UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE ENGENHARIAS
CURSO DE ENGENHARIA ELÉTRICA

Leonardo Guimarães Braga

Desenvolvimento de um Processador CORDIC para Leitura de Informação Quântica

CARAÚBAS
2025

Leonardo Guimarães Braga

Desenvolvimento de um Processador CORDIC para Leitura de Informação Quântica

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Francisco de Assis Brito Filho, Prof. Dr.

CARAÚBAS

2025

© Todos os direitos estão reservados a Universidade Federal Rural do Semi-Árido. O conteúdo desta obra é de inteira responsabilidade do (a) autor (a), sendo o mesmo, passível de sanções administrativas ou penais, caso sejam infringidas as leis que regulamentam a Propriedade Intelectual, respectivamente, Patentes: Lei n° 9.279/1996 e Direitos Autorais: Lei n° 9.610/1998. O conteúdo desta obra tomar-se-á de domínio público após a data de defesa e homologação da sua respectiva ata. A mesma poderá servir de base literária para novas pesquisas, desde que a obra e seu (a) respectivo (a) autor (a) sejam devidamente citados e mencionados os seus créditos bibliográficos.

B813d Braga, Leonardo Guimarães .
Desenvolvimento de um processador CORDIC para leitura de informação quântica / Leonardo Guimarães Braga. - 2025.
62 f. : il.

Orientador: Francisco de Assis Brito Filho.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Engenharia Elétrica, 2025.

1. CORDIC . 2. Digital. 3. ASIC. 4. Tapeout. I. Brito Filho, Francisco de Assis, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Caraúbas
Bibliotecária: Sarah Freire Bezerra
CRB: 15/1052

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

Dedico este trabalho aos meus pais, Euzirene Guimarães Alves Braga e Aldenizio Braga dos Santos, em memória de meus avós paternos, Nazaré e Manoel Braga, e a todos os meus amigos e familiares. Por toda ajuda, carinho, incentivo e por não deixarem de acreditar em mim.

AGRADECIMENTOS

Primeiramente agradeço aos meus pais Euzirene Guimarães e Aldenizio Braga que sempre estiveram ao meu lado, e sempre fizeram o possível e o impossível para me ajudar e nunca me deixar faltar nada. Agradeço ao meu irmão Daniel Guimarães, que sempre me alegrou em momentos que precisava, à minha namorada Carolayne Linhares que nunca soltou minha mão nos momentos difíceis. Aos meus familiares que me apoiaram e me incentivaram, moralmente, até os dias de hoje, e mesmo os que já partiram no meio dessa jornada.

Não me esqueceria dos meus amigos e colegas de curso que foram de grande apoio durante os momentos de dificuldade do curso, tanto para me ajudar com os estudos como para me fazer rir. Adriel, Rafael, Antônio Paulo, Saulo Gabryel, Franarly, Adriano, Carlos Henrique, Caelison, Randres, João Vitor, Hugo, Marcello, Joilson, Nadson e Gustavo um obrigado para vocês e muitos outros que estiveram comigo até aqui. Agradeço também a todos do corpo docente do curso de engenharia elétrica que me ensinaram, e principalmente, ao meu orientador Francisco de Assis Brito Filho, por sua ajuda com este trabalho, me fornecendo orientação e aprendizado profissional. Sem ele não teria sido possível, muito obrigado.

Não se esqueça de sorrir em qualquer situação. Contanto que você esteja vivo, haverá coisas boas mais pra frente, e serão muitas.

- Eiichiro Oda.

RESUMO

Este trabalho apresenta o desenvolvimento e verificação de um processador digital baseado no algoritmo CORDIC, implementado em Verilog e direcionado para integração ao fluxo de fabricação educacional *TinyTapeout*. O projeto tem como objetivo realizar a conversão de sinais cartesianos (componentes *In-phase e Quadrature* — I/Q) em suas representações polares de magnitude e fase, utilizando aritmética em ponto fixo e uma arquitetura totalmente síncrona. A implementação adotou o modo vetorial do CORDIC, que permite calcular diretamente a magnitude e o ângulo do vetor de entrada, por meio de rotações iterativas sucessivas aplicadas de acordo com o sinal da componente y. A arquitetura foi equipada com pipeline completo, tabela de arctangentes pré-quantizada e compensação implícita do ganho inerente ao método, garantindo maior estabilidade numérica, ampla cobertura angular dos quatro quadrantes e latência fixa por processamento. Para avaliar o desempenho e a precisão do bloco, foi realizada também uma implementação em modo rotacional, utilizada como referência para comparação estrutural e funcional. Ambos os módulos foram extensivamente verificados em ambiente *open-source*: simulação com *Icarus Verilog*, análise temporal com *GTKWave* e validação de estimativas contra um modelo matemático em ponto flutuante. A análise comparativa demonstrou que o modo vetorial é mais adequado para aplicações que envolvem processamento I/Q e medidas de amplitude e fase, enquanto o modo rotacional apresenta desempenho mais eficiente para geração direta de seno e cosseno. Por fim, desenvolveu-se um *wrapper* de interface com multiplexação de entradas e saídas, possibilitando que o CORDIC vetorial opere com apenas 8 pinos de entrada e 8 pinos de saída, atendendo às restrições de *FPGAs* educacionais e ao padrão *TinyTapeout*, sem prejuízo à precisão do cálculo. Os resultados obtidos comprovam que o processador CORDIC projetado apresenta baixo custo em hardware, alta reprodutibilidade e desempenho compatível com aplicações reais em sistemas de comunicação e conversores I/Q para radiofrequência.

Palavras-chave: CORDIC; Sistemas I/Q; Verilog; TinyTapeout; ASIC; Heteródino.

ABSTRACT

This work presents the development and verification of a digital processor based on the CORDIC algorithm, implemented in Verilog and aimed at integration into the TinyTapeout educational fabrication workflow. The project aims to convert Cartesian signals (In-phase and Quadrature components — I/Q) into their polar magnitude and phase representations, using fixed-point arithmetic and a fully synchronous architecture. The implementation adopted the vector mode of CORDIC, which allows directly calculating the magnitude and angle of the input vector, through successive iterative rotations applied according to the signal of the y component. The architecture was equipped with a complete pipeline, a pre-quantized arctangent table, and implicit compensation for the gain inherent to the method, ensuring greater numerical stability, wide angular coverage of the four quadrants, and fixed latency per processing. To evaluate the performance and accuracy of the block, a rotational mode implementation was also carried out, used as a reference for structural and functional comparison. Both modules were extensively verified in an open-source environment: simulation with Icarus Verilog, temporal analysis with GTKWave, and validation of estimates against a floating-point mathematical model. The comparative analysis demonstrated that the vector mode is more suitable for applications involving I/Q processing and amplitude and phase measurements, while the rotational mode presents more efficient performance for direct sine and cosine generation. Finally, an interface wrapper with input and output multiplexing was developed, enabling the vector CORDIC to operate with only 8 input pins and 8 output pins, meeting the restrictions of educational FPGAs and the TinyTapeout standard, without compromising calculation accuracy. The results obtained prove that the designed CORDIC processor has low hardware cost, high reproducibility, and performance compatible with real-world applications in communication systems and radio frequency I/Q converters.

Keywords: CORDIC; I/Q systems; Verilog; TinyTapeout; ASIC; Heterodyne.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CORDIC	*Coordinate Rotation Digital Computer*
ASIC	*Application-Specific Integrated Circuits*
CTS	*Clock Tree Synthesis*
NISQ	*Noisy Intermediate-Scale Quantum*
QSP	*Quantum Signal Processing*
QSVT	*Quantum Singular Value Transformation*
VQE	*Variational Quantum Eigensolver*
QAOA	*Quantum Approximate Optimization Algorithm*
QV	*Quantum Volume*
RDP	Sociedade Brasileira de Gestão do Conhecimento
SoC	*System On-Chip*
OpenROAD	*Open Routing, Placement and Optimization for Analog and Digital circuits*

SUMÁRIO

1 INTRODUÇÃO

O CORDIC (*Coordinate Rotation Digital Computer*) consolidou-se como uma técnica de baixo custo para calcular funções trigonométricas e afins por meio de microrrotações iterativas que utilizam basicamente somas e shifts, o que o torna especialmente atraente em *ASICs* e *FPGAs*. Nos últimos anos, linhas de pesquisa têm buscado reduzir latência e otimizar a escalação (*scaling*) por meio de pré-rotação, escolha eficiente de ângulos e arquiteturas pipeline, gerando ganhos práticos em funções como *arctan*, DDS e conversões polar-cartesiano (DAI et al., 2024; HWANG; KIM; LEE, 2021; TANG et al., 2022).

Essas melhorias têm impacto direto em processamento de sinais e comunicações. FFTs de alto desempenho substituem multiplicações complexas por rotações CORDIC, geradores de desvanecimento (*fading*) múltiplo aceleram cálculos trigonométricos com variantes otimizadas, e arquiteturas de rádios definidos por software exploram pipelines CORDIC para operar em altas taxas com baixo consumo de área e energia (GUPTA et al., 2019; LI et al., 2021; CHEN et al., 2020).

Em paralelo, a computação quântica entrou na era NISQ (*Noisy Intermediate-Scale Quantum*), marcada por dispositivos ruidosos de média escala capazes de executar circuitos úteis, mas limitados em profundidade. Essa fase definiu tanto expectativas realistas quanto a necessidade de algoritmos de baixa profundidade e maior robustez para aplicações práticas, sendo notável o marco da "supremacia quântica" reportada pelo *Google* em 2019 (PRESKILL, 2018; ARUTE et al., 2019).

No cerne de muitos algoritmos quânticos modernos está a síntese eficiente de sequências de rotações de um *qubit*. Estruturas como *Quantum Signal Processing* (QSP) e sua generalização *Quantum Singular Value Transformation* (QSVT) demonstram que polinômios e funções podem ser implementados por composições cuidadosamente projetadas de rotações de fase, um conceito que dialoga com a ideia clássica do CORDIC de aproximar ângulos por microrrotações elementares (LOW; CHUANG, 2017; GILYÉN et al., 2019; MARTYN et al., 2021).

Mais recentemente, essa analogia deixou de ser apenas conceitual: foi proposta uma versão de *Quantum CORDIC*, que adapta o esquema de microrrotações para computar funções como *arcsin* de forma reversível em circuitos quânticos. Esse avanço analisa contagens de portas e *qubits* e aponta aplicações em sub-rotinas como HHL, conversão D/A quântica e

estimativa de valores, evidenciando um caminho direto para transplantar ideias do CORDIC clássico ao domínio quântico (MELNIKOV; KHARIN; FEDOROV, 2022).

Com isso, abre-se um espaço fértil de pesquisa aplicada: usar a intuição de decomposição angular do CORDIC para arquitetar blocos quânticos de rotação e aproximação funcional compatíveis com NISQ, QSP e QSVT, mirando sub-rotinas essenciais como *amplitude estimation* em variantes de baixa profundidade (sem QFT). Tais métodos vêm sendo refinados desde 2019 e representam um fio condutor comum: construir funções úteis a partir de composições de rotações simples, otimizando profundidade e erro de aproximação (SUZUKI et al., 2020; GRIBBEN et al., 2023).

1.1 Objetivos

Neste tópico serão abordados os objetivos, geral e específico, relacionados ao trabalho proposto.

1.1.1 Objetivos Gerais

O objetivo geral deste projeto é desenvolver e validar uma arquitetura baseada no algoritmo CORDIC voltada à conversão de sinais *In-phase* (I) e *Quadrature* (Q) em fase e amplitude. Além disso, pretende-se explorar e comparar os modos de operação vetorial e rotacional, analisando suas vantagens e limitações em diferentes cenários de processamento de sinais. Complementarmente, busca-se implementar um *wrapper* para integração do módulo em um fluxo completo de projeto de circuito integrado, desde a descrição em RTL até a geração do GDSII para *tapeout*, utilizando ferramentas *open source*.

1.1.2 Objetivos Específicos

• Estudar os fundamentos teóricos e práticos do algoritmo CORDIC, abordando de forma aprofundada seus modos de operação vetorial e rotacional;
• Desenvolver em Verilog os módulos CORDIC vetorial e rotacional, realizando simulações funcionais para validar sua correta implementação;
• Realizar testes comparativos entre os modos vetorial e rotacional, avaliando precisão, latência, ocupação de área e consumo de potência em diferentes condições de entrada;
• Projetar um *wrapper* de multiplexação para reduzir a quantidade de pinos de entrada e saída, adaptando os módulos ao *template* de pinagem exigido para *tapeout*;

• Implementar e verificar o fluxo de síntese, *floorplanning, placement, routing* e verificação física utilizando ferramentas open source como Yosys, OpenROAD, OpenLane e KLayout;
• Comparar os resultados de simulação funcional e *pós-layout*, estabelecendo métricas de desempenho entre o CORDIC vetorial e o rotacional;
• Gerar os arquivos finais no formato GDSII, garantindo que o projeto esteja adequado para submissão em um processo de fabricação *open source*.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta os fundamentos teóricos essenciais para a compreensão do desenvolvimento e implementação de um processador digital baseado no algoritmo CORDIC, aplicado à conversão de sinais I/Q em magnitude e fase. Inicialmente, discute-se o conceito de sinais em quadratura, amplamente utilizados em arquiteturas de comunicação modernas, onde os componentes In-phase (I) e Quadrature (Q) representam a informação do sinal no domínio cartesiano. Em seguida, aborda-se a necessidade de transformar essas coordenadas cartesianas em coordenadas polares, visando a extração da amplitude e da fase do sinal, parâmetros fundamentais para demodulação, análise espectral e detecção de informações em sistemas de rádio definidos por software (SDR).

Na continuidade, apresenta-se o algoritmo CORDIC, criado com o objetivo de realizar operações trigonométricas e hiperbólicas utilizando apenas deslocamentos binários e somas, eliminando a dependência de multiplicadores e favorecendo arquiteturas eficientes em hardware. São discutidos seus dois modos principais de operação: rotacional, empregado para o cálculo de seno e cosseno a partir de um ângulo conhecido, e vetorial, utilizado quando se deseja determinar magnitude e ângulo a partir de componentes I/Q. As diferenças estruturais, implicações matemáticas e aplicação adequada de cada modo também são exploradas. Complementarmente, são introduzidos conceitos de aritmética em ponto fixo, ressaltando importância, limitações e impactos na precisão numérica para aplicações em hardware. Destaca-se a utilização de tabelas pré-computadas de arco-tangente e técnicas de compensação do ganho intrínseco do CORDIC, garantindo confiabilidade nos resultados obtidos.

Por fim, apresenta-se o contexto de integração ao fluxo digital TinyTapeout, que permite a fabricação de circuitos integrados utilizando ferramentas e bibliotecas abertas, porém com restrições severas no número de pinos e área disponível. A necessidade de estratégias como multiplexação de entradas e saídas, bem como o encapsulamento do processador em um

wrapper digital reduzido, é discutida, evidenciando o alinhamento do projeto à realidade de implementação física em ASICs de baixo custo.

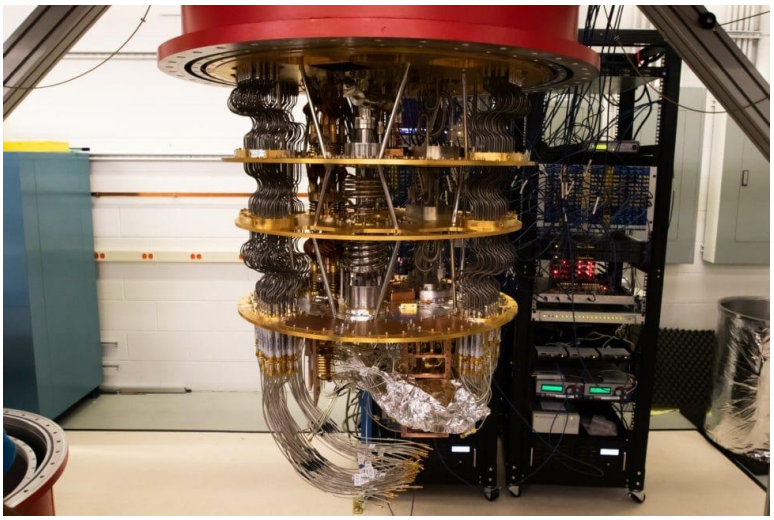
2.1 Computação Quântica

A computação quântica representa um novo paradigma no processamento da informação, fundamentando-se em princípios da mecânica quântica como a superposição, o emaranhamento e a interferência. Diferentemente dos sistemas clássicos, que manipulam bits binários, os sistemas quânticos utilizam *qubits*, que podem assumir simultaneamente os estados 0 e 1, conferindo-lhes um potencial exponencial de paralelismo. Essa característica abre caminho para a resolução de problemas complexos, como a fatoração de números primos, a simulação de materiais quânticos e a otimização combinatória, de forma muito mais eficiente do que os algoritmos clássicos (PRESKILL, 2018).

Atualmente, o campo encontra-se na chamada era NISQ, caracterizada pela existência de dispositivos de pequena e média escala, contendo entre dezenas e poucas centenas de *qubits*, mas ainda fortemente afetados por ruído e decoerência. Esses sistemas, embora limitados, têm permitido experimentos relevantes que aproximam a teoria quântica de aplicações práticas. De acordo com Preskill (2018), essa etapa é um marco de transição entre o domínio teórico e a construção de arquiteturas quânticas tolerantes a falhas.

Um dos marcos mais importantes da área foi a demonstração experimental da chamada supremacia quântica, realizada pela equipe da *Google* em 2019. Utilizando o processador *Sycamore* (Figura 1), composto por 53 *qubits* supercondutores, os pesquisadores conseguiram realizar uma tarefa de amostragem aleatória em poucos minutos, algo que, segundo estimativas, levaria milhares de anos em supercomputadores clássicos (ARUTE et al., 2019). Esse feito evidenciou o potencial das arquiteturas supercondutoras, estabelecendo uma base experimental sólida para novas arquiteturas quânticas de grande escala.

Figura 1: Sycamore, Computador Quântico da Google



Fonte: Giz Brasil (2020).

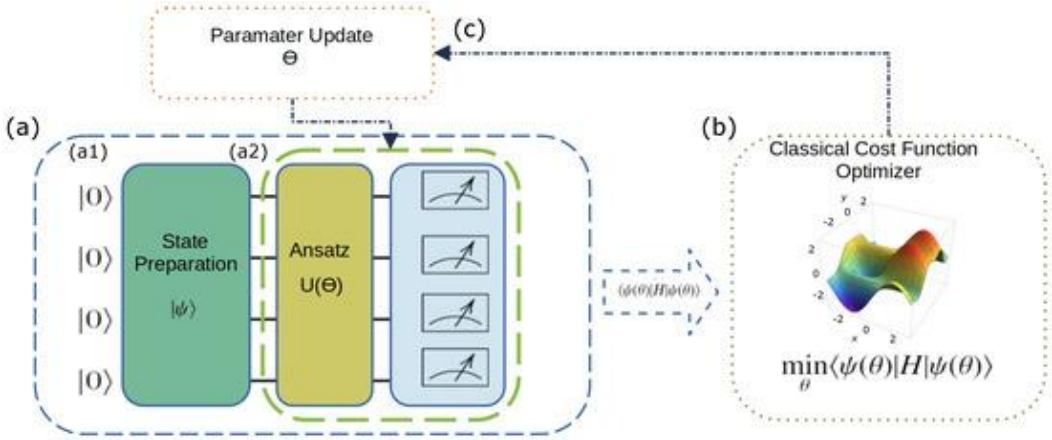
Diversas plataformas físicas vêm sendo exploradas para a implementação de qubits. Entre elas destacam-se os supercondutores, os íons aprisionados, os átomos neutros de *Rydberg* e os fótons. Os dispositivos supercondutores, atualmente os mais maduros, são amplamente utilizados por empresas como IBM e *Google*, devido à sua escalabilidade e compatibilidade com circuitos integrados convencionais. Já os sistemas de íons aprisionados apresentam alta fidelidade de porta e conectividade total, embora apresentem limitações de tempo de operação (KJAERGAARD et al., 2020; BRUZEWICZ et al., 2019; BROWAEYS; LAHAYE, 2020).

Do ponto de vista teórico, a base matemática que sustenta a computação quântica contemporânea foi ampliada por estruturas como o QSP e o QSVT. Essas técnicas demonstraram que funções matemáticas podem ser implementadas de forma eficiente por meio de sequências de rotações quânticas controladas, o que reduz a profundidade dos circuitos e melhora a escalabilidade de algoritmos aplicados à simulação e otimização (LOW; CHUANG, 2017; GILYÉN et al., 2019). Essas formulações são consideradas um elo teórico entre a aproximação funcional clássica e a síntese de circuitos quânticos rasos, o que cria uma ponte direta com algoritmos iterativos baseados em rotações, como o CORDIC.

Paralelamente, a busca por algoritmos híbridos variacionais, como o *Variational Quantum Eigensolver* (VQE) (Figura 2) e o *Quantum Approximate Optimization Algorithm* (QAOA) (Figura 3), tem permitido aproveitar o potencial dos dispositivos NISQ para resolver problemas de química quântica, aprendizado de máquina e otimização. Esses algoritmos combinam processamento clássico e quântico, ajustando parâmetros para minimizar funções de

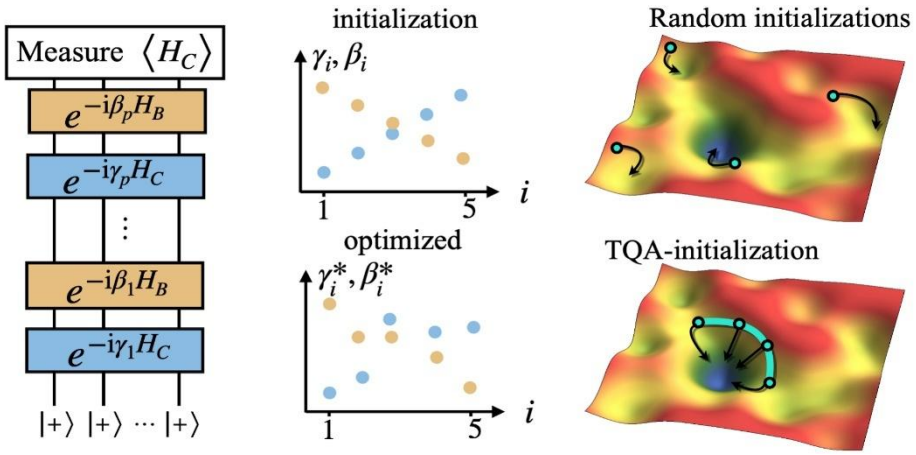
custo, e são compatíveis com dispositivos de baixa profundidade (KANDALA et al., 2017; HADFIELD et al., 2019).

Figura 2:Esquema conceitual do algoritmo Variational Quantum Eigensolver (VQE), mostrando o fluxo híbrido clássico-quântico



Fonte: MDPI (2024).

Figura 3: Quantum Approximate Optimization Algorithm (QAOA)



Fonte: Quantum Jornal (2024).

A avaliação do desempenho de processadores quânticos também evoluiu por meio da introdução de métricas como o *Quantum Volume* (QV), que sintetiza a relação entre fidelidade de portas, coerência e conectividade dos *qubits* em uma única medida de capacidade. Essa métrica, proposta por Cross et al. (2019), tornou-se um padrão experimental amplamente adotado para comparar diferentes arquiteturas.

Outro campo de pesquisa de crescente relevância é o da mitigação e correção de erros quânticos. Métodos como extrapolação a ruído zero (ZNE) e *randomized compiling* têm sido

amplamente estudados e aplicados em experimentos práticos, reduzindo significativamente o impacto do ruído sem exigir correção completa de erros (TEMME; BRAVYI; GAMBETTA, 2017; HASHIM et al., 2021). Em paralelo, experimentos recentes demonstraram códigos de superfície escaláveis, capazes de reduzir erros lógicos à medida que o tamanho do código aumenta, representando um passo decisivo rumo à computação quântica tolerante a falhas (ACHARYA et al., 2023).

Assim, a computação quântica contemporânea se posiciona como um campo interdisciplinar que combina física, matemática e engenharia, e que, ao mesmo tempo, demanda inovações em algoritmos, arquiteturas e metodologias de projeto. Seu avanço contínuo aponta para uma convergência natural com técnicas clássicas de decomposição e aproximação, como o algoritmo CORDIC, que também se baseia em microrrotações elementares. Essa relação conceitual entre rotações quânticas e aproximações trigonométricas clássicas sugere um caminho promissor para o desenvolvimento de arquiteturas híbridas e algoritmos *quantum-inspired*, nos quais o raciocínio geométrico do CORDIC possa ser explorado em implementações quânticas de baixa profundidade.

2.2 O algoritmo CORDIC

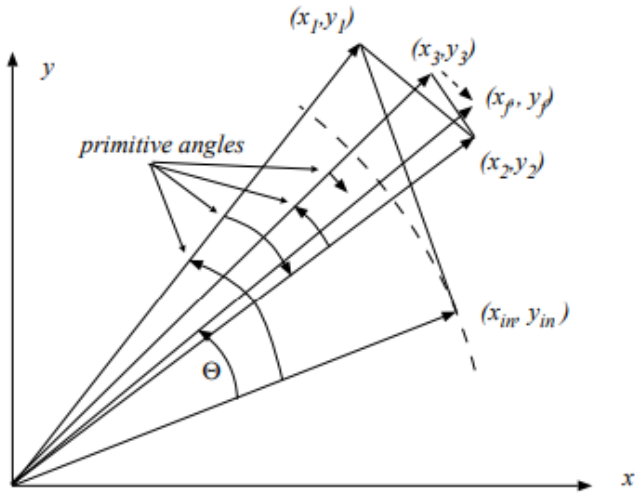
O CORDIC é um algoritmo desenvolvido originalmente por Jack Volder na década de 1950 para facilitar cálculos de navegação aérea. Ele se baseia em decompor uma rotação em uma sequência de microrrotações pré-definidas, utilizando apenas operações de soma, subtração e deslocamento de *bits*, o que elimina a necessidade de multiplicadores. Essa característica o torna altamente atrativo em implementações de hardware, principalmente em sistemas embarcados, processadores digitais de sinais (DSPs) e circuitos integrados de aplicação específica (ASICs) (HWANG; KIM; LEE, 2021).

Nos últimos anos, variantes e otimizações do CORDIC vêm sendo propostas com o objetivo de reduzir a latência, aumentar a precisão e melhorar o aproveitamento de área e energia em plataformas reconfiguráveis. Métodos como pré-rotação, previsão da direção de microrrotações e arquitetura pipeline têm mostrado resultados significativos em aplicações de comunicações digitais, processamento de imagem e geração de sinais (DAI et al., 2024; TANG et al., 2022).

2.2.1 CORDIC Rotacional

O modo rotacional do algoritmo CORDIC é responsável por rotacionar um vetor inicial (x_0, y_0) por um ângulo θ através de uma sequência de microrrotações elementares (Figura 4). Em cada iteração i, define-se o sentido da rotação, $d_i \in \{+1, -1\}$, somando-se ou subtraindo-se o ângulo $\arctan(2^{-i})$. As novas coordenadas são obtidas apenas por operações de soma, subtração e deslocamento de bits, eliminando multiplicadores e tornando o método extremamente eficiente em hardware digital (Hwang; Kim; Lee, 2021; Chen et al., 2020).

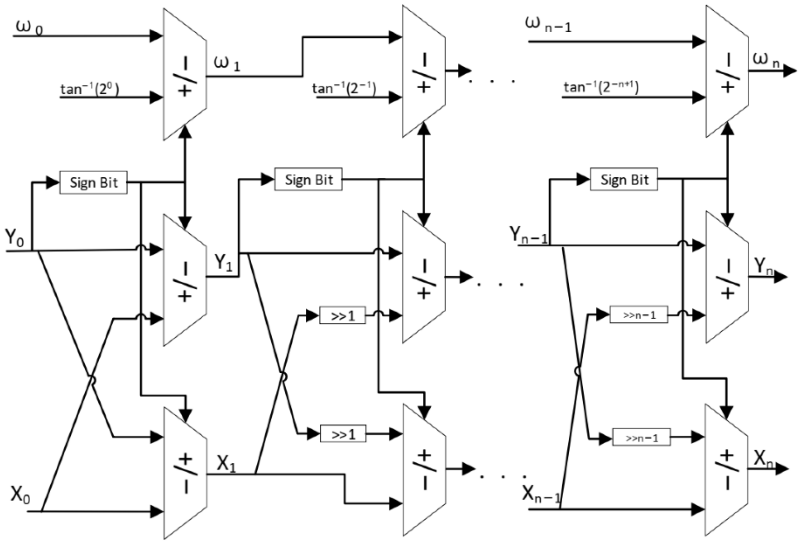
Figura 4: Rotação de Vetores Usando Microrrotações



Fonte: Digital Arithmetic - Ercegovac/Lang (2003).

O desempenho do modo rotacional depende do fator de escala $K_N = \prod_{i=0}^{N-1} \sqrt{1+2^{-2i}}$, resultado da composição das microrrotações. Esse fator pode ser compensado ao final ou distribuído entre as iterações, dependendo da arquitetura. A convergência do algoritmo ocorre no intervalo $[-\pi/2, \pi/2]$, sendo necessária uma etapa de pré-rotação para mapear o ângulo de entrada a essa faixa. Essa técnica também reduz o número de iterações e a latência global, viabilizando arquiteturas *pipeline* de alta taxa de transferência (Figura 5) (Li et al., 2024).

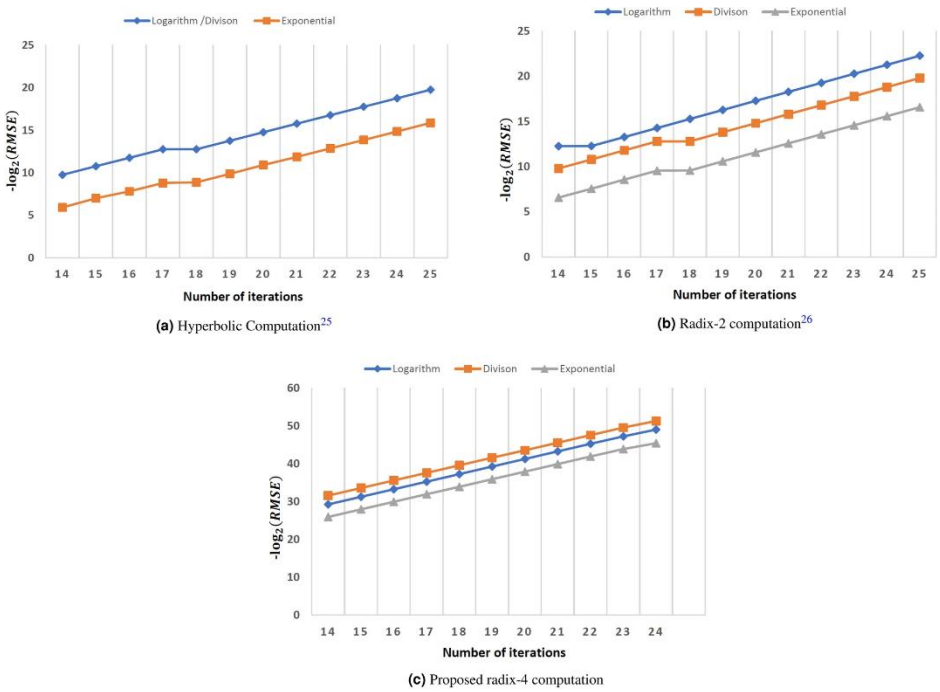
Figura 5: Fluxo de dados do CORDIC rotacional com pré-rotação e pipeline de alta taxa de transferência



Fonte: Li et al. (2024).

Nos últimos anos, diversas otimizações têm sido propostas para o modo rotacional. Entre elas, destacam-se: a pré-rotação dedicada, que antecipa a escolha da direção de microrrotação; o RDP-CORDIC (*Rotation Direction Prediction*), que pré-calcula todas as direções e reduz o número de iterações (Qin et al., 2022). Os métodos *scale-free*, que eliminam a necessidade de aplicar explicitamente o fator K_n , e os algoritmos de *radix* elevado, como o *radix-4* (Figura 6), que substituem múltiplas iterações por etapas de maior complexidade, reduzindo a latência total (Changela; Trivedi; Modi, 2023). Comparações recentes entre o CORDIC e abordagens alternativas, como cálculos diretos de normas em ponto fixo, confirmam que o CORDIC mantém vantagens de área e energia em aplicações de hardware embarcado (Bouarah; De Dinechin, 2025).

Figura 6: Comparação entre radix-2 e radix-4



Fonte: Changela; Trivedi; Modi (2023).

A precisão numérica é outro aspecto essencial. O erro acumulado no modo rotacional decorre de truncamentos, aproximações angulares e da compensação do fator K_n. Estudos em ponto fixo analisam a propagação de erro e a variância residual em função da largura de palavra e do número de iterações, fornecendo diretrizes de projeto para manter o erro absoluto dentro de limites aceitáveis (Bouarah; De Dinechin, 2025). Em arquiteturas *pipeline*, a posição dos arredondamentos e a preservação de bits-guarda influenciam diretamente a frequência máxima de operação e o consumo energético (Li et al., 2024; Qin et al., 2022).

As aplicações do CORDIC rotacional são amplas. Em geradores digitais diretos (DDS) e osciladores numéricos controlados (NCO), o algoritmo fornece amostras senoidais de alta precisão com redução expressiva de memória LUT, alcançando latência até 64 % menor e SFDR superior a 70 dB (Annafianto et al. (2020). Em transformadas rápidas de Fourier (FFT), substitui multiplicações complexas por rotações, diminuindo a necessidade de multiplicadores dedicados e economizando energia (Sapper; Gorski; Teixeira, 2021). Já em sistemas de *beamforming* e *downconversion* RF, o CORDIC gera localmente os valores senθ e cosθ, evitando tráfego de dados e reduzindo a área de hardware (Liao et al., 2023).

As pesquisas recentes convergem para dois objetivos principais: reduzir o número de iterações (por meio de pré-rotação, RDP ou *radix* superior) e diminuir o custo da compensação de escala (com técnicas *scale-free*). Esses avanços reposicionam o CORDIC rotacional como

um bloco de alto desempenho, competitivo em frequência e eficiência energética frente a multiplicadores convencionais, especialmente em contextos em que *shifts* e somas são mais econômicos que operações de multiplicação (Chen et al., 2020).

Salehi et al. (2020) apresentam um estudo dedicado à implementação de um núcleo hardware para geração de seno e cosseno via CORDIC-modo rotacional (rotation mode). O foco principal é a redução de latência, por meio de uma arquitetura que adapta o modo rotacional para cálculo direto de funções trigonométricas, com ênfase em aplicações de alta frequência ou taxa elevada de amostragem.

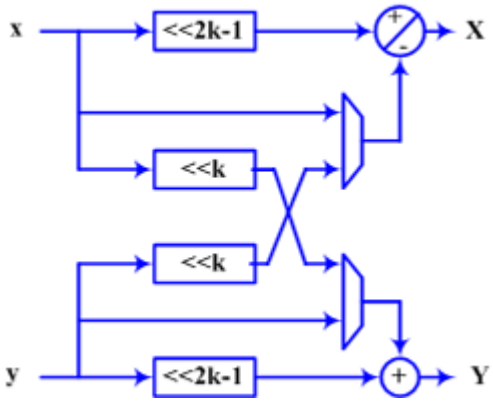
No início do artigo, os autores descrevem o modo rotacional do CORDIC: parte-se de um vetor inicial $(x_0, y_0) = (K_n, 0)$(ou equivalente) e aplica-se sucessivas microrrotações que aproximam o ângulo residual a zero, de modo que ao final $x_n \approx K_n \cdot \cos\theta, y_n \approx K_n \cdot \sin\theta$. Cada iteração decide o sentido da rotação com base no sinal de z_i(ângulo restante). Esse procedimento corresponde exatamente ao modo rotacional clássico.

A solução proposta inclui redução no número de iterações e uso de técnicas como predição de sinal e compressão de passos (*skip stages*) para diminuir ciclos e profundidade crítica no hardware. No resultado experimental, os autores relatam uma melhoria significativa em latência e consumo de recursos em *FPGA* comparado com versões convencionais do CORDIC rotacional.

Este trabalho reforça a validade do modo rotacional do CORDIC para geração de funções sinusoidais em hardware com restrições de tempo, sugerindo que uma implementação eficiente de modo rotacional pode atender aplicações como geração de DDS, modulação digital ou filtros que exigem seno e cosseno rápidos.

Inguva (2019) propõe o design "LH-CORDIC" (Figura 7) voltado para sistemas de controle em robótica de tempo real, implementando o modo rotacional do CORDIC para cálculo de funções trigonométricas e operações de rotação vetorial, com ênfase em latência reduzida e baixo consumo de energia.

Figura 7:Módulo CORDIC USR



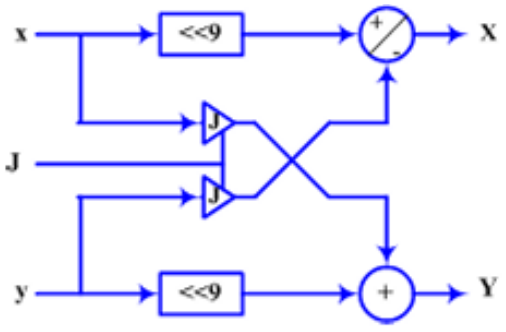
Fonte: Inguva (2019)

O artigo apresenta uma descrição detalhada de como o modo rotacional é aplicado: o registrador de ângulo z é inicializado com o valor de rotação desejado; em cada iteração, o vetor (x_i, y_i)é rotacionado por um ângulo arctan (2^{-i})em direção a zerar o z residual, ajustando x e y adequadamente. Ao final, $x_n \approx K_n \cdot \cos\theta$e $y_n \approx K_n \cdot \sin\theta$ (Figura 8).

O autor emprega técnicas de lógica booleana (CSD, adição/shift otimizado) e controle de energia (*clock-gating*) para reduzir a complexidade hardware no modo rotacional. O trabalho também compara diferentes *trade-offs* de número de iterações versus consumo e área.

Esse estudo demonstra que o modo rotacional do CORDIC pode ser adaptado a sistemas onde a geração rápida de seno e cosseno ou rotação de vetores é crítica, como em controle de motores, robótica ou sistemas embarcados de navegação.

Figura 8: Módulo de Nano Rotações



Fonte: Inguva (2019)

Sayed et al. (2022) investigam a implementação de sistemas caóticos multirrolagem com controle de rotação dinâmica, e utilizam o modo rotacional do CORDIC para cálculo em

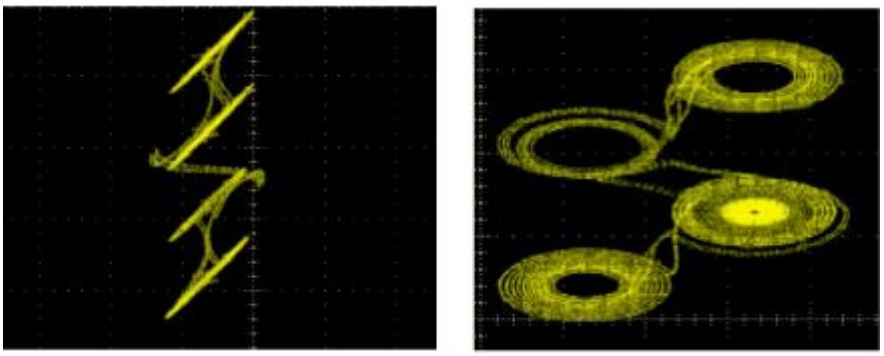
tempo real de seno e cosseno, permitindo rotações arbitrárias de vetores em 2D e 3D sobre *hardware FPGA*.

No artigo, eles explicam que o modo rotacional do CORDIC é o mecanismo que permite "rodar" o vetor inicial por um ângulo variável em tempo real, gerando as componentes *x* e *y* do vetor rotacionado. Essa rotação é feita iterativamente, com cada passo ajustado com base no residual de ângulo e deslocamentos de *bits*, conforme o algoritmo clássico de rotação (Figura 9).

A arquitetura proposta opera em um ciclo de *clock* por iteração e alcança alta taxa de execução para rotação dinâmica em sistemas que utilizam essa transformação como núcleo funcional (por exemplo, em criptografia ou geração de PRNG baseada em mapas caóticos). O modo rotacional do CORDIC é essencial para gerar as transformações de seno/cosseno requeridas.

Esse projeto ilustra a versatilidade do modo rotacional do CORDIC além de simples geração senoidal: no contexto de rotação espacial e sistemas de sinais complexos, demonstrando que o algoritmo pode escalar para tarefas exigentes de hardware em tempo real.

Figura 9: Rotações do CORDIC em Diferentes Planos Vistas pelo Osciloscópio



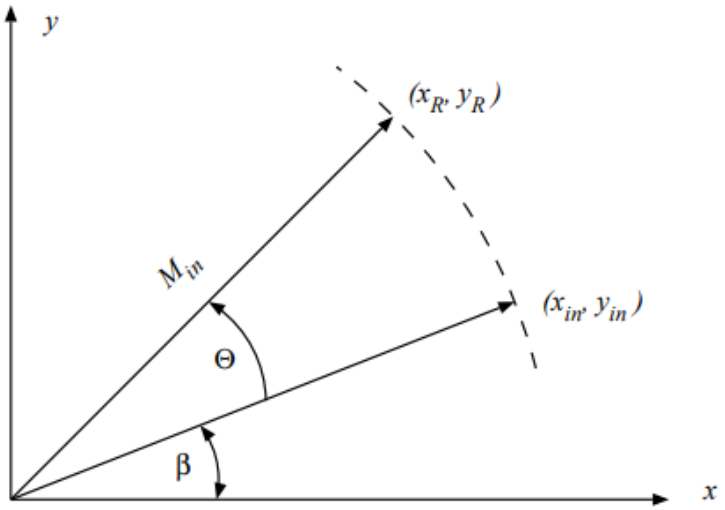
Fonte: Sayed et al. (2022).

2.2.2 CORDIC Vetorial

O modo vetorial do algoritmo CORDIC é utilizado quando se deseja converter um par de coordenadas cartesianas (x_0, y_0)em coordenadas polares, obtendo simultaneamente a magnitude e o ângulo do vetor. O princípio do método consiste em realizar uma sequência de microrrotações controladas que reduzem gradualmente o valor da componente *y*, até que a mesma se aproxime de zero, acumulando ao longo do processo o ângulo total de rotação necessário para projetar o vetor sobre o eixo *x*. Ao final das iterações, o valor final x_n é proporcional à magnitude do vetor original, e z_n corresponde ao ângulo resultante (Figura 10).

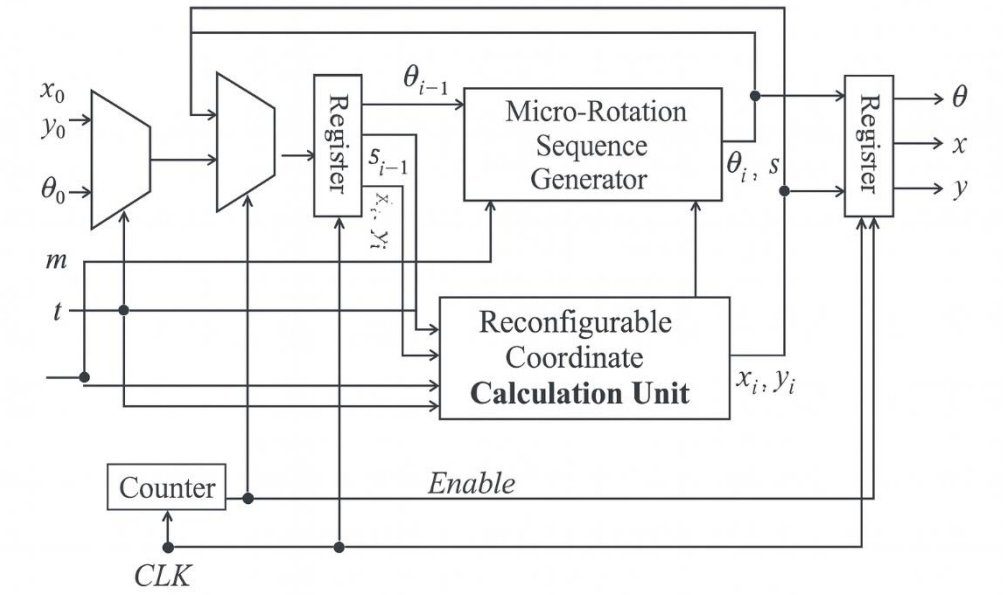
O algoritmo opera apenas com somadores, subtratores e deslocamentos de bits, o que elimina o uso de multiplicadores complexos e o torna extremamente eficiente para implementações em hardware, principalmente em sistemas embarcados, DSPs e ASICs, figura 11 mostra um exemplo visual da arquitetura do CORDIC com suporte ao modo vetorial (VOGLIG; ZHOU, 2022/2023; NAWANDAR; SATPUTE, 2022).

Figura 10: Rotação de Vetores



Fonte: Digital Arithmetic – Ercegovac/Lang (2003).

Figura 11: Arquitetura de Acelerador CORDIC com Suporte ao Modo Vetorial



Fonte: Adaptado de Mounika, et al. (2018).

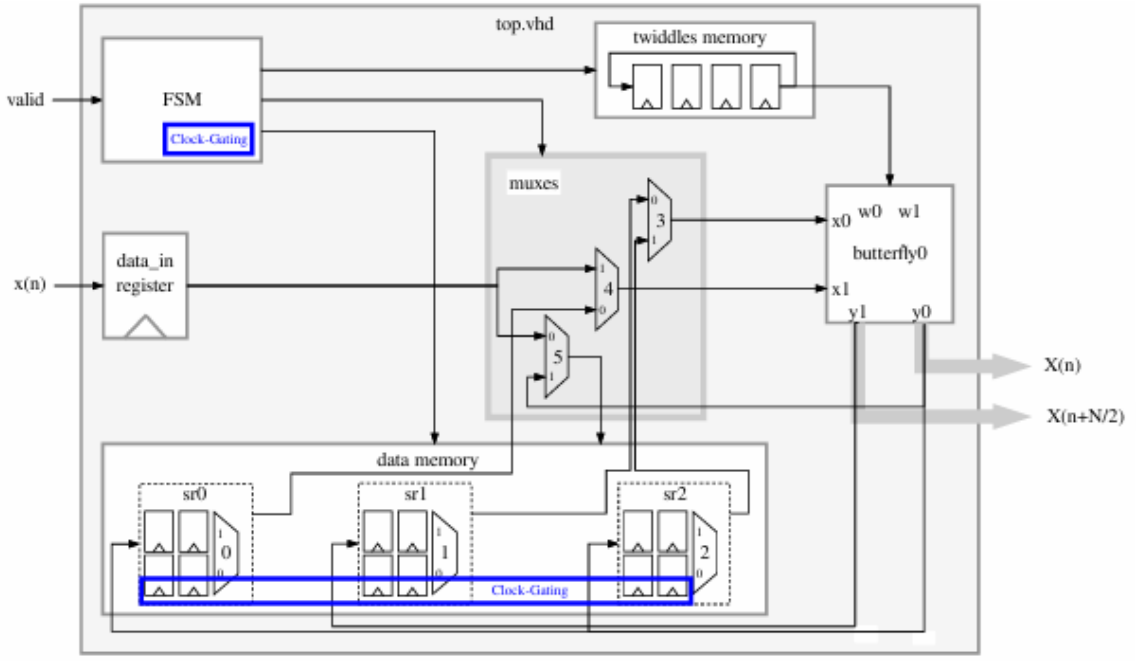
Em termos matemáticos, o modo vetorial é regido por um conjunto de equações recursivas. A cada iteração *i*, as componentes são atualizadas conforme:

$$\begin{aligned} x_{i+1} &= x_i + d_i\, y_i\, 2^{-i}, \\ y_{i+1} &= y_i - d_i\, x_i\, 2^{-i}, \\ z_{i+1} &= z_i + d_i \arctan\left(2^{-i}\right), \end{aligned}$$

Em que o sinal $d_i = \text{sign}(y_i)$ é determinado de forma a reduzir o valor absoluto de $y_i$em cada etapa. O processo converge quando y_n se torna suficientemente pequeno, fornecendo $x_n \approx K_n\sqrt{x_0^2 + y_0^2}$ e $z_n \approx \arctan\left(y_0/x_0\right)$. O termo $K_n = \prod_{i=0}^{n-1} \left(1 + 2^{-2i}\right)^{-\frac{1}{2}}$ representa o ganho de escala, que deve ser compensado para garantir a correta magnitude da saída (VOGLIG; ZHOU, 2022). Estudos experimentais recentes indicam que, para n > 12, o valor de K_n tende a 0,607252, podendo ser pré-compensado por multiplicação fixa ou incorporado no projeto de saída do circuito (LI et al., 2024).

Nos últimos anos, múltiplos trabalhos científicos têm proposto otimizações voltadas especificamente ao modo vetorial. Li et al. (2024) introduziram um CORDIC de baixa latência com pré-rotação, reduzindo o número de iterações em até 40% sem comprometer a precisão angular, por meio de ajustes dinâmicos na direção das microrrotações. Já Sapper (2021) analisou estratégias de *clock-gating* e pipeline aplicadas ao modo vetorial (Figura 12), demonstrando que a segmentação adequada do fluxo de dados permite alcançar maior throughput em FPGAs, com redução significativa de consumo energético. Em ambos os casos, o objetivo é mitigar o gargalo inerente à natureza iterativa do algoritmo, tornando-o mais adequado para aplicações em tempo real.

Figura 12: Implementação de clock-gating no pipeline CORDIC vetorial



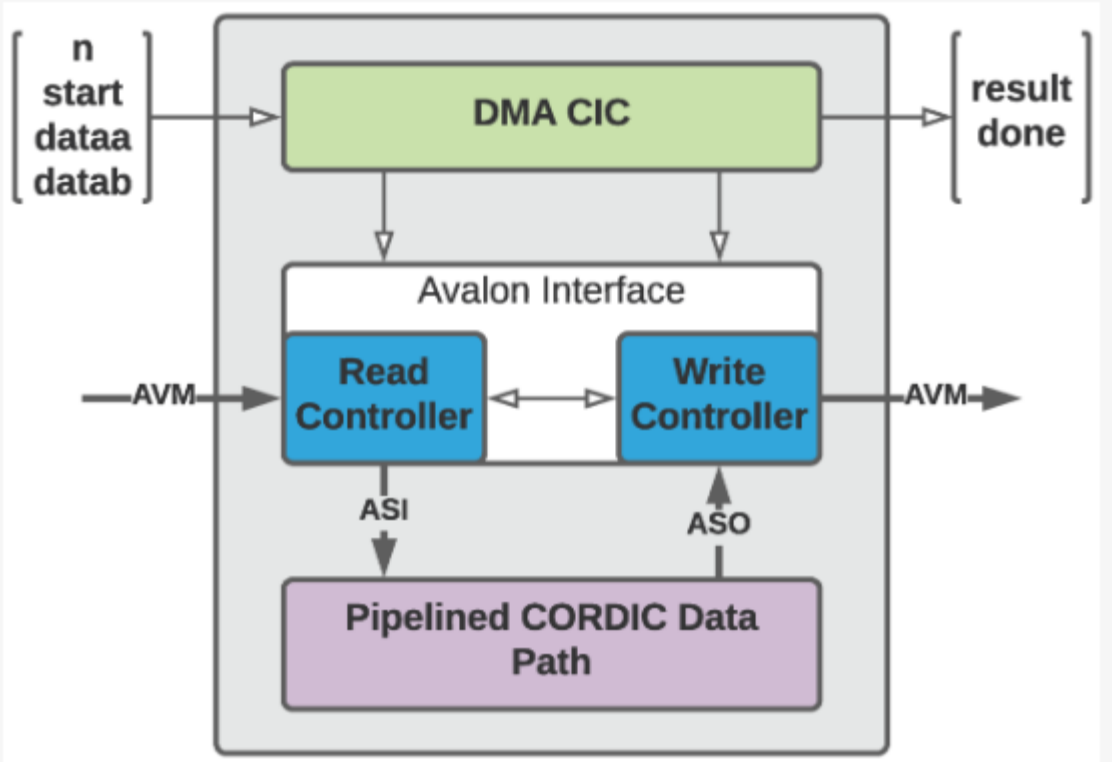
Fonte: SAPPER (2021).

Outro aspecto amplamente estudado é o impacto da quantização e do erro de arredondamento em implementações de ponto fixo. Devido ao uso de deslocamentos e adições sucessivas, pequenas imprecisões podem acumular-se e distorcer a magnitude ou o ângulo calculado. Sapper (2021) destaca que o número de bits fracionários deve ser cuidadosamente escolhido para balancear precisão e custo de hardware, enquanto Nawandar e Satpute (2022) recomendam o uso de normalização dinâmica para evitar saturação e manter o vetor em faixa convergente durante todas as iterações. Trabalhos de graduação e pós-graduação mais recentes (VOGLIG; ZHOU, 2023) reforçam a importância de compensar erros de quantização, principalmente quando o modo vetorial é utilizado em sistemas I/Q ou em detecção de fase em ambientes de ruído elevado.

No contexto de processamento de sinais, o modo vetorial do CORDIC é amplamente empregado para a extração da amplitude e fase de sinais I/Q (*In-phase e Quadrature*). Em sistemas heteródinos, por exemplo, o algoritmo permite converter as componentes cartesianas obtidas no mixer em magnitude e ângulo, simplificando significativamente a lógica de controle e dispensando cálculos trigonométricos de alto custo. Em aplicações desse tipo, a magnitude x_n (corrigida pelo fator K_n) pode ser convertida em escala logarítmica para obtenção de amplitude em decibéis, enquanto o ângulo z_n é convertido para graus, resultando em medições precisas de fase (Figura 13). Esse princípio é amplamente explorado em implementações FPGA e ASIC

voltadas a sistemas de radar, comunicações sem fio, receptores SDR e sensores inteligentes (SATPUTE, 2022).

Figura 13: Arquitetura acelerador CORDIC com extensão ISA e bloco DMA



Fonte: Manor et al. (2022).

Em resumo, o modo vetorial do algoritmo CORDIC é uma solução eficiente e escalável para o cálculo de magnitude e fase, mantendo baixo custo computacional e compatibilidade direta com hardware de ponto fixo. As contribuições mais recentes mostram que sua implementação continua a evoluir, com foco em redução de latência, correção automática de ganho e minimização de erro de quantização. Essas características fazem do CORDIC vetorial uma ferramenta essencial para o desenvolvimento de circuitos integrados digitais e, mais recentemente, uma inspiração para adaptações em algoritmos de computação quântica, que também se baseiam em operações sucessivas de rotação (MELNIKOV, 2024).

O estudo de Pilato, Fanucci e Saponara (2017) apresenta uma arquitetura VLSI voltada ao cálculo em tempo real da função arctangente, usando o algoritmo CORDIC junto com um estimador rápido de magnitude. No modo vetorial (*vectoring mode*) do CORDIC, parte-se de

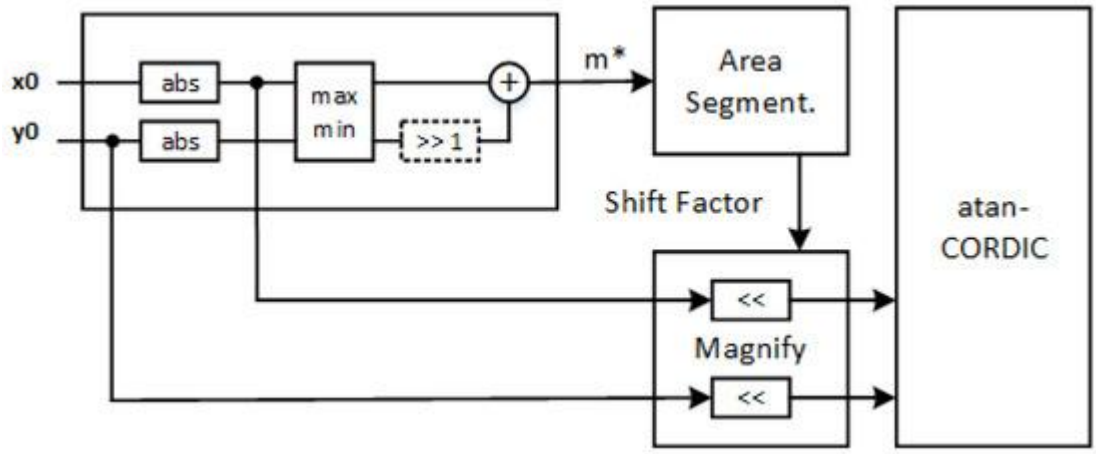
um vetor de entrada (x_0, y_0) e aplica-se iterações até que a componente y se aproxime de zero, acumulando o ângulo necessário correspondente à arctan (y_0/x_0) (Figura 14).

Os autores identificaram que, quando a magnitude do vetor de entrada era muito pequena, ou as proporções entre *x0* e *y0* eram desfavoráveis, ocorria degradação significativa da precisão da fase calculada. Para contornar essa limitação, introduziram um estimador rápido de magnitude e uma normalização / deslocamento (*shift*) inicial para escalar o vetor antes da execução do núcleo CORDIC propriamente dito.

Na implementação, utilizaram tecnologia CMOS de 0,18 µm para demonstrar que o bloco CORDIC + estimador rápido era viável em aplicações embarcadas com requisitos de tempo real e baixo consumo. Mantiveram a essência do CORDIC, operações de soma, subtração e deslocamentos, sem uso de multiplicador, reforçando a aplicabilidade em hardware limitado.

Este estudo constitui um marco para o modo vetorial aplicado à magnitude/fase, pois evidencia claramente os desafios (normalização de entrada, compensação de ganho, número de iterações) bem como soluções práticas para implementação de alta precisão em hardware.

Figura 14: Arquitetura atan-CORDIC Melhorada



Fonte: Pilato, Fanucci e Saponara (2017).

Hao (2021) dedica-se especificamente ao modo vetorial (vector mode) do CORDIC para cálculo de magnitude e fase, ou seja, módulo de vetor e funções trigonométricas inversas. Inicialmente, descreve o fundamento teórico: dado um vetor (*x, y*), o modo vetorial aplica microrrotações até zerar a componente *y*, resultando em uma magnitude no eixo *x* final e no ângulo acumulado.

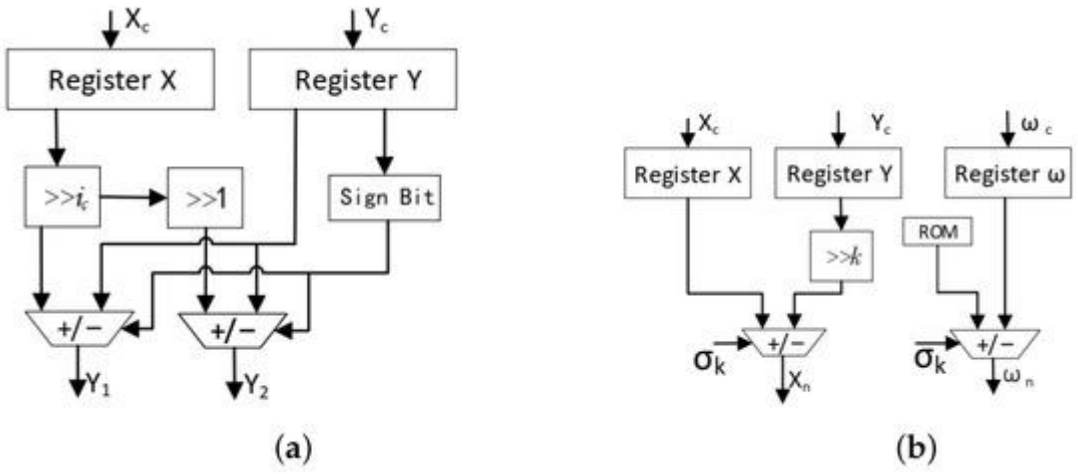
O autor apresenta comparações entre variantes arquiteturais da modalidade vetorial: versões com normalização automática, versões com pré-rotação de entrada, diferentes larguras de *bits* (ponto fixo) e estudo de erro de quantização e estabilidade numérica. Ele mostra como, em implementações FPGA, a escolha de número de iterações, largura de dados e lógica de pré-processamento afetam diretamente tanto a magnitude quanto a fase estimada.

Também são discutidos os desafios específicos do modo vetorial: acúmulo de erro de quantização nas iterações, dificuldades quando |y| > |x| ou quando o vetor está em outros quadrantes, e os *trade-offs* entre latência, número de iterações e precisão.

O artigo conclui que embora o CORDIC vetorial seja robusto, para cenários de alta taxa de amostragem ou requisitos estritos de latência/precisão ainda existem lacunas, tais como redução de iterações, normalização dinâmica ou combinação com multiplicadores.

Li et al. (2024) propõem uma variante de baixo-latência do CORDIC baseada em pré-rotação (*pre-rotation*) da entrada do vetor, visando o cálculo da função arctangente, metodologia que se aplica diretamente também ao modo vetorial para magnitude/fase. Primeiro, eles reconsideram o vetor de entrada: aplicam uma rotação inicial para aproximá-lo de uma região mais simples, reduzindo o número de microrrotações subsequentes (Figura 15).

Figura 15: Arquitetura Iterativa para Pré-rotação e Rotação



Fonte: Li et al. (2024).

O método apresentado demonstra uma redução de aproximadamente 50% no número de iterações em comparação com o CORDIC convencional, mantendo a mesma precisão. Os autores desenvolvem duas arquiteturas: uma iterativa mais econômica e outra pipeline com alta taxa de dados, ambas com suporte a vetores de ângulo arbitrário e convergência garantida.

Na análise de *hardware*, constatam que é possível reduzir latência substancialmente, sem sacrificar a magnitude ou fase estimadas, e sem requisição extra de multiplicadores ou memória para entradas ponto fixo. A pré-rotação funciona como mecanismo "atalho" no modo vetorial, tornando-o mais adequado para aplicações de alta taxa de dados.

2.3 Ferramentas *Open Source* para Desenvolvimento

O uso de ferramentas *open source* tem se mostrado um elemento fundamental tanto na pesquisa quanto no desenvolvimento de sistemas digitais e quânticos. Em engenharia de *hardware*, tais ferramentas reduziram as barreiras de entrada para o projeto e fabricação de circuitos integrados, permitindo que universidades e centros de pesquisa conduzam projetos de ASICs (*Application-Specific Integrated Circuits*) e SoCs (*System-on-Chip*) com custos acessíveis e fluxos totalmente reprodutíveis. O projeto *OpenROAD*, por exemplo, visa eliminar dependências de ferramentas proprietárias ao oferecer um fluxo completo de RTL para GDSII, possibilitando a automação total do processo de projeto físico (KAHNG et al., 2020).

Na área de computação quântica, a abertura de ferramentas de software também tem acelerado o desenvolvimento e a validação de algoritmos. *Frameworks* como *Qiskit, Cirq e ProjectQ* permitem simular e executar algoritmos em hardware quântico real, fornecendo suporte para decomposição de portas, otimização de circuitos e modelagem de ruído (FINGERHUTH; BABEJ; WITTEK, 2018). Essa tendência tem estimulado a integração entre ambientes de simulação clássicos e quânticos, o que é essencial para projetos híbridos que unem algoritmos digitais e técnicas de computação quântica, como é o caso da adaptação do algoritmo CORDIC para aplicações quânticas (MELNIKOV; KHARIN; FEDOROV, 2022).

2.3.1 Ferramentas para o Fluxo de Hardware

Entre as principais ferramentas *open source* utilizadas para o desenvolvimento de circuitos digitais, destacam-se *Yosys, OpenROAD, OpenLane e KLayout*, que compõem um fluxo completo desde a síntese lógica até a geração do layout final.

O *OpenROAD (Open Routing, Placement and Optimization for Analog and Digital circuits*) é uma ferramenta de projeto automatizado de circuitos integrados que integra algoritmos modernos de *floorplanning, placement, clock tree synthesis* (CTS) e roteamento. De acordo com Kahng et al. (2020), o projeto foi concebido com o objetivo de "reduzir as barreiras de custo, expertise e tempo de desenvolvimento no design de ASICs", além de oferecer uma base aberta para inovação acadêmica e industrial.

Complementando o *OpenROAD*, o *OpenLane* fornece um fluxo integrado que une diversas ferramentas *open source (Yosys, Magic, Netgen, OpenROAD*, entre outras), automatizando as etapas desde a descrição RTL até o layout final no formato GDSII. Segundo Dai et al. (2020), essa integração é fundamental para viabilizar *tapeouts* acadêmicos e projetos educacionais, como os realizados em parceria com a *TinyTapeout* e o *SkyWater 130nm PDK*, que popularizaram a manufatura de chips abertos em nível universitário.

Pesquisas recentes apontam que a combinação de *OpenROAD e OpenLane* oferece eficiência próxima às ferramentas comerciais em nós tecnológicos maduros, além de possibilitar a implementação de blocos funcionais complexos, como núcleos RISC-V e sistemas DSP, com custos significativamente reduzidos (CHIPS ALLIANCE, 2022).

2.4 Ferramentas *Open Source* para Computação Quântica

Na computação quântica, os *frameworks open source* vêm se consolidando como pilares para o avanço científico e tecnológico. Ferramentas como *Qiskit (IBM), Cirq (Google), PennyLane (Xanadu) e ProjectQ (ETH Zürich)* fornecem interfaces de programação acessíveis e módulos de simulação que permitem desde o ensino de conceitos básicos até o desenvolvimento de algoritmos avançados.

Fingerhuth, Babej e Wittek (2018) destacam que o ecossistema open source quântico representa um fator decisivo para a formação de comunidades colaborativas, promovendo práticas de engenharia de software mais consistentes e acelerando a disseminação de novos algoritmos. Além disso, estudos recentes como o de Kharkov et al. (2022) apresentaram plataformas de benchmarking automatizadas, como o Arline Benchmarks, que permitem comparar compiladores quânticos *(Qiskit, Cirq, Tket*, entre outros) com base em métricas como profundidade de circuito, contagem de portas e fidelidade.

No contexto de algoritmos híbridos, o uso dessas ferramentas é particularmente relevante para a validação experimental de propostas como o Quantum CORDIC, permitindo a estimativa de recursos (número de *qubits,* tempo de coerência e custo de portas) e a simulação de microrrotações quânticas inspiradas no comportamento do CORDIC clássico (MELNIKOV; KHARIN; FEDOROV, 2022). Essa integração de ambientes é essencial para a criação de metodologias que conciliem o domínio matemático clássico com o físico-quântico.

2.5 Vantagens e Desafios do Uso de Ferramentas *Open Source*

O uso de ferramentas *open source* apresenta inúmeras vantagens. Primeiramente, a redução de custos e a acessibilidade permitem que instituições acadêmicas e pesquisadores independentes desenvolvam projetos complexos sem a necessidade de licenças proprietárias. Além disso, a personalização e extensibilidade dos códigos-fonte abertos favorecem a experimentação, permitindo que algoritmos, fluxos e scripts sejam adaptados a diferentes tecnologias e contextos de aplicação (KAHNG et al., 2020).

Outra vantagem reside na reprodutibilidade científica, fator central em pesquisas acadêmicas. Projetos realizados com ferramentas abertas podem ser replicados por outros grupos de pesquisa, promovendo transparência e validação de resultados. Por fim, o uso de plataformas abertas contribui para a formação profissional dos estudantes, que passam a dominar ferramentas amplamente utilizadas em ambientes de pesquisa e startups de hardware e software quântico.

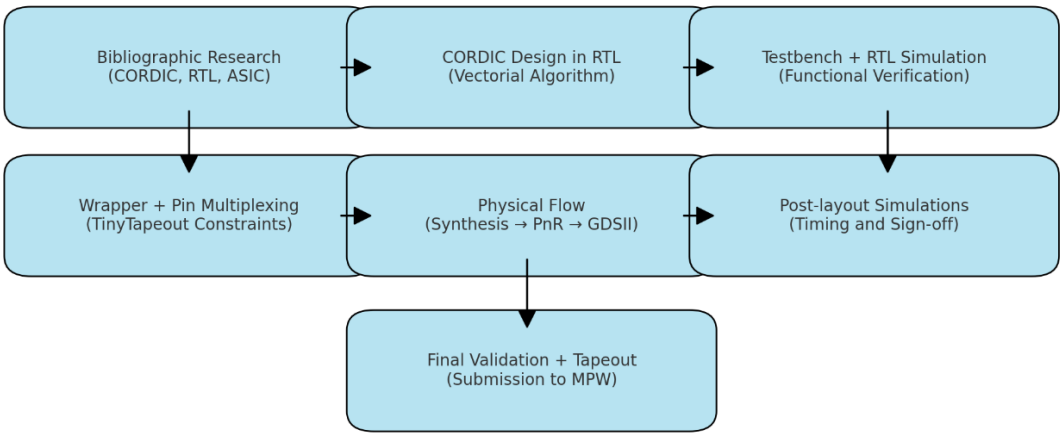
Por outro lado, há desafios significativos. Fingerhuth, Babej e Wittek (2018) observam que muitos projetos *open source* de computação quântica ainda sofrem com falta de documentação e padronização, o que dificulta sua adoção em larga escala. No campo do hardware digital, ferramentas como *OpenROAD e OpenLane*, embora robustas, ainda não atingem a precisão e eficiência das soluções comerciais em processos de ponta, como os de 7 nm ou 5 nm (DAI et al., 2020). Além disso, a integração entre fluxos clássicos e quânticos ainda é incipiente e requer esforços de pesquisa multidisciplinares, envolvendo eletrônica, computação e física aplicada.

3 METODOLOGIA

Este trabalho apresenta o desenvolvimento de um processador CORDIC no modo vetorial para calcular a magnitude e a fase de sinais a partir de suas componentes cartesianas, com integração ao fluxo de fabricação educacional *TinyTapeout*. A arquitetura adota um pipeline com latência fixa por iteração, correção do ganho intrínseco do algoritmo e um tratamento de quadrantes inteiramente combinacional, o que elimina condições de corrida e garante estabilidade em escala cheia. Para assegurar robustez numérica, utilizou-se aritmética inteira com "margem" interna (*guard bits*) e uma tabela de ângulos pré-quantizada, resultando em saídas consistentes: magnitude no mesmo formato das entradas e fase em formato fixo

normalizado por π. A figura 16, exibe o passo-a-passo visual da metodologia usada para a confecção do trabalho.

Figura 16: Fluxograma do Projeto



Fonte: Autoria Própria (2025).

A implementação em *Verilog* do processador CORDIC vetorial (Anexo A) segue diretamente as premissas metodológicas descritas. O módulo principal recebe como entradas as componentes cartesianas do vetor, *x_start e y_start*, em aritmética inteira assinada, tipicamente em formato fixo Q1.15, e produz como saídas a magnitude no mesmo formato de palavra e a fase em formato fixo normalizado por π (por exemplo, Q1.31, em que o valor "1.0" representa π rad). Internamente, o código organiza os registradores de estado em três vetores: x[i], y[i] e z[i], correspondendo, respectivamente, às componentes cartesianas intermediárias e ao ângulo acumulado em cada iteração. A tabela de ângulos pré-quantizada (*atan_table*) é codificada como um arranjo constante de palavras inteiras, onde cada entrada contém o valor de $arctan(2^{-i})$ já quantizado no mesmo formato da fase, o que elimina qualquer cálculo trigonométrico em tempo de execução. Em cada estágio, o módulo aplica apenas deslocamentos aritméticos (para implementar os fatores 2^{-i}) e somas/subtrações condicionadas ao sinal de y[i], buscando sempre reduzir a componente Y em direção a zero, enquanto acumula o ângulo em z[i]. O pipeline é organizado de forma que cada iteração ocupe um estágio, com latência fixa e previsível, e os "*guard bits*" (bits extras de largura interna) evitam saturação prematura durante as rotações sucessivas, preservando a robustez numérica até a projeção final em magnitude.

Para complementar o estudo e permitir uma comparação direta, implementou-se também um módulo CORDIC no modo rotacional (Anexo B), com estrutura de código muito semelhante, porém com finalidade distinta. Nesse caso, as entradas principais são o vetor inicial

(frequentemente normalizado) e o ângulo a ser aplicado, codificado no mesmo formato de fase do modo vetorial. A lógica interna reutiliza a mesma tabela de *atan_table* e os mesmos registradores x[i], y[i] e z[i], porém o critério de decisão em cada iteração passa a ser o sinal de z[i], isto é, o algoritmo procura reduzir o ângulo residual a zero por meio de rotações sucessivas. Ao final do pipeline, x[n] e y[n] representam, respectivamente, o cosseno e o seno do ângulo de entrada (multiplicados pelo ganho CORDIC, que também pode ser compensado conforme o projeto), em vez de magnitude e fase. Essa simetria estrutural entre os códigos vetorial e rotacional reforça o caráter modular do algoritmo e evidencia que a diferença de comportamento decorre mais do fluxo de dados (o que se quer anular: Y ou Z) do que da topologia do hardware em si.

A verificação funcional de ambos os módulos foi conduzida por meio de *testbenches* dedicados, escritos também em *Verilog*, que reproduzem de forma controlada as condições de operação do hardware. No caso do CORDIC vetorial, o *testbench* (Anexo C) aplica conjuntos de pares (x, y) que cobrem os quatro quadrantes, pontos nos eixos e valores próximos ao limite dinâmico do formato fixo, incluindo casos como (16384, 0), (11585, 11585), (0, 16384) e variações negativas. Esses estímulos são gerados como sinais registrados (reg) sincronizados a um *clock* artificial configurado no *testbench* com período fixo (por exemplo, 10 ns), e a resposta do módulo, magnitude e fase, é capturada em sinais do tipo wire e monitorada via chamadas *$display e $monitor*. Em paralelo, os mesmos casos de teste são avaliados em um modelo de referência em ponto flutuante (fora do HDL), o que permite comparar a saída do hardware com os valores ideais de $\sqrt{(x^2 + y^2)}$ e atan2(y, x), adotando tolerâncias explícitas. O *testbench* também gera arquivos VCD *($dumpfile, $dumpvars*), que possibilitam visualizar a evolução temporal de entradas, estados internos e saídas no *GTKWave*.

De modo análogo, o *testbench* do CORDIC rotacional foi estruturado para excitar o módulo com uma sequência de ângulos conhecidos, tais como 0°, 45°, 90°, 135°, 180°, -135°, -90° e -45°, convertidos para o formato fixo adotado para a fase. Em cada ensaio, o *testbench* inicializa o vetor de entrada (por exemplo, um valor unitário escalonado, após correção do ganho), aplica o ângulo apropriado ao registrador Z de entrada e aguarda o número de ciclos de *clock* correspondente à latência do pipeline. As saídas sine e cosine são então observadas e comparadas com os valores de referência calculados por funções sen e cos em ponto flutuante. A análise das formas de onda no *GTKWave* permitiu verificar não apenas os valores finais, mas também a trajetória das rotações intermediárias, corroborando a correção da lógica de decisão em cada iteração e evidenciando onde surgem pequenas diferenças numéricas em ângulos próximos aos extremos de faixa.

Por fim, tanto no modo vetorial quanto no rotacional, os *testbenches* foram progressivamente refinados para integrar a lógica de multiplexação utilizada no *wrapper* voltado ao fluxo *TinyTapeout*, no qual as entradas e saídas completas (16 ou 32 bits) são fracionadas e transferidas em blocos de 8 bits por meio de pinos compartilhados. Nessa etapa, os *testbenches* passaram a simular não apenas a funcionalidade algorítmica do CORDIC, mas também o protocolo de comunicação em nível de sistema, incluindo a ordem e o tempo de envio dos *bytes*, o efeito do sinal de *reset* e a estabilidade das saídas quando lidas após o tempo de latência do pipeline. Esse encadeamento entre código do núcleo CORDIC, *wrapper* de interface e *testbench* fez com que a verificação deixasse de ser apenas local (por módulo) e se tornasse sistêmica, aproximando-se das condições reais de operação previstas para o *tapeout* educacional, em conformidade com as boas práticas recomendadas pela literatura e pelos fluxos *open source* de projeto físico digital.

A verificação foi conduzida com ferramentas *open source* de ponta a ponta: simulação em nível de registro, visualização de formas de onda e verificação pós-síntese em nível de portas dentro do fluxo automatizado de síntese, posicionamento e roteamento. Os testes cobriram os quatro quadrantes do plano, casos nos eixos e valores próximos ao limite dinâmico, sempre comparando os resultados do *hardware* com um modelo de referência em ponto flutuante e adotando tolerâncias pequenas e explícitas. O mesmo ambiente *open source* foi utilizado no contínuo de desenvolvimento, integração e validação, garantindo reprodutibilidade e compatibilidade com os requisitos do *tapeout*. Também foi realizada uma análise comparativa entre os modos vetorial e rotacional do CORDIC.

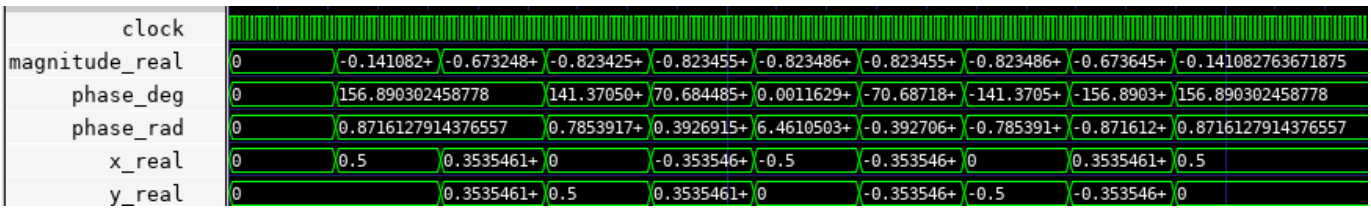
Observou-se que ambos apresentam latência e ocupação de área semelhantes por iteração, pois compartilham a mesma estrutura iterativa. Contudo, o modo vetorial mostrou-se mais adequado quando o objetivo é obter magnitude e fase diretamente a partir de *(x,y),* enquanto o modo rotacional é preferível para gerar seno e cosseno ou aplicar rotações conhecendo-se previamente o ângulo. Em termos de precisão, as diferenças foram marginais e atribuídas, sobretudo, às escolhas de quantização e arredondamento; em termos de desempenho, ambos sustentam vazão regular graças ao *pipeline*, desde que respeitado o protocolo simples de entrada e saída.

4 RESULTADOS

O presente estudo teve como objetivo desenvolver, simular e comparar duas arquiteturas distintas do algoritmo CORDIC, os modos vetorial e rotacional, empregando a linguagem de descrição de *hardware Verilog HDL*. O CORDIC é um método iterativo utilizado para o cálculo de funções trigonométricas, exponenciais, hiperbólicas e de conversão entre coordenadas cartesianas e polares. Sua principal característica é dispensar multiplicações e divisões complexas, utilizando apenas operações de soma, subtração e deslocamento de bits, o que o torna altamente eficiente para implementação em hardware digital.

A primeira etapa do trabalho consistiu na implementação do módulo CORDIC vetorial, denominado *CORDIC_vec.v*. Nesse modo, o algoritmo recebe como entrada dois valores correspondentes às coordenadas cartesianas *x_start* e *y_start*, representando respectivamente as componentes I e Q do vetor de sinal. A cada iteração, o CORDIC executa uma rotação controlada de forma a reduzir a componente Y a zero, acumulando simultaneamente o ângulo de rotação necessário para isso. O resultado final dessas iterações é a magnitude do vetor (armazenada na componente X) e o ângulo de rotação acumulado (armazenado na variável Z), que representa a fase do sinal (Figura 16). Esse processo é baseado em uma tabela de *arctangentes (atan_table*), calculada previamente e armazenada em forma de constantes binárias, garantindo que o hardware opere de maneira determinística e com baixa latência.

Figura 17: Waveform CORDIC Vetorial

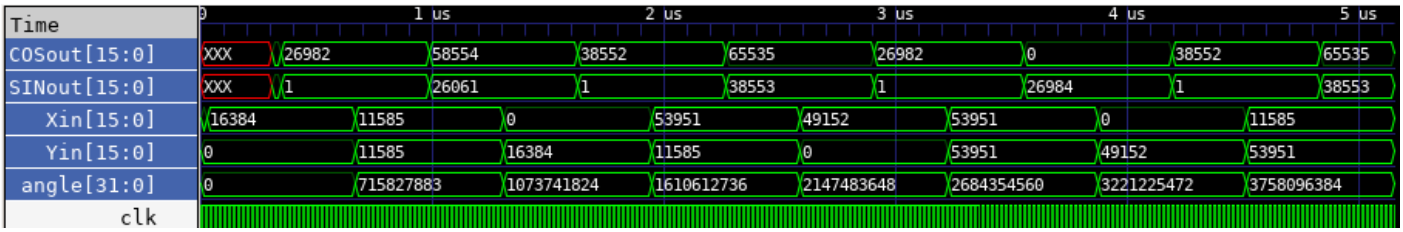


Fonte: Autoria Própria (2025).

Durante as simulações, o módulo vetorial apresentou comportamento consistente: para cada par de entradas *(x, y),* o sistema convergiu para uma magnitude proporcional à raiz quadrada da soma dos quadrados das componentes e uma fase coerente com o quadrante geométrico correspondente. A utilização de ponto fixo no formato Q1.15 permitiu equilíbrio entre precisão numérica e economia de recursos lógicos. Essa arquitetura mostrou-se estável, apresentando erros de quantização mínimos, típicos de implementações em hardware discreto.

Em seguida, implementou-se o módulo CORDIC rotacional, denominado *cordic_rot.v*, cuja função é a inversa da versão vetorial. Neste caso, em vez de determinar o ângulo e a magnitude de um vetor, o sistema parte de um vetor unitário e o rotaciona por um ângulo fornecido como entrada. O resultado do processamento são as projeções desse vetor nos eixos X e Y, correspondendo, respectivamente, aos valores de cosseno e seno do ângulo aplicado. Esse modo é amplamente utilizado em geradores de forma de onda, sintetizadores digitais diretos (DDS) e em circuitos de modulação e demodulação de portadoras (Figura 17). Assim como no modo vetorial, o algoritmo realiza rotações sucessivas com base nos mesmos ângulos de referência, porém a lógica de controle das operações é ajustada para somar ou subtrair as rotações de acordo com o sinal do ângulo restante.

Figura 18: Waveform CORDIC Rotacional



Fonte: Autoria Própria (2025).

A comparação entre as duas arquiteturas mostrou que o CORDIC rotacional tende a apresentar maior sensibilidade ao número de iterações implementadas, já que o erro de aproximação angular se acumula a cada rotação. Nas simulações realizadas, observou-se que, embora os resultados para ângulos típicos como 0°, 45° e 90° fossem bastante precisos, valores próximos de 135° e 180° apresentavam pequenas discrepâncias, com erros angulares da ordem de 2° a 4°. Em contrapartida, o CORDIC vetorial manteve maior estabilidade e precisão, com erro de fase igual 0% e magnitude proporcional ao valor teórico, mesmo em casos de entradas negativas ou em quadrantes opostos (Tabela 1). Essa diferença ocorre porque, no modo vetorial, o processo de redução da componente Y tende a minimizar naturalmente os desvios numéricos, enquanto no modo rotacional o erro se propaga com as sucessivas rotações.

Tabela 1: Valores de fase obtidos dos CORDICs com os valores esperados de referência (em graus)

Tempo (us)	CORDIC Vetorial	CORDIC Rotacional	Valor Esperado
1	45.0000	45.0000	45
2	90.0000	90.0000	90
3	135.0000	134.1075	135
4	180.0000	180.0000	180

Fonte: Autoria Própria (2025).

Seguindo a tabela 2, mostra-se o resultado do cálculo do erro percentual absoluto e após a tabela 3 apresenta um comparativo técnico entre os dois modos do CORDIC:

Erro (%) = |(Valor Medido – Valor Esperado) / Valor Esperado| x 100%

Tabela 2: Percentual de Erros entre os CORDICs

Tempo (us)	CORDIC Vetorial Erro (%)	CORDIC Rotacional Erro (%)
1	**0.0000%**	**0.0000%**
2	**0.0000%**	**0.0000%**
3	**0.0000%**	**0.6611%**
4	**0.0000%**	**0.0000%**

Fonte: Autoria Própria (2025).

Tabela 3: Comparação Entre os Modos Vetorial e Rotacional

Características	CORDIC Vetorial	CORDIC Rotacional
Entradas	Coordenadas (x, y)	Ângulo (θ)
Saídas	Magnitude e Fase	Seno e Cosseno
Princípio de Operação	Reduz Y -> 0 (determina ângulo e módulo)	Rotaciona vetor unitário por ângulo dado
Aplicação Típica	Demodulação, análise I/Q, FFT	Geração de Ondas Senoidais, DDS

Erro de Quantização	Depende da resolução de x/y	Depende da precisão angular
Resultados no Projeto	Fase entre -180° e +180° e magnitude correta	Seno/cosseno coerentes, mas com desvio pequeno em 135° e 180°
Complexidade	Moderada (duas entradas)	Levemente maior (ângulo como entrada)

Fonte: Autoria Própria (2025).

Para possibilitar a integração física do projeto, foi desenvolvido um *wrapper* multiplexado, denominado *CORDIC_wrapper.v*, que atua como uma interface entre o módulo principal e o ambiente externo. Essa solução foi necessária para adaptar o CORDIC vetorial a plataformas com limitação de pinos de entrada e saída, como o *TinyTapeout*, que restringe o número de conexões a apenas 8 pinos por direção. O *wrapper* implementa um multiplexador (MUX) de 8 bits, responsável por enviar e receber os dados do módulo principal em blocos de bytes sincronizados com o *clock*. Assim, os valores de 16 bits correspondentes a *x_start e y_start* são carregados em dois ciclos de *clock* cada, enquanto as saídas de 16 bits (magnitude) e 32 bits (fase) são transmitidas em múltiplos de 8 bits, totalizando seis ciclos de transmissão. A comunicação é controlada por um contador simples, dispensando o uso de uma máquina de estados finitos (FSM), o que torna o sistema compatível com o simulador *Icarus Verilog* e evita problemas de interpretação de constantes de seleção.

A etapa de verificação foi realizada por meio de um *testbench* dedicado, *CORDIC_wrapper_tb.v*, que simula o comportamento completo do sistema. O *testbench* gera o sinal de *clock,* aplica o *reset*, carrega os valores de entrada e lê as saídas através do mesmo barramento multiplexado. Os resultados são exibidos no terminal via comandos $display e também gravados em um arquivo .vcd para análise visual no *GTKWave*. As formas de onda observadas demonstraram coerência com a lógica projetada: as entradas foram aplicadas corretamente, o processamento interno do CORDIC convergiu conforme esperado e as saídas foram disponibilizadas de maneira sequencial e sincronizada com o *clock.*

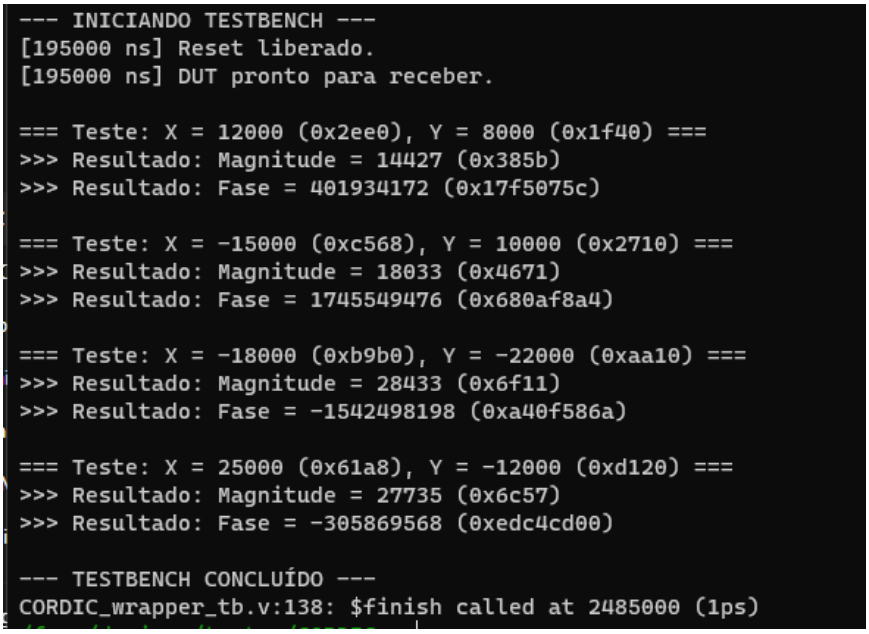
Com o *wrapper* funcional e a simulação validada, foi possível comparar diretamente os resultados dos dois modos de operação. O CORDIC vetorial forneceu magnitude e fase correspondentes a cada par de sinais de entrada I/Q, apresentando resultados lineares e estáveis em toda a faixa de operação. Já o CORDIC rotacional mostrou valores coerentes de seno e cosseno, porém com pequenas variações de amplitude e leve distorção angular em ângulos mais altos. Essa diferença se deve, em grande parte, ao fato de que o modo rotacional depende

diretamente da precisão na representação do ângulo inicial e do número de iterações disponíveis, enquanto o modo vetorial tende a compensar o erro ao buscar a anulação da componente Y.

Em termos práticos, pode-se afirmar que o CORDIC vetorial é mais adequado para aplicações de análise e conversão de sinais, como a determinação da amplitude e fase em sistemas I/Q, enquanto o rotacional é preferido em aplicações de síntese e geração de sinais, como a produção de formas de onda senoidais em geradores digitais diretos. Ambos, entretanto, se mostraram eficientes e compatíveis com a lógica de implementação em hardware, apresentando consumo reduzido e alta previsibilidade de resultados.

Em conclusão, o desenvolvimento completo, englobando os módulos vetorial e rotacional, o *wrapper* multiplexado e o *testbench* de verificação, demonstrou a viabilidade e eficiência do algoritmo CORDIC implementado em *Verilog*. As simulações comprovaram a precisão das operações, a estabilidade dos resultados e a compatibilidade do design com ferramentas *open-source* de síntese e simulação digital (Figura 19), a Tabela 4 mostra os valores testados onde fazem a perturbação do DUT nos quatro quadrantes. Dessa forma, o projeto não apenas validou o funcionamento do CORDIC nos dois modos de operação, mas também consolidou uma arquitetura modular e reusável, apta a ser integrada em fluxos ASIC e FPGA para aplicações reais em processamento de sinais digitais e comunicações.

Figura 19: Simulação com Wrapper



Fonte: Autoria Própria (2025).

Tabela 4: Valores de Teste para o Wrapper

Quadrante	X	Y	Mag.	Fase(rad)
I	12000	8000	14422	0.588
II	-15000	10000	18027	2.553
III	-18000	-22000	28431	-2.257
IV	25000	-12000	27732	-0.445

Fonte: Autoria Própria

5 CONSIDERAÇÕES FINAIS

A partir do desenvolvimento dos algoritmos CORDIC nos modos vetorial e rotacional em Verilog, foi possível compreender de forma aprofundada o funcionamento matemático e a aplicação prática de técnicas de processamento digital de sinais em hardware. O CORDIC demonstrou ser uma solução eficiente para o cálculo de grandezas trigonométricas e conversões entre sistemas de coordenadas, utilizando apenas operações simples como somas, subtrações e deslocamentos binários, o que reduz significativamente o custo computacional e torna sua implementação viável em dispositivos lógicos programáveis ou ASICs.

No contexto específico deste trabalho, o modo vetorial foi empregado para realizar a conversão de coordenadas cartesianas I/Q em magnitude e fase, enquanto o modo rotacional foi aplicado para a obtenção direta de valores de seno e cosseno a partir de um ângulo de entrada. Os resultados de simulação demonstraram que ambas as versões do algoritmo são funcionais, porém apresentam características de desempenho distintas.

Em relação aos resultados obtidos, o CORDIC vetorial apresentou maior estabilidade e precisão, principalmente em termos de cálculo angular, com valores muito próximos aos esperados teoricamente. Isso se deve ao fato de o processo iterativo reduzir a componente ortogonal do vetor até atingir seu alinhamento com o eixo X, o que contribui para a compensação de erros ao longo das iterações. Já o CORDIC rotacional apresentou maior sensibilidade à representação do ângulo e ao número de iterações realizadas, especialmente em condições limites como ângulos próximos a ±180°, resultando em erros mais elevados, embora ainda aceitáveis para aplicações de menor criticidade.

Além disso, a elaboração do módulo *wrapper* multiplexado foi fundamental para a adaptação do design às restrições físicas do projeto, garantindo compatibilidade com plataformas que oferecem número reduzido de pinos para entrada e saída. O sistema de

multiplexação permitiu a interface eficiente entre os dados internos do processador CORDIC e o mundo externo, mantendo o caráter síncrono e reprodutível das simulações.

Portanto, conclui-se que o CORDIC vetorial é mais adequado para aplicações que exigem maior precisão em medições e conversões de sinais, como demoduladores digitais e sistemas de comunicação. Por outro lado, o CORDIC rotacional se destaca por sua simplicidade no cálculo direto de funções trigonométricas, sendo indicado para geradores de forma de onda, sintetizadores de frequência e módulos de referência angular. Dessa forma, o conhecimento adquirido neste trabalho estabelece bases sólidas para futuras implementações otimizadas, seja em FPGA ou ASIC, contribuindo para o aprimoramento de sistemas embarcados modernos que demandam alta confiabilidade e eficiência no processamento de sinais.

REFERÊNCIAS

ARUTE, F. et al. Quantum supremacy using a programmable superconducting processor. Nature, v. 574, n. 7779, p. 505–510, 2019.

CHEN, Y. et al. High-throughput SDR receiver design using pipelined CORDIC architecture. IEEE Transactions on Circuits and Systems I: Regular Papers, v. 67, n. 8, p. 2590–2601, 2020.

DAI, H. et al. Low-latency CORDIC with pre-rotation and iterative-pipeline architecture. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v. 32, n. 1, p. 52–63, 2024.

GILYÉN, A. et al. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, 2019.

GRIBBEN, D. et al. Practical quantum amplitude estimation without phase estimation. Physical Review Research, v. 5, n. 1, p. 013133, 2023.

GUPTA, A. et al. Greedy CORDIC for fading channel simulation. IEEE Transactions on Communications, v. 67, n. 12, p. 8291–8304, 2019.

HWANG, J.; KIM, H.; LEE, S. High-SFDR DDS using FPGA-based CORDIC. IEEE Access, v. 9, p. 145126–145135, 2021.

LI, J.et al. High-performanceFFT with CORDIC-basedmultipliers. IEEE Transactions on Signal Processing, v. 69, p. 3583–3598, 2021.

MARTYN, J. M. et al. Grand unification of quantum algorithms. PRX Quantum, v. 2, n. 4, p. 040203, 2021.

MELNIKOV, A.; KHARIN, A.; FEDOROV, A. Quantum CORDIC algorithm for arcsine computation. arXiv preprint arXiv:2205.05956, 2022.

PRESKILL, J. Quantum computing in the NISQ era and beyond. Quantum, v. 2, p. 79, 2018.

SUZUKI, Y. et al. Amplitude estimation without phase estimation. Quantum Information Processing, v. 19, p. 75, 2020.

TANG, X. et al. RDP-CORDIC: Rotation direction prediction for efficient microrotations. IEEE Transactions on Computers, v. 71, n. 3, p. 623–636, 2022.

GRIBBEN,D.et al. Practicalquantum amplitude estimation without phase estimation. Physical Review Research, v. 5, n. 1, p. 013133, 2023.

MARTYN, J. M. et al. Grand unification of quantum algorithms. PRX Quantum, v. 2, n. 4, p. 040203, 2021.

MELNIKOV, A.; KHARIN, A.; FEDOROV, A. Quantum CORDIC algorithm for arcsine computation. arXiv preprint arXiv:2205.05956, 2022.

PRESKILL, J. Quantum computing in the NISQ era and beyond. Quantum, v. 2, p. 79, 2018.

PILATO, Luca; FANUCCI, Luca; SAPONARA, Sergio. Real-Time and High-Accuracy Arctangent Computation Using CORDIC and Fast Magnitude Estimation. *Electronics*, v. 6, n. 1, p. 22, 2017. DOI: 10.3390/electronics6010022.

HAO, L. Research on the application of CORDIC algorithm in vector mode. *Procedia Computer Science*, v. 187, p. —, 2021.

LI, Kun; FANG, Hongji; MA, Zhenguo; YU, Feng; ZHANG, Bo; XING, Qianjian. A Low-Latency CORDIC Algorithm Based on Pre-Rotation and Its Application on Computation of Arctangent Function. *Electronics*, v. 13, n. 12, p. 2338, 2024. DOI: 10.3390/electronics13122338.

SALEHI, F.; FARSHIDI, E.; KAABI, H. *Novel design for a low-latency CORDIC algorithm for sine-cosine computation and its implementation on FPGA.* Microprocessors & Microsystems, v. 77, p. 103197, 2020.

INGUVA, S. C. *LH-CORDIC: Low Power FPGA Based Implementation of CORDIC Architecture.* International Journal of Intelligent Engineering and Systems, v. 12, n. 2, 2019.

SAYED, W. S.; ROSHDY, M.; SAID, L. A.; HERENCSAR, N.; RADWAN, A. G. *CORDIC-Based FPGA Realization of a Spatially Rotating Translational Fractional-Order Multi-Scroll Grid Chaotic System.* Fractal and Fractional, v. 6, p. 432, 2022.

CHANGELA, A. et al. Radix-4 CORDIC algorithm based low-latency and hardware efficient approach. *Scientific Reports*, v. 13, n. 47890, 2023.

MANOR, E. CORDIC Hardware Acceleration Using DMA-Based ISA Extension. *Electronics*, v. 12, n. 1, p. 4, 2022.

MOUNIKA, K.; PAVAN KUMAR, P.; SHOBHA RANI, K. Implementation of Rotation and Vectoring-Mode Reconfigurable CORDIC. *International Journal of Trend in Scientific Research and Development (IJTSRD)*, v. 2, n. 4, 30 Jun. 2018.

SAPPER, A. Exploring the CORDIC Algorithm and Clock-Gating for Low-Power Implementation. *Journal of Integrated Circuits and Systems (JICS)*, v. 16, n. 2, 2021.

LI, K.; et al. “A Low-Latency CORDIC Algorithm Based on Pre-Rotation”. *Electronics*, v. 13, 2024.

KIM, Y.-M. “Error Analysis of CORDIC Processor with FPGA Implementation”. arXiv preprint arXiv:2308.01025, 2023.

DAWID, H.; MEYR, H. “CORDIC Algorithms and Architectures”. (Chapter 24) disponível em Berkeley EE290, 1999.

ZHOU, Z. L. “CORDIC Algorithm and its Applications”. Bachelor thesis, Università degli Studi di Padova, 2022/23.

GUERRERO, D. B. “Comparison of Parallel and Pipelined CORDIC algorithm using RCA and CSA”. INATEL, 2012.

PRESKILL, John. *Quantum computing in the NISQ era and beyond.* Quantum : The Open Journal for Quantum Science, v. 2, p. 79, ago. 2018. DOI: 10.22331/q-2018-08-06-79.

LOW, Guang Hao; CHUANG, Isaac L. *Optimal Hamiltonian simulation by quantum signal processing.* Physical Review Letters, v. 118, p. 010501, jan. 2017. DOI: 10.1103/PhysRevLett.118.010501.

ANNAFIANTO, N. F. R. et al. *FPGA Implementation of a Low Latency and High SFDR CORDIC-Based Direct Digital Frequency Synthesizer*. NIST Technical Publication

FINGERHUTH, M.; BABEJ, T.; WITTEK, P. Open source software in quantum computing. *PLOS ONE*, v. 13, n. 12, e0208561, dec. 2018.

KHARKOV, Y.; IVANOVA, A.; MIKHANTIEV, E.; KOTELNIKOV, A. Arline Benchmarks: Automated Benchmarking Platform for Quantum Compilers. *arXiv preprint arXiv:2202.14025*, fev. 2022.

POWERS, C. et al. MISTIQS: An open-source software for performing quantum many-body dynamics. *SoftwareX*, vol. 15, 2021.

A. KAHNG et al. “The OpenROAD Project: Unleashing Hardware Innovation”. *Proceedings of the 2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, “circa 2020”.

“OpenLANE: The Open-Source Digital ASIC Implementation Flow”. A21 Workshop (2020) (H. Dai et al.)

ANEXO A – CÓDIGO DO CORDIC VETORIAL

```
module CORDIC_vec #(parameter width = 16)(
  input wire clock,
  input wire signed [width-1:0] x_start,
  input wire signed [width-1:0] y_start,
  output wire signed [width-1:0] magnitude,
  output wire signed [31:0] phase // Q1.31
);

  // Declara a tabela como reg para poder inicializar
  reg signed [31:0] atan_table [0:width-1];

  // Inicializa os valores da tabela no bloco initial
  initial begin
    atan_table[0] = 32'sd843314857;
    atan_table[01] = 32'sd497837829;
    atan_table[02] = 32'sd263043836;
    atan_table[03] = 32'sd133525159;
    atan_table[04] = 32'sd67021687;
    atan_table[05] = 32'sd33543516;
    atan_table[06] = 32'sd16775851;
    atan_table[07] = 32'sd8388437;
    atan_table[08] = 32'sd4194285;
    atan_table[09] = 32'sd2097149;
    atan_table[10] = 32'sd1048575;
    atan_table[11] = 32'sd524287;
    atan_table[12] = 32'sd262143;
    atan_table[13] = 32'sd131071;
    atan_table[14] = 32'sd65535;
    atan_table[15] = 32'sd32767;
end

  reg signed [width:0] x [0:width-1];
```

```
 reg signed [width:0] y [0:width-1];
 reg signed [31:0] z [0:width-1];

 always @(posedge clock) begin
  x[0] <= x_start;
  y[0] <= y_start;
  z[0] <= 0;
 end

 genvar i;
 generate
  for (i = 0; i < width-1; i = i + 1) begin : stage
   wire signed [width:0] x_shift = x[i] >>> i;
   wire signed [width:0] y_shift = y[i] >>> i;

   always @(posedge clock) begin
  if (y[i] >= 0) begin
     x[i+1] <= x[i] + y_shift;
     y[i+1] <= y[i] - x_shift;
     z[i+1] <= z[i] - atan_table[i];
  end
  else begin
     x[i+1] <= x[i] - y_shift;
     y[i+1] <= y[i] + x_shift;
     z[i+1] <= z[i] + atan_table[i];
     end
   end
  end
 endgenerate

 assign magnitude = x[width-1][width-1:0];
 assign phase = z[width-1];

endmodule
```

ANEXO B – CÓDIGO DO CORDIC ROTACIONAL

```
module cordic_rot(clock, cosine, sine, x_start, y_start, angle);

  parameter width = 16;

  // Inputs
  input clock;
  input signed [width-1:0] x_start,y_start;
  input signed [31:0] angle;

  // Outputs
  output signed  [width-1:0] sine, cosine;

  wire signed [31:0] atan_table [0:30];

  assign atan_table[00] = 'b00100000000000000000000000000000;
  assign atan_table[01] = 'b00010010111001000000010100011101;
  assign atan_table[02] = 'b00001001111111011001110000101l011;
  assign atan_table[03] = 'b00000101000100010001000111010100;
  assign atan_table[04] = 'b00000010100010110000110101000011;
  assign atan_table[05] = 'b00000001010001011101011111100001;
  assign atan_table[06] = 'b00000000101000101111011000011110;
  assign atan_table[07] = 'b00000000010100010111110001010101;
  assign atan_table[08] = 'b00000000001010001011111001010011;
  assign atan_table[09] = 'b00000000000101000101111100101110;
assign atan_table[10] = 'b00000000000010100010111110011000;
  assign atan_table[11] = 'b00000000000001010001011111001100;
  assign atan_table[12] = 'b00000000000000101000101111100110;
  assign atan_table[13] = 'b00000000000000010100010111110011;
  assign atan_table[14] = 'b00000000000000001010001011111001;
  assign atan_table[15] = 'b00000000000000000101000101111100;
  assign atan_table[16] = 'b00000000000000000010100010111110;
  assign atan_table[17] = 'b00000000000000000001010001011111;
```

```
assign atan_table[18] = 'b00000000000000000000101000101111;
assign atan_table[19] = 'b00000000000000000000010100010111;
assign atan_table[20] = 'b00000000000000000000001010001011;
assign atan_table[21] = 'b00000000000000000000000101000101;
assign atan_table[22] = 'b00000000000000000000000010100010;
assign atan_table[23] = 'b00000000000000000000000001010001;
assign atan_table[24] = 'b00000000000000000000000000101000;
assign atan_table[25] = 'b00000000000000000000000000010100;
assign atan_table[26] = 'b00000000000000000000000000001010;
assign atan_table[27] = 'b00000000000000000000000000000101;
assign atan_table[28] = 'b00000000000000000000000000000010;
assign atan_table[29] = 'b00000000000000000000000000000001;
assign atan_table[30] = 'b00000000000000000000000000000000;

reg signed [width:0] x [0:width-1];
reg signed [width:0] y [0:width-1];
reg signed    [31:0] z [0:width-1];

wire [1:0] quadrant;
assign quadrant = angle[31:30];

always @(posedge clock)
begin // make sure the rotation angle is in the -pi/2 to pi/2 range
  case(quadrant)
    2'b00,
    2'b11: // no changes needed for these quadrants
    begin
      x[0] <= x_start;
      y[0] <= y_start;
      z[0] <= angle;
    end

    2'b01:
    begin
```

```
    x[0] <= -y_start;
    y[0] <= x_start;
    z[0] <= {2'b00,angle[29:0]}; // subtract pi/2 for angle in this quadrant
   end

   2'b10:
   begin
    x[0] <= y_start;
    y[0] <= -x_start;
    z[0] <= {2'b11,angle[29:0]}; // add pi/2 to angles in this quadrant
   end
  endcase
end
genvar i;

generate
for (i=0; i < (width-1); i=i+1)
begin: xyz
  wire z_sign;
  wire signed [width:0] x_shr, y_shr;

  assign x_shr = x[i] >>> i; // signed shift right
  assign y_shr = y[i] >>> i;

  //the sign of the current rotation angle
  assign z_sign = z[i][31];

  always @(posedge clock)
  begin
   // add/subtract shifted data
   x[i+1] <= z_sign ? x[i] + y_shr : x[i] - y_shr;
   y[i+1] <= z_sign ? y[i] - x_shr : y[i] + x_shr;
   z[i+1] <= z_sign ? z[i] + atan_table[i] : z[i] - atan_table[i];
  end
```

```
  end
  endgenerate

  // assign output
  assign cosine = x[width-1];
  assign sine = y[width-1];

endmodule
```

ANEXO C – CÓDIGO DO TESTBENCH VETORIAL

```
`timescale 1ns/1ps
module tb_CORDIC_vectorial;

  reg clock = 0;
  reg signed [15:0] x_start, y_start;
  wire signed [15:0] magnitude;
  wire signed [31:0] phase;

  // Conversões para visualização
  real x_real, y_real, magnitude_real, phase_rad, magnitude_dB, phase_deg;

  // DUT
  CORDIC_vec #(16) dut (
    .clock(clock),
    .x_start(x_start),
    .y_start(y_start),
    .magnitude(magnitude),
    .phase(phase)
  );

  always #5 clock = ~clock;

  initial begin
    $dumpfile("cordic.vcd");
    $dumpvars(0, tb_CORDIC_vectorial);

    x_start = 16'sd0; y_start = 16'sd0;
    #20;
    // Amostras
    send_sample(16'sd16384, 16'sd0);      // 0°
    send_sample(16'sd11585, 16'sd11585);  // 45°
    send_sample(16'sd0, 16'sd16384);      // 90°
```

```
    send_sample(-16'sd11585, 16'sd11585); // 135°
    send_sample(-16'sd16384, 16'sd0);     // 180°
    send_sample(-16'sd11585, -16'sd11585);// -135°
    send_sample(16'sd0, -16'sd16384);     // -90°
    send_sample(16'sd11585, -16'sd11585); // -45°
    send_sample(16'sd16384, 16'sd0);      // 0° novamente

    #1000;
    $finish;
  end

  task send_sample;
    input signed [15:0] xi;
    input signed [15:0] yi;
    begin
      x_start = xi;
      y_start = yi;

      // Espera a convergência do CORDIC (~16 ciclos + margem)
      repeat (20) @(posedge clock);
      // Converte para ponto flutuante para observação
      x_real = xi / 32768.0;
      y_real = yi / 32768.0;
      magnitude_real = magnitude / 32768.0;
      phase_rad = $itor(phase) / (2.0**31); // Q1.31 -> [-1, 1) rad
      magnitude_dB = 20 * $log10(magnitude_real);
      phase_deg = phase_rad * 180.0;

      $display("I = %.4f, Q = %.4f => Magnitude = %.4f (%.2f dB), Fase = %.2f°",
        x_real, y_real, magnitude_real, magnitude_dB, phase_deg);
      #20;
    end
  endtask
endmodule
```

ANEXO D – CÓDIGO DO TESTBENCH ROTACIONAL

```
`timescale 1ns/1ps
module tb_cordic_rot;

  parameter width = 16;

  // Clock
  reg clk = 0;
  always #10 clk = ~clk; // 50 MHz

  // Entradas
  reg signed [width-1:0] Xin, Yin;
  reg signed [31:0] angle;

  // Saídas
  wire signed [width-1:0] COSout, SINout;
  // DUT
  cordic_rot #(.width(width)) dut (
    .clock(clk),
    .x_start(Xin),
    .y_start(Yin),
    .angle(angle),
    .cosine(COSout),
    .sine(SINout)
  );
  // Tarefa para aplicar um vetor de teste
  task apply_input;
    input signed [width-1:0] x_in, y_in;
    input [31:0] theta;
    begin
      Xin = x_in;
      Yin = y_in;
      angle = theta;
```

```
    end
  endtask
  initial begin
    $dumpfile("tb_cordic_rot.vcd");
    $dumpvars(0, tb_cordic_rot);
    // Inicializa entradas
    Xin = 0;
    Yin = 0;
    angle = 0;
    repeat (2) @(posedge clk);
    // Aplica testes
    apply_input(16'sd16384, 16'sd0, 32'd0);            // 0°
    repeat (32) @(posedge clk);
    apply_input(16'sd11585, 16'sd11585, 32'd715827883); // 45°
    repeat (32) @(posedge clk);
    apply_input(16'sd0, 16'sd16384, 32'd1073741824);    // 90°
    repeat (32) @(posedge clk);
    apply_input(-16'sd11585, 16'sd11585, 32'd1610612736); // 135°
    repeat (32) @(posedge clk);
    apply_input(-16'sd11585, 16'sd11585, 32'd1610612736); // 135°
    repeat (32) @(posedge clk);
    apply_input(-16'sd16384, 16'sd0, 32'd2147483648);   // 180°
    repeat (32) @(posedge clk);
    apply_input(-16'sd11585, -16'sd11585, 32'd2684354560); // 225°
    repeat (32) @(posedge clk);
    apply_input(16'sd0, -16'sd16384, 32'd3221225472);   // 270°
    repeat (32) @(posedge clk);
    apply_input(16'sd11585, -16'sd11585, 32'd3758096384); // 315°
    repeat (32) @(posedge clk);

    $finish;
  end

endmodule
```

ANEXO E – CÓDIGO DO WRAPPER

```
// tt_um_cordic_wrapper.v — TinyTapeout topo (Verilog-2001)
module tt_um_cordic_wrapper #(parameter integer WIDTH = 16)
(
   input  wire       clk,
   input  wire       rst_n,
   input  wire       ena,
   input  wire [7:0]  ui_in,
   output wire [7:0]  uo_out,
   input  wire [7:0]  uio_in,
   output wire [7:0]  uio_out,
   output wire [7:0]  uio_oe
);

   // ------------------------------
   // --- Configuração de latência ---
   // ------------------------------
   localparam integer PIPE_LAT = WIDTH;          // nº de ciclos até saída válida
   reg  [PIPE_LAT:0]  latency_shifter;          // 1 bit extra
   wire               result_is_valid;

   // ------------------------------
   // --- Handshake I/O (uio) ---
   // ------------------------------
   wire in_valid;
   reg  in_ready;
   reg  out_valid;
   wire out_ready;

   // uio[2]=out_valid (saída), uio[1]=in_ready (saída)
   assign uio_oe       = 8'b0000_0110;
   assign in_valid     = uio_in[0];
   assign out_ready    = uio_in[3];
```

```
assign uio_out[1]   = in_ready;
assign uio_out[2]   = out_valid;
assign uio_out[7:3] = 5'b0;
assign uio_out[0]   = 1'b0;

// Consumir bits de uio_in não usados (silencia verilator UNUSED)
wire _unused_uio_in;
assign _unused_uio_in = &{1'b0, uio_in[7:4], uio_in[2:1]};

// ------------------------------
// --- Estados Entrada/Saída ---
// ------------------------------
// Entrada
localparam [2:0]
   S_IN_IDLE  = 3'd0,
   S_IN_X_MSB = 3'd1,
   S_IN_Y_LSB = 3'd2,
   S_IN_Y_MSB = 3'd3,
   S_IN_WAIT  = 3'd4;
reg [2:0] in_state;

// Saída
localparam [2:0]
   S_OUT_IDLE     = 3'd0,
   S_OUT_MAG_LSB  = 3'd1,
   S_OUT_MAG_MSB  = 3'd2,
   S_OUT_PHASE_B0 = 3'd3,
   S_OUT_PHASE_B1 = 3'd4,
   S_OUT_PHASE_B2 = 3'd5,
   S_OUT_PHASE_B3 = 3'd6;
reg [2:0] out_state;

// ------------------------------
// --- Registradores de dados ---
```

```
// ------------------------------
reg  signed [WIDTH-1:0] x_input_reg;
reg  signed [WIDTH-1:0] y_input_reg;
reg  signed [WIDTH-1:0] magnitude_reg;
reg  signed [31:0]      phase_reg;
reg  [7:0]              uo_out_reg;

reg start_cordic_pipeline;
reg result_ready_for_tx;

// ------------------------------
// --- Núcleo CORDIC ---
// ------------------------------
wire signed [WIDTH-1:0] cordic_mag_out;
wire signed [31:0]      cordic_phase_out;

CORDIC_vec #(.width(WIDTH)) u_cordic_core (
   .clock     (clk),
   .x_start   (x_input_reg),
   .y_start   (y_input_reg),
   .magnitude (cordic_mag_out),
   .phase     (cordic_phase_out)
);

// Busy flags
wire pipeline_busy = |latency_shifter;
wire tx_busy       = (out_state != S_OUT_IDLE) | result_ready_for_tx;
wire core_busy     = pipeline_busy | tx_busy;

always @(posedge clk or negedge rst_n) begin
   if (!rst_n) begin
      in_state             <= S_IN_IDLE;
      in_ready             <= 1'b1;
      start_cordic_pipeline <= 1'b0;
```

```
    x_input_reg          <= {WIDTH{1'b0}};
    y_input_reg          <= {WIDTH{1'b0}};
end else if (ena) begin
    start_cordic_pipeline <= 1'b0;
    in_ready             <= (in_state != S_IN_WAIT);

    case (in_state)
        S_IN_IDLE: begin
            if (in_valid & in_ready) begin
                x_input_reg[7:0] <= ui_in;
                in_state <= S_IN_X_MSB;
            end
        end
        S_IN_X_MSB: begin
            if (in_valid & in_ready) begin
                x_input_reg[15:8] <= ui_in;
                in_state <= S_IN_Y_LSB;
            end
        end
        S_IN_Y_LSB: begin
            if (in_valid & in_ready) begin
                y_input_reg[7:0] <= ui_in;
                in_state <= S_IN_Y_MSB;
            end
        end
        S_IN_Y_MSB: begin
            if (in_valid & in_ready) begin
                y_input_reg[15:8] <= ui_in;
                start_cordic_pipeline <= 1'b1; // dispara o CORDIC
                in_state <= S_IN_WAIT;
            end
        end
        S_IN_WAIT: begin
            if (!core_busy)
```

```
                in_state <= S_IN_IDLE;
            end
            default: begin
                // Cobertura total para o linter
                in_state <= S_IN_IDLE;
            end
        endcase
    end
end

// ------------------------------
// --- Controle de latência ---
// ------------------------------
assign result_is_valid = latency_shifter[PIPE_LAT];

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        latency_shifter    <= {(PIPE_LAT+1){1'b0}};
        magnitude_reg      <= {WIDTH{1'b0}};
        phase_reg          <= 32'sd0;
        result_ready_for_tx <= 1'b0;
    end else if (ena) begin
        latency_shifter <= {latency_shifter[PIPE_LAT-1:0], start_cordic_pipeline};

        if (result_is_valid) begin
            magnitude_reg      <= cordic_mag_out;
            phase_reg          <= cordic_phase_out;
            result_ready_for_tx <= 1'b1;
        end

        if (out_state != S_OUT_IDLE)
            result_ready_for_tx <= 1'b0;
    end
end
```

```
assign uo_out = uo_out_reg;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        out_state  <= S_OUT_IDLE;
        out_valid  <= 1'b0;
        uo_out_reg <= 8'h00;
    end else if (ena) begin
        case (out_state)
            S_OUT_IDLE: begin
                out_valid  <= 1'b0;
                uo_out_reg <= 8'h00;
                if (result_ready_for_tx) begin
                    out_state  <= S_OUT_MAG_LSB;
                    uo_out_reg <= magnitude_reg[7:0];
                    out_valid  <= 1'b1;
                end
            end
            S_OUT_MAG_LSB: if (out_ready) begin
                out_state  <= S_OUT_MAG_MSB;
                uo_out_reg <= magnitude_reg[15:8];
            end
            S_OUT_MAG_MSB: if (out_ready) begin
                out_state  <= S_OUT_PHASE_B0;
                uo_out_reg <= phase_reg[7:0];
            end
            S_OUT_PHASE_B0: if (out_ready) begin
                out_state  <= S_OUT_PHASE_B1;
                uo_out_reg <= phase_reg[15:8];
            end
            S_OUT_PHASE_B1: if (out_ready) begin
                out_state  <= S_OUT_PHASE_B2;
                uo_out_reg <= phase_reg[23:16];
```

```
            end
            S_OUT_PHASE_B2: if (out_ready) begin
               out_state  <= S_OUT_PHASE_B3;
               uo_out_reg <= phase_reg[31:24];
            end
            S_OUT_PHASE_B3: if (out_ready) begin
               out_state  <= S_OUT_IDLE;
               out_valid  <= 1'b0;
               uo_out_reg <= 8'h00;
            end
            default: begin
               // Cobertura total para o linter
               out_state  <= S_OUT_IDLE;
               out_valid  <= 1'b0;
               uo_out_reg <= 8'h00;
            end
         endcase
      end
   end
endmodule
```