UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA

JUCEMAR PAES NETO

# PROJETO DE FONTE DE CORRENTE SENOIDAL PARA MEDIDAS DE SINAL DE BIOIMPEDÂNCIA

Florianópolis
2023

Jucemar Paes Neto

# PROJETO DE FONTE DE CORRENTE SENOIDAL DIGITAL PARA MEDIDAS DE SINAL DE BIOIMPEDÂNCIA

Trabalho de Conclusão de Curso submetido ao curso de Graduação em Engenharia Eletrônica do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharel em Engenharia Eletrônica.

Orientador: Prof. Márcio Cherem Schneider, Dr.

Florianópolis

2023

Paes Neto, Jucemar
PROJETO DE FONTE DE CORRENTE SENOIDAL DIGITAL PARA MEDIDAS DE SINAL DE BIOIMPEDÂNCIA / Jucemar Paes Neto ; orientador, Márcio Schneider, 2023.
72 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Eletrônica, Florianópolis, 2023.

Inclui referências.

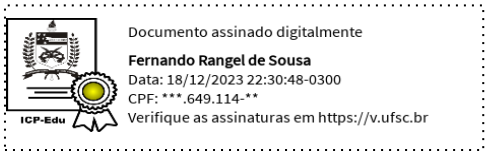
1. Engenharia Eletrônica. 2. Síntese Digital direta. 3. Bioimpedância. 4. Driver de Corrente. I. Schneider, Márcio. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Eletrônica. III. Título.

Jucemar Paes Neto

# PROJETO DE FONTE DE CORRENTE SENOIDAL DIGITAL PARA MEDIDAS DE SINAL DE BIOIMPEDÂNCIA

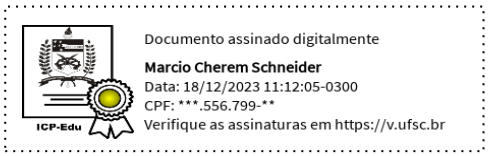
Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel em Engenharia Eletrônica e aprovado em sua forma final pelo Curso Engenharia Eletrônica.

Florianópolis, 14 de dezembro de 2023.



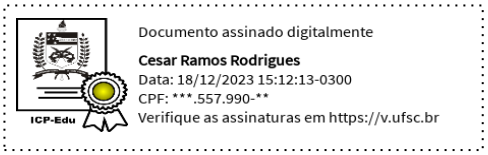

Coordenação do Curso

**Banca examinadora**



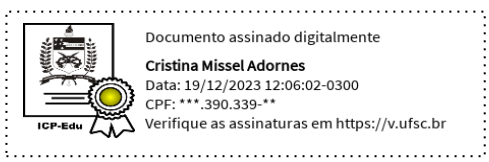

Prof. Márcio Cherem Schneider, Dr.

Orientador




Prof. César Ramos Rodrigues, Dr.

UFSC




Cristina Missel Adornes, Ma.

UFSC

Florianópolis, 2023.

Dedico este trabalho aos meus pais e meus avós, sem os quais jamais teria chegado longe o bastante para fazê-lo.

## AGRADECIMENTOS

Agradeço ao meu orientador professor Márcio Schneider, Cristina Adornes e o professor César Rodrigues a ajuda no desenvolvimento deste trabalho. Agradeço também aos professores Fabian Riaño e Hector Pettenghi do Laboratório de Sistemas Digitais pelo auxílio no uso de ferramentas e disponibilização de equipamentos. E ainda a todos os colegas do Laboratório de Circuitos Integrados pelo apoio.

“Mas a densidade e crescimento vertiginoso nas velocidades e densidades de circuitos eletrônicos puramente digitais trouxe uma quebra de paradigma, que é: o uso de condicionamento e processamento digitais para quase todas as grandezas “analógicas”. [...] A conclusão é que as técnicas de conversão A/D e D/A se tornaram fundamentais para todos os aspectos de medição e controle analógico”
(HOROWITZ e HILL, 2017, p. 879)

## RESUMO

Equipamentos de medida para aplicações médicas não invasivas se tornam cada vez mais comuns para realizar diagnósticos de patologias e a integração de *wearables* no cotidiano das pessoas se torna cada vez mais comum. Além disso, análises por processamento de imagem e a disseminação de tecnologias de *machine learning* tem tornado técnicas como a Tomografia de Impedância Elétrica alternativas viáveis para diagnósticos de lesões ortopédicas e até mesmo ensaios em tempo real para atletismo de alto desempenho. Contudo, para atingir tais objetivos de dispositivos altamente não intrusivos e que realizem medidas confiáveis são necessários um alto gral de integração de componentes eletrônicos e técnicas adequadas de instrumentação. Por isso, esse trabalho propõe-se a desenvolver um *driver* de corrente para a realização de medidas de bioimpedância através do método de Síntese Digital Direta. Isso porque essa técnica possui diversas vantagens sobre suas contrapartes analógicas, como redução na dependência da temperatura, controle superior sobre a frequência de saída e ajuste fino sobre a fase através de controle digital além de permitir alto grau de integração por não requerer componentes reativos dispendiosos em área. Utilizando essa técnica e através de otimizações das lógicas e circuitos digitais foi obtido um circuito capaz de fornecer uma saída com erro de frequência de 5,95 ppm com *clock* máximo de 512,5 MHz e distorção harmônica total de 0,258%.



**Palavras-chave**: Bioimpedância; Síntese Digital Direta; Driver de Corrente.

## ABSTRACT

Measurement equipment for non-invasive medical applications is becoming increasingly common to diagnose pathologies and the integration of wearables into people's daily lives is becoming more and more common. In addition, image processing analyzes, and the spread of machine learning technologies have made techniques such as Electrical Impedance Tomography viable alternatives for diagnosing orthopedic injuries and even real-time tests for high-performance athletics. However, to achieve such goals of highly non-intrusive devices that perform reliable measurements, a high degree of integration of electronic components and adequate instrumentation techniques are required. Therefore, this work proposes to develop a current driver to carry out bioimpedance measurements through the Direct Digital Synthesis method. This is because this technique has several advantages over its analog counterparts, such as reduced temperature dependence, superior control over the output frequency and fine adjustment over the phase through digital control, in addition to allowing a high degree of integration by not requiring area consuming reactive components. Using this technique and trough logical and digital circuitry optimizations the circuit achieved was able to supply an output with 5,95 ppm frequency error with a maximum clock of 512,5 MHz and total harmonic distortion of 0,258%.



**Keywords**: Bioimpedance; Direct Digital Synthesis; Current Driver.

## LISTA DE FIGURAS

## LISTA DE QUADROS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| AC | *Alternate Current* (Corrente Alternada) |
| DA | Digital-Analógico |
| DAC | *Digital to Analog Converter* (Conversor Digital Analógico) |
| DC | *Direct Current* (Corrente Direta ou Contínua) |
| DDS | *Direct Digital Synthesizer* (Sintetizador Digital Direto) |
| FSM | *Finite State Machine* (Máquina de Estados Finitos) |
| LUT | *Look Up Table* (Tabela de Consulta) |
| THD | *Total Harmonic Distorsion* (Distorção Harmônica Total) |
| ROM | *Read Only Memory* (Memória Somente de Leitura) |
| RTL | *Register Transfer Level* (Nível de Transferência Entre Registradores) |
| SoC | *System on Chip* (Sistema em um chip) |
| VHDL | *VHSIC Hardware Description Language* (Linguagem de Descrição de *Hardware* VHSIC) |
| VHSIC | *Very High Speed Integrated Circuits* (Circuitos Integrados de Alta Velocidade) |

**SUMÁRIO**

# 1 INTRODUÇÃO

Durante os últimos anos, diversas técnicas de análise e diagnóstico para uso médico através de métodos não invasivos têm sido pesquisadas e desenvolvidas (BERA, 2014), uma vez que esses métodos são menos incômodos ao paciente e abrem possibilidades de avaliações médicas em regimes fora do ambiente clínico-hospitalar. Além disso, o uso de *wearables* tem crescido substancialmente, pois permitem não só diagnósticos não invasivos como também medidas de sinais biomédicos em tempo real e fornecem ao paciente alternativas acessíveis de *home care*(LIN, SONG, *et al.*, 2021).

Junto a isso, tecnologias de processamento de imagem e técnicas de tomografia computadorizada se tornaram amplamente difundidas na área da saúde e são cada vez mais utilizadas para fornecer informações relevantes para o tratamento de doenças (SMITH-BINDMAN, MIGLIORETTI e LARSON, 2008). Contudo, esse crescente uso de imagens também promove um risco à saúde, uma vez que grande parte desses equipamentos de imagem fazem uso intensivo de radiografia, cujo uso indiscriminado é causa de diversos efeitos adversos à saúde do paciente (EINSTEIN, 2009).

Nesse contexto, portanto, pode-se destacar que vários métodos utilizados para realizar medidas de forma não invasiva sem uso de radiação ionizante envolvem a medida de bioimpedâncias (bioZ), tais quais: Análise de Impedância Bioelétrica (BIA), Espectroscopia de Impedância Elétrica (IPG), Cardiografia de Impedância (ICG) e Tomografia de Impedância Elétrica (EIT). Todos esses métodos fazem uso de alguma forma da medida de uma impedância associada a um fenômeno fisiológico ao qual se deseja monitorar ou diagnosticar.

Sendo a impedância uma grandeza dada pela razão entre tensão e corrente, é possível realizar sua medida indireta aplicando uma tensão conhecida sobre o elemento de interesse e realizando a medida da corrente que flui por ele, ou alternativamente, injetar uma corrente pré-determinada no elemento e realizar a medida da queda de tensão que ocorre sobre ele. Dessas duas alternativas, a posterior é a mais utilizada nesse âmbito, visto que se deseja ter controle sobre o fluxo de cargas elétricas que passa pelos tecidos do objeto em estudo, haja vista que este é um potencial fator de danos. Além disso, a medida de tensão costuma ser mais

natural ao se trabalhar com circuitos integrados, uma vez que a medida de corrente requer uma conversão corrente-tensão para o processamento da informação.

Como a impedância biológica possui tanto componente real como imaginária, não bastaria realizar a medida apenas em corrente contínua. Mas não só isso, a aplicação de corrente contínua pode causar danos fisiológicos e é desincentivada para realizar a medida de sinais bioZ.

Portanto, é necessário um *driver* de corrente alternada para realizar tais medidas. Este, deve, contudo, utilizar uma fonte de corrente alternada com significativa precisão em termos de frequência, fase e amplitude, bem como, ser de uma topologia que permita alto grau de integração, a fim de que possa ser utilizada em um SoC que naturalmente será povoado por outros blocos de processamento e aquisição de dados. Esses requerimentos são bem atendidos pela topologia de Síntese Digital Direta; tal método será abordado em mais detalhes adiante, mas consiste, essencialmente, em uma memória com valores de amplitude de uma forma de onda discretizada a qual é acessada sequencialmente e repetidamente. Outra vantagem dessa forma de aplicação é que a frequência do sinal de saída depende apenas da frequência de acesso à memória, a qual pode também ser facilmente alterada utilizando divisores de *clock*, permitindo, assim, varreduras em frequência que podem ser úteis para a análise de um grande número de condições médicas (ANALOG DEVICES, 1999).

Dito isso, o presente trabalho se propõe a desenvolver um *driver* de corrente para realizar a excitação do elemento a ser medido. Esse *driver*, portanto, se trata de uma fonte de corrente senoidal com baixa amplitude de saída e baixa distorção.

## 1.1 OBJETIVO GERAL

Desenvolver o projeto de um sintetizador digital integrável que possa, através de um conversor digital analógico, produzir uma corrente senoidal de pelo menos 50 kHz para excitar um dispositivo medidor de bioimpedâncias.

## 1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desse trabalho são:

- Determinar a topologia de projeto do sintetizador digital.

- Descrever os blocos necessários para sua implementação.
- Otimizar o sistema em termos de área e frequência.
- Validar os resultados através de simulação.

# 2 FUNDAMENTAÇÃO TEÓRICA

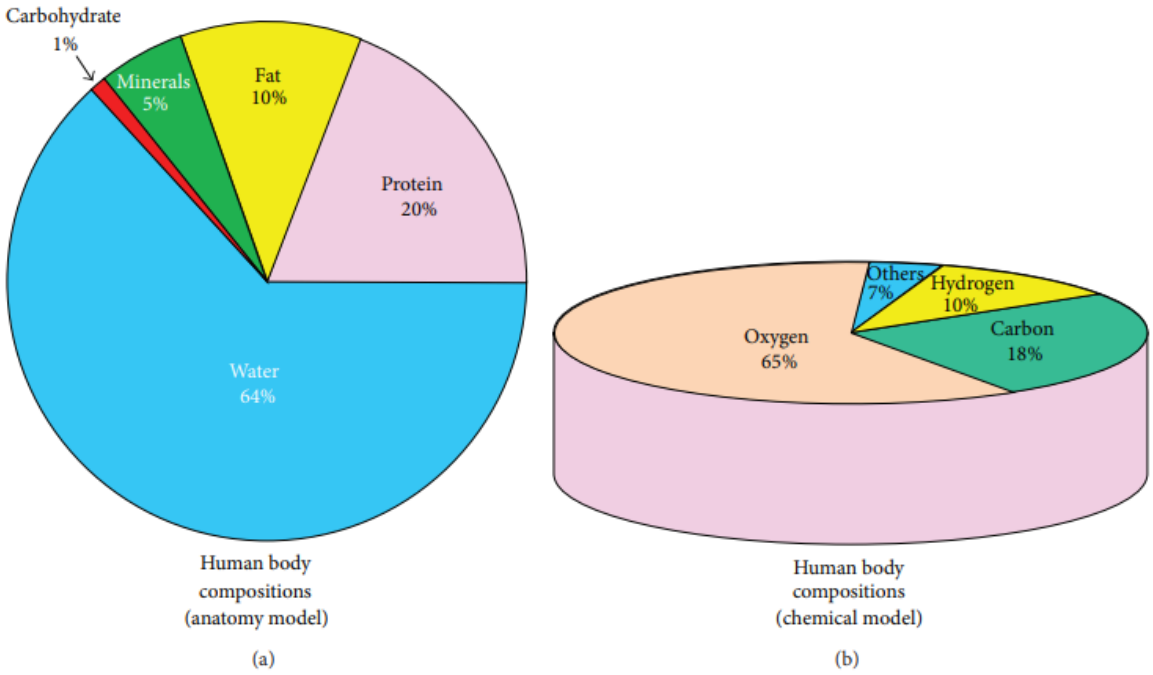
Nesse capítulo serão abordados os fundamentos teóricos que justificam as especificações do projeto bem como o princípio de funcionamento da técnica de Síntese Digital Direta e os impactos causados pelas escolhas da arquitetura do sintetizador.

## 2.1 MEDIDA DE BIOIMPEDÂNCIA

As estruturas orgânicas fisiológicas que compõem o organismo humano são extremamente complexas. É possível a um nível atômico observar a composição dos elementos fundamentais que compõem um indivíduo típico, sendo Oxigênio, Carbono, Hidrogênio e Nitrogênio responsáveis por mais de 95% da constituição total dos elementos de um ser humano, ainda que esse possua outros elementos em frações menores como cloro, fósforo, enxofre entre outros elementos ametais e ainda elementos metálicos como potássio, ferro e sódio para citar alguns. Os seres humanos, contudo, não são apenas um amontoado aleatório destes componentes. Eles interagem e se organizam em moléculas e outras estruturas que podem ser classificadas em cinco grandes grupos de estruturas: lipídios, água, proteínas, carboidratos e minerais (HEYMSFIELD, WANG, *et al.*, 1997).

No nível molecular é possível observar as diferenças entre as características elétricas das diferentes estruturas que compõem o corpo humano, desde compostos com características fortemente dielétricas como carboidratos formados por elementos unidos por ligações covalentes, as quais têm característica isolante, a exemplo de polissacarídeos bem como a presença de íons solutos em água como potássio e sódio que, em contraste, se tornam soluções de alta condutividade elétrica.

Figura 1 - Distribuição dos compostos e elementos químicos no corpo humano




Fonte: (BERA, 2014).

Esses compostos, então, se organizam em células e tecidos que, por sua vez, dão às estruturas atômicas características elétricas específicas e criam caracterizações particulares para cada estrutura anatômica humana, que podem então ser caracterizadas através dessas propriedades. Desse modo, é possível realizar uma observação de estruturas biológicas através da chamada bioimpedância que se trata da impedância elétrica observada em um sistema biológico.

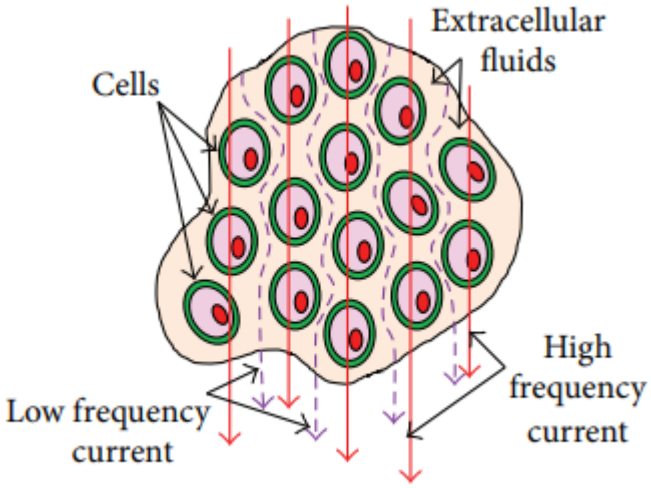
Contudo, a análise dessas características não é algo trivial, pois vale lembrar que essa distribuição de compostos químicos está organizada em um meio tridimensional com uma geometria complexa. E, como mostra a Figura 1, a presença de elementos de elevada condutividade em difusão com elementos essencialmente isolantes apresenta-se a um observador de impedância como um sistema não homogêneo constituído de resistências e reatâncias devido aos efeitos capacitivos presentes nesse meio.

Além disso, como nos tecidos há uma organização estrutural das células, bem como das membranas e fluídos que compõem os órgãos e sistemas, e essas células possuem geometrias e orientações específicas, as propriedades elétricas apresentadas por elas não são homogêneas ao longo da distribuição do organismo e, muitas vezes, devido às orientações celulares, apresentam comportamento

anisotrópico (MARTINSEN, GRIMNES e SCHWAN, 2002). A Figura 2 ilustra como a dispersão de estruturas dentro das células podem causar variações no comportamento entre correntes de alta frequência e baixa frequência devido as propriedades elétricas presentes nela.

Esses elementos tornam necessária a medida da bioimpedância através de corrente alternada para a determinação das componentes real e imaginária, obtidas através da medida da diferença de fase entre corrente injetada e tensão medida. Uma estimação desses parâmetros a 50 kHz é típica e também traz informações referentes a outros fenômenos presentes nos organismos biológicos como dispersão e relaxação dielétrica. Portanto, essa será a mínima frequência na qual este projeto deverá operar (BERA, 2014).

Figura 2 – Ilustração da distribuição de correntes por uma célula orgânica



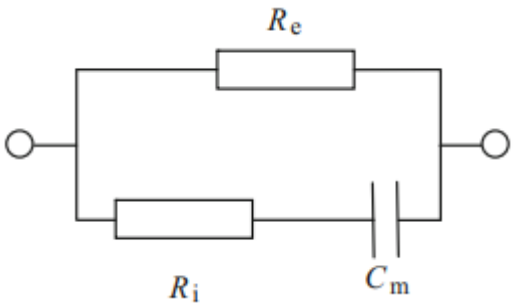

Fonte: (BERA, 2014).

Outra consideração importante ao se realizar a excitação por corrente para medida de bioimpedância diz respeito aos níveis de corrente com que o *driver* deve atuar. Como a medida de impedância é realizada de forma ativa, visto que se trata de uma grandeza que é medida baseada em seu efeito modulante (WOLFFENBUTTEL, 2006) requer-se uma excitação externa do objeto em estudo. Porém, ao mesmo tempo, a excitação necessária para realizar a medida deve ser ponderada uma vez que será realizada sobre um organismo vivo. E tal excitação pode causar efeitos

adversos e provocar deterioração dos tecidos (KOUWENHOVEN, 1949) algo que deve ser mitigado.

Portanto, a corrente elétrica que deverá fluir deve ser da ordem de dezenas de microampères até poucos miliamperes, o que pode trazer dificuldades para realizar a medida da tensão posteriormente, uma vez que os níveis de tensão serão proporcionalmente baixos trazendo assim complicações como ruído e polarização DC. Além do problema de instrumentação, uma polarização DC também poderia implicar em um acúmulo de cargas na amostra. Utilizando o modelo apresentado por González-Correa (2018), os tecidos biológicos podem ser modelados como um circuito equivalente de dois resistores e um capacitor como mostra a Figura 3. Assim, na eventual existência de uma corrente contínua $I_{DC}$ não nula, decorrido um tempo $t_1$, haverá um acúmulo de $I_{DC}t_1$ Coulombs de carga no tecido devido ao efeito da componente capacitiva. Enquanto uma corrente senoidal de pico $I_P$ e período $T$ provocaria no máximo um acúmulo de cargas de $I_P T/\pi$. Por exemplo então, uma corrente senoidal de frequência 50 kHz e pico de 20 μA provocaria um acúmulo de carga máximo de 12,7 nC, enquanto uma corrente contínua de mesmo valor eficaz levaria apenas 89,8 ms para acumular a mesma carga.

Figura 3 – Modelo elétrico dos tecidos biológicos




Fonte: (GONZÁLEZ-CORREA, 2018)

Para mitigar esses problemas, o sistema digital projetado neste trabalho prevê o uso de um *switch* diferencial na saída do conversor digital analógico para prover à carga uma corrente totalmente diferencial, evitando assim os problemas anteriormente citados. Esse módulo será mais bem explicado nos tópicos seguintes.

## 2.2 SÍNTESE DIGITAL DIRETA

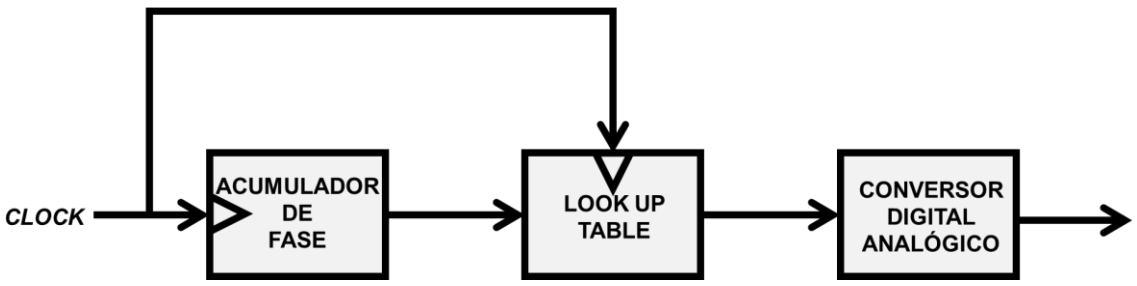
O método escolhido para implementar a fonte de sinal que irá excitar o tecido biológico em análise foi o de síntese digital direta, capaz de produzir digitalmente sinais periódicos, contínuos no tempo. Este método foi proposto pela primeira vez em Tierney, Rader e Gold (1971) com a intenção de ser uma alternativa às fontes de frequência com implementação analógica. Essa proposta tinha o propósito não só de realizar a implementação dessas fontes de forma mais simples utilizando circuitos digitais, mas também possuía uma série de vantagens sobre suas contrapartes analógicas como por exemplo baixo ruído de fase, alta estabilidade e controle sobre a fase e grande configurabilidade.

Como a implementação digital não está associada à ressonância de componentes, que em geral se projeta em circuitos analógicos de sintetização de frequências, o tom gerado por um sintetizador digital depende exclusivamente do sinal de relógio que o controla; sendo assim, simples circuitos de divisão de *clock* podem ser usados (e até configurados através de registradores e microcontroladores) para se variar a frequência desses geradores em tempo de execução.

O DDS (*Direct Digital Synthesis*, em inglês, síntese digital direta) utiliza essencialmente três macro blocos funcionais:

- Acumulador de fase;
- *Look Up Table;*
- Conversor Digital Analógico.

Figura 4 – Diagrama de blocos de um DDS




Fonte: O autor.

Apesar de o terceiro bloco não estar no escopo deste trabalho, o projeto deste DDS se preocupará com sua implementação incluindo certas características que facilitarão seu projeto, como será melhor discutido mais adiante.

O primeiro bloco, ao receber um evento de *clock* realiza o incremento de uma posição de seu valor, fazendo assim uma varredura sequencial de seus termos. Com isso ele acessa cada um dos valores da LUT sequencialmente. A LUT, por sua vez, ao perceber um evento de *clock*, entrega para o DAC o valor em sua memória cujo endereço corresponde ao que foi fornecido pelo acumulador de fase. Por fim, o DAC realiza a conversão do valor digital para um valor de tensão ou corrente (ANALOG DEVICES, 1999).

Faz-se uso aqui do termo “evento” de “*clock”,* pois não necessariamente os níveis ou bordas do sinal de relógio em si ativam os blocos, podendo ser usadas, por exemplo, condições de acumulação de eventos para reduzir a frequência de saída.

A escolha do número de bits que serão usados para representar os níveis de amplitude da forma de onda a ser sintetizada tem impacto direto na resolução da forma de onda.

Outra escolha a ser feita com relação à LUT é o tamanho da memória em si, ou seja, quantos elementos precisam ser armazenados. Para se obter a menor memória com a melhor resolução possível, deve-se aproveitar ao máximo a extensão de valores de representação; contudo, isso irá depender de cada forma de onda em em particular. No capítulo 3.1.1 será apresentada uma análise de como foi escolhido o valor apropriado para o projeto em questão.

## 2.3 CURRENT STEERING DAC

Para essa aplicação percebe-se que a taxa de atualização do conversor deve ser, pelo menos, a frequência desejada na saída multiplicada pelo número de bits em que será quantizada a forma de onda, de modo que essa taxa pode se tornar substancial. Outra exigência que é imposta sobre esse conversor é uma alta precisão em relação ao valor de saída na medida o que terá um impacto direto na qualidade do sinal gerado e, consequentemente, um impacto na medida da bioimpedância.

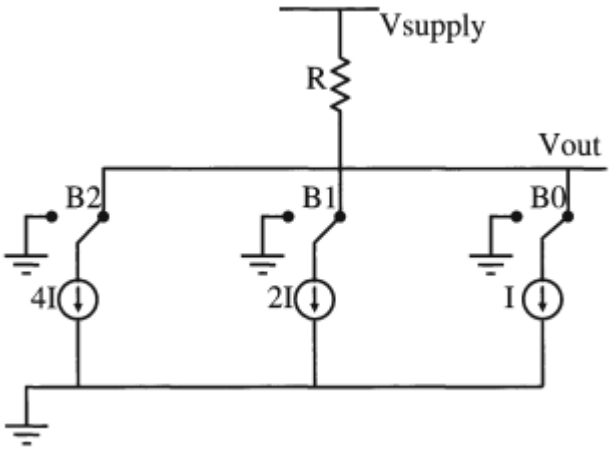
Por isso, a escolha da arquitetura do DAC deve ser tal que supra essa necessidade. De acordo com Bosch, Steyaert e Sansen (2004), a arquitetura de *current steering* tende a ser escolhida para aplicações de telecomunicações

justamente por apresentar as características de alta taxa de atualização e alta precisão, que é justamente o que se necessita para o DDS.

Além disso, a saída do conversor *current steering* é naturalmente em corrente. Para diversas aplicações uma conversão tensão-corrente é feita posteriormente; contudo, no caso em que se deseja utilizar o método DDS para geração de corrente um conversor DA com saída direta em corrente se torna ainda mais atrativo.

O princípio de funcionamento desse conversor é baseado no somatório de fontes de corrente individualmente chaveadas, como exemplifica a Figura 5. Essas fontes de corrente podem ser ponderadas para representar uma escala binária, como é o caso do exemplo na Figura 5, ou podem ser todas referências de corrente idênticas para converter um código de entrada termométrico (ou unário). A primeira apresenta a vantagem de exigir um menor número de fontes individualmente controladas para o mesmo número de códigos unívocos, enquanto a segunda apresenta um comportamento mais monotônico como será mais bem explicado no capítulo 2.5. Há ainda uma terceira alternativa que é usar uma arquitetura mista termométrica-binária para tirar proveito das vantagens de ambas as arquiteturas.

Figura 5 – Exemplo de arquitetura de DAC *current steering*




Fonte: (BOSCH, STEYAERT e SANSEN, 2004).

## 2.4 CHAVE DIFERENCIAL

Como discutido no tópico 2.1, faz-se necessário que a fonte de corrente possua saída diferencial. A solução encontrada para atender esse critério foi

considerar a implementação de um *switch* diferencial na saída da fonte de corrente. Esse circuito tem o mesmo princípio de funcionamento do circuito de ponte-h, muito utilizado para o controle de motores DC e para aplicações em conversores de tensão com vasta aplicação em eletrônica de potência (RASHID, 2017).

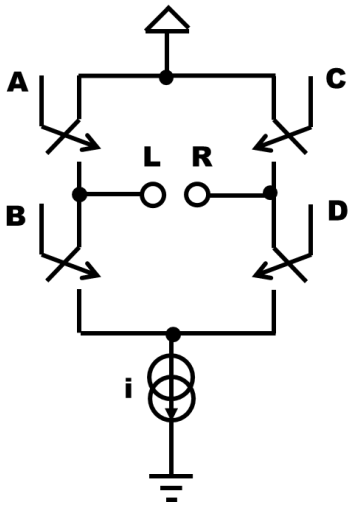
Para esta aplicação em particular, será usado de forma muito similar ao que seria seu uso em controle de motores DC, ou seja, a partir de uma fonte de corrente unipolar exclusivamente positiva produzir uma saída diferencial tal que a corrente na carga flua em ambos os sentidos.

Isso é realizável através do circuito apresentado na Figura 6, cujo funcionamento é bem evidente considerando uma carga entre os nós L e R e que o circuito em ponte opere em três modos de funcionamento:

a) **Corrente nula**. Na situação em que A e C encontrem-se simultaneamente abertas, não haverá corrente fluindo pela carga, uma vez que não há circuito fechado. O mesmo ocorreria para B e D simultaneamente abertas ou caso a fonte de corrente i seja nula.
**b) Corrente positiva no sentido LR**. Na situação em que a corrente i seja diferente de zero e com sentido indicado pela seta nela contida (conforme a Figura 5), quando A e D encontram-se fechadas e C e B encontram-se abertas, a corrente i flui do terminal L para R.
**c) Corrente negativa no sentido LR**. Na situação em que a corrente i seja diferente de zero e com sentido indicado pela seta nela contida (conforme a Figura 5), quando C e B encontram-se fechadas e A e D encontram-se abertas, a corrente i flui do terminal R para L.

Nota-se, contudo, que como há 4 sinais de controle cujos valores podem assumir 2 estados distintos (aberto ou fechado), percebe-se que haveria 16 condições distintas e que foram apresentadas apenas 9, as demais condições, todavia, serão consideradas condições proibidas, pois se tratam de condições em que há fuga de corrente por chaves do mesmo ramo causando um curto de braço, o que é uma situação indesejada.

Figura 6 – Esquemático básico de um circuito de ponte-H

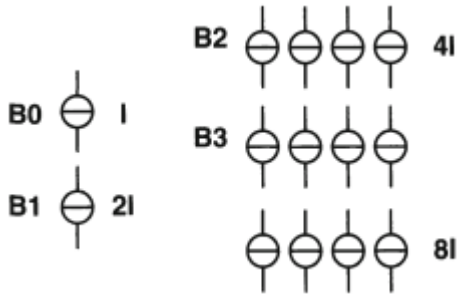



Fonte: O autor.

## 2.5 CÓDIGO TERMOMÉTRICO-BINÁRIO

Uma implementação muito utilizada do DAC *current steering* é a segmentada (CHENG, 2012). Nessa implementação os bits menos significativos são implementados como fontes de corrente cuja magnitude cresce de um fator 2 a cada bit (em geral utilizando associações de fontes de corrente unitárias para contornar problemas de descasamento de componentes) enquanto os bits mais significativos são implementados utilizando codificação termométrica. Ou seja, se, ao total utilizam-se $N$ bits, dos quais $T$ são unários, onde são utilizadas associações de uma fonte de corrente unitária $I_{REF}$ sendo essa a corrente do LSB, então o último bit binário ($N - T - 1$) tem corrente de magnitude $2^{N-T-1}I_{REF}$ e as demais $2^T - 1$ fontes possuem cada uma corrente de magnitude $2^{N-T}I_{REF}$. A Figura 7 mostra um exemplo dessa aplicação para $N = 4$ e $T = 2$.

Figura 7 – Exemplo da implementação das fontes de corrente em arquitetura segmentada




Fonte: (BOSCH; STEYAERT; SANSEN, 2004), editada.

Para a implementação de uma arquitetura de um DAC *current steering* supondo que cada uma das fontes independentes de corrente possua um erro relativo igual a $\varepsilon_n$ de modo que $I_{eff,n} = (1 + \varepsilon_n) I_{REF}\, 2^n$ onde $n$ é o número do bit relativo à fonte e $I_{REF}$ é uma corrente de referência para o conversor DA (essa análise considera o bit $n = 0$ como LSB). Sendo esse erro aleatório de média zero e variância $\sigma^2$ , percebe-se que quanto maior $n$, maior o erro da corrente associada à fonte individual.

O maior erro se dará na transição de $\left(\frac{N}{2} - 1\right)$ onde há N-1 1's para $(\frac{N}{2})$ onde há N-1 0's. Nessa situação, idealmente a corrente de saída passaria de $(2^{N-1} - 1) I_{REF}$ para $2^{(N-1)} I_{REF}$, portanto, a diferença da transição é $I_{REF}$ e, consequentemente, positiva. Contudo, há um erro aleatório sobre cada fonte de corrente, sendo assim a soma das correntes de fato será: $(2^{N-1} - 1) I_{REF} + \sum_{n=0}^{N-1} \varepsilon(n) I_{REF} 2^n$. Portanto, sua média é $\bar{\Sigma} = 0$ uma vez que a média de cada um dos termos é nula e a variância é $Var(\Sigma) = (2^{N-1} - 1) I_{REF} \sigma^2$.

Porém, uma falha de não monotonicidade ocorrerá apenas no caso em que a variação de corrente é negativa. Assim, deseja-se determinar a probabilidade de que um $\Delta I < 0$ sendo que essa variável aleatória possui média $I_{REF}$ e variância $(2^{N-1} - 1) I_{REF} \sigma^2 + 2^{N-1} I_{REF} \sigma^2 = (2^N - 1) I_{REF} \sigma^2$. Usando essas estatísticas e supondo uma distribuição Gaussiana pode-se calcular a probabilidade desejada.

Para o caso termométrico todas as transições positivas envolvem apenas o acionamento de uma única fonte independente de corrente. Portanto, sua motonicidade é garantida uma vez que todas as fontes de corrente são positivas.

E na arquitetura mista ou segmentada, é preciso considerar um outro parâmetro $T$, o número de bits termométricos utilizado na codificação. A transição

crítica é aquela em que todos os dígitos binários estão em estado ‘1’ e ocorre um novo incremento, alterando a posição de um dos algarismos termométricos e zerando os algarismos binários. Utilizando um resultado análogo ao anterior, a variância será dada pela soma das variâncias dos  bits mais a variância de uma única fonte devido a transição do algarismo unário, de modo que a generalização para $N$ bits sendo $T$ desses bits representados em código termométrico resulta em:

$$\sigma_{N,T}^2 = (2^{N-T} - 1)I_{REF}\sigma^2 + 2^{N-T}I_{REF}\sigma^2 = (2^{N-T+1} - 1)I_{REF}\sigma^2$$

Esse resultado mostra que aumentando o número de fontes em codificação termométrica reduz-se a variância e, consequentemente, a probabilidade de que o conversor seja não monotônico, em contrapartida, o número de fontes individualmente controladas também aumenta e consequentemente aumentam o consumo de potência e área utilizada. Deve-se então realizar um *trade-off* entre performance dinâmica do conversor DA e eficiência em área e potência.

# 3 PROJETO DO SINTETIZADOR DIGITAL DIRETO

Nos capítulos a seguir serão apresentadas as técnicas utilizadas para o desenvolvimento do DDS proposto para gerar a fonte de corrente, conforme discutido no capítulo 1.

Primeiramente, serão apresentadas as decisões com relação às especificações do projeto digital, nesse caso, o número de bits para quantificar a senoide e a quantidade desses bits a ser usada em implementação segmentada. Após isso, a Figura 4 será mais bem detalhada.

O diagrama de blocos da Figura 4 será então transformado em um projeto em nível RTL, separando o projeto em bloco de controle e bloco operativo. Cada bloco separado será então descrito e desenvolvido em seu respectivo capítulo. Então serão interconectados.

O último item a ser tratado neste capítulo corresponderá a algumas otimizações que serão feitas a fim de melhorar o desempenho do projeto dentro das especificações determinadas no capítulo 1.

## 3.1 DESDOBRAMENTO DO DIAGRAMA DE BLOCOS

No capítulo 2.2 foi apresentada uma forma geral de funcionamento de Síntese Digital Direta; contudo, para um projeto dedicado a um fim específico faz-se ajustes para melhorar o funcionamento do DDS. A primeira análise a ser feita diz respeito a LUT, quanto ao conteúdo que ela deverá armazenar para representar a senoide e como é possível reduzir o tamanho da memória utilizando recursos de controle. Tais recursos serão implementados em um bloco avulso que fará alterações no acumulador de fase. Por fim, um bloco que não aparece no diagrama de blocos original, mas é necessário para realizar a conversão do DAC em arquitetura segmentada é o decodificador binário para termométrico-binário, bem como a escolha do nível de segmentação para o conversor.

### *3.1.1 Look Up Table*

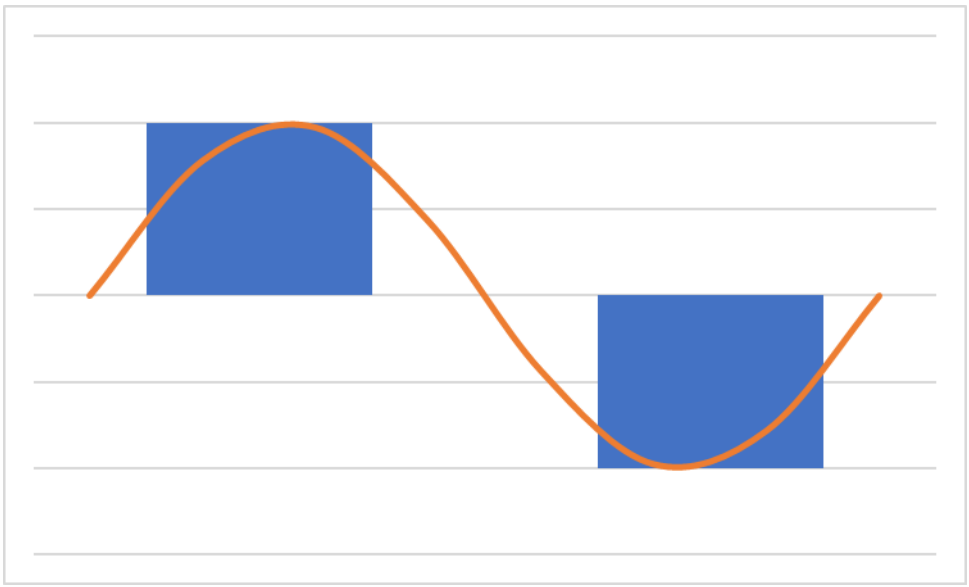
A LUT deve conter todos os valores de quantização da forma de onda desejada. Porém, surgem duas questões: Quantos bits serão necessários para

quantizar de forma adequada a senoide? E quantos valores distintos (endereços) serão necessários para representá-la apropriadamente?

Além disso, vale lembrar que o bit de sinal será utilizado para realizar o controle do *switch* diferencial. O bit de sinal será tratado em separado e não como parte da LUT. Até porque na senoide, o semi-ciclo negativo é idêntico em módulo ao positivo, de forma que não é necessário de fato utilizar uma memória com o dobro de tamanho para armazenar valores que serão distintos apenas por um único bit. Assim, daqui em diante, o bit de sinal será alheio à LUT e será inserido nas equações apenas como o valor inteiro (-1).

Dito isso, pode-se analisar agora o número de bits necessário para quantização. A escolha do número de bits para quantizar o sinal deve levar em conta que o passo de quantização tem impacto direto na qualidade do sinal, e uma boa forma de se avaliar esse impacto é observando a distorção harmônica do sinal discretizado. Para isso, é interessante partir, primeiramente, para o caso mais simples com um único bit de representação, ou seja, a senoide discretizada pode assumir apenas os valores 1, 0 e -1 (lembrando que o bit de sinal não faz parte da LUT, porém será considerado uma vez que será gerado de outro modo).

Figura 8 – Exemplo de discretização da senoide utilizando apenas um bit



Fonte: O autor.

Nesse caso, é bem clara a distorção do sinal como pode ser visto na Figura 8, porém, é necessário quantificar a distorção. Isso pode ser feito através do THD.

Para isso, é necessário calcular as amplitudes das harmônicas. Como se trata de um sinal periódico no tempo, cabe o cálculo da série de Fourier. Como a função é composta por segmentos de valores constantes, a análise é relativamente simples. Já que se trata de uma função ímpar, as componentes em cosseno são nulas e sendo uma função de média também nula, a componente DC também é nula restando apenas:

$$b_n = \frac{2}{T}\int_0^T f(t)\,\sin\left(\frac{2\pi n t}{T}\right)dt$$

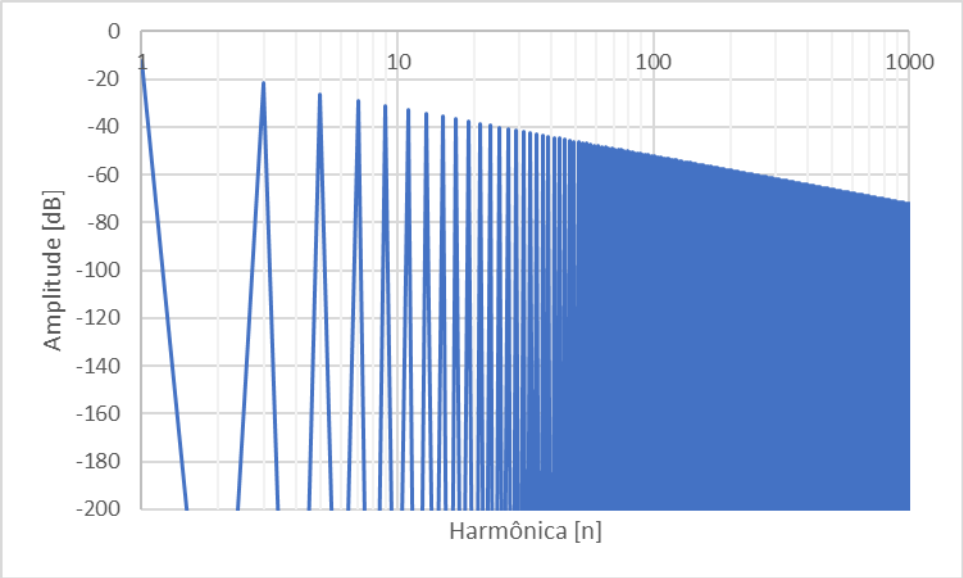
Partindo a forma de onda em 8 períodos iguais, tem-se:

$$b_n = \frac{1}{4T'}\left(\int_{T'}^{3T'} \sin\left(\frac{n\pi t}{4T'}\right)dt - \int_{5T}^{7T} \sin\left(\frac{n\pi t}{4T}\right)dt\right)$$

$$b_n = \frac{1}{n\pi}\left(\cos\left(\frac{n\pi}{4}\right) - \cos\left(\frac{3n\pi}{4}\right) + \cos\left(\frac{7n\pi}{4}\right) - \cos\left(\frac{5n\pi}{4}\right)\right)$$

Normalizando esses coeficientes e representando-os em uma escala decibel ao longo de um eixo de frequências logarítmicos, obtém-se o espectro apresentado na Figura 9, claramente um espectro altamente poluído como era de se esperar analisando o comportamento da forma de onda no tempo.

Figura 9 – Espectro da forma de onda utilizando apenas um bit




Fonte: O autor

A partir disso, é possível calcular o THD e determinar uma métrica para quantificar a qualidade da representação. Uma análise análoga foi feita para representações de 2 a 8 bits e com isso, foi obtida a Tabela 1 com o THD para cada representação.

Tabela 1 – THDs calculados para várias configurações de bits

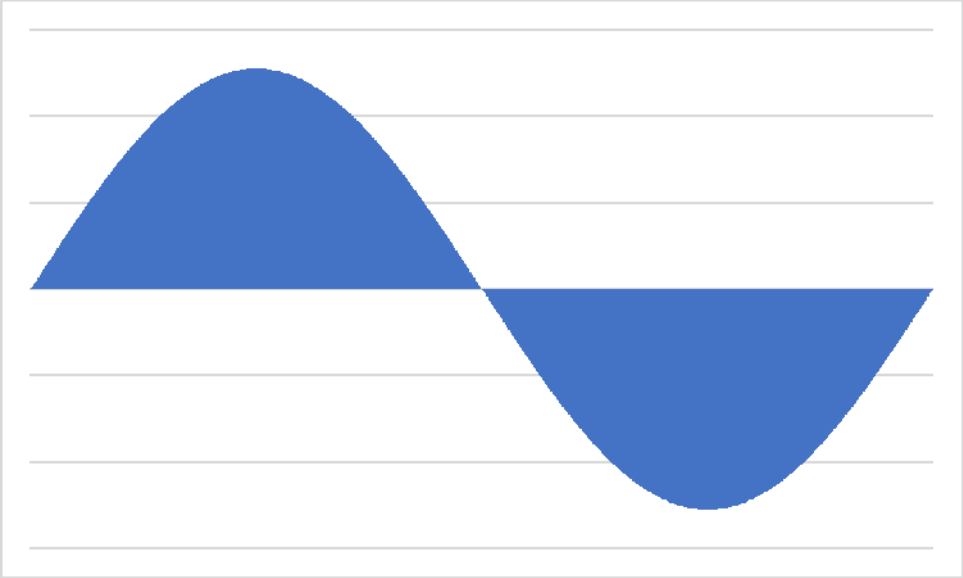
| Número de Bits | THD | Maior Harmônica Ímpar | |
|---|---|---|---|
| | | Número | Amplitude |
| 1 | 48,3% | 3 | -21,8 dB |
| 2 | 28,9% | 7 | -26,7 dB |
| 3 | 9,05% | 3 | -30,9 dB |
| 4 | 4,12% | 63 | -38,4 dB |
| 5 | 2,31% | 127 | -43,7 dB |
| 6 | 1,13% | 255 | -49,1 dB |
| 7 | 0,266% | 511 | -60,4 dB |
| 8 | 0,233% | 1023 | -60,4 dB |

Fonte: O autor

Com essa análise, verifica-se que com o aumento do número de bits, o ganho em termos de melhora do THD se torna cada vez menos significativo, sendo a representação com 7 e 8 bits muito próxima em termos de distorção, diferindo por apenas 0,033%. Contudo, a análise do espectro também mostra que a significância das harmônicas se dá cada vez mais distante da fundamental com o aumento do número de bits, sendo que a distância dobra de 7 para 8 bits, o que torna muito mais simples o trabalho de filtrar as componentes espúrias por parte do projetista do DAC.

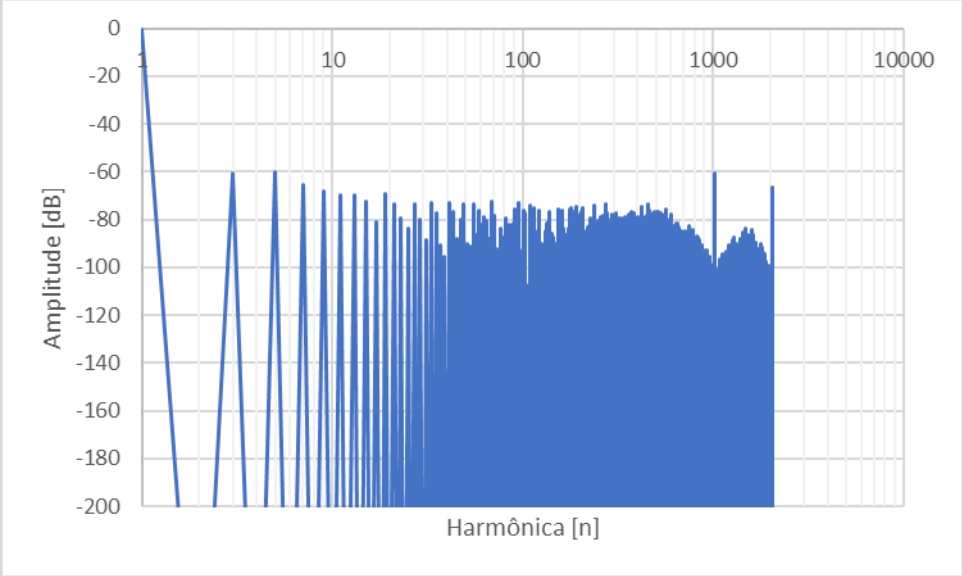
Portanto, será feita a arquitetura utilizando 8 bits de informação mais um bit de representação de sinal fora da LUT. As figuras 10 e 11 apresentam, respectivamente, a forma de onda esperada com essa representação e o espectro determinado utilizando a série de Fourier.

Figura 10 – Forma de onda utilizando 8 bits de dados



Fonte: O autor.

Figura 11 – Espectro para a representação de 8 bits

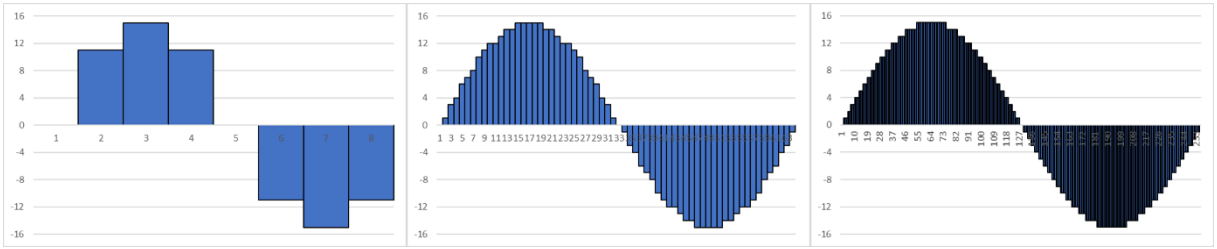



Fonte: O autor.

A outra questão que fica a ser respondida é o número de endereços ou linhas que a memória deverá conter. A resposta para essa pergunta pode ser obtida quando se sabe o impacto que o número de endereços tem na discretização da forma de onda. Para isso, vale observar na Figura 12 a comparação utilizando 4 bits de quantização em todas as representações, porém, do lado esquerdo foi utilizada uma tabela com 8

endereços, ao meio uma tabela com 64 endereços e a direita uma tabela com 256 endereços.

Figura 12 – Comparação entre o número de endereços na tabela



Fonte: O autor.

Percebe-se que em sub amostragem, há diversos valores que não são utilizados, uma vez que o número de amostras é tão baixo que não cabem todos os valores possíveis na representação. Em sobreamostragem, ocorre o oposto, todos os valores são utilizados mais de uma vez, desse modo, há um excesso de valores repetidos na memória, ocupando espaço desnecessariamente.

Deve se encontrar, portanto, um ponto de equilíbrio entre ter amostras o bastante para se aproveitar toda a gama de valores representáveis, mas não tantas a ponto de se desperdiçar área de memória.

Isso pode ser definido encontrando o menor valor que melhor aproveite o número de valores distintos possíveis de serem representados. Para isso, percebe-se que a maior taxa de variação de uma senoide encontra-se na origem, ou seja, é próximo dela onde irão ocorrer saltos maiores; esse, portanto, é o ponto crítico desta análise. Como só são admitidos valores inteiros, a análise será normalizada em função do número de bits, assim, para que não haja sub-amostragem, o valor da amostra após zero deverá ser menor que 1,5, já que será feito o arredondamento da função seno normalizada pelo número de bits, ou seja, o menor número $K$ de amostras que não resultará em sub amostragem será o menor inteiro que satisfaça:

$$(2^N - 1)\sin\left(\frac{2\pi}{K}\right) < \frac{3}{2}$$ [1]

Portanto, o menor $K$ em função do número de bits $N$ será dado por:

[1] Essa equação é válida para qualquer $N > 1$, para $N = 1$ a análise deve ser feita a parte

$$K > \frac{2\pi}{\arcsin\frac{3}{2^{N+1}-2}}$$

A Tabela 2, então, apresenta esses valores de K para diferentes números de bits de quantização. Como já foi decidido que serão utilizados 8 bits, portanto, seriam necessárias 1069 amostras da senoide para representá-la com esse número de bits, contudo, os endereços devem ser potências de 2 e 1069, o que está muito mais próximo de 1024 do que de 2048, portanto, serão utilizadas $2^{10}$ amostras com um endereçamento de 10 bits.

Tabela 2 – Número de amostras necessárias em função do número de bits de quantização

| Número de Bits de Quantização | Menor Número de Amostras | Potência de 2 mais próxima | Tamanho do Endereço em Bits |
|---|---|---|---|
| 1 | 12 | 16 | 4 |
| 2 | 12 | 16 | 4 |
| 3 | 30 | 32 | 5 |
| 4 | 63 | 64 | 6 |
| 5 | 130 | 128 | 7 |
| 6 | 264 | 256 | 8 |
| 7 | 532 | 512 | 9 |
| 8 | 1069 | 1024 | 10 |

Fonte: O autor.

Porém, ao observar o comportamento de uma senoide, percebe-se que da mesma forma como o segundo semi-ciclo é uma versão espelhada do segundo semi-ciclo em torno do eixo horizontal, o segundo quadrante é uma versão espelhada do primeiro quadrante em torno do eixo horizontal; portanto, pode-se representar a LUT com apenas o primeiro quadrante e apenas endereçar o segundo quadrante no sentido contrário para obter o mesmo resultado com metade da área de memória. Assim, são necessárias apenas 256 amostras na tabela que ao multiplicar por 2 em função da repetição da tabela no sentido contrário e novamente por 2 ao considerar o bit de sinal, obtém-se as 1024 amostras que foram determinadas.

Esses valores serão então obtidos avaliando a função:

$$f(x) = \text{round}\left(255 \cdot \sin\left(\frac{2\pi x}{1024}\right)\right),\ x \in \mathbb{Z} \cap [0, 255].$$

Contudo, como aqui, tratam-se de pontos discretos há uma questão a ser levantada: após o primeiro quadrante completo, a amostra seguinte deve ser a mesma que foi apresentada no intervalo anterior (255) ou então a amostra imediatamente adjacente a essa (254)? Para responder a essa questão, pode-se atentar para o ponto próximo ao zero (de amplitude) da senoide. Próximo ao zero é onde a inclinação da função é máxima de modo que seus valores de amplitude variam com maior intensidade. Sendo assim, inserir um valor repetido bem no ponto onde ocorre essa alta taxa significaria inserir artificialmente uma componente de baixa derivada, como um ponto morto entre as transições. Assim, deve-se passar para o valor adjacente e não repetir o último valor da lista.

Mas isso leva a outra observação importante. Ao não repetir os valores, o primeiro quadrante terá de fato 256 amostras pois começa da amostra 0 e vai até a amostra 255; o segundo quadrante vai da amostra 254 até a amostra 0 ou seja, tem apenas 255 valores; do mesmo modo, o terceiro quadrante também possui 255 amostras; e o último quadrante vai de 254 até 1 visto que a próxima amostra 0 faz parte já do ciclo seguinte da senoide, e assim, possui apenas 254 amostras. Totalizando por ciclo 1020 amostras e não 1024. Esse fator de 4 seria o mesmo independentemente do número de bits, visto que depende apenas do número de ciclos em que se dividiu a senoide.

### 3.1.2 Acumulador de fase

Conforme explicitado no capítulo anterior ficará a cargo do acumulador de fase diferenciar o primeiro do segundo quadrante através de uma contagem progressiva para o primeiro quadrante e regressiva para o segundo quadrante.

Para isso, o acumulador de fase deve ser um bloco tal que possua saída de endereçamento de 8 bits para acessar a LUT, retendo o valor do endereço durante um ciclo de *clock*, sendo capaz de receber um sinal de controle que realize a mudança entre incrementos sucessivos de um endereço para decrementos sucessivos de um endereço.

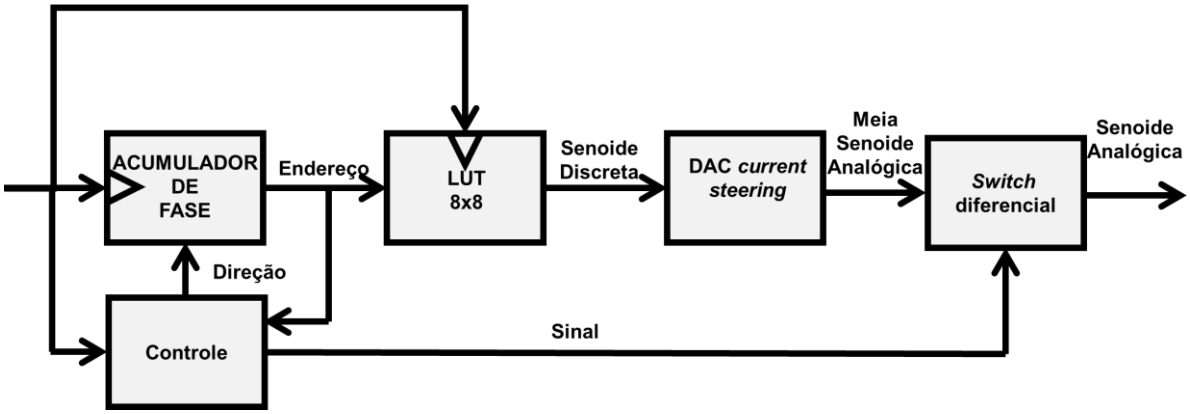
### 3.1.3 Controle

Como algumas das funções que a princípio seriam realizadas pela LUT não foram implementadas nela para redução da memória, cabe então a um bloco de controle executar as seguintes tarefas:

- Identificar que o acumulador de fase chegou ao limite superior de endereçamento;
- Quando isso ocorrer, enviar sinal de controle para alterar o sentido de contagem;
- Identificar quando o primeiro semi-ciclo foi encerrado;
- Quando isso ocorrer, alterar o bit de sinal enviado para o *switch* diferencial

Com isso, a customização do DDS para esse problema em específico produz. um diagrama de blocos conforme a Figura 13.

Figura 13 – Diagrama de blocos do DDS específico da aplicação




Fonte: O autor.

## 3.2 PROJETO EM RTL

Para transformar o diagrama de blocos em um circuito funcional, decidiu-se utilizar o processo de projeto em nível de transição entre registradores; com isso, é possível não só desenvolver o sistema de forma metódica, mas também fazer uso de linguagens de descrição de *hardware* para que o projeto possua portabilidade de modo que possa tanto ser utilizado em seu propósito final para gerar síntese de circuito digital para ser integrado, como também fazer uso de outros artifícios de síntese lógica, como o uso de FPGA para a aplicação do bloco digital.

O método escolhido para gerar o circuito foi dividi-lo em um bloco operativo composto de registradores e elementos de lógica combinacional para produzir o fluxo de dados propriamente dito, e um bloco de controle onde é implementada uma máquina de estados finita e realiza o controle do bloco operativo através do uso de multiplexadores. Os capítulos seguintes detalham cada bloco, na sequência será apresentada a fusão dos blocos para o projeto completo.

A linguagem de descrição de *hardware* escolhida foi a VHDL por ser um padrão amplamente usado no meio acadêmico e possuir compatibilidade com diversos ambientes de compilação. Todos os códigos completos se encontrarão no Apêndice A – VHDLs do sistema otimizado e no corpo de texto serão apresentados apenas pseudocódigos quando necessário para facilitar a explicação sem poluir a visualização.

### 3.2.1 Projeto do bloco operativo

A tabela de consulta (LUT) foi implementada utilizando uma simples memória ROM onde todos os valores calculados conforme explicado na seção 3.1.1, foram associados a um endereço ordenado de 0 a 255. Em linguagem VHDL foi utilizada a estrutura *when-else* para associar à saída um dos valores possíveis da memória realizando a seleção da saída por endereço fornecido, como exemplificado no Quadro 1.

Quadro 1 – Pseudocódigo para a LUT

```
ENTITY ROM IS
PORT(
      endereço    : IN (8);
      dado        : OUT (8)
);
END ROM;

ARCHITECTURE circuito OF ROM IS
BEGIN
      dado <=
            0 WHEN endereço = 0x00 ELSE
            2 WHEN endereço = 0x01 ELSE
            ...
            255 WHEN endereço = 0xFE ELSE
            255;
END circuito;
```

Fonte: O autor

Quanto ao acumulador de fase, sua implementação exige que ele retenha o valor do endereço uma vez que a LUT é implementada com lógica combinacional; além disso, com essa implementação é possível reaproveitar o valor retido no registrador para realizar a operação de soma ou subtração com esse valor e realimentá-lo no próprio registrador. Assim, quando ocorrer uma borda de *clock* o valor que o registrador irá assumir será o valor que estava contido nele acrescido ou decrescido de um. Essa alteração entre incremento e decremento pode ser feita através de um multiplexador que contenha em uma de suas entradas o valor +1 e em outra o valor -1, sendo a seleção feita pelo Bloco de Controle e o valor então é somado em um somador que alimenta o registrador.

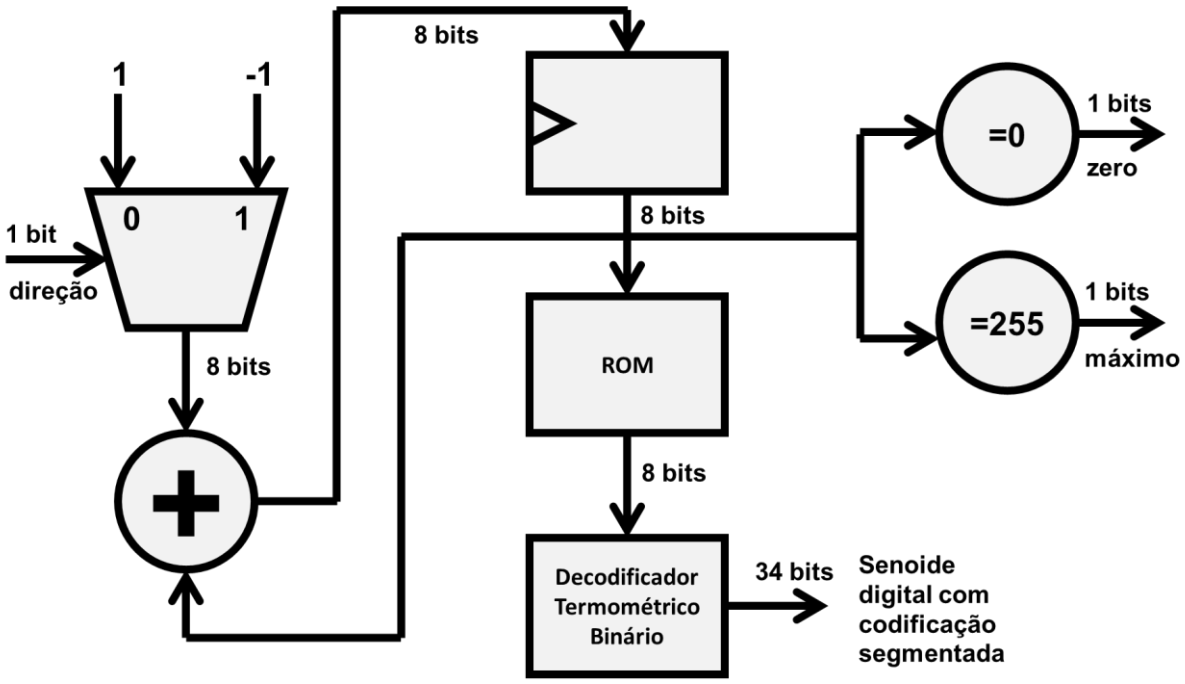
Além disso, é necessário um decodificador que converta os valores binários contidos na memória ROM para uma palavra segmentada. A implementação desse decodificador é análoga à utilizada para implementar a ROM, porém com mais ênfase no valor binário de cada dado uma vez que seu valor algébrico em um *radix* não faria sentido.

Os únicos elementos que ainda faltam para o bloco operativo são os detectores de zero e máximo. Isso é necessário para informar ao bloco de controle quando trocar de estado para alterar então a direção de contagem e a posição em que se encontra da senoide, se está no semi-ciclo superior ou no inferior. A implementação lógica dessa função é feita simplesmente com uma sequência de portas ANDs e ORs

que comparam o valor armazenado no registrador do acumulador de fase com 0 para a comparação com zero ou 255 para a comparação com o valor máximo.

Desse modo, o bloco operativo pode ser representado em um diagrama de blocos funcionais através da Figura 14.

Figura 14 – Diagrama de blocos do bloco operativo




Fonte: O autor.

### 3.2.2 Projeto do bloco de controle

Conforme explicado nos dois últimos capítulos, o bloco de controle será responsável por realizar a alternância entre a contagem progressiva e regressiva de endereços de acesso a ROM, bem como enviar para fora do bloco digital o sinal que controla o *switch* diferencial para realizar a troca entre o semi-ciclo positivo e o negativo.

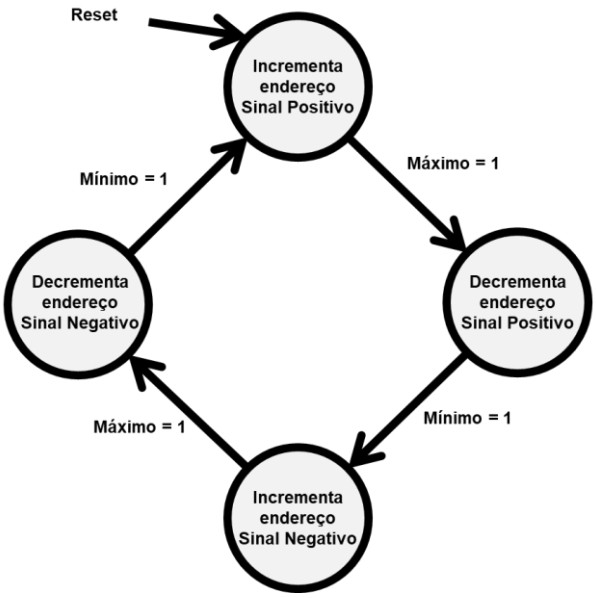
O bloco de controle foi implementado utilizando um modelo de máquina de estados finitos. Existem dois principais tipos de FSM transdutoras, ou seja, aquelas em que há uma saída baseada nas condições de entrada: as máquinas de Moore e as máquinas de Mealy. Sendo que em uma máquina de Mealy as saídas dependem tanto do estado atual onde a máquina se encontra como do valor das entradas enquanto máquinas de Moore, as saídas dependem exclusivamente do estado atual

da máquina, isso as torna mais robustas e previsíveis apesar de também, em geral, as tornarem menos eficientes em termos de uso de recursos (BOOTH, 1967).

Como a estabilidade do sistema é mais crítica que o uso de recursos para esse propósito, escolheu-se implementar uma máquina de estados de modelo Moore.

Modelando o problema, percebe-se que há 4 estados possíveis representando justamente os 4 quadrantes da senoide, sendo que as transições entre os estados ocorrem em função do endereço de acesso da LUT e do estado atual da máquina, uma vez que, estando no primeiro quadrante, o próximo certamente será o segundo, mas apenas quando chegar-se no endereço máximo. Assim como quando se está no segundo quadrante, o próximo certamente será o terceiro, mas apenas quando chegar-se no endereço mínimo. Assim, na Figura 15, tem-se o diagrama de estados representando a FSM.

Figura 15 – FSM do bloco de controle




Fonte: O autor.

A lógica de estados então fica explicitada através da Tabela 3, onde estão as condições para cada estado e se faz uso do símbolo $X$ para indicar *don't cares* ou seja, condições onde o valor do dado sinal não é relevante para a lógica. Assim como o valor das saídas para cada estado da máquina. Todas as representações numéricas nessa tabela são dadas em binário para já adequar ao utilizado na codificação em VHDL e com o *hardware* digital utilizado.

Tabela 3 – Tabela de transição de estados da FSM de controle

| Entradas | | | Saídas | | |
|---|---|---|---|---|---|
| **Estado Atual** | **Máximo** | **Zero** | **Próximo Estado** | **Sinal** | **Direção** |
| 00 | 0 | $X$ | 00 | 0 | 0 |
| 00 | 1 | $X$ | 01 | 0 | 0 |
| 01 | $X$ | 0 | 01 | 0 | 1 |
| 01 | $X$ | 1 | 10 | 0 | 1 |
| 10 | 0 | $X$ | 10 | 1 | 0 |
| 10 | 1 | $X$ | 11 | 1 | 0 |
| 11 | $X$ | 0 | 11 | 1 | 1 |
| 11 | $X$ | 1 | 00 | 1 | 1 |

Fonte: O autor.

Não consta nessa tabela, mas é importante notar que há dois sinais que foram omitidos até então: o sinal de relógio e o reset da máquina. As transições só irão ocorrer quando houver uma borda de subida de relógio, caso o sinal de reset esteja inativo, caso contrário, a máquina é dirigida ao estado inicial “00”.

Para implementação desse sistema em VHDL, foi utilizada uma estrutura que faz uso de *process* que permite a utilização de *if-then* e *case-then*, de modo que a implementação fica relativamente simples, como pode ser visto no Quadro 2, que descreve um pseudocódigo dessa implementação.

Quadro 2 – Pseudocódigo do bloco do controle inicial

```
ENTITY controle IS
PORT
(
      reset, clock : IN;
      máximo, zero : IN;
      sinal, sinalBarado, direção: OUT
);
END controle;

ARCHITECTURE estrutura OF controle IS
      TYPE state_type IS (
            quadrante1,
            quadrante2,
            quadrante3,
            quadrante4);
      SIGNAL estado: state_type;
BEGIN
      --Processo da lógica de transição de estados
      PROCESS (clock, reset)
      BEGIN
            IF(reset = 1) THEN
                  estado <=  quadrante1;
            ELSIF (clock'EVENT AND clock = 1) THEN
                  CASE estado IS
                        WHEN qudrante1 =>
                              IF máximo = 1 then
                                    estado <= quadrante2;
                              ELSE
                                    estado <= quadrante1;
                              END IF;
                              ...
                  END CASE;
            END IF;
      END PROCESS;
      --Processo da relação estado-saída
      PROCESS (estado)
      BEGIN
            CASE estado IS
                  WHEN quadrante1 =>
                        direção <= 0;
                        sinal <= 0;
                        sinalBarrado <= 1;
                        ...
            END CASE;
      END PROCESS;
END estrutura;
```

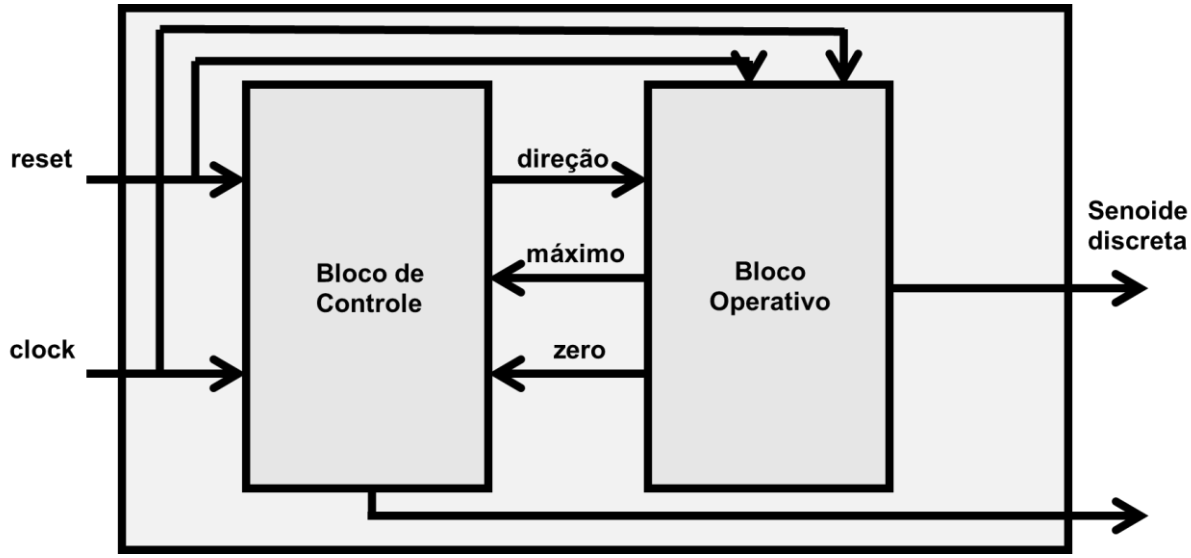
Fonte: O autor.

### 3.2.3 Conexão dos blocos

Os blocos de controle e operativo são conectados para formarem uma unidade, de modo que os únicos sinais de entrada são o reset e o *clock,* e os únicos sinais de saída são o barramento de dados com a informação da senoide discretizada e em codificação segmentada e o sinal de “sinal” e seu complementar que controlam o *switch* diferencial. A necessidade do par diferencial no bloco digital surge para evitar o uso de inversores após a saída dos sinais do bloco, isso pode incorrer em atrasos de geração significativos e provocar *glitches* ou mesmo curto-circuito de braço na ponte H.

O resultado dessa união é representado pelo diagrama da Figura 16 onde são apresentados como macro blocos. Todos os sinais contidos na caixa externa não são visíveis a um observador externo do bloco, tendo acesso apenas às entradas e saídas.

Figura 16 – Diagrama de blocos da conexão dos blocos operativo e controle




Fonte: O autor.

## 3.3 OTIMIZAÇÕES

A princípio, a descrição dos blocos nos dois últimos capítulos seria a utilizada para a síntese do circuito integrado do DDS; contudo, como será apresentado no capítulo 4, alguns resultados preliminares mostraram que havia tempo de operação mais que suficiente para que o sistema operasse na frequência desejada; desse

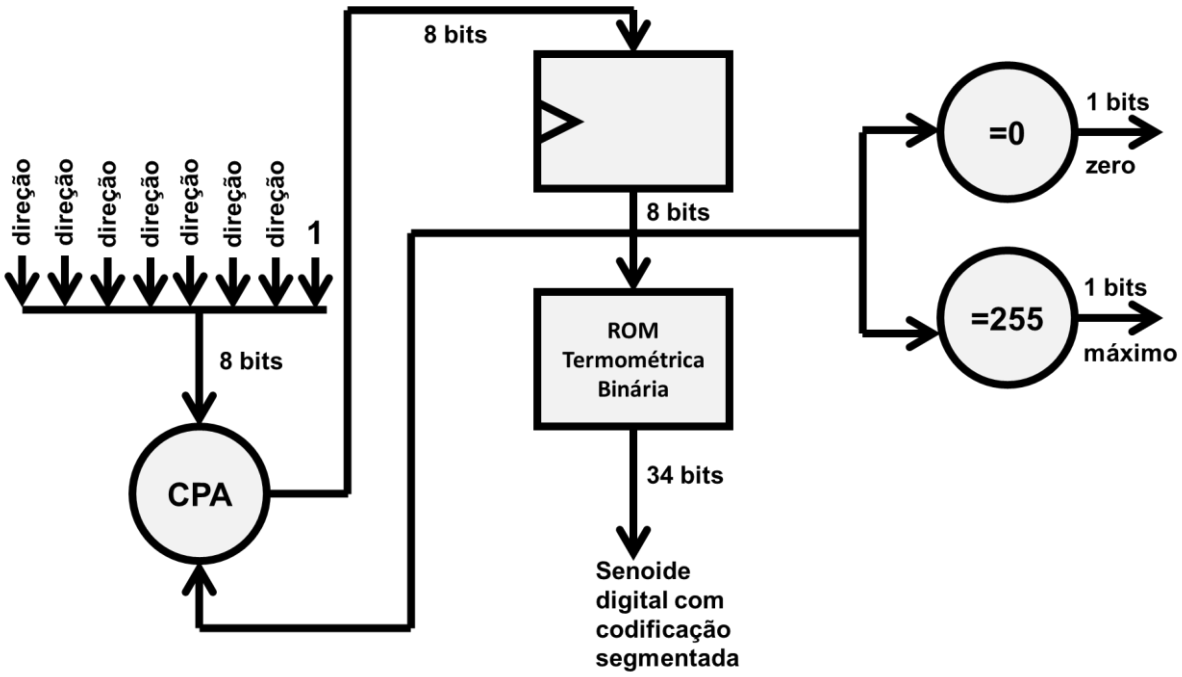
modo, havia margem de manobra para otimizar o sistema em área sem pecar no requisito de frequência. Então, foram realizadas algumas alterações para realizar esse efeito.

A primeira alteração foi eliminar o multiplexador uma vez que seu valor era 1 ou -1 em função do sinal de direção que valia 0 para a direção positiva ou 1 para a direção negativa. Utilizando a representação binária em complemento de 2 para 8 bits, o número 1 é a sequência 00000001 e -1 a sequência 11111111. De modo que ambos mantêm o mesmo LSB e os demais são idênticos e justamente iguais ao sinal de direção, ou seja, a operação de multiplexação é desnecessária, bastando concatenar o sinal de direção com ele mesmo 7 vezes e concatenar isso ao sinal constante ‘1’. Essa operação não tem custo de área ativa e tem impacto muito inferior em timing quando comparado a alternativa de utilizar um multiplexador, sendo assim, uma otimização tanto para velocidade quanto para uso de área.

A segunda alteração foi realizar a implementação manual de um somador CPA (*Carry Propagate Adder*), uma vez que, comparado a outros esquemas de somador, este possui maior atraso; porém, é o que resulta na menor área ocupada por ser o esquema de somador mais simples (ERCEGOVAC e LANG, 2003).

Uma terceira alteração implementada foi realizar a conversão da LUT para o código segmentado, desse modo, não é necessário o uso do decodificador, reduzindo assim a área ocupada, bem como o atraso, e, simplificando de forma geral o sistema. Essas três alterações fazem com que o esquema final do bloco operativo fique de acordo com a Figura 17.

Figura 17 – Bloco Operativo otimizado




Fonte: O autor.

A última alteração realizada foi no bloco de controle. A implementação apresentada no capítulo anterior, ilustrada pelo pseudocódigo descrito no Quadro 2 deixa para o compilador resolver a atribuição de codificação dos estados e lógica implementada.

Como muitos compiladores fazem uso de otimização em função de velocidade em detrimento do consumo de área, foi feita uma implementação manual do bloco de controle, visto que é simples o bastante para tal.

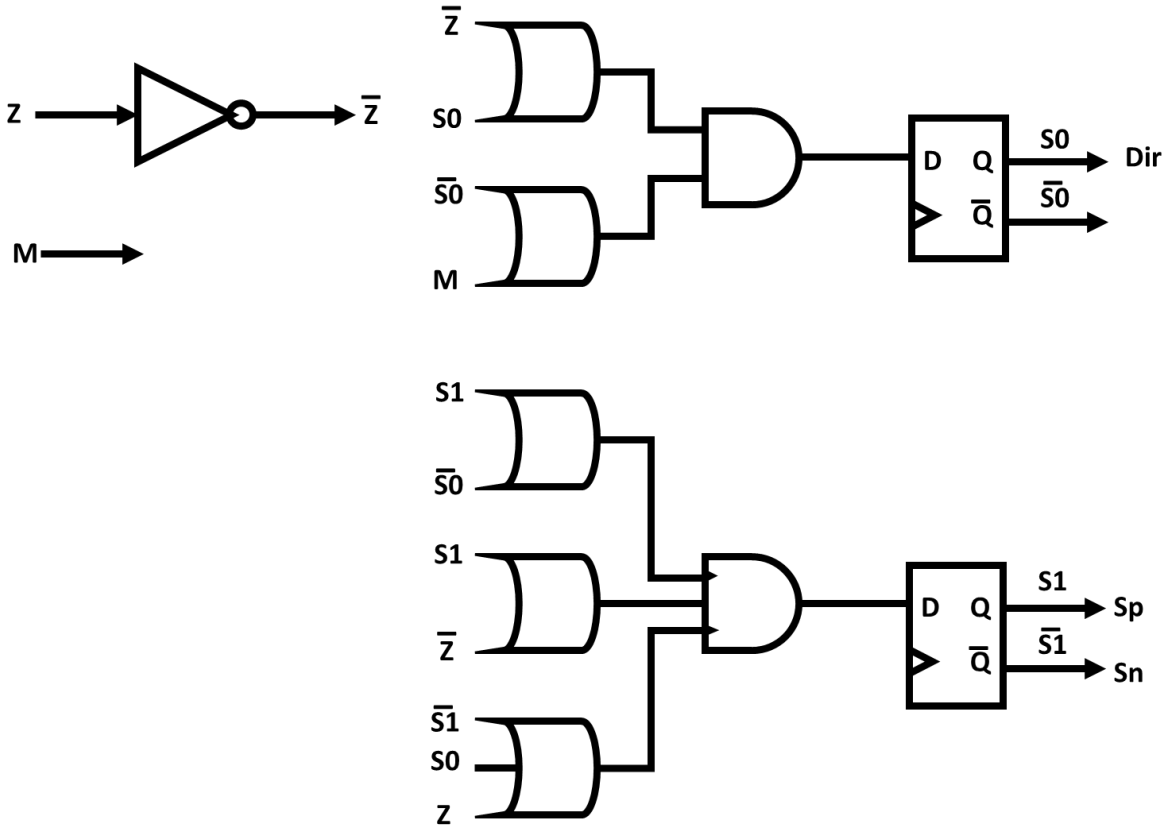
Percebe-se que são necessários apenas dois bits para armazenar o estado atual da máquina; sendo assim, são utilizados dois flip-flops para isso. A lógica de próximo estado e dos sinais de saída foi determinada utilizando a Tabela 3 para aplicar o método de mapas de Karnaugh para resolver as relações de próximo estado x entrada/estado atual e saída x estado atual.

O desenvolvimento dos mapas de Karnaugh será incluído no Apêndice B – Simplificação em mapas de Karnaugh, e sua solução, considerando que a codificação dos estados escolhida foi a binária natural e $P0$ é o LSB do próximo estado, $P1$ o MSB do próximo estado, $S0$ o LSB do estado atual, $S1$ o MSB do estado atual, $D$ o sinal de direção, $M$ o sinal de máximo, $Z$ o sinal de zero, $Sp$ o sinal de “sinal” e $Sn$ o seu complemento:

$$\begin{cases} P1 = (S1\&\overline{S0}) \mid (S1\ \&\bar{Z}) \mid (\overline{S1}\&S0\&Z) \\ P0 = (S0\&\bar{Z}) \mid (\overline{S0}\&M) \\ Sp = S1 \\ Sn = \overline{S1} \\ D = S0 \end{cases}$$

Com isso, o bloco de controle inteiro pode ser implementado com apenas dois flip-flops, 3 portas lógicas NOT, 6 portas lógicas AND e 3 portas lógicas OR.

Figura 18 - Ilustração da lógica otimizada do controle




Fonte: O autor

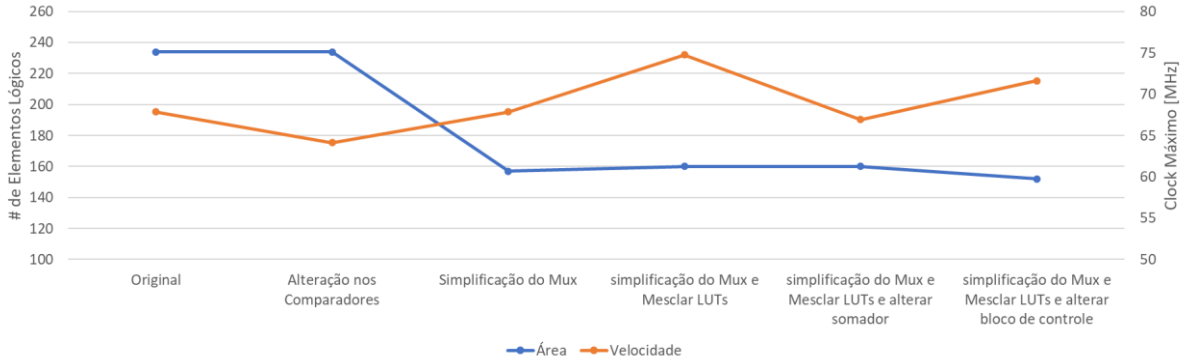
# 4 RESULTADOS

Ao longo do desenvolvimento do projeto foram obtidos alguns resultados intermediários. Além, claro, do processo de verificação e validação dos blocos em VHDL, chegou-se no ponto onde o diagrama de blocos apresentado na Figura 16 estava implementado e funcional em simulações utilizando o software ModelSim. Mas desejava-se obter uma noção da eficiência antes de realizar a implementação no Cadence Virtuoso, por se tratar de um ambiente onde o processo de *debbuging* é mais complicado e o fluxo de projeto mais longo. A alternativa foi então realizar a síntese lógica para uma FPGA utilizando o software Quartus Prime 18.1, onde foi realizada a síntese lógica para a FPGA MAX V 5M570ZM100I5, uma vez que sua organização interna é implementada em elementos lógicos (ALTERA, 2017) ao invés de LUTs programáveis, o que torna sua estrutura mais semelhante à implementação em circuito integrado e, portanto, fornece um comparativo mais próximo.

O resultado ao implementar o diagrama de blocos original para a FPGA foi um *clock* máximo de 67,85 MHz e 234 elementos lógicos necessários para sua implementação. Como a tecnologia utilizada para a FPGA é de um processo de 180 nm e sua arquitetura não é dedicada para o fim desejado, o esperado é que ao implementar o mesmo sistema em circuito integrado o resultado seja significantemente mais rápido. E, de fato, ao realizar a síntese digital no Cadence para a tecnologia de 65 nm da TSMC, o resultado obtido foi um circuito digital com *clock* máximo de 409,5 MHz, com um número total de 328 portas lógicas.

Visto isso, a implementação das otimizações apresentadas no capítulo 3.3 foi realizada uma de cada vez no Quartus para a FPGA para observar o impacto individual de cada uma. O resultado dessa iteração está apresentado na Figura 19.

Figura 19 – Resultados da implementação em FPGA




Fonte: O autor.

Contudo, alguns dos resultados observados não estavam próximos do que se esperava, principalmente no que diz respeito à pouca melhora em área ao se utilizar o somador CPA. Então, foi feito um comparativo com três implementações no Cadence, a original, o melhor resultado obtido no Quartus (todas as otimizações exceto a troca do somador nativo pelo CPA) e o melhor resultado do Quartus mais o uso de um CPA, ao invés do somador implementado pelo compilado. O resultado é apresentado na Tabela 4 e mostra que, de fato, o uso do somador CPA melhora o uso de área do CI, sendo essa a implementação final do sistema com uma área total ocupada pelo DDS sintetizado na tecnologia digital de 65 nm da TSMC de 899 μm² fazendo uso de 273 portas lógicas ao total. E sendo capaz de um *clock* máximo de 512,5 MHz, ou seja, poderia produzir uma senoide de saída de até 502,4 kHz muito superior aos 50 kHz desejados como especificação inicial.

Tabela 4 – Resumo dos resultados do processo iterativo

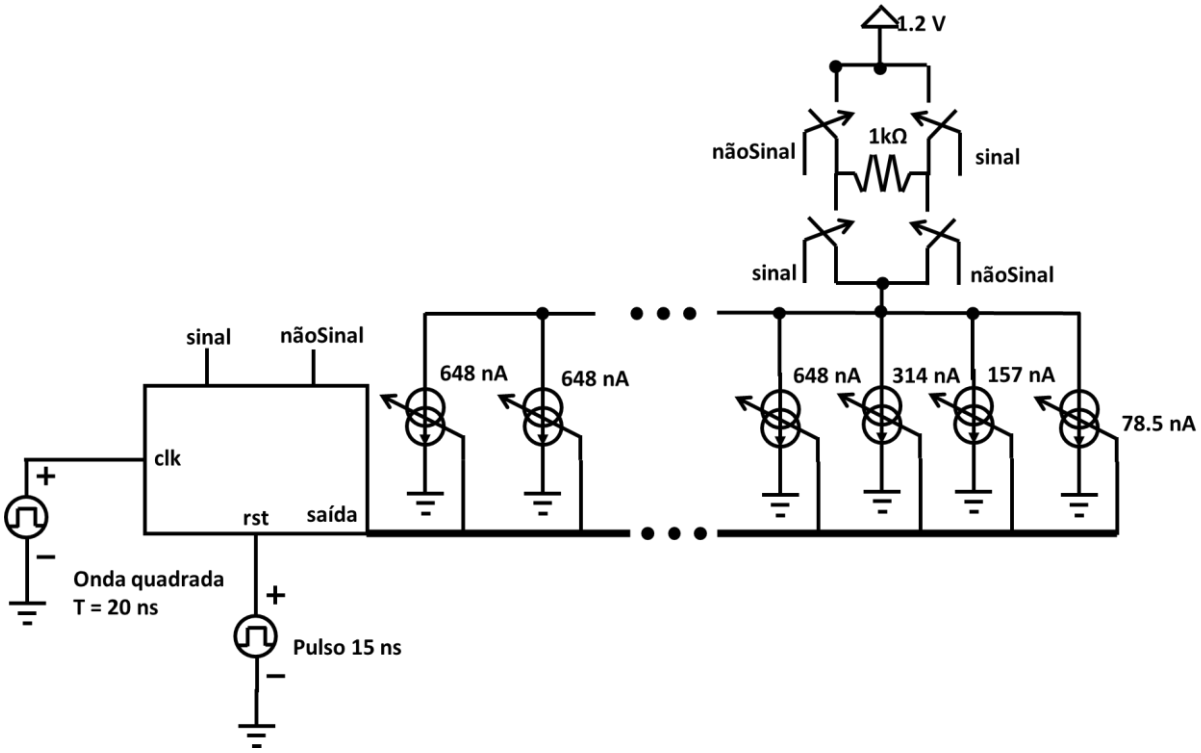
| **Implementação** | **Área Total [μm²]** | **Número de Portas Lógicas** | **Clock Máximo [MHz]** |
|---|---|---|---|
| Original | 1099 | 328 | 409,5 |
| Melhor resultado obtido no Quartus | 916 | 282 | 504,8 |
| Totas as otimizações | 899 | 273 | 512,5 |

Fonte: O autor.

Com isso, resta apenas verificar, por fim, se o resultado da saída digital ao ser convertido para um sinal analógico através de um DAC com as características especificas de fato atende aos critérios de especificação do projeto.

O *testbench* projetado para realizar essa verificação consiste de uma sequência de 31 fontes de corrente ideais controladas por tensão com corrente de 648 nA quando excitadas por uma tensão de 1.2 V (determinada como tensão de alimentação para o circuito digital), 0 A quando excitadas por uma tensão de 0 V e um comportamento linear entre esses pontos correspondendo aos 5 bits termométricos e mais três fontes com mesmo comportamento, porém com corrente superior sendo reduzida por um fator de dois a cada uma representando os 3 bits binários. Além disso, a soma dessas correntes passa por um *switch* diferencial implementado por chaves controladas e um resistor de 1 kΩ de teste. A arquitetura desse teste está disposta na Figura 20.

Figura 20 – *Testbench* para o sistema



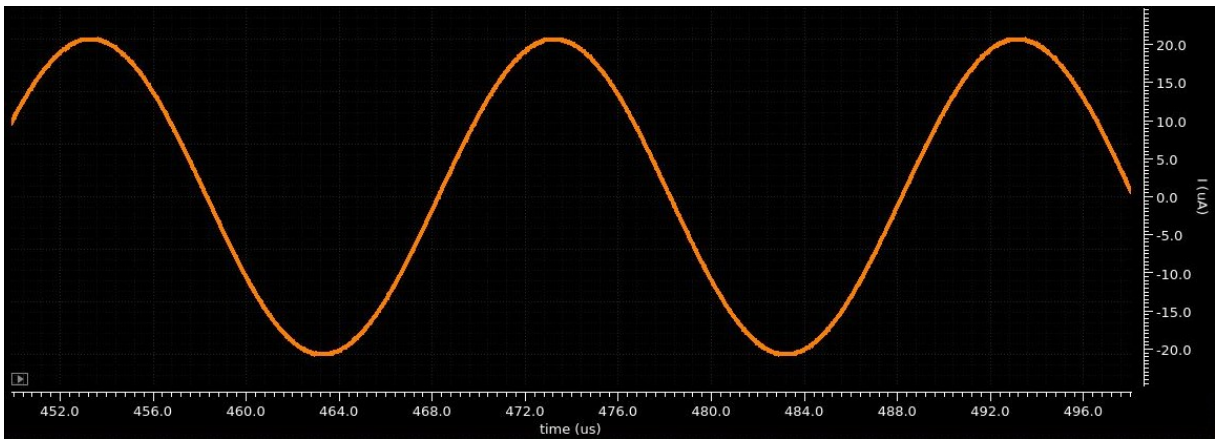

Fonte: O autor.

Para o controle da simulação são utilizados uma fonte de onda quadrada de frequência 51,0 MHz e um gerador de pulso que mantém a entrada de reset ativa por 15 ns e então o desativa.

A simulação realizada foi uma simulação de sinais mistos AMS que permite a implementação de blocos analógicos em conjunto com digitais assim como o tratamento das interfaces AD e DA. Os resultados dessa simulação foram a forma de

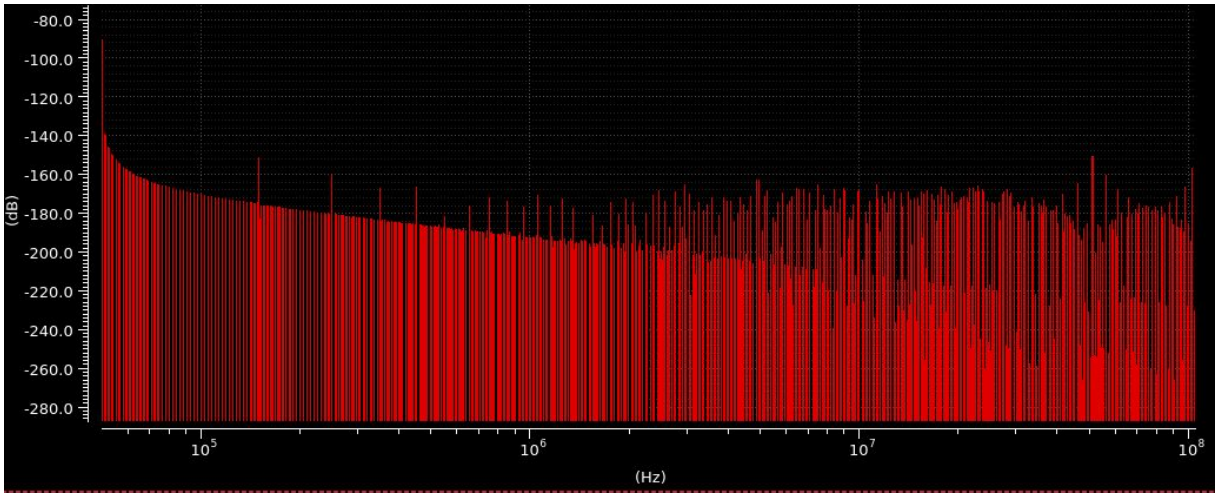
onda apresentada na Figura 21 e o espectro dessa forma de onda obtido através de FFT apresentado na Figura 22.

Figura 21 – Forma de onda do resultado da simulação de sinais mistos




Fonte: o autor.

Figura 22 – Espectro obtido por FFT do resultado da simulação de sinais mistos




Fonte: O autor.

As métricas obtidas desses resultados foram que para um clock de 51 MHz, esperava-se uma frequência exata de 50 kHz, porém foi medida uma frequência de 50,002975 kHz, um erro de 5,95 ppm em relação ao esperado. O valor de pico-a-pico objetivo era de 40 µA, porém o resultado medido em simulação foi de 41,27 µA, um erro de 3,17%. E com relação à distorção, o cálculo de THD resultou em uma distorção harmônica total de 0,258% muito próximo ao calculado no capítulo 3.1.1 que havia sido 0,233% mesmo com métodos de obtenção bem distintos.

Vale lembrar ainda que para os dois primeiros resultados, pode haver melhorias a depender dos periféricos que contornam o DDS, para a melhora da precisão da frequência, um ajuste de *clock* pode ser implementado para fazer um ajuste fino desse valor, e a amplitude da saída fica na realidade a critério da implementação do DAC, cabendo a este resolver o problema do controle de amplitude.

Por fim, como dito no capítulo 3.1.1, como a frequência de chaveamento do sistema é mais de mil vezes superior à frequência de operação, um simples filtro de primeira ordem após o DAC já seria capaz de realizar uma melhoria ainda sobre o resultado da distorção harmônica obtido na implementação digital.

# 5 CONSIDERAÇÕES FINAIS

Nesse trabalho foi descrito o uso da técnica de Síntese Digital Direta para implementação de um gerador de senoide para os fins já descritos no capítulo 1. Foi feito uso de técnicas de análise de circuitos digitais para implementar otimizações a fim de particularizar o método de DDS para os fins específicos da aplicação desejada visando obter um circuito que possa ocupar menos área de silício.

Os objetivos específicos foram atingidos ao se determinar a topologia do sintetizador digital e descrever os blocos para sua implementação no capítulo 3, assim como sua otimização para manter as características desejadas e melhorar o consumo de área e, inclusive, aumentar o escopo de frequências ao se obter um circuito com *clock* máximo superior ao necessário no capítulo 3.3. Esses resultados foram verificados e validados no capítulo 4 ao realizar simulações comportamentais e de sinais mistos verificando que, de fato, o circuito obtido atingiu o objetivo principal ao reproduzir uma senoide de 50 kHz com baixa distorção harmônica em um circuito digital realizando a conversão para sinal analógico através de um DAC com arquitetura *current steering* ideal.

Fica, contudo, como um trabalho futuro a implementação do circuito de conversão digital analógica. Contudo este trabalho já define o uso da arquitetura *current steering* com 8 bits e segmentação de 5 bits em código termométrico e 3 bits binários. Também já fica definido e facilitado o uso de um *switch* diferencial para implementação do semi-ciclo negativo da senoide controlado através de sinais diferenciais “sinal” e “sinal barrado”. Fica também a recomendação do uso de um filtro passa-baixas de primeira ordem para realizar uma suavização do sinal de saída e assim obter uma fonte de corrente ainda mais ajustada para realizar medidas de alta precisão.

Um outro trabalho futuro seria ainda o desenvolvimento do gerador de *clock* controlável, que permita assim um ajuste fino sobre a frequência de saída bem como o uso de uma malha fechada de controle que poderia ser implementada com o próprio sinal de controle do *switch* diferencial para manutenção de uma frequência constante de saída. Além do circuito digital permitir uma variação de *clock* para realizar análises em um *sweep* de frequências para poder realizar diagnósticos diversos.

# 6 REFERÊNCIAS


ALTERA. **MAX V Device Handbook**. San Jose: Altera Corporation, 2017.

ANALOG DEVICES. **A Technical Tutorial on Digital Signal Synthesis**. Analog Devices. Wilmington, p. 122. 1999.

BERA, T. K. Bioelectrical Impedance Methods for Noninvasive Health Monitoring: a review. **Journal of Medical Engineering**, Londres, 28 Junho 2014. 1-28.

BOOTH, T. L. Sequential Machines. In: BOOTH, T. L. **Sequential Machines and Automata Theory**. 1. ed. Nova Iorque: John Wiley and Sons, Inc, 1967. Cap. 3, p. 68-116.

BOSCH, A. V. D.; STEYAERT, M.; SANSEN, W. **Static and Dynamic Performance Limitations for High Speed D/A Converters**. 1. ed. Coston: Kluwer Academic Publishers, 2004.

BRAYNER, A. R. A.; MEDEIROS, C. B. **A 0.5V, 6.2μW, 0.059mm2 Sinusoidal Current Generator IC with 0.088% THD for Bio-Impedance Sensing**. 2020 IEEE Symposium on VLSI Circuits. Honolulu: IEEE. 2020. p. 34.

CHENG, L. et al. **A digitally calibrated current-steering DAC with current-splitting array**. 1 ago. 2012.

EINSTEIN, A. J. Medical imaging: the radiation issue. **Nature Reviews Cardiology**, Londres, Junho 2009. 436.

ERCEGOVAC, M. D.; LANG, T. **Digital Arithmetic**. Elsevier Science. Los Angeles. 2003.

GONZÁLEZ-CORREA, C.-A. Clinical Applications of Electrical Impedance Spectroscopy. In: SIMINI, F.; BERTEMES-FILHO, P. **Bioimpedance in Biomedical Applications and Research**. Nova Iorque: Springer International Publisher, 2018. p. 187-218.

HEYMSFIELD, S. B. et al. Human Body Composition: advances in models and methods. **Annual Review of Nutrition**, San Mateo, 17, Julho 1997. 527-558.

HOROWITZ, P.; HILL, W. **A Arte da Eletrônica:** circuitos eletrônicos e microeletrônica. 3. ed. Porto Alegre: Booksman, 2017. 1192 p.

KIM, K. et al. A 0.5-V Sub-10-μW 15.28-mΩ/ √Hz Bio-Impedance Sensor IC With Sub-1º Phase Error. **IEEE Journal of Solid-State Circuits**, Nova Iorque, Agosto 2020. 2161-2173.

KOUWENHOVEN, W. B. Effects of electricity on the human body. **Electrical Engineering**, Nova Iorque, 68, Março 1949. 199-203.

LIN, Q. et al. Wearable Multiple Modality Bio-Signal Recording and Processing on Chip: a review. **IEEE Sensors Journal**, nova Iorque, 15 Janeiro 2021. 1108-1123.

MARTINSEN, Ø. G.; GRIMNES, S.; SCHWAN, H. P. Interface Phenomena and Dielectric Properties of Biological Tissue. In: SOMASUNDARAN, P. **Encyclopedia of Surface and Colloid Science**. 3. ed. Nova Iorque: Marcel Dekker, 2002. p. 2643-2652.

NESHATVAR, N. et al. Analog Integrated Current Drivers for Bioimpedance Applications: a review. **Sensors**, Basel, 19, 13 Fevereiro 2019. 756-774.

RASHID, M. H. **Power Electronics Handbook**. 4. ed. Oxford: Butterworth-Heinemann, 2017. 1522 p.

SMITH-BINDMAN, R.; MIGLIORETTI, D. L.; LARSON, E. B. Rising Use of Diagnostic Medical Imaging In a Large Integrated Health System. **Health Affairs**, 27, Novembro 2008. 1491-1502.

TIERNEY, J.; RADER, C.; GOLD, B. A Digital Frequency Synthesizer. **IEEE Transactions On Audio And Electroacoustics**, Nova Iorque, 19, Março 1971. 48-57.

VAHID, F. **Sistemas Digitais:** projeto, otimização e HDLs. 1. ed. Porto Alegre: Artmed, 2008. 560 p.

VANKKA, J. **Direct Digital Synthesizers; theory, design and applications**. Espoo: Helsinki University of Technology, 2000. 193 p.

WOLFFENBUTTEL, R. E. Transduction of Information. In: WOLFFENBUTTEL, R. E. **Electronic Instrumentation I:** ET8017. Delft: TU Delft, 2006. Cap. 2, p. 21-58.

YATES, R. D.; GOODMAN, D. J. **Probabilidade e Processos Estocásticos:** uma introdução amigável para engenheiros eletricistas e da computação. 3. ed. Rio de Janeiro: LTC, 2019. 444 p.

# 7 APÊNDICE A – VHDLS DO SISTEMA OTIMIZADO

Arquivo de topo do projeto, utiliza os componentes “datapath” e “control”:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY DDS IS
PORT (
      rst,
      clk         : IN STD_LOGIC;
      sign,
      notSign     : OUT STD_LOGIC;
      saida       : OUT STD_LOGIC_VECTOR(33 DOWNTO 0)
);
END DDS;

ARCHITECTURE estrutura OF DDS IS

COMPONENT datapath IS
PORT (
      clk,
      dir,
      rst   : IN STD_LOGIC;
      max,
      zero  : OUT STD_LOGIC;
      saida : OUT STD_LOGIC_VECTOR(33 DOWNTO 0)
);
END COMPONENT;

COMPONENT control IS
PORT (
      rst,
      clk,
      max,
      zero        : IN STD_LOGIC;
      sign,
      notSign,
      dir         : OUT STD_LOGIC
);
END COMPONENT;

SIGNAL max, zero, dir : STD_LOGIC;

BEGIN

BO : datapath PORT MAP( clk => clk,
                        dir => dir,
                        rst => rst,
                        max => max,
                        zero => zero,
```

```
                        saida => saida);
BC : control PORT MAP(  rst => rst,
                        clk => clk,
                        max => max,
                        zero => zero,
                        sign => sign,
                        notSign => notSign,
                        dir => dir);

END estrutura;
```

Arquivo do bloco de controle, utiliza o componente “flipflop”:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY control IS
PORT (
      rst,
      clk,
      max,
      zero  : IN STD_LOGIC;
      sign,
      notSign,
      dir   : OUT STD_LOGIC
);
END control;

ARCHITECTURE estrutura OF control IS

COMPONENT flipflop IS
PORT (
      clk,
      reset,
      d     : IN STD_LOGIC;
      q     : OUT STD_LOGIC
);
END COMPONENT;

SIGNAL S0, S1, P0, P1, notS0, notS1, notZero, notMax : STD_LOGIC;

BEGIN
      notS0 <= NOT S0;
      notS1 <= NOT S1;
      notZero <= NOT zero;
      notMax <= NOT max;

      state0 : flipflop PORT MAP(   clk => clk,
                                    rst => rst,
                                    P0 => d,
                                    S0 => q);
      state1 : flipflop PORT MAP(   clk => clk,
                                    rst => rst,
```

```
                                    P1 => d,
                                    S1 => q);

      P1 <= (S1 AND notS0) OR
            (S1 AND notZero) OR
            (notS1 AND S0 AND zero);
      P0 <= (S0 AND notZero) OR
            (notS0 AND max);
      sign <= S1;
      notSign <= notS1;
      dir <= S0;

END estrutura;
```

Arquivo do componente flipflop:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY flipflop IS
PORT (
      clk,
      reset,
      d     : IN STD_LOGIC;
      q     : OUT STD_LOGIC
);
END flipflop;

ARCHITECTURE estrutura OF flipflop IS
BEGIN
      PROCESS(clk, reset)
      BEGIN
            IF(reset = '1') THEN
                  q <= '0';
            ELSIF(clk'EVENT AND clk = '1') THEN
                  q <= d;
            END IF;
      END PROCESS;
END estrutura;
```

Arquivo do bloco operativo, utiliza os componentes “registrador”, “sineDecoder” e “somador”:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY datapath IS
PORT (
      clk,
      dir,
```

```
      rst   : IN STD_LOGIC;
      max
      zero  : OUT STD_LOGIC;
      saida : OUT STD_LOGIC_VECTOR(33 DOWNTO 0)
);
END datapath;

ARCHITECTURE estrutura OF datapath IS

COMPONENT registrador IS
PORT (
      clk,
      reset : IN STD_LOGIC;
      d,
      q     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
);
END COMPONENT;

COMPONENT sineDecoder IS
PORT (
      A : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      Y : OUT STD_LOGIC_VECTOR(33 DOWNTO 0)
);
END COMPONENT ;

COMPONENT somador IS
PORT (
      A,
      B: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      S: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
);
END component;

SIGNAL outReg, inReg, outDecoder : STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN
      REG  : registrador PORT MAP(  clk => clk,
                                    rst => reset,
                                    inReg => d,
                                    outReg => q);
      dec0 : sineDecoder PORT MAP(  outReg => A,
                                    saida => Y);
      add : somador PORT
            MAP(  outReg => A,
                  dir & dir & dir & dir & dir & dir & dir & '1' => B,
                  inReg => S);
      max <=      outReg(7) AND outReg(6) AND outReg(5) AND
                  outReg(4) AND outReg(3) AND outReg(2) AND
                  outReg(1) AND NOT(outReg(0));
      zero <=     NOT(outReg(7)) AND NOT(outReg(6)) AND
                  NOT(outReg(5)) AND NOT(outReg(4)) AND
                  NOT(outReg(3)) AND NOT(outReg(2)) AND
                  NOT(outReg(1)) AND outReg(0);
```

```
END estrutura;
```

Arquivo do bloco “registrador”:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY registrador IS
PORT (
      clk,
      reset : IN STD_LOGIC;
      d     : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      q     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
);
END registrador;

ARCHITECTURE estrutura OF registrador IS
BEGIN
      PROCESS(clk, reset)
      BEGIN
            IF(reset = '1') THEN
                  q <= "00000000";
            ELSIF(clk'EVENT AND clk = '1') THEN
                  q <= d;
            END IF;
      END PROCESS;
END estrutura;
```

Arquivo do bloco “LUT”:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY sineDecoder IS
PORT (
     A     : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      Y    : OUT STD_LOGIC_VECTOR(33 DOWNTO 0)
);
END sineDecoder ;

ARCHITECTURE comportamento OF sineDecoder IS
BEGIN
Y <=
"0000000000000000000000000000000000" WHEN A =  "00000000" ELSE
"0000000000000000000000000000000010" WHEN A =  "00000001" ELSE
"0000000000000000000000000000000011" WHEN A =  "00000010" ELSE
"0000000000000000000000000000000101" WHEN A =  "00000011" ELSE
"0000000000000000000000000000000110" WHEN A =  "00000100" ELSE
"0000000000000000000000000000001000" WHEN A =  "00000101" ELSE
```

```
"00000000000000000000000000001001" WHEN A =  "00000110" ELSE
"00000000000000000000000000001011" WHEN A =  "00000111" ELSE
"00000000000000000000000000001101" WHEN A =  "00001000" ELSE
"00000000000000000000000000001110" WHEN A =  "00001001" ELSE
"00000000000000000000000000011000" WHEN A =  "00001010" ELSE
"00000000000000000000000000011001" WHEN A =  "00001011" ELSE
"00000000000000000000000000011011" WHEN A =  "00001100" ELSE
"00000000000000000000000000011100" WHEN A =  "00001101" ELSE
"00000000000000000000000000011110" WHEN A =  "00001110" ELSE
"00000000000000000000000000111000" WHEN A =  "00001111" ELSE
"00000000000000000000000000111001" WHEN A =  "00010000" ELSE
"00000000000000000000000000111011" WHEN A =  "00010001" ELSE
"00000000000000000000000000111100" WHEN A =  "00010010" ELSE
"00000000000000000000000000111110" WHEN A =  "00010011" ELSE
"00000000000000000000000000111111" WHEN A =  "00010100" ELSE
"00000000000000000000000001111001" WHEN A =  "00010101" ELSE
"00000000000000000000000001111010" WHEN A =  "00010110" ELSE
"00000000000000000000000001111100" WHEN A =  "00010111" ELSE
"00000000000000000000000001111110" WHEN A =  "00011000" ELSE
"00000000000000000000000001111111" WHEN A =  "00011001" ELSE
"00000000000000000000000011111001" WHEN A =  "00011010" ELSE
"00000000000000000000000011111010" WHEN A =  "00011011" ELSE
"00000000000000000000000011111100" WHEN A =  "00011100" ELSE
"00000000000000000000000011111101" WHEN A =  "00011101" ELSE
"00000000000000000000000011111111" WHEN A =  "00011110" ELSE
"00000000000000000000000111111000" WHEN A =  "00011111" ELSE
"00000000000000000000000111111010" WHEN A =  "00100000" ELSE
"00000000000000000000000111111011" WHEN A =  "00100001" ELSE
"00000000000000000000000111111101" WHEN A =  "00100010" ELSE
"00000000000000000000000111111111" WHEN A =  "00100011" ELSE
"00000000000000000000001111111000" WHEN A =  "00100100" ELSE
"00000000000000000000001111111010" WHEN A =  "00100101" ELSE
"00000000000000000000001111111011" WHEN A =  "00100110" ELSE
"00000000000000000000001111111101" WHEN A =  "00100111" ELSE
"00000000000000000000001111111110" WHEN A =  "00101000" ELSE
"00000000000000000000011111111000" WHEN A =  "00101001" ELSE
"00000000000000000000011111111001" WHEN A =  "00101010" ELSE
"00000000000000000000011111111011" WHEN A =  "00101011" ELSE
"00000000000000000000011111111100" WHEN A =  "00101100" ELSE
"00000000000000000000011111111110" WHEN A =  "00101101" ELSE
"00000000000000000000011111111111" WHEN A =  "00101110" ELSE
"00000000000000000000111111111001" WHEN A =  "00101111" ELSE
"00000000000000000000111111111010" WHEN A =  "00110000" ELSE
"00000000000000000000111111111100" WHEN A =  "00110001" ELSE
"00000000000000000000111111111101" WHEN A =  "00110010" ELSE
"00000000000000000000111111111111" WHEN A =  "00110011" ELSE
"00000000000000000001111111111000" WHEN A =  "00110100" ELSE
"00000000000000000001111111111010" WHEN A =  "00110101" ELSE
"00000000000000000001111111111011" WHEN A =  "00110110" ELSE
"00000000000000000001111111111101" WHEN A =  "00110111" ELSE
"00000000000000000001111111111110" WHEN A =  "00111000" ELSE
"00000000000000000011111111111000" WHEN A =  "00111001" ELSE
"00000000000000000011111111111001" WHEN A =  "00111010" ELSE
```

```
"00000000000000000000111111111011" WHEN A =  "00111011" ELSE
"00000000000000000000111111111100" WHEN A =  "00111100" ELSE
"00000000000000000000111111111110" WHEN A =  "00111101" ELSE
"00000000000000000000111111111111" WHEN A =  "00111110" ELSE
"00000000000000000001111111111001" WHEN A =  "00111111" ELSE
"00000000000000000001111111111010" WHEN A =  "01000000" ELSE
"00000000000000000001111111111011" WHEN A =  "01000001" ELSE
"00000000000000000001111111111101" WHEN A =  "01000010" ELSE
"00000000000000000001111111111110" WHEN A =  "01000011" ELSE
"00000000000000000001111111111000" WHEN A =  "01000100" ELSE
"00000000000000000011111111111001" WHEN A =  "01000101" ELSE
"00000000000000000011111111111011" WHEN A =  "01000110" ELSE
"00000000000000000011111111111100" WHEN A =  "01000111" ELSE
"00000000000000000011111111111101" WHEN A =  "01001000" ELSE
"00000000000000000011111111111111" WHEN A =  "01001001" ELSE
"00000000000000000011111111111000" WHEN A =  "01001010" ELSE
"00000000000000000011111111111010" WHEN A =  "01001011" ELSE
"00000000000000000011111111111011" WHEN A =  "01001100" ELSE
"00000000000000000011111111111101" WHEN A =  "01001101" ELSE
"00000000000000000011111111111110" WHEN A =  "01001110" ELSE
"00000000000000000011111111111111" WHEN A =  "01001111" ELSE
"00000000000000000111111111111001" WHEN A =  "01010000" ELSE
"00000000000000000111111111111010" WHEN A =  "01010001" ELSE
"00000000000000000111111111111011" WHEN A =  "01010010" ELSE
"00000000000000000111111111111101" WHEN A =  "01010011" ELSE
"00000000000000000111111111111110" WHEN A =  "01010100" ELSE
"00000000000000000111111111111000" WHEN A =  "01010101" ELSE
"00000000000000001111111111111001" WHEN A =  "01010110" ELSE
"00000000000000001111111111111010" WHEN A =  "01010111" ELSE
"00000000000000001111111111111100" WHEN A =  "01011000" ELSE
"00000000000000001111111111111101" WHEN A =  "01011001" ELSE
"00000000000000001111111111111110" WHEN A =  "01011010" ELSE
"00000000000000001111111111111000" WHEN A =  "01011011" ELSE
"00000000000000011111111111111001" WHEN A =  "01011100" ELSE
"00000000000000011111111111111010" WHEN A =  "01011101" ELSE
"00000000000000011111111111111100" WHEN A =  "01011110" ELSE
"00000000000000011111111111111101" WHEN A =  "01011111" ELSE
"00000000000000011111111111111110" WHEN A =  "01100000" ELSE
"00000000000000011111111111111000" WHEN A =  "01100001" ELSE
"00000000000000111111111111111001" WHEN A =  "01100010" ELSE
"00000000000000111111111111111010" WHEN A =  "01100011" ELSE
"00000000000000111111111111111011" WHEN A =  "01100100" ELSE
"00000000000000111111111111111101" WHEN A =  "01100101" ELSE
"00000000000000111111111111111110" WHEN A =  "01100110" ELSE
"00000000000000111111111111111111" WHEN A =  "01100111" ELSE
"00000000000001111111111111111000" WHEN A =  "01101000" ELSE
"00000000000001111111111111111010" WHEN A =  "01101001" ELSE
"00000000000001111111111111111011" WHEN A =  "01101010" ELSE
"00000000000001111111111111111100" WHEN A =  "01101011" ELSE
"00000000000001111111111111111101" WHEN A =  "01101100" ELSE
"00000000000001111111111111111111" WHEN A =  "01101101" ELSE
"00000000000001111111111111111000" WHEN A =  "01101110" ELSE
"00000000000001111111111111111001" WHEN A =  "01101111" ELSE
```

```
"00000000000111111111111111111010" WHEN A =  "01110000" ELSE
"00000000000111111111111111111100" WHEN A =  "01110001" ELSE
"00000000000111111111111111111101" WHEN A =  "01110010" ELSE
"00000000000111111111111111111110" WHEN A =  "01110011" ELSE
"00000000000111111111111111111111" WHEN A =  "01110100" ELSE
"00000000001111111111111111111000" WHEN A =  "01110101" ELSE
"00000000001111111111111111111010" WHEN A =  "01110110" ELSE
"00000000001111111111111111111011" WHEN A =  "01110111" ELSE
"00000000001111111111111111111100" WHEN A =  "01111000" ELSE
"00000000001111111111111111111101" WHEN A =  "01111001" ELSE
"00000000001111111111111111111110" WHEN A =  "01111010" ELSE
"00000000001111111111111111111111" WHEN A =  "01111011" ELSE
"00000000011111111111111111111001" WHEN A =  "01111100" ELSE
"00000000011111111111111111111010" WHEN A =  "01111101" ELSE
"00000000011111111111111111111011" WHEN A =  "01111110" ELSE
"00000000011111111111111111111100" WHEN A =  "01111111" ELSE
"00000000011111111111111111111101" WHEN A =  "10000000" ELSE
"00000000011111111111111111111110" WHEN A =  "10000001" ELSE
"00000000011111111111111111111111" WHEN A =  "10000010" ELSE
"00000000111111111111111111111000" WHEN A =  "10000011" ELSE
"00000000111111111111111111111001" WHEN A =  "10000100" ELSE
"00000000111111111111111111111010" WHEN A =  "10000101" ELSE
"00000000111111111111111111111100" WHEN A =  "10000110" ELSE
"00000000111111111111111111111101" WHEN A =  "10000111" ELSE
"00000000111111111111111111111110" WHEN A =  "10001000" ELSE
"00000000111111111111111111111111" WHEN A =  "10001001" ELSE
"00000001111111111111111111111000" WHEN A =  "10001010" ELSE
"00000001111111111111111111111001" WHEN A =  "10001011" ELSE
"00000001111111111111111111111010" WHEN A =  "10001100" ELSE
"00000001111111111111111111111011" WHEN A =  "10001101" ELSE
"00000001111111111111111111111100" WHEN A =  "10001110" ELSE
"00000001111111111111111111111101" WHEN A =  "10001111" ELSE
"00000001111111111111111111111110" WHEN A =  "10010000" ELSE
"00000001111111111111111111111111" WHEN A =  "10010001" ELSE
"00000011111111111111111111111000" WHEN A =  "10010010" ELSE
"00000011111111111111111111111001" WHEN A =  "10010011" ELSE
"00000011111111111111111111111010" WHEN A =  "10010100" ELSE
"00000011111111111111111111111011" WHEN A =  "10010101" ELSE
"00000011111111111111111111111100" WHEN A =  "10010110" ELSE
"00000011111111111111111111111101" WHEN A =  "10010111" ELSE
"00000011111111111111111111111110" WHEN A =  "10011000" ELSE
"00000011111111111111111111111111" WHEN A =  "10011001" ELSE
"00000111111111111111111111111111" WHEN A =  "10011010" ELSE
"00000111111111111111111111111000" WHEN A =  "10011011" ELSE
"00000111111111111111111111111001" WHEN A =  "10011100" ELSE
"00000111111111111111111111111010" WHEN A =  "10011101" ELSE
"00000111111111111111111111111011" WHEN A =  "10011110" ELSE
"00000111111111111111111111111100" WHEN A =  "10011111" ELSE
"00000111111111111111111111111101" WHEN A =  "10100000" ELSE
"00000111111111111111111111111110" WHEN A =  "10100001" ELSE
"00000111111111111111111111111111" WHEN A =  "10100010" ELSE
"00001111111111111111111111111111" WHEN A =  "10100011" ELSE
"00001111111111111111111111111000" WHEN A =  "10100100" ELSE
```

```
"00001111111111111111111111111001" WHEN A =  "10100101" ELSE
"00001111111111111111111111111010" WHEN A =  "10100110" ELSE
"00001111111111111111111111111011" WHEN A =  "10100111" ELSE
"00001111111111111111111111111100" WHEN A =  "10101000" ELSE
"00001111111111111111111111111100" WHEN A =  "10101001" ELSE
"00001111111111111111111111111101" WHEN A =  "10101010" ELSE
"00001111111111111111111111111110" WHEN A =  "10101011" ELSE
"00001111111111111111111111111111" WHEN A =  "10101100" ELSE
"00011111111111111111111111111000" WHEN A =  "10101101" ELSE
"00011111111111111111111111111000" WHEN A =  "10101110" ELSE
"00011111111111111111111111111001" WHEN A =  "10101111" ELSE
"00011111111111111111111111111010" WHEN A =  "10110000" ELSE
"00011111111111111111111111111011" WHEN A =  "10110001" ELSE
"00011111111111111111111111111011" WHEN A =  "10110010" ELSE
"00011111111111111111111111111100" WHEN A =  "10110011" ELSE
"00011111111111111111111111111101" WHEN A =  "10110100" ELSE
"00011111111111111111111111111101" WHEN A =  "10110101" ELSE
"00011111111111111111111111111110" WHEN A =  "10110110" ELSE
"00011111111111111111111111111111" WHEN A =  "10110111" ELSE
"00011111111111111111111111111111" WHEN A =  "10111000" ELSE
"00111111111111111111111111111000" WHEN A =  "10111001" ELSE
"00111111111111111111111111111001" WHEN A =  "10111010" ELSE
"00111111111111111111111111111001" WHEN A =  "10111011" ELSE
"00111111111111111111111111111010" WHEN A =  "10111100" ELSE
"00111111111111111111111111111011" WHEN A =  "10111101" ELSE
"00111111111111111111111111111011" WHEN A =  "10111110" ELSE
"00111111111111111111111111111100" WHEN A =  "10111111" ELSE
"00111111111111111111111111111101" WHEN A =  "11000000" ELSE
"00111111111111111111111111111101" WHEN A =  "11000001" ELSE
"00111111111111111111111111111110" WHEN A =  "11000010" ELSE
"00111111111111111111111111111110" WHEN A =  "11000011" ELSE
"00111111111111111111111111111111" WHEN A =  "11000100" ELSE
"00111111111111111111111111111111" WHEN A =  "11000101" ELSE
"01111111111111111111111111111000" WHEN A =  "11000110" ELSE
"01111111111111111111111111111001" WHEN A =  "11000111" ELSE
"01111111111111111111111111111001" WHEN A =  "11001000" ELSE
"01111111111111111111111111111010" WHEN A =  "11001001" ELSE
"01111111111111111111111111111010" WHEN A =  "11001010" ELSE
"01111111111111111111111111111011" WHEN A =  "11001011" ELSE
"01111111111111111111111111111011" WHEN A =  "11001100" ELSE
"01111111111111111111111111111100" WHEN A =  "11001101" ELSE
"01111111111111111111111111111100" WHEN A =  "11001110" ELSE
"01111111111111111111111111111101" WHEN A =  "11001111" ELSE
"01111111111111111111111111111101" WHEN A =  "11010000" ELSE
"01111111111111111111111111111101" WHEN A =  "11010001" ELSE
"01111111111111111111111111111110" WHEN A =  "11010010" ELSE
"01111111111111111111111111111110" WHEN A =  "11010011" ELSE
"01111111111111111111111111111111" WHEN A =  "11010100" ELSE
"01111111111111111111111111111111" WHEN A =  "11010101" ELSE
"11111111111111111111111111111000" WHEN A =  "11010110" ELSE
"11111111111111111111111111111000" WHEN A =  "11010111" ELSE
"11111111111111111111111111111000" WHEN A =  "11011000" ELSE
"11111111111111111111111111111001" WHEN A =  "11011001" ELSE
```

```
"11111111111111111111111111111001" WHEN A =  "11011010" ELSE
"11111111111111111111111111111001" WHEN A =  "11011011" ELSE
"11111111111111111111111111111010" WHEN A =  "11011100" ELSE
"11111111111111111111111111111010" WHEN A =  "11011101" ELSE
"11111111111111111111111111111010" WHEN A =  "11011110" ELSE
"11111111111111111111111111111011" WHEN A =  "11011111" ELSE
"11111111111111111111111111111011" WHEN A =  "11100000" ELSE
"11111111111111111111111111111011" WHEN A =  "11100001" ELSE
"11111111111111111111111111111100" WHEN A =  "11100010" ELSE
"11111111111111111111111111111100" WHEN A =  "11100011" ELSE
"11111111111111111111111111111100" WHEN A =  "11100100" ELSE
"11111111111111111111111111111100" WHEN A =  "11100101" ELSE
"11111111111111111111111111111101" WHEN A =  "11100110" ELSE
"11111111111111111111111111111101" WHEN A =  "11100111" ELSE
"11111111111111111111111111111101" WHEN A =  "11101000" ELSE
"11111111111111111111111111111101" WHEN A =  "11101001" ELSE
"11111111111111111111111111111110" WHEN A =  "11101010" ELSE
"11111111111111111111111111111110" WHEN A =  "11101011" ELSE
"11111111111111111111111111111110" WHEN A =  "11101100" ELSE
"11111111111111111111111111111110" WHEN A =  "11101101" ELSE
"11111111111111111111111111111110" WHEN A =  "11101110" ELSE
"11111111111111111111111111111111" WHEN A =  "11101111" ELSE
"11111111111111111111111111111111" WHEN A =  "11110000" ELSE
"11111111111111111111111111111111" WHEN A =  "11110001" ELSE
"11111111111111111111111111111111" WHEN A =  "11110010" ELSE
"11111111111111111111111111111111" WHEN A =  "11110011" ELSE
"11111111111111111111111111111111" WHEN A =  "11110100" ELSE
"11111111111111111111111111111111" WHEN A =  "11110101" ELSE
"11111111111111111111111111111111" WHEN A =  "11110110" ELSE
"11111111111111111111111111111111" WHEN A =  "11110111" ELSE
"11111111111111111111111111111111" WHEN A =  "11111000" ELSE
"11111111111111111111111111111111" WHEN A =  "11111001" ELSE
"11111111111111111111111111111111" WHEN A =  "11111010" ELSE
"11111111111111111111111111111111" WHEN A =  "11111011" ELSE
"11111111111111111111111111111111" WHEN A =  "11111100" ELSE
"11111111111111111111111111111111" WHEN A =  "11111101" ELSE
"11111111111111111111111111111111" WHEN A =  "11111110" ELSE
"11111111111111111111111111111111";
END comportamento;
```

Arquivo do componente “somador”, utiliza os componentes “halfadder” e “fulladder”:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY somador IS
PORT (
     A,
     B     : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
     S     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
     );
```

```
END somador;

ARCHITECTURE circuito_logico OF somador IS

COMPONENT fulladder IS
PORT(
     A,
     B,
     Cin   : IN STD_LOGIC;
     S,
     Cout  : OUT STD_LOGIC
);
END COMPONENT;

COMPONENT halfadder IS
PORT(
     A,
     B     : IN STD_LOGIC;
     S,
     C     : OUT STD_LOGIC
);
END COMPONENT;

SIGNAL Cout: STD_LOGIC_VECTOR (8 DOWNTO 0);

BEGIN

cpa_0 : halfadder PORT MAP(  A => A(0),
                             B => B(0),
                             S => S(0),
                             C => Cout(0));

cpa_1 : FOR j IN 1 TO 7 GENERATE
     cpa_j: fulladder PORT MAP(    A => A(j),
                                   B => B(j),
                                   Cin => Cout(j-1),
                                   S => S(j),
                                   Cout => Cout(j));
END generate cpa_1;
END circuito_logico;
```

Componente “fulladder”:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fulladder IS
PORT(
     A,
     B,
     Cin   : IN STD_LOGIC;
     S,
     Cout  : OUT STD_LOGIC
```

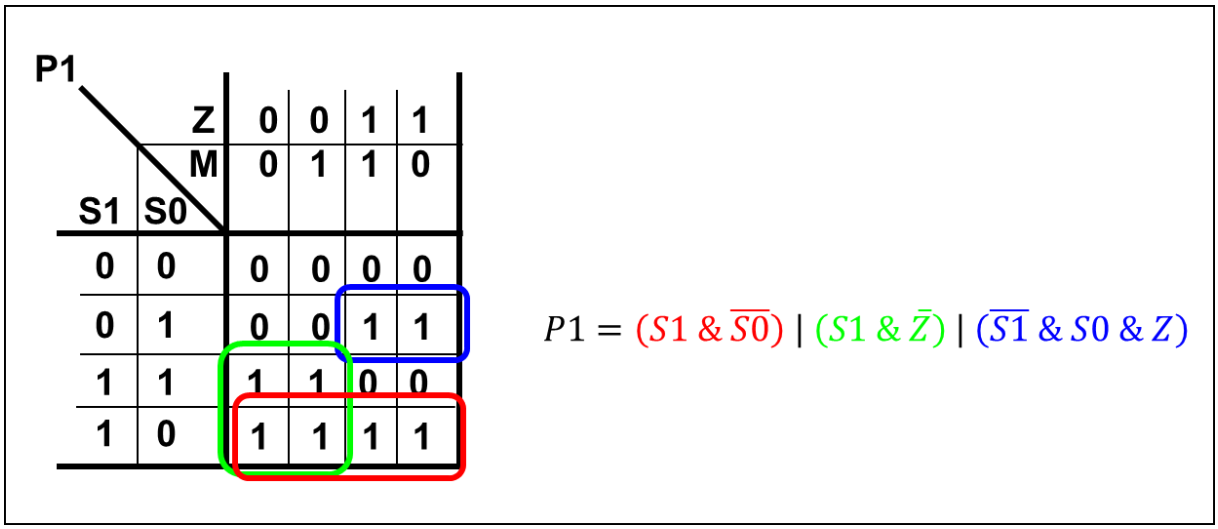
```
        );
        END fulladder;

        ARCHITECTURE circuito OF fulladder IS

        SIGNAL F1, F2, F3 : STD_LOGIC;

        BEGIN
                    F1 <= A XOR B;
                    F2 <= A AND B;
                    S <= F1 XOR Cin;
                    F3 <= F1 AND Cin;
                    Cout <= F2 OR F3;
        END circuito;
```

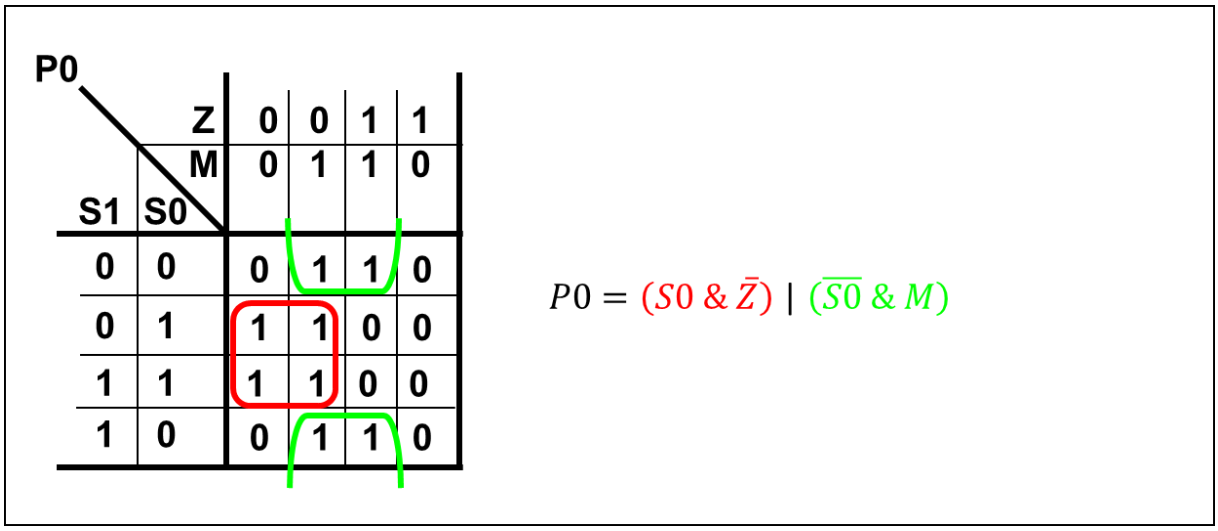
Componente “halfadder”:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY halfadder IS
PORT(
      A,
      B     : IN STD_LOGIC;
      S,
      C     : OUT STD_LOGIC
);
END halfadder;

ARCHITECTURE circuito OF halfadder IS

BEGIN
            S <= A XOR B;
            C <= A AND B;
END circuito;
```

# 8 APÊNDICE B – SIMPLIFICAÇÃO EM MAPAS DE KARNAUGH

Conforme citado na seção 3.3, foi realizada a simplificação manual das equações lógicas utilizando o método dos mapas de Karnaugh. Utilizando a Tabela 3, foram montados 5 mapas de Karnaugh para as 5 variáveis a serem determinadas. Abaixo, se encontram os mapas ao lado de suas respectivas equações simplificadas utilizando a mesma notação para as variáveis que havia sido utilizada na seção 3.2.2.



| Sp | Z | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|
| | M | 0 | 1 | 1 | 0 |
| S1 | S0 | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 |

$Sp = S1$

| Sn | Z | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|
| | M | 0 | 1 | 1 | 0 |
| S1 | S0 | | | | |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |

$Sn = \overline{S1}$

| D | Z | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|
| | M | 0 | 1 | 1 | 0 |
| S1 | S0 | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |

$D = S0$