Fonte: Khan (2011).

convertida em uma arquitetura *pipelined* inserindo registradores em cada etapa de iteração. Como na maioria dos FPGAs já existem registradores presentes em cada célula lógica, assim a adição de registradores *pipelined* não tem custo adicional de *hardware* (ANDRAKA, 1998). O RTL (*Register-Transfer Level*) da arquitetura paralela desenvolvida é apresentado na Figura 3.

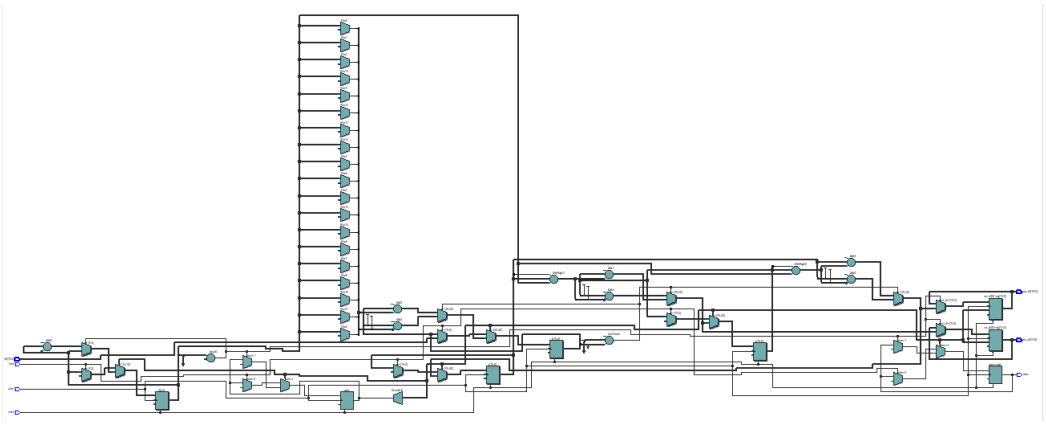


Figura 3 – RTL da arquitetura desenvolvida
Fonte: Desenvolvido pelo autor (2019).

4 IMPLEMENTAÇÃO NO FPGA

Esta seção apresenta a implementação da arquitetura proposta no FPGA. Utilizando o *software* Quartus Prime, foi realizado a síntese do design em Verilog. As características da síntese para o FPGA MAX10 estão presentes na Tabela (2).

Tabela 2 – Saída de dados referente a síntese do design do *software* Intel Quartus

Propriedades	**Valores**
Flow Status	Successful - Mon Apr 8 21:10:58 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	cordic
Family	MAX 10
Device	10M50DAF484C7G
Total logic elements	319 / 49,760 ($<$ 1 %)
Total combinational functions	318 / 49,760 ($<$ 1 %)
Dedicated logic registers	102 / 49,760 ($<$ 1 %)

Fonte: Desenvolvido pelo autor (2019).

Utilizou-se o *software* ModelSim para realizar a simulação do código do CORDIC. Para realizar o *testbench*, ângulos em um intervalo de 0° a 360° foram utilizados. O resultado do *testbench* é apresentado na Figura 4, que em destaque apresenta o cálculo do seno e cosseno para o ângulo de 90°.

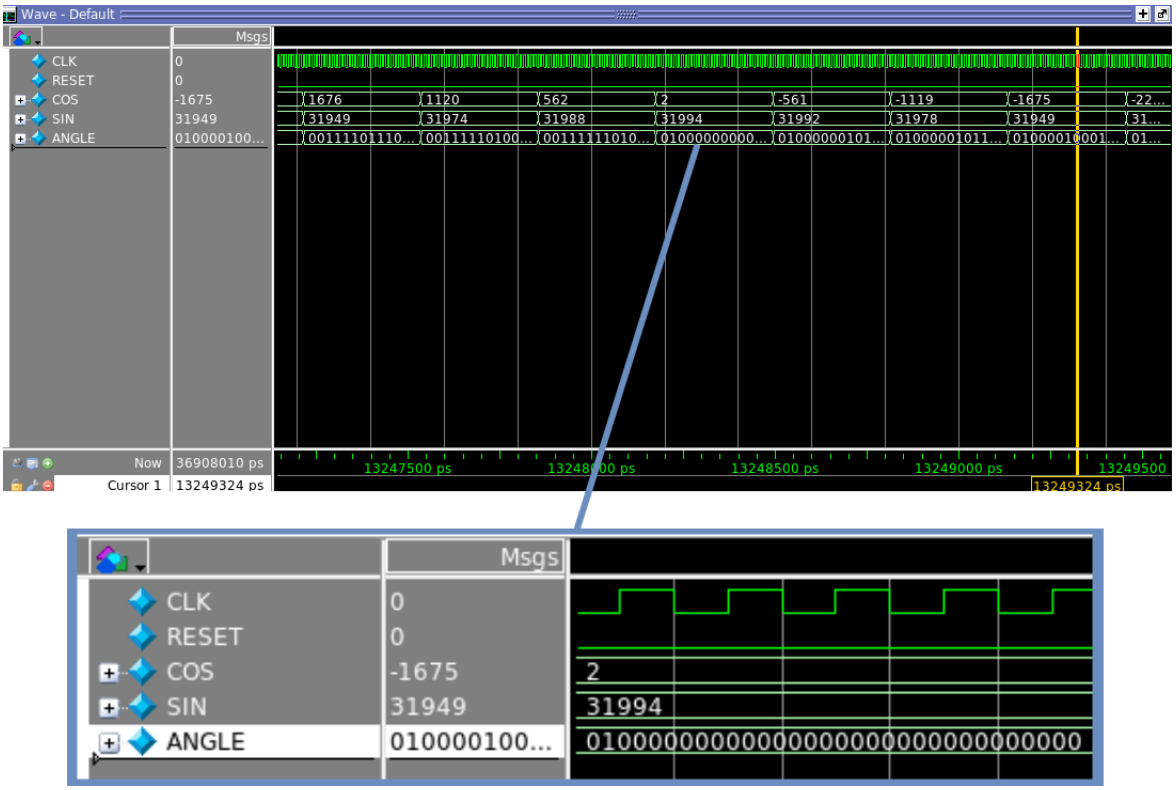


Figura 4 – Waveforms para o test bench
Fonte: Desenvolvido pelo autor (2019).

Pode-se observar que os valores na figura em destaque não são os valores reais de seno e cosseno de 90 °. Para obter os valores reais, basta dividir o valor do sinal 'SIN' e 'COS' da simulação por 32000. O valor 32000 é simplesmente uma escala utilizada para facilitar a implementação do algoritmo CORDIC no FPGA. Sendo assim, teremos o valor 1 e 0 para o seno e cosseno de 90 ° respectivamente.

O sinal 'ANGLE' representa o ângulo de entrada, ele foi representado utilizando 32 bits. O cálculo utilizado para representar o valor do sinal 'ANGLE' é apresentado na Equação 14.

$$Angle = \frac{90^{\circ}}{360^{\circ}} * 2^{32} \tag{14}$$

Como nem o FPGA MAX10, nem a placa de desenvolvimento DE10-Lite possui DAC, foi implementado um conversor de sinal analógico para digital (DAC) de 19 bits então. Para realizar os testes, o algoritmo CORDIC foi sintetizado no FPGA e seus GPIOs foram conectados no DAC implementado. Os GPIOs utilizados são apresentados na Figura 5 em vermelho. Para analisar a saída do DAC, foi utilizado um osciloscópio. A Figura 6 apresenta a bancada onde foram realizados os testes e a senoide produzida. A frequência de operação do MAX10 é de 50Mhz.

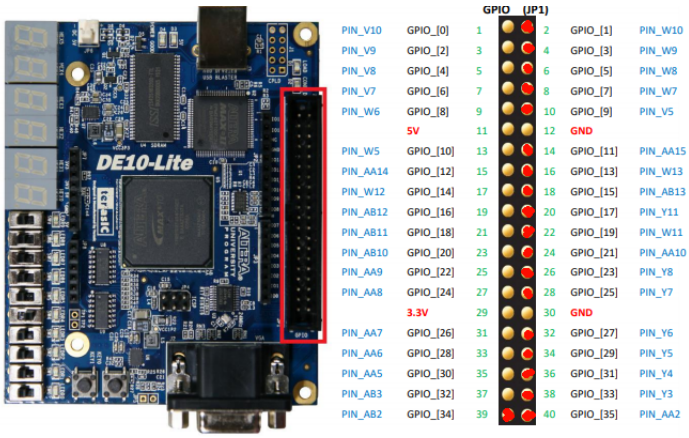
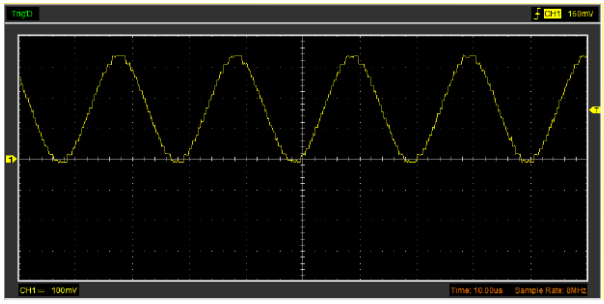
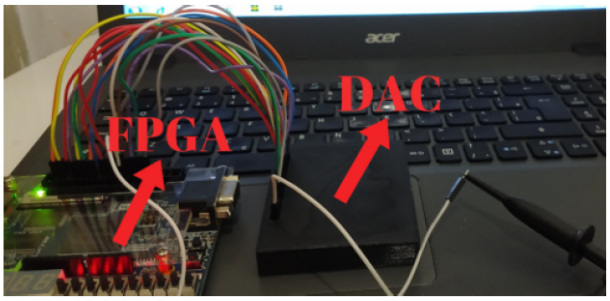


Figura 5 – GPIOs utilizados
Fonte: (De10-lite. . . ,).



(a) Senoide produzida pelo CORDIC no FPGA.



(b) Bancada de testes.

Figura 6 – Implementação na FPGA
Fonte: Desenvolvido pelo autor (2019).

4.1 Implementação em PYTHON

Esta seção apresenta a proposta de implementação do CORDIC em um computador. O computador utilizado foi um Dell Inspiron 15-7559 com um porcessador Intel(R) Core(TM) i7-600HQ CPU @ 2.60GHz, 16GB de memória RAM e Windows 10 (compilação 17134.885). O interpretador utilizado foi o Python 3.7.4. O código do CORDIC foi implementado em Python de maneira que gere um arquivo

texto com os valores de seno e cosseno para que possa ser observada a precisão numérica em relação aos valores produzidos pelo FPGA.

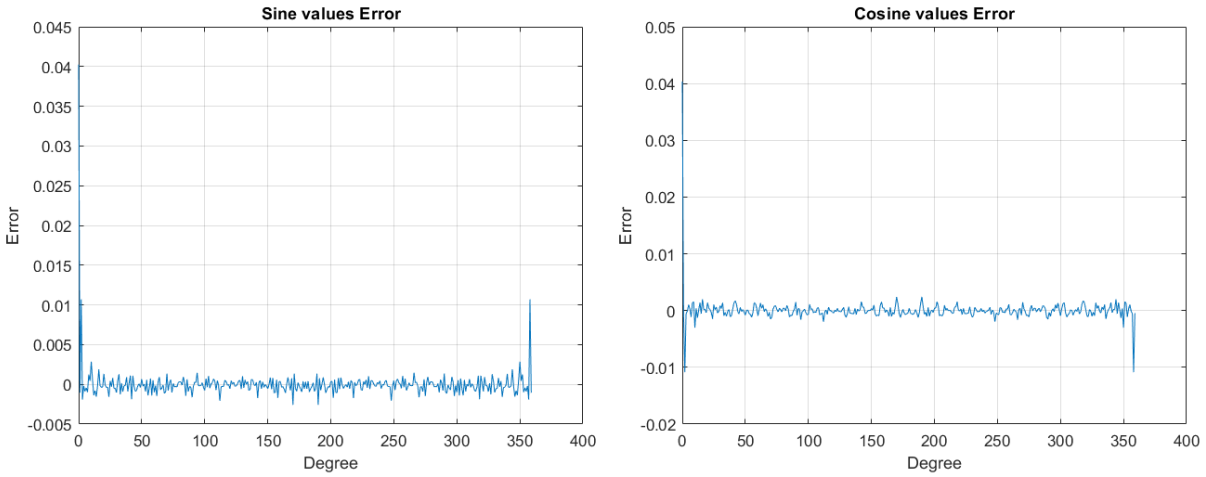
5 RESULTADOS E DISCUSSÕES

Esta seção apresenta os resultados da implementação no FPGA e análise de desempenho da arquitetura proposta. O desempenho da implementação do CORDIC depende unicamente de somadores utilizados, diferentemente das CPUs que tem outros recursos dedicados em *hardware* para este fim. Além disso, o objetivo deste trabalho é apresentar a precisão numérica de cada arquitetura e não considera o tempo de execução para os resultados obtidos.

A fim de avaliar o desempenho da arquitetura proposta, o algoritmo CORDIC foi implementado em Python e a partir de seus resultados foi possível calcular o erro com base na implementação feita em *hardware* no FPGA. O cálculo do erro foi baseado na Equação 15.

$$E = |CORDIC_{Python} - CORDIC_{FPGA}| \tag{15}$$

A fidelidade dos resultados do CORDIC implementado em FPGA quando comparado com os resultados obtidos em Python, como explicado anteriormente, é confirmada quando observa-se o resultado do erro obtido através da Equação 15, filtrados pela transformada rápida de Fourrier, nas Figuras 7a e 7b, referentes respectivamente ao erro do seno e cosseno.



(a) Erro dos valores da função seno entre os dados obtidos do FPGA e Python.

(b) Erro dos valores da função cosseno entre os dados obtidos do FPGA e Python.

Figura 7 – Erro seno e cosseno
Fonte: Desenvolvido pelo autor (2019).

Estes resultados mostram um erro extremamente pequeno entre as duas abordagens, com pico no cálculo do seno e cosseno de 0 e 360 graus, no entanto o erro pode ser desconsiderado devido a sua magnitude. Esses resultados confirmam que o CORDIC é uma ótima opção para o cálculo de funções trigonométricas em *hardwares* sem um multiplicador.

Pode-se observar que as curvas plotadas são extremamente similares representado pelas Figuras 8 (seno) e 9 (cosseno). Isso mostra que o algoritmo CORDIC reproduz os resultados das funções de maneira fiel quando implementado em *hardware* em comparação a sua implementação no computador utilizado. No entanto, vale ressaltar que o CORDIC foi desenvolvido para *hardwares* nos quais não há um multiplicador físico implementado.

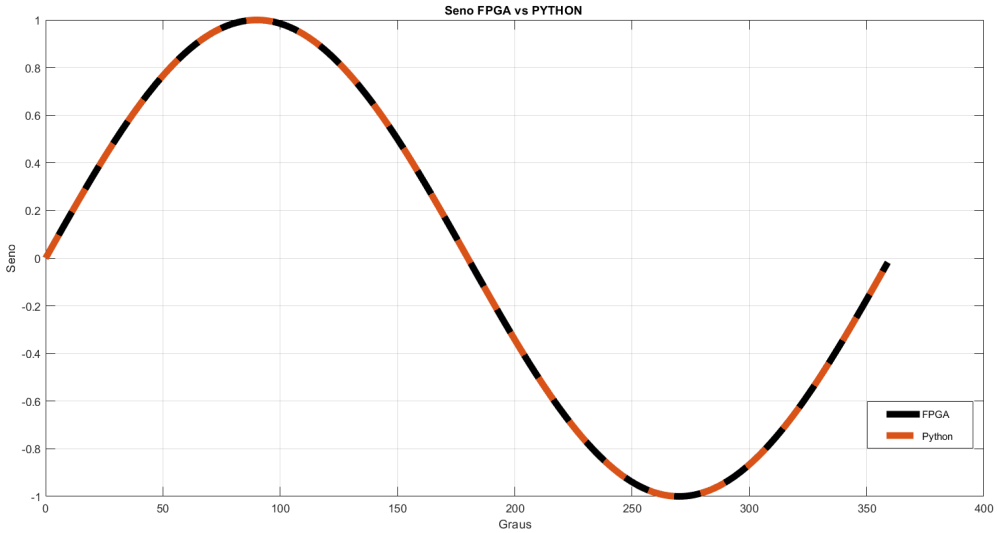


Figura 8 – Gráfico da função seno plotada com os valores obtidos com Python e o FPGA
Fonte: Desenvolvido pelo autor (2019).

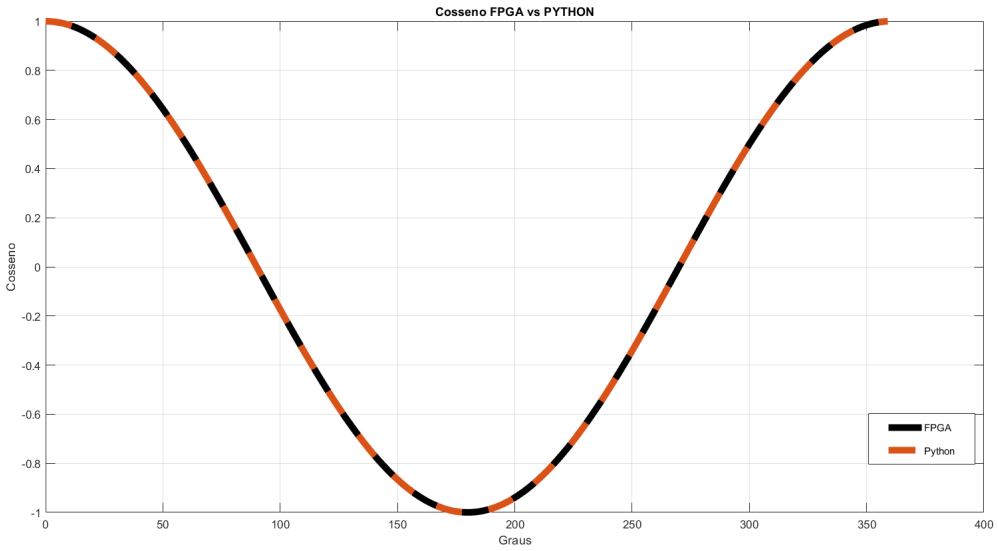


Figura 9 – Gráfico da função cosseno plotada com os valores obtidos com Python e o FPGA
Fonte: Desenvolvido pelo autor (2019).

6 CONCLUSÃO

Nota-se que o CORDIC implementado em Verilog FPGA obteve ótimos resultados. Também é notável a quantidade de memória que a implementação ocupou no FPGA. Isso se deve ao fato de ser utilizada uma arquitetura paralela, na qual a iteração principal é repetida por um determinado número de vezes. Portanto, a arquitetura apresentada nesse trabalho tem a vantagem de ser simples de se implementar, pois não há a necessidade de criar uma unidade de controle. O algoritmo CORDIC torna-se uma ótima opção para ser utilizada quando houver a necessidade de empregar o uso de funções trigonométricas seno e cosseno em design digital.

REFERÊNCIAS

ANDRAKA, R. A survey of cordic algorithms for fpga based computers. p. 191–200, 1998.

BISWAL, P.; BANERJEE, S. Implementation of katsevich algorithm for helical cone-beam computed tomography using cordic. In: **2010 International Conference on Systems in Medicine and Biology**. [S.l.: s.n.], 2010. p. 313–317.

De10-lite user manual. ⟨https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-2912030810549-de10-lite-user-manual.pdf⟩. Accessed: 2019-07-17.

HAUCK, A. D. S. **Reconfigurable computing the theory and practice of FPGA-based computation/edited**. [S.l.]: Elsevier Inc, 2008.

KHAN, S. A. **Digital design of signal processing systems : a practical approach**. 1. ed. [S.l.]: John Wiley & Sons, 2011. 579 p.

MASRAM, B. Y.; KARULE, P. T. High performance analysis of a cordic architectures based on fpga: A comparative approach. In: **2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies**. [S.l.: s.n.], 2014. p. 569–574.

MEHER, P. K. et al. 50 Years of CORDIC: Algorithms, Architectures and Applications. **IEEE Transactions on Circuits & Systems-I: Regular Papers**, v. 56, p. 1893–1907, 2009.

MULLER, J.-M. **Elementary Functions: Algorithms and Implementation**. 2. ed. [S.l.]: Boston: Birkhäuser, 1961. 134 p. ISBN 9780817643720.

SWARTZLANDER JR., E. E. **Computer Arithmetic. 1 (2 ed.)**. [S.l.]: Los Alamitos: IEEE Computer Society Press, 1990.

TANG, A. et al. Cordic-based fft real-time processing design and fpga implementation. In: **2016 IEEE 12th International Colloquium on Signal Processing Its Applications (CSPA)**. [S.l.: s.n.], 2016. p. 233–236.

VOLDER, J. E. The CORDIC Computing Technique. **San Francisco, California, USA: National Joint Computer Committee (NJCC)**, v. 1, p. 257–261, 1959.

____. The CORDIC Trigonometric Computing Technique. **IRE Transactions on Electronic Computers**, v. 8, p. 330–334, 1959.

DADOS DOS AUTORES

Ricardo Gonçalves de Aguiar

Lattes: http://lattes.cnpq.br/6096646659838022

E-mail: rigoaguia@gmail.com

Graduando em Engenharia da Computação pela Universidade Federal de Mato Grosso.

Valfride Wallace do Nascimento

Lattes: http://lattes.cnpq.br/4643088501691412

E-mail: valfridewallace@gmail.com

Graduando em Engenharia da Computação pela Universidade Federal de Mato Grosso.

Fernando Lessa de Oliveira Magalhães

Lattes: http://lattes.cnpq.br/4124771908354942

E-mail: nando.lessa@gmail.com

Graduando em Engenharia de Controle e Automação pela Universidade Federal de Mato Grosso.

Hugo Daniel Hernandez Herrera

Lattes: http://lattes.cnpq.br/8883830935165755

E-mail: hugoelectro83@gmail.com

Hugo gradou-se em Engenharia Eletrônica pela Universidade industrial Santander (Colômbia), possui Mestrado em Engenharia Elétrica (2008) e doutorado em Microeletrônica (2015), os dois pela Universidade de São Paulo. Tem sete (7) anos de experiencia na industria como projetista de circuitos integrados analógicos e mixed-signal, trabalhando na Design House do LSITEC e na empresa DFCHIP. Foi professor de dedicação exclusiva do curso de Controle e Automação no Instituto de Engenharia (CUVG) da UFMT por 2 anos. Atualmente é professor Adjunto do Departamento de Engenharia Elétrica da Universidade Federal de Minas Gerais (UFMG).Tem experiência na área de Engenharia Elétrica, com ênfase em Circuitos Eletrônicos, atuando principalmente nos seguintes temas: Microeletrônica, Circuitos Mixed Signal, CADs para Microeletrônica, Circuitos Eletrônicos e Projeto de Circuitos Integrados.