MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE ELETRÔNICA

1º Ten KILMER DE SOUZA E SILVA
Alu Civ BRUNO WAGNER DE ASSIS VIANA

IMPLEMENTAÇÃO EM FPGA DE CONTROLADORES LINEARES

Rio de Janeiro
2016

INSTITUTO MILITAR DE ENGENHARIA

1º Ten KILMER DE SOUZA E SILVA
ALU CIV BRUNO WAGNER DE ASSIS VIANA

IMPLEMENTAÇÃO EM FPGA DE CONTROLADORES LINEARES

Projeto de Final de Curso apresentado ao Curso de Graduação em Engenharia Eletrônica do Instituto Militar de Engenharia.

Orientador: Maj QEM Alberto Mota SIMÕES – D.C.

Rio de Janeiro
2016

INSTITUTO MILITAR DE ENGENHARIA

Praça General Tibúrcio, 80 – Praia Vermelha

Rio de Janeiro – RJ CEP: 22290-270

621.3 Silva, Kilmer de Souza e

S586i Implementação em FGA de controladores lineares / Kilmer de Souza e Silva, Bruno Wagner de Assis Viana; orientados por Alberto Mota Simões– Rio de Janeiro: Instituto Militar de Engenharia, 2016.

56p. : il.

Projeto de Fim de Curso (PROFIC) – Instituto Militar de Engenharia, Rio de Janeiro, 2016.

1. Curso de Engenharia Eletrônica – Projeto de Fim de Curso. 2. Controlador linear. 3. Filtros digitais I. Viana, Bruno Wagner de Assis. II. Simões, Alberto Mota. III. Título. IV. Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

1º Ten KILMER DE SOUZA E SILVA
Alu Civ BRUNO WAGNER DE ASSIS VIANA

IMPLEMENTAÇÃO EM FPGA DE CONTROLADORES LINEARES

Projeto de Final de Curso apresentado ao Curso de Graduação em Engenharia Eletrônica do Instituto Militar de Engenharia.
Orientador: Maj QEM Alberto Mota SIMÕES – D.C

Aprovada em 26 de Setembro de 2016 pela seguinte Banca Examinadora:

Maj QEM Alberto Mota SIMÕES – D.C.ISAE, França

Maj QEM Leandro Teixeira Dornelles – D.C.UFRJ

André Rotava– M.C. IME

Rio de Janeiro
2016

Sumário

RESUMO

O presente trabalho, apresenta a implementação de um controlador linear representado por uma função de transferência de um filtro digital. Foram realizadas a simulação e implementação de filtros digitais de resposta infinita ao impulso (IIR), seletivos em frequência, em um FPGA (Field Programmable Gate Array). Utilizando MATLAB® e um codificador VHDL, juntamente com a ferramenta ISE Design Suite 14.7 foi possível criar e simular os filtros digitais. Esses foram implementados na placa de desenvolvimento BASYS 2 da Digilent®, que contém o FPGA da família SPARTAN 3E XC3S100E. Também foi utilizada a ferramenta ADEPT da Digilent® para carregar a placa com o programa em linguagem VHDL.

Os resultados foram verificados usando a simulação da ferramenta iSIM presente na ISE, e em hardware (tempo real), procurando comparar o comportamento do sinal de saída em ambos cenários. Os resultados obtidos com as simulações para os filtros passa baixa e passa alta, implementadas em FPGA, foram de acordo com a teoria de filtros.

Palavras-chave: Controlador Linear, Filtros digitais, FPGA, MATLAB®, Simulação.

1. Introdução

Os sinais oriundos de suas fontes naturais são essencialmente analógicos. No entanto, o desenvolvimento da microeletrônica propiciou o desenvolvimento de processadores digitais de sinais (DSPs) e circuitos integrados de aplicação específica (ASICs) poderosos e de baixo custo, reduzindo assim os empecilhos do tratamento discreto dos sinais e fazendo a eletrônica digital ocupar cada vez mais o espaço que antes era predominantemente da eletrônica analógica. Na área de controle observa-se também essa tendência e as matrizes de porta de campo programáveis FPGAs se mostram opções interessantes.

Algumas vantagens do tratamento digital dos sinais por FPGA são a menor influência da degradação frente a atenuação sofrida por circuitos integrados quando comparados com circuitos discretos usados no controle analógico, a possibilidade de reconfiguração rápida da lei de controle e a menor sensibilidade ao ruído. Algumas desvantagens são a limitação de banda, a possível perda de resolução e distorção ocasionados no processo de discretização podendo levar sistemas analógicos estáveis a tornarem-se instáveis.

A ideia desse trabalho é realizar a função de transferência de um controlador de um sistema de realimentação simples através de um FPGA. Logo, o FPGA faria as vezes do computador digital e o controlador seria o bloco composto pelo conversor A/D o FPGA e o conversor D/A. As figuras a seguir ilustram de maneira genérica o controlador C(s) que será implementado.

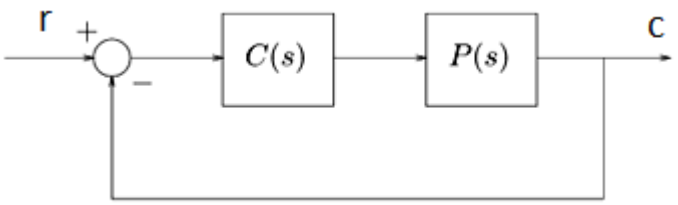

Figura 1 – Realimentação Unitária com controlador analógico

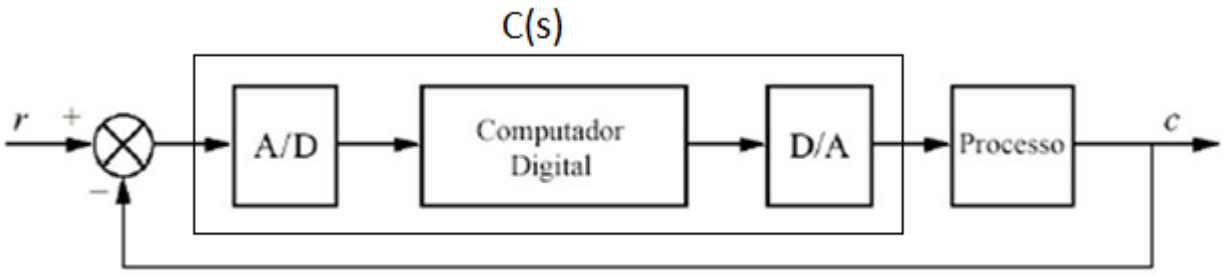

Figura 2 – Realimentação Unitária com controlador digital

Como o controlador é uma função de transferência racional e os filtros digitais IIR também o são, a realização destes é um bom ponto de partida para a realização do controlador.

.

1.1. Motivação

A crescente necessidade de redução de tempo, entre a concepção e a produção de bene de consumo, tem levado indústrias eletrônicas a uma constante busca de soluções que não tornem seus custos elevados. A ascensão dos métodos ao se trabalhar com hardware, alavancados pelo uso de ferramentas EDA (Electronic Design Automation) tem possibilitado o encurtamento do ciclo de desenvolvimento. Em especial, podemos citar os dispositivos que possuem células reconfiguráveis, como os FPGAs [1]. A capacidade de reconfigurabilidade, torna a simulação de diferentes circuitos lógicos, a partir de um programação de descrição de hardware, factível e relativamente simples. Assim, é possível hoje, conceber um projeto de sistema digital sem fazer uso da fabricação de inúmeros chips, em busca de uma versão final [2].

1.2. Objetivo

Este projeto tem como objetivo propor e desenvolver um controlador implementado em FPGA (Field Programmable Gate Array). Assim, este trabalho possibilitará o desenvolvimento de um sistema DSP (Digital Signal Processor), ou mais precisamente, filtros digitais utilizando configuração reconfigurável através de um FPGA.

1.3. Justificativa

Operações relacionadas a sinais, que intrinsicamente estão envolvidas na construção de sistemas, como por exemplo o sistemas de controle ou mesmo um sensoriamento remoto, podem ser realizadas de duas formas: processamento do sinal de forma analógica ou digital. Especificamente, o processo utilizando um sinal digital é feito sobre um hardware que ofereça suporte para um alto desempenho computacional, como no caso de sistemas DSP (Digital Signal Processor) implementados em FPGA. Nesse caso, há flexibilidade de implementar diferentes operações de processamento de sinais (por exemplo, filtragem de sinais), que pode ser alcançado fazendo modificações no software para atingir os resultados esperados pelo projeto.

1.4. Metodologia

Inicialmente será realizado um estudo referente aos conceitos relacionados a filtros digitais, dando enfoque especial ao IIR (Infinite Impulse Response), com o objetivo de conhecer as estruturas básicas e fornecer subsídio para o desenvolvimento do controlador a partir dos parâmetros e limitação do hardware.

Também será estudado detalhadamente os conceitos de dispositivos reprogramáveis para o uso de eletrônica embarcada[1], com o objetivo de tornar sólida a compreensão de FPGAs e posteriormente possibilitando a partir de uma ferramenta computacional gerar um código em linguagem de hardware.

Uma vez determinado os parâmetros mais próximos aos ideais, para o controlador em malha aberta, implementaremos o filtro em linguagem VHDL (VHSIC[2] Hardware Description Language). Após alcançarmos o objetivo exposto, verificaremos se o mesmo atenderá as especificações e resultados pretendidos no início do trabalho.

1.5. Estrutura

No capítulo 2 desse trabalho são introduzidos conceitos básicos de filtros digitais, sistemas de resposta ao impulso e as estruturas para equações de diferenças para forma direta I e II. Neste capítulo, também são discutidos aritmética de precisão finita e conceitos relacionados a área de processamento digital de sinais.

No capítulo 3 são discutidos os conceitos relativos ao FPGA de maneira geral, descrevendo os principais parâmetros e valores padrões dos mesmo para determinadas aplicações e fabricantes.

No capítulo 4 são apresentados as características da família SPARTAN 3, descrevendo o uso do FPGA em múltiplas aplicações. Ainda, apresentamos a placa de desenvolvimento BASYS 2, descrevendo suas características relevantes e como otimizar seu uso.

No capítulo 5, a linguagem de descrição de hardware é analisada, apresentando todos os seus atributos, descrevendo as formas de uso e técnicas usadas para a implementação do código de VHDL.

No capítulo 6 é desenvolvida a lógica de implementação do filtro na linguagem VHDL, apresentando como o código é usado para gerar a programação das células lógicas usando a ferramenta ISE Design Suite 14.7.

No capítulo 7 é descrita a simulação feita como experimento prático a fim de comprovar o resultado esperado pelos filtros.

---

[1] Eletrônica embarcada: é a eletrônica desenvolvida para uma aplicação móvel

[2] Very High Speed Integrated Circuits

O capítulo 8 contém a conclusão do trabalho quanto ao desenvolvido até o momento

2. Processamento de sinais em tempo discreto

Sinais de tempo discreto são representados matematicamente como sequências de números inteiros $x[n]$. Tais sequências surgem frequentemente da amostragem de um sinal analógico (ou seja, de tempo contínuo) $x_a(t)$. O valor numérico do n-ésimo número da sequência é igual ao valor do sinal analógico, $x_a(t)$ no instante nT, isto é:

$$x[n] = x_a(nT), -\infty < n < \infty$$

Um sistema em tempo discreto é definido matematicamente como uma transformação de um operador que mapeia uma sequência de entrada com valores $x[n]$ em uma sequência de saída com valores $y[n]$. Isso pode ser indicado como:

$$y[n] = T\{x[n]\}.$$

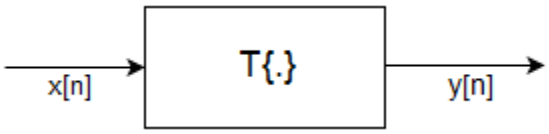

Figura 3 – Transformação em tempo discreto

2.1. Sistemas lineares e invariantes no tempo - LIT

Se $y_1[n]$ e $y_2[n]$ são as respostas de um sistema quando $x_1[n]$ e $x_2[n]$ são as respectivas entradas, então um sistema é linear se apresenta duas propriedades:

- Homogeneidade:

$$T\{x_1[n] + x_2[n]\} = T\{x_1[n]\} + T\{x_2[n]\} = y_1[n] + y_2[n].$$

- Aditividade:

$$T\{ax[n]\} = aT\{x[n]\} = ay[n].$$

Essas duas propriedades juntas, constituem o princípio da superposição:

$$T\{ax_1[n] + bx_2[n]\} = aT\{x_1[n]\} + bT\{x_2[n]\}.$$

Um sistema é invariante no tempo se para todo n_0 a sequência de entradas com valores $x_1[n] = x[n - n_0]$ produz a sequência de saída $y_1[n] = y[n - n_0]$, ou seja é um

sistema no qual um atraso no tempo da sequência de entrada provoca um deslocamento correspondente na sequência de saída.

2.2. Sequência amostra unitária – Impulso $\delta[n]$

A sequência em que apenas um valor vale 1 e todos os outros são nulos é o impulso.

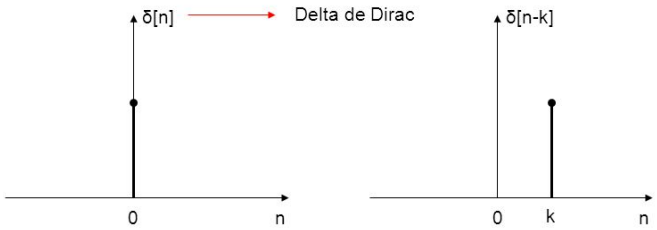

Figura 4 – Função impulso

$$\delta[n] = \begin{cases} 0, & n \neq 0. \\ 1, & n = 0. \end{cases}$$

Portanto a resposta ao impulso $h(n) = T\{\delta[n]\}$ é a resposta do sistema à entrada:

.

$$x[n] = \delta[n]$$

2.3. Convolução

De modo geral, qualquer sequência pode ser escrita da forma

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k].$$

Logo a resposta de qualquer sistema LIT será:

$$y[n] = T\{\sum_{k=-\infty}^{\infty} x[k]\delta[n-k]\} = \sum_{k=-\infty}^{\infty} x[k]T\{\delta[n-k]\} = \sum_{k=-\infty}^{\infty} x[k]h[n-k].$$

A equação acima é chamada soma de convolução e é representada pela notação operacional:

$$y[n] = x[n] * h[n]$$

2.4. Sistemas FIR e IIR

Os sistemas podem ser divididos em sistema de resposta ao impulso de duração finita FIR (do inglês *finite-duration impulse response*), e em sistema de resposta ao impulso de duração infinita IIR (do inglês *infinite-duration impulse response*).

2.5. Resposta em frequência

Para um sistema LIT com resposta ao impulso $h[n]$, se a entrada for uma exponencial complexa $x[n] = e^{jwn}$, a saída segundo a soma de convolução será:

$$y[n] = H\left(e^{jw}\right)e^{jwn}$$

em que:

$$H\left(e^{jw}\right) = \sum_{k=-\infty}^{\infty} h[k]e^{-jwk}$$

é a resposta em frequência do sistema que é um autovalor da autofunção e^{jwn}. Esse resultado mostra que para sistemas LIT a resposta a uma entrada senoidal também é senoidal com amplitude e fase determinadas pelo sistema.

2.6. Transformada Z

A transformada z unilateral de uma sequência $x[n]$ é definida como:

$$X(z) = \sum_{n=0}^{\infty} x[n]z^{-n}$$

e possui propriedades muito úteis na análise de sistemas de tempo discreto. A principal delas é que a transformada da convolução é o produto das transformadas:

$$x_1[n] * x_2[n] \overset{Z}{\leftrightarrow} X_1(Z)X_2(Z)$$

Dessa forma aplicando essa propriedade na saída obtêm-se:

$$Y(z) = H(z)X(Z)$$

em que $H(z)$ é a chamada função do sistema. Como uma sequência e sua transformada z formam um par único, segue-se que qualquer sistema LIT é completamente caracterizado pela sua função de sistema.

2.7. Sistemas Caracterizados por Equações de diferenças de coeficientes constantes

Filtros são uma classe importante de sistema LIT - linear e invariante no tempo e possuem o objetivo de passar certos componentes de frequência de um sinal de entrada e rejeitar outros. Os filtros digitais realizam essa tarefa através da realização de operações matemáticas em um sinal de entrada uniformemente amostrado e discreto no tempo. Os filtros de tempo discreto tipicamente são realizados por meio da implementação de uma equação de diferenças linear com coeficientes constantes da seguinte forma:

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k].$$

Aplicando a transformada z em ambos os membros da equação obtêm-se:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}},$$

que é a função de sistema. A função de sistema, a resposta ao impulso e a equação de diferenças são descrições equivalentes da relação entrada saída de um sistema LTI de tempo discreto. A equação de diferenças pode ser obtida por inspeção da função de sistema que é a transformada z da resposta ao impulso. Os filtros digitais podem ser divididos em dois tipos IIR e FIR segunda a sua resposta ao impulso unitário.

2.8. Representação em diagrama de blocos de equações de diferenças lineares

Para implementar um sistema LIT de tempo discreto através do cálculo de uma equação de diferenças é necessário que os valores de entrada e saída atrasados estejam disponíveis. Para isso são necessários elementos de memória, meios para a multiplicação de valores atrasados e para somá-los.

2.8.1. Forma direta I

Rearranjado os coeficientes da equação de $H(z)$, temos:

$$y[n] = \sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k]$$

que pode ser descrita graficamente pelo seguinte diagrama:

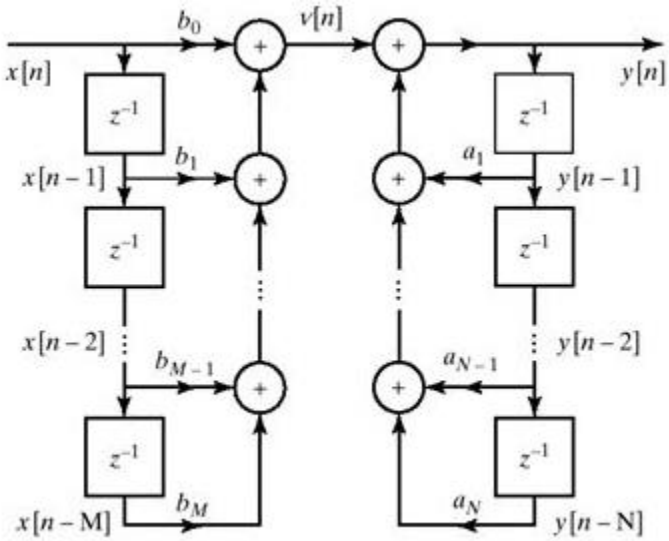

Figura 5 – Forma direta I

em que $v[n]$ e $y[n]$ são:

$$v[n] = \sum_{k=0}^{M} b_k x[n-k]$$

$$y[n] = \sum_{k=1}^{N} a_k y[n-k]$$

A representação por diagrama da figura 5 pode ser implementada diretamente por inspeção da função do sistema.

2.8.2. Forma direta II

Um diagrama de blocos pode ser reorganizado ou modificado de diversas maneiras sem que se mude a função global do sistema. Dessa forma, pode se alterar o diagrama com o intuito por exemplo de minimizar ou otimizar o hardware que implementa o diagrama. Uma maneira de fazer isso seria minimizar a quantidade de elementos de atraso e com menos multiplicadores por constante. Pode-se mostrar que a forma que possui a quantidade mínima de elementos de atraso é a forma direta II. A figura 6 traz a representação gráfica da equações em diferenças para essa forma.

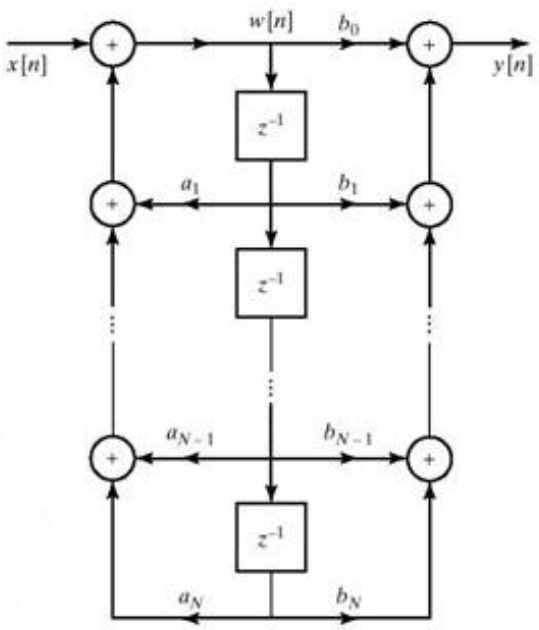

Figura 6 – Forma direta II

Essa forma também conhecida como forma canônica. Sabendo que se trata de uma equação de diferenças, torna-se possível a obtenção da mesma por inspeção da forma direta II.

2.9. Aritmética de precisão finita

Sistemas de tempo discreto podem ser implementados em diferentes estruturas computacionais equivalentes se usada precisão infinita. Quando a precisão é finita (sistemas reais) essas diferentes formas podem se comportar de maneiras distintas. As amostras de saída de um conversor A/D são digitalizadas e podem ser representadas por números binários de ponto fixo. A representação sinal-magnitude mais comum é a representação complemento de dois. Um número real pode ser representado com a lógica de complemento de dois com precisão infinita como:

$$x = X_m\left(-b_0 + \sum_{i=1}^{\infty} b_i 2^{-i}\right)$$

sendo X_m um fator de escala, e os $b_i s$, 0 ou 1. A quantidade b_0 é o bit de sinal. Se $b_0 = 0$, então $0 \leq x \leq X_m$ e, se $b_0 = 1$, então $-X_m \leq x \leq 0$. No processo de conversão A/D o fator de escala seria provavelmente a amplitude de tensão analógica e são usados apenas um número finito de bits $(B + 1)$, o que modifica a equação acima para:

$$\hat{x} = Q_B[x] = X_m\left(-b_0 + \sum_{i=1}^{B} b_i 2^{-i}\right) = X_m \hat{x}_B,$$

em que os números digitalizados estão no intervalo $-X_m \leq x \leq X_m$. A parte fracionária de $\hat{x}$ pode ser representada com a notação posicional:

$$\hat{x}_B = b_0 \blacksquare b_1 b_2 b_3 \ldots b_B$$

Em que ■ representa o ponto binário. A menor diferença entre dois números é chamada passo de quantização Δ

$$\Delta = X_m 2^{-B}$$

A operação de digitalização ou quantização pode ser realizada por arredondamento ou por truncamento. As relações entrada-saída são mostradas nos gráficos a seguir:

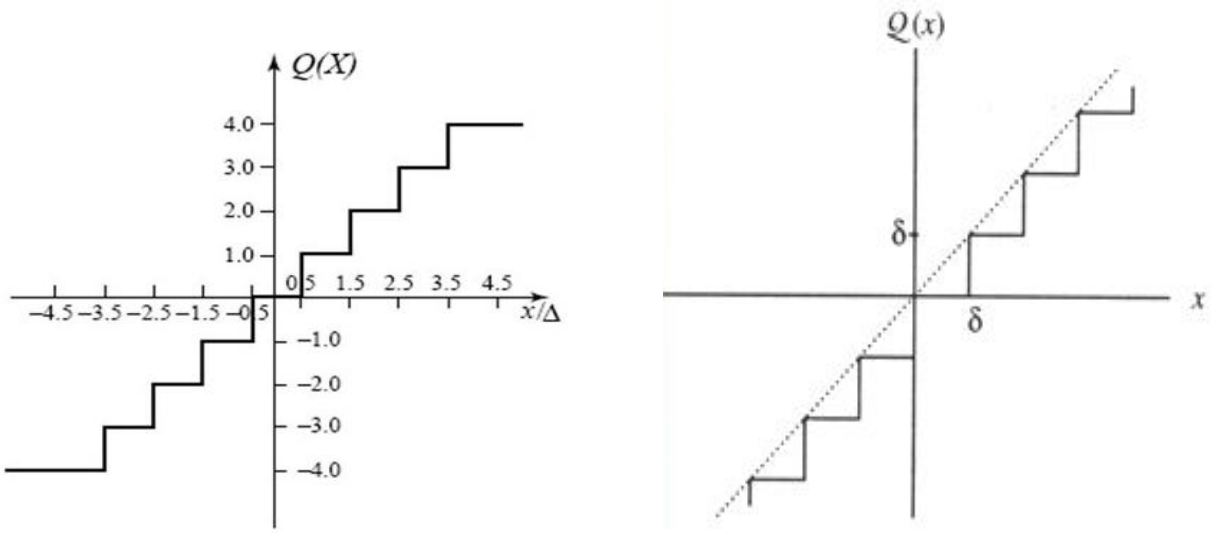

Figura 7 – Processo de quantização

Para avaliar os efeitos da digitalização, é definido o erro de digitalização como:

$$e = Q_B[x] - x$$

Para o arredondamento em complemento de dois, $-\frac{\Delta}{2} \leq e \leq \frac{\Delta}{2}$, e para o truncamento em complemento de dois $-\Delta \leq e \leq 0$. Se um número é maior que X_m, ocorre o chamado overflow (transbordamento). Essa situação pode ocorrer por exemplo quando é efetuado uma soma cujo resultado é maior do que X_m. Existe o transbordamento natural e o transbordamento de saturação. No transbordamento natural a soma de dois

números positivos pode resultar em um número negativo devido a mudança no bit de sinal, como pode ser observado na figura abaixo. No transbordamento de saturação o comprimento do erro não aumenta abruptamente.

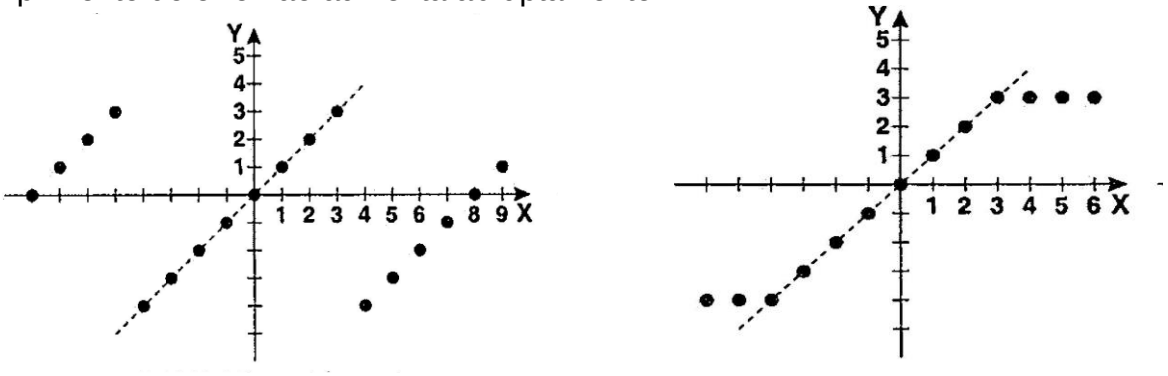

Figura 8 – Transbordamento Natural e saturação

O processo de cálculo que resulta na saída do filtro digital envolve operações de soma e multiplicação por constante e digitalização de coeficientes. Devido a digitalização e aos erros de arredondamentos a saída dos diagramas implementados não é exatamente igual a gerada pela função de sistema que se torna não linear. Dessa forma torna-se necessário um estudo dos efeitos da digitalização dos coeficientes e do efeito do ruído de arredondamento.

A não linearidade do filtro digitalizado causa um comportamento que não ocorre em um sistema linear, o chamado ciclo limite, que consiste numa oscilação periódica da saída quando a entrada se torna nula posteriormente à ocorrência de valores não nulos. Os ciclos limite são causados tanto por digitalização tanto por transbordamento. Portanto são quatro os principais efeitos devido ao uso de precisão finita:

- Efeito da digitalização dos coeficientes
- Efeito do ruído de arredondamento
- Efeito de ciclo limite devido a digitalização
- Efeito de ciclo limite devido ao transbordamento

Os efeitos acima citados serão analisados individualmente a seguir, com ênfase principalmente em sistemas IIR com aritmética de ponto fixo na forma direta II.

2.9.1. Efeitos da digitalização dos coeficientes

Quando os coeficientes de uma função de sistema ou uma equação de diferenças correspondente, são digitalizados os polos e zeros são alterados no plano z e se a estrutura utilizada na implementação for muito sensível à perturbação dos coeficientes,

o sistema IIR resultante pode não mais atender às especificações e até tornar-se instável. Nesse trabalho esse efeito é verificado através de simulação com o matlab®.

2.9.2. Efeito do ruído de arredondamento

A análise de ruído de arredondamento é feita utilizando aproximações lineares de ruído aditivo linear, em que as fontes de ruído consideradas são oriundas das multiplicações dos coeficientes pelos sinais nos ramos da estrutura utilizada na implementação e nos acumuladores. É utilizado um tratamento estatístico sendo feitas as seguintes considerações:

- Cada fonte de ruído de digitalização $e[n]$ é um processo de ruído branco estacionário no sentido amplo.
- Cada fonte de ruído de digitalização tem uma distribuição uniforme de amplitudes sobre todo o intervalo de digitalização
- Cada fonte de fonte de ruído de digitalização é não correlacionada com a entrada do digitalizador correspondente, com todas as outras fontes de ruído de digitalização e com a entrada do sistema.

Fazendo o devido tratamento matemático conclui-se que as diferentes formas de se implementar os filtros geram expressões de ruído que dependem dos coeficientes de cada filtro, sendo difícil a comparação entre os mesmos. No entanto, a utilização de acumuladores de comprimento duplo pode ser usada para reduzir significativamente o ruído de digitalização nos sistemas em forma direta

2.9.3. Ciclo Limite

Para sistemas de tempo discreto IIR de precisão finita quando a excitação se torna zero e permanece assim, a saída pode oscilar indefinidamente com um padrão periódico ao invés de tender assintoticamente para zero, como acontece para o mesmo sistema de precisão infinita. Esse comportamento é chamado ciclo limite de entrada nula, e é uma consequência dos digitalizadores não lineares no laço de realimentação do sistema ou do transbordamento nas adições.

2.9.3.1. Efeito de ciclo Limite devido a digitalização

Esse tipo de ciclo limite pode ser referente ao arredondamento ou ao truncamento sucessivo de produtos em uma equação de diferença iteradas por padrões repetitivos. A

figura 9 mostra um exemplo de ciclo limite devido ao arredondamento para o seguinte de sistema de primeira ordem:

$$y[n] = ay[n-1] + x[n]$$

Em que $a = 1/2$, $ay[n-1]$ é o arredondamento do produto e a entrada é $x[n] = \left(\frac{7}{8}\right)\delta[n]$.

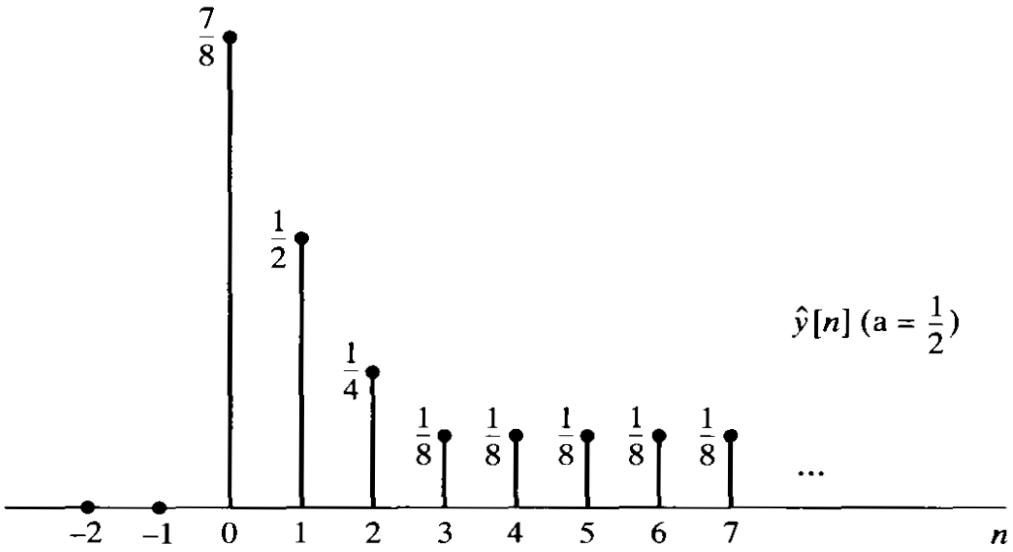

Figura 9 – Ciclo limite de arredondamento

Na figura pode-se perceber que a entrada tornar-se nula a partir de $n = 1$ e a saída não se anula e permanece constante a partir de $n = 3$.

2.9.3.2. Efeito de ciclo Limite devido ao transbordamento

Esse ciclo limite ocorre devido as sucessivas adições de produtos arredondados pelos acumuladores do sistema. Ele pode produzir oscilações de grande amplitude causando grande erro na saída do filtro pois o transbordamento de um número positivo produz um número negativo e vice-versa.

2.9.4. Evitando Ciclos Limite

A oscilações por transbordamento podem ser evitadas usando a característica de transbordamento por saturação. Podem também ser evitadas acrescentando mais bits ao comprimento da palavra computacional, o que causa a redução da amplitude do ciclo limite pelo aumento de bits menos significativos. Além disso, se um acumulador de comprimento duplo for utilizado, de modo que a digitalização ocorra após o acúmulo de produtos, então é muito menos provável que os ciclos limite decorrentes do arredondamento ocorram nos sistemas de segunda ordem. Assim, o compromisso entre

comprimento da palavra e complexidade computacional do algoritmo surge para ciclos limite assim como para digitalização dos coeficientes e ruído de arredondamento.

2.10. Filtro digital

Para a implementação de um filtro digital, se faz necessário um processamento anterior que garanta a entrada de um sinal discreto no dispositivo. Para esse devido fim é necessário a utilização de um conversor analógico-digital (A/D). Assim, a partir da entrada de um sequência binária, a filtragem por um dos métodos discutidos em seção anterior, poderá ser empregado. Com a filtragem realizada, haverá uma nova palavra binária que deverá ser convertida em sinal analógico, a fim desse processo será utilizado um conversor digital-analógico (D/A) na saída desse dispositivo. A figura a seguir ilustra esse procedimento para uma filtragem do sinal.

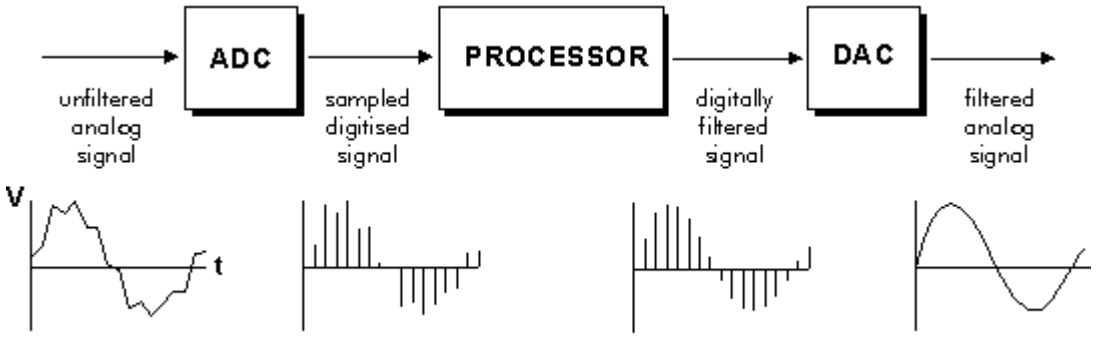

Figura 10 – Filtragem digital

2.10.1. Conversor A/D

O conversor A/D utilizado nesse trabalho é o módulo de expansão da Digilent® PmodAD1.

Figura 11 – PMOD AD1

Esse módulo possui as seguintes características:

- Dois canais de 12-bits com conversores A/D Analog Devices AD7476A simultâneos;
- Taxa de amostragem variável podendo alcançar até 1 milhão de amostras por segundo por canal;
- Dois filtros anti-aliasing Sallen-Key de dois polos;
- Comunicação serial com protocolo semelhante ao SPI com clk serial de até 20 Mhz;
- Baixo consumo de potência;
- Tensão de alimentação de 3.3V ou 5V.

2.10.2. Conversor D/A

O conversor D/A que será utilizado é o modulo de expansão da Digilent PmodAD2.

Figura 12 - PMOD AD2

Esse módulo possui as seguintes características:

- Dois canais de 12-bits com conversores D/A Texas Instruments DAC121S101 simultâneos;
- Comunicação serial com protocolo semelhante ao SPI com clk serial de até 30 Mhz;
- Atinge taxas de até 16,5 milhões de amostras por segundo;
- Baixo consumo de potência;
- Tensão de alimentação de 3.3V ou 5V

3. FPGA

O FPGA surgiu na década de 80, com a empresa Xilinx Inc., como um dispositivo programável que apresentava a possibilidade de ser configurado em uma ampla variedade de aplicações, que poderia ser implementado pelo usuário. Comparativamente aos projetos CMOS VLSI, os dispositivos programáveis evitam o custo inicial e o risco inerente de uma máscara convencional "*gate array*" [3]. Assim, ao se trabalhar com essa tecnologia o desenvolvedor obterá exatamente o hardware pretendido, não fazendo uso de ASSP (Same Application-Specific Standard Product) [4]. Eles são programados com o carregamento de uma sequência adequada de dados, de uma memória externa para células internas de configuração, de forma que interconectem blocos funcionais de circuitos digitais fortemente condensados. O FPGA perde a configuração na ausência de alimentação, porém pode ser programado um número ilimitado de vezes, chegando a admitir relógios acima de 500MHz.

O FPGA é um CI (Integrated Circuit) que permite sua configuração por software e possibilita a implementação de circuitos digitais, como processadores, interfaces, controladores e decodificadores. Essencialmente, consiste em uma disposição condensada de blocos idênticos de circuitos menores, compostos por portas lógicas e flip-flops, com alguns sinais de interface. As conexões entre as saídas de determinados blocos com as entradas de outros são programáveis através de um protocolo simples e de implementação facilitada [3].

FPGAs modernos consistem da mistura de acesso à memória SRAM ou FLASH através de pinos de entrada e saída de alta velocidade, blocos lógicos e roteamento. Mais especificamente, um FPGA contém elementos lógico programáveis (LE), que possuem uma hierarquia de interconexões reconfiguráveis que permite que as LEs sejam fisicamente conectada uma a outra. Pode-se configurar as LEs para que componham funções complexas ou simples portas lógicas, tal como AND ou OR [4].

3.1. Tecnologias de FPGA

Há quatro tipos de tecnologias de FPGA: RAM estática, transistores de passagem, EPROM e EEPROM. Dependendo da aplicação, uma determinada tecnologia poderá apresentar melhor desempenho para o objetivo desejado.

3.1.1. RAM estática

É aquela que implementa as conexões entre blocos lógicos através de portas de transmissão ou multiplexadores controlados por células SRAM. Essa tecnologia possui a vantagem de permitir uma rápida reconfiguração de circuito. Porém, exige um considerável hardware auxiliar, a ser integrado junto aos blocos lógicos, para que seja garantido a sua reconfiguração, como uma memória FLASH EEPROM [5].

3.1.2. Transistores de Passagem

Nesse tipo de estrutura é exigível uma alta densidade de transistores que admitem configuração em modo corte ou condução. Em corte, introduz alta impedância entre dois nós internos. Em condução, implementa a condução entre eles. Por conta dessas características, a solução a transistores torna-se mais barata.

3.1.3. EPROM e EEPROM

Tecnologia semelhante à utilizada na fabricação de memórias EPROM/EEPROM. Possuindo vantagem de permitir reprogramação sem a necessidade de armazenamento da configuração externa, tornando-o mais rápido [5].

3.2. Arquitetura de um FPGA

Como foi dito anteriormente, o FPGA é composto por elementos lógicos. Esses, podem ser divididos em CLB (Configuração Lógica do Bloco), IOB (Bloco de entrada ou saída) e Switch Matrix (Matriz de roteamento). Para entender melhor esses blocos, tomemos como exemplo a ilustração abaixo:

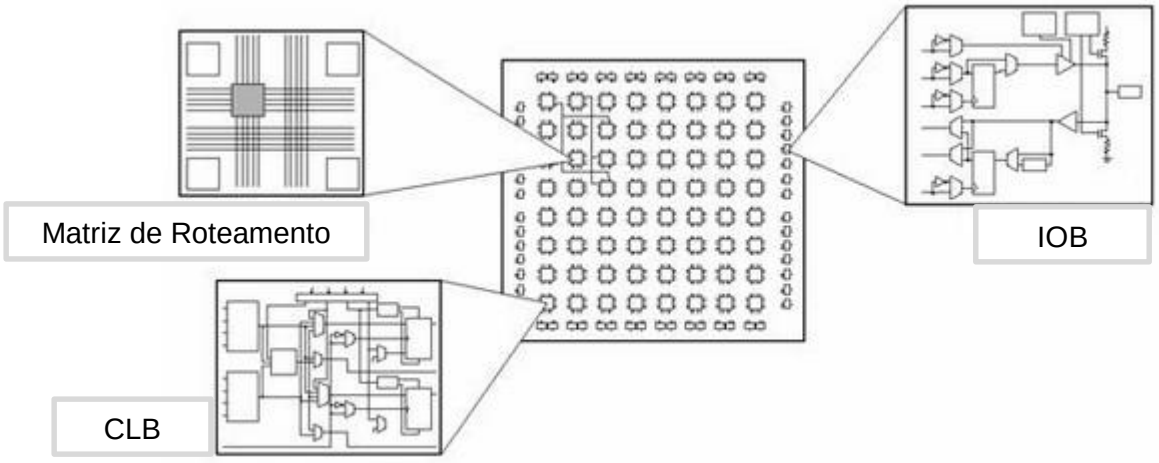

Figura 13 - Arquitetura do FPGA

As CLBs possuem circuitos com a mesma estrutura, são constituídos por flip-flops e lógica combinacional. Assim, o usuário utilizará as CLBs para projetar os elementos funcionais lógicos. Os circuitos IOBs permitem o interfaceamento entre a saída das CLBs e a saída do dispositivo. Podem ser representados por buffers bidirecionais. A matriz de roteamento são trilhas utilizadas para a conexão dos outros blocos. Essas conexões são

estabelecidas por programação, e a essa é dado o nome de roteamento [3]. Outros parâmetros desses blocos serão mais detalhados no próximo capítulo.

3.3. Modos de configuração de um FPGA

Genericamente é possível imaginar o desenvolvimento de um projeto a partir da lógica de cada CLB, porém existem ferramentas disponibilizadas pelo fabricante que tornam a atividade de desenvolvimento da circuitaria mais amigável. Podemos citar os editores de esquemáticos e editores de linguagem descritiva de hardware(HDL). A maioria desses editores possuem um grande número de macros que fornecem a implementação dos blocos funcionais mais comuns. Assim, é possível que projetos e suas atualizações sejam realizados com rapidez e confiabilidade.

Os editores mencionados, geram ao ser simulados e sintetizados, uma sequência binária que poderá ser carregada no FPGA. Para esse carregamento utiliza-se programação direta pelo computador ou por auxílio de uma PROM associada a uma lógica auxiliar. Em ambas, é obtido como produto a configuração do dispositivo.

A configuração de um FPGA pode ser compreendida como um processo de carregamento dos dados de configuração num dispositivo específico, de forma que ele passe a operar de acordo com o projeto lógico desenvolvido em esquemático ou VHDL [5]. Por exemplo a Família XC4000 da Xilinx, possui 350 bits de configuração por CLB, onde cada bit define o estado de uma célula de memória estática, que irá controlar uma função booleana, um multiplexador de entrada ou um transistor de interconexão.

Podemos ainda citar algumas características das ferramentas, conforme escreve Ordonez (2003):

- Especificação do comportamento do FPGA através de diagramas esquemáticos, linguagens de descrição de Hardware (HDL - Hardware Description Language) e ou diagrama de fluxo (máquina de estados);
- Sintetização de circuito obedecendo às restrições impostas pelo projetista; não havendo restrições, a ferramenta de síntese busca a configuração padrão para a síntese do circuito;
- Verificação do funcionamento do circuito através de simulação funcional e temporal, já considerando os tempos de atraso gerados pela lógica resultante do processo de síntese;
- Capacidade de gerar relatórios estatísticos, com dados de comportamento e desempenho do circuito desenvolvido;
- Possibilita a implementação do projeto em nível físico, fazendo com que o circuito programável assuma o comportamento descrito no projeto.

Como existem no mercado inúmeras ferramentas, e o escopo do trabalho abordará apenas a família SPARTAN da XILINX Inc., usaremos a ISE Design Suite 14.7, ferramenta essa que será abordada em detalhes em capítulo posterior.

4. FPGA XILINX SPARTAN 3

No capítulo anterior foi abordado de maneira generalizada o FPGA, porém, na prática, o uso de fabricantes e modelos variam de acordo com a necessidade do projeto a ser implementado. Por isso, esse capítulo, explorará o FPGA do fabricante Xilinx Inc. da família SPARTAN 3E em detalhes, buscando o esclarecimento de sua estrutura, arquitetura e programação.

4.1. Características gerais

O FPGA da família Spartan, mais precisamente a 3E, foi lançado ao mercado em 2005, buscando ser comercializado como um chip com um valor acessível, e que pudesse ser utilizados por uma grande quantidade de produtos, como Roteadores, Modems e Televisões [8]. No presente trabalho, será utilizado o FPGA **XC3S100E**, possuindo as seguintes características, listadas pelo guia do usuário fornecido pelo fabricante [9]:

- Quantidade de recursos lógicos e de IO reduzidos, facilitando a produção de uma placa de teste, pois é minimizada a quantidade de pinos;
- Possui um hardware mais específico, de forma que este FPGA apenas possui recursos lógicos e de roteamento;
- Sua interconexão é similar à família VIRTEX, possibilitando a esse dispositivo uma maior capacidade de processamento.

Essas características são obtidas a partir da arquitetura empregada pelo Spartan 3E, para elucidá-la é mostrado a tabela disponibilizada pelo fabricante[9]:

Tabela 1 – SPARTAN 3E (XC3S100E)

Dispositivo	Portas Lógicas	Equivalente de Portas Lógicas	Matriz de CLB (1 CLB = 4 Slices)			
			Linhas	Colunas	Total CLBs	Total Slices
XC3S100E	100K	2,160	22	16	240	960

Pode-se observar que o modelo empregado, assim como todos os elementos da família Spartan 3E, apresentam uma arquitetura baseada em colunas e linhas, onde estarão localizados os blocos disponíveis à programação [9]. Esses blocos, estão

distribuídos na estrutura do dispositivo, conforme a sua finalidade. Uma visão geral está representado pela figura 14 [9].

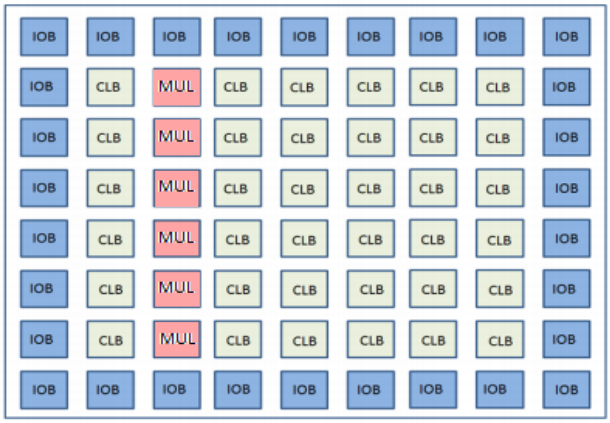

Figura 14 – Arquitetura SPARTAN 3E

As mesmas definições já elucidadas a respeito dos blocos IOB e CLB se repetem nessa arquitetura. Além desses blocos, como observa-se na figura 14, existe o bloco MUL, que é na prática estruturas multiplicadoras em forma de blocos.

Um CLB é compreendido pela junção de 4 slices (Figura 15), sendo que cada slice é composto por duas LUTs (da expressão em Inglês, Look-UP Table) de quatro entradas ligadas a dois flip-flops, portas lógicas e multiplexadores [9]. As LUTs contêm a tabela verdade da função lógica representada, enquanto os flip-flops são utilizados para realizar a função sequencial no slice.

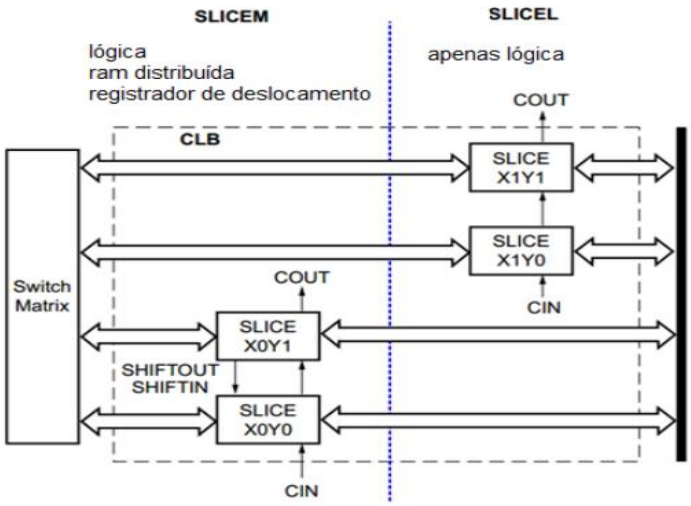

Figura 15 – Arranjo interno ao CLB

Particularmente nessa família de FPGA, existem 2 slices diferentes. O SLICEL só poderá ser configurado em funções lógicas. O SLICEM permite a adição de componentes que o levam a funcionar como elementos de uma RAM ou como registradores [9]. Através da figura 16, pode-se visualizar um slice [9].

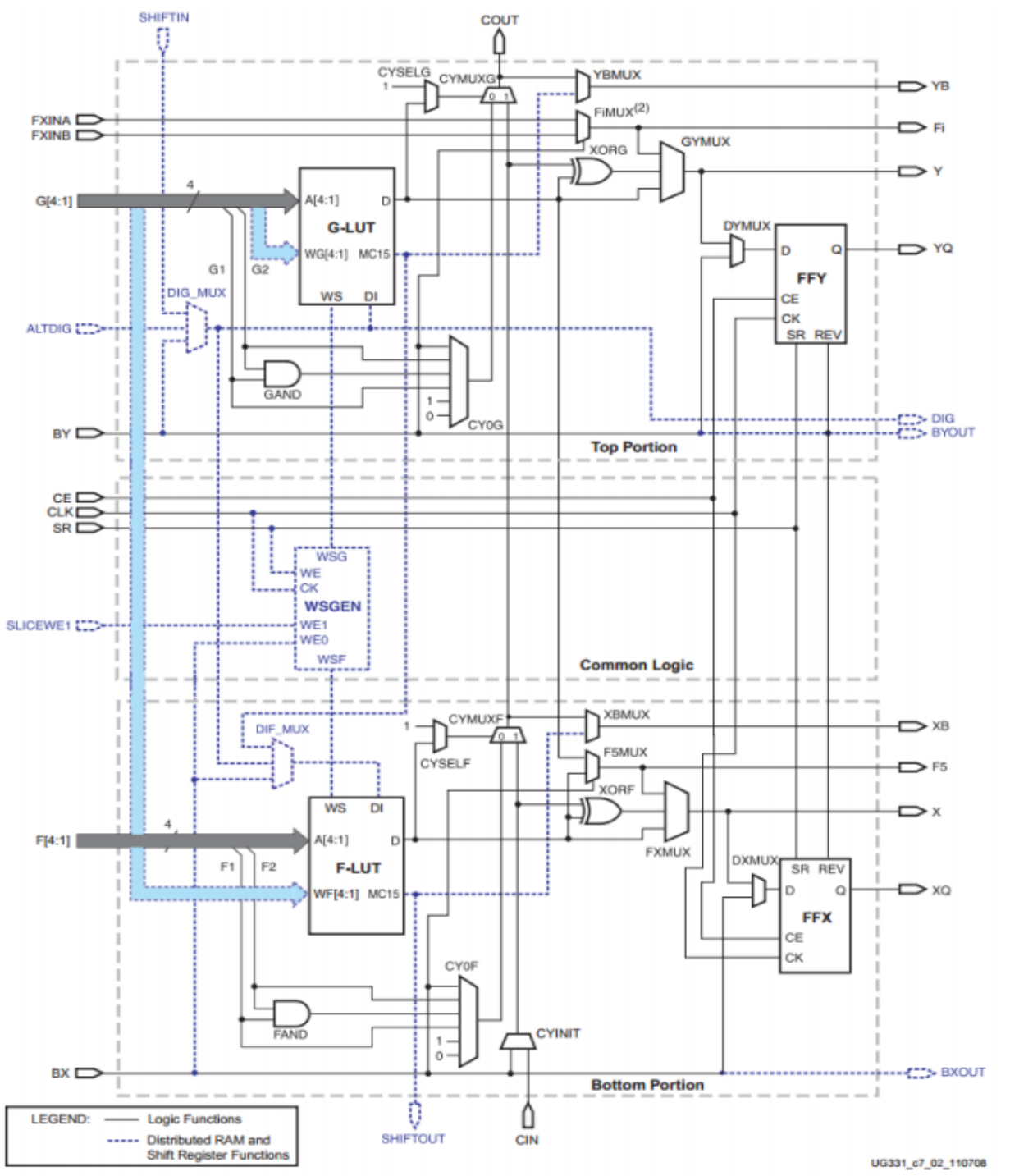

Figura 16 – Visão geral do Slice

Para os multiplicadores introduzidos nesse capítulo, é importante destacar as seguintes características [9]:

- São rápidos e eficientes para a implementação de multiplicação de signed ou unsigned de até 18 bits;
- Pode ser utilizado a fim de obter o deslocamento, magnitude ou o complemento 2 de retorno de um valor;
- Alguns dispositivos da família Spartan 3, permitem o cascateamento com outros CLBs;

Pode-se encontrar na literatura diferenças quanto ao uso e quantidade de multiplicadores entre os integrantes da família Spartan, como é mostrado na figura 17 [9].

Device	Multiplier Columns	Multipliers
Spartan-3A DSP FPGAs		
XC3SD1800A	4	84 DSP48A
XC3SD3400A	5	126 DSP48A
Spartan-3A/3AN FPGAs		
XC3S50A/AN	1	3
XC3S200A/AN	2	16
XC3S400A/AN	2	20
XC3S700A/AN	2	20
XC3S1400A/AN	2	32
Spartan-3E FPGAs		
XC3S100E	1	4
XC3S250E	2	12
XC3S500E	2	20
XC3S1200E	2	28
XC3S1600E	2	36

Figura 17 – Número de multiplicadores por SPARTAN - 3

Como é elucidado pelo fabricante, as famílias Spartan-3E e Spartan-3A / 3AN possuem recursos multiplicadores incorporados semelhantes, porém as famílias Spartan-3A / 3AN oferecem flexibilidade de roteamento adicional. Embora os recursos sejam semelhantes, há diferenças temporal entre as famílias. As famílias compartilham o mesmo multiplicador primitivo, MULT18X18SIO (Figura 18), com registros de entrada e saída opcionais e também possibilidade de sinais em cascata.

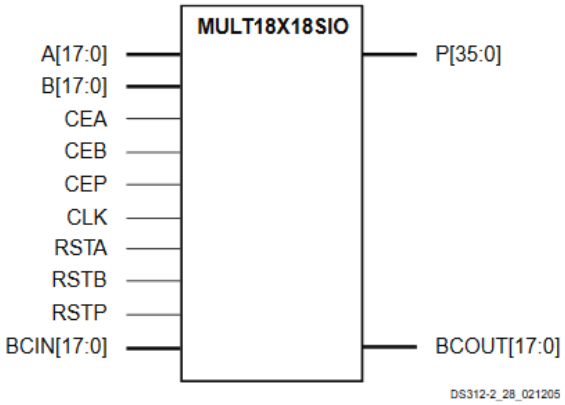

Figura 18 - MULT18X18SIO

O bloco multiplicador funciona de forma a oferecer na saída P o produto AxB. Esses pinos são habilitados respectivamente por CEP, CEA e CEB, existindo ainda o pino para *reset* (RSTA, RSTB e RSTP) e os pinos de resto de operação (BCIN e BCOUT), para entrada e saída, respectivamente.

4.2. Placa de desenvolvimento Basys2

A placa Basys2 é uma plataforma de desenvolvimento e implementação de circuitos. Construída em torno do Fpga Spartan – 3E e do controlador USB Atmel AT90USB2, possibilitando-a ser um hardware pronto para uso. A Basys2 é apropriada para a hospedagem de circuitos que vão desde dispositivos lógicos básicos até controladores complexos [10]. Não é a intenção desse tópico detalhar todos os componentes pertencentes a essa placa, portanto será discutido apenas alguns itens considerados relevantes para o trabalho.

4.2.1. Características e componentes básicos

A plataforma de desenvolvimento em questão, foi criada pela fabricante Digilent® e segundo seu manual de referência possui os seguintes componentes [10]:

- 100,000 portas lógicas;
- Controlador Atmel AT90USB2, que permite além do alimentação da placa a programação do FPGA;
- Memória ROM, para armazenar as configurações iniciais do FPGA;
- 8 LEDs, 4 Displays de 7 segmentos, 4 botões e 8 chaves;
- Porta PS/2 e porta VGA com resolução de 8 bits;

- Relógio de 25/50/100MHz;
- 4 portas para expansão de 6 pinos.

Todos os componentes listados que possuam pinos possuem um endereço atribuído pelo fabricante (Figura 19), e que a partir dos mesmos, possibilita-se o correto roteamento para as funções lógicas pretendidas. Além disso, a placa foi concebida em um circuito impresso de 4 camadas, com as mais internas dedicadas a VCC e GND. O corpo do FPGA e os outros CIs, possuem capacitores cerâmicos de passagem, que são colocados o mais próximo possível ao pino VCC, de forma a resultar em uma fonte de alimentação limpa e de baixo ruído [10]. Caso faça-se uso de alimentação por bateria (não pelo cabo USB), é necessário uma tensão entre 3.5v e 5.5v.

Basys2 Spartan-3E - DEFINIÇÃO DOS PINOS											
Pin	Sinal	Pin	Sinal	Pin	Sinal	Pin	Sinal	Pin	Sinal	Pin	Sinal
C12	JD1	P11	SW0	N14	CC	B2	JA1	P8	MODE0	M7	GND
A13	JD2	M2	USB-DB1	N13	DP	C2	USB-WRITE	N7	MODE1	P5	GND
A12	NC	N2	USB-DB0	M13	AN2	C3	PS2D	N6	MODE2	P10	GND
B12	NC	M9	NC	M12	CG	D1	NC	N12	CCLK	P14	GND
B11	NC	N9	NC	L14	CA	D2	USB-WAIT	P13	DONE	A6	VDDO-3
C11	BTN1	M10	NC	L13	CF	L2	USB-DB4	A1	PROG	B10	VDDO-3
C6	JB1	N10	NC	F13	RED2	L1	USB-DB3	N8	DIN	E13	VDDO-3
B6	JB2	M11	LD1	F14	GRN0	M1	USB-DB2	N1	INIT	M14	VDDO-3
C5	JB3	N11	CD	D12	JD4	L3	SW1	P1	NC	P3	VDDO-3
B5	JA4	P12	CE	D13	RED1	E2	SW6	B3	GND	M8	VDDO-3
C4	NC	N3	SW7	C13	JD3	F3	SW5	A4	GND	E1	VDDO-3
B4	SW3	M6	UCLK	C14	RED0	F2	USB-ASTB	A8	GND	J2	VDDO-3
A3	JA2	P6	LD3	G12	BTN0	F1	USB-DSTB	C1	GND	A5	VDDO-2
A10	JC3	P7	LD2	K14	AN3	G1	LD7	C7	GND	E12	VDDO-2
C9	JC4	M4	BTN2	J12	AN1	G3	SW4	C10	GND	K1	VDDO-2
B9	JC2	N4	LD5	J13	BLU2	H1	USB-DB6	E3	GND	P9	VDDO-2
A9	JC1	M5	LD0	J14	HSYNC	H2	USB-DB5	E14	GND	A11	VDDO-1
B8	MCLK	N5	LD4	H13	BLU1	H3	USB-DB7	G2	GND	D3	VDDO-1
C8	RCCLK	G14	GRN2	H12	CB	B14	TMS	H14	GND	D14	VDDO-1
A7	BTN3	G13	GRN1	J3	JA3	B13	TCK-FPGA	J1	GND	K2	VDDO-1
B7	JB4	F12	AN0	K3	SW2	A2	TDO-USB	K12	GND	L12	VDDO-1
P4	LD6	K13	VSYNC	B1	PS2C	A14	TDO-S3	M3	GND	P2	VDDO-1

Não disponível
Dispositivos IOs
Porta de dados
Conectores Pmods
USB

Figura 19 – Definição dos pinos

4.2.2. Configuração

Após a placa estar ligada, é necessário que o FPGA seja reconfigurado. Durante a configuração, um arquivo de extensão “.bit” será transferido para dentro das células de memória do FPGA, definindo as funções lógicas e o circuito de interconexões(roteamento). Como esse arquivo é gerado, será oportunamente explorado, devendo-se apenas esclarecer que o mesmo é produto da síntese feita pela ferramenta da Xilinx Inc., ISE Design Suite.

O programa Adept, desenvolvido pela fabricante da placa, pode ser utilizado para configurar o dispositivo, a partir de qualquer sistema operacional no qual reside o arquivo ".bit". O Adept utilizará o cabo USB para transferir o arquivo (via porta de programação JTAG). Caso a placa esteja no modo JUMPER (JP3), o arquivo .bit será programado em

uma memória não volátil, chamada de "plataforma FLASH", permitindo o carregamento das configurações sempre que a placa for inicializada. A configuração para as duas formas de programação é representada pela figura 20 [10].

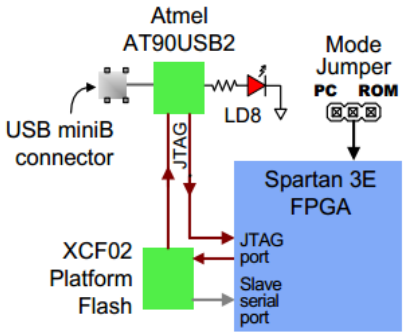

Figura 20 – Modos de programação

4.2.3. Osciladores

Como foi mencionado, a placa possui um oscilador padrão de 50Mhz e outro configurável. Ao se alterar a posição do jumper JP4 o usuário poderá utilizar 25Mhz, 50Mhz ou 100Mhz. Essa alteração deverá ser feita sobre o circuito impresso da placa. Assim, após a alteração, ambos os relógios estarão disponíveis na entrada do FPGA. A placa ainda possui no encaixe IC6, a possibilidade de uso de um cristal oscilador para aplicações sobre a porta VGA (Video Graphics Array), permitindo que esse oscilador seja de 25Mhz ou 50Mhz. A configuração pode ser representada pela figura 21 [10].

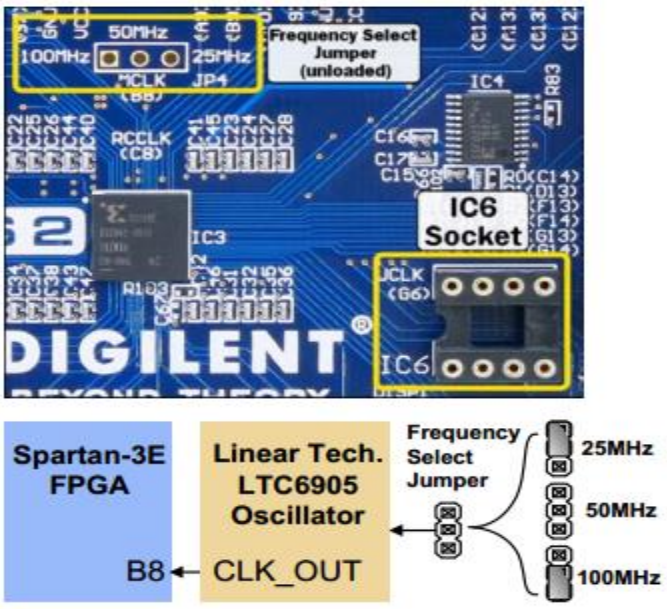

Figura 21 – Opções de Osciladores

4.2.4. Endereçamento de entrada e saída

O circuito que representa chaves, botões, displays e Leds é representado pela figura 22. Já a figura 23, mostra o circuito a ser utilizado em projetos que façam uso de módulos de expansão, como no uso de conversores AD.

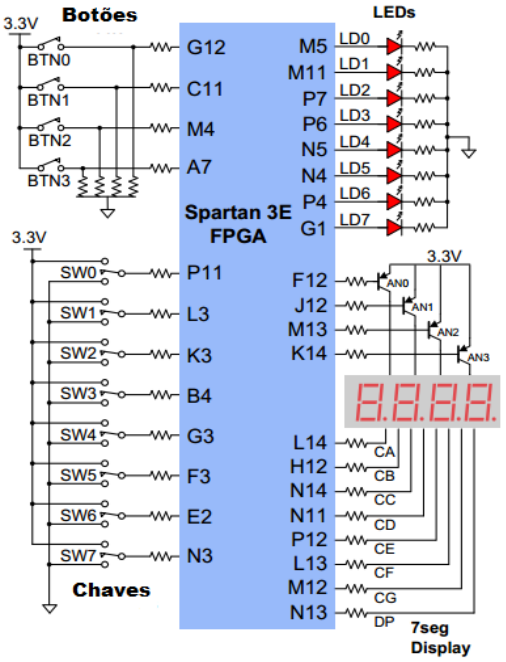

Figura 22 – Circuito IOs

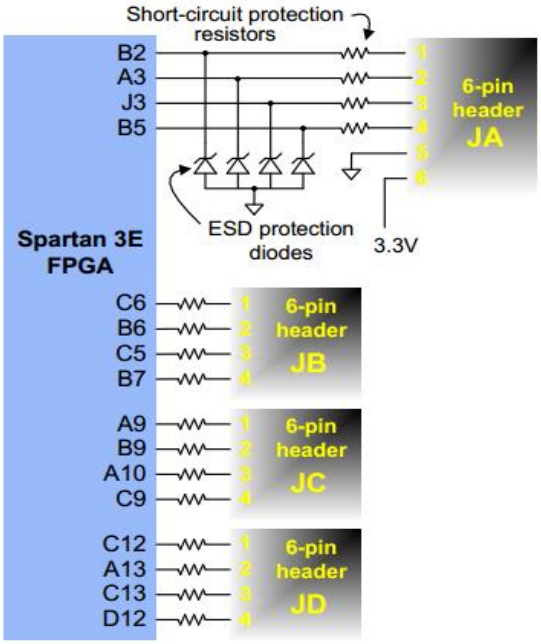

Figura 23 – Circuito Pmods

5. VHDL

Faz-se necessário para o prosseguimento do trabalho o entendimento da linguagem utilizada para descrever os circuitos que serão implementados no hardware estudado. Para tanto, é importante salientar que existem no mercado duas principais linguagens, o VHDL e o VERILOG. Esta útilma possui uma linguagem bem próxima ao C, levando por vezes, por parte do desenvolvedor, a perda do real funcionamento do circuito. Aquela, por sua vez, preocupa-se com as ligações entre os elementos e os sinais gerados, possuindo inclusive a arquitetura comportamental como aplicação. Assim, esse trabalho desenvolverá a descrição de seus hardwares em VHDL.

5.1. Aspectos gerais da linguagem

A linguagem VHDL suporta projetos com múltiplos níveis de hierarquia, a descrição pode consistir na interligação de outras descrições menores, a um código que representa o comportamento esperado do circuito. Esses estilos são denominados estrutural e comportamental, e podem ser mesclados em uma mesma descrição [11]. Com exceção de regiões específicas no código, todos os comandos são executados concorrentemente. Isto significa que a ordem na apresentação dos comandos é irrelevante para o comportamento da descrição. A ocorrência de um evento em um sinal leva à execução de todos os comandos sensíveis àquele sinal, da mesma forma que, em um circuito, a mudança de um valor em um determinado nó afeta todas as entradas ligadas a esse ponto do circuito [11].

Na linguagem *VHDL* não é feita nenhuma distinção entre comandos empregando caracteres maiúsculos ou minúsculos, como ocorre, por exemplo, na linguagem C. Os comentários se iniciam após dois hifens seguidos, "-", e terminam no final da linha. Devido a sua potencialidade, a linguagem *VHDL* é complexa, e muitas vezes de difícil entendimento, dadas as inúmeras opções para modelar o comportamento de um mesmo circuito. Entretanto, o entendimento de um pequeno número de comandos, suficiente para o modelamento de estruturas medianamente complexas, pode ser rapidamente atingido [12]. Para ilustrar o descrito acima, considere o exemplo da figura abaixo, que apresentará uma estrutura lógica e sua descrição em VHDL.

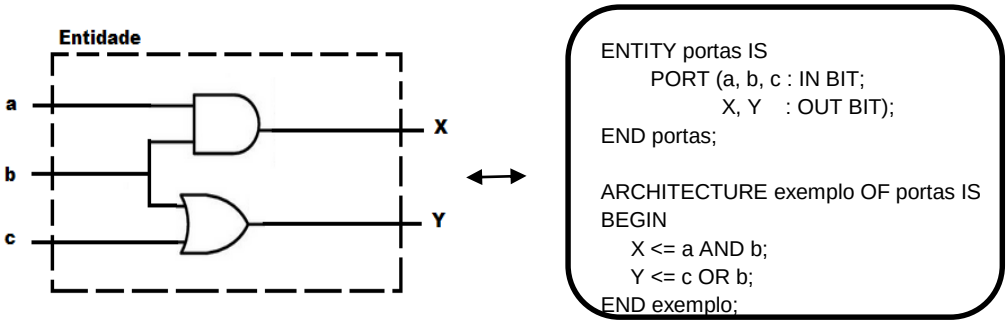

Figura 24 – Circuito e código VHDL de portas lógicas

5.2. Síntese do circuito

O VHDL inicialmente não foi concebido para sintetizar circuitos digitais, portanto, podem existir construções definidas que não poderão ser sintetizadas, por não serem suportadas pela ferramenta. Esses problemas surgem da impossibilidade quanto à precisão requerida ou relativo a falha de detalhamento do circuito pretendido. Na prática, pode-se, por exemplo, propor um elemento de memória, sensível tanto a subida quanto a descida do relógio, e que não geraria problemas na simulação. Porém, a ferramenta que sintetizará o circuito, pode não o fazê-lo, por falta de um elemento real que possibilite sua implementação. Buscando diminuir os problemas provenientes da linguagem de descrição de circuitos, seguiremos três passos em um projeto, como é mostrado pela figura 25 [11].

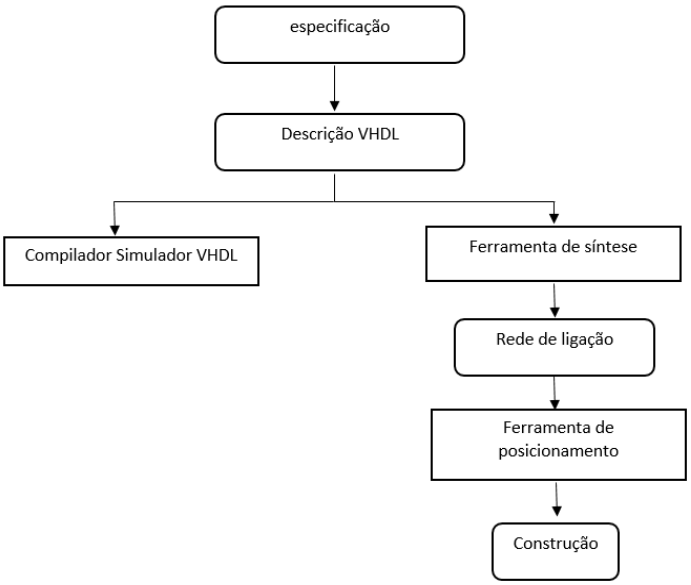

Figura 25 – Etapas de um projeto

Inicialmente, a partir da especificação de um projeto, obtêm-se uma descrição VHDL. Essa, em um segundo momento será submetida a um simulador para verificação da correspondência entre a especificação e o código. A mesma descrição será interpretada por uma ferramenta de síntese, que inferirá as estruturas necessárias para um circuito que corresponda à descrição. O resultado dessa etapa será um arquivo contendo a rede de ligações de elementos básicos disponíveis no hardware a ser implementado. Esse arquivo, também será a base de dados para a ferramenta que fará o posicionamento e a interligação dos componentes, *place and route* [11]. Assim, a saída dessa ferramenta de posicionamento, será um arquivo que conterá os dados necessários para a confecção do dispositivo, que no capítulo anterior foi apresentado como um arquivo “bit”.

5.3. Entidade e Arquitetura

Um dispositivo digital pode ser definido na linguagem VHDL, por uma entidade e uma arquitetura. A entidade, é declarada usando a palavra reservada *entity*, define a interface do dispositivo com o exterior, listando os sinais de entrada, saída e bidirecional do dispositivo. Já a arquitetura, declarada pela palavra reservada *architecture* descreve o comportamento lógico interno do dispositivo [5]. Ao ser compilado, o código é varrido pela ferramenta buscando as entidades de maior nível, para que a partir de sua arquitetura seja sintetizado o circuito. A figura 26 ilustra a entidade como o bloco do circuito com as entradas e saídas e a arquitetura como o comportamento interior do bloco:

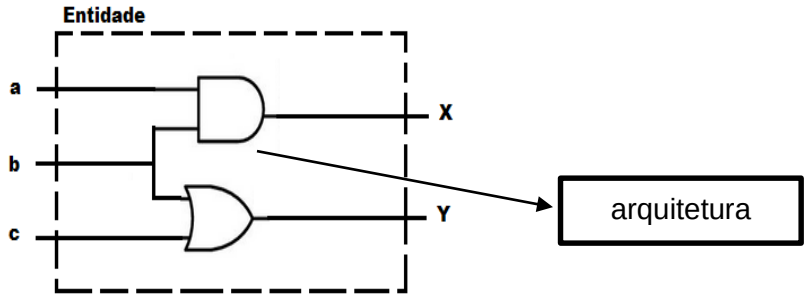

Figura 26 – exemplo

É importante ressaltar que o uso do termo “maior nível”, refere-se a entidade que não possui dependência em relação as outras. Assim, caso houvesse a necessidade de produzir um CI de várias entidades portas (exemplo anterior), haveria uma nova entidade, por exemplo *entity* CI, que reuniria as outras entidades (que agora possuem dependência a entidade de maior nível).

5.4. Classe de objetos, Tipos e Operadores

Objetos são elementos que contêm um valor armazenado, na linguagem VHDL existem quatros classes definidas:

- Constante ou “CONSTANT”, que possui um valor estático;
- Variável ou “VARIABLE”, seu valor pode ser alterado no decorrer do código, e é utilizada em regiões de código sequencial;
- Sinal ou “SIGNAL”, seu valor pode ser alterado no decorrer do código, e é utilizada em regiões de código concorrente e sequencial;
- Arquivo ou “FILE”, são associados à criação de arquivos.

Conforme o manual de referência da linguagem, um tipo é caracterizado por um conjunto de valores e um conjunto de operações [13]. Os objetos “CONSTANT”, “VARIABLE” e “SIGNAL” devem ser declarados segundo um tipo definido, para que fiquem estabelecidos os valores dos objetos podem assumir e as operações que podem ser realizadas. Pode-se verificar alguns tipos e suas operações na tabela abaixo.

Tabela 2 – Tipos e suas operações

TIPO	SUBTIPO	VALOR	EXEMPLOS
BIT	escalar	Um, zero	1, 0
BOOLEAN	escalar	Verdadeiro ou Falso	TRUE, FALSE
INTEGER	escalar	$-2^{31} - 1 \leq x \leq 2^{31} - 1$	123, 8#173#
BIT_VECTOR	composto	1, 0	“1011”, B “10_11”
STRING	composto	Tipo character	“texto”

Os operadores definidos são divididos em classes que estabelecem a precedência na execução das operações. Na tabela abaixo são apresentadas em ordem crescente de precedência, as classes de operadores. Os operadores de uma mesma classe possui o mesmo nível de precedência. Parêntesis podem ser empregados para definir a ordem das operações[13].

Tabela 3 – operadores definidos por classe

PRECEDÊNCIA	CLASSE	OPERADORES
Menor	lógicos	Nor, or ,and
*	relacionais	=, /=
*	deslocamento	sll , srl
*	adição	+, - , &
*	sinal	+ ou -
Maior	multiplicação	*, /

6. Implementação do Filtro

Como foi exposto no item 5.2, haverá um ciclo a ser seguido a fim de sintetizar um circuito a partir de um código. Portanto, para que haja o fluxo completo de um projeto, há que se considerar um quarto passo a ser somado aos anteriores, o debug de sistema[14]. Essa etapa do fluxo poderá ser realizada pela ferramenta de desenvolvimento, via JTAG (Joint Test Action Group), ou pelo próprio desenvolvedor ao testar o circuito implementado. As fases do fluxo de um projeto completo em FPGA são apresentados na figura abaixo [14].

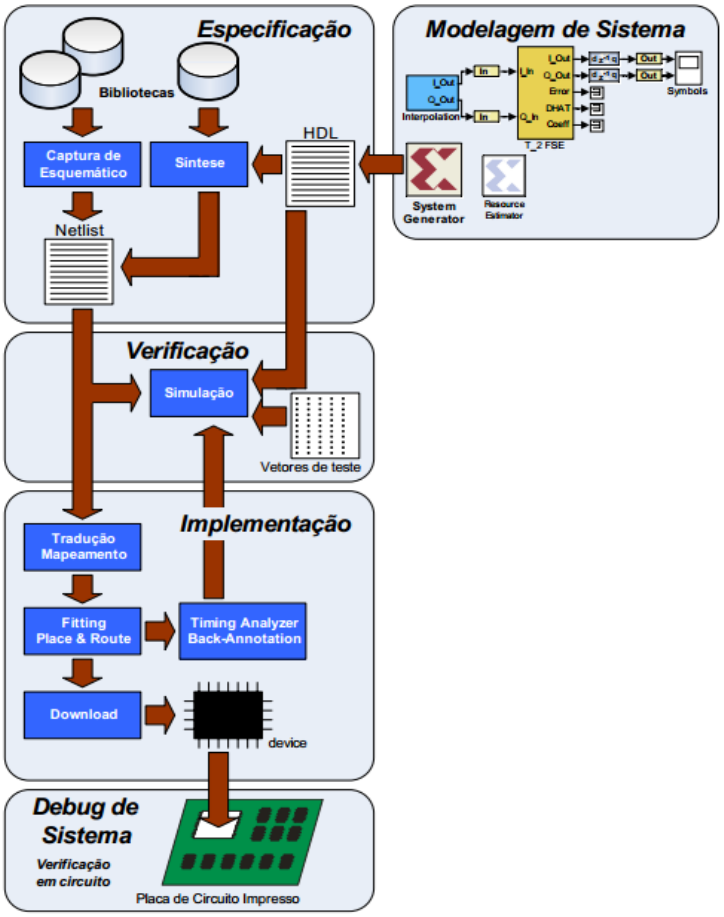

Figura 27 – Fluxo de projeto de um FPGA

6.1. Especificações dos filtros

Serão descritas as especificações de dois filtros IIR, que serão implementados nos itens que seguem.

6.1.1. Chebyshev

- Passa-baixa: de 4ª ordem com frequência de 3db igual a 1Khz e ripple de 1db.

6.1.2. Butterworth

- Passa-banda: de 6ª ordem com frequências de 3db valendo 1,5Khz e 2,5Khz.

6.2. Obtenção do filtro quantizado

Para a obtenção de um filtro quantizado (discreto), iremos usar a ferramenta computacional Matlab®, na qual serão especificados diversos parâmetros para que os coeficientes do filtro possam ser gerados. Os parâmetros usados serão os seguintes:

- Arithmetic: ajustará a aritmética do filtro. Será utilizada a aritmética de ponto fixo;
- InputWordLength: ajusta o tamanho da palavra de entrada do filtro. Como está sendo usado um conversor A/D de 12 bits será usado 12;
- OutputWordLength: ajusta o tamanho da palavra de saída do filtro. Como o conversor D/A é de 12 bits o valor do parâmetro é 12;
- OutputMode: ajusta o modo como o filtro ajusta o fatoe de escala do dado filtrado de saída. Quando ajustado para a opção SpecifyPrecision, permite que a parte fracionária da palavra de saída seja ajustada;
- OutputFracLength: ajusta o tamanho da parte fracionária da palavra de saída;
- CoeffWordLength: ajusta o tamanho da palavra usada para representar os coeficientes do filtro;
- AccumWordLength: define o tamanho da palavra usada no buffer do acumulador;

- StateWordLength: ajusta o tamanho da palavra de estado, que é maneira que são armazenadas as propriedades de ponto fixo do filtro;
- CastBeforeSum: Determina como o filtro lida com os dados de entrada de somadores internos ao filtro, principalmente com relação a valores armazenados de outras somas;
- RoundMode: ajusta a forma de arredondamento utilizada na quantização;
- OverflowMode: Especifica o procedimento a ser tomado em caso de alguma operação gerar overflow. Pode ser usado saturação ou modo “wrap” em que são utilizados apenas os bits menos significativos

Eles controlam o fluxo através da estrutura de implementação da forma direta II pelo Matlab® como pode ser verificado através da figura a seguir:

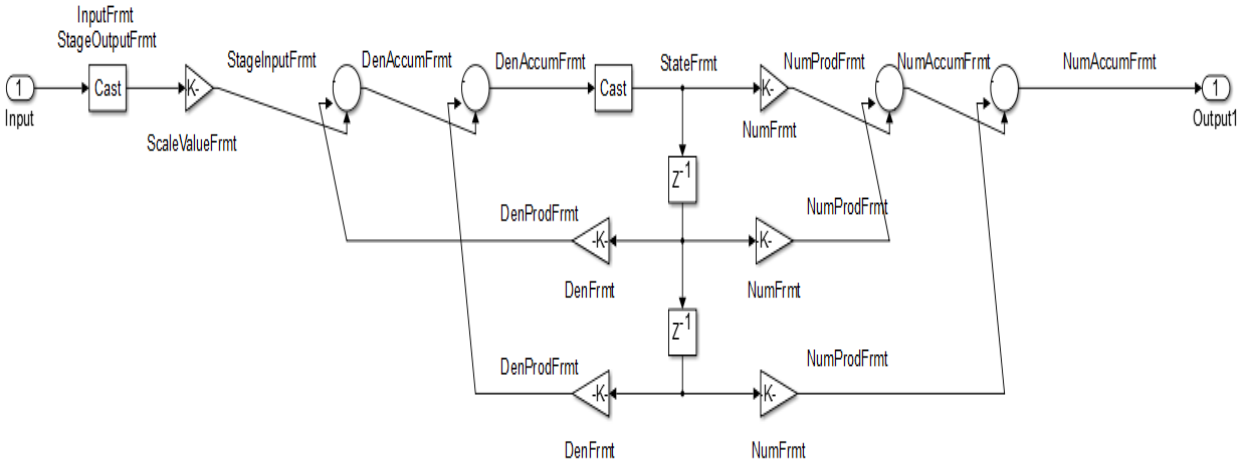

Figura 28 – Estrutura Direta II do Matlab®

Na figura acima o fluxo do sinal é marcado por rótulos com os locais em que os parâmetros acima citados podem configurar as propriedades que determinam a maneira que são efetuados as operações aritméticas.

Abaixo será apresentado o código implementado para o filtro butterworth passa-banda, os códigos dos outros filtros serão disponibilizados no CD em anexo.

```
%% Criando o filtro
close all;
clear all;
Fs = 50e3; % freq de amostragem
% Design do filtro passa-faixa
% filtro de ordem 6
Hd = fdesign.bandpass('N,F3dB1,F3dB2',6,1500,2500,Fs);
Hd = design(Hd,'butter');
%Conversão para estrutura direta
Hd = convert(Hd, 'df2sos');
```

```
% Examinando a resposta
fvtool(Hd, 'Fs', Fs, 'FrequencyScale', 'log');
%% Criando o filtro quantizado
Hd.arithmetic = 'fixed';
Hd.InputWordLength = 12;
Hd.InputFracLength = 11;
Hd.OutputWordLength = 12;
Hd.OutputMode = 'SpecifyPrecision';
Hd.OutputFracLength = 10;
Hd.CoeffWordLength = 12;
Hd.AccumWordLength = 40;
HD.StateWordLength = 40;
Hd.CastBeforeSum = false;
Hd.RoundMode = 'nearest';
Hd.OverflowMode = 'saturate';

%fvtool(Hd, 'Fs', Fs, 'FrequencyScale', 'log');
%% Examinando os valores
scales = Hd.scalevalues .* 2^Hd.InputFracLength
scale(Hd,'Linf');
scales = Hd.scalevalues

%% Gerando o HDL do filtro quantizado
% Criando o local para armazenar o arquivo
workingdir = 'hdl_work';
generatehdl(Hd, 'Name', 'hdlbutter', 'TargetLanguage', 'VHDL', ...
'TargetDirectory', workingdir);
edit(fullfile(workingdir, 'hdlbutter.vhd'));

%% Gerando o teste bench para o estímulo do sinal
userstim = [];
for n = [ ...
1400,1500,1600,1700,1800,1900,2200,2300,2400,2500,2600,2700]
userstim =[userstim,sin(2*pi*n/Fs*(0:Fs/n))];
end
generatetb(Hd, 'VHDL', 'TestBenchName', 'hdlbutter_tb',...
'TestBenchStimulus', [],...
'TestBenchUserStimulus', userstim,...
'TargetDirectory', workingdir);
edit(fullfile(workingdir, 'hdlbutter_tb.vhd'));

%% Simulação
% Após a criação do VHDL e VHDL test bench
xrange = (0:length(userstim) - 1);
y = filter(Hd, userstim);
plot(xrange, y);
title('HDL BUTTER');
xlabel('Amostras#');
```

Após a obtenção do filtro quantizado, usaremos um codificador HDL(ferramenta do Matlab®), para gerar um arquivo que descreva o filtro em linguagem VHDL. Abaixo apresentamos um trecho do código gerado por esse codificador (entidade).

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;

ENTITY hdlcheby IS
   PORT( clk                             :   IN    std_logic;
         clk_enable                      :   IN    std_logic;
         reset                           :   IN    std_logic;
         filter_in                       :   IN    std_logic_vector(11 DOWNTO 0); -- sfix12_En11
         filter_out                      :   OUT   std_logic_vector(11 DOWNTO 0)  -- sfix12_En11
         );

END hdlcheby;
```

6.3. Implementação da conversão AD/DA

Para recepção do sinal a ser filtrado e a transmissão do sinal egresso do filtro, será necessário a implementação da interface dos PMODs e o FPGA. Para lograr êxito faz-se necessário o conhecimento da folha de dados dos dispositivos. A figura a seguir mostra a máquina de estados que será implementada.

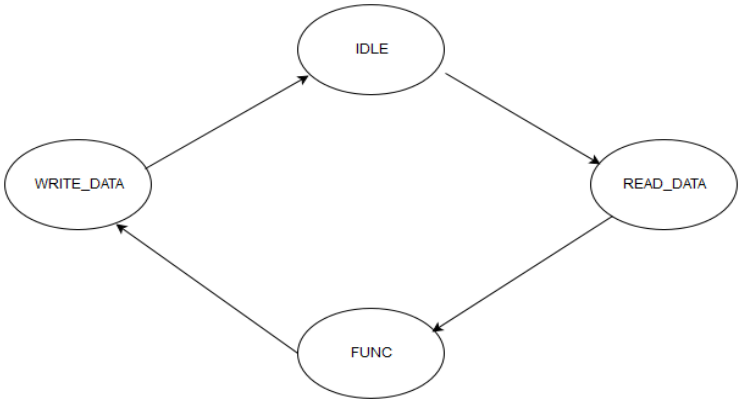

Figura 29 – Máquina de estado do filtro digital

O estado IDLE é um estado de espera. O estado READ_DATA é o estado em que ocorre a conversão A/D. O estado FUNC é o estado em que o ocorre o cálculo da equação de diferenças do filtro e o estado WRITE_DATA é o estado em que ocorre a conversão D/A.

O processo de conversão A/D se dá de acordo com diagrama de tempo mostrado a seguir retirado da folha de dados do dispositivo.

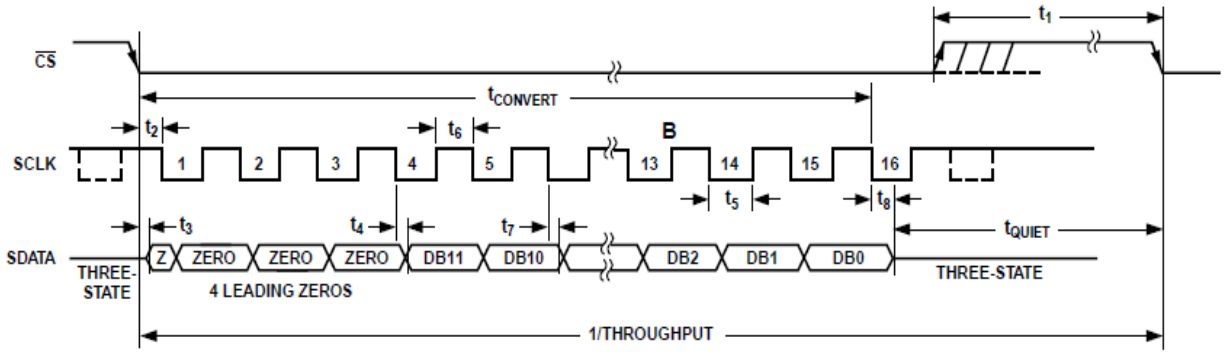

Figura 30 – Diagrama de tempo A/D

Pelo diagrama percebe-se que o início do processo de conversão se inicia quando o sinal $\overline{CS}$ vai a nível baixo, sendo então transmitidos quatro zeros seguidos dos doze bits que descrevem o nível de tensão adquirido pelo circuito. Tendo terminado o processo de conversão é iniciado o próximo estado.

O estado FUNC é o estado que irá se relacionar com o código VHDL gerado pela ferramenta de HDL Coder do Matlab®.

O processo de conversão D/A segue o diagrama mostrado a seguir:

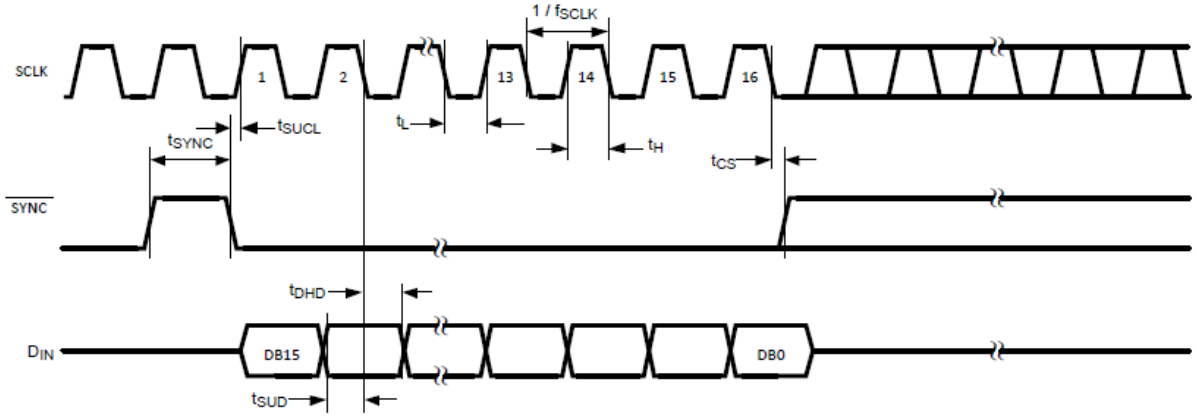

Figura 31 – Diagrama de tempo D/A

O processo é similar ao de conversão D/A. O sinal $\overline{SYNC}$ em nível alto dá início ao processo de conversão com os dois primeiros bits sendo zero e os próximos dois sendo bits de configuração. Os próximos doze bits correspondem aos bits que serão lidos do filtro para a transformação do sinal analógico.

Desta forma, disponibilizaremos no CD em anexo o código da implementação dos conversores AD e DA em linguagem VHDL, bem como o filtro e o circuito RTL gerado.

7. Testes com a implementação

O principal objetivo na parte da simulação é validar o funcionamento das estruturas lógicas do filtro, procurando as falhas mais latentes. Também buscou-se analisar os efeitos que o filtro trouxe ao sinal. Essa seção será dividida em cenários de teste, explicando os objetivos e resultados de cada um.

O sinal de entrada é gerado por gerador de sinais com 1 Vpp e offset de 1.0 V, de maneira que ele se encontra dentro do intervalo aceitado pelo conversor A/D que é de 0 a 3.3 V. Dessa forma o código do filtro tem que deslocar o sinal para em torno de 0V para realizar as operações e depois retorna-lo para em torno de 1,5V.

7.1. Teste Butterworth

O objetivo desse teste é validar o funcionamento do filtro passa-banda de butterworth. Foi usado a forma direta II e os seguintes parâmetros de quantização:

```
Hd.arithmetic = 'fixed';
Hd.InputWordLength = 12;
Hd.InputFracLength =11;
Hd.OutputWordLength = 12;
Hd.OutputMode = 'SpecifyPrecision';
Hd.OutputFracLength =10;
Hd.CoeffWordLength = 12;
Hd.AccumWordLength = 40;
Hd.StateWordLength = 40;
Hd.CastBeforeSum = false;
Hd.RoundMode = 'nearest';
Hd.OverflowMode = 'saturate';
```

Esse conjunto de parâmetros gera a seguinte resposta em frequência:

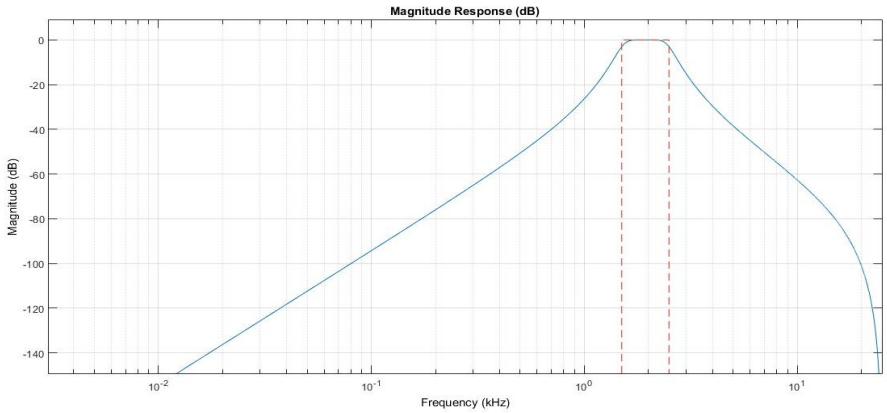

Figura 32 – Resposta em frequência passa-faixa

As figuras a seguir ilustram o funcionamento do filtro. Foram tiradas imagens da saída do filtro nas frequências de corte inferior (1500 Hz) e superior (2500 Hz), além de imagens nas bandas de rejeição.

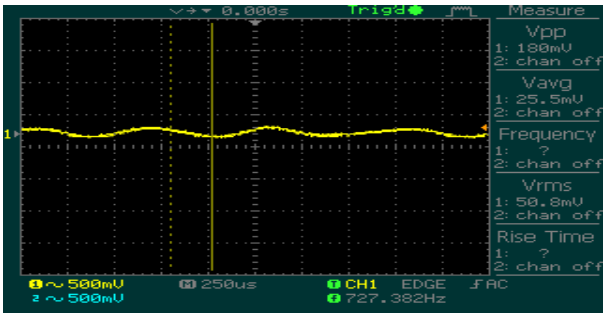

Figura 33 – Banda de rejeição inferior

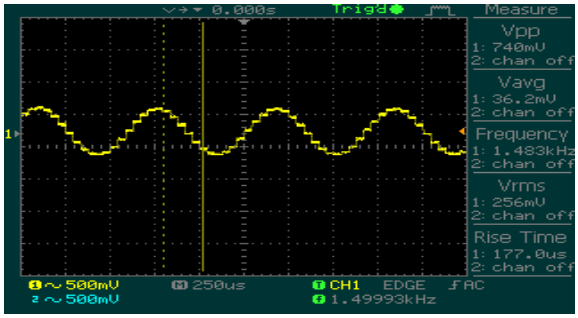

Figura 34 – Saída do filtro a 1.5 Khz

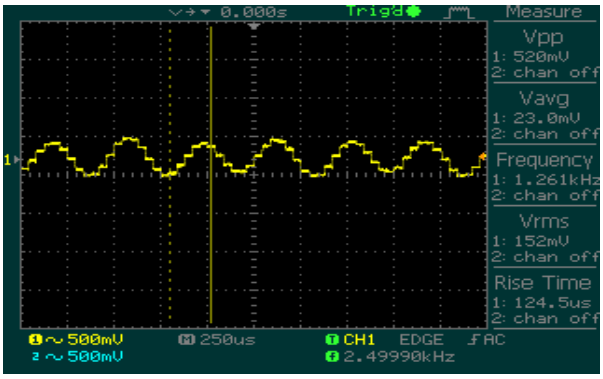

Figura 35 – Saída do filtro a 2.5 Khz

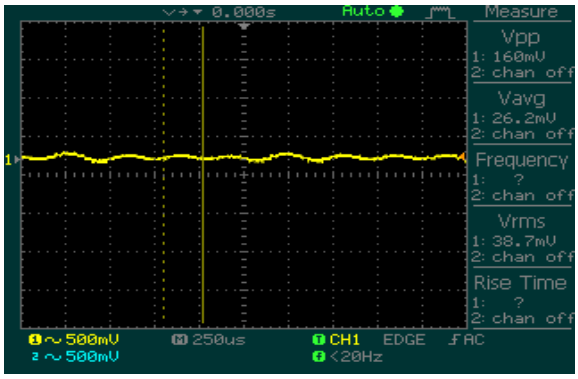

Figura 36 – banda de rejeição superior

Como podemos ver, nas frequências de corte as amplitudes são próximas do sinal da metade da amplitude do sinal na banda de passagem. Além disso, percebe-se uma menor quantidade de níveis de quantização no sinal do filtro passa banda.

As figuras a seguir mostram os polos e zeros dos filtros de referência e quantizado. Como foi dito no capítulo 2, o processo de quantização altera a localização dos polos e zeros do filtro. Apesar de na figura 31, eles aparentarem serem coincidentes, a figura 32 mostra que eles foram alterados. Estas alterações também mudam a resposta em frequência do filtro e podem causar distorções sendo mais perigosa quanto mais próxima do ciclo de raio unitário se encontram os polos, pois a alteração da quantização pode levar esse polo fora do ciclo de raio unitário.

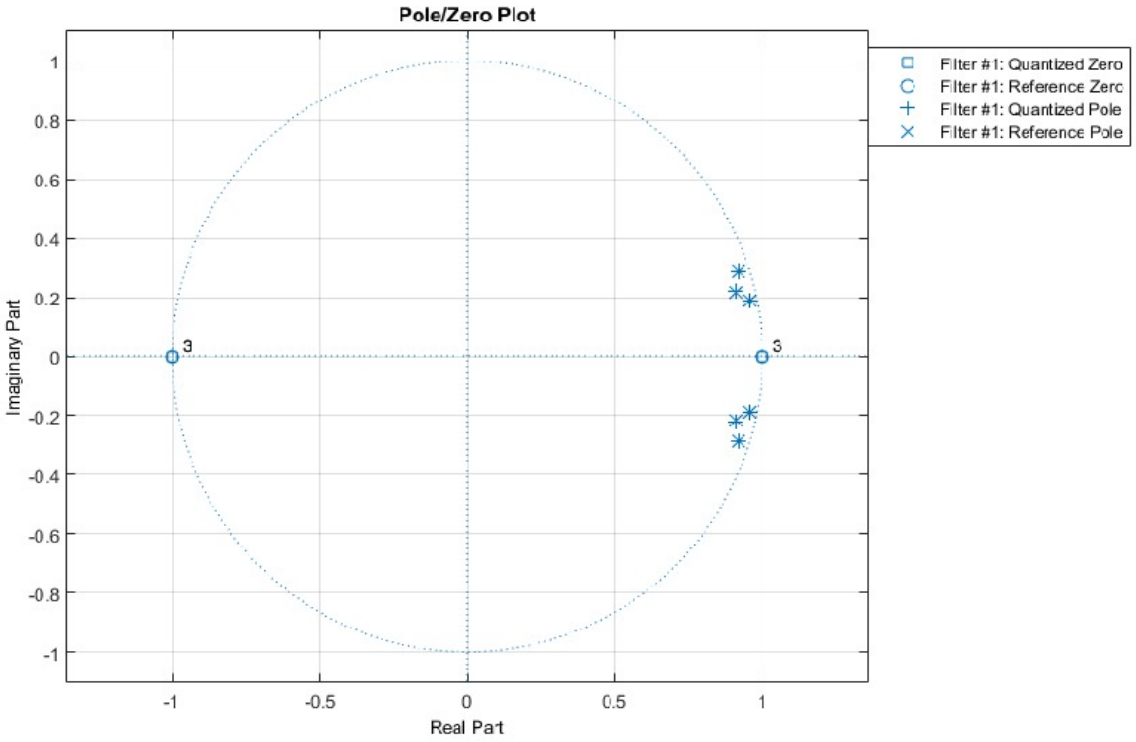

Figura 37 – Diagrama de polos e zeros do filtro passa faixa

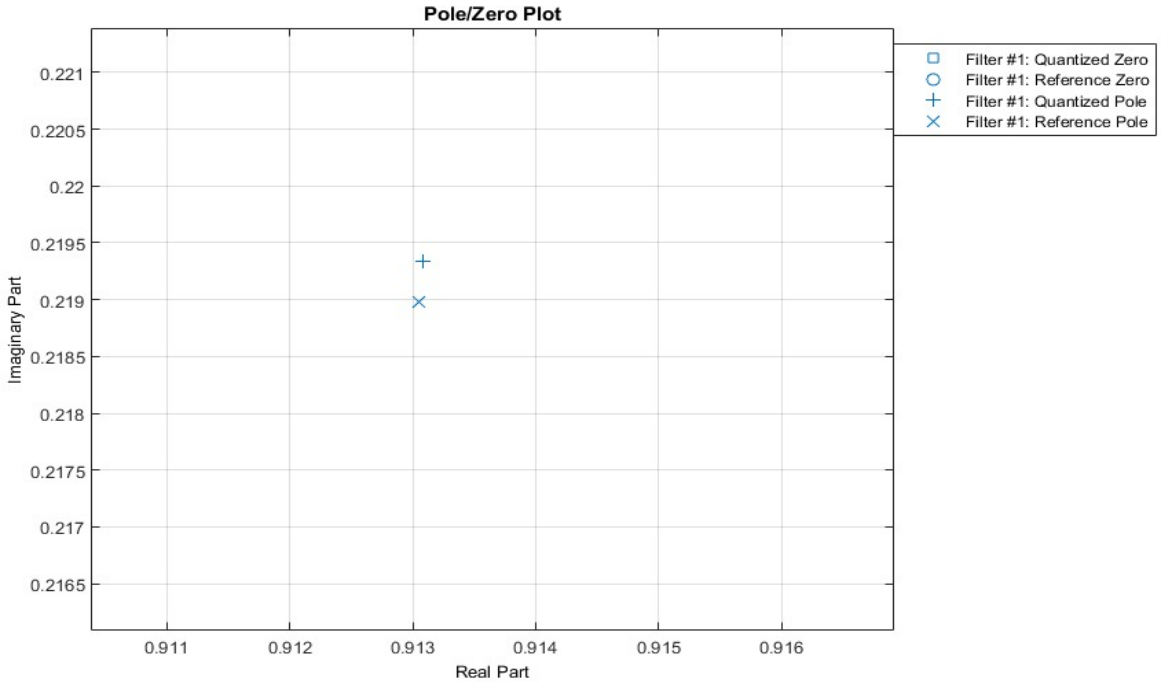

Figura 38 – Deslocamento do polo do filtro passa faixa

Foi medida a amplitude em função da frequência para a faixa de frequência entre 100 Hz e 3.2 kHz. Os dados obtidos foram plotados e interpolados gerando a seguinte figura:

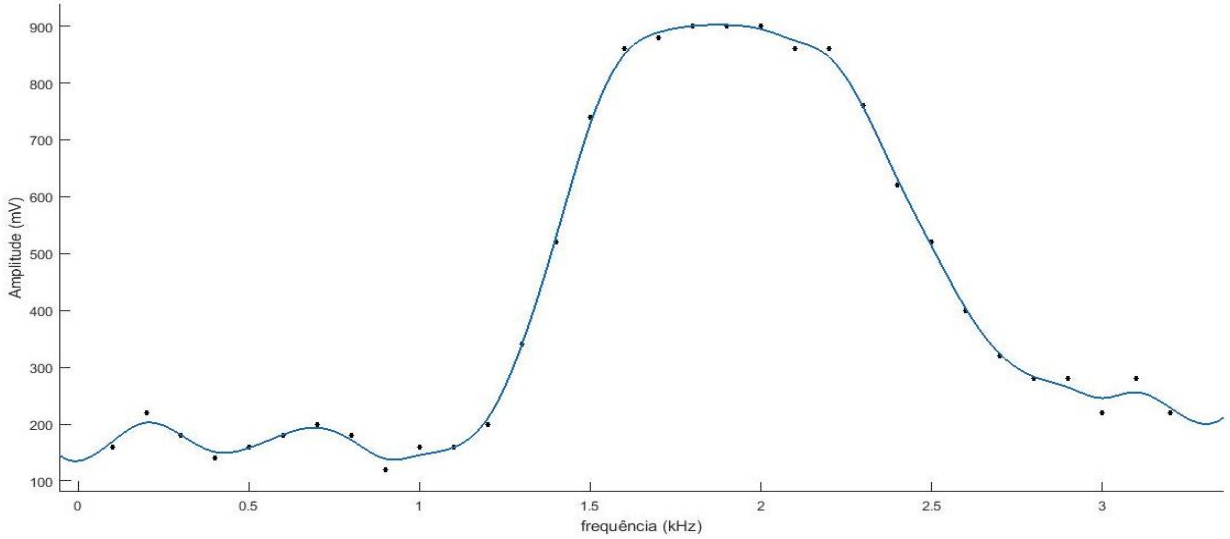

Figura 39 – Tensão pico a pico em função da frequência

A curva do ganho em dB pela frequência é a seguinte:

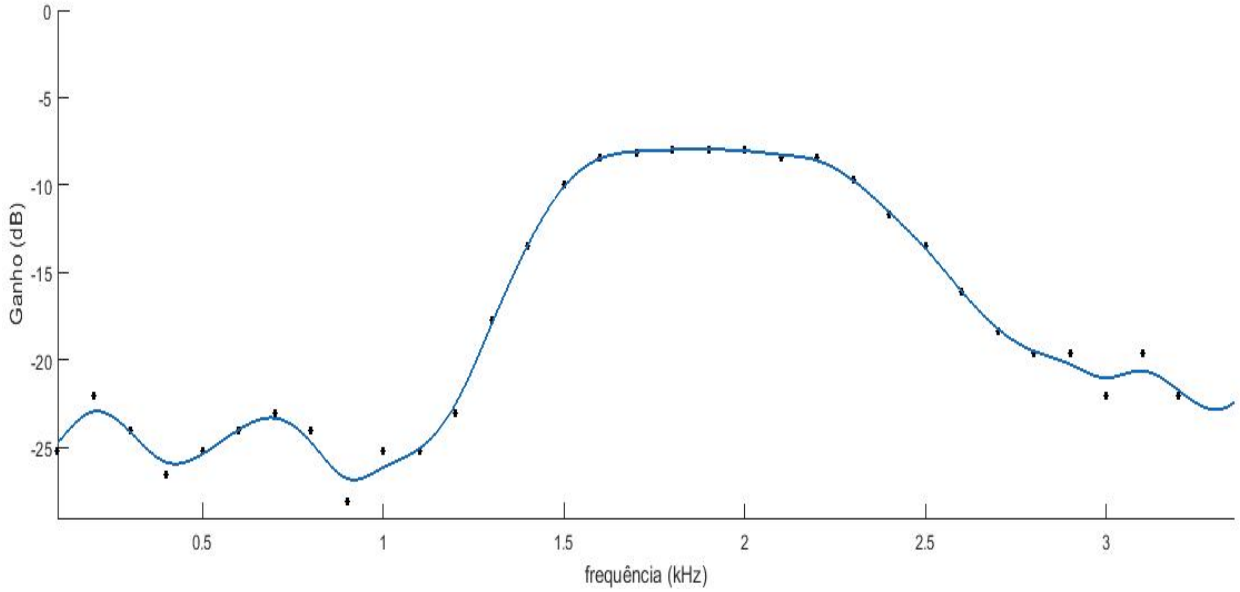

Figura 40 – Ganho em dB em função da frequência

7.2. Teste Chebyshev

O objetivo desse teste é validar o funcionamento do filtro passabaixa de Chebyshev. Também foi utilizada a forma direta II e os seguintes parâmetros de quantização para realizar o filtro:

```
Hd.arithmetic = 'fixed';
Hd.InputWordLength = 12;
Hd.InputFracLength =11;
Hd.OutputWordLength = 12;
Hd.OutputMode = 'SpecifyPrecision';
Hd.OutputFracLength =11;
Hd.CoeffWordLength = 12;
Hd.AccumWordLength = 24;
Hd.StateWordLength = 20;
Hd.CastBeforeSum = true;
Hd.RoundMode = 'nearest';
Hd.OverflowMode = 'saturate';
```

Esse conjunto de parâmetros gera a seguinte resposta em frequência:

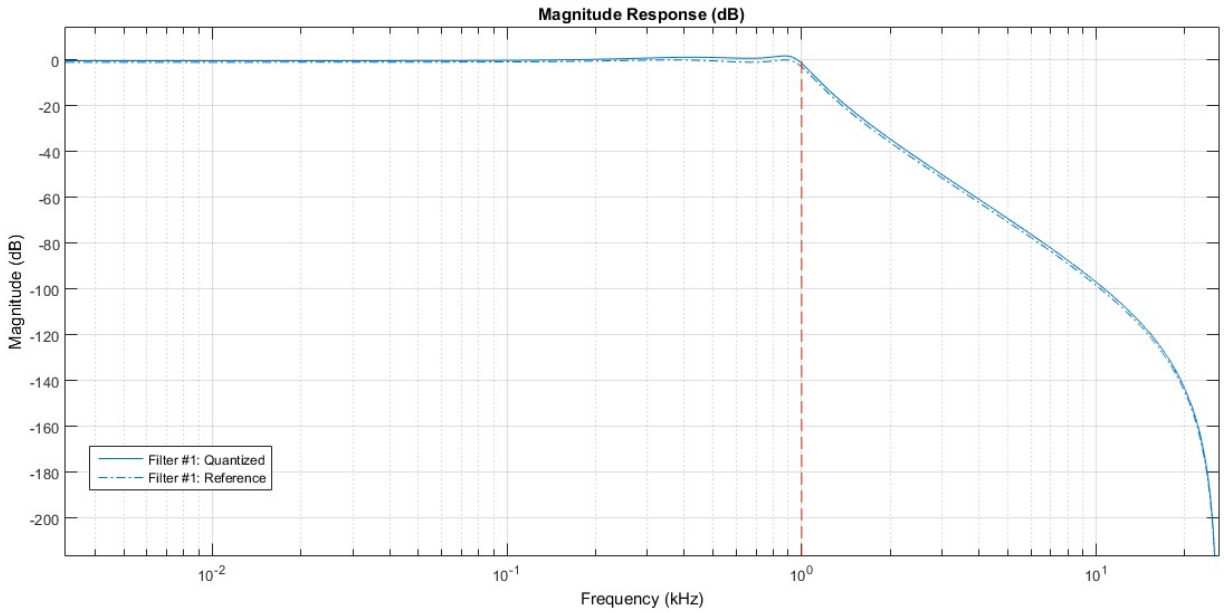

Figura 41 – Resposta em frequência do filtro passa-baixa

As figuras a seguir ilustram o funcionamento do filtro. Foram tirada imagens da saída do filtro, captadas por um osciloscópio digital na banda de passagem (próximo a 500 Hz), na frequência de corte de 3 dB e em frequências da banda de rejeição.

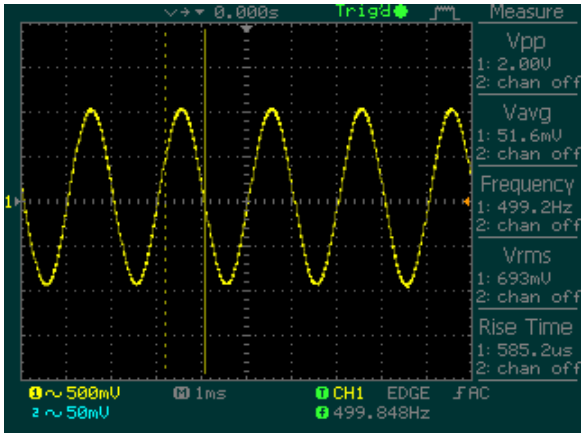

Figura 42 – Saída do filtro a 500hz

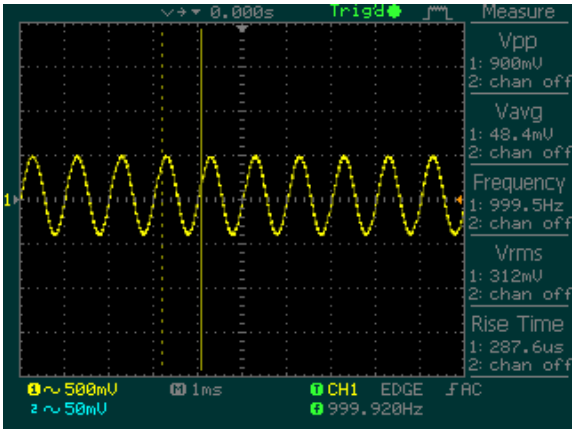

Figura 43 – Saída do filtro a 1Khz

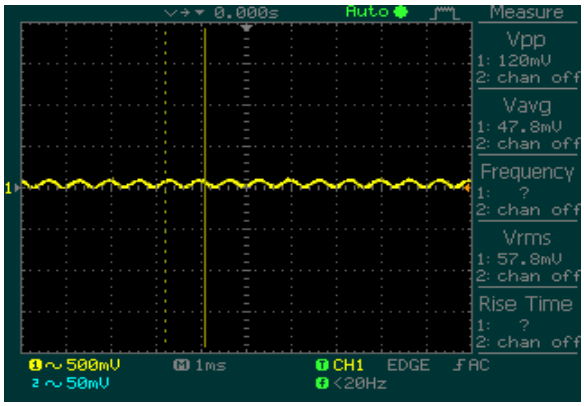

Figura 44 – Saída do filtro a 1.5 KHz

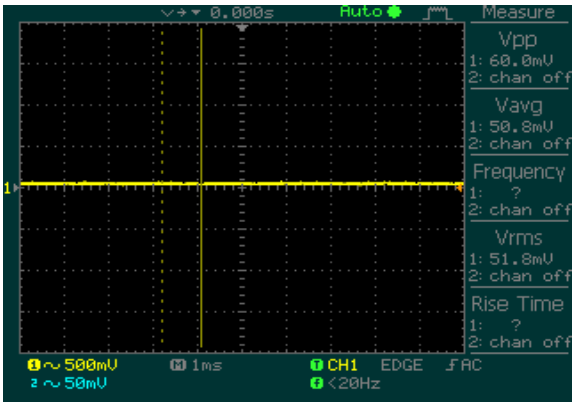

Figura 45 – Saída do filtro a 2.5 KHz

Como pode-se observar nas figuras acima, na frequência de corte da banda de passagem do filtro, o sinal possui aproximadamente a metade da amplitude do sinal de entrada, como era esperado. Em 1.5 KHz encontra-se quase totalmente atenuado e em 2.5KHz totalmente. Como o sinal na banda de rejeição está cortado, o osciloscópio não consegue medir a frequência do sinal.

As figuras a seguir mostram os polos e zeros dos filtros de referência e quantizado. No filtro passa baixa, os polos e zeros do filtro de referência e do filtro quantizado se encontram ainda mais próximos uns dos outros do que no filtro passa faixa, o que é melhor percebido na figura 39. Isso traduz-se no melhor cumprimento das especificações do filtro, com menos distorções.

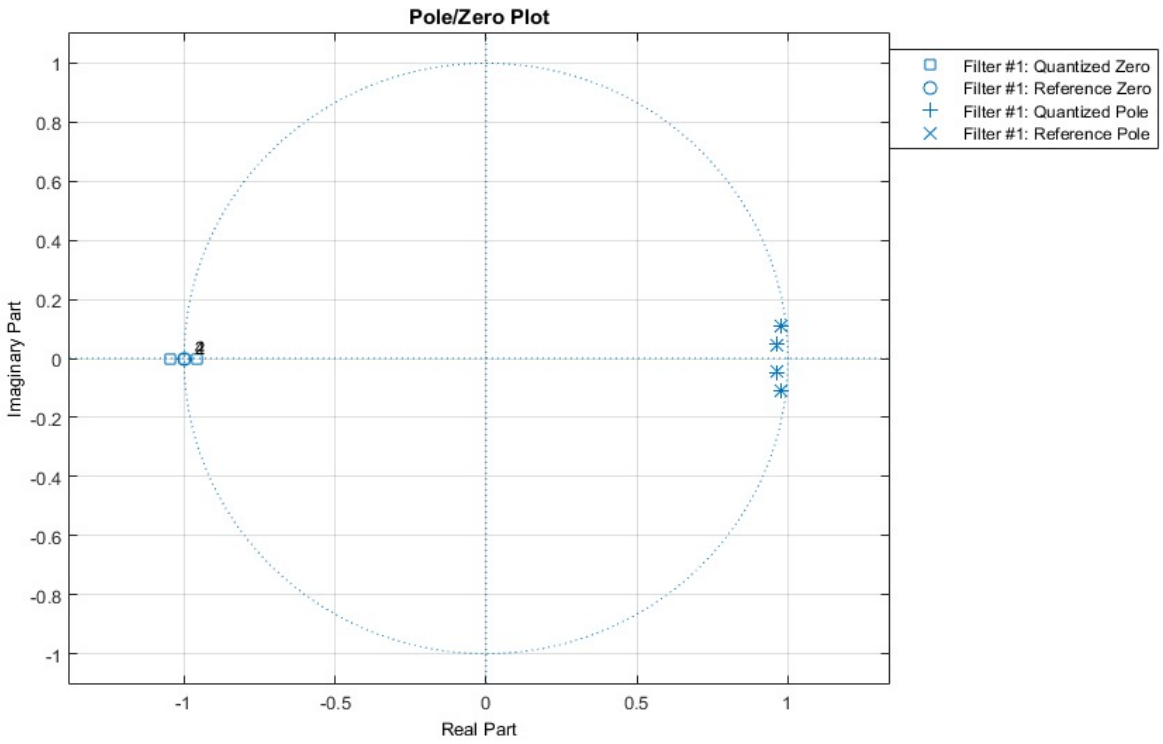

Figura 46 – Saída do filtro a 500hz

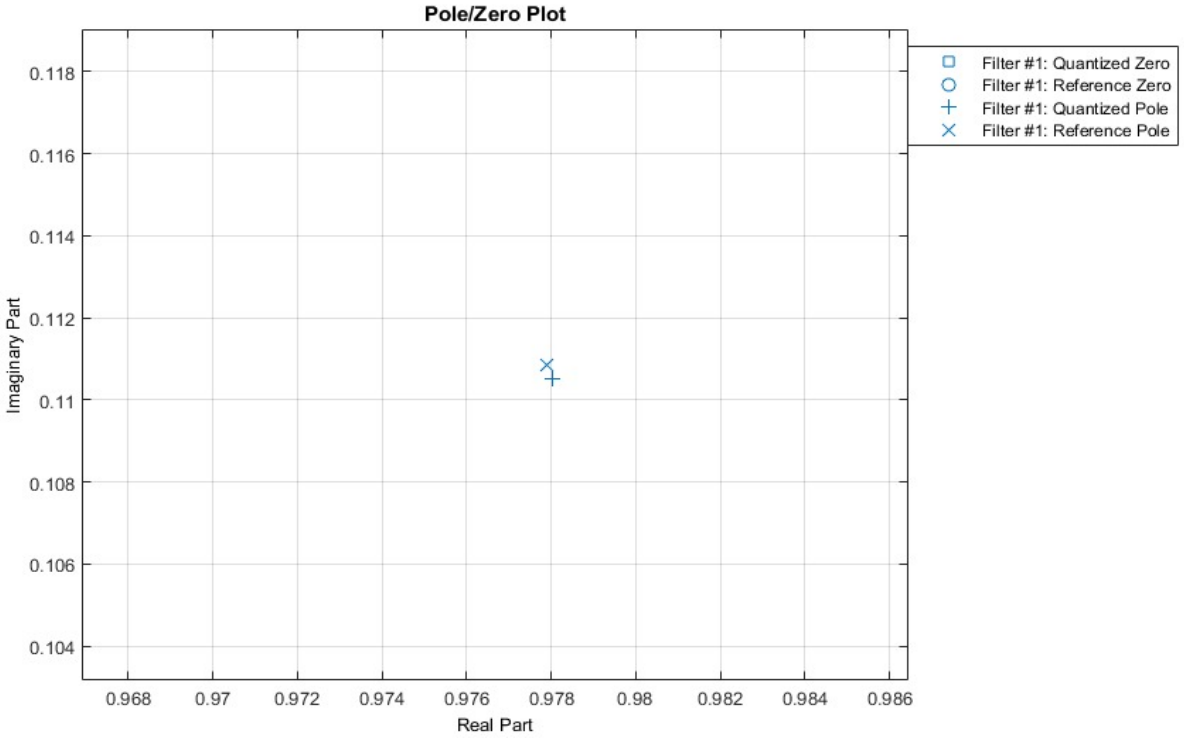

Figura 47 – deslocamento de polo do filtro passa baixa

Foi medida a amplitude em função da frequência para a faixa de frequência entre 100 Hz e 3.2 kHz. Os dados obtidos foram plotados e interpolados gerando a seguinte figura:

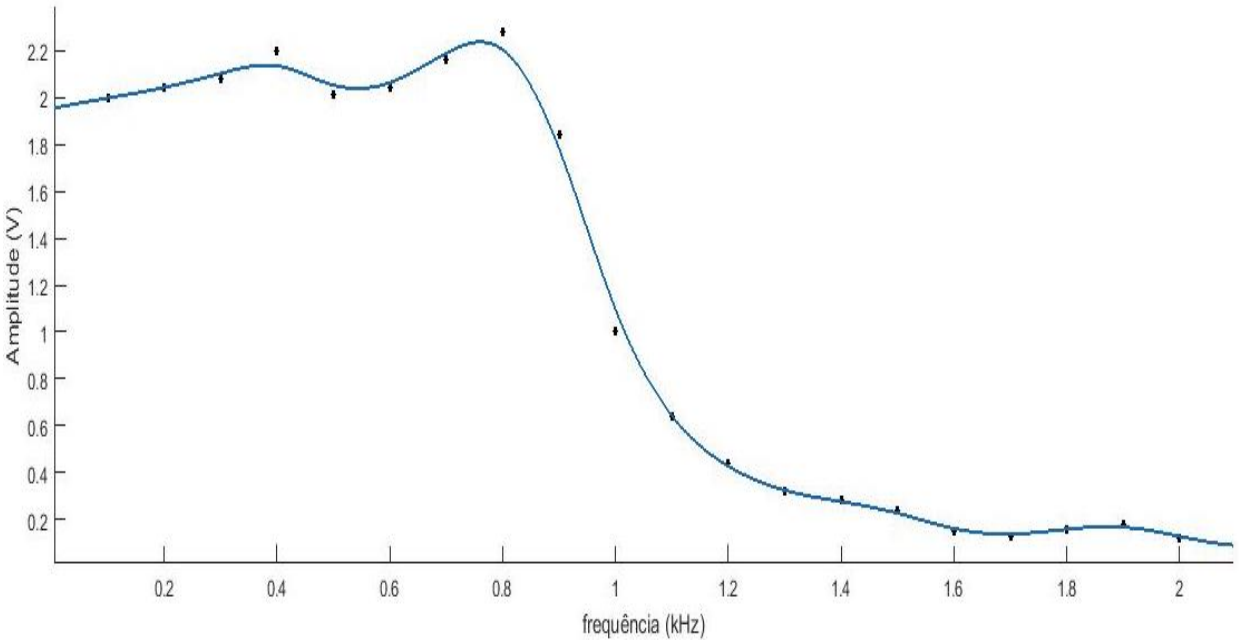

Figura 48 – Tensão pico a pico em função da frequência

A curva do ganho em dB pela frequência é a seguinte:

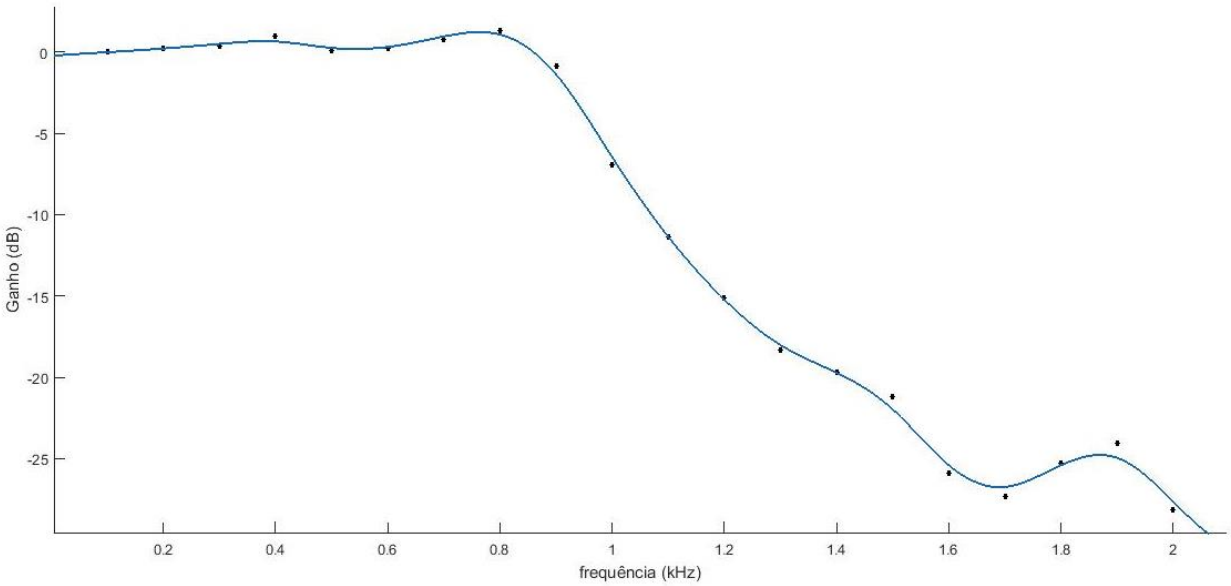

Figura 49 – Ganho em dB em função da frequência

8. Conclusão

Esse trabalho buscou apresentar conceitos de implementação de um controlador linear por FPGA, o que consiste na realização de uma função de transferência racional. Para alcançar esse objetivo buscou-se primeiramente a implementação de filtros digitais uma vez que os filtros também são funções de transferência racionais. No processo de construção do filtro foi necessário aprendizado de conhecimentos de processamento digital de sinais e conhecimento estruturas lógico programáveis e sua implementação em FPGA. Aqui foram abordadas noções desse tipo de tecnologia, apontando como implementar um filtro em um FPGA.

O desenvolvimento desses filtros trouxe a relação de compromisso entre a utilização dos recursos de hardware e a precisão do sinal, pois a medida que se aumenta a ordem do filtro, mais células lógicas eram usadas, e por vezes, não possibilitando sua implementação, em virtude do FPGA extremamente limitado. Os testes mostraram para baixas ordens o filtro atende com tranquilidade as especificações com precisão. Já para ordens mais altas os requisitos são atingidos parcialmente, ocorrendo distorções no sinal de saída e atunação.

9. Próximos passos

Para etapas futuras, com relação a implementação, sugere-se o uso de um FPGA com mais recursos de CLB, o que possibilitaria a implementação de controladores mais robustos.

10. Referências Bibliográficas

1. Dehon, A.,” RECONFIGURABLE ARCHITECTURES FOR GENERAL-PURPOSE COMPUTING ”,Massachussetts Institute of Technology, Ph.D. Thesis, October 1996.

2. Field-programmable gate array. Disponível em: <https://pt.wikipedia.org/wiki/Field-programmable_gate_array>. Acesso em: 4 ago. 2016.

3. Mendonça, A.Zelenovsky, R. Eletrônica Digital - Curso Prático e Exercícios. 2. ed. Rio de Janeiro: MZ, 2007.

4. Moore, A. FPGA for Dummies. Traducao . Hoboken: Wiley, 2014.

5. TAVARES REZENDE, T.CARNEVALI, M. Tecnologia de Lógica Programável Especificada como Hardware Reconfigurável, FPGA (Field Programmable Gate Array) ou Dispositivo Lógico Programável em Campo. Traducao . 1. ed. Vila Rica: Faculdade de Extrema, 2013.

6.Ordonez, E. et al. Projeto, Desempenho e Aplicações de Sistemas Digitais em Circuitos Programáveis (FPGAs). Traducao . São Paulo: Bless, 2003..

7. FPGA Fundamentals - National Instruments. Disponível em: <http://www.ni.com/white-paper/6983/pt/>. Acesso em: 4 set. 2016

8. Spartan 3 Family Datasheet, Family Datasheet DS099, Xilinx, Inc., 2013. Disponível em: <http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf>. Acesso em: 4 ago. 2016.

9. Spartan 3E User Guide, USer guide UG331, Xilinx, INc.,2011. Disponível em: <http://www.xilinx.com/support/documentation/User_guide/ug331.pdf>. Acesso em: 4 ago. 2016

10. Basys 2 ™ FPGA Board Reference Manual. Disponível em: <https://reference.digilentinc.com/_media/basys2:basys2_rm.pdf>. Acesso em: 10 jun. 2016.

11. D'Amore, R. VHDL. Traducao . Rio de Janeiro: Grupo Gen - LTC, 2012.

12.Marcelo Chiesse da Silva, L. Luiz Marcelo Chiesse da Silva. Disponível em: <http://webcache.googleusercontent.com/search?q=cache:6FFiH8n3fwMJ:paginapessoal.utfpr.edu.br/chiesse/disciplinas/logicareconfiguravel/vhdl/VHDL.pdf/at_download/file+&cd=4&hl=pt-BR&ct=clnk&gl=br&client=firefox-b-ab>. Acesso em: 3 set. 2016.

13. Manual Reference VHDL. Disponível em: <http://www.ics.uci.edu/~jmoorkan/vhdlref/Synario%20VHDL%20Manual.pdf>. Acesso em: 3 ago. 2016.

14. S. ARANTES, D. FPGA e FLUXO DE PROJETO. Traducao . 1. ed. CAMPINAS - SP: UNICAMP, 2016.

15. Oppenheim, A.Schafer, R. Processamento em tempo discreto de sinais. Traducao . 3. ed. São Paulo: Pearson, 2012.

16. PmodAD1 Reference Manual. Traducao . 1. ed. [s.l.] DIGILENT, 2016.

17. PmodDA2 Reference Manual. Traducao . 1. ed. [s.l.] DIGILENT, 2016.

18. Data sheet AD7476. Traducao . 1. ed. [s.l.] ANALOG DEVICES, 2016.

19. BOMAR, B. Digital Signal Processing Handbook. Traducao . Boca Raton: Vijay K. Madisetti, 1999.