Projeto de Graduação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

10 de Dezembro de 2019

Controle de Drift da função de transferência de modulador de Mach-Zehnder

Breno Perlingeiro Corrêa

PUC RIO

Projeto de Graduação

DEPARTAMENTO
DE ENGENHARIA
ELÉTRICA

Controle de Drift da função de transferência de modulador de Mach-Zehnder

Aluno: Breno Perlingeiro Corrêa

Orientador: Guilherme Penello Temporão

Corientador: Gustavo Castro do Amaral

Trabalho apresentado com requisito parcial à conclusão do curso de Engenharia Elétrica na Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

Agradecimentos

Agradeço aos meus avós que me deram suporte para chegar até aqui e cuidaram de mim, à minha mãe que, apesar da distância, me apoiou e aconselhou em todos os momentos, à minha namorada pelo apoio emocional, ao professor Gustavo do Amaral pela orientação e a Felipe Calliari, Pedro Tovar e Christiano Nascimento pela ajuda no projeto. No mais, sou grato aos outros familiares e amigos.

Resumo

Nas comunicações ópticas é muito utilizado a modulação de amplitude para transmitir dados e o equipamento mais comum para tal é o modulador de Mach-Zhender. Apesar de ser muito usado, possui certos empecilhos. A função de transferência é uma senoide que relaciona tensão de entrada e potência óptica de saída. Por isso, ao utiliza-lo é necessário ligar uma tensão de bias que o faz operar no regime mais linear possível.

Porém, com o passar do tempo, a tensão de bias necessária para operação linear é deslocada. Neste projeto será analisado um método autônomo de corrigir o erro do bias. O método será realizado a partir da modulação de um tom de baixa frequência para não atrapalhar a comunicação e servir de indicador do drift.

Basicamente, o sistema monitora a operação do modulador a partir do sinal modulado, comparando a magnitude do tom de baixa frequência e o segundo harmônico dele. Por isso, foram estudados algoritmos de analise espectral para detectar a influêcia das frequencias de interesse e terminar realizando um controle PID com a razão de suas magnitudes.

Palavras-chave: Comunicações ópticas, Modulador de Mach-Zehnder, FPGA, Algoritmo Goertzel, Arduino, PID, Optoeletrônica

Mach-Zehnder modulator transfer function Drift Control

Abstract

In optical communications, amplitude modulation is widely used to transmit data and the most common equipment for that is Mach-Zhender modulators. Despite being widely used, has certain obstacles. The transfer function is an sinusoidal curve that relates input voltage to output optical power. Therefore, when using it is necessary to connect a bias voltage that makes it operate in the most linear regime possible.

However, as the time passes by, the bias voltage required for linear operation shifts. This project will analyze a method of autonomously correcting the bias error. The method will be performed by modulating a low frequency tone so as not to disturb communication and serve as an indicator of drift.

Basically, the system monitors the modulator operation from the modulated signal, comparing between the magnitude of the low frequency tone and it's second harmonic. Therefore, spectral analysis algorithms were studied to detect the influence of the frequencies of interest and concluding performing a PID control with the ratio of their magnitudes.

Keywords: Optical communication, Mach-Zehnder Modulator, FPGA, Goertzel algorithm, Arduino, PID, Optoeletronics

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Em um enlace de fibras óticas, um dos métodos mais comuns para transmissão de dados é por meio de modulação de amplitude (AM) da portadora óptica. Tal modulação pode ser implementada através de um sistema de interferometria eletro-óptico conhecido como modulador de Mach-Zehnder (MZM). Porém, este sistema apresenta um problema com o tempo de uso. Devido a fenômenos físicos no substrato do dispositivo, o ponto de operação do sistema é deslocado. Em razão deste drift da função de transferência, o sinal de saída do modulador é distorcido [1].

Recentemente, a análise de tal distorção no sinal de saída foi observada: conforme a fase foi deslocada, a amplitude do segundo harmônico do sinal modulante aumentou, indicando a resposta não-linear. Além disso, concluiu-se que o segundo harmônico pode ser analisado de forma a caracterizar a distorção do sinal [2]. Desta forma, este projeto tem o intuito de criar um controle da tensão de polarização a partir da magnitude do segundo harmônico.

O sistema de controle funcionaria utilizando um sinal de baixa frequência, desta forma não seria necessário parar a transmissão e nem distorceria as informações transmitidas. O único empecilho é que seá necessário um Bean spliter após saída do modulador, como um 90-10 ou 99-1, para realizar a análise da operação do modulador, a apartir da segunda harmônica. Nas secções a seguir serão apresentados o funcionamento do modulador de Mach-Zhender, o drift da sua função de transferência, o metodo e os equipamentos utilizados para medir a segunda harmônica no sinal e a elaboração do sistema de controle.

2 Modulador de Mach-Zehnder

a Modulador de Amplitude

O modulador de Mach-Zehnder consiste em um divisor de feixes óticos (Beam Splitter) que separa o campo elétrico na entrada igualmente entre os braços do interferômetro. Em sua configuração mais simples, um dos braços possui um elemento que apresenta o chamado efeito Pockels (a variação linear do índice de refração em função de um campo elétrico externo). Por fim, 3 eletrodos permitem a aplicação do sinal modulante e um acoplador de feixes ópticos une os sinais provenientes dos dois braços [1].

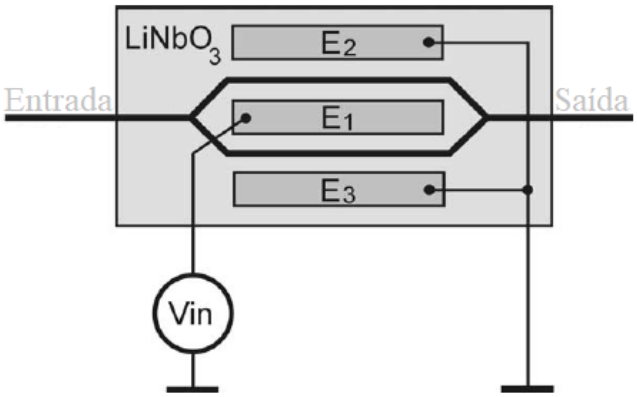

Figura 1: Configuração do modulador de Mach-Zehnder [1].

O funcionamento do modulador consiste na variação do índice de refração do substrato a partir da excitação dos eletrodos pelo sinal de entrada. Índices de refração diferentes implicam em uma defasagem das ondas que percorrem os braços, que, ao se recombinarem, podem gerar interações destrutivas, se a diferença de fase for de 180°, ou construtivas, se não houver diferença de fase. De modo geral, a função de transferência (definida pela potência de saída em função da diferença de potencial) do MZM é representada por um cosseno, como apresentado a seguir.

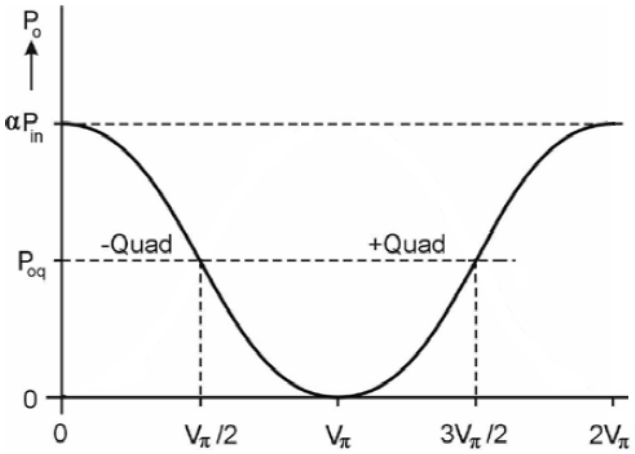

Figura 2: Função de transferência do Modulador de Mach-Zehnder [1].

$$P_o = \alpha \frac{P_{in}}{2} \left[1 + cos \left(\frac{V_{in}}{V_\pi} \pi \right) \right] \tag{1}$$

Sendo P_o e P_{in} a potência ótica na saída e na entrada do modulador, respectivamente. A tensão de entrada é representada por V_in e V_π é a tensão que causa a interação puramente destrutiva, ou seja,

anula a potência na saída, além disso é um dos parâmetros apresentados na folha de características do dispositivo. O coeficiente α representa a perda de inserção do modulador quando a saída é máxima, ou seja, interação puramente construtiva. A figura 2 apresenta o gráfico da função de transferência do MZM.

Como pode ser visto, o modulador não é linear; entretanto, nas proximidades das tensões de $3V_\pi/2$ e $V_\pi/2$ (comumente referidas como quadratura), a função pode ser considerada linear se o sinal modulado for limitado a um certo intervalo de tensões. Note que, para $V_\pi/2$, apesar de linear, o coeficiente associado é negativo, já que incrementos na tensão geram reduções na potência ótica na saída e vice-versa. Portanto, em geral se utiliza uma tensão de polarização igual a $3V_\pi/2$ e um sinal AC com amplitude tal que a operação linear seja garantida. Portanto a tensão de entrada é composta por:

$$V_{in} = V_{DC} + V_{AC} \tag{2}$$

A partir disso, substituindo a tensão de entrada por esta e expandindo utilizando a soma dos arcos foi obtido:

$$P_o = \alpha \frac{P_{in}}{2}\left[1 + cos\left(\frac{V_{DC}}{V_\pi}\pi\right) cos\left(\frac{V_{AC}}{V_\pi}\pi\right) - sen\left(\frac{V_{DC}}{V_\pi}\pi\right) sen\left(\frac{V_{AC}}{V_\pi}\pi\right)\right] \tag{3}$$

E utilizando o valor sugerido para a tensão de polarização ($V_{DC} = 3V_\pi/2$) é obtido a seguinte função de transferência:

$$P_o = \alpha \frac{P_{in}}{2}\left[1 + sen\left(\frac{V_{AC}}{V_\pi}\pi\right)\right] \tag{4}$$

Por fim, utilizando a aproximação para pequenos ângulos é obtido a relação linear.

$$P_o = \alpha \frac{P_{in}}{2}\left[1 + \frac{V_{AC}}{V_\pi}\pi\right] \tag{5}$$

b Análise do Dirft da função de transferência do MZM

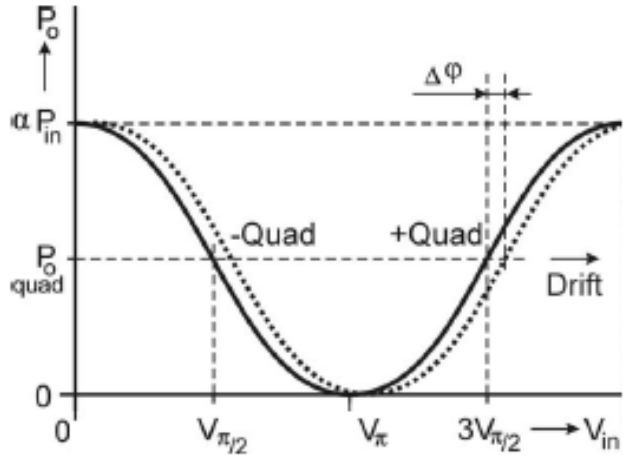

Figura 3: Ilustração do drift da função de transferência [1].

Em razão de fenômenos piroelétricos, fotorefrativos e fotocondutivos no substrato (comumente de $LiNbO_3$), a fase da função de transferência do modulador é alterada. Com a variação da fase o ponto

de operação é deslocado (representado pela linha tracejada na figura 3). Como a tensão de bias (V_{DC}) não acompanha a variação do V_π, o sistema começa a operar na região não-linear causando distorções no sinal de saída do MZM.

Estudos recentes analisaram a distorção harmônica total do sinal de saída do modulador de Mach-Zehnder com a tensão de bias errada. A conclusão foi que a componente mais efetiva na distorção é a segunda harmônica do sinal modulante e tem uma grande sensibilidade em relação as variações da tenção de polarização [2].

c Medidas do drift da função de transferência

A fim de averiguar a variação de fase da função de transferência do modulador de amplitude foram realizadas medidas desta em diferentes instantes. Para tal, fora utilizado um laser, um controlador de polarização, um modulador, um fotodetector, uma fonte DC, um Arduino Uno e um computador.

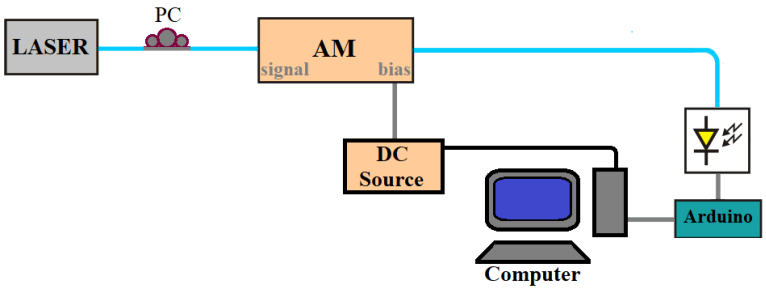

Figura 4: Sistema de medida utilizado para realizar a medida do drift da função de transferência.

A parte óptica do sistema é montada da seguinte forma. o laser passa pelo controle de polarização. A saída do modulador é ligado a um fotodetector com ganho controlável. Antes de realizar as medidas é necessário maximizar a potência de saída do AM variando a polarização com o PC e depois ajustar o ganho do detector para que a saída máxima não supere 5 V, para não danificar o conversor AD do Arduino Uno. Entretanto o ganho não pode ser muito baixo senão a medida fica sem resolução.

O sistema elétrico é implementado junto ao computador de forma que o sistema realize as medidas de forma autonoma. No computador foi implementado um código em Python que realiza a comunicação com o Arduino via USB e com a fonte DC via GPIB.

O código segue uma rotina para realizar a medida. Primeiro, é estabelecida a comunicação com os de mais aparelhos. Em seguida, varia a tensão da fonte de 0 a 6 V, em passos de 0.2 V. Ao fim de cada incremento, o Arduino realiza 100 medidas da saída fotodetector e envia ao computador que faz a média e salva em um vetor. Feito isso, o computador fixa a fonte em uma tensão por um tempo, ambos são parâmetros escolhidos pelo usuário.

Passado o tempo desejdo, o processo de variação de tensão e espera em uma tensão fixa se repete sequêncialmente. O número repetições é definida pelo usuário no código. Na ultima repetição não é realizado a espera e o vetor com todas medidas é salvo no computador em um arquivo CSV.

Como as medidas são referentes a potência de saída do MZM, as curvas são normalizadas. Após as medidas serem tiradas e normalizadas é realizado um tratamento dos dados obtidos. Com auxílio do MATLAB é feita a interpolação das funções de transferência medidas em instantes diferentes. A partir das curvas interpoladas é medido a diferença de tensão entre os pontos de quadratura de uma para a seguinte. Desta forma, é obtido um vetor com o drift a cada instante. Com a medida do drift em função do tempo, é realizado um fit no origin para obter as constante de tempo.

A primeira medida foi realizada esperando 20 minutos entre cada varição. Este processo se repitiu por 10 vezes, portanto permitindo observar o drift numa janela de 20 a 180 minutos. O intuito desta

medida foi para comparar com os resultados do sitema de medida com os apresentados em [3]. Pois, tanto o sistema quanto o modulador eram diferentes.

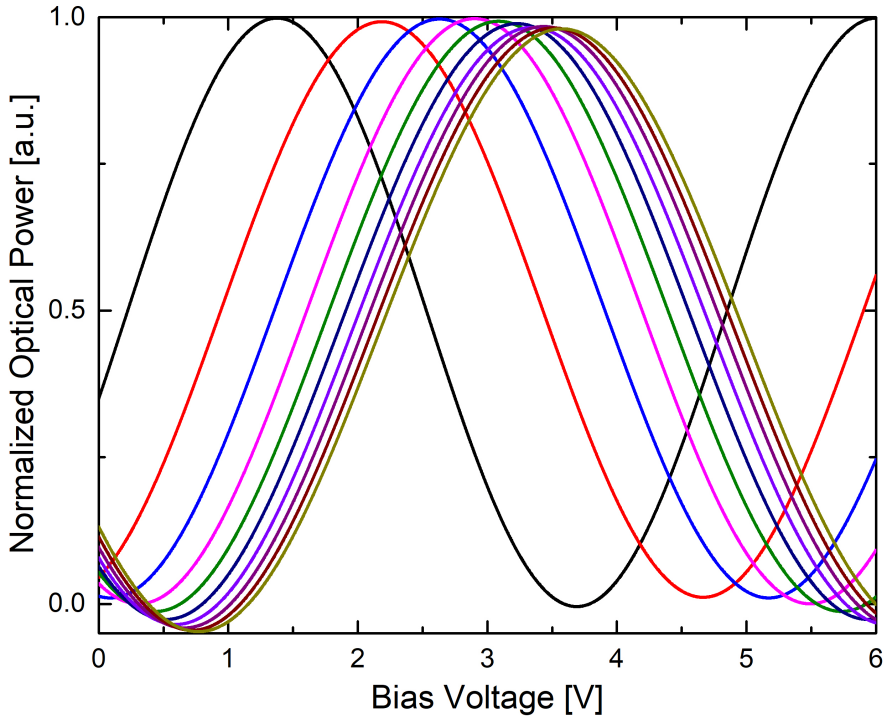

Figura 5: 10 medidas da função de transferência realizadas em sequência, esperando 20 minutos com a fonte fixa em 4 V.

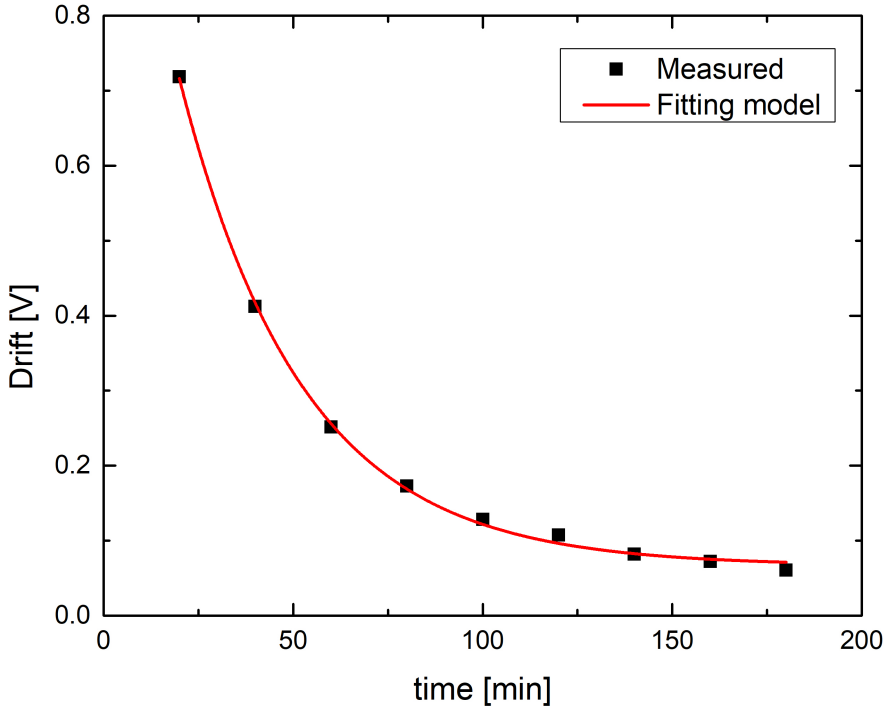

Figura 6: Amostras do drift em função do tempo e o fit exponecial relacionado a curva de 10 medidas esperando 20 minutos em 4 V.

Asim como no artigo, foi realizado o fitting exponecial, como a expressão 6 representa. Os resul-

tados encontrados foram similares aos do artigo. A tabela 1 apresenta as constantes obtidas no fit exponencial da curva e o fator R^2 que representa o percentual de erro que é eliminado na previsão do drift considerando esta expressão.

$$y = A \cdot e^{-x/\tau} + B \tag{6}$$

A [V]	τ [min]	B [V]	R^2
1.20185	22.51548	0.06665	0.99876

Tabela 1: Constantes obtidas no fitting para medida esperando 20 minutos em 4 V e coeficiente R^2.

Analisando os resultados, observa-se que drift é sempre positvio, ou seja o ponto de operação se desloca cada vez mais para direita. Como há a cnstante B, com passar de muito tempo o drift tende a ser constante. O τ está associado ao tempo que o sistema demora para se estabilizar.

A caracterização do drift é importante para o controle, pois apresenta como é o comportamento do mesmo em funação do tempo. Desta forma é possível entender a robustez necessária para realizar o controle. Apesar de ser uma informação muito importante a operação a longo prazo do drift é necessário entender seu funcionamento em pequenos intervalos de tempo.

Pensando nisso, foram realizadas novas medidas agora esperando apenas 2 minutos em 4 V. O tempo de espera de 2 minutos é o mínimo possível, pois o tempo para realizar uma medida variando de 0 a 6 V é de, aproximademente, 1 minuto. É esperado nesse caso encontrar uma tendência diferente conforme se aproxima do zero, algo que não comece em um valor fixo como a exponencial. Espera-se que o drift se comporte igual a uma resposta impulsional de um sistema criticamente amortecido.

Como agora as medidas duram menos tempo foram realizadas mais de 10 medidas por vez. A seguir, são apresentados os resultados, já interpolados, para 20, 30 e 40 medidas da funação de transferência em sequência.

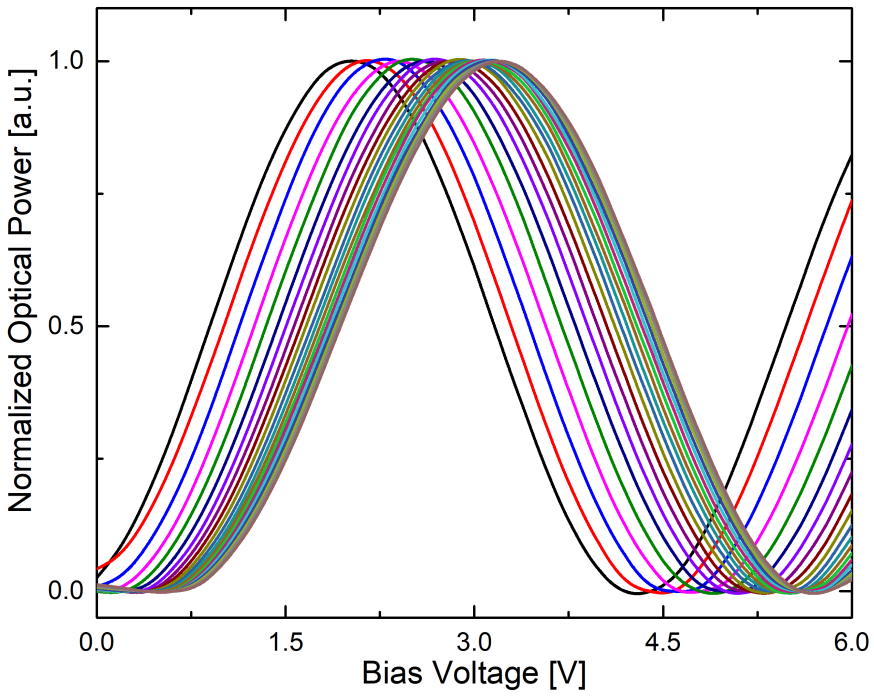

Figura 7: 20 medidas da função de transferência realizadas em sequência, esperando 2 minutos com a fonte fixa em 4 V.

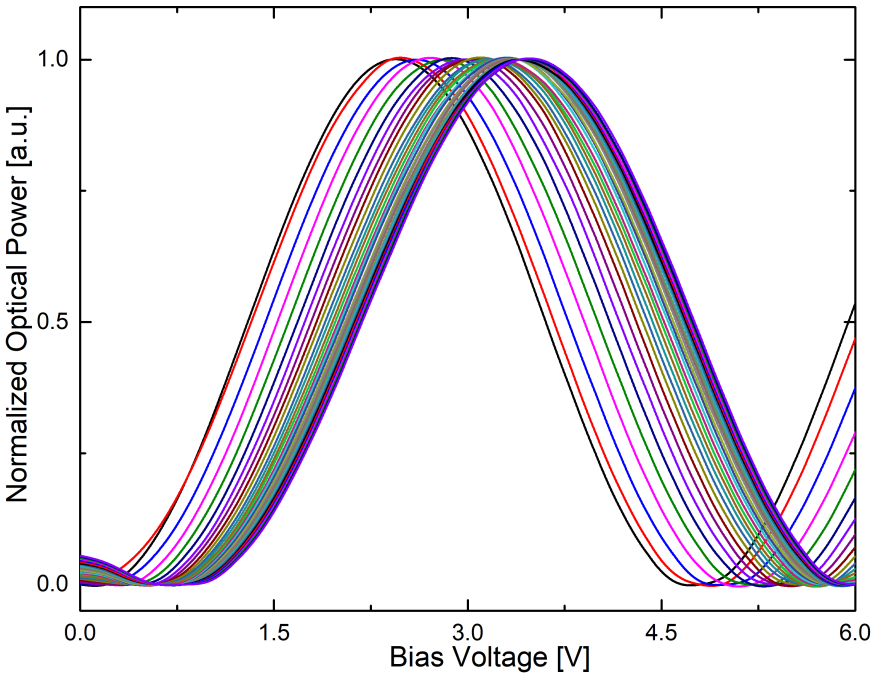

Figura 8: 30 medidas da função de transferência realizadas em sequência, esperando 2 minutos com a fonte fixa em 4 V.

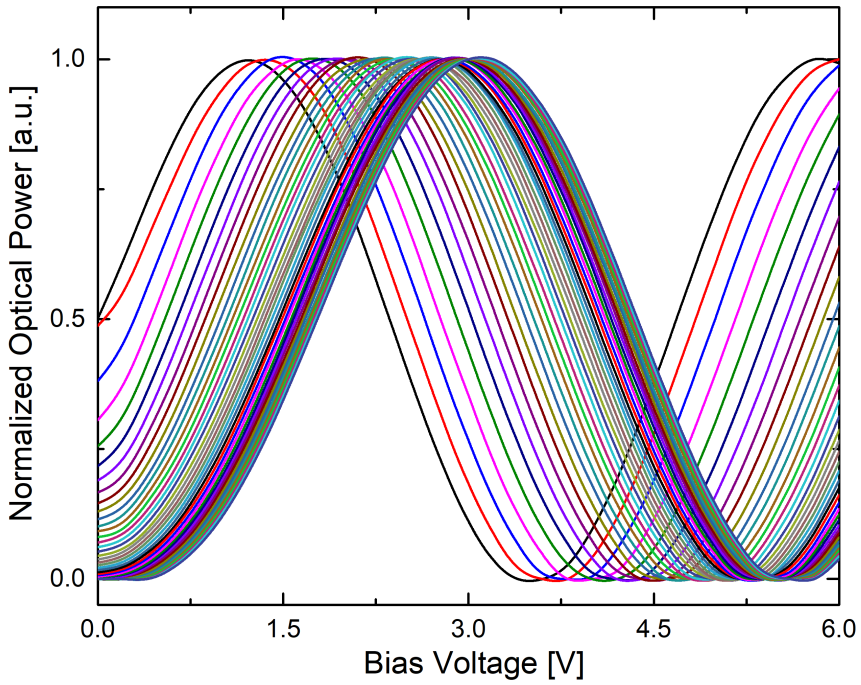

Figura 9: 40 medidas da função de transferência realizadas em sequência, esperando 2 minutos com a fonte fixa em 4 V.

A partir das curvas interpoladas foram calculadas as diferenças de tensão entre os pontos de operação em função do tempo. Para fitar as curvas foi utilizado uma expressão difernete, que tenha o comportamentoigual a resposta impulsional de um sistema críticamente amortecido.

$$y = C_1 \cdot x \cdot e^{-(x/\tau_1+\phi_1)} + C_2 \cdot e^{-(x/\tau_2+\phi_2)} + C_3 \tag{7}$$

Os resultados do fit das três curvas utilizando a expressão 7 seguem a baixo. Importante ressaltar que o drift nesse caso atingirá valores menores que os da medida de 20 minutos, pois o tempo de espera na tensão é próximo do tempo de variação. No caso de 20 minutos, a variação corresponde a 5% do tempo de uma medida, enquanto que para de 2 minutos, 33%.

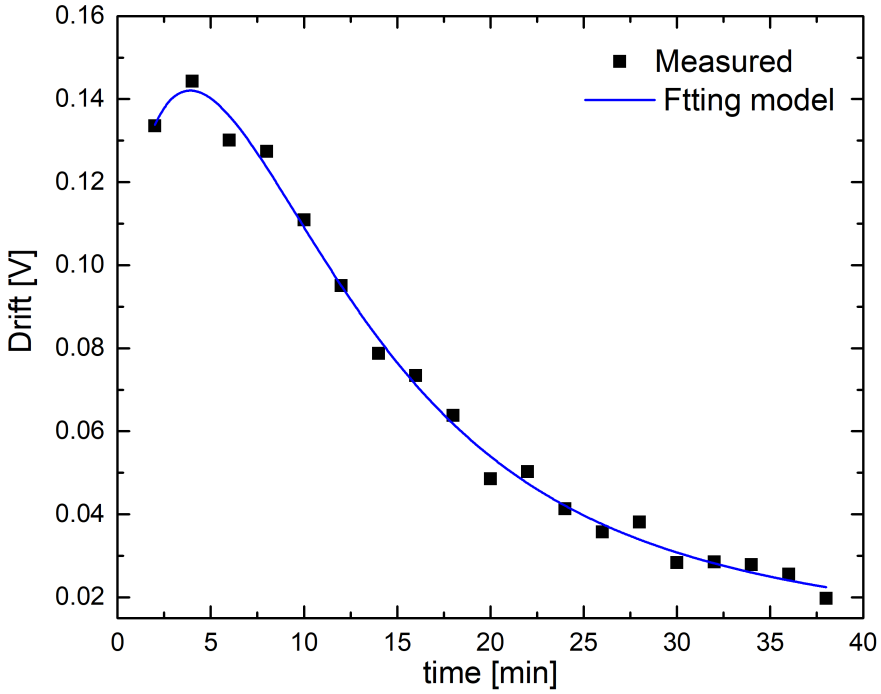

Figura 10: Amostras do drift em função do tempo e o fit exponecial relacionado a curva de 20 medidas esperando 2 minutos em 4 V.

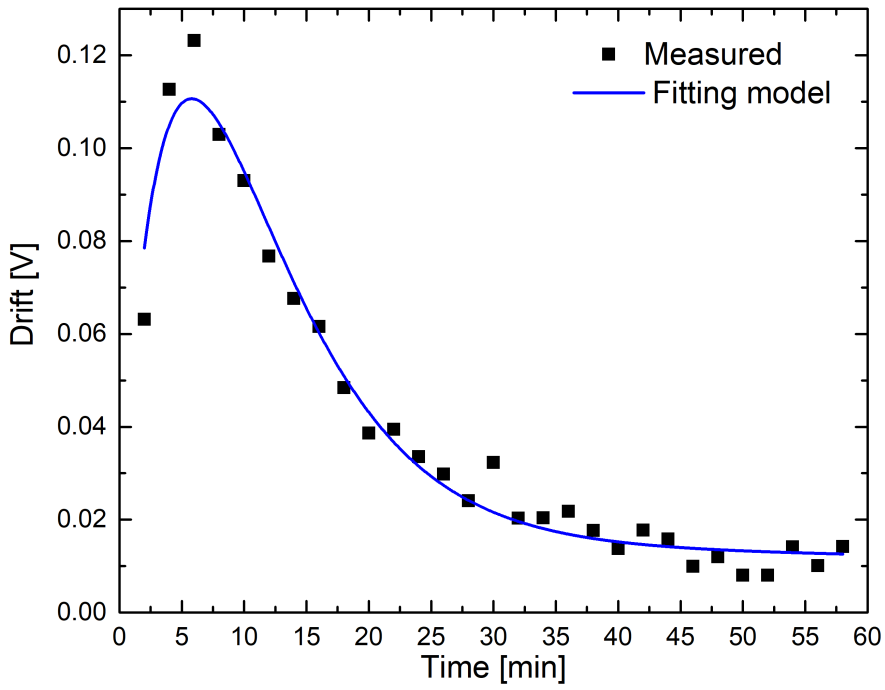

Figura 11: Amostras do drift em função do tempo e o fit exponecial relacionado a curva de 30 medidas esperando 2 minutos em 4 V.

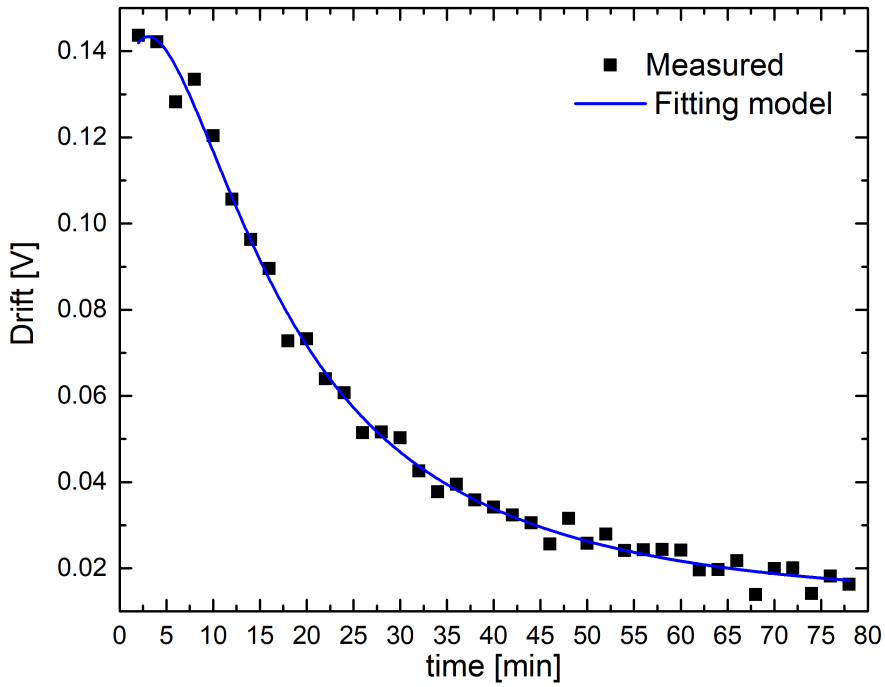

Figura 12: Amostras do drift em função do tempo e o fit exponecial relacionado a curva de 40 medidas esperando 2 minutos em 4 V.

Obsevando os pontos do drift dos gráficos, nota-se que eles induzem o comportamento criticamente amortecido, como esperado. Então, o drift começa crescendo até atingir ao máximo e depois decai até ficar constante. A parte do decaimento exponencial em diante é bem semelhante ao obtido com as medida de 20 minutos, mostrando a coerência entre as medidas.

A constante c_2 representa o quão expressivo é o decaimento exponencial, c_1 representa o crescimento no início e c_3 a constante que o sistema se estabiliza. As constantes de tempo τ_1 e τ_2 são se relacionam com o tempo para estabilização. ϕ_1 e ϕ_2 correspondem a um deslocamento temporal nas curvas para que o erro de ajuste seja reduzido. A seguir é apresentao uma tabela com as constantes resultantes dos fits e o coeficiente de determinação, R^2.

Número de medidas	20	30	40
c_1 [V]	0.0894	0.01048	0.00582
τ_1 [min]	5.12117	5.82524	5.88646
ϕ_1	1.3226	1.44834	1.00257
c_2 [V]	0.11407	0.29526	0.12823
τ_2 [min]	19.21912	14152.27492	22.50183
ϕ_2	-0.26597	0.85366	-0.10348
c_3 [V]	0.00957	-0.67801	0.01361
R^2	0.99275	0.96794	0.99371

Tabela 2: Constantes obtidas no fitting para as medidas esperando 2 minutos em 4 V e o coeficiente R^2.

Observando a tabela 2, é notável que os valores das contantes encontrados para o fit das medidas de 20 e 40 são próximos. Os valores que do fit de 30 medidas fogem do resultados das outras. A segunda contante de tempo (τ_2) é muito grande o que representa que o sistema demora muito para estabilizar. O drift em que essa medida se estabiliza é negativo, ou seja o drift começa a inverter de sentido, o que não corresponde aos outros resultados. O coeficiente de determinação para 30 medidas não é tão bom como das outras medidas, indicando que o fit apresenta mais erros que os outros.

As pequenas discrepâncias entre os resultados dos fits de 20 e 40 medidas se devem ao fato das medidas terem sido tiradas em dias diferentes. Ademais, antes da secção de 20 medidas outras medidas já haviam sido tiradas e isso afeta a função de transferência. Apesar disso, As curvas fitadas para 20 e 40 medidas apresentam comportamento muito parecido com esperado. Os coeficientes de determinação de ambas são satisfatórios e próximos da medida de 20 minutos.

3 Algoritmo Goertzel e sua implementação

a Algoritmo Goertzel

Como visto anteriormente, a amplitude da segunda harmônica pode ser utilizada como indicador da operação na região não linear do MZM. A fim de desenvolver um equipamento que realize o controle da tensão de polarização do modulador, é necessário um sistema que meça a magnitude do segundo harmônico do sinal.

Seguindo este pensamento foram estudados sistemas digitais para detecção de frequência. O que melhor se encaixa no problema em questão é o algoritmo Goertzel. Resumidamente, este algoritmo calcula individualmente a componente desejada da DFT (Discrete Fourier Transform) do sinal e, por isso, é a melhor opção encontrada.

Método	Multiplicadores reais	Somadores reais
FFT	$2Nlog_2N$	$Nlog_2N$
Goertzel	$N+2$	$2N+1$

Tabela 3: Comparação da necessidade de processamento do Goertzel e da FFT [4].

Em comparação com o método mais utilizado para analise espectral, a FFT (Fast Fourier Transform), o Goertzel necessita de menos processamento computacional para análise de um único tom. Mais especificamente, caso o Goertzel seja impleentado M vezes para M diferentes frequências, ele é mias eficiente que a FFT se $M < log_2N$, onde N é o número de amostras [4] [5].

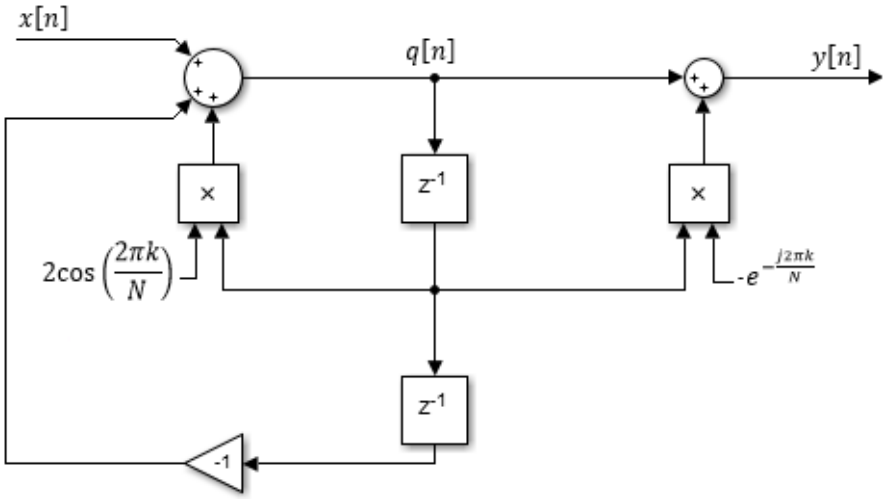

Figura 13: Diagram de blocos do Filtro Goetzel.

Na figura 13 é apresentado o sistema do algoritmo . É importante ressaltar que N é o número de amostras e k é dado por:

$$k = N\frac{f_t}{f_s} \tag{8}$$

Sendo f_t a frequência alvo e f_s a frequência de amostragem. Afim de comprovar que a saída após N termos corresponde a transformada discreta de Fourier do sinal de entrada, são escritos os sinais q[n] e y[n]:

$$q[n] = 2cos\left(\frac{2\pi k}{N}\right)q[n-1] - q[n-2] + x[n] \tag{9}$$

$$y[n] = q[n] - e^{-j\frac{2\pi k}{N}} q[n-1] \tag{10}$$

Utilizando a transformada Z e suas propriedades:

$$\frac{Q(z)}{X(z)} = \frac{1}{z^{-2} - 2cos\left(\frac{2\pi k}{N}\right) z^{-1} + 1} \tag{11}$$

$$\frac{Y(z)}{Q(z)} = 1 - e^{-j\frac{2\pi k}{N}} z^{-1} \tag{12}$$

A partir destas expressões é possível obter a função de transferência (H(z)) do sinal e utilizando a formula de Euler simplifica-la:

$$H(z) = \frac{Y(z)}{Q(z)}\frac{Q(z)}{X(z)}\frac{Y(z)}{X(z)} = \frac{1 - e^{-j\frac{2\pi k}{N}} z^{-1}}{z^{-2} - 2cos\left(\frac{2\pi k}{N}\right) z^{-1} + 1} = \frac{1}{1 - e^{j\frac{2\pi k}{N}} z^{-1}} \tag{13}$$

Com isso, realizando a transformada Z inversa é obtido a resposta impulsional do sistema (sendo $u_{-1}[n]$ o degrau unitário em tempo discreto):

$$h[n] = e^{j\frac{2\pi k}{N} n} u_{-1}[n] \tag{14}$$

Portanto, o sinal de saída (y[n]) do sistema pode ser reescrito como a convolução entre o sinal de entrada (x[n]) e a resposta impulsional.

$$y[n] = h[n] * x[n] = \sum_{r=-\infty}^{N-1} x[r] e^{j\frac{2\pi k}{N}(n-r)} u_{-1}[n-r] \tag{15}$$

$$y[n] = \sum_{r=0}^{N-1} x[r] e^{j\frac{2\pi k}{N}(n-r)} \tag{16}$$

Analisando a saída para o último valor de amostra.

$$y[N] = \sum_{r=0}^{N-1} x[r] e^{j\frac{2\pi k}{N}(N-r)} = \sum_{r=0}^{N-1} x[r] e^{-j\frac{2\pi k}{N} r} \tag{17}$$

Agora, observando a transformada de Fourier discreta do sinal de entrada avaliada na frequência alvo (f_t):

$$X(f_t) = \sum_{r=0}^{N-1} x[r] e^{-j\frac{2\pi f_t}{f_s} r} = \sum_{r=0}^{N-1} x[r] e^{-j\frac{2\pi k}{N} r} \tag{18}$$

E assim é provado que a N-ésima saída representa a componente desejada da DFT do sinal de entrada:

$$X(f_t) = y[N] \tag{19}$$

Para simplificar ainda mais, nesta aplicação basta a informação da magnitude do sinal, portanto apenas o módulo quadrado do mesmo é necessário:

$$|X(f_t)|^2 = |q[N-1]|^2 + |q[N-2]|^2 - 2cos\left(\frac{2\pi k}{N}\right) q[N-1]q[N-2] \quad (20)$$

Apesar do Goertzel não ser utilizado como um filtro, pois a saída é apenas o N-ésima termo, pode-se analisar sua resposta para diferentes frequências. Seu comportamento é como a de uma função $\frac{sen(x)}{x}$. Os zeros da função de transferência ocorrem em k acrescidos de números inteiros, ou seja, a cada $\frac{f_s}{N}$ a mais ou menos da frequência alvo. Portanto, é facil notar quanto maior N mais estreito é a banda e mais seletivo o filtro será.

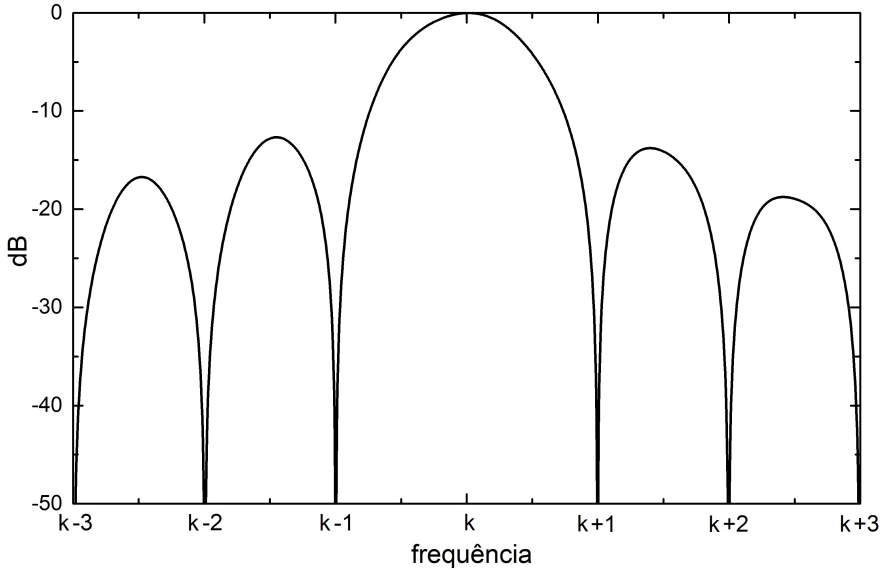

Figura 14: Resposta na frequência do filtro Goertzel.

b FPGA

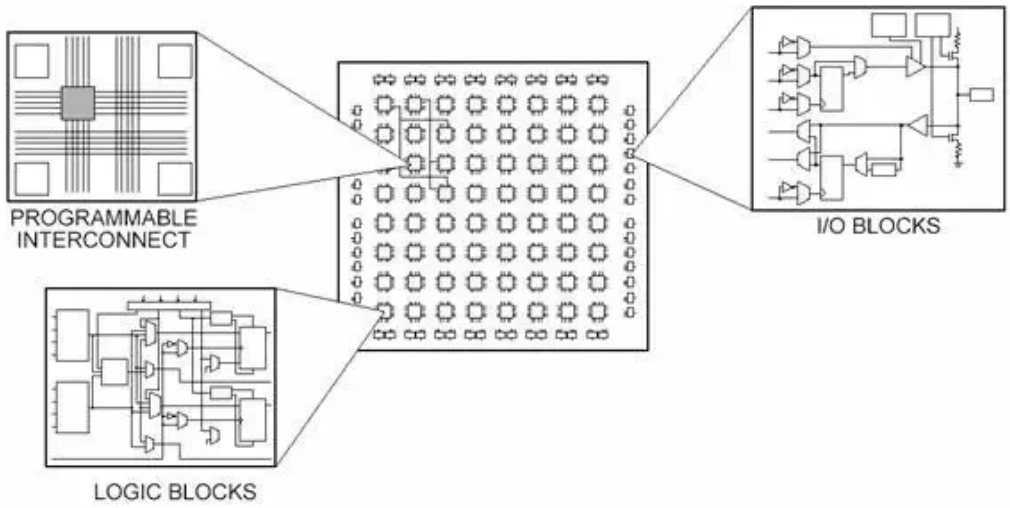

Figura 15: Representação interna de uma FPGA.

Para implementação do Goertzel foi escolhido uma FPGA em razão de sua velocidade de processamento. Outro fator para esta escolha, é sua capacidade de paralelização de tarefas, sendo assim,

possível realizar multiplas operações ao mesmo tempo. Mais que isso, possível implementar dois Goetzels ao mesmo tempo sem alterar o tempo de processamento.

A FPGA (Field Programmable Gate Array) consiste em um dispositivo semicondutor composto por blocos lógicos Configuráveis, blocos de entrada e saída e chaves de interconexões (figura 15). Descrevendo a maneira como estes blocos serão utilizados e como devem ser interligados às entradas e às saídas o usuário é capaz de implementar qualquer circuito digital concebível [6].

Os blocos lógicos configuráveis (CLB) são o coração das FPGAs. Basicamente, eles que compõe os circuitos digitais desenvolvidos. Os CLBs mais simples são compostos por LUTs (Lookup Table) de 4 bits, flipflops e multiplexadores, como mostra a figura 16. As LUTs são elementos lógicos que armazenam uma tabela verdade de entrada e saída e flipflops são elementos de memória.

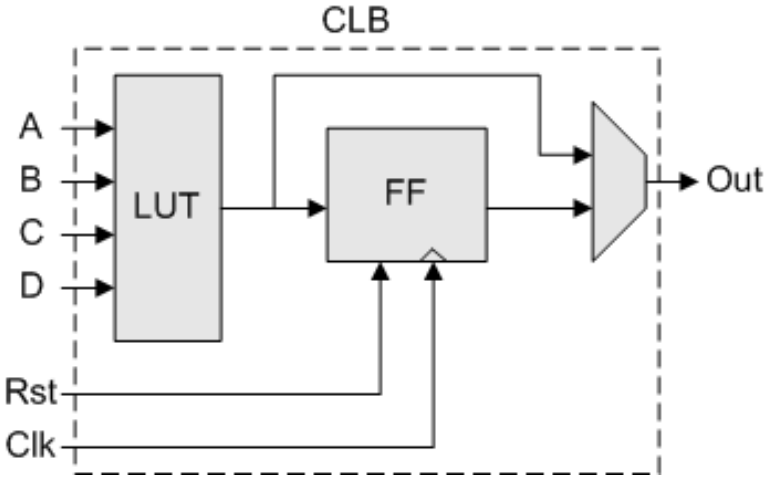

Figura 16: Representação do circuito lógico de uma CLB básica. Nesta imagem observa-se um LUT de 4 bits com saída ligada a um FF (flipflop) e um multiplexador.

O funcionamento do CLB consiste na lógica armazenda na LUT. A saída deste elemento vai tanto para o flipflop quanto para o multiplexador. O flipflop armazena (atrasa) a saída e o multiplexador escolhe entre a saída atrada e atual. Desta forma com mais de mil destes e mais elementos, a FPGA é capaz de realizar os circuitos lógicos desevolvidos pelos usuários.

A implementação do código se dá a partir de softwares capazes de gerar arquivos contendo a informação de como os blocos da FPGA devem ser interligados. A linguagem mais comum utilizada para descrever o comportamento do circuito nas FPGA's é o VHDL (VHSIC Hardware Description Language). Ela difere bastante das linguagens de programação comuns justamente por se tratar de design de hardware.

Apesar da dificuldade, esta linguagem permite uma programação modular do seu circuito, em que é possível separar seu projeto em pequenos blocos simples e juntá-los aos poucos até que a estrutura seja completada. Além disso, é possível simular individualmente o funcionamento de cada bloco e observar se condiz com o esperado.

Após a simulação e a verificação da mesma, é realizado a implementação do design na placa. Para tal, o software realiza o processo de síntese que gera uma netlist, que descreve as conexões lógicas do design. Em seguida, essa netlist ajustada para o despositivo que será utilizado (há diversos modelos de FPGAs: Spartan da Xilinx, Cyclone da Altera etc). Concluindo o processo, o arquivo de netlist ajustado é enviado para FPGA [6].

c Implementação do Goertzel na FPGA

Para compreender a implementação do algorítmo é necessário alguns conhecimentos prévios sobre a representação de um número real em ponto flutuante (float). IEEE 754 (IEEE Standard for

Floating-Point Arithmetic) é a padronização da representação em ponto flutuante. A representação de precisão sigular, utilizada no projeto, ocupa 32 bits. Esses bits são organizados em: 1 bit de sinal (s), 8 bits de expoente (E) e 23 bits de mantissa (M), respectivamente.

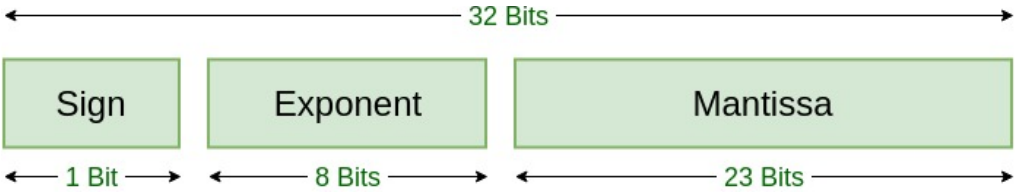

Figura 17: Reresentação de ponto flutuante segundo o padrão IEEE 754.

O bit de sinal é o mais simples, 0 representa um número positivo e 1, negativo. Agora, Tomando o número a ser representado em base binária, desloca-se o ponto até o ultimo 1. A matissa corresponderá a este número atrás da vírgula, se necessário, acrescido de zero até completar os 23 bits ou truncado para que caiba. Por fim, o exponte é o número de vezes que a vírgula precisou ser deslocada acrescido de 127, caso deslocamente seja para esquerda. Senão o exponte será 127 subtraido por este deslocamento. Para facilitar a compreensão, segue um exemplo para obter a representação de -12.75:

$$N = -12.75 = -\frac{51}{4} = -51 \cdot 2^{-2} \qquad S = 1$$

$$N_{bin} = 1100.11 \qquad M = 10011\ \ 00000000000000000$$

$$E = (127 + 3)_{bin} = 10000010 \qquad N_{float} = 11000001010011000000000000000000$$

No exemplo, primeiro é escrito o número de forma que seja um inteiro divido por um multipolo de 2. Em seguida, encontra-se a represntação binária do inteiro e anda com a vírgula a potência de dois. Com isso, encontra-se a mantissa flutuando o ponto até antes do ultimo 1. Para o exponte basta somar a quantidade de vez que foi shiftado com 127. Neste exemplo foi utilizado um número com representação exata em binário, entretanto se não fosse seria necessário encontrar o número que mais se aproxima para ser a mantissa.

Agora sabendo um pouco acerca da representação de ponto flutuante será discutido sobre a implementação do algoritmo. A aplicação foi desenvolvida a partir de uma máquina de estados com 7 estados. Em cada estado é realizado uma ou mais operações, e a passagem de estado só ocorre ao fim de todas operações do mesmo. Como o sinal será amostrado por um DAC (Conversor analógico-digital), as entradas serão números inteiros, porém para multiplicar pelo cosseno (vide equação 20) é necessário converter para ponto flutuante. Além disso, o coeficiente $2cos\left(\frac{2\pi k}{N}\right)$ é uma constante pré calculada para o algoritmo.

A conversão é realizada no primeiro estado (estado 0). A fim de realizar a operação em um ciclo do clock, é criado um vetor com todos shifts possíveis para gerar a mantissa. Sabendo o indice desse vetor é obtido expoente no mesmo instante com um mulltiplexador. Apesar do gasto excessivo de memória para guardar todos shifts possíveis, a aplicação atende aos requisitos de duração. Ademais, não é necessário testar o fim da operação, pois smepre no ciclo seguinte a resposta já está pronta.

Nos dois estados seguintes (estados 1 e 2) são realizadas as operações para calcular o novo valor de q (vide equação 9). No estado 1 é implementado a multiplicação do sinal q atrasado uma unidade com o coeficiente $(2cos\left(\frac{2\pi k}{N}\right))$ e a subtração da entrada com o sinal q atrasado duas unidades. No estado 2, os resultados do estado anterior são somados resultando no sinal q atual. Ao fim deste estado, o valor do sinal q[n-1] é salvo na posição do q[n-2] e o q atual é salvo na posição do q[n-1].

Nos de mais estados (3,4,5 e 6) são implementadas as operações para o cálculo da magnitud (vide equação 20). No estado 3, são realizadas as multiplicações para obter o módulo quadrado dos sinais

de q atrasado. No estado 4, é realizado a soma dos resultados do estado anterior e a multiplicação de q[n-1] com q[n-2]. No estado 5 multiplica-se o coeficiente ($2cos\left(\frac{2\pi k}{N}\right)$) com o resultado da multiplicação anterior. Por fim, no estado 6, é somado o resultado soma do estado 4 com o resultado do estado 5, obtendo a magnitude do sinal de saída.

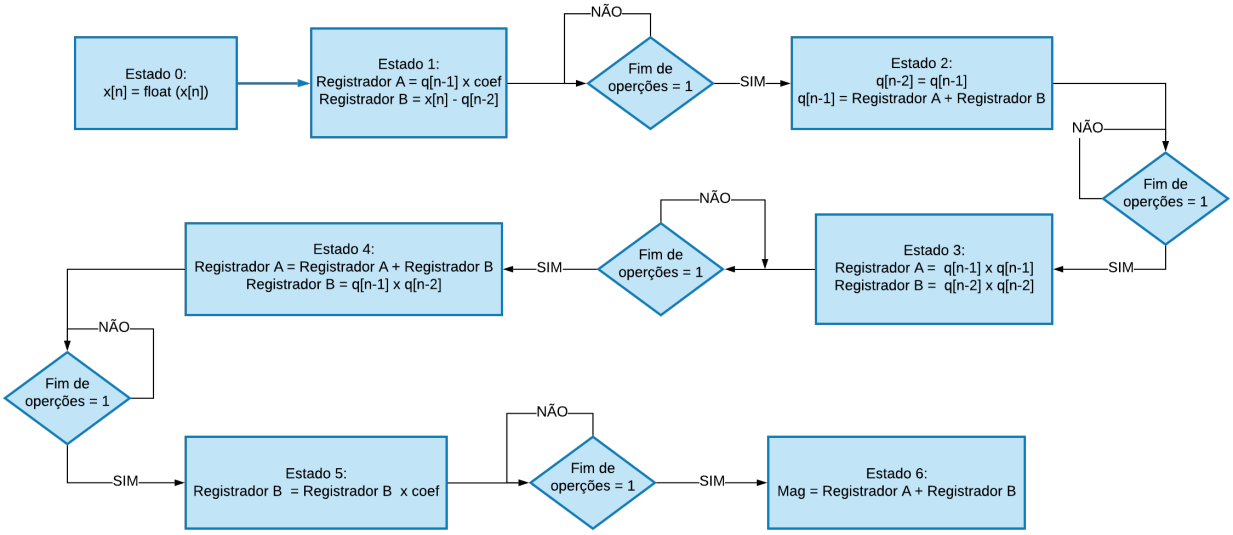

Figura 18: Fluxograma simplificado do algoritmo que ilustra os estados e as operações realizadas neles.

O sistema tem um contador que é acrescido a cada iteração que ocorrere. Após realizar N iterações, sendo N um número prédefinido, o sistema zera todos elementos de memória e a saída. Além disso um sinal done de saída indica que foram realizadas as N iterações.

Além da amostra, do clock e da magnitude resultante, o sistema tem outros sinais de entrada e saída para facilitar a comunicação com outros elementos digitais. Há mais dois sinais de entrada: um enable que habilita a passagem do estado 0 para estado 1, ou seja habilita a utilizar a amostra e um reset que zera todos elementos de memória (registradores) e faz com que o estado retorne ao zero.

Ademais, há outro sinal de saída: um available que representa se o sistema pode receber outro dado. Ao todo foram utilizados um conversor de inteiro para float, um somador e dois multiplicadores e nessa configuração o sistema demora 33 ciclos do relógio para realizar uma iteração. Para facilitar o entendimento, o código é apresentado no apêndice A.

d Simulações do algoritmo

A fim de observar o funcionamento do algoritmo Goertzel implementado na FPGA, foi desenvolvido uma rotina de simulação. Esta é realizada no Test bench em conjunto com o MATLAB. Primeiramente, é gerado no MATLAB, um sinal composto 80% por um cosseno de frequência 1 kHz e 20% por um cosseno de frequência 2 kHz. A taxa de amostragem do sinal é de 200 kHz (200 kSamples/s). Um arquivo contendo os pontos dessa entrada é lido pelo Test bench e MATLAB. Ambos processam duas vezes o algoritmo Goertzel e apresentam a magnitude a cada.

o algoritmo foi implementado duas vezes, um para a frequência de 1 kHz e outra para frequência de 2 kHz. As figuras 19 e 19 apresentam os resultados para 1000 iterações. Em seguida, para comprovar se ambos estão corretos, foi feita a razão entre a saída de 2 kHz com a de 1 kHz (figura 21). Espera-se que a saída se estabilize em 0.0625, pois é razão quadrada entre as magnitudes:

$$\left|\frac{X(2000)}{X(1000)}\right|^2 = \left(\frac{0.2}{0.8}\right)^2 = 0.0625 \tag{21}$$

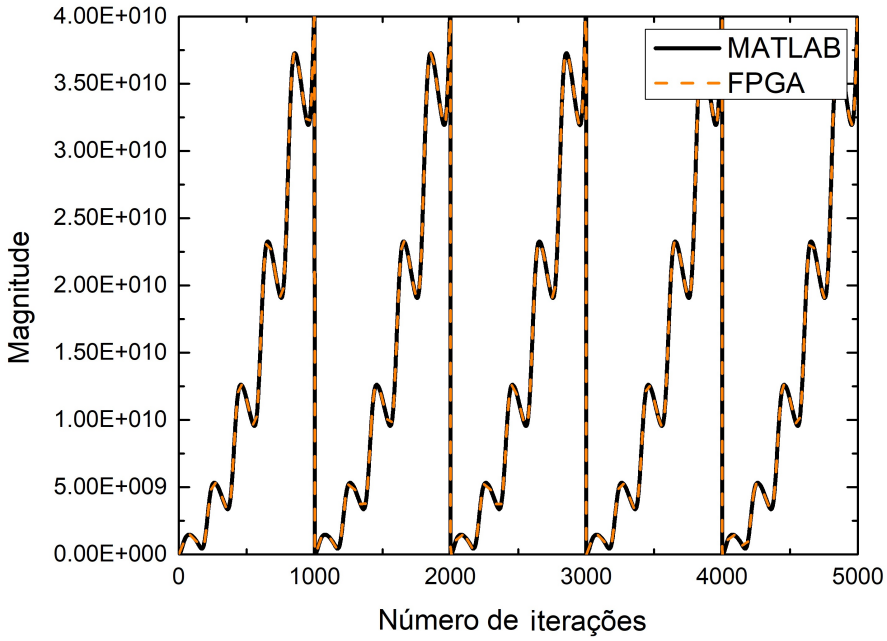

Figura 19: Utilizando 1000 iterações e frequência alvo de 1 kHz.

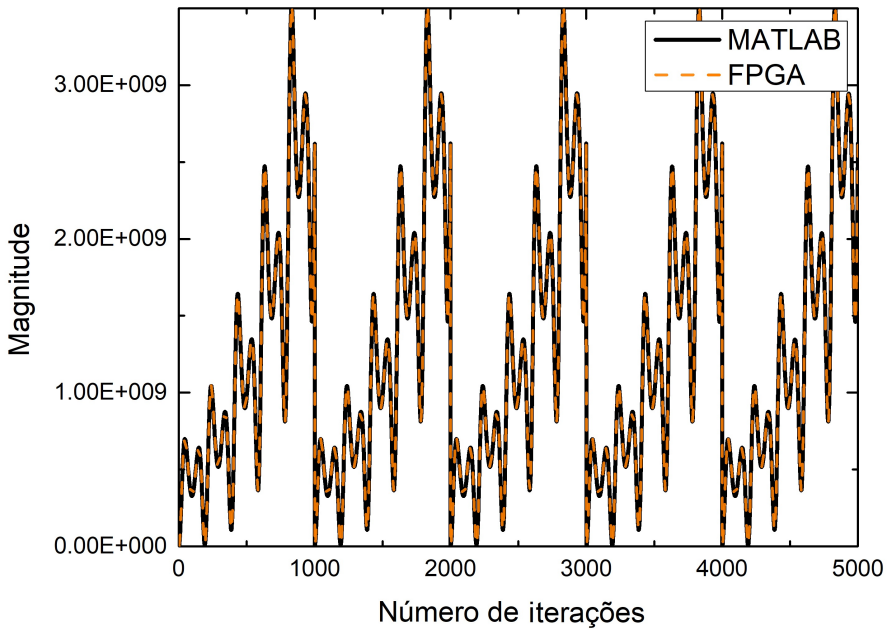

Figura 20: Utilizando 1000 iterações e frequência alvo de 2 kHz.

A partir dos primieros resultados nota-se que a implementação do Goertzel na FPGA ocorreu igual ao do MATLAB. Os valores do sinal são muito altos, por isso é difícil observar que há um ligeira diferença entres os resultados (quarta casa decimal). Entretanto, já era esperado, pois o MATLAB trabalha com dupla precisão, reduzindo os erros associados à aproximação.

Apesar do resultado obtido com a fFPGA condizer com esperado do MATLAB isso ainda não prova que os dois estão certos.Para tal é necessário observar a razão entre as magnitudes e comprovar se elas correspondem ao esperado (21). Portanto, com as medidas tiradas, foi feito o gráfico da figura 21.

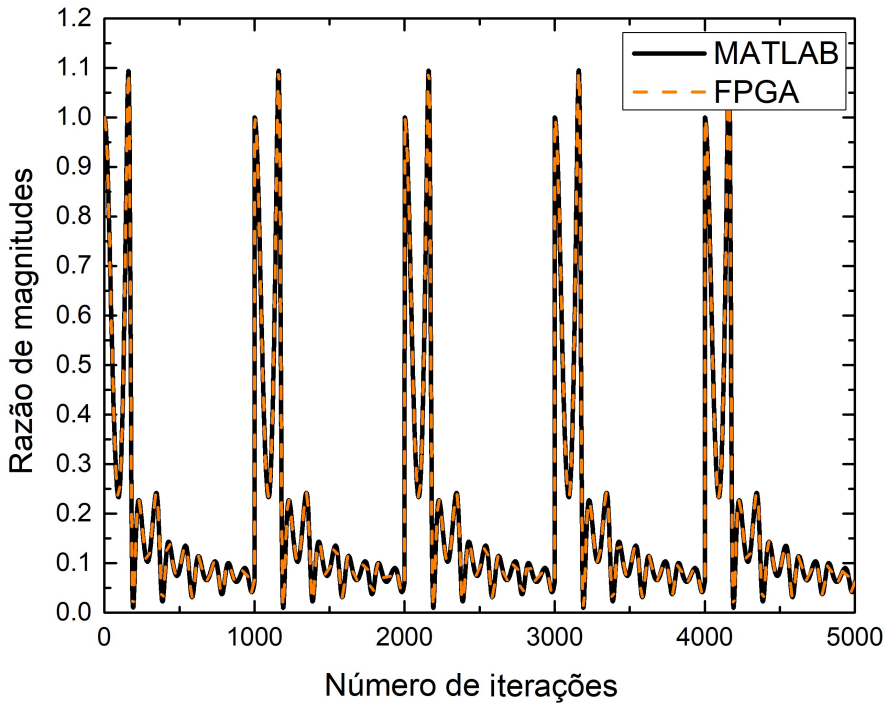

Figura 21: Razão entre os resultados dos Goertzels de frequência alvo 1 kHz e 2kHz utilizando 1000 iterações.

Com este resultado da figura 21, observa-se que no início há muita oscilação e o erro e muito alto, porém conforme número de iterações aumenta, o valor vai diminuindo e tendendo ao valor esperado. A razão entre as medidas na ultima iteração da FPGA foi de 0.062419 e do MATLAB de 0.062418. O resultado obtido é promissor, porém é necessário observar mais iterações.

A fim de notar se de fato o algoritmo Goertzel tende ao valor esperado foi dobrado o número de interções e realizado as mesmas medidas para mesma entrada. Os resultados obtidos para o Goertzel de 1 kHz de frequência alvo, o Goertzel de 2 kHz de frequência alvo e a razão entre elas são apresentados nas figuras 22, 23 e 22, respectivamente.

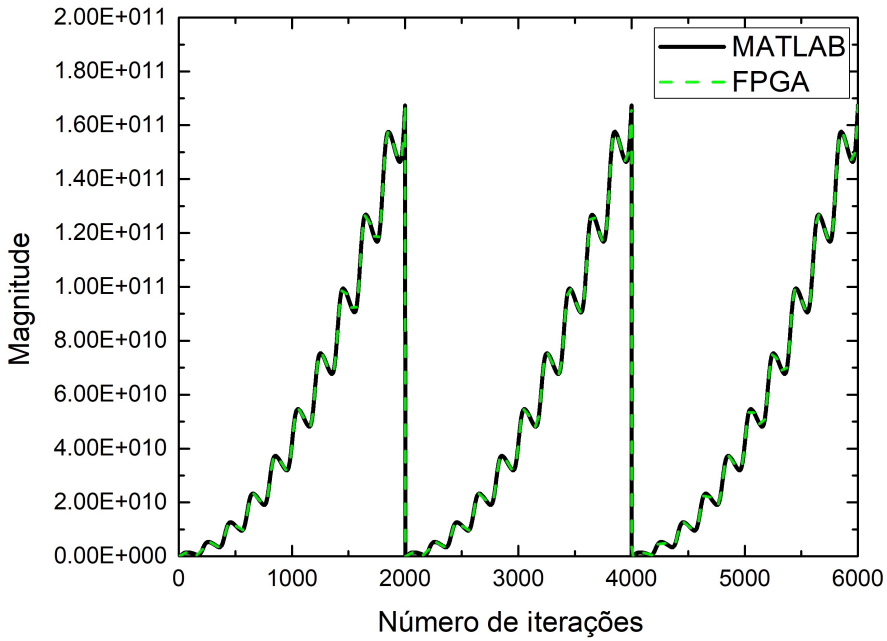

Figura 22: Utilizando 2000 iterações e frequência alvo de 1 kHz.

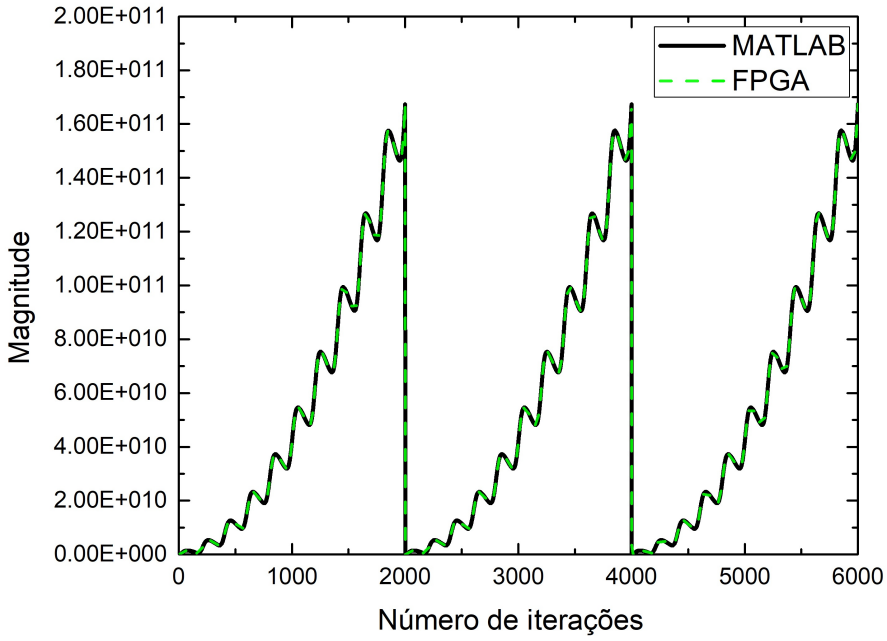

Figura 23: Utilizando 2000 iterações e frequência alvo de 2 kHz.

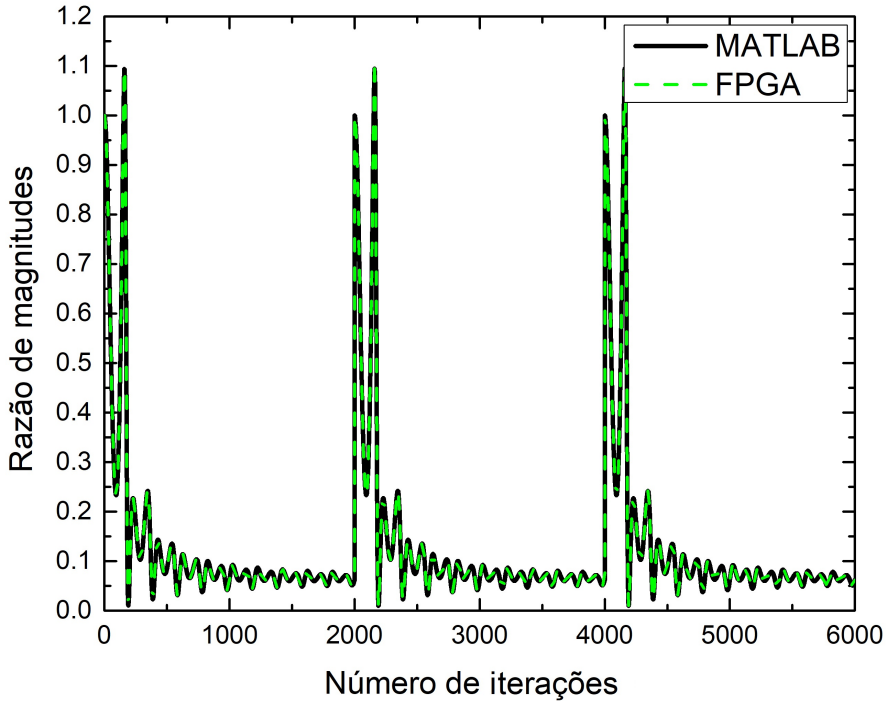

Figura 24: Razão entre os resultados dos Goertzels de frequência alvo 1 kHz e 2kHz utilizando 2000 iterações.

Observando os resultados para 2000 iterações observa-se que os resultados do MATLAB e da FPGA continuam parecidos como esperado. E a figura 24 comprova o que fora dito: conforme aumenta o número de iterações as oscilações diminuem e o valor vai estabilizando no vlaor desejado. Por tanto, comprova-se o funcionamento do algoritmo implementado na FPGA.

Apesar da indicação de que qunato maior o número de iterações melhor, não é necessário um sistema com N muito grande para compara as frequências. Foi observado que 1000 iterações já suficente para medidr a razão entre as magnitudes das frequências. Outra possibilidade é utilizar o

Goertzel em tempo real, sem resetar, porém é problemático pois o algoritmo é instável e os resultados podem dar overflow na representação de ponto flutuante.

Com os bons resultados do algoritmo implementatodo partiu-se para uma simulação com uma entrada mais próxima do real, com as imperfeições do DAC, fotodetector e modulador. Utilizado um gerador de sinal arbitrário (AWG) foi transmitido um tom de frequência de 1 kHz na entrada elétrica de um modulador de Mach-Zehnder. A entrada óptica do MZM é um laser em 1550 nm que passa por um contorolador de polarização (PC) e a saída vai para um fotodetector com ganho controlável. O controle de polarização necessário para maximizar a potência de saída do MZM.

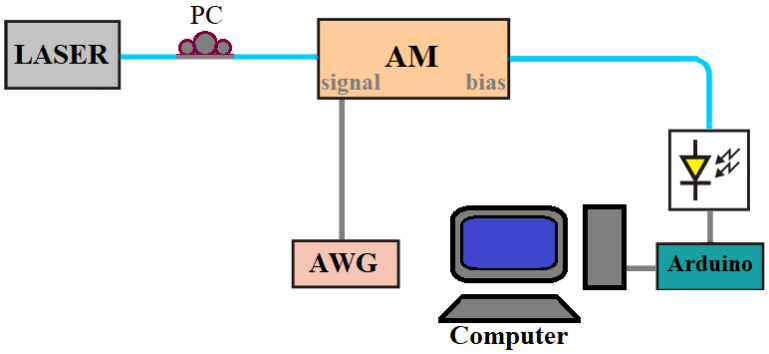

Figura 25: Sistema para medir a saída do AM.

A saída do fotodetector é ligada a um Arduino Due que amostrou o sinal com 200 kSamples/s e salvou os valores durante um segundo no computador, em formato CSV. As figuras 26 e 30 representam um pedaço do sinal que foi amostrado em duas situações: na região linear do MZM e fora da região linear do MZM. Em seguida, foi aplicado está entrada nos algoritmos da FPGA e do MATLAB.

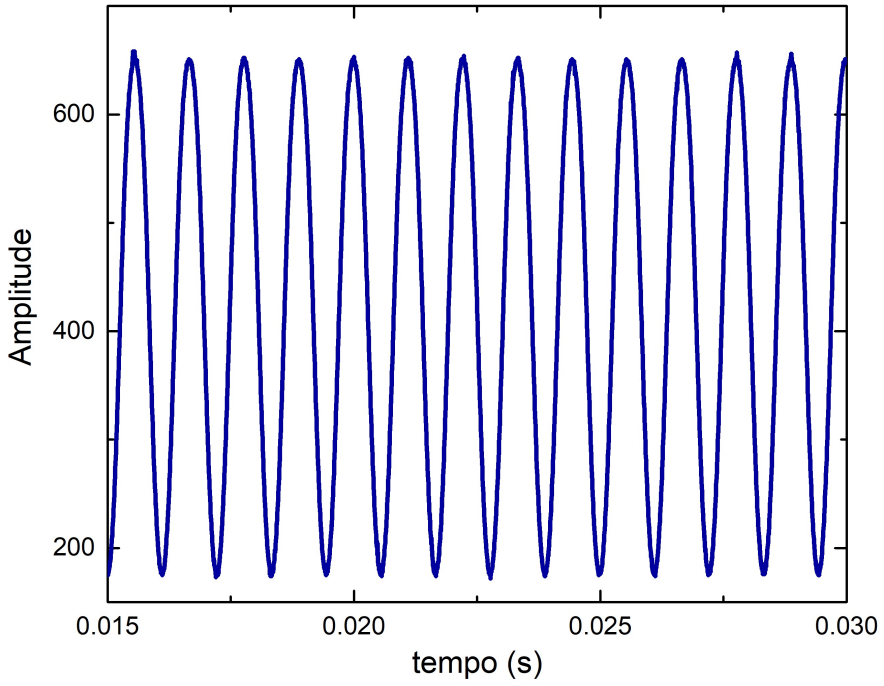

Figura 26: Sinal de 900 Hz amostrado pelo Arduino Due com o MZM na região linear.

A fins comparativos foi aplicado a FFT, implementada no MATLAB, ao sinal amostrado para ter o valor de razão entre a segunda harmônica e a fundamental. Observou-se que a frequência do tom na verdade está deslocada 100 Hz do que era esperado, ou seja o tom fudamental está em 900 Hz,

consequentemente a segunda harmônica está em 1.8 kHz. Este erro no valor provavelmnete se deve ao gerador arbitrário de sinal. Por isso, foram implementados Goertzels nestas frequências de 900 Hz e de 1.8 kHz.

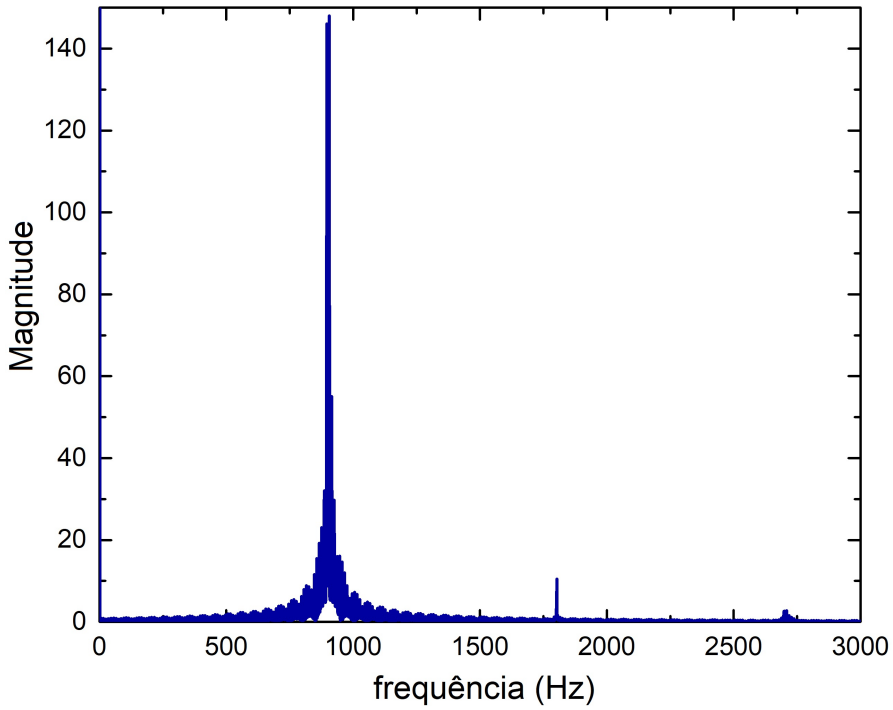

Figura 27: FFT do sinal amostrado na região linear do MZM.

Observando a FFT do sinal, nota-se a presença das harmonicas, porém muito inferiores comparados a fundamental. A razão quadrática entre a segunda harônica e a fundamental obtida neste caso é de 0.00504 ou -22.98 dB.

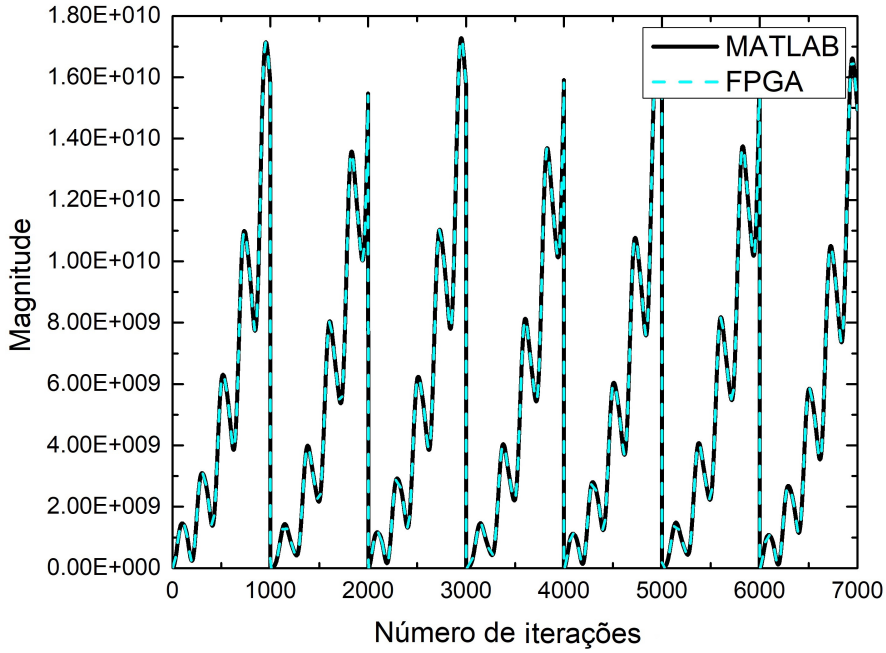

Figura 28: Resposta do Goertzel de frequência alvo 900 Hz para o sinal amostrado na região linear do MZM.

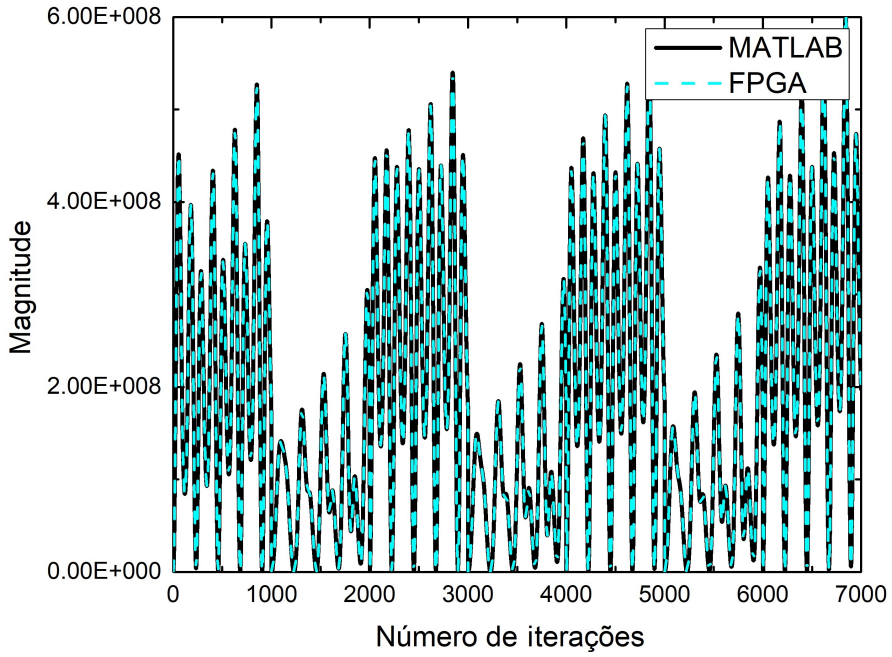

Figura 29: Resposta do Goertzel de frequência alvo 1.8 kHz para o sinal amostrado na região linear do MZM.

Com os resultados do Goertzel tirados, nota-se que mesmo com as imperfeições no sinal, as duas implementações ainda retornam resultados similares. Entretanto a magnitude final (após as 1000 iterações) fica oscilando. Portanto, para comparar a razão encontrada com da FFT foi realizada a média das razões obtidas: 0.0101 ou -19.97 dB. O valor encontrado é maior que o da FFT, pois o Goertzel tem uma banda, ou seja as outras frequências influenciam no resultado, como visto na figura 14.

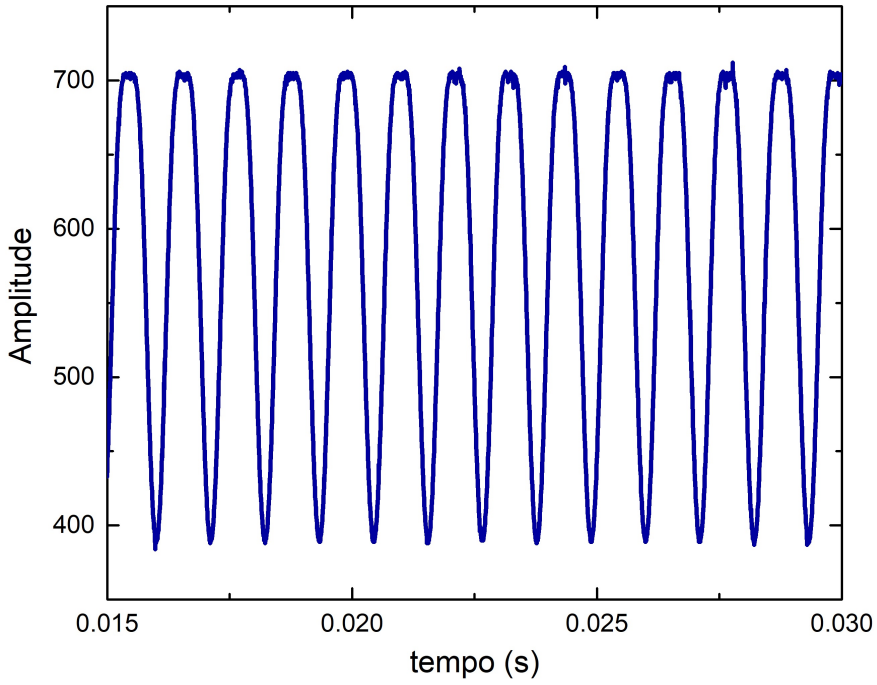

Figura 30: Sinal de 900 Hz amostrado pelo Arduino Due com o MZM fora da região linear.

Agora para a operação fora da região linear do MZM. Assim como realizado para o sinal na região

linear, foi implementado a FFT para compara os resultados obtidos. Novamente, os Goertzels foram para as frequências de 900 Hz e de 1.8 kHz devido ao deslocamento da frequência do gerador de sinal.

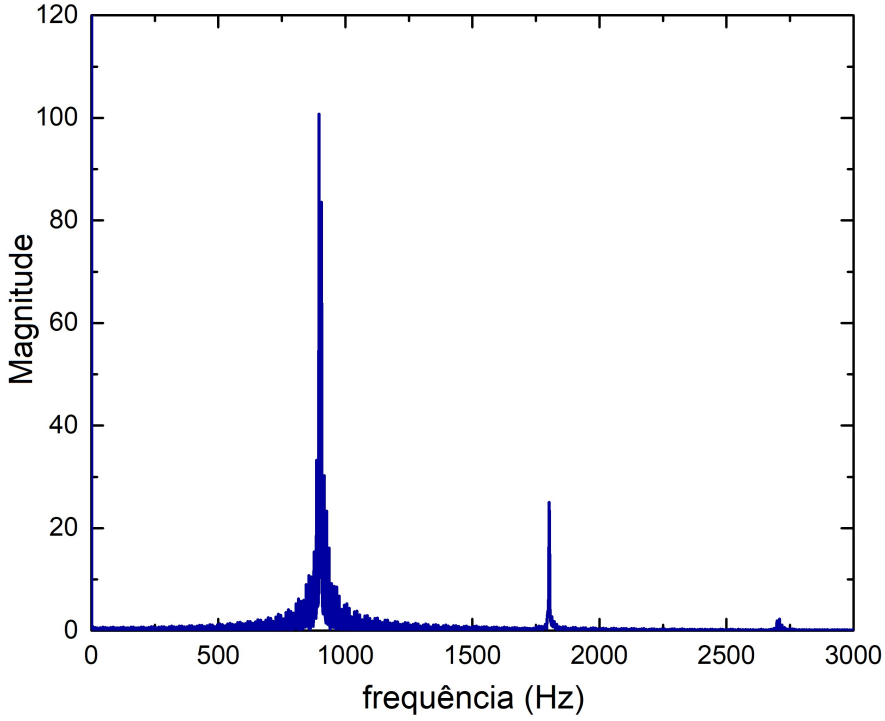

Figura 31: FFT do sinal amostrado fora da região linear do MZM.

Observando a FFT do sinal, nota-se a presença das harmonicas mais intenças que no caso anterior, principalmente a segunda harmônica. Observadno os efeitos citados em [2]. A razão quadrática entre a segunda harônica e a fundamental obtida neste caso é de 0.0616 ou -12.11dB.

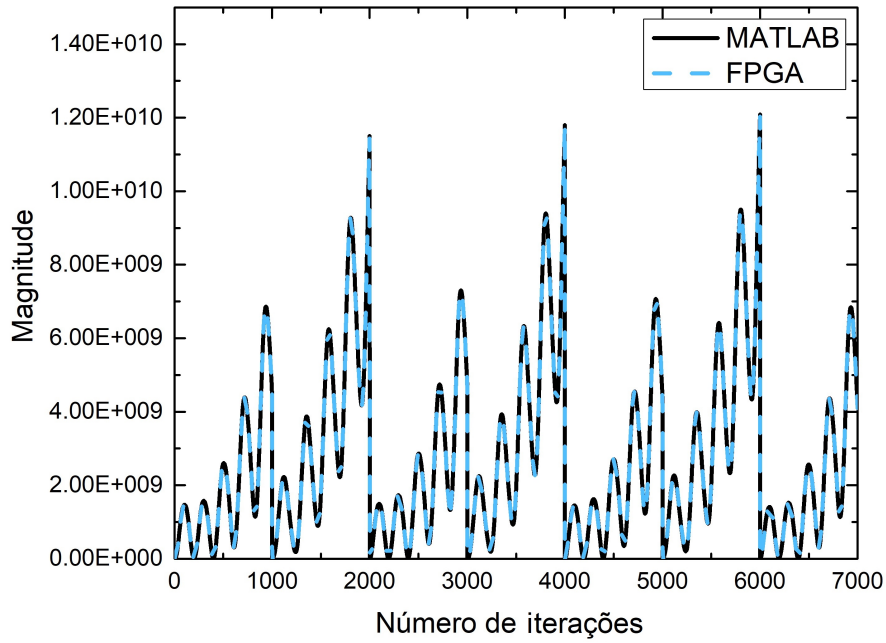

Figura 32: Resposta do Goertzel de frequência alvo 900 Hz para o sinal amostrado fora da região linear do MZM.

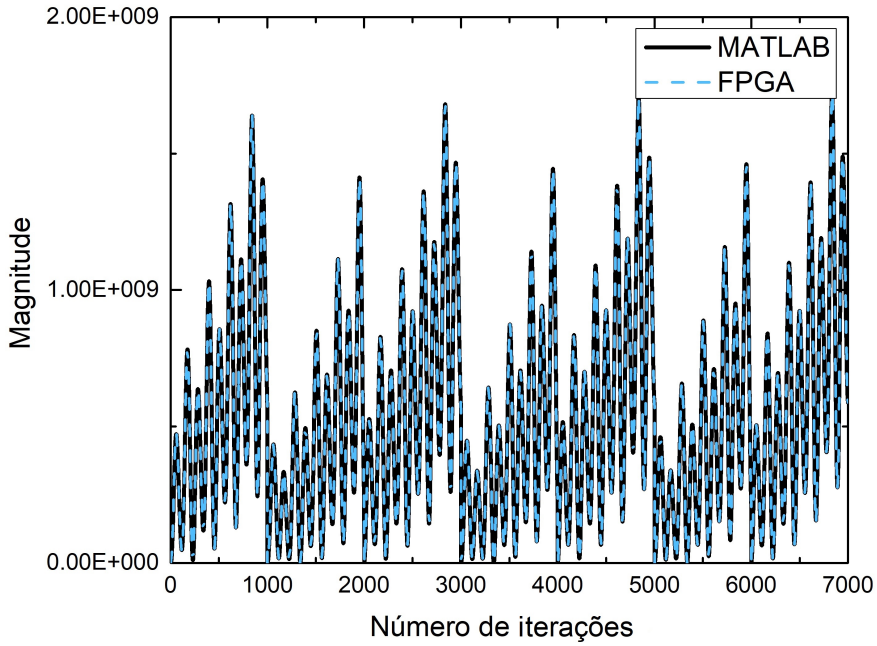

Figura 33: Resposta do Goertzel de frequência alvo 1.8 kHz para o sinal amostrado fora da região linear do MZM.

A implementações da FPGA e do MATLAB retornam resultados similares, como esperado. A magnitude final (após as 1000 iterações) oscila ainda mais que no caso anterior. Portanto, para comparar a razão encontrada com da FFT foi realizada a média das razões obtidas: 0.0101 ou -19.97 dB. O valor encontrado é maior que o da FFT, pois o Goertzel tem uma banda, ou seja as outras frequências influenciam no resultado, como visto na figura 14.

4 Sistema de controle

a DAC e ADC

Haja vista que a FPGA é um dispositivo digital e que o sinal de saída do fotodetector a entrada de bias do MZM são analógicos, é de suma importância o uso de conversores analógico-digital e digital-analógico no projeto.

O ADC (Anolog-to-Digital Converter) são equipamentos eletrônicos capazes de amostrar sinais analógicos (contínuos) a uma dada frequência (taxa de amostragem) e gerar uma representação digital (discreta) destes. A representação do ADC dependerá do número de bits (n) e da faixa de tensão ou corrente que pode receber, estes fatores estão associados a resolução como será apresentado.

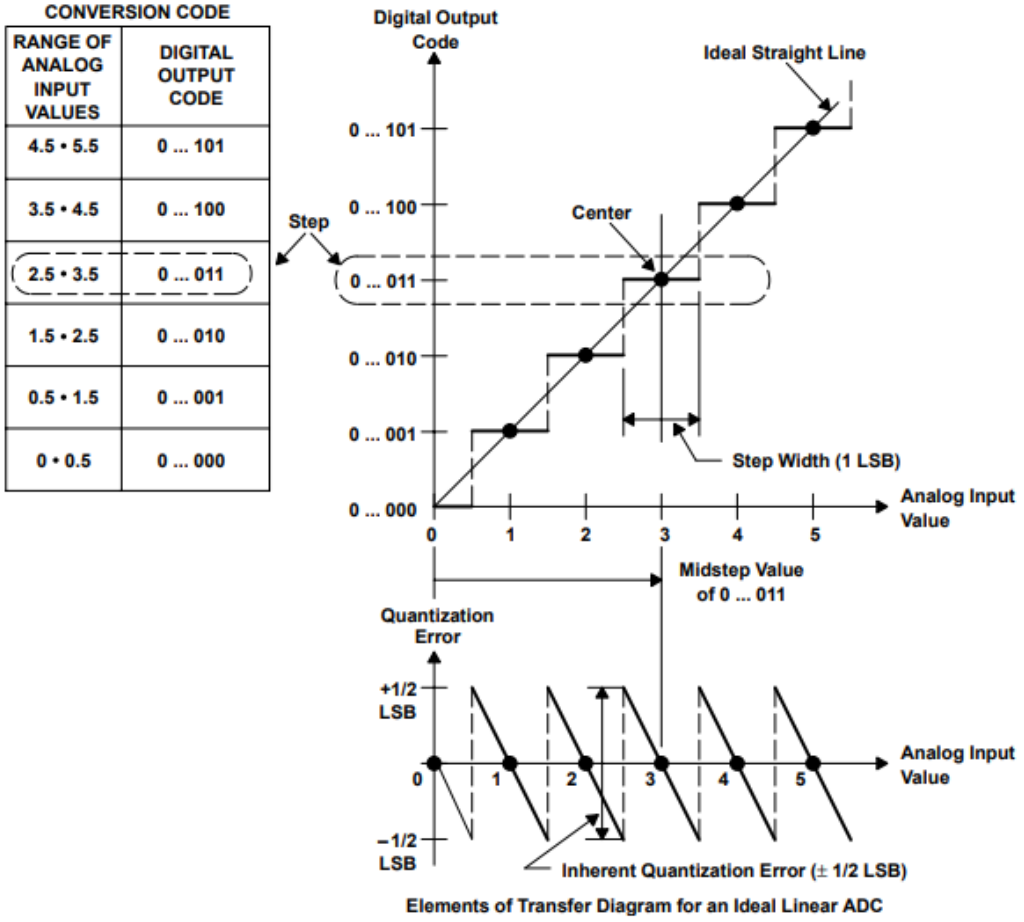

Figura 34: Função de transferência de um ADC ideal [7].

A figura 34 representa a função de transferência de um ADC ideal. A entrada analógica é quantizada em passos de um bit menos significativo (LSB, Least Significant Bit), esta é resolução do ADC. Um LSB corresponde a faixa de valores anológicos (FSR, Full-Scale Range) sobre o número de representações possíveis (2^n, em que n é o númeor de bits) menos um (vide equação 22). Este menos um da expressão, é devido o fato de que o primeiro e o ultimo valor da FSR correspondem a meio LSB. Portanto, o erro associado a quantização do ADC é $1/2$ LSB.

$$LSB = \frac{FSR}{2^n - 1} \tag{22}$$

O DAC (Anolog-to-Digital Converter) são equipamentos eletrônicos que convertem um sinal digital

com um determinado número de bits para um sinal analógico de tensão ou de corrente. A resolução do DAC é igual a do ADC, um LSB (vide equação 22). A cada bit a mais é acrescido um LSB na saída. A imagem 35 apresenta como ocorre a conversão ideal de digital para analógico. Outro fator importante é a taxa de atualização que a frequência com que o DAC altera a saída em relação a entrada.

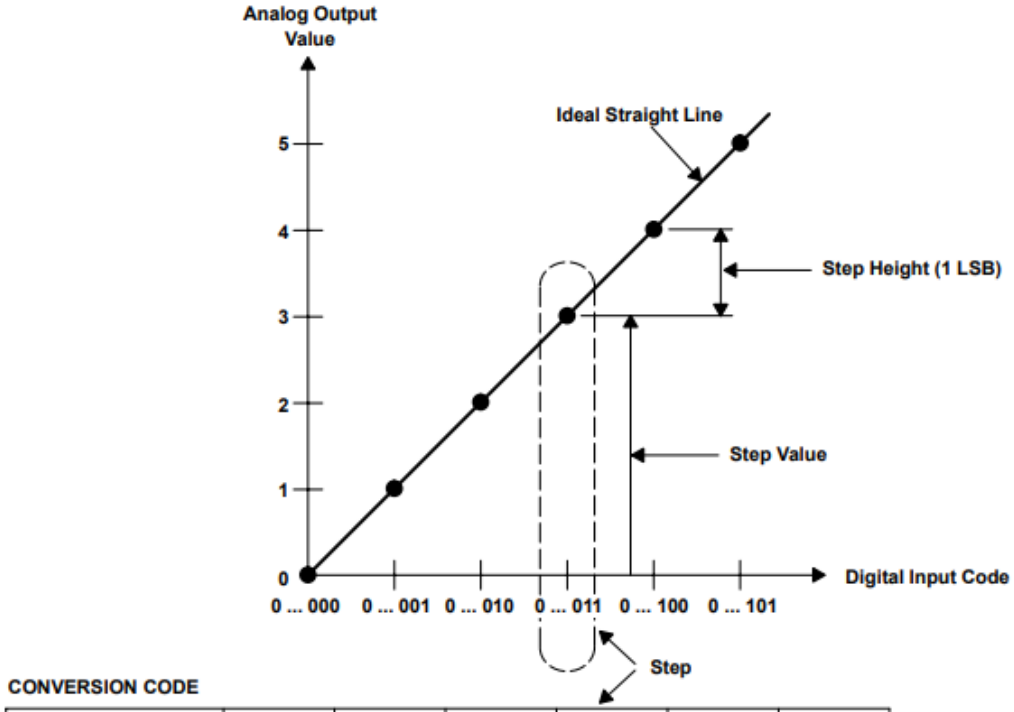

Figura 35: Função de transferência de um DAC ideal [7].

A taxa de amostragem do ADC é o limitante para frequência do sinal analógico de entrada. Pelo Teorema da amostragem de Nyquist–Shannon, a frequência de amostragem do sinal deve ser no mínimo maior que o dobro da frequência máxima do sinal a ser amostrado (frequência de Nyquist) [5]. Analogamente, a taxa de atualização do DAC deve seguir a mesma regra.

$$f_a \geq 2f_{max} \tag{23}$$

No projeto foram utilizados os convrsores AD e DA do Arduino Due para amostrar a saída do fotodetector e converter a resposta do sitema a um valor de tensão de bias do MZM. O DAC do Due tem 12 bits de entrada, equivalente a 4096 níveis. A saída pode variar de 0.55 a 2.75 V. Com isso, o LSB é de 0.5 mV. A taxa de atualização não é apresentado, entretanto segundo o datasheet do microcontrolador Atmel SAM3X8E [8], a saía do DAC esta apta a ser lida após 50 ciclos do relógio. Como o relógio do Due é de 84 MHz, a taxa de atualização é de 1.68 MHz [9]. Esta taxa de atualização atende ao projeto dado que o drift é lento.

O ADC interno do Due é de no máximo 12 bits (pode ser reduzido para comunicação com outras placas do Arduino), com uma faixa de 0 até 3.3 V. Portanto, o LSB vale, aproximadamente, 0.8 mV. A taxa de amostragem do ADC é de 200 kSamples/s, que é maior que o dobro da frequência de interesse no projeto (a segunda harmônica de 1 kHz) [9].

b Comunicação UART

Como visto anteriormente, o Arduino Due fará as conversõe necessárias do sistema de controle e a FPGA vai aplicar o algoritmo Goertzel as entradas amostradas. A comunicação entre estes dois elementos pode ser realizado de duas formas: paralelo ou serial. Paralelo, seria um canal (fio) para cada bit a ser transmitido, o que seriam muitos fios e utilizaria muitas portas e o Due não teria o suficiente. Portanto, foi escolhido a comunicação serial.

A comunicação deve seguir um protocolo para que uma placa entenda a mensagem da outra. Por simplicidade, foi escolhido a comunicação assíncrona UART (Universal Asynchronous Receiver Transmitter). Esta comunicação necessita de três portas para realizar a comunicação. Uma porta é o transmissor (Tx), canal onde transmitirá informações, outra é o receptor (Rx), canal pelo qual receberá informações e por fim tem o terra. Portanto, para realizar a comunicação serial UART de um equipamento com outro é necessário ligar o Tx ao Rx do outro e vice-versa como apresentado na figura 36.

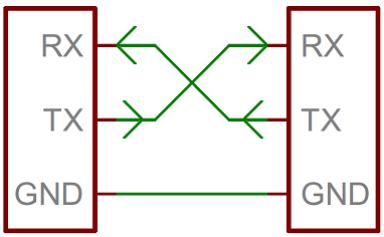

Figura 36: Ligações para realizar a comunicação UART ente dois equipamentos.

Como a comunicação é assíncrona, ou seja, não tem um sinal de relógio de referência entre os dois equipamentos. Por isso, os dois comunicadores devem ter a mesma taxa de transmissão de bits (Baud Rate). O inverso da baud rate (bit slot) é o tempo para de duração de um bit. Há alguns valores padronizados de Baud Rate que vão de 300 bps a 115.200 kbps. No projeto fora utilizado um taxa de 115.200 kbps.

O número de palavras bits a ser transmitidos deve ser conhecido entre os dois equipamentos. Assim como a taxa de bit há padrões. Normalemnte é enviado uma palavra de 5 a 8 bits de de informação por transmissão. Nesta menssagem é acrescentado um bit de início (start bit) com nivel lógico 0 e um ou dois bits de fim (stop bit) com nível lógico 1. Ademais, pode conter ou não um bit de paridade, que informa se o número de bits 1 é par (paridade par) ou impar (paridade impar). A paridade é um método indetificador de erros simples, pois se o número de bits errados for multiplo de dois, ele não é detectado.

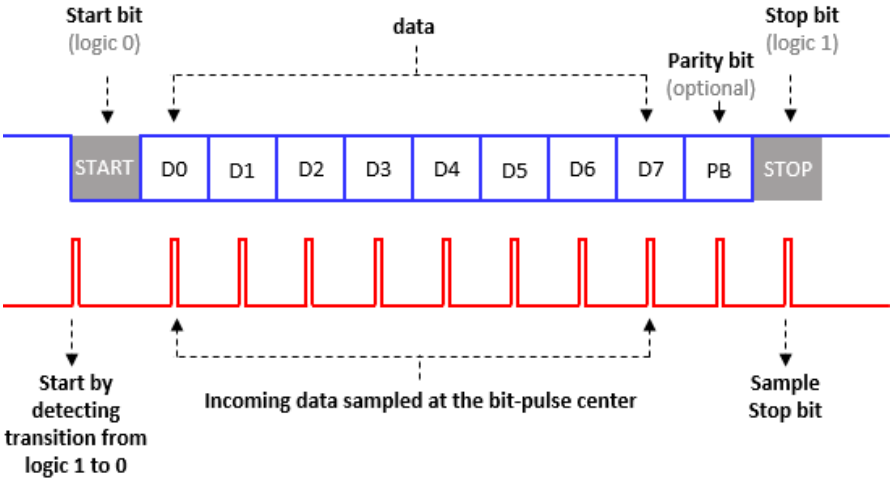

Figura 37: Formato de pacote da comunicação UART [10].

O transmissor tem um baud rate conhecido pelo receptor e segue algum dos padrões de palavra citados. Por exemplo o da figura 37, 8 bits de dados, 1 bits de fim e com bit paridade. O transmissor fica em nível lógico 1 até inicar a transmissão com o bit zero de inicio e volta ao nível lógico 1 após o bit de fim.

O receptor sabe a taxa de bits da transmissão e sabe como é a organização da informação que irá receber. Então, no instante que o o sinal vindo do transmissor muda de 1 para 0 (strat bit), o receptor conta o tempo de um bit slot e meio deixando no meio do primeiro bit de dado. Após a espera ele começa a ler e contar os dados a cada bit slot. Ao atingir o número de dados que deve receber bits da palavra, o receptor para de ler os dados e aguardar outro start bit.

c Controle PID

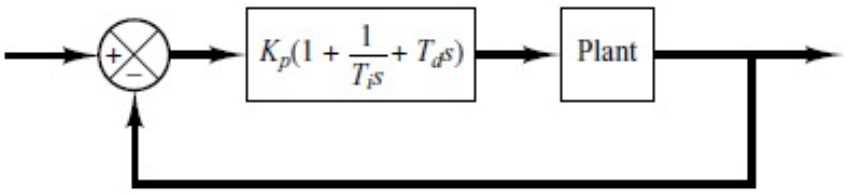

Figura 38: Diagrama de blocos da implementação de um PID para controlar uma planta [11].

O controlador PID e suas variações são sistemas muito utilizados nas industrias para realizar o controle de uma processo. Um sistema de controle PID simples é apresentado na figura 38. A planta é o sistema a ser controlado e a entrada do sitema de malha fechada é a referência (setpoint). A diferença entre a saída da planta e o setpoint é o erro. O PID têm a função de minimizar o erro alterando a entrada da planta.

O controlador é composto por três elementos básicos: proporcional, integral e diferencial. O proporcional é basicamente um ganho no valor do erro. O integral, como nome diz é a integral do erro com um ganho, na figura 38 é representado como k_p/T_i, porém também pode ser indicado apenas como k_i. O diferencial é a derivada do erro com um ganho, na figura de $k_p\, T_d$, mas, também pode ser representado como k_d [11].

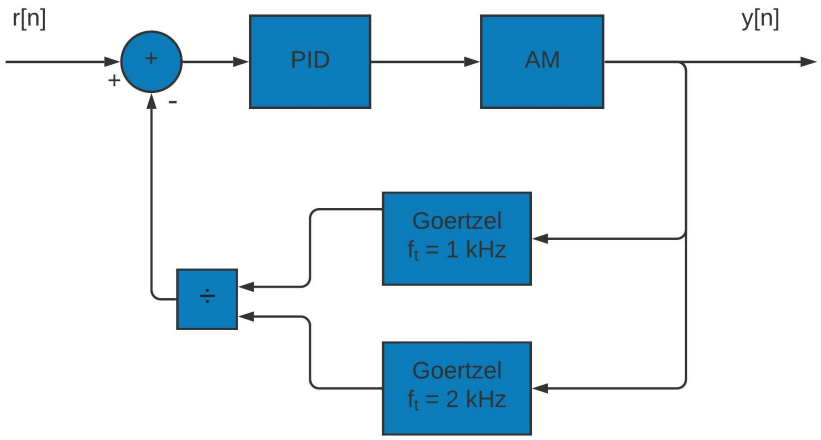

Figura 39: Diagrama de blocos simplificado do sistema de controle.

As variáveis do sistema de controle são os ganhos de cada elemento do PID, k_p, k_i e k_d. A qualidade e até mesmo a controlabilidade do sistema estão associadas a estes números. Existem diversos métodos para encontrar os valores para as constantes do PID de forma atender algum requisito de

projeto. Porém, a planta no nosso caso é um sistema não linear e variante no tempo, o que torna as contas bem complexas. Por tanto o método mais simples para encontrar os melhores valore é o método empirico, variando as constantes e observando os resultados.

O diagrama de blocos apresentado na figura 39 representa a idéia básica do sistema de controle pensado. As única diferenças para o sistema anterior é que na malha de realimentação foi adicionado os dois algoritmos Goertzels e a realimentação é a razão entre eles. Outra diferença é no tempo, que agora é discreto, porém não altera funcionameto, as única mudaças relevantes são que integral é um somatório e o diferencial é a difereça entre o valor anterioe o atual.

d Montagem experimental

Como o Arduino já seria utilizado para a conversão analógica digital e digital analógica, optou-se por realizar um controlador PID no próprio Arduino Due. A própra empresa Arduino fornece uma biblioteca PID, que permite a alteração dos ganhos do k_p, k_i e k_d, limites de saída, setpoint e o tempo para ler uma próxima amostra.

Portanto, o sistema proposto para o controle do drift da função de transferênca é desenvolvido. Primeiro o laser a ser modulado, passa por um controlador de polarização em seguida vai para o MZM. A função do PC é maximizar a a potência de saído modulador, pois a modulação esta associada a polarização como visto.

Ao passar pelo AM, o laser será modulado pelo tom de baixa frequência de 1 kHz mais o sinal de informação na porta de sinal. Na porta de bias estará ligado a saída do sistema de controle. Ao sair da do MZM, o sinal modulado é divido por um divior de feixes (BS, Bean Splitter) de 90-10 ou 99-1. O sistema de controle recebe a porta de menor percentual, enquanto o outro é a saída modulada.

A porta do BS que vai ara o sistema de controle entra num fotodetector de ganho controlável e que seja bom em baixas frequências. O sinal então é amostrado pelo Arduino Due e após 1000 amotras, elas são enviada para FPGA, de forma serial. São realizadas as interações de dois Goertel, um com 1 kHz de frequência alvo, outro com o dobro. Ao término das iterações a FPGA envia os dois resultados para o Arduino.

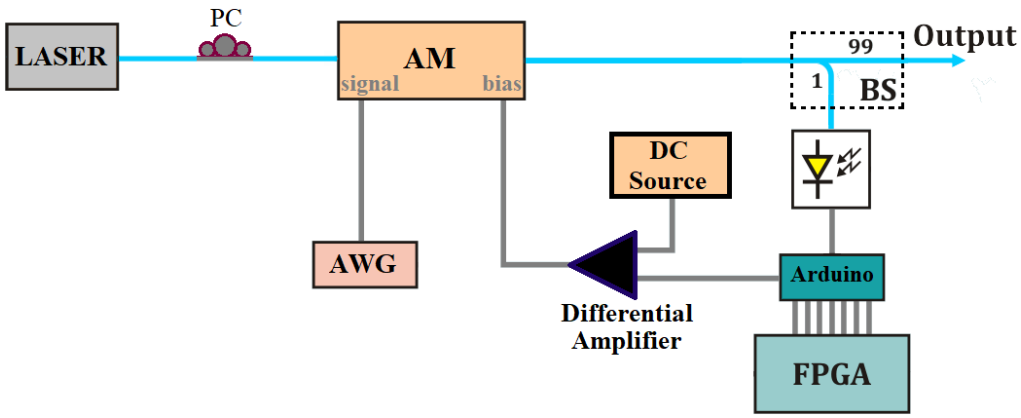

Figura 40: Montagem do sitema de controle do bias do AM para operação região linear.

O arduino faz a razão entre os dois sinais e processa o PID com este valor. A saída do PID vai para o DAC do Due que responde uma representação de tensão entre 0.55 e 2.75 V. Como estes valores não podem ser negativo nem zero, foi pensado em utiziar um AmpOP diferencial.

Este elemento, realiza a diferença entre o sinal do DAC e 1.1 V, a tensão média da representação. Além disso, ele pode dar amplificar o sinal da realimentação com fator 2. Mesmo o ganho reduzindo a sensibiidade para 1 mV, é mais enconta, pois aumenta a excursão de tensão. Este ganho também pode ser controlado utilizizando potênciometros, dado que ele altera o controle. Nesse caso,

para controla o ganho ideal utilziar um amplificador diferencial de precisão, porque o amplificador diferencial simples precisa de dosi potênciometros para contorlar o gnho a saída.

5 Conclusões & trabalhos futuros

Apesar dos rsultaods promissores erros na comunicação serial entre o Arduino e a FPGA estragam as amostras recebidas pelo Goertzel, impossibilitando a realização do controle.

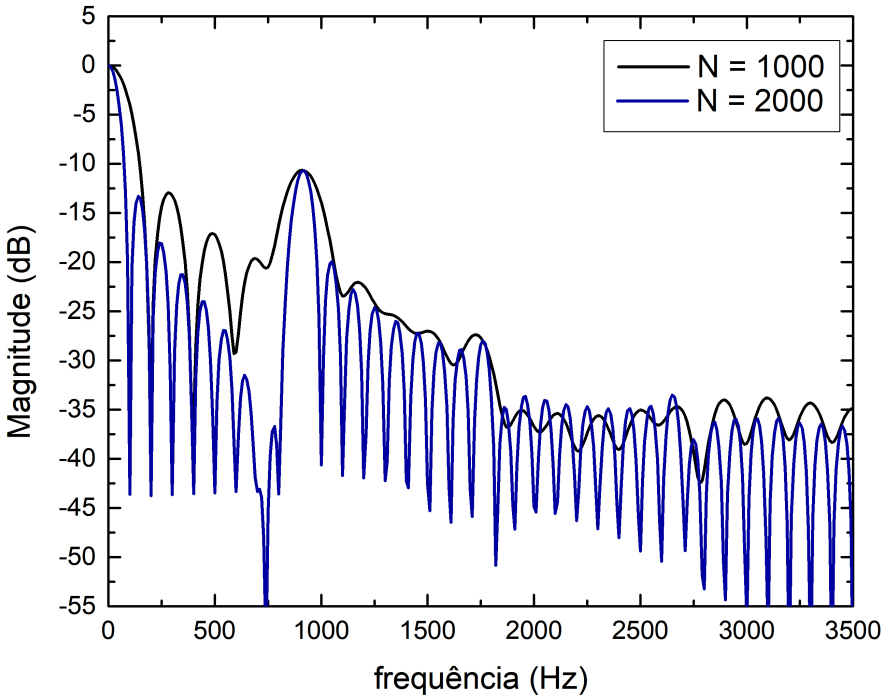

Figura 41: Função de transferência do Goertzel em dB para 1000 e 2000 iterações, com o sinal modulado na região linear com 900 Hz.

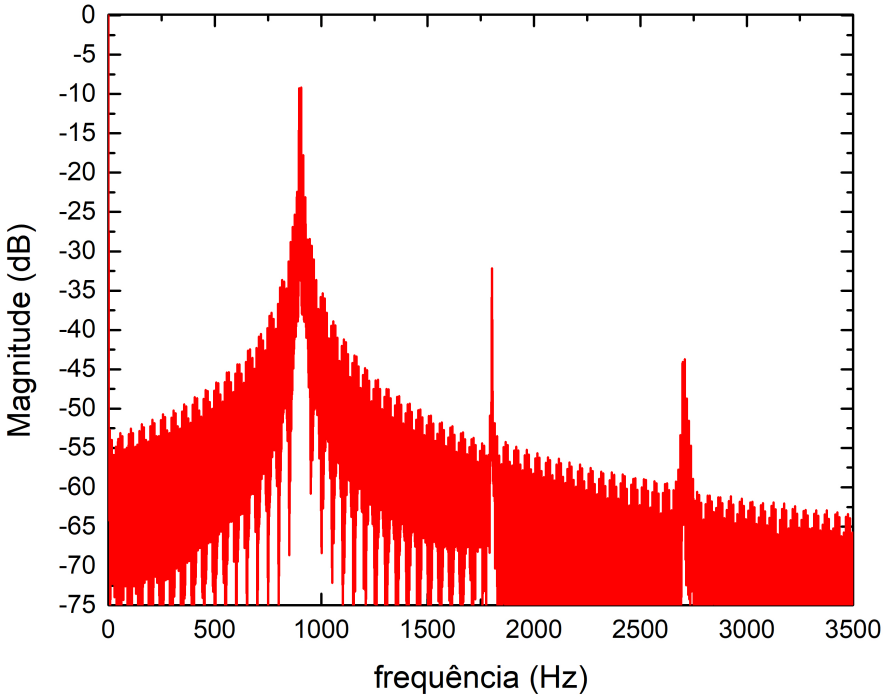

Figura 42: FFT do sinal modulado na região linear com 900 Hz em dB.

A fim de apresentar o funcionamento promissor do algoritmo Goertzel implementado na FPGA foram realizadas medidas da função de transferêcia do Goertzel, reccebendo um sinal modulado dentro e fora da região linear do MZM. As medidas passaram pelo sistema da figura 25 e então foi variado a frequência target do Goertzel em passos de 10 Hz. Portanto, as medidas comparam a função de transferência do Goertzel para 1000 e 2000 iterações com a FFT do MATLAB.

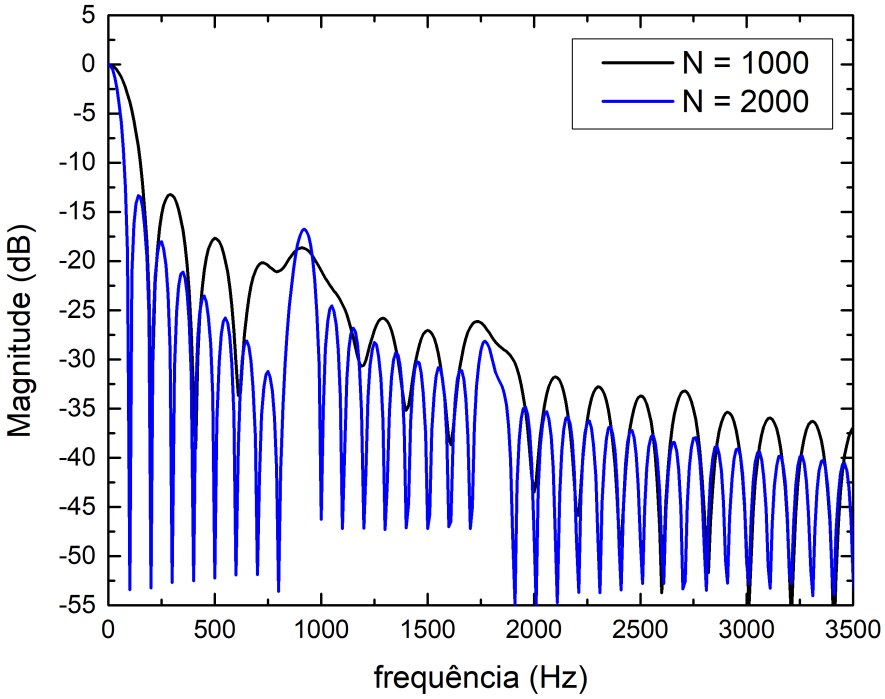

Figura 43: Função de transferência do Goertzel em dB para 1000 e 2000 iterações, com o sinal modulado na região linear com 900 Hz.

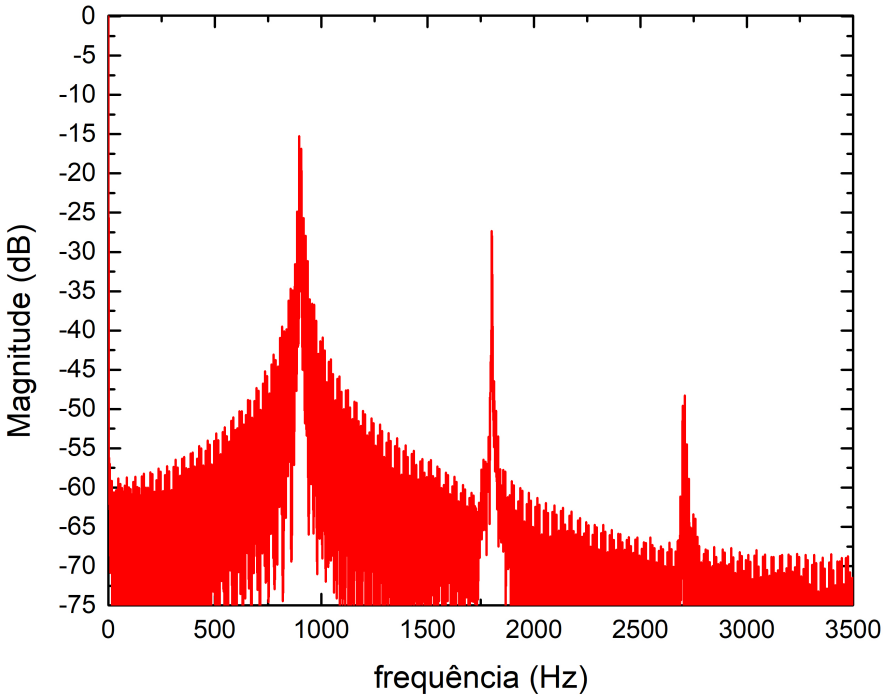

Figura 44: FFT do sinal modulado fora da região linear com 900 Hz em dB.

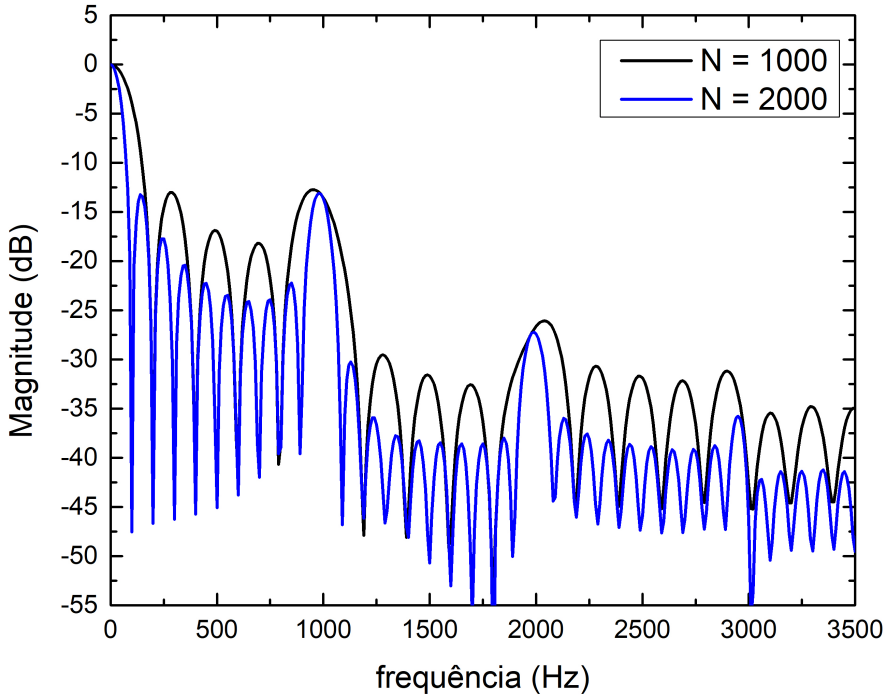

Figura 45: Função de transferência do Goertzel em dB para 1000 e 2000 iterações, com o sinal modulado fora região linear com 1 kHz Hz.

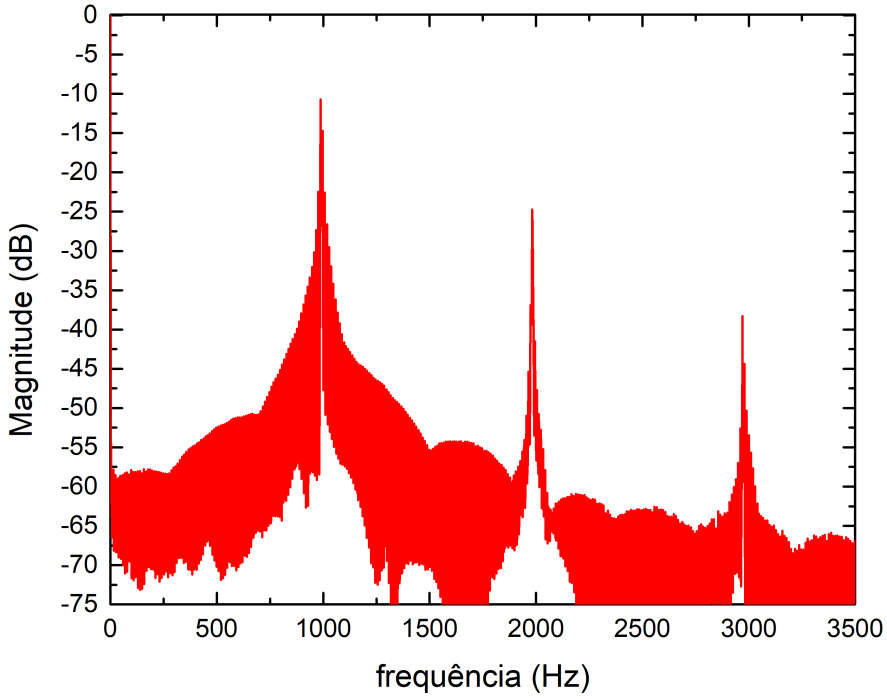

Figura 46: FFT do sinal modulado fora da região linear com 1 kHz em dB.

A partir dos resultados apresentados a cima, observase que na proximidade das frequências mais intensas no sinal, a resposta do Goertzel se coporta semlhante ao teórico. Claro que há interçãoes das outras frequências criando vales ou pics onde não deveria ter, porém já muito significante esse resultado. Ademias, nota-se que em alguns casos, ao dobrar o número de interações o número de ondulções dobra, seguindo de acordo com a teoria. Além da semelhaça com os reultado teorico, é

possível identificar com olho as frequências do sinaldesde o DC, até a terceira harmonica, como na figura 45.

A partir das informações apresentadas, observa-se que o prjeto integra diversar áreas estudadas ao longo do curso de engenharia elétrica: Contorle, processamento digital de sinais, eletrônica analógica, eletrônica digital e comunicações ópticas. Mesmo não tendo obtido o resultado esperado, muito conhecimento foi adquirido e o projeto ainda pode ser terminado.

Portanto, agora para seguir o projeto é ncessário realizar a comunicação com o Arduino, descobrir onde está ocorrendo o erro na transmissão, ou mesmo utilizar outro protocolo de comunicação serial, como I2C. Em seguida, é tirar medidas do sistema de controle e obter a melhor configuração de constantes pro controlador PID.

Referências

[1] J. Svarný, “Analysis of quadrature bias-point drift of mach-zehnder electro-optic modulator,” *12th Biennial Baltic Electronics Conference*, pp. 231–234, 2010.

[2] ——, “Bias driver of the mach-zehnder intensity electro-optic modulator, based on harmonic analysis,” *Advances in Robotics, Mechatronics and Circuits*, pp. 184–189, 2014.

[3] *On the Characterization of Long-Term Drift of LiNbO3 Intensity Modulators*, Aveiro, Portugal, 2019.

[4] R. G. Lyons, *Understanding Digital Signal Processing*, 3rd ed. Prentice Hall, 2010.

[5] A. Oppenheim and R. Shaffer, *Discrete-Time Signal Processing*, 2nd ed. Prentice Hall, 1999.

[6] P. Harowitz, *The Art of Eletronics*, 3rd ed. Cambridge University Press, 1980.

[7] Texas Instruments, “Understanding data converters,” Texas Instruments, Tech. Rep., 1995.

[8] *SAM3X / SAM3A Series*, Atmel, 3 2015.

[9] Arduino, “Arduino due,” December 2019. [Online]. Available: https://store.arduino.cc/usa/due

[10] Soliton, “Uart protocol validation service,” December 2019. [Online]. Available: https://www.solitontech.com/uart-protocol-validation-service/

[11] K. Ogata, *Modern Control Engineering*, 5th ed. Prentice Hall, 2010.

A Algoritmo Goertzel implementado em VHDL

Algoritmo Goertzel

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6
7  entity Goertzel_MZM is
8     port(
9        LOCAL_CLOCK : IN  STD_LOGIC;
10       ENABLE      : IN  STD_LOGIC;
11       RESET       : IN  STD_LOGIC;
12       SAMPLE      : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);  -- 10 bits ADC
13       COEF        : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
14
15       -- DUBUG --
16       DEBUG_STATE      : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
17       DEBUG_SAMPLE_IN  : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
18       DEBUG_A_MUL1     : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
19       DEBUG_B_MUL1     : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
20       DEBUG_MUL1_DONE  : OUT STD_LOGIC;
21       DEBUG_A_MUL2     : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
22       DEBUG_B_MUL2     : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
23       DEBUG_MUL2_DONE  : OUT STD_LOGIC;
24       DEBUG_A_SUM      : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
25       DEBUG_B_SUM      : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
26       DEBUG_SUM_DONE   : OUT STD_LOGIC;
27       DEBUG_BUF_A      : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
28       DEBUG_BUF_B      : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
29       DEBUG_Q1         : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
30       DEBUG_Q2         : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
31       -- END DEBUG --
32
33
34       DONE         : OUT STD_LOGIC;
35       MAG          : OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
36 end Goertzel_MZM;
37
38 architecture Behavioral of Goertzel_MZM is
39
40    COMPONENT FAST_INT_2_FLOAT
41         PORT(
42                 LOCAL_CLOCK  : IN std_logic;
43                 INPUT_DATA   : IN std_logic_vector(9 downto 0);
44                 OUTPUT_DATA : OUT std_logic_vector(31 downto 0));
45         END COMPONENT;
46
47         COMPONENT FLOAT_MULTIPLIER
48         PORT(
49                 ENABLE_MULTIPLIER : IN std_logic;
50                 LOCAL_CLOCK       : IN std_logic;
51                 X_IN              : IN std_logic_vector(31 downto 0);
52                 Y_IN              : IN std_logic_vector(31 downto 0);
53                 RESULT            : OUT std_logic_vector(31 downto 0);
54                 MULT_DONE         : OUT std_logic);
55         END COMPONENT;
56
57    COMPONENT PIPELINED_FLOAT_ADDER_TM
58         PORT(
59                 LOCAL_CLOCK : IN std_logic;
60                 RESET       : IN std_logic;
61                 SYNC_IN     : IN std_logic;
62                 X           : IN std_logic_vector(31 downto 0);
63                 Y           : IN std_logic_vector(31 downto 0);
64                 SYNC_OUT    : OUT std_logic;
65                 FLOAT_SUM   : OUT std_logic_vector(31 downto 0));
66         END COMPONENT;
67
68    SIGNAL Q1              : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
69    SIGNAL Q2              : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
70
71    SIGNAL SAMPLE_IN       : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
72    SIGNAL SAMPLE_IN_BUF   : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
73
74    SIGNAL SUM             : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
75    SIGNAL A_SUM           : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
76    SIGNAL B_SUM           : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
77    SIGNAL SUM_DO          : STD_LOGIC := '0';
78    SIGNAL SUM_DONE        : STD_LOGIC := '0';
```

```
79
80      SIGNAL MUL1             : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
81      SIGNAL A_MUL1           : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
82      SIGNAL B_MUL1           : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
83      SIGNAL MUL1_DO          : STD_LOGIC := '0';
84      SIGNAL MUL1_DONE        : STD_LOGIC := '0';
85
86      SIGNAL MUL2             : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
87      SIGNAL A_MUL2           : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
88      SIGNAL B_MUL2           : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
89      SIGNAL MUL2_DO          : STD_LOGIC := '0';
90      SIGNAL MUL2_DONE        : STD_LOGIC := '0';
91
92
93      SIGNAL BUF_A            : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
94      SIGNAL BUF_B            : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
95
96      SIGNAL MAG_BUF          : STD_LOGIC_VECTOR(31 DOWNTO 0):= (OTHERS => '0');
97
98      SIGNAL STATE            : STD_LOGIC_VECTOR(2 DOWNTO 0) := (OTHERS => '0');
99
100     SIGNAL COUNT            : STD_LOGIC_VECTOR(11 DOWNTO 0) := (OTHERS => '0');
101     SIGNAL DONE_BUF         : STD_LOGIC := '0';
102 begin
103
104
105         INT2FLOAT: FAST_INT_2_FLOAT
106     PORT MAP(
107                 LOCAL_CLOCK  => local_clock,
108                 INPUT_DATA   => sample,
109                 OUTPUT_DATA  => sample_in_buf
110     );
111
112     FLOAT_ADDER: PIPELINED_FLOAT_ADDER_TM PORT MAP(
113                 LOCAL_CLOCK => local_clock,
114                 RESET       => reset,
115                 SYNC_IN     => sum_do,
116                 X           => a_sum,
117                 Y           => b_sum,
118                 SYNC_OUT    => sum_done,
119                 FLOAT_SUM   => sum
120         );
121
122         FLOAT_MULT_1: FLOAT_MULTIPLIER PORT MAP(
123                 ENABLE_MULTIPLIER => mul1_do,
124                 LOCAL_CLOCK       => local_clock,
125                 X_IN              => a_mul1,
126                 Y_IN              => b_mul1,
127                 RESULT            => mul1,
128                 MULT_DONE         => mul1_done
129         );
130
131         FLOAT_MULT_2: FLOAT_MULTIPLIER PORT MAP(
132                 ENABLE_MULTIPLIER => mul2_do,
133                 LOCAL_CLOCK       => local_clock,
134                 X_IN              => a_mul2,
135                 Y_IN              => b_mul2,
136                 RESULT            => mul2,
137                 MULT_DONE         => mul2_done
138         );
139
140 process(local_clock,reset)
141 begin
142
143     if (reset = '1') then
144         state     <= "000";
145
146         count     <= "000000000000";
147
148         a_sum     <= x"00000000";
149         b_sum     <= x"00000000";
150         sum_do    <= '0';
151
152         a_mul1    <= x"00000000";
153         b_mul1    <= x"00000000";
154         mul1_do   <= '0';
155
156         a_mul2    <= x"00000000";
157         b_mul2    <= x"00000000";
158         mul2_do   <= '0';
159
160         buf_a     <= x"00000000";
161         buf_b     <= x"00000000";
```

```
162
163         q1        <= x"00000000";
164         q2        <= x"00000000";
165
166       elsif (local_clock'event and local_clock = '1') then
167         case state is
168           when "000" =>         -- Convert Sample into float
169             done_buf <= '0';
170             if(enable = '1') then
171               sample_in <= sample_in_buf;
172               state     <= state + "1";
173             end if;
174           when "001" =>         -- BUF_A = COEF*Q1   ;   BUF_B = S-Q2
175             a_mul1  <= coef;
176             b_mul1  <= q1;
177             mul1_do <= '1';
178
179             a_sum  <= not(q2(31)) & q2(30 downto 0);
180             b_sum  <= sample_in;
181             sum_do <= '1';
182
183             if (sum_done = '1') then
184               mul1_do <= '0';
185               sum_do  <= '0';
186               buf_a   <= mul1;
187               buf_b   <= sum;
188               state   <= state + '1';
189             end if;
190
191           when "010" =>         -- Q0 = COEF*Q1 + S-Q2 = BUF_A + BUF_B
192             a_sum  <= buf_a;
193             b_sum  <= buf_b;
194             sum_do <= '1';
195             q2     <= q1;
196
197             if (sum_done = '1') then
198               sum_do  <= '0';
199               q1      <= sum;
200               state   <= state + '1';
201             end if;
202
203           when "011" =>         -- BUF_A = Q1*Q1   ;   BUF_B = Q2*Q2
204             a_mul1  <= q1;
205             b_mul1  <= q1;
206             mul1_do <= '1';
207
208             a_mul2  <= q2;
209             b_mul2  <= q2;
210             mul2_do <= '1';
211
212             if(mul1_done = '1' and mul2_done = '1') then
213               mul1_do <= '0';
214               mul2_do <= '0';
215               buf_a   <= mul1;
216               buf_b   <= mul2;
217               state   <= state + '1';
218             end if;
219
220           when "100" =>         -- BUF_A = BUF_A + BUF_B   ;   BUF_B = Q1*Q2
221             a_sum  <= buf_a;
222             b_sum  <= buf_b;
223             sum_do <= '1';
224
225             a_mul1  <= q1;
226             b_mul1  <= q2;
227             mul1_do <= '1';
228
229             if (sum_done = '1') then
230               sum_do  <= '0';
231               mul1_do <= '0';
232               buf_a   <= sum;
233               buf_b   <= mul1;
234               state   <= state + '1';
235             end if;
236
237           when "101" =>         -- BUF_B = COEF*BUF_B
238             a_mul2  <= coef;
239             b_mul2  <= buf_b;
240             mul2_do <= '1';
241
242             if(mul2_done = '1') then
243               mul2_do <= '0';
244               buf_b   <= mul2;
```

```
245                 state   <= state + '1';
246               end if;
247
248             when "110" =>         -- MAG = BUF_A - BUF_B
249               a_sum  <= buf_a;
250               b_sum  <= not(buf_b(31)) & buf_b(30 downto 0);
251               sum_do <= '1';
252
253               if (sum_done = '1') then
254                  sum_do   <= '0';
255                  mag_buf  <= sum;
256                  count    <= count + "1";
257                  state    <= "000";
258                  if (count = STD_LOGIC_VECTOR(to_unsigned(1000 - 1, count'length))) then -- STD_LOGIC_VECTOR(to_unsigned
                         (100, count'length)) samples
259                     done_buf <= '1';
260                     count    <= "000000000000";
261                     q1       <= x"00000000";
262                     q2       <= x"00000000";
263                     a_sum    <= x"00000000";
264                     b_sum    <= x"00000000";
265                     a_mul1   <= x"00000000";
266                     b_mul1   <= x"00000000";
267                     a_mul2   <= x"00000000";
268                     b_mul2   <= x"00000000";
269                     buf_a    <= x"00000000";
270                     buf_b    <= x"00000000";
271                  end if;
272               end if;
273
274              when others =>
275                  mag_buf <= x"00000000";
276
277         end case;
278      end if;
279   end process;
280
281   mag  <= mag_buf;
282   done <= done_buf;
283
284
285    -- DUBUG --
286   DEBUG_STATE      <= STATE      ;
287   DEBUG_SAMPLE_IN  <= SAMPLE_IN  ;
288   DEBUG_A_MUL1     <= A_MUL1     ;
289   DEBUG_B_MUL1     <= B_MUL1     ;
290   DEBUG_MUL1_DONE  <= MUL1_DONE  ;
291   DEBUG_A_MUL2     <= A_MUL2     ;
292   DEBUG_B_MUL2     <= B_MUL2     ;
293   DEBUG_MUL2_DONE  <= MUL2_DONE  ;
294   DEBUG_A_SUM      <= A_SUM      ;
295   DEBUG_B_SUM      <= B_SUM      ;
296   DEBUG_SUM_DONE   <= SUM_DONE   ;
297   DEBUG_BUF_A      <= BUF_A      ;
298   DEBUG_BUF_B      <= BUF_B      ;
299   DEBUG_Q2         <= Q2         ;
300   DEBUG_Q1         <= Q1         ;
301    -- END DEBUG --
302
303   end Behavioral;
```