UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Universidade Federal do Ceará - Campus Quixadá

Aplicação de Filtro Digital em Sistemas Embarcados
QXD0143 - Processamento Digital de Sinais

Prof. Carlos Igor Bandeira

Aluno: Hugo Bessa - **Matrícula:** 496870

Aluno: Isaac Vinícius - **Matrícula:** 500935

Aluna: Kassia Lopes - **Matrícula:** 49367

Quixadá-CE

Sumário

1 Introdução

Este documento descreve a modelagem de um filtro **Passa-Baixa FIR** utilizando *MATLAB*, com o objetivo de atenuar frequências acima de 100 Hz em um sinal de entrada composto por 50 Hz e 300 Hz. A modelagem serve como base para implementação em sistemas embarcados, utilizando um microcontrolador com o suporte da biblioteca CMSIS-DSP.

A implementação embarcada visa processar sinais em tempo real utilizando um *ADC* para aquisição de dados e um *DAC* para saída dos dados filtrados.

1.1 Filtro Passa-Baixa FIR

Filtros são uma classe de sistemas lineares invariante no tempo que selecionam determinadas componentes de frequência de um sinal de entrada, rejeitando todas as demais. Os filtros podem ser divididos em duas classes: analógicos e digitais. O primeiro é construído a partir de componentes eletrônicos, como resistores, capacitores e amplificadores. Já o segundo realiza a filtragem a partir de cálculos numéricos com o uso de processadores digitais[1].

Filtros FIR se referem a filtros cuja resposta ao impulso é finita. Com relação a sua função de transferência os filtros são classificados em quatro tipos: passa-baixa, passa-alta, passa-faixa e rejeita-faixa. Para essa implementação foi utilizada o filtro passa-faixa, que se caracteriza pela passagem de baixas frequências. Assim, quando adicionado uma frequência de corte, ele atenua as frequências maiores que a frequência de referência.

1.2 Microcontrolador e Ambiente de Desenvolvimento

O microcontrolador utilizado para a implementação é o **STM32F407VET6 Cortex-M4**, que oferece suporte a operações em ponto flutuante, tornando-o

adequado para processamento digital de sinais (DSP). Ele possui os seguintes periféricos relevantes para este projeto:

- **ADC (Conversor Analógico-Digital)**: 12 bits, com 3 unidades e 16 canais.
- **DAC (Conversor Digital-Analógico)**: 12 bits, com 1 unidade e 2 canais.

O ambiente de desenvolvimento utilizado foi o **STM32CubeIDE**, que integra ferramentas de desenvolvimento para STM32, incluindo configuração de periféricos, suporte a DMA, timers e bibliotecas CMSIS-DSP.

A figura 12 ilustra o microcontrolador STM32F407VET6 utilizado no projeto.

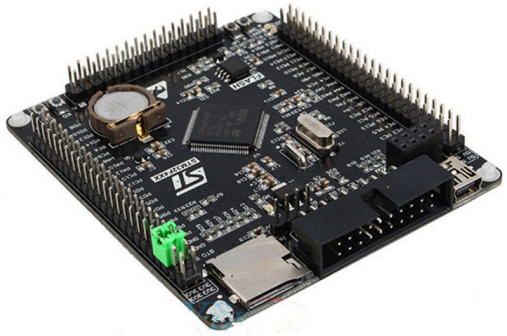

Figura 1: Microcontrolador STM32F407VET6 Cortex-M4

2 Parâmetros do Filtro Passa-Baixa

Para definir os parâmetros de um filtro passa-baixa (FIR ou IIR), é necessário considerar algumas especificações fundamentais. Abaixo estão os principais parâmetros e como eles afetam o comportamento do filtro.

2.1 Frequência de Corte (Fc)

- **Definição**: A frequência de corte é o ponto onde o filtro começa a atenuar as componentes de frequência mais alta. Em um filtro passa-baixa, ele permite que as frequências abaixo de F_c passem com pouca atenuação e bloqueia as frequências acima de F_c.

- **Como definir**: No MATLAB, a frequência de corte é definida de acordo com os requisitos do sinal. Para o exemplo atual, foi definido $F_c = 100\,\text{Hz}$.

$$F_c = 100\,\text{Hz}$$

2.2 Taxa de Amostragem (Fs)

- **Definição**: A taxa de amostragem F_s é a frequência com que o sinal contínuo é amostrado. Ela deve ser pelo menos o dobro da maior frequência presente no sinal (Teorema de Nyquist).

- **Como definir**: No exemplo, a taxa de amostragem foi definida como $F_s = 1000\,\text{Hz}$, o que é suficiente para sinais de até $500\,\text{Hz}$.

$$F_s = 1000\,\text{Hz}$$

2.3 Ordem do Filtro (N)

- **Definição**: A ordem do filtro é o número de coeficientes menos um no caso de filtros FIR. Um filtro de ordem maior tem uma transição mais abrupta entre as bandas de passagem e rejeição, mas também requer mais cálculos.

- **Como definir**: Neste projeto, foi escolhida a ordem $N = 29$, resultando em 30 coeficientes no filtro.

$$N = 29$$

2.4 Tipo de Janela (para FIR)

- **Definição**: O tipo de janela afeta a suavidade e a resposta de frequência do filtro FIR. As janelas mais comuns são: retangular, Hamming, Hanning, Blackman, entre outras.
- **Como definir**: No MATLAB, foi utilizada a função `fir1`, que por padrão usa a janela de Hamming.

$$b = \texttt{fir1}(N, F_c/(F_s/2))$$

2.5 Tamanho do Bloco (*blockSize*)

- **Definição**: O tamanho do bloco (*blockSize*) é o número de amostras processadas de uma vez e é importante tanto na modelagem quanto na implementação embarcada. Ele afeta o desempenho em tempo real e a latência.
- **Como definir**: Na implementação embarcada, este parâmetro deve ser definido para otimizar o processamento em tempo real.

2.6 Resposta à Banda de Transição

- **Definição**: A banda de transição é a região entre a banda de passagem e a banda de rejeição. Um filtro com uma banda de transição estreita tem uma resposta mais rápida, mas é mais difícil de implementar com eficiência.

- **Como definir**: Depende da ordem do filtro e do tipo de janela utilizada.

2.7 Atenuação na Banda de Rejeição

- **Definição**: A atenuação na banda de rejeição refere-se a quanto as frequências indesejadas são atenuadas após a frequência de corte.
- **Como definir**: A janela utilizada e a ordem do filtro influenciam a atenuação.

3 Modelagem do Filtro no MATLAB

Script MATLAB:

```
% Limpar variáveis e figuras
clear; close all; clc;

% Parâmetros de simulação
Fs = 1000;          % Taxa de amostragem (1 kHz)
T = 1/Fs;           % Período de amostragem
L = 1000;           % Número de amostras (duração da simulação)
t = (0:L-1)*T;      % Vetor de tempo

% Gerar uma senoide composta por duas frequências: 50 Hz e 300 Hz
f1 = 50;            % Frequência baixa (50 Hz)
f2 = 300;           % Frequência alta (300 Hz)
x = sin(2*pi*f1*t) + sin(2*pi*f2*t);  % Sinal de entrada (soma de senos)

% Projetar o filtro FIR passa-baixa
fc = 100;           % Frequência de corte (100 Hz)
n = 29;             % Ordem do filtro FIR (número de coeficientes -1)
```

```
b = fir1(n, fc/(Fs/2));  % Coeficientes do filtro FIR passa-baixa

% Aplicar o filtro ao sinal
y = filter(b, 1, x);

% Plotar o sinal original e o sinal filtrado
figure;
subplot(2,1,1);
plot(t, x);
title('Sinal de entrada (senoide composta 50 Hz + 300 Hz)');
xlabel('Tempo (s)');
ylabel('Amplitude');

subplot(2,1,2);
plot(t, y);
title('Sinal filtrado (passa-baixa com fc = 100 Hz)');
xlabel('Tempo (s)');
ylabel('Amplitude');

% Exibir a resposta em frequência do filtro
figure;
freqz(b, 1, 1024, Fs);
title('Resposta em frequência do filtro passa-baixa FIR');
```

4 Configurações de Hardware do Microcontrolador STM32F407VET6

Nesta seção, será descrito como o microcontrolador **STM32F407VET6 Cortex-M4**, operando a uma frequência de 168 MHz, foi configurado para receber o sinal de entrada via ADC no pino PA7, e realizar a filtragem do sinal

utilizando o filtro passa-baixa FIR implementado. Após o processamento, o sinal filtrado é enviado via DAC para o pino PA4, com a saída controlada por DMA. As configurações de hardware para o temporizador, ADC e DAC estão detalhadas a seguir.

4.1 Configuração do TIMER2 para 1 kHz

Para que o sistema processasse os dados com uma taxa de amostragem de 1 kHz, foi necessário configurar o **TIMER2** do microcontrolador para operar a esta frequência. A frequência total de operação do microcontrolador é de 168 MHz, e os seguintes ajustes de hardware foram feitos para atingir 1 kHz:

1. O clock do temporizador foi inicialmente ajustado para **30 MHz**.
2. Esse clock foi dividido por **300** para reduzir a frequência do temporizador.
3. Em seguida, foi feita uma nova divisão por **100**, resultando na frequência de **1 kHz**.

O **TIMER2** foi configurado para gerar eventos de atualização a cada 1 ms, funcionando como o clock mestre para sincronizar as operações de aquisição e saída dos dados via ADC e DAC. Na figura 2, é possível ver as configurações realizadas.

Na configuração do TIMER2, foi selecionado o item **"Update Event"** na opção **"Trigger Event Selection"**, de forma que ele gere os eventos de gatilho para os periféricos dependentes.

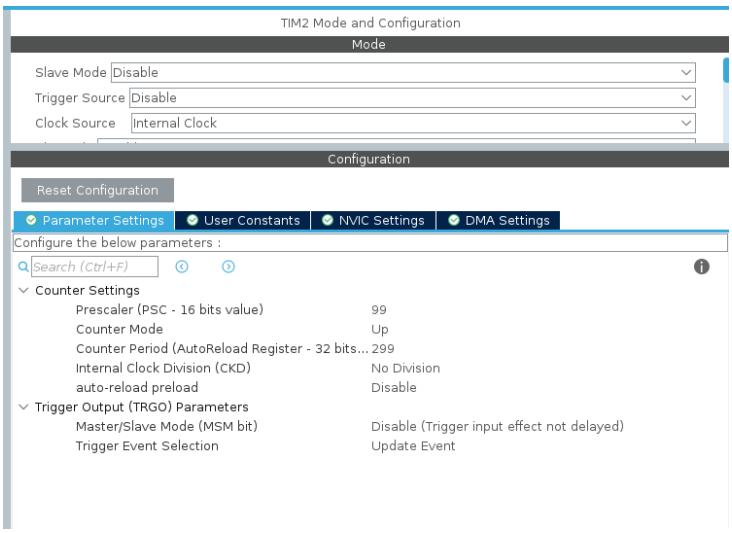

Figura 2: Configuração do TIMER2 para operação a 1 kHz

4.2 Configuração do ADC para Recebimento do Sinal de Entrada

O sinal de entrada, composto por uma soma de frequências de 50 Hz e 300 Hz, é recebido via o **ADC** de 12 bits do microcontrolador. As configurações para o ADC foram feitas da seguinte maneira:

- O **ADC1** foi configurado para operar com DMA (**Direct Memory Access**) para transferir automaticamente os dados amostrados para um buffer de memória.
- O DMA foi configurado para operar em modo circular, permitindo a coleta contínua de dados, sem a necessidade de reinicialização manual.
- A resolução de 12 bits foi utilizada para garantir a precisão na aquisição do sinal de entrada.
- O pino utilizado para receber o sinal de entrada foi o **pino PA7**, configurado como entrada analógica no **ADC1**.

- No item **"ADC_regular_ConversionMode"**, o **"External Trigger Conversion Mode"** foi selecionado como **TIMER2 Trigger Out Event**, garantindo o sincronismo com o mesmo temporizador usado para o DAC.

- O modo **DMA Circular** foi utilizado, similar ao DAC, para garantir uma aquisição contínua e automatizada. O **data width** do DMA foi configurado como **word** (32 bits), e a opção **"DMA Continuous Request"** foi habilitada.

Essas configurações asseguram uma operação sincronizada entre o ADC, o TIMER2 e o DAC, permitindo a captura precisa do sinal de entrada e sua filtragem em tempo real. Nas figuras 3 e 4 é possível ver as configurações realizadas.

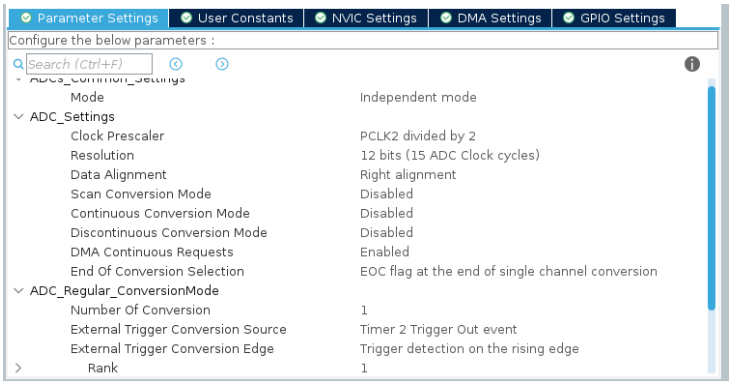

Figura 3: Configuração do ADC para recepção de sinal no pino PA7

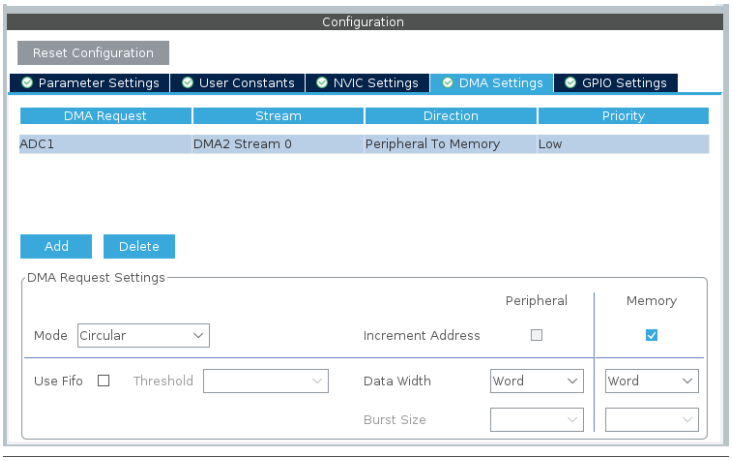

Figura 4: Configuração do DMA para o ADC em modo circular

4.3 Configuração do DAC para Geração do Sinal Filtrado

O sinal de saída filtrado é gerado pelo **DAC1** do microcontrolador, com o pino **PA4** configurado como saída analógica. As seguintes configurações foram realizadas para o DAC:

- O canal **DACout1** foi selecionado para operar no **pino PA4**.
- O **output buffer** foi habilitado para garantir que a saída fosse estável.
- O **trigger** foi configurado como **TIMER2 Trigger Out Event**, de forma que o TIMER2 controlasse a taxa de amostragem do DAC.
- O modo de geração de onda foi configurado para **Triangle Wave Generation**.
- A amplitude máxima da onda triangular foi ajustada para **15**, permitindo a geração de uma forma de onda suave no DAC.

Essas configurações garantiram que o DAC fosse capaz de gerar uma saída contínua, com o DMA configurado para operar no modo circular. O **data**

width do DMA foi configurado como **word** (32 bits), assegurando a transmissão eficiente dos dados filtrados. Nas figuras 5 e 6 é possível ver as configurações realizadas.

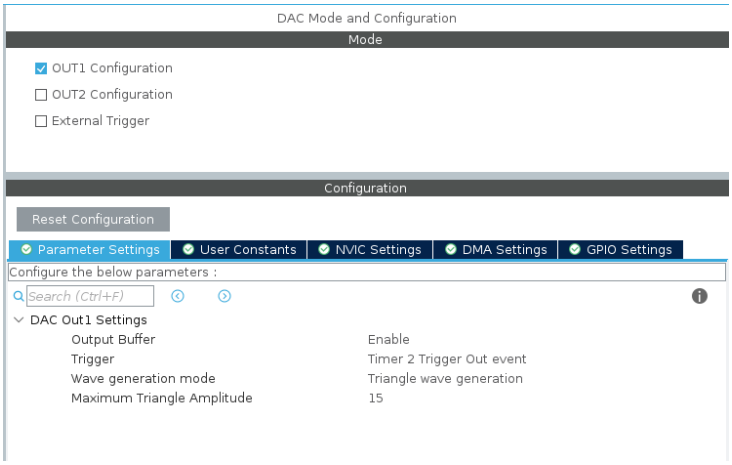

Figura 5: Configuração do DAC para geração de sinal no pino PA4

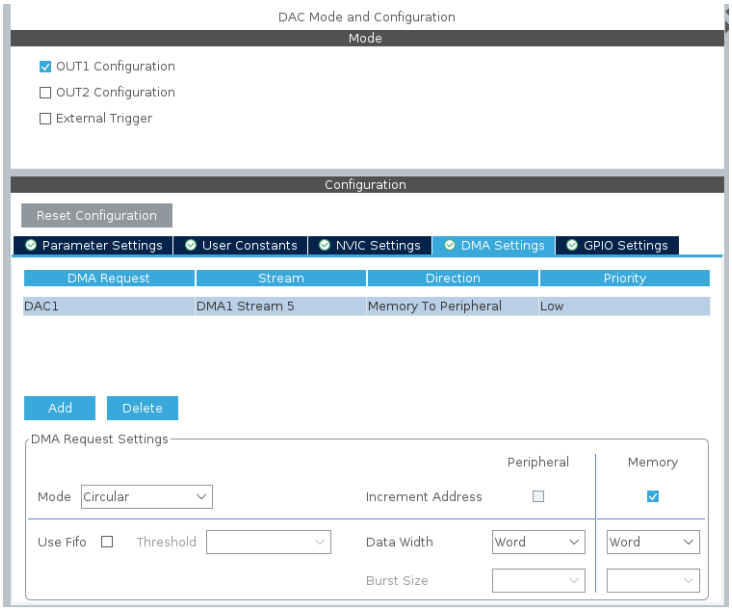

Figura 6: Configuração do DMA para o DAC em modo circular

4.4 Fluxo de Dados Entre ADC e DAC

O sistema foi projetado para que, após o ADC capturar o sinal de entrada (com 300 Hz de ruído), os dados fossem processados em blocos utilizando a biblioteca CMSIS-DSP para aplicar o filtro FIR passa-baixa. O DMA transfere os dados amostrados do ADC diretamente para a memória, e a função de callback do ADC é chamada assim que o bloco de dados é recebido. Dentro do callback, os dados são filtrados, e o sinal processado é enviado ao DAC para ser convertido em uma saída analógica.

Os dados filtrados são então enviados de volta ao DAC em blocos, e o DAC gera a saída no pino PA4 com a resolução de 12 bits. Isso cria uma representação analógica contínua do sinal filtrado, que pode ser observada por um osciloscópio ou outro dispositivo de medição.

5 Biblioteca CMSIS-DSP

A **CMSIS-DSP (Cortex Microcontroller Software Interface Standard - Digital Signal Processing)** é uma biblioteca desenvolvida pela ARM que oferece um conjunto de funções otimizadas de processamento digital de sinais para dispositivos baseados nos processadores **Cortex-M** e **Cortex-A**. Essa biblioteca é uma parte fundamental da **CMSIS (Cortex Microcontroller Software Interface Standard)**, que visa padronizar a interface entre o hardware e o software em microcontroladores ARM.

A biblioteca **CMSIS-DSP** foi projetada para oferecer desempenho eficiente em processadores que não possuem unidades de ponto flutuante (como o Cortex-M3), mas também tira proveito de processadores com unidades de ponto flutuante (como o Cortex-M4 e Cortex-M7). Essa biblioteca é amplamente utilizada em sistemas embarcados, oferecendo várias funções prontas para uso que facilitam a implementação de algoritmos de processamento de sinal.

5.1 Principais Funções da CMSIS-DSP

A biblioteca CMSIS-DSP contém diversas funções otimizadas que cobrem áreas essenciais de processamento de sinal, como:

- **Filtros Digitais**: Funções para implementar filtros FIR, IIR, interpolação e decimação.
- **Transformadas**: Funções para cálculos de FFT (Fast Fourier Transform) e DCT (Discrete Cosine Transform).
- **Operações Matemáticas**: Funções para operações com vetores, matrizes e operações aritméticas.
- **Estatísticas**: Cálculo de média, variância, desvio padrão, máximo e mínimo de conjuntos de dados.
- **Funções de Controle**: Funções para cálculos de PID (Proporcional-Integral-Derivativo) e filtragem adaptativa.
- **Funções de Conversão**: Conversão entre números de ponto flutuante e ponto fixo, e vice-versa.

5.2 Filtro FIR (Finite Impulse Response) com CMSIS-DSP

O **Filtro de Resposta ao Impulso Finito (FIR)** é um tipo de filtro digital utilizado para diversas aplicações em processamento de sinais. A biblioteca **CMSIS-DSP** oferece funções otimizadas para o processamento de filtros FIR, que podem ser implementadas em microcontroladores da família **Cortex-M**, como o STM32F407VET6.

Para o presente projeto, a função `arm_fir_init_f32()` foi utilizada para inicializar o filtro FIR com os coeficientes definidos no MATLAB. A função

`arm_fir_f32()` é responsável por aplicar o filtro sobre os blocos de dados amostrados.

As principais funções relacionadas ao filtro FIR na biblioteca CMSIS-DSP são:

- **arm_fir_init_f32()**:
  - Inicializa a estrutura do filtro FIR de ponto flutuante.
  - **Parâmetros**: Número de coeficientes (`numTaps`), coeficientes do filtro (`pCoeffs`), e o estado do filtro (`pState`).
- **arm_fir_f32()**:
  - Processa os blocos de dados aplicando o filtro FIR.
  - **Parâmetros**: Ponteiro para a estrutura FIR (`S`), ponteiro para o sinal de entrada (`pSrc`), ponteiro para o sinal de saída (`pDst`), e o tamanho do bloco (`blockSize`).

Abaixo está um exemplo básico de inicialização do filtro FIR, onde:

- `numTaps` representa o número de coeficientes do filtro.
- `pState` aponta para o buffer de estado do filtro.
- `pCoeffs` é o buffer que contém os coeficientes do filtro.

```
void arm_fir_init_f32 (
    arm_fir_instance_f32 * S,   // Estrutura do filtro
    uint16_t numTaps,           // Número de coeficientes do filtro
    const float32_t * pCoeffs,  // Ponteiro para os coeficientes
    float32_t * pState,         // Ponteiro para o estado do filtro
    uint32_t blockSize          // Tamanho do bloco de amostras
);
```

5.3 Aplicação do Filtro FIR no Processamento do Sinal

A aplicação deste filtro FIR foi essencial para atenuar o ruído de alta frequência (300 Hz) e permitir a passagem da frequência de interesse (50 Hz), conforme a modelagem desenvolvida no MATLAB. O algoritmo do filtro FIR é baseado em uma sequência de operações de multiplicação-acumulação (MAC), onde cada coeficiente do filtro $b[n]$ é multiplicado por uma variável de estado que corresponde a uma amostra de entrada anterior $x[n]$. O resultado da filtragem é dado pela seguinte equação:

$$y[n] = b[0]\cdot x[n]+b[1]\cdot x[n-1]+b[2]\cdot x[n-2]+\cdots+b[\text{numTaps}-1]\cdot x[n-\text{numTaps}+1]$$

Esse ciclo contínuo de filtragem é repetido para cada bloco de dados recebido via DMA no *callback* do ADC. Após o processamento, o sinal filtrado é enviado ao DAC para ser convertido em uma saída analógica. A figura 7 ilustra o fluxo do algoritmo.

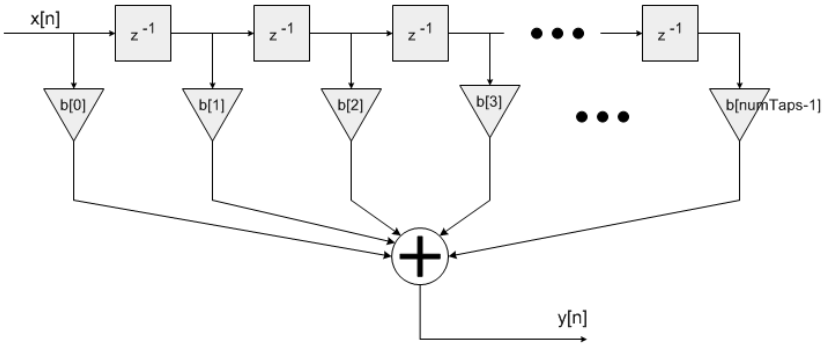

Figura 7: Fluxo de processamento do algoritmo filtro FIR aplicado ao sinal de entrada

5.4 Utilização do CMSIS-DSP no Projeto

No presente trabalho, a biblioteca **CMSIS-DSP** foi utilizada para implementar um filtro FIR passa-baixa, que tem como objetivo atenuar as frequências superiores a 100 Hz, preservando as frequências mais baixas no sinal de entrada.

A função `arm_fir_init_f32` foi utilizada para inicializar o filtro com os coeficientes gerados no MATLAB. Em seguida, a função `arm_fir_f32` foi utilizada para processar o sinal de entrada em blocos. Essas funções são altamente otimizadas para o processador Cortex-M4, garantindo que o filtro opere de maneira eficiente, mesmo em tempo real.

5.5 Vantagens da Utilização da CMSIS-DSP

A biblioteca CMSIS-DSP oferece várias vantagens ao se implementar algoritmos de processamento de sinal, entre elas:

- **Desempenho Otimizado**: As funções da CMSIS-DSP são otimizadas para tirar proveito das instruções SIMD (*Single Instruction Multiple Data*) e da unidade de ponto flutuante (FPU) presente no Cortex-M4.
- **Facilidade de Uso**: A biblioteca disponibiliza funções prontas, reduzindo significativamente o tempo de desenvolvimento.
- **Padronização**: A CMSIS-DSP é uma biblioteca amplamente aceita e padronizada, o que facilita a interoperabilidade entre diferentes projetos e a reutilização de código.

Com a utilização da CMSIS-DSP, foi possível integrar de maneira eficiente as operações de filtragem digital, permitindo que o filtro FIR passa-baixa processasse o sinal em tempo real no microcontrolador **STM32F407VET6**.

6 Configuração de Software para Processamento do Sinal

O código a seguir descreve a implementação do sistema de filtragem digital de sinal utilizando o microcontrolador STM32F407VET6. O objetivo é receber um sinal de entrada ruidoso via **ADC**, aplicar um filtro **FIR** passa-baixa utilizando a biblioteca CMSIS-DSP e, por fim, enviar o sinal filtrado via **DAC**. O **DMA** é utilizado tanto no **ADC** quanto no **DAC** para garantir que os dados sejam processados de forma eficiente, sem intervenção da CPU.

6.1 Descrição do Código Principal

O código abaixo mostra a configuração e o fluxo principal da aplicação implementada no microcontrolador. As principais tarefas incluem a inicialização do sistema, configuração de periféricos e processamento do sinal:

```
/* Includes */
#include "main.h"
#include "arm_math.h" // Inclus o das fun  es da
    biblioteca CMSIS-DSP

/* Private define
    ------------------------------------------------------------
    */
#define NUM_SAMPLES 1000  // N mero total de amostras
#define BLOCK_SIZE 256     // Tamanho do bloco de
    processamento
#define NUM_TAPS 30 // N mero de coeficientes do filtro
#define DAC_MAX_VALUE 4096  // Resolu  o do DAC de 12 bits


/* Private variables
    ---------------------------------------------------------
    */
ADC_HandleTypeDef hadc1;
```

```
14 DMA_HandleTypeDef hdma_adc1;
15
16 DAC_HandleTypeDef hdac;
17 DMA_HandleTypeDef hdma_dac1;
18
19 TIM_HandleTypeDef htim2;
20
21 /* Buffer de estado do filtro FIR (armazenamento
       intermedi rio para o filtro)  */
22 float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1];
23
24 /* Coeficientes do filtro FIR (passa-baixa projetado no
       MATLAB) */
25 float32_t firCoeffs32[NUM_TAPS] = {
26     0.000542, 0.001727, 0.003113, 0.003883, 0.002264,
       -0.003353,
27     -0.012604, -0.021814, -0.024289, -0.012675, 0.017408,
       0.064163,
28     0.118682, 0.167181, 0.195770, 0.195770, 0.167181,
       0.118682,
29     0.064163, 0.017408, -0.012675, -0.024289, -0.021814,
       -0.012604,
30     -0.003353, 0.002264, 0.003883, 0.003113, 0.001727,
       0.000542
31 };
32
33 /* Buffer para armazenar amostras de entrada (sinal ruidoso)
       e sa da (sinal filtrado)  */
34 float32_t inputBuffer[BLOCK_SIZE];
35 float32_t outputBuffer[BLOCK_SIZE];
36
37 /* Vari vel para controle de amostras */
38 uint32_t sampleIndex = 0;
39
40 /* Inst ncia do filtro FIR */
41 arm_fir_instance_f32 S;
```

```
42 /* End private variables
       -----------------------------------------------------------
       */
43
44 /* Fun   o principal */
45 int main(void) {
46   HAL_Init();  // Inicializa a HAL (Hardware Abstraction
       Layer)
47
48   SystemClock_Config();  // Configura o sistema de clock
49   MX_GPIO_Init();  // Inicializa GPIOs
50   MX_DMA_Init();   // Inicializa o DMA
51   MX_TIM2_Init();  // Inicializa o temporizador 2
52   MX_DAC_Init();   // Inicializa o DAC
53   MX_ADC1_Init();  // Inicializa o ADC
54
55   /* Inicializando o filtro FIR */
56   arm_fir_init_f32(&S, NUM_TAPS, firCoeffs32, firStateF32,
       BLOCK_SIZE);
57
58   /* Iniciando o Timer e o DMA para ADC e DAC */
59   HAL_TIM_Base_Start(&htim2);  // Iniciando o temporizador
60   HAL_ADC_Start_DMA(&hadc1, (uint32_t*)inputBuffer,
       BLOCK_SIZE); // Iniciando DMA no ADC
61   HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t*)
       outputBuffer, BLOCK_SIZE, DAC_ALIGN_12B_R); // Iniciando
       DMA no DAC
62
63   /* Loop infinito */
64   while (1) {
65     // Processamento continuo
66   }
67 }
```

Listing 1: Filtro FIR

6.2 Callback do ADC e Função de Processamento

O **DMA** é configurado para operar em modo circular, e o sistema utiliza um **callback** que é chamado assim que o **DMA** conclui a transferência dos dados do ADC. Nesta função, o sinal é processado, filtrado e enviado para o DAC.

```
/* Callback chamado quando o DMA do ADC termina a convers o */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
    if (hadc->Instance == ADC1) {
        // Processando as amostras atuais com o filtro FIR
        process_signal();

        // Enviando o sinal filtrado ao DAC
        for (int i = 0; i < BLOCK_SIZE; i++) {
            uint32_t dac_output = (uint32_t)((outputBuffer[i] + 1.0f) * (DAC_MAX_VALUE / 2)); // Ajusta o sinal para o DAC (0-4095)
            HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, dac_output); // Envia para o DAC
        }
    }
}

/* Fun   o de processamento do sinal */
void process_signal(void) {
    arm_fir_f32(&S, inputBuffer, outputBuffer, BLOCK_SIZE); // Aplicando o filtro FIR ao bloco de dados
}
```

Listing 2: Processamento digital

6.3 Descrição do Fluxo de Processamento

O sistema funciona da seguinte maneira:

1. O **ADC1** coleta o sinal de entrada no **pino PA7** e armazena os dados no **inputBuffer** usando o **DMA**.

2. Assim que o **DMA** completa a transferência, o **callback** do **ADC** é chamado, onde o bloco de dados é processado pelo filtro FIR.

3. Após o processamento, o sinal filtrado é armazenado no **outputBuffer** e enviado para o **DAC1** para ser convertido em sinal analógico.

4. O **TIMER2** controla a taxa de amostragem, gerando eventos de gatilho para o ADC e o DAC de forma sincronizada.

6.4 Resumo do Código de Software

Este código principal integra o filtro passa-baixa desenvolvido no MATLAB com o hardware do STM32F407VET6. Utilizando o ADC para adquirir o sinal ruidoso e o DAC para gerar o sinal filtrado, o sistema implementa uma solução de processamento digital de sinais em tempo real.

7 Resultados

Como apresentado na seção 3, foi implementado a modelagem de um filtro passa-baixa para atenuação de frequências a partir de 100Hz. Para efeitos de teste, foi gerada uma onda senóide composta por frequências de 50 e 300 Hz e em seguida a aplicação do filtro FIR passa baixa com frequência de corte de 100Hz e ploatagem dos sinais de entrada e saída, apresentados nas figuras 8 e 9, respectivamente.

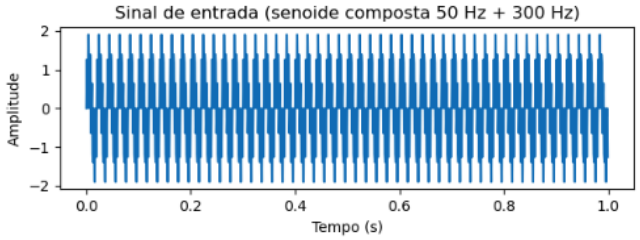

Figura 8: Sinal original

Figura 9: Sinal filtrado

A taxa de amostragem de 1Hz dificultou a visualização, foi utilizada a função *xlim* da biblioteca *matplotlib* para ajustar a visualização do sinal, focando apenas nos primeiros 0.1 segundos. Assim, foi possivel visualizar melhor o sinal, podendo observar que ele se tornou mais limpo após a filtragem.

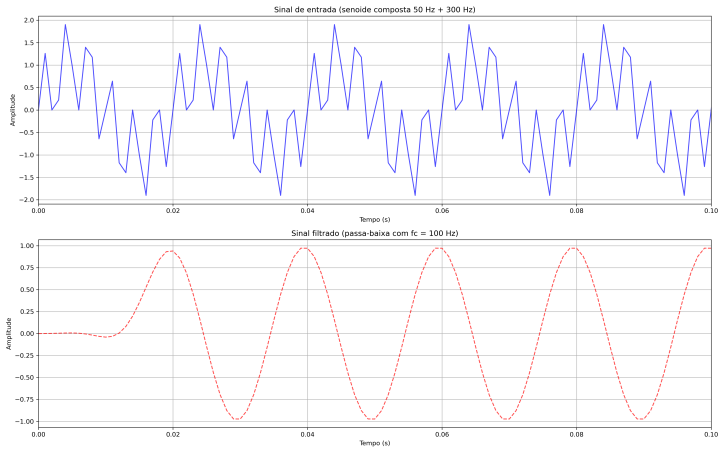

Figura 10: Visualização dos primeiros 0.1s do sinal filtrado

Para testar a implementação utilizando a placa, o mesmo teste foi realizado. Para isso, foi gerado um sinal de 300Hz utilizando o gerador de função e enviado para a placa por meio dos pinos analógicos. Após o processamento, o sinal de saída foi obtido por meio da porta analógico e visualizado com o osciloscópio(Figura 11)

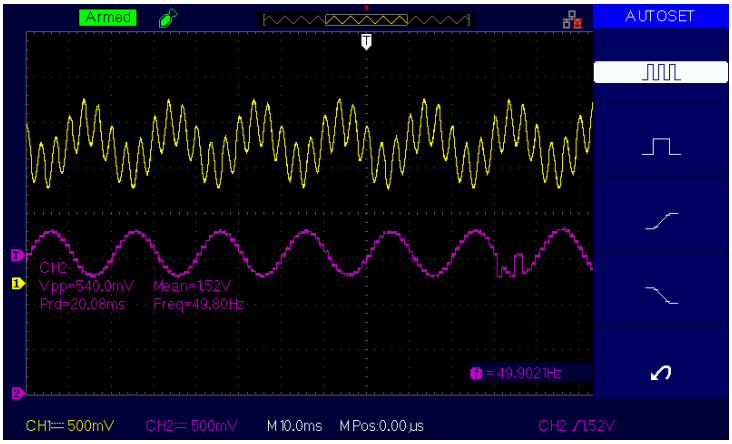

Figura 11: Microcontrolador STM32F407VET6 Cortex-M4

Para verificar a eficiência do filtro, foi realizado um outro teste, dessa vez gerando um sinal ruidoso por meio de uma função do gerador de função,

aplicando-o a um sinal de 50Hz e em seguida realizada a filtragem. Para visulização foi feito o uso do osciloscópio, onde temos o sinal de ruidoso(canal amarelo) e o sinal após a filtragem(canal roxo).

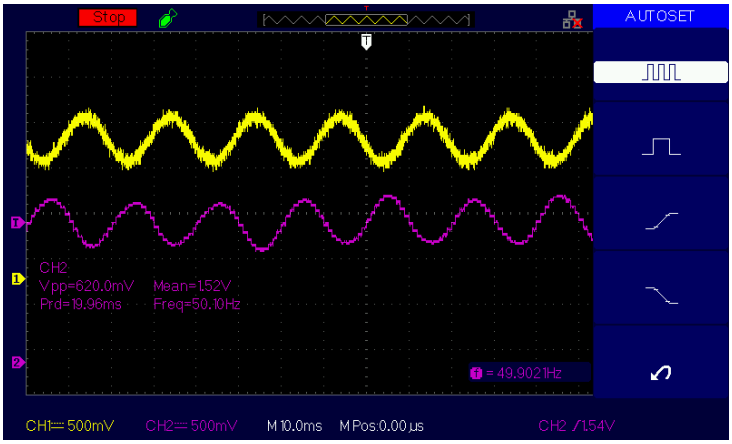

Figura 12:

A partir dos resultados, pode-se notar o bom desempenho do filtro, com a atenuação das frequências indesejadas, tornando o sinal "mais limpo". Observa-se também um defassagem do sinal, ambos estão fora de sincronia, indicando a ocorrência de um deslocamento. Isso se deve ao tempo de processamento do sinal, levando-o a estar "atrasado"em relação ao sinal original.

8 Conclusão

A modelagem do filtro passa-baixa em MATLAB fornece uma base sólida para implementação em sistemas embarcados. A transição para ambientes de hardware como microcontroladores deve levar em consideração os mesmos parâmetros definidos na modelagem, ajustando-os conforme a capacidade de processamento do dispositivo embarcado.

O microcontrolador STM32F407VET6, com suporte a DSP e periféricos

ADC e DAC de 12 bits, é ideal para a implementação de filtragem digital de sinais em tempo real. O uso do *STM32CubeIDE* facilita a integração de DMA e outros recursos para uma implementação eficiente.

Referências

[1] Marcelo Basílio Joaquim. Processamento digital de sinais. *São Carlos*, 2010.