Instituto Superior Técnico

Signal Processing Electronic System

Introduction to the Digital Signal Processing

Bandpass Tracking Filter

Vasco Luz - 105747
Victor Macedo - 105095
MEE - 2022/2023

16 de abril de 2023

Índice

1 Introdução

Neste trabalho foi desenvolvido um filtro passa banda de rastreamento, o qual ajusta a frequência central para a frequência mais significativa da onda de entrada de forma automática. O projeto tem como objetivo aplicar os conceitos aprendidos durante as aula teóricas de "Sistema de Processamento Digital de Sinais". Para melhor entender o funcionamento desse o projeto foi dividido em 6 blocos de funcionamento (fig 1) os quais devem ser explicado com maior detalhe na secção 3 Por fim na secção 4 esta descrito os resultados obtidos dos blocos testados individualmente e combinados.

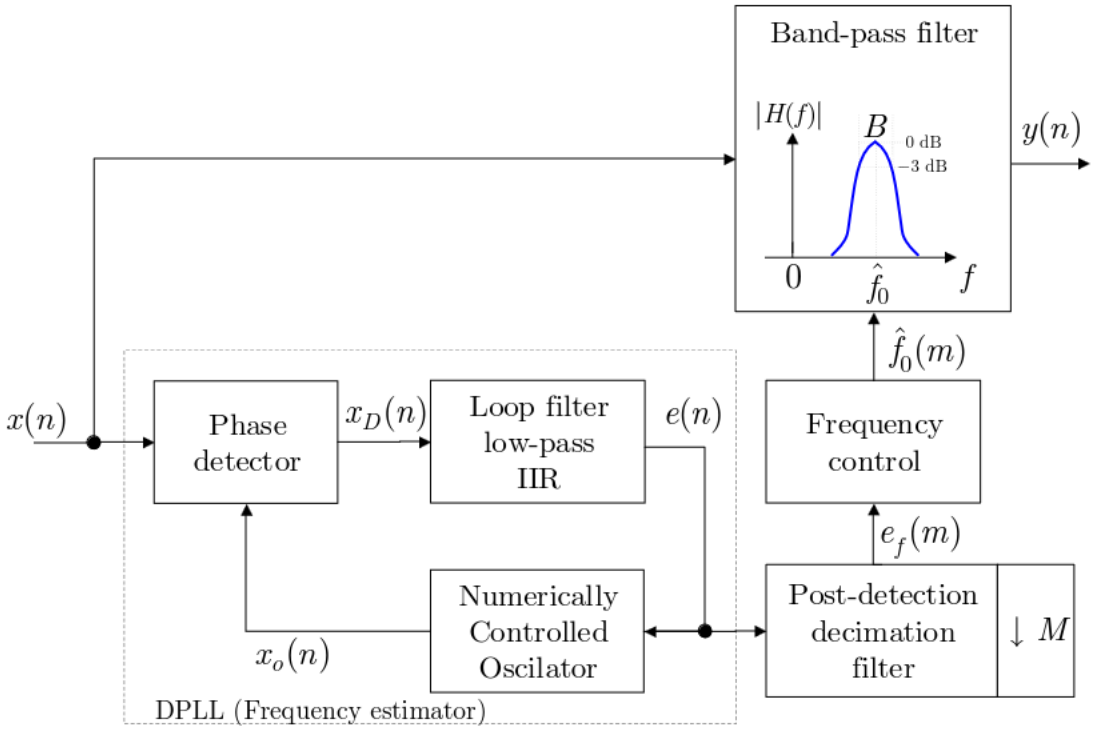

Figura 1: Diagrama de blocos do filtro passa banda de rastreamento

2 Materiais

Para a realização dos experimentos foi utilizado Starter Kit DSK TMDX5515eZDSP da *Texas Instrument*, o qual funciona com 16 bits, possui um processador de ponto fixo digital com 240 MIPS de performance e 320kB de memória interna no chip. O kit ainda apresenta um codec de áudio estéreo de 16 bits e interface dedicada a USB 2.0.

Para o desenvolvimento do software utilizou-se o programa *Code Composer Studio* (CCS), na versão 8.3.1. O mesmo fornece um ambiente integrado de desenvolvimento (IDE) baseado em

Eclipse e inclui todos os requisitos da aplicação como *assembler*, compilador, *linker* e *debugger*. Além de permitir debugging em tempo real e analise usando RTDX (Real Time Data Exchange).

3 Teoria

3.1 Phase Detector

Neste caso o Phase Detector implementado no sistema, é a multiplicação da onda de entrada pela onda gerada pelo Oscilador NCO exemplo, assumindo duas ondas sinusoidais $\mathrm{x}_{in}(t)$ e $x_{NCO}(t)$:

$$y(t) = A_{IN}(sin(w_{in}t + \theta_{in})) * A_{NCO}(sin(w_{NCO}t + \theta_{NCO})) \tag{1}$$

$$y(t) = (1/2) * A_{IN} * A_{NCO} * (sin(w_{in}t + \theta_{in} + w_{NCO}t + \theta_{NCO}) + sin(w_{in}t - \theta_{in} + w_{NCO}t + \theta_{NCO})) \tag{2}$$

como geralmente a frequência de entrada tem aproximadamente a mesma frequência que a onda do oscilador pode-se simplificar a formula para, onde :

$$X(\triangle\theta) = (1/2) * A_{IN} * A_{NCO} * sin(\triangle\theta) \tag{3}$$

Para o calculo do ganho do detetor de fase utiliza a seguinte expressão:

$$KD = \frac{X(\triangle\theta)}{\partial\triangle\theta)} \tag{4}$$

$$KD = \frac{A_{IN} * A_{NCO}cos(\triangle\theta)}{2} \tag{5}$$

Como a derivada, e um coseno, pode-se concluir que o ganho não é linear.

Como A_IN max e A_NCO max menores que 1 estão os dois representados em Q15. faz-se a multiplicação e depois retira-se o bit de sinal repetido, e depois da transforma-se o numero outra vez em Q15. usa-se o long para o numero temporariamente ter 32 bits durante a multiplicação.

3.2 Loop Filter

O Loop filter é um simples filtro passa baixa de primeira ordem que segue a equação 6

$$e(n) = \alpha * e(n-1) + (1-\alpha) * x_d(n) \tag{6}$$

fazendo a transformada z

$$e(z) = \alpha * Z^{-1}e(z) + (1-\alpha) * X_d(Z)$$

$$e(z) - \alpha * Z^{-1}e(z) = (1-\alpha) * X_d(Z)$$

$$\frac{e(z)}{x_d(z)} = \frac{1-\alpha}{1-Z^{-1}}$$

$$\frac{e(z)}{x_d(z)} = \frac{Z(1-\alpha)}{Z-1}$$

usando a formula de frequência de corte: calculou-se o mapa de polos-zeros para uma frequência de 10, 100 e 1000 Hz

$$f(s) = \frac{1-\alpha}{\alpha} * \frac{f_s}{\pi} \tag{7}$$

usou-se o matlab para por o mapa de polos-zeros como mostra as figuras

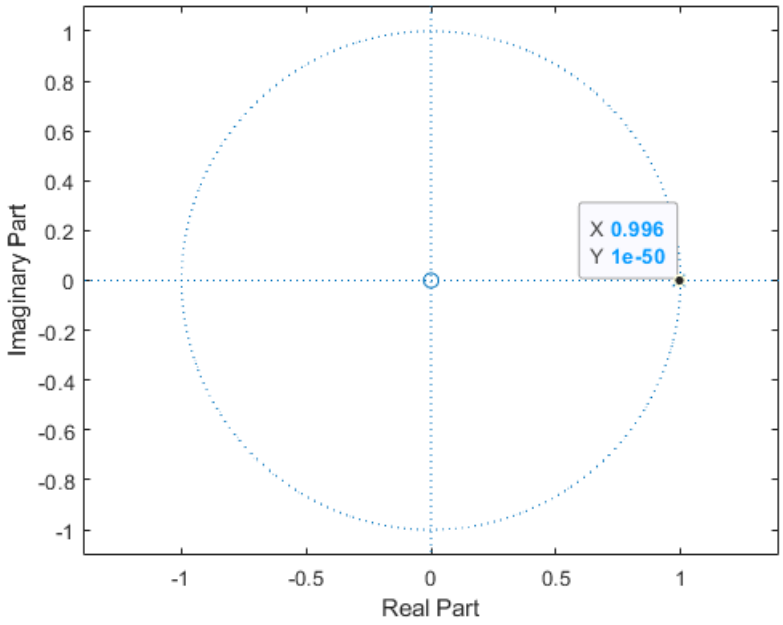

Figura 2: Polo e zeros do filtro para f=10 Hz

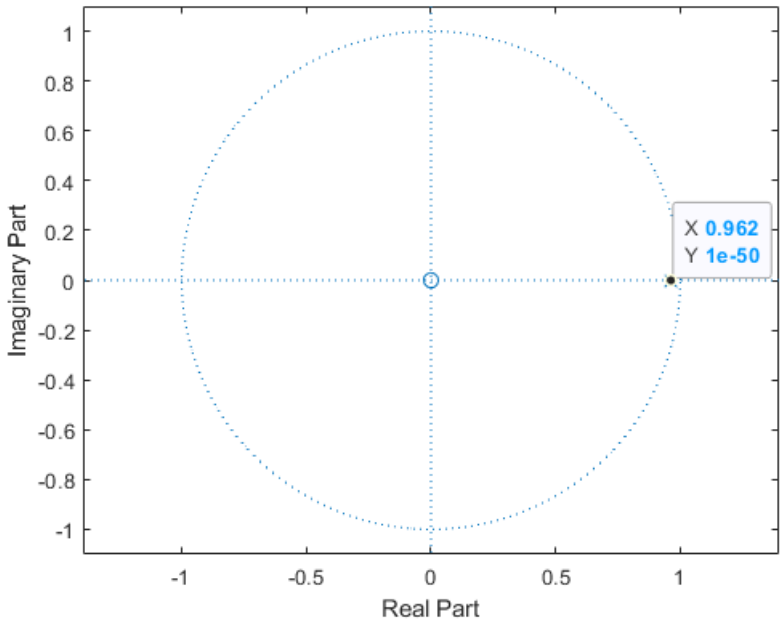

Figura 3: Polo e zeros do filtro para f=100 Hz

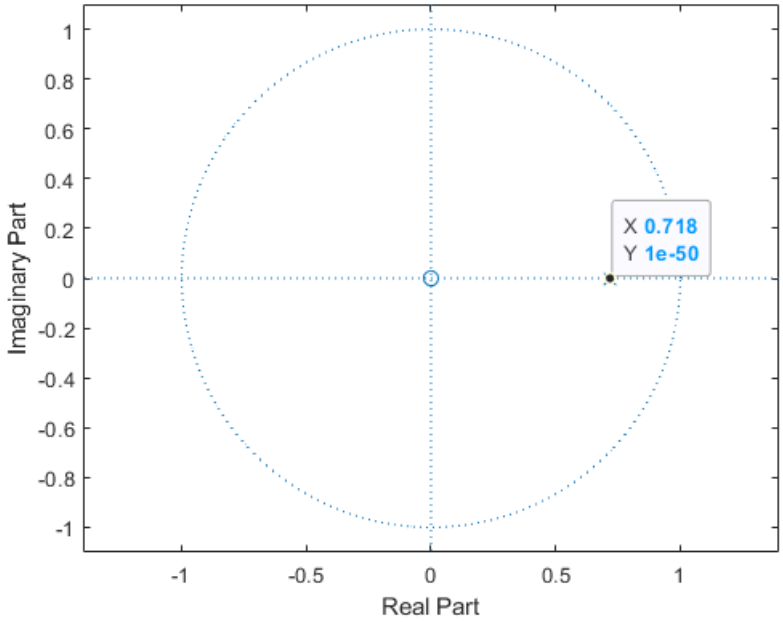

Figura 4: Polo e zeros do filtro para f=1 kHz

α nunca poderá ser maior do que 1 para o filtro ser estável. isto pois se α for maior do que 1(em modulo), e(n) tende para infinito ou para menos infinito(caso o α seja negativo e menor que 1) ou noutras palavras instável. Se α menor do que zero o filtro funciona como um filtro passa alta.

Por fim foi calculado os valores de α para as seguintes frequências:

- $f_c = 10Hz \rightarrow \alpha = 0,996$
- $f_c = 100Hz \rightarrow \alpha = 0,962$
- $f_c = 250Hz \rightarrow \alpha = 0,910$
- $f_c = 1kHz \rightarrow \alpha = 0,718$

3.3 Numerical controlled Oscillator

O *Numerical controlled Oscillator* já foi desenvolvido na parte 1 deste projeto, sendo assim o mesmo deve apenas ser integrado junto aos demais blocos. Mais detalhes do código utilizado pode ser visto no Index, na secção 6.1

3.4 Post-detection CIC filter

Após a sincronização do loop a saída do filtro apresenta uma componente de ruido devido a multiplicação senoidal do *phase detector*. Além disso como foi explicado previamente há a necessidade de reduzir a taxa de atualização por um fator M. Somando essas duas operações (filtragem e redução da taxa de atualização) tem-se um *decimator*, como mostra a figura 5

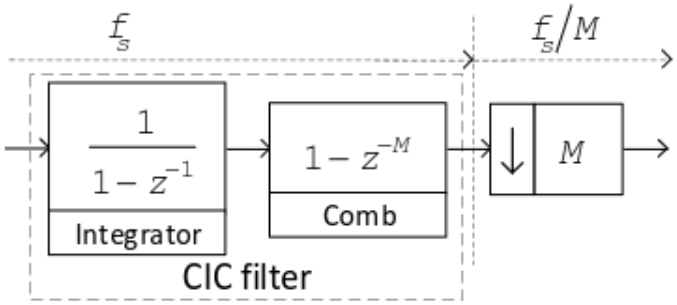

Figura 5: Decimator por M usando um estagio CIC Filter

3.5 Second order bandpass IIR filter

O filtro passa banda é um filtro de segunda ordem de resposta infinita ao impulso (IIR), com frequência central programável e $-3dB$ de largura de banda lateral. A função transferência é dado por:

$$T_{BP}(z) = K_B \frac{1 - z^{-2}}{1 - 2K_f(1 - K_B)z^{-1} + (1 - 2K_B)z^{-2}} \tag{8}$$

Os parâmetros K_f e K_b controlam frequência central f_0 e a largura de banda bilateral do filtro no ganho de $-3dB$. E são calculados pelas equações a seguir:

$$K_f = \cos(2\pi \frac{f_0}{f_s}) \tag{9}$$

$$K_b = \frac{1}{1 + \cot(\pi \frac{B}{f_s})} \tag{10}$$

O calculo dos valores de K_b e K_f foi feito utilizando o software Excel e resultou nos seguintes valores:

Fo	2000	4000	6000
Kf	0.707	0	-0.707
Kb	0.073	0.073	0.073

Tabela 1: Cálculos dos valores de K_f e K_b

Como pode ser observado na tabela 1 todos os valores podem ser representados em Q15. O resultado teórico foi calculado usando o software Matlab, o código utilizado esta na secção 6.2 e os gráficos 6 e 7 apresentam as características do filtro.

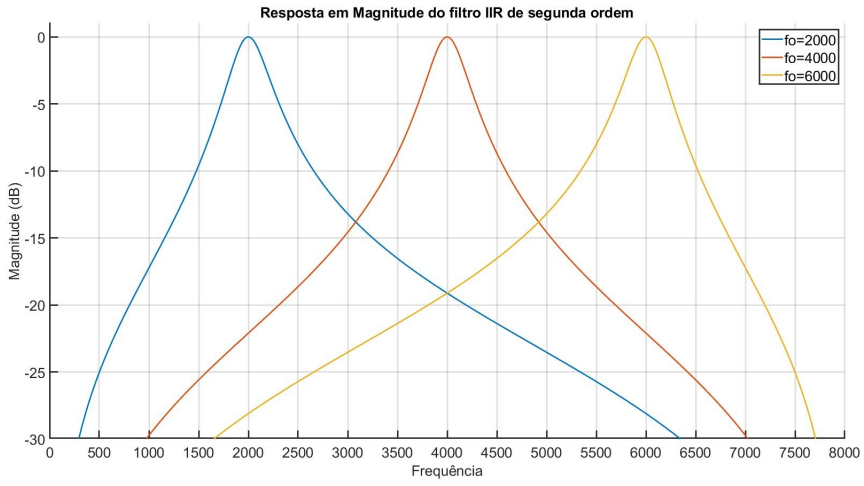

Figura 6: Resposta em magnitude do filtro IIR de segunda ordem

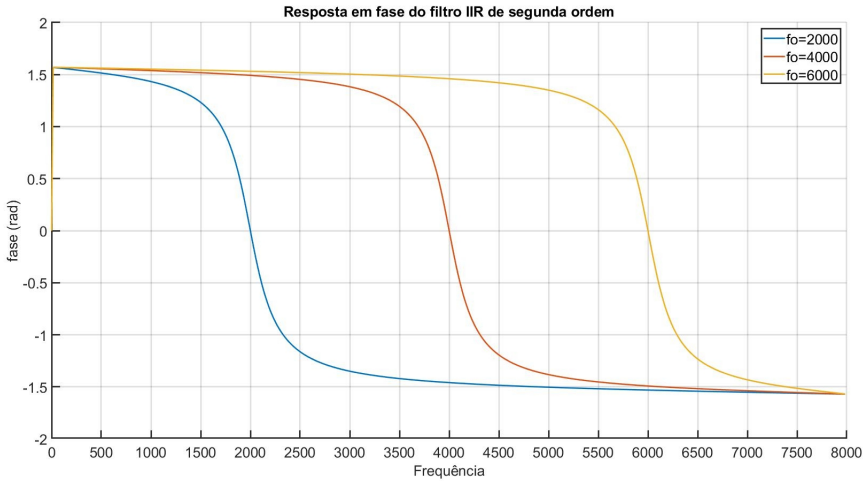

Figura 7: Resposta em fase do filtro IIR de segunda ordem

Se utilizássemos o valor de $f_0 = 0Hz$ seria obtido um filtro passa baixa como mostra a figura 8.

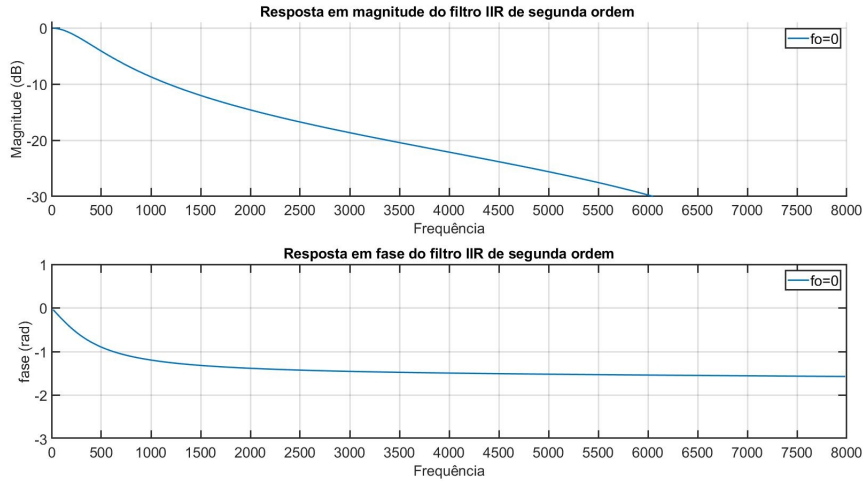

Figura 8: Filtro IIR de segunda ordem com $F_0 = 0Hz$

Utilizando o valor de $f_0 = 8000Hz$ o filtro obtido é um passa alta, como mostra a figura 9.

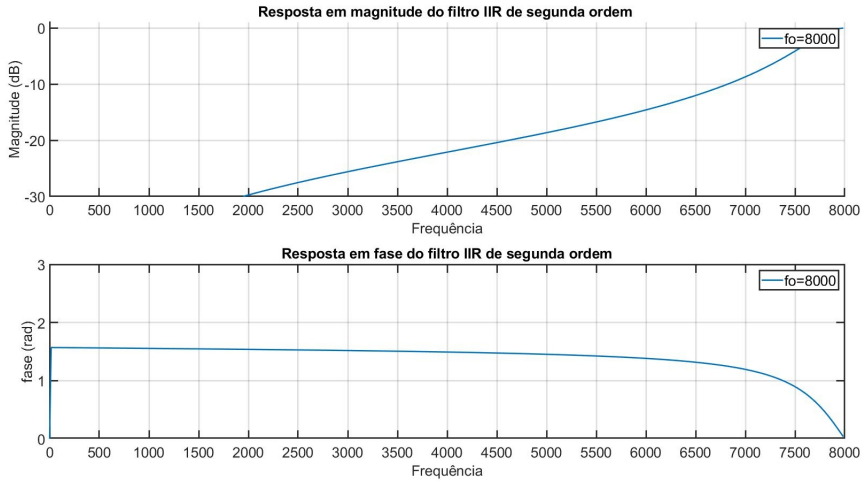

Figura 9: Filtro IIR de segunda ordem com $F_0 = 8000Hz$

A implementação do filtro é feita usando a forma direta II, conforme o diagrama da figura 10

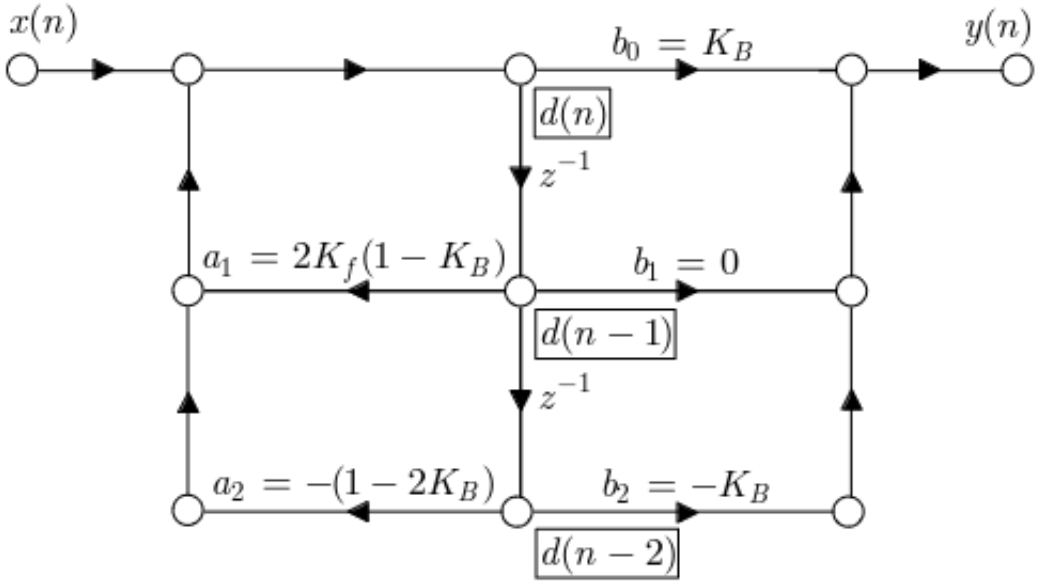

Figura 10: Diagrama do fluxo de sinal do filtro passa banda

O qual resulta na seguinte equação para implementação no código:

$$y(n) = a_1 y(n-1) + a_2 y(n-2) + b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) \tag{11}$$

3.6 Frequency control

O objetivo deste bloco é calcular o valor de K_f da equação 9, o qual controla a frequência central do filtro, usando a informação do erro $e_f(n)$. A frequência central se limita a $2kHz \leq f_0 \leq 6kHz$ para $-0.5 \leq e_f(n) \leq 0.5$. Para efetuar o calculo deve ser feita uma aproximação linear como mostra a figura 11

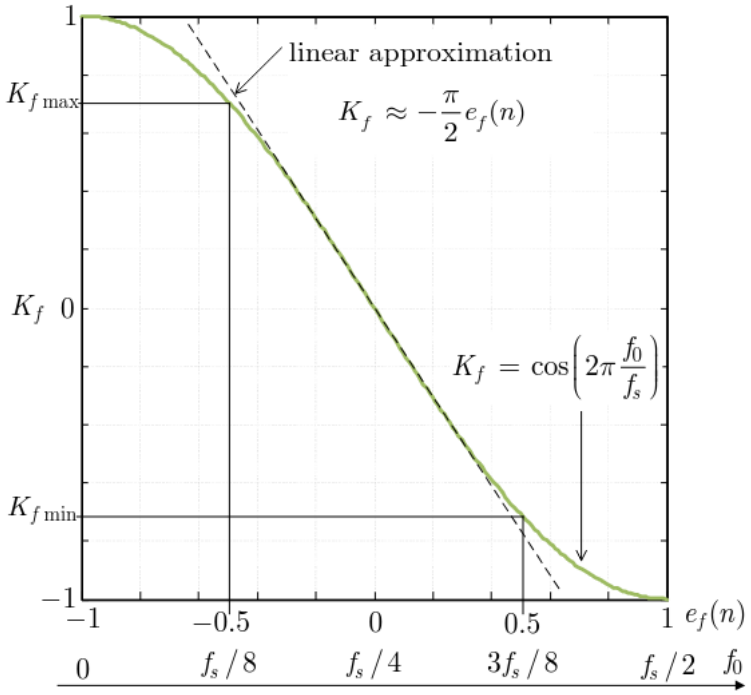

Figura 11: Variação de K_f em função de $\epsilon_f(n)$

Sendo assim o valor de K_f deve ser calculado segundo a equação 12

$$K_f \approx -\frac{\pi}{2} e_f(n) \tag{12}$$

O único parâmetro que varia com K_f é $a1$ logo o mesmo deve ser recalculado após alteração do K_f

4 Laboratorial

4.1 Desenvolvimento do filtro passa banda IIR

Reescrevendo a equação 11 de acordo com o ponto fixo tem-se a equação 13

$$(Q_{14} * Q_{14}) - (Q_{14} * Q_{14}) + (Q_{14} + Q_{15}) - (Q_{14} + Q_{15}) \tag{13}$$

$$(Q_{13} - Q_{14}) + (Q_{15} - Q_{15})$$

$$Q_{12} + Q_{14} = Q_{11}$$

De acordo com a teoria de ponto fixo o resultado em Q_{11} garante a representação do valor desejado, porém o uso de mais casas decimais pode trazer maior precisão.

Para aplicar o filtro IIR foi necessário aplicar a equação 11 utilizado o seguinte pedaço de código:

```
xband[0] = DataInLeft;
yband[0] = (((((long)a1*yband[1])<< 1) -(long)27986*yband[2]+(long)2391*xband
    [0]-(long)2391*xband[2]) << 1) >> 16;


yband[2]=yband[1];
yband[1]=yband[0];

xband[2]=xband[1];
xband[1]=xband[0];
```

Figura 12: Sinal de entrada de 4 kHz e filtro centrado em 4kHz

A figura 12 foi calculada usando o formato Q15 e como pode ser observado não apresentou *overflow* logo é o formato mais adequado para a aplicação.

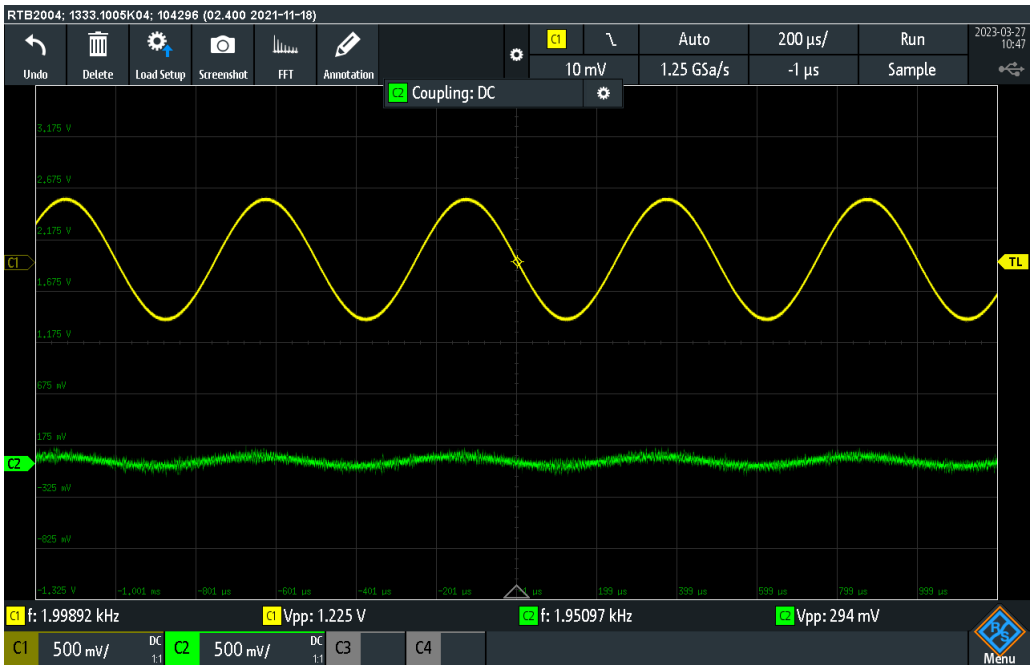

Figura 13: Sinal de entrada de 2 kHz e filtro centrado em 4 kHz

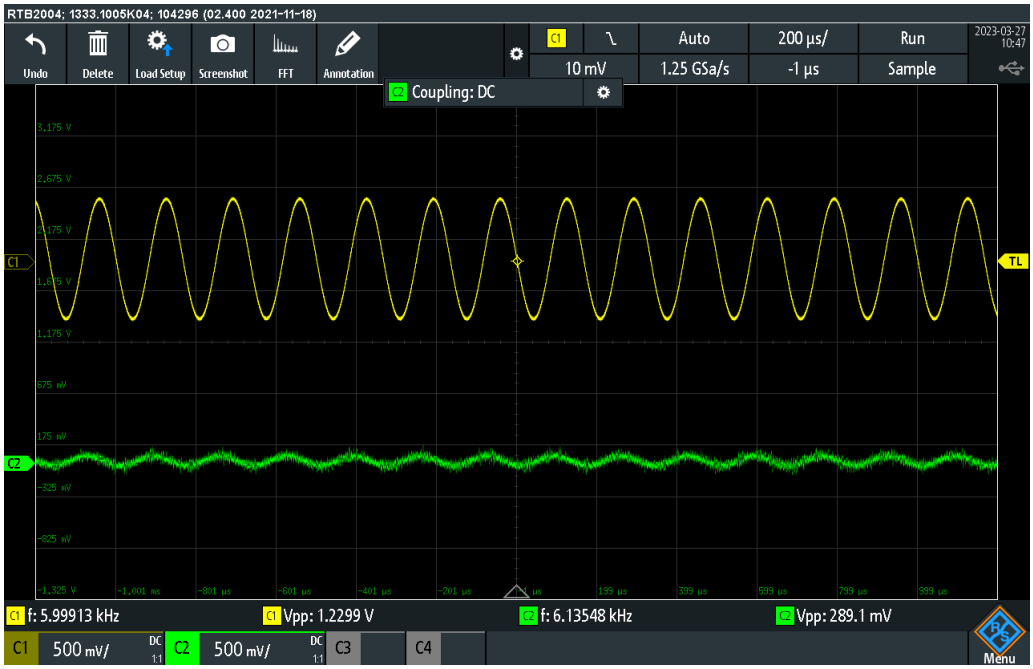

Figura 14: Sinal de entrada de 6 kHz e filtro centrado em 4 kHz

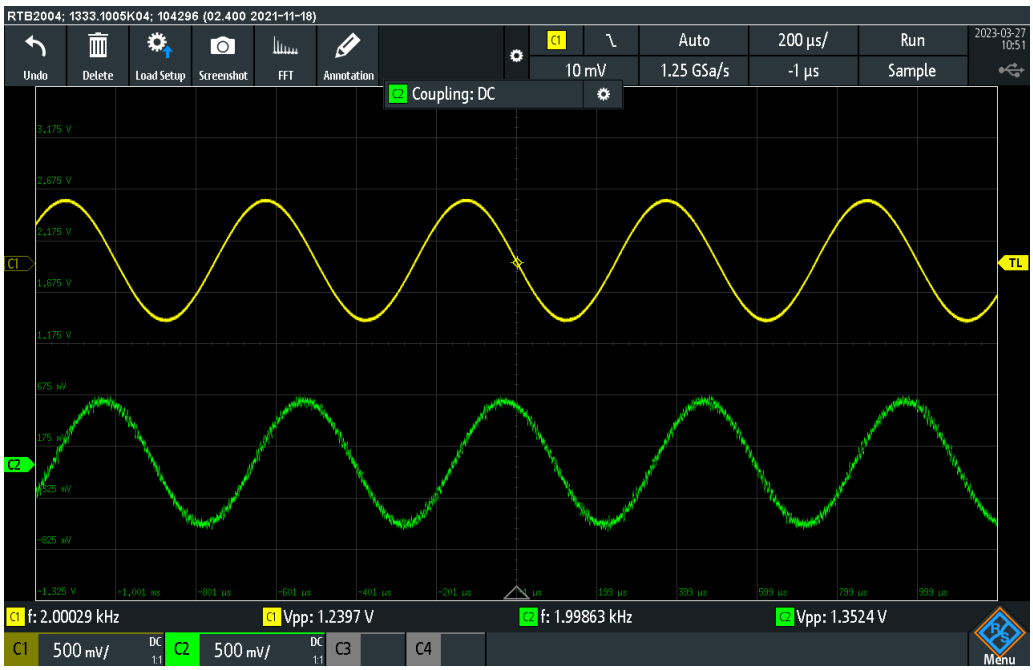

Figura 15: Sinal de entrada de 2 kHz e filtro centrado em 2 kHz

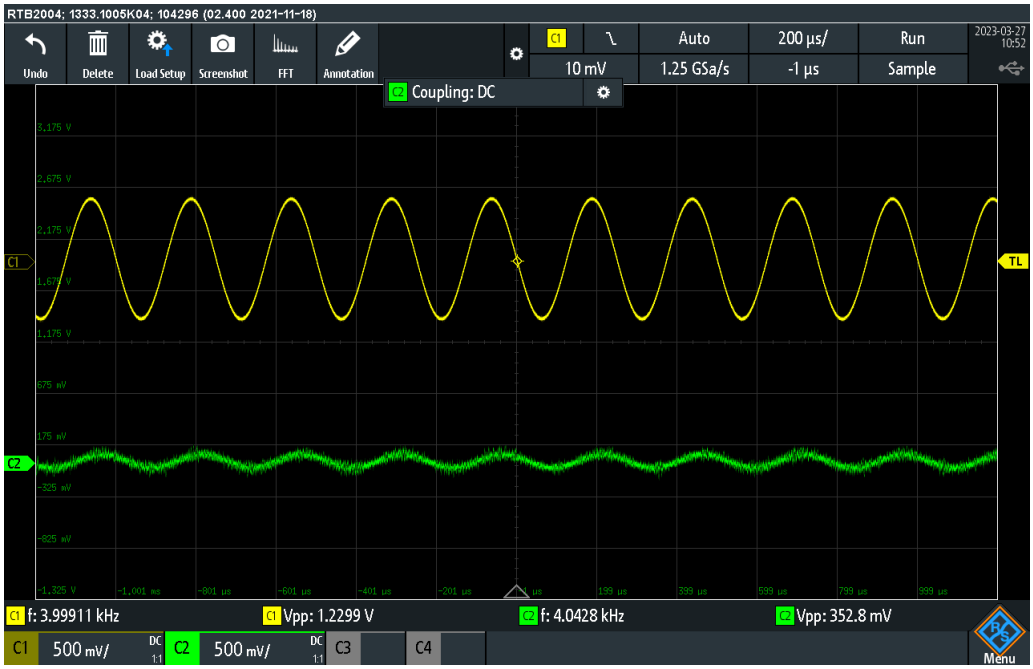

Figura 16: Sinal de entrada de 4 kHz e filtro centrado em 2 kHz

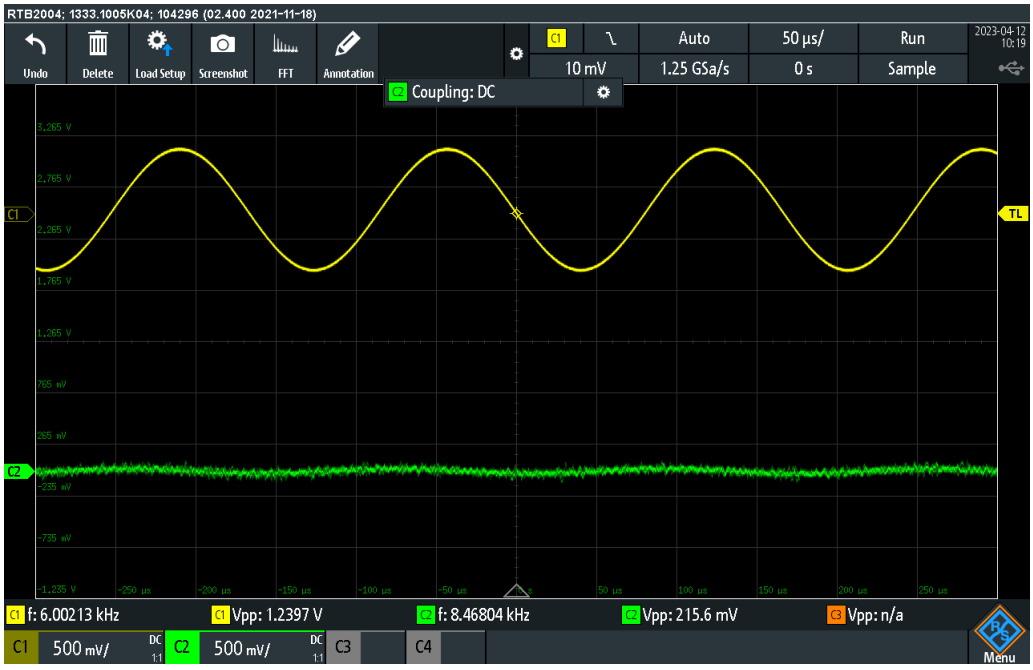

Figura 17: Sinal de entrada de 6 kHz e filtro centrado em 2 kHz

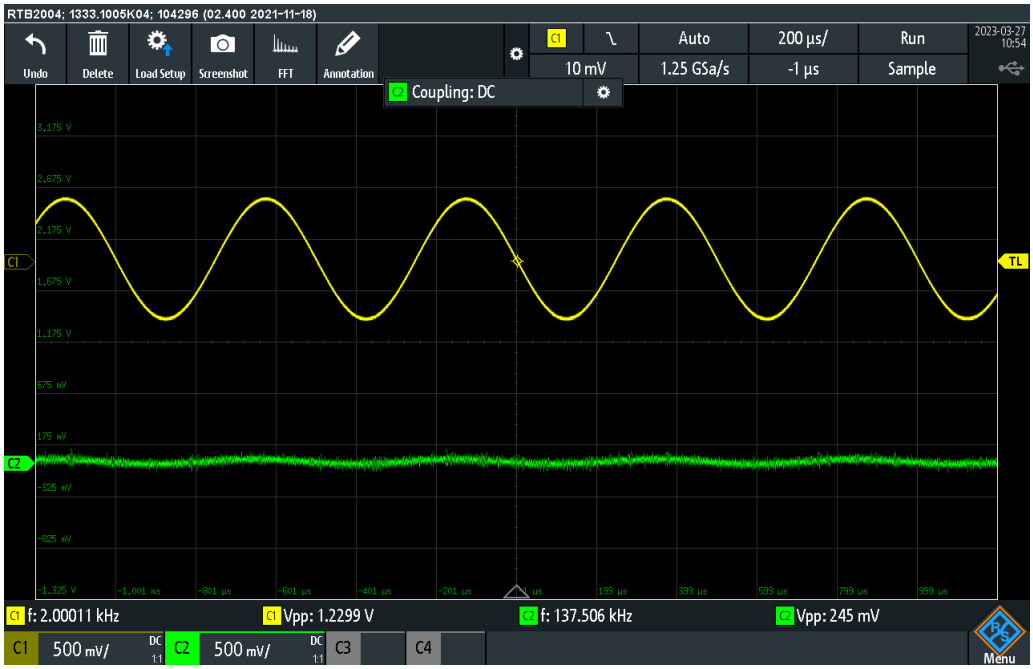

Figura 18: Sinal de entrada de 2 kHz e filtro centrado em 6kHz

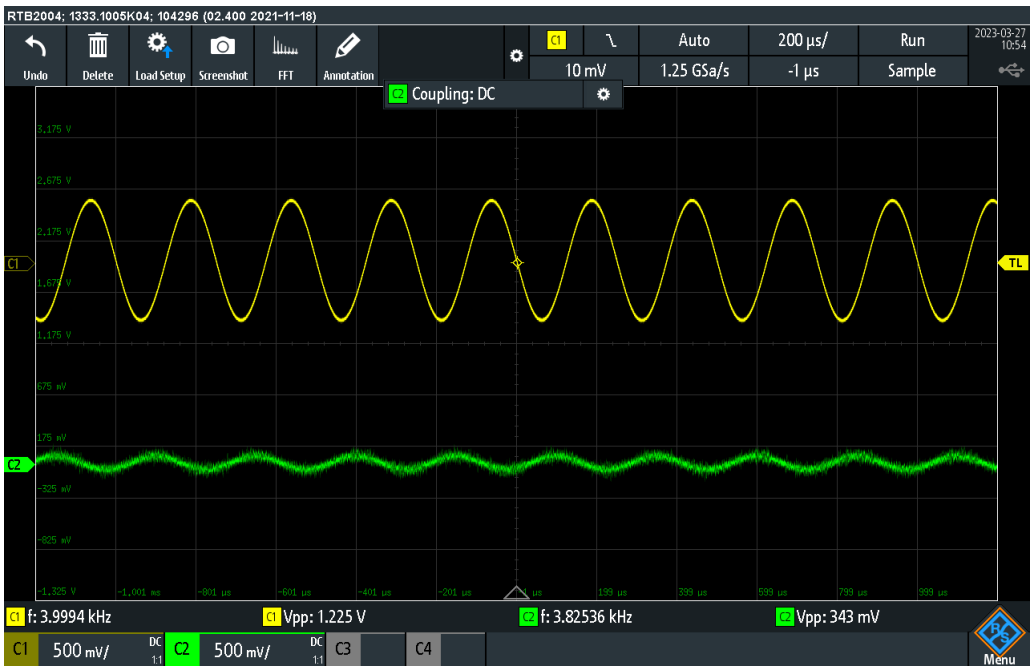

Figura 19: Sinal de entrada de 4 kHz e filtro centrado em 6 kHz

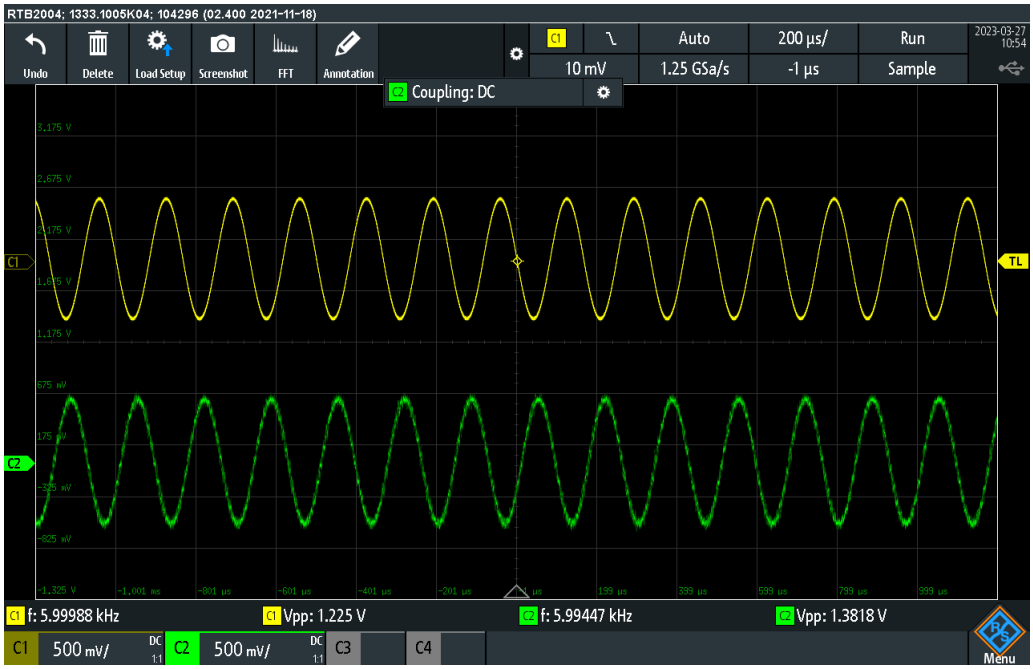

Figura 20: Sinal de entrada de 6 kHz e filtro centrado em 6 kHz

Como pode ser observado nas imagens acima o filtro cumpre seu requisito nas 3 frequências desejadas.

4.2 Desenvolvimento DPLL

	Parameters	NCO gain k = 0.25		Amax = 1.2 Vpp	
	Amplitude	Amax = 1.2 Vpp		Amax/2 = 0.6 Vpp	
	Loop filter	fc=100 hz	fc=10 hz	fc=100 hz	fc=10 hz
Ascending sweep	fc-	4455	4835	4390	4390
	fl+	3765	3895	3850	3960
Descending sweep	fc+	4245	4115	4180	4040
	fl-	3565	3165	3580	3610
	Δfc	-210	-720	-210	-350
	Δfl	200	730	270	350

Tabela 2: Características do DPLL com o primeiro ganho

Aumentando o valor do NCO até que o limite do *hold* esteja entre 1500Hz e 6500Hz obtém-se os resultados da tabela 3

	Parameters	NCO gain k = 085		Amax = 1.2 Vpp	
	Amplitude	Amax = 1.2 Vpp		Amax/2 = 0.6 Vpp	
	Loop filter	fc=100 hz	fc=10 hz	fc=100 hz	fc=10 hz
Ascending sweep	fc-	6630	6630	5470	5390
	fl+	3350	3830	3700	3958
Descending sweep	fc+	4550	4230	4420	4050
	fl-	1310	1370	2550	2590
	Δfc	-2080	-2400	-1050	-1340
	Δfl	2040	2460	1150	1368

Tabela 3: Características do DPLL com o segundo ganho

O ganho foi alterado para próximo de 0.85

4.3 Controle de frequência do filtro passa banda

Como foi explicado na secção 3.6 é preciso implementar a equação:

$$a1 = 2 * K_f * (1 - K_B) \tag{14}$$

Para garantir $a1$ variável foi utilizado o seguinte pedaço de código

```
a1 = ((2*(long)Kf*(32767-2555)) << 1 ) >> 16; //a1 = 2*Kf*(1-Kb) Valor
varia entre -1.3 e 1.3  Q14
```

O resultado obtido após a implementação foi:

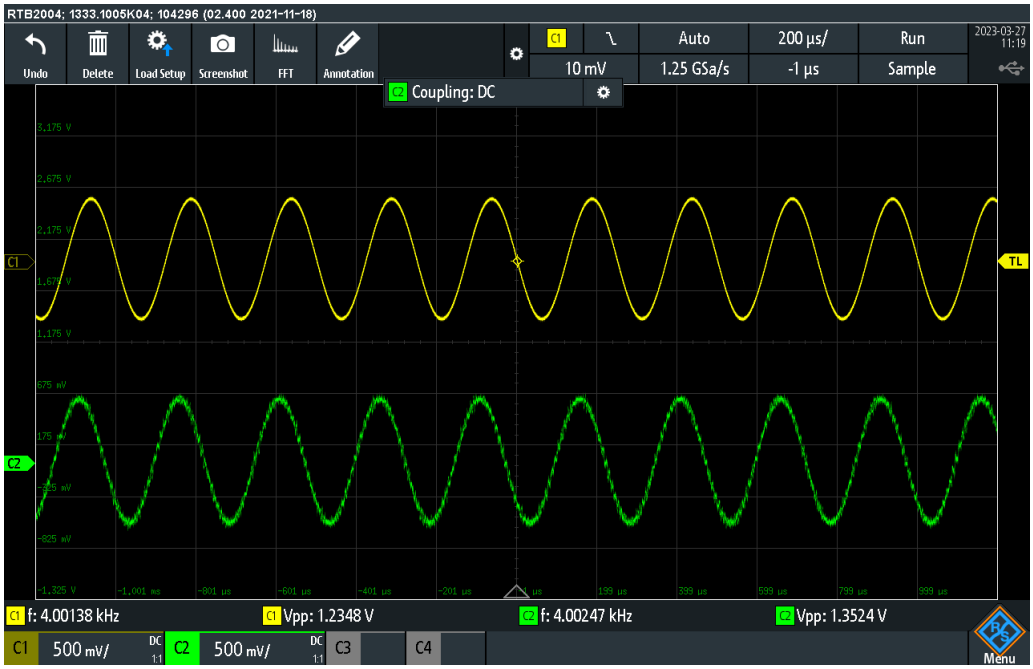

Figura 21: a1 sendo calculado em tempo real para ef = 0

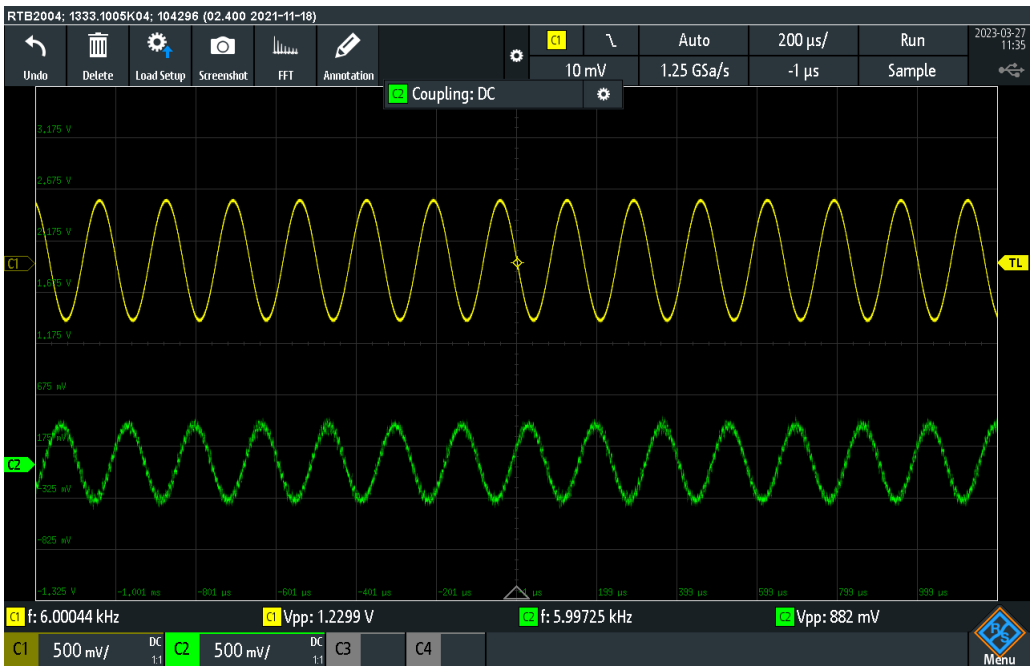

Figura 22: a1 sendo calculado em tempo real para ef = 0.5

Vale ressaltar que o filtro centrado em 6 kHz apresenta certa deformação devido a aproximação. Sendo assim a frequência de corte real se aproxima de 6350 Hz.

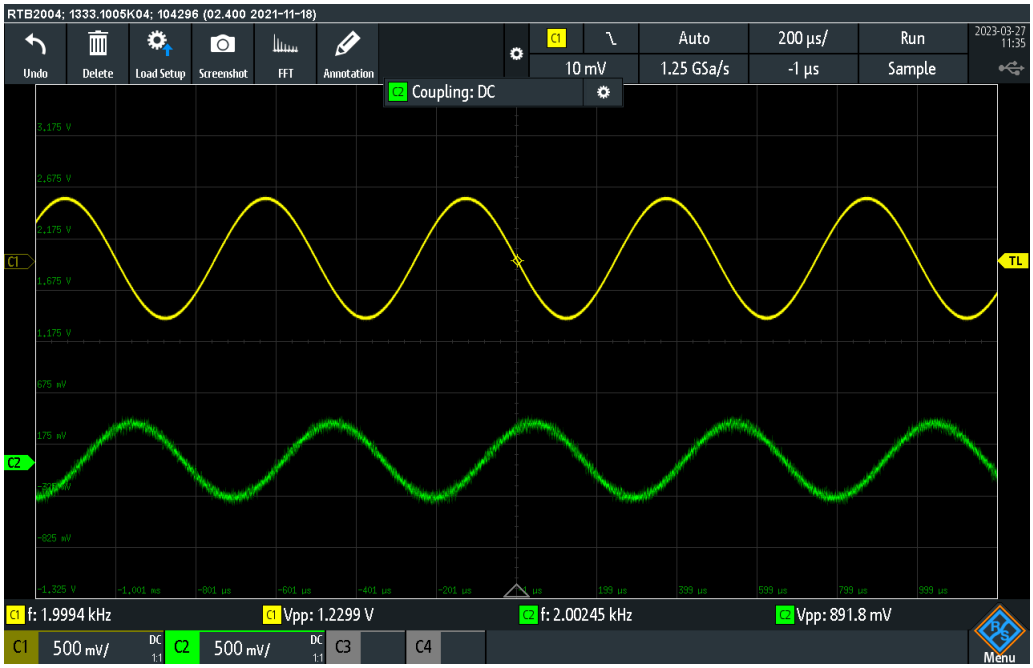

Figura 23: a1 sendo calculado em tempo real para ef = -0.5

O filtro centrado em 2 kHz apresenta o mesmo problema que o anterior, possuindo a frequência de corte real próximo de 1650 Hz.
A deformação vista nos filtros de 2 kHz e 6 kHz já era esperada devido a aproximação visto na secção 3.6, sendo assim o resultado esta dentro do esperado.

4.4 Post detection filter and Decimation

A metodologia do filtro passa baixa utilizada foi IIR ao invés de CIC, devido a prévia experiencia. Sendo assim utilizo-se a equação 6 com $f_c = 250$ Hz, o qual resultou em $\alpha = 0,91$. Seguido de um compressor de $M = 32$, a implementação foi feita usando o seguinte trecho de código:

```
    //Filtro passa baixa IIR Fc=250, alpha = 0,91
fd = ((((long)29839*fd + (32767-29839)*(long)ec))<<1)>>16; //Resultado em Q15
dec++;
//Salva o valor de fd a cada 32 amostras para realizar a decimacao
if (dec == 32)
    {
        dec = 0 ;
        yd = fd;
```

```
    }
```

Variando a frequência de entrada (amarelo) de 20 Hz até 250 Hz foram obtidos os seguintes resultados da saída do filtro (verde) e da saída do decimador (laranja):

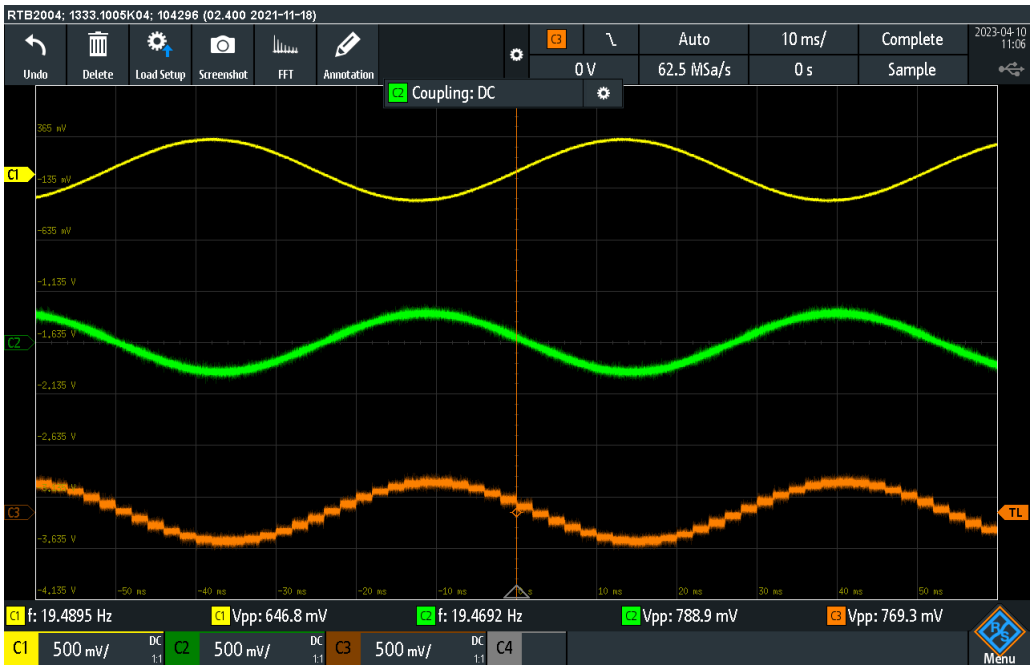

Figura 24: Resposta do filtro IRR/decimador para entrada de 20 Hz

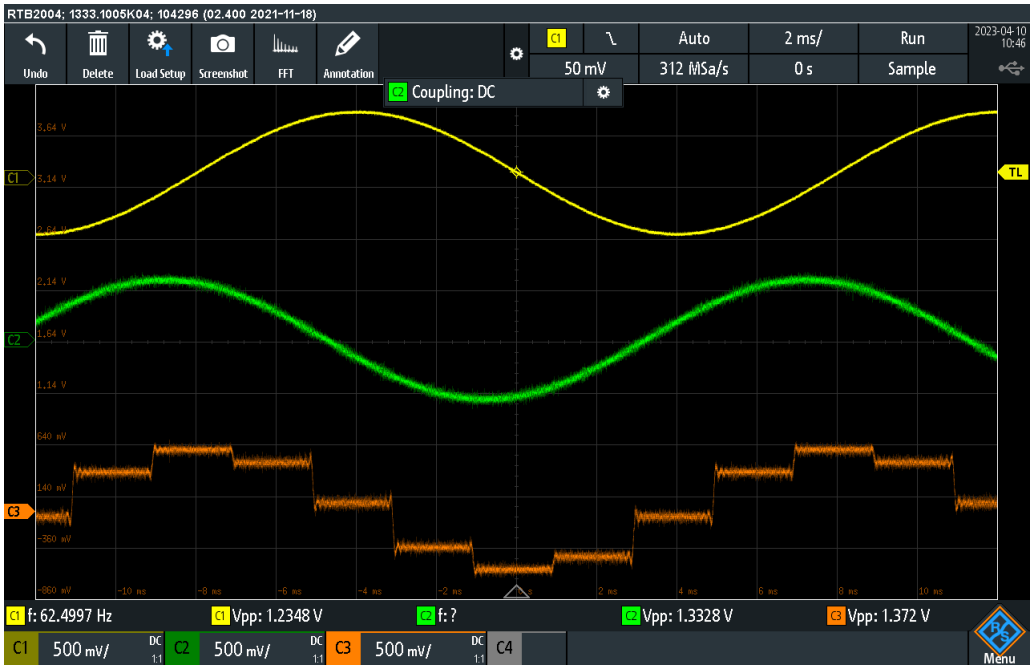

Figura 25: Resposta do filtro IRR/decimador para entrada de 62.5 Hz

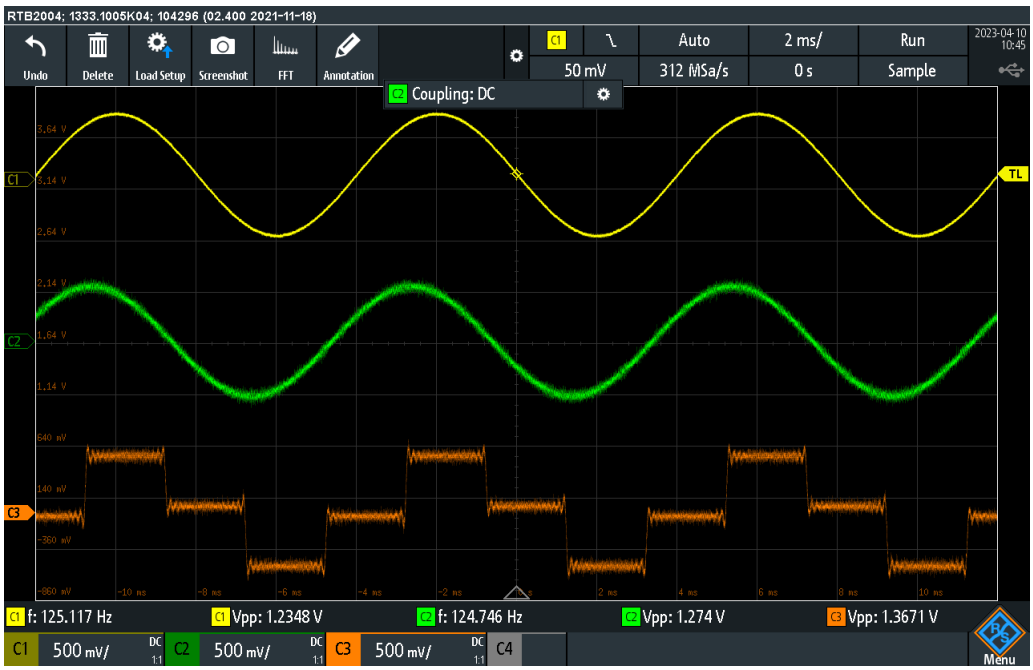

Figura 26: Resposta do filtro IRR/decimador para entrada de 125 Hz

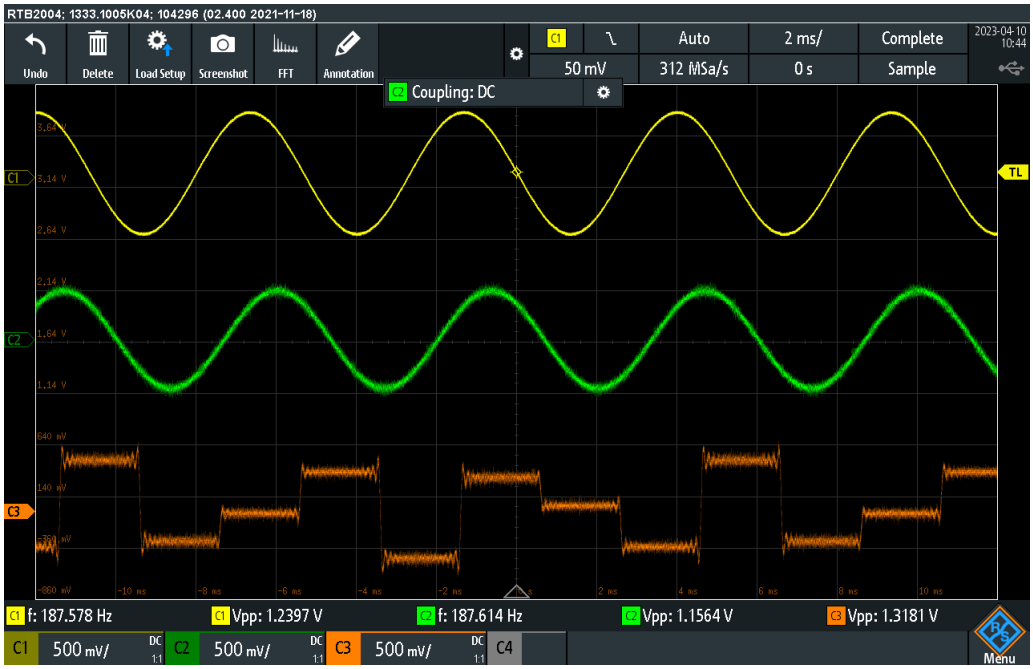

Figura 27: Resposta do filtro IRR/decimador para entrada de 187 Hz

Vale ressaltar que na frequência próxima de 187 Hz a saída do decimador apresentou instabilidade.

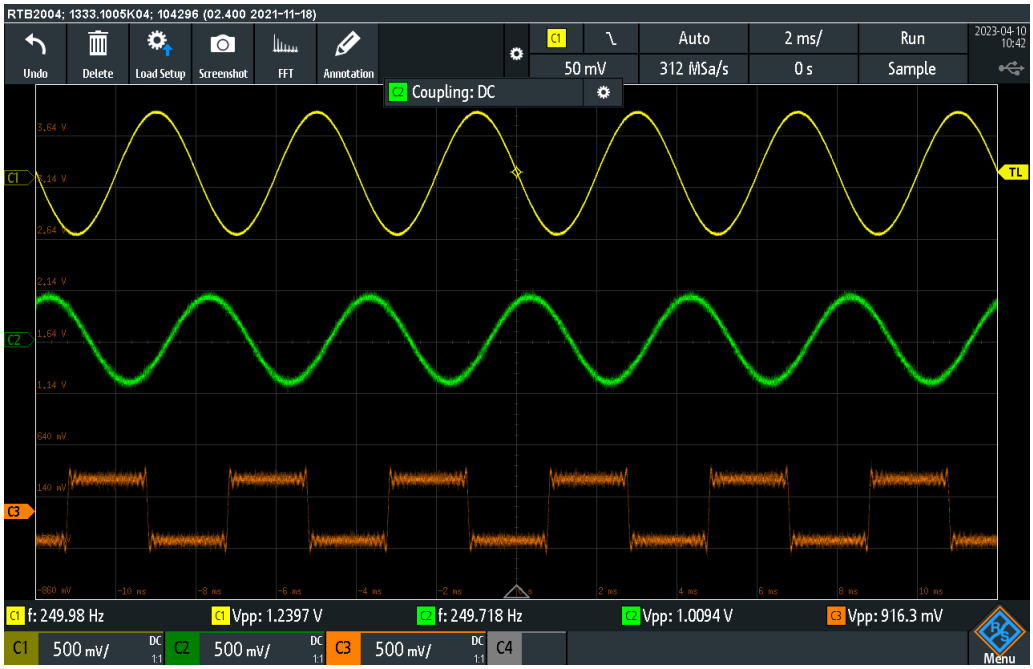

Figura 28: Resposta do filtro IRR/decimador para entrada de 250hz

Como observados nas figuras 24 a 28 obteve-se os resultados de acordo com a teoria. Obteve-se resultados estáveis para frequências que mantivesse um relação fracionaria par com 250 Hz como pode-se ser observado na figura 25 e 26.

4.5 Sistema completo

Ao conectar todos os blocos foi possível testar o sistema completo. Para realizar a testagem foi observado a frequência de entrada do gerador de função (amarelo), a saída do filtro passa banda (verde) e a saída do NCO (laranja) para monitorar a sincronização do DPLL.

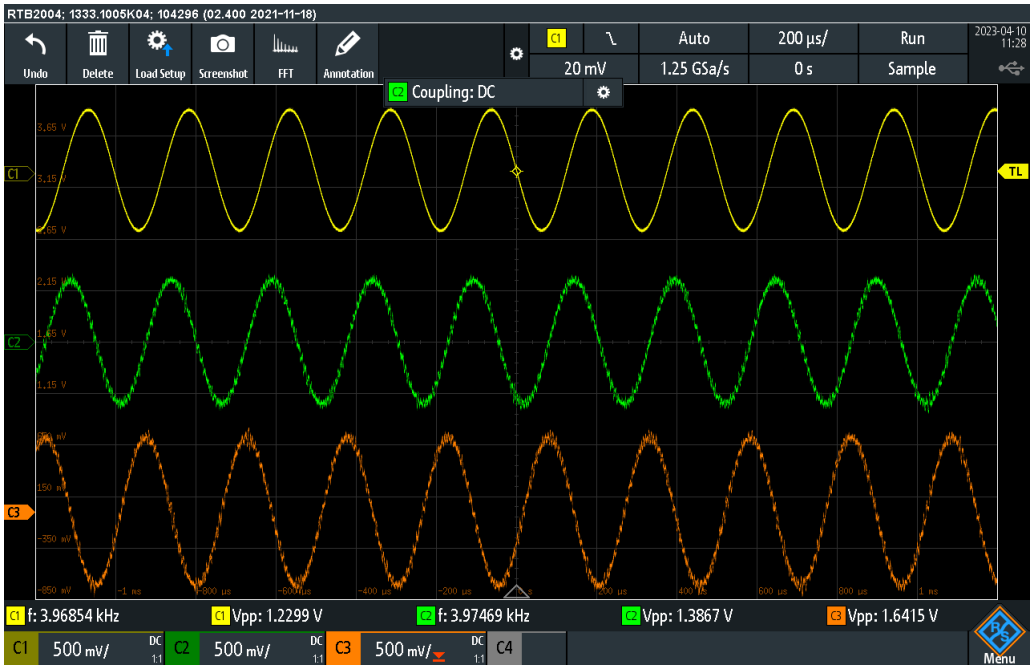

Figura 29: Exemplo teste do sistema completo

Na figura 29 pode-se observar o bom funcionamento do Bandpass tracking filter. Pode-se observar um desfazamento entre a onda de entrada(amarelo) e a onda de saída do filtro(verde), este desfazamento esta associado ao delay do dsp utilizado.

	Parameters	NCO gain k = 1	$A = A_{max} \approx 1.2V$
		M = 32	$f_c = 100$
	Input frequency (kHz)	Output amplitude (V)	
Ascending frequency sweep	1	0.240	
	1.5	0.269	
	2	0.308	
	2.5	0.343	
	3	0.421	
	3.5	1.320	
	4	1.370	
	4.5	1.310	
	5	1.254	
	5.5	1.270	
	6	1.360	
	6.5	1.060	
	7	0.240	
Descending frequency sweep	7	0.240	
	6.5	0.260	
	6	0.294	
	5.5	0.343	
	5	0.441	
	4.5	1.360	
	4	1.370	
	3.5	1.330	
	3	1.259	
	2.5	1.260	
	2	1.360	
	1.5	1.010	
	1	0.235	

Tabela 4: Caracterização global do Tracking filter

Ao utilizar o software *Waveform* foi possível adquirir os valores da tabela 4 para 200 pontos e então plotar um gráfico com os valores:

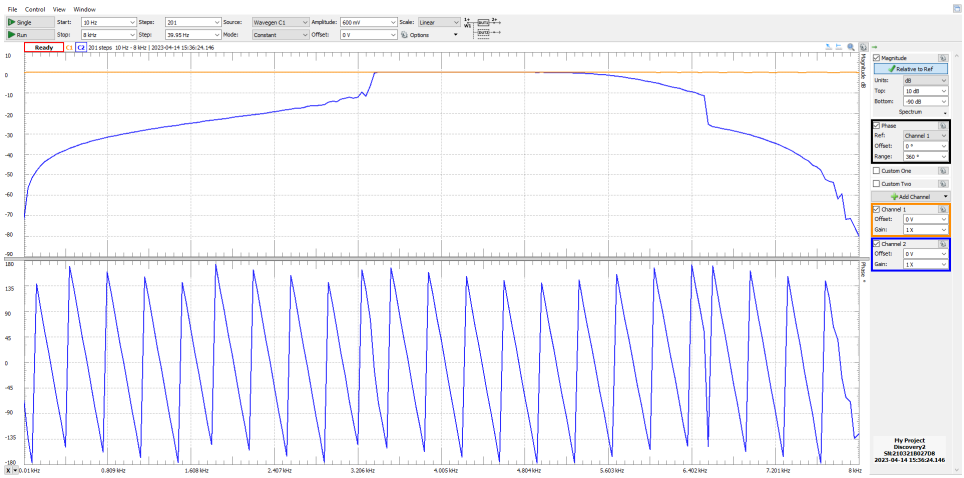

Figura 30: Ascending sweep para K_f aproximado

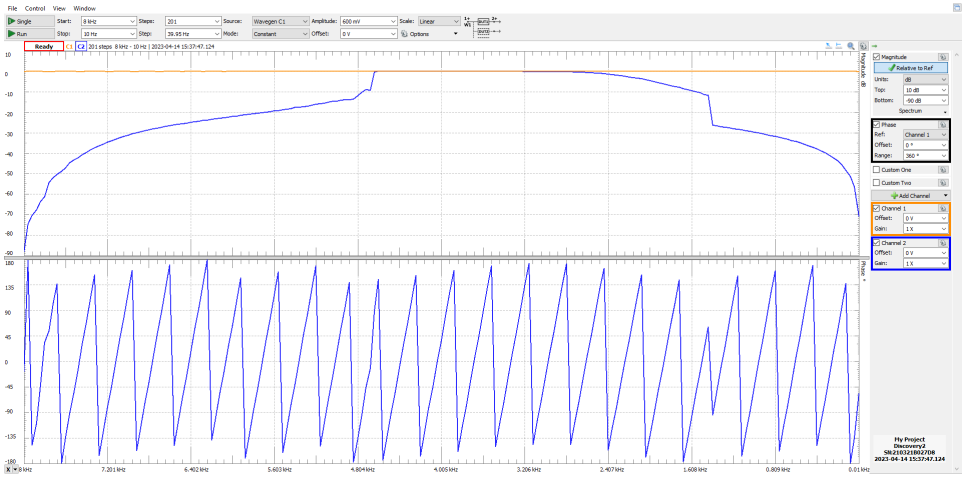

Figura 31: Descending sweep para K_f aproximado

4.6 Variável de controle exata

No bloco de controle de frequência é calculado o valor de K_f, o qual deve controlar a frequência central do filtro passa banda, porém o valor utilizado até então era uma aproximação do valor real:

$$K_f = \cos(2\pi\frac{f_0}{fs}) = -\sin(\frac{\pi}{2}e_f(n)) \approx -\frac{\pi}{2}e_f(n) \tag{15}$$

Utilizando o seguinte trecho de código foi possível substituir a aproximação de K_f pelo valor do seno, utilizou-se também de interpolação para obter melhores resultados.

```
    senkf = (((16384*(long)yd)) << 1 ) >> 16; //ef/2 resultado final em Q15
senkf2 = (senkf & 31744) >> 10;           //0b1111 1100 0000 000  e Utiliza os
     5 MSB + bit de sinal e converte para Q10


Kf = (-look_sen[senkf2] >> 1);    //Converte o valor do sen de Q15 para Q14
Kf2 = (-look_sen[senkf2+1] >> 1);
if(senkf<0){
    Kf = -Kf;
}
I_Kf = (senkf & 1023) << 5;           //0b0000 0011 1111 1111 Utiliza os
    outros bits para interpolacao
inter = Kf+((((Kf2-Kf)*(long)I_Kf)<<1)>>16); //Interpolacao do seno
```

	Parameters	NCO gain k = 1	A = $A_{max} \approx 1.2V$
		M = 32	$f_c = 100$
	Input frequency (kHz)	Output amplitude (V)	
Ascending frequency sweep	1	0.196	
	1.5	0.260	
	2	0.278	
	2.5	0.340	
	3	0.438	
	3.5	1.320	
	4	1.330	
	4.5	0.906	
	5	0.886	
	5.5	0.878	
	6	0.846	
	6.5	0.528	
	7	0.200	
Descending frequency sweep	7	0.200	
	6.5	0.214	
	6	0.256	
	5.5	0.306	
	5	0.402	
	4.5	0.904	
	4	1.336	
	3.5	1.332	
	3	1.324	
	2.5	1.310	
	2	1.302	
	1.5	0.838	
	1	0.222	

Tabela 5: Caracterização global do Tracking filter, com K_f exato

Devido ao facto que a única forma de calcular senos e com a utilização de uma lookup table(com 32 valores mas realmente só 8 serão utilizados) pode-se observar algumas discrepâncias na amplitude das saídas em certas ocasiões durante alterações da frequência de entrada, isto devido a mudança súbita do valor por causa da lookup table.
Assim como no caso aproximado utilizou-se o software *Waveform* para uma aquisição mais exata dos dados, resultando nos seguintes gráficos

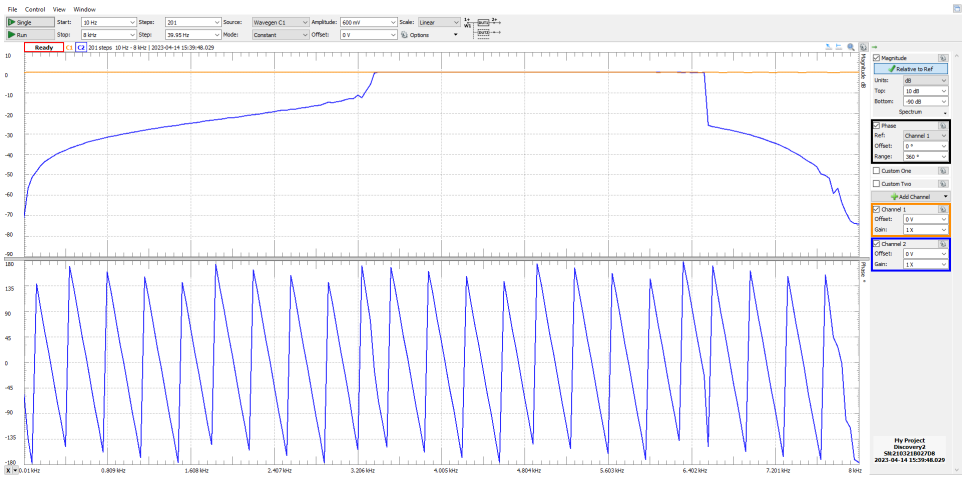

Figura 32: Ascending sweep para K_f exato

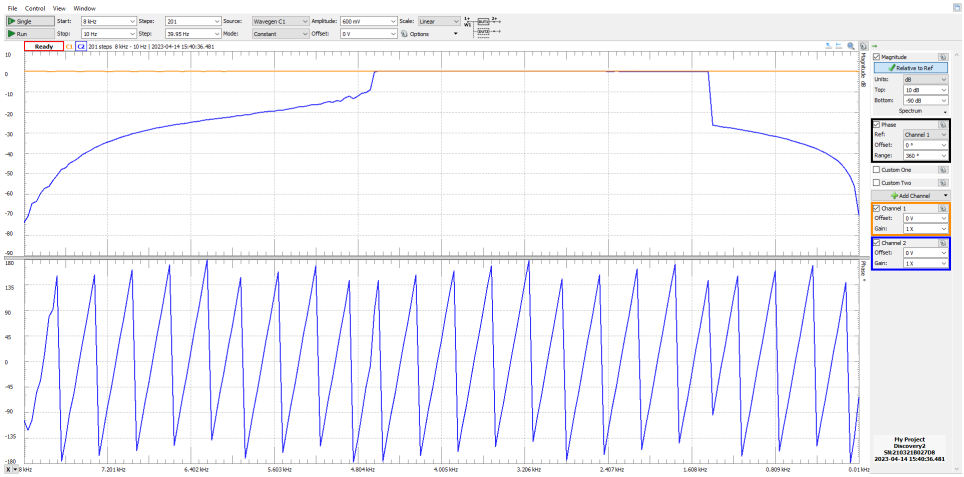

Figura 33: Descending sweep para K_f exato

Comparando as figuras 30 e 32 é notável a melhora ao utilizar o valor exato, ao utilizar o valor aproximado o valor da tensão começa a ser atenuado a partir de 4800 Hz, enquanto o mesmo mantém-se até 6500 Hz, ao usar o valor exato. O mesmo ocorre, de forma inversa, ao compararmos as figuras 31 e 33.

5 Conclusão

Neste laboratório o grupo teve o primeiro contato com uma placa de processamento digital de sinais. Além disso foi possível aplicar os conceitos teóricos vistos em sala de aula como filtros,

decimadores e detetores de fase.
Por fim foi possível entender e desenvolver um filtro passa banda seguidor, o qual é muito importante em diversas aplicações, desde sistemas de cancelamento de ruido até comunicação com satélites.

6 Index

6.1 Código NCO

```
    delta = delta0 + ((((long)k * e)<<1)>>16);
//Soma dos sinais
r = r + delta;
//rmin = rmin + deltamin;  Nao necessario no projeto?
//rmax = rmax +deltamax;    Nao necessario no projeto?

s = r & 31744;            //0b1111110000000000
s =s >> 10;               //Utiliza os 5 MSB + bit de sinal e converte para Q15

t = look_sen[s];         //Converte o valor da rampa em uma senoide
y2 = look_sen[s+1];
if(r<0){                  // if para inverter o valor do sen
    t = -t;
    y2= -y2;
}
f = (r & 1023) << 5;           //0b0000 0011 1111 1111 Utiliza os outros bits
    da rampa para interpolacao
inter = t+((((y2-t)*(long)f)<<1)>>16); //Interpolacao do seno
NCO = (((long)gain * inter)<<1) >> 16;//Calcula a nova saida do NCO
```

6.2 Código Matlab filtro passa banda

```
    Kb = 0.07295965727;
Kf = 0.7071067812;
b0 = Kb;
b2 = [1  0  -1];
a2 = [1  -2*Kf*(1-Kb)  1-2*Kb];

b = b0*b2;
a = a2;
[h0,w0] =freqz(b,a,512,16000);

Kf = 0;
```

```
b0 = Kb;
b2 = [1  0  -1];
a2 = [1  -2*Kf*(1-Kb)  1-2*Kb];
b = b0*b2;
a = a2;
[h1,w1] = freqz(b,a,512,16000);

Kf = -0.7071067812;
b0 = Kb;
b2 = [1  0  -1];
a2 = [1  -2*Kf*(1-Kb)  1-2*Kb];
b = b0*b2;
a = a2;
[h2,w2] = freqz(b,a,512,16000);

figure(1)
hold on
plot(w0,20*log10(abs(h0)))
plot(w1,20*log10(abs(h1)))
plot(w2,20*log10(abs(h2)))

ax = gca;
grid on;
ax.YLim = [-30  1];
ax.XLim = [0  8000];
ax.XTick = 0:500:8000;
xlabel('Frequencia')
ylabel('Magnitude (dB)')
lh = legend("fo=2000","fo=4000","fo=6000");
title("Resposta em Magnitude do filtro IIR de segunda ordem")

figure(2)
hold on
grid on
plot(w0,(angle(h0)))
plot(w1,(angle(h1)))
plot(w2,(angle(h2)))



ax = gca;
grid on;
ax.YLim = [-2  2];
ax.XLim = [0  8000];
ax.XTick = 0:500:8000;
xlabel('Frequencia')
ylabel('fase (rad)')
lh = legend("fo=2000","fo=4000","fo=6000");
```

```
title("Resposta em fase do filtro IIR de segunda ordem")
```

6.3 Código completo

```
    #define AIC3204_I2C_ADDR 0x18
#include "usbstk5515.h"
#include "usbstk5515_gpio.h"
#include "usbstk5515_i2c.h"
#include "stdio.h"
#include "usbstk5515.h"
extern Int16 AIC3204_rset( Uint16 regnum, Uint16 regval);
#define Rcv 0x08
#define Xmit 0x20


/* ------------------------------------------------------------------------ *
 *                                                                          *
 *  main( )                                                                 *
 *                                                                          *
 * ------------------------------------------------------------------------ */
void main( void )


{
    Int16 r = 0,s, rmax, rmin,delta, delta0, deltamin, deltamax,gain = 32767,
    SDD,t,k = 27853,y2,inter,f,phase, alfa = 31523,//fc = 100 -> alfa = 0.962
    look_sen[33]= {0,3212,6393 ,9512
    ,12539,15446,18204,20787,23170,25329,27245,28898,30273,31356,32137,32609,

    32767,32609,32137,31356,30273,28898,27245,25329,23170,20787,18204,15446,12539,
     9512,6393,3212,0},
    NCO, e,ec, xband[3] = {0,0,0}, yband[3] = {0,0,0},Kf,ef,a1,fd,dec = 0,yd,
    senkf,senkf2,Kf2,I_Kf;
    // Definicao do Delta = 4,096 * Fo
    delta0 = 16384;                                 //Variavel Delta a ser
    incrementada
    deltamin = 8192;                                //4.096*2000
    deltamax = 24576;                               //4.096*6000

    /* Initialize BSL */
    USBSTK5515_init( );

    /* Configure AIC3204 */

       AIC3204_rset( 0, 0 );          // Select page 0
```

```
    AIC3204_rset( 1, 1 );          // Reset codec
    AIC3204_rset( 0, 1 );          // Select page 1
    AIC3204_rset( 1, 8 );          // Disable crude AVDD generation from
DVDD
    AIC3204_rset( 2, 1 );          // Enable Analog Blocks, use LDO power
    AIC3204_rset( 0, 0 );          // Select page 0
    /* PLL and Clocks config and Power Up  */
    AIC3204_rset( 27, 0x0d );      // BCLK and WCLK is set as o/p to
AIC3204(Master)
    AIC3204_rset( 28, 0x00 );      // Data ofset = 0
    AIC3204_rset( 4, 3 );          // PLL setting: PLLCLK <- MCLK,
CODEC_CLKIN <-PLL CLK
    AIC3204_rset( 6, 7 );          // PLL setting: J=7
    AIC3204_rset( 7, 0x06 );       // PLL setting: HI_BYTE(D=1680)
    AIC3204_rset( 8, 0x90 );       // PLL setting: LO_BYTE(D=1680)
    AIC3204_rset( 30, 0x88 );      // For 32 bit clocks per frame in Master
 mode ONLY
                                   // BCLK=DAC_CLK/N =(12288000/8) = 1.536
MHz = 32*fs
    AIC3204_rset( 5, 0x91 );       // PLL setting: Power up PLL, P=1 and R
=1
    AIC3204_rset( 13, 0 );         // Hi_Byte(DOSR) for DOSR = 128 decimal
or 0x0080 DAC oversamppling
    AIC3204_rset( 14, 0x80 );      // Lo_Byte(DOSR) for DOSR = 128 decimal
or 0x0080
    AIC3204_rset( 20, 0x80 );      // AOSR for AOSR = 128 decimal or 0x0080
 for decimation filters 1 to 6
    AIC3204_rset( 11, 0x86 );      // Power up NDAC and set NDAC value to 6
 (fs=16kHz)
    AIC3204_rset( 12, 0x87 );      // Power up MDAC and set MDAC value to 7
    AIC3204_rset( 18, 0x87 );      // Power up NADC and set NADC value to 7
    AIC3204_rset( 19, 0x86 );      // Power up MADC and set MADC value to 6
 (fs=16kHz)
    /* DAC ROUTING and Power Up */
    AIC3204_rset(  0, 0x01 );      // Select page 1
    AIC3204_rset( 12, 0x08 );      // LDAC AFIR routed to HPL
    AIC3204_rset( 13, 0x08 );      // RDAC AFIR routed to HPR
    AIC3204_rset(  0, 0x00 );      // Select page 0
    AIC3204_rset( 64, 0x02 );      // Left vol=right vol
    AIC3204_rset( 65, 0x00 );      // Left DAC gain to 0dB VOL; Right
tracks Left
    AIC3204_rset( 63, 0xd4 );      // Power up left,right data paths and
set channel
    AIC3204_rset(  0, 0x01 );      // Select page 1
    AIC3204_rset( 16, 0x00 );      // Unmute HPL , 0dB gain
    AIC3204_rset( 17, 0x00 );      // Unmute HPR , 0dB gain
    AIC3204_rset(  9, 0x30 );      // Power up HPL,HPR
    AIC3204_rset(  0, 0x00 );      // Select page 0
```

```
        USBSTK5515_wait( 500 );         // Wait

        /* ADC ROUTING and Power Up */
        AIC3204_rset( 0, 1 );           // Select page 1
        AIC3204_rset( 0x34, 0x30 );     // STEREO 1 Jack
                                        // IN2_L to LADC_P through 40 kohm
        AIC3204_rset( 0x37, 0x30 );     // IN2_R to RADC_P through 40 kohmm
        AIC3204_rset( 0x36, 3 );        // CM_1 (common mode) to LADC_M through
    40 kohm
        AIC3204_rset( 0x39, 0xc0 );     // CM_1 (common mode) to RADC_M through
    40 kohm
        AIC3204_rset( 0x3b, 0x18 );        // MIC_PGA_L unmute
        AIC3204_rset( 0x3c, 0x18 );        // MIC_PGA_R unmute
        AIC3204_rset( 0, 0 );           // Select page 0
        AIC3204_rset( 0x51, 0xc0 );     // Powerup Left and Right ADC
        AIC3204_rset( 0x52, 0 );        // Unmute Left and Right ADC

        AIC3204_rset( 0, 0 );
        USBSTK5515_wait( 200 );         // Wait
        /* I2S settings */
        I2S0_SRGR = 0x0;
        I2S0_CR = 0x8010;     // 16-bit word, slave, enable I2C
        I2S0_ICMR = 0x3f;     // Enable interrupts



        Int16 DataInLeft, DataInRight, DataOutLeft, DataOutRight;
        while(1) {
                    /* Read Digital audio */
                    while((Rcv & I2S0_IR) == 0);  // Wait for interrupt
    pending flag
                    DataInRight = I2S0_W0_MSW_R;  // 16 bit right channel
    received audio data
                    DataInLeft = I2S0_W1_MSW_R;  // 16 bit left channel
    received audio data
                    /* Write Digital audio */
                    while((Xmit & I2S0_IR) == 0);  // Wait for interrupt
    pending flag
                    I2S0_W0_MSW_W = DataOutRight;  // 16 bit right channel
    transmit audio data
                    I2S0_W1_MSW_W = DataOutLeft;  // 16 bit left channel
    transmit audio data

//-------------------------------------------------------------------
 // Your program here!!!
//-------------------------------------------------------------------
//            NCO
//-------------------------------------------------------------------
```

```
delta = delta0 + ((((long)k * e)<<1)>>16);
//Soma dos sinais
r = r + delta;
//rmin = rmin + deltamin;  Nao necessario no projeto?
//rmax = rmax +deltamax;    Nao necessario no projeto?

s = r & 31744;            //0b1111110000000000
s =s >> 10;               //Utiliza os 5 MSB + bit de sinal e converte para Q15


t = look_sen[s];         //Converte o valor da rampa em uma senoide
y2 = look_sen[s+1];
if(r<0){                  // if para inverter o valor do sen
    t = -t;
    y2= -y2;
}
f = (r & 1023) << 5;           //0b0000 0011 1111 1111 Utiliza os outros bits
    da rampa para interpolacao
inter = t+((((y2-t)*(long)f)<<1)>>16); //Interpolacao do seno
NCO = (((long)gain * inter)<<1) >> 16;//Calcula a nova saida do NCO

// Phase detector
//a simple multiplier which multiplies the sample of the input signal by the
    sample generated on the previously developed NCO
//-----------------------------------------------------------
phase = ((((long)DataInLeft * NCO)<< 1)>>16);

//-----------------------------------------------------------
// Loop Filter
//-----------------------------------------------------------
e = ((((long)alfa*e + (32767-alfa)*(long)phase))<<1)>>16; //Resultado em Q15
ec = ((((long)e*24932) <<1 ) << 1 ) >> 16;    //Valor de e corrigido e*1.76 (
    Q15)
//-------------------------------------------------------------
// Decimation
//
//-----------------------------------------------------------
//Filtro passa baixa IIR Fc=250, alpha = 0,91
fd = ((((long)29839*fd + (32767-29839)*(long)ec))<<1)>>16; //Resultado em Q15
//fd = ((((long)29839*fd + (32767-29839)*(long)DataInLeft))<<1)>>16; //
    Resultado em Q15
dec++;
//Salva o valor de fd a cada 32 amostras para realizar a decimacao
if (dec == 32)
    {
        dec = 0 ;
        yd = fd;
    }
```

```
//yd=16384>>1;
//-------------------------------------------------------------
//      Frequency Control
//-------------------------------------------------------------
senkf = (((16384*(long)yd)) << 1 ) >> 16; //ef/2 resultado final em Q15
senkf2 = (senkf & 31744) >> 10;           //0b1111 1100 0000 000  e Utiliza os
     5 MSB + bit de sinal e converte para Q10


Kf = (-look_sen[senkf2] >> 1);    //Converte o valor do sen de Q15 para Q14
Kf2 = (-look_sen[senkf2+1] >> 1);
if(senkf<0){
    Kf = -Kf;
}
I_Kf = (senkf & 1023) << 5;           //0b0000 0011 1111 1111 Utiliza os
    outros bits da rampa para interpolacao
inter = Kf+((((Kf2-Kf)*(long)I_Kf)<<1)>>16); //Interpolacao do seno
//Kf = (((-25736*(long)yd)) << 1 ) >> 16; //pi/2*ef resultado final em Q14

//-------------------------------------------------------------
//      Band Pass Filter
//-------------------------------------------------------------
xband[0] = DataInLeft;
a1 = ((2*(long)Kf*(32767-2555)) << 1 ) >> 16; //a1 = 2*Kf*(1-Kb) Valor varia
    entre -1.3 e 1.3  Q14
yband[0] = (((((long)a1*yband[1])<< 1) -(long)27986*yband[2]+(long)2391*xband
    [0]-(long)2391*xband[2]) << 1) >> 16;


yband[2]=yband[1];
yband[1]=yband[0];

xband[2]=xband[1];
xband[1]=xband[0];
//
// Saida
//-------------------------------------------
DataOutLeft = yband[0] ;        // loop left channel samples
DataOutRight = NCO;          // loop right channel samples
  }

 } // main()
```

Referências

[1] G. Tavares. "Sistemas de Processamento Digital de Sinais: Slides", IST, 2023

[2] G. Tavares "Sistemas de Processamento Digital de Sinais: Bandpass Tracking Filter", IST, 2023