

DSP – FILTRO DIGITAL

Autor: Lucas Daudt Franck

GitHub: github.com/LDFranck/DSP-with-CMSIS

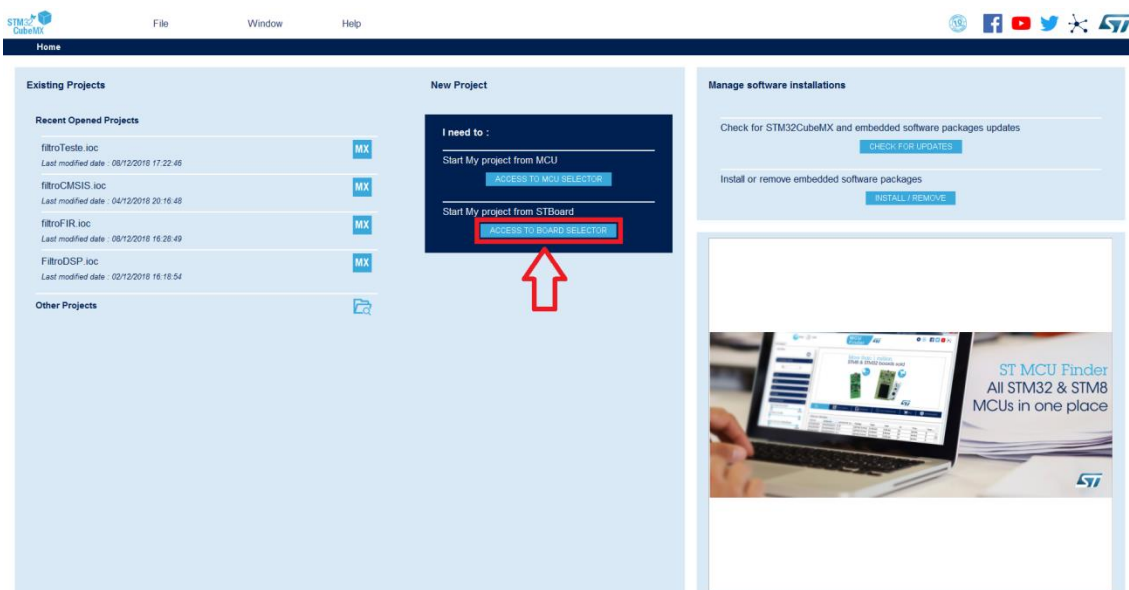
INTRODUÇÃO:

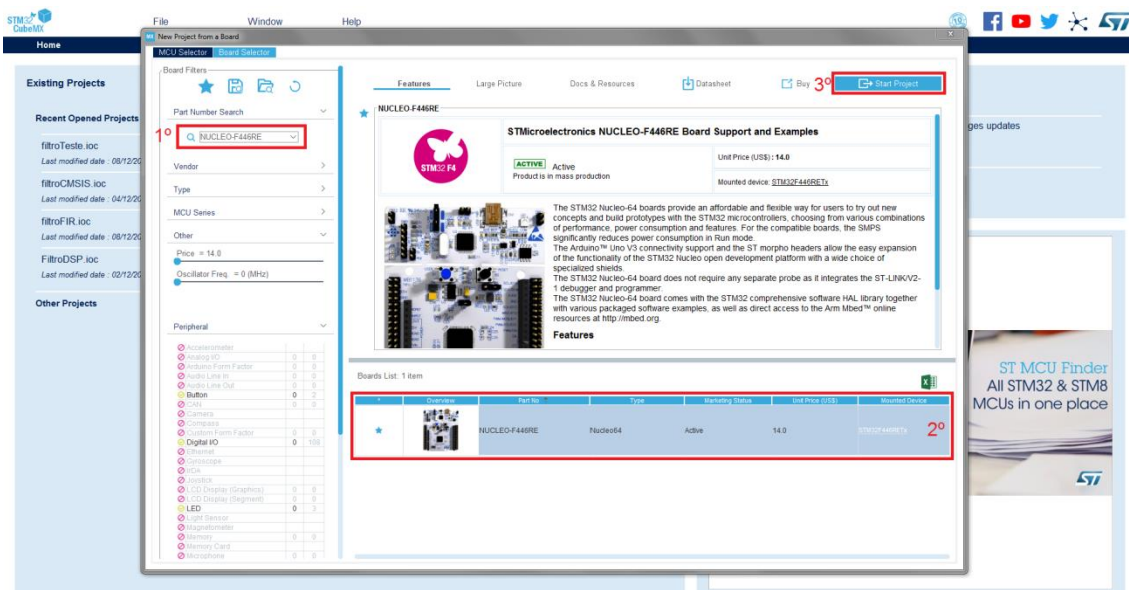
Este documento é um breve tutorial a respeito da implementação de filtros digitais na placa de desenvolvimento STM32F446RE utilizando a CMSIS. O projeto dos filtros pode ser realizado no site www.micromodeler.com/dsp/ ou em CADs matemáticos como o MATLAB e o OCTAVE. Um código exemplo está disponibilizado no GitHub do autor.

INICIANDO UM NOVO PROJETO NO STM32CUBEMX:

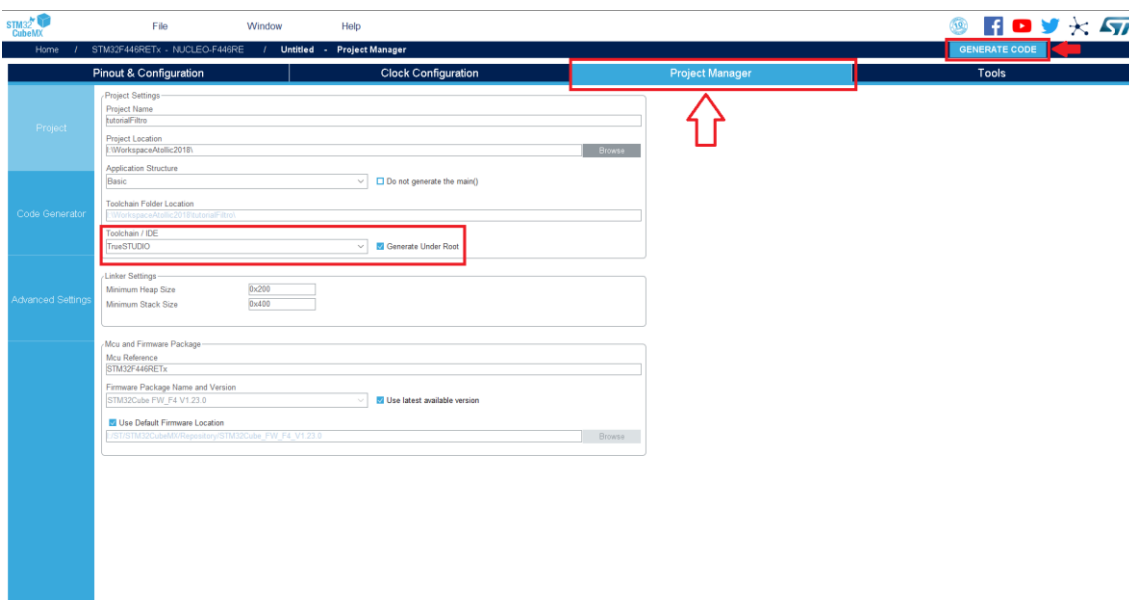
Antes de realizar o projeto dos filtros, é necessário preparar o projeto para que seja possível a utilização das funções da CMSIS. Neste tutorial será utilizado a IDE TrueSTUDIO – Atollic e a versão 5.0.0 do STM32CubeMX.

A primeira etapa consiste em gerar os arquivos base do projeto através do STM32CubeMX. Para isto, inicializar o programa e criar um novo projeto. Para facilitar a utilização do filtro, é interessante que a placa de desenvolvimento utilizada tenha um conversor digital-analógico. Neste tutorial será utilizada a placa de desenvolvimento STM32F446RE.

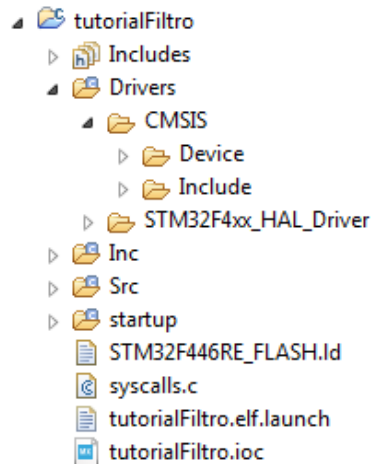




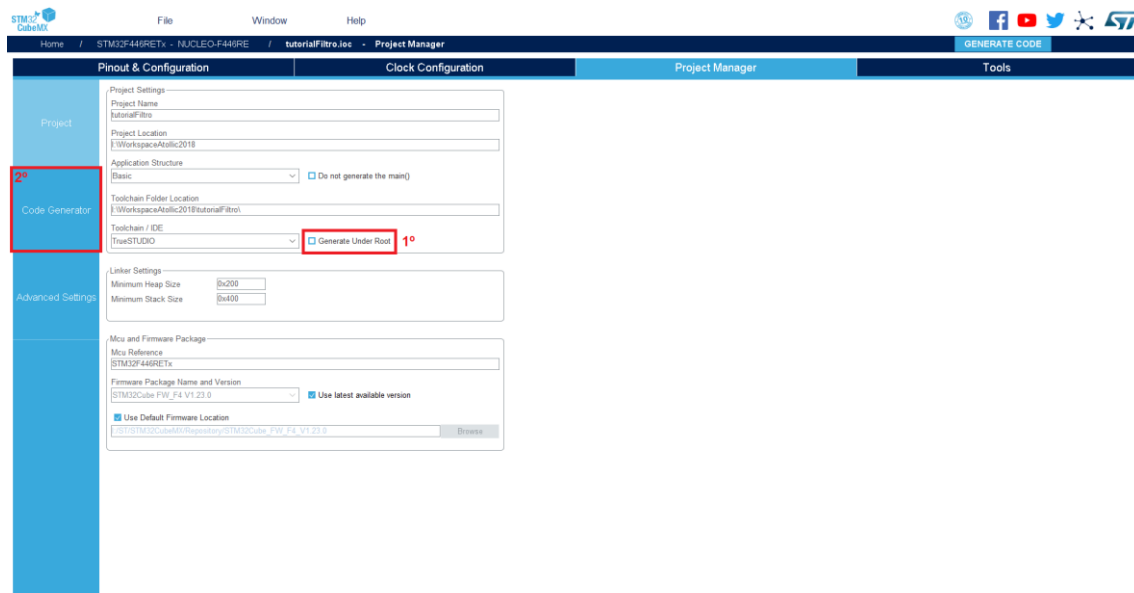
Após criar o projeto e configurar os periféricos, é necessário gerar os arquivos do projeto. Para isso, clicar na aba “Project Manager” e preencher os campos conforme a figura abaixo. É necessário escolher corretamente a Toolchain / IDE a ser utilizada. Para a utilização do TrueSTUDIO – Atollic, escolha “TrueSTUDIO” e confirme que a opção “Generate Under Root” esteja marcada. Com isto feito, gerar o código clicando na opção “Generate Code” no canto superior direito.



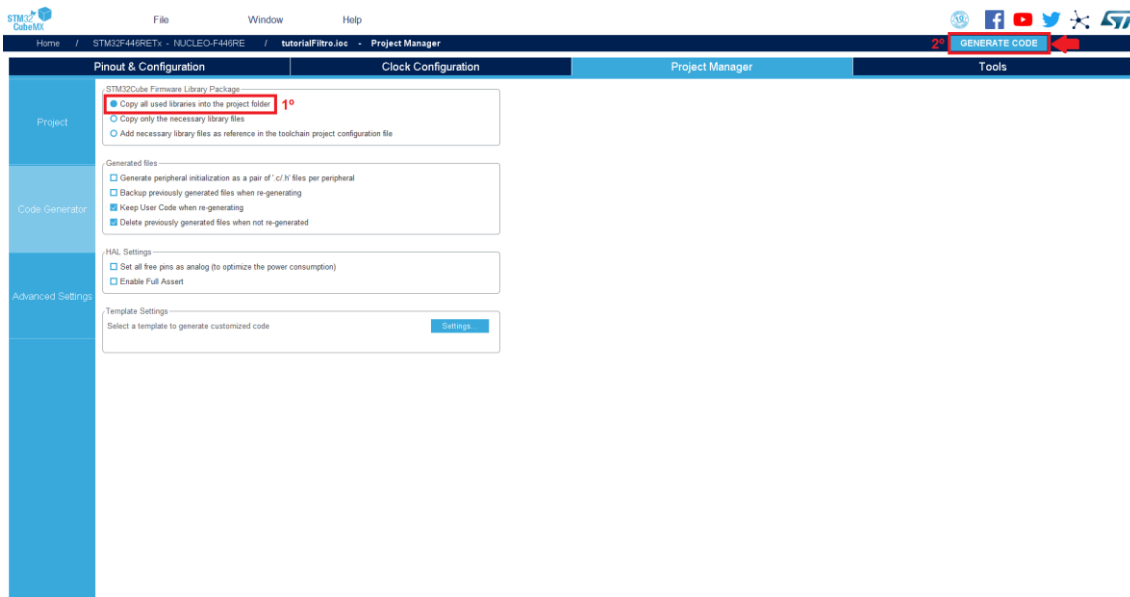
Após essa etapa, importar o projeto gerado para o TrueSTUDIO – Atollic. É interessante notar que os arquivos “.c” da CMSIS não foram gerados. Você deve ter um projeto semelhante à figura abaixo.



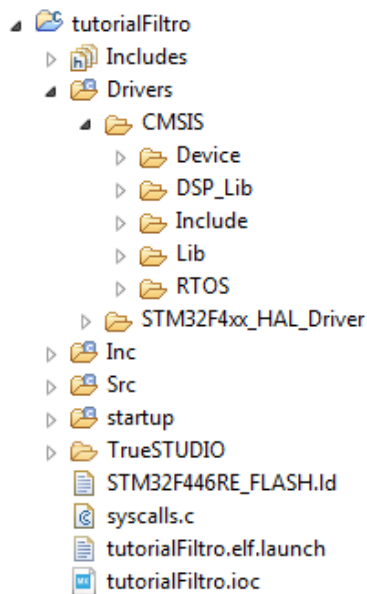
Para gerar os arquivos “.c” da CMSIS é necessário retornar ao STM32CubeMX do projeto e desmarcar a opção “Generate Under Root”. Após isto feito, clicar na aba “Code Generator” localizada na esquerda.



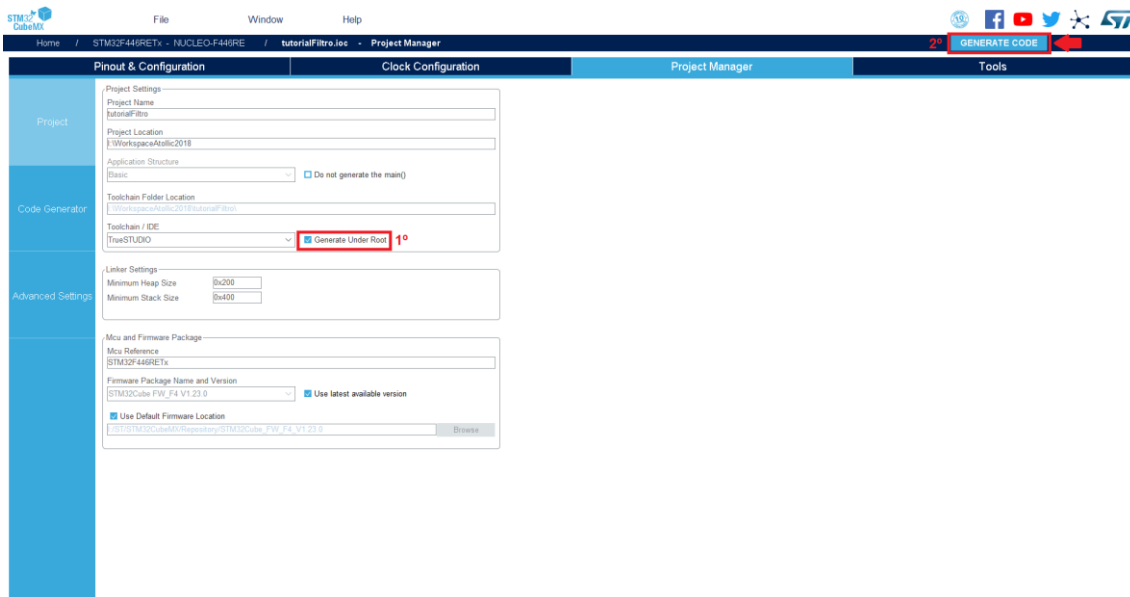
Na aba “Code Generator”, marcar a opção “Copy all used libraries into the project folder” e gerar novamente o projeto clicando no botão “Generate Code” no canto superior direito.



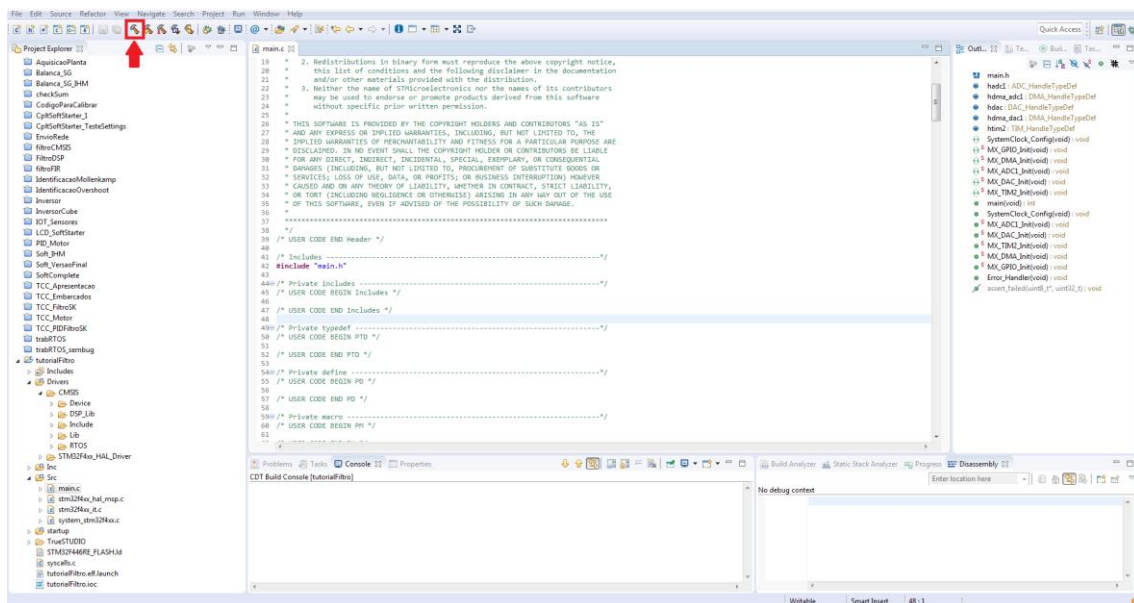
Agora o seu projeto deve estar semelhante à figura abaixo.



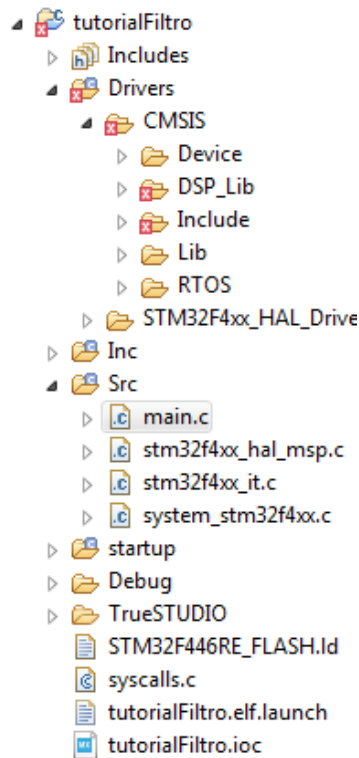
Agora será necessário retornar ao STM32CubeMX do projeto e gerar novamente os arquivos do projeto com a opção “Generate Under Root” marcada. É interessante notar que os arquivos da CMSIS não serão apagados do seu projeto.



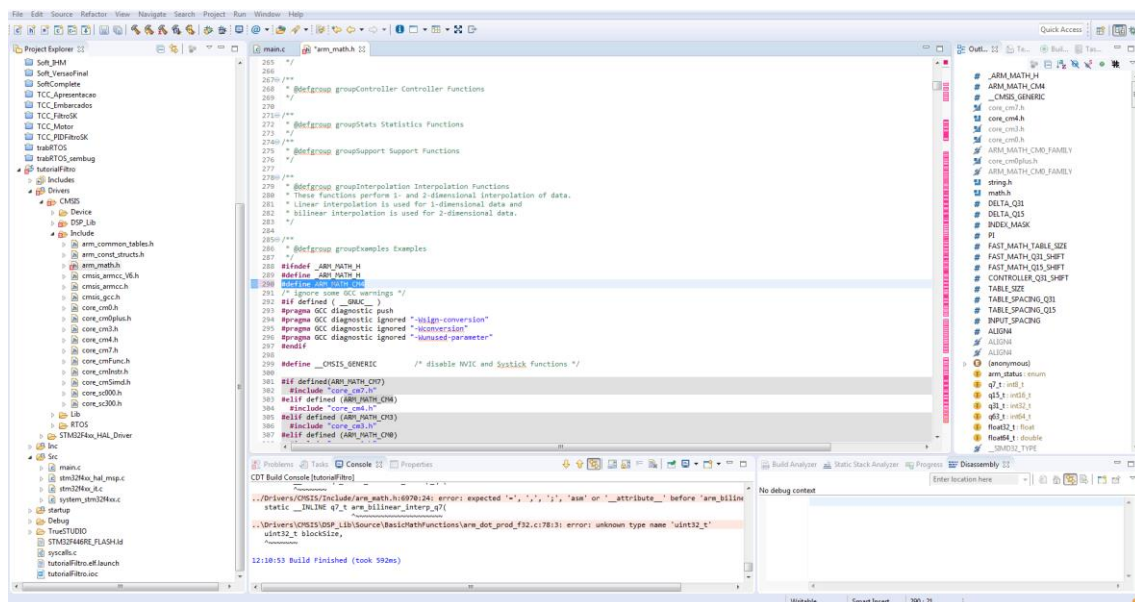
A próxima etapa consiste em dar um “build” no projeto. Para isto, entrar no arquivo “main.c” e clicar sobre o ícone do martelo localizado na barra de ferramentas superior.



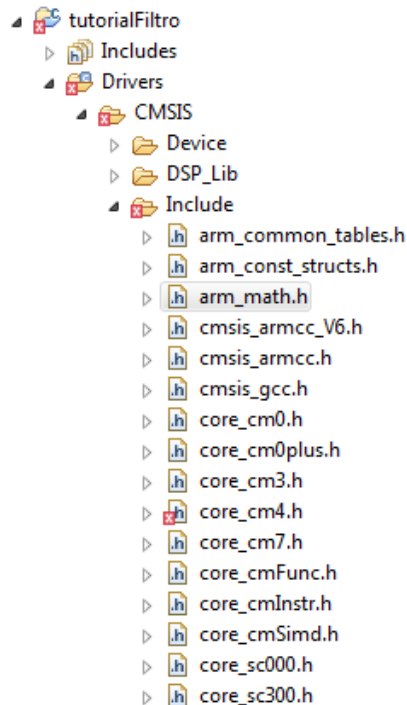
Com isso feito, o seu projeto deve apresentar erros em algumas pastas como na figura abaixo.



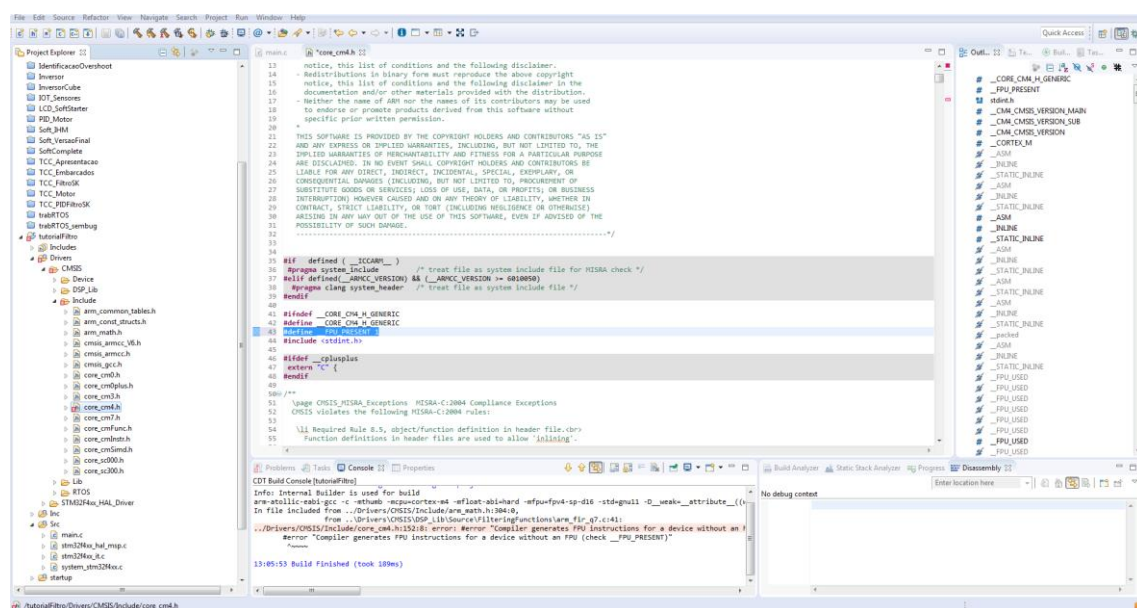
O primeiro erro a ser corrigido encontra-se no arquivo “arm_math.h” na pasta “Include”. Para corrigir este erro, é necessário incluir na linha 290 do arquivo “arm_math.h” a seguinte instrução: “#define ARM_MATH_CM4”.



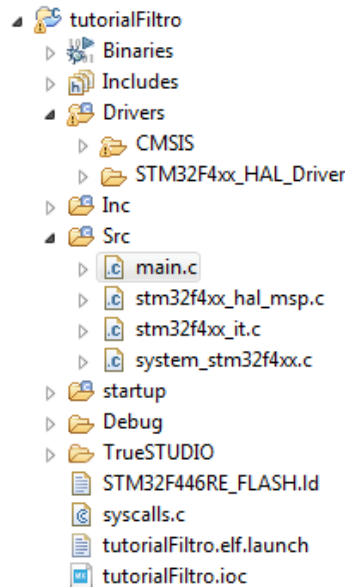
Com isso feito, salvar o arquivo “arm_math.h” e dar um build no projeto. O erro no arquivo “arm_math.h” deve desaparecer e um novo erro no arquivo “core_cm4.h” na pasta “Include” vai aparecer.



Para solucionar esse erro, é necessário incluir na linha 43 do arquivo “core_cm4.h” a seguinte instrução: “#define __FPU_PRESENT 1”.



Após completar a alteração, salvar o arquivo “core_cm4.h” e dar um build no projeto. Os erros do projeto devem desaparecer. O projeto ainda irá apresentar “warnings”, mas estes permitem fazer o debug do código.

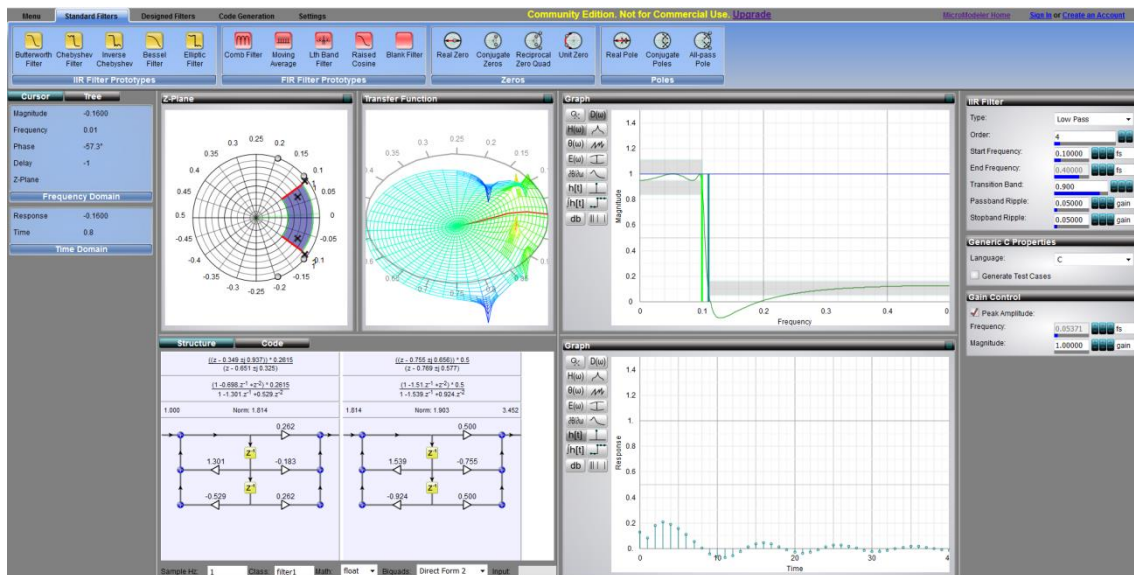
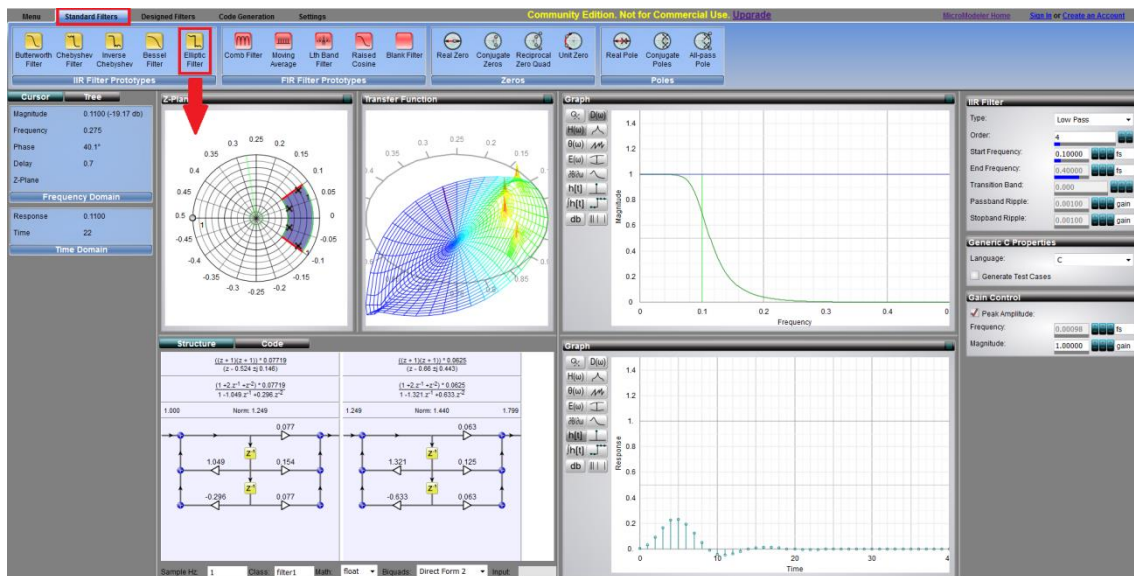


Agora o projeto se encontra pronto para a utilização das funções da CMSIS. O código exemplo já está pronto para o uso, não havendo necessidade de realizar todos estes procedimentos.

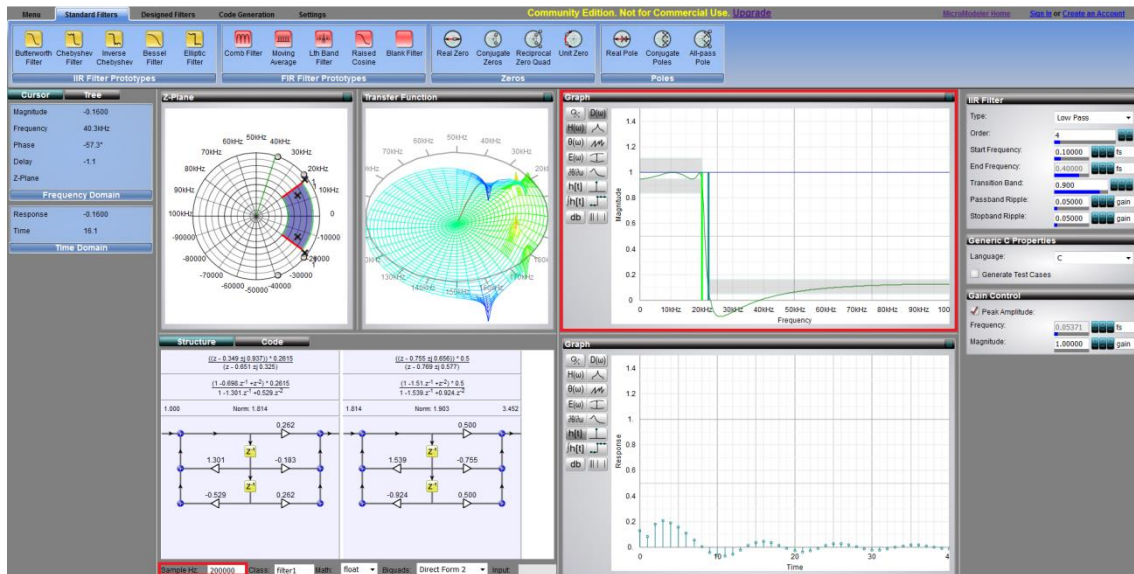
PROJETO A PARTIR DO SITE MICROMODELER:

O filtro digital pode ser facilmente projetado com o auxílio do site www.micromodeler.com/dsp/. A primeira vista o site pode parecer complexo, mas sua utilização é relativamente simples. Esta ferramenta é capaz de projetar filtros IIR (*Infinite Impulse Response*) e filtros FIR (*Finite Impulse Response*). Em sua versão gratuita, o site possibilita ao usuário criar filtros IIR de até 4ª ordem e filtros FIR de até 21ª ordem. Como exemplo, será projetado um filtro passa-baixas Elíptico de 4º ordem com uma frequência de corte de 1kHz. A frequência de amostragem adotada será de 200kHz (frequência de amostragem do código exemplo – configurada no STM32CubeMX).

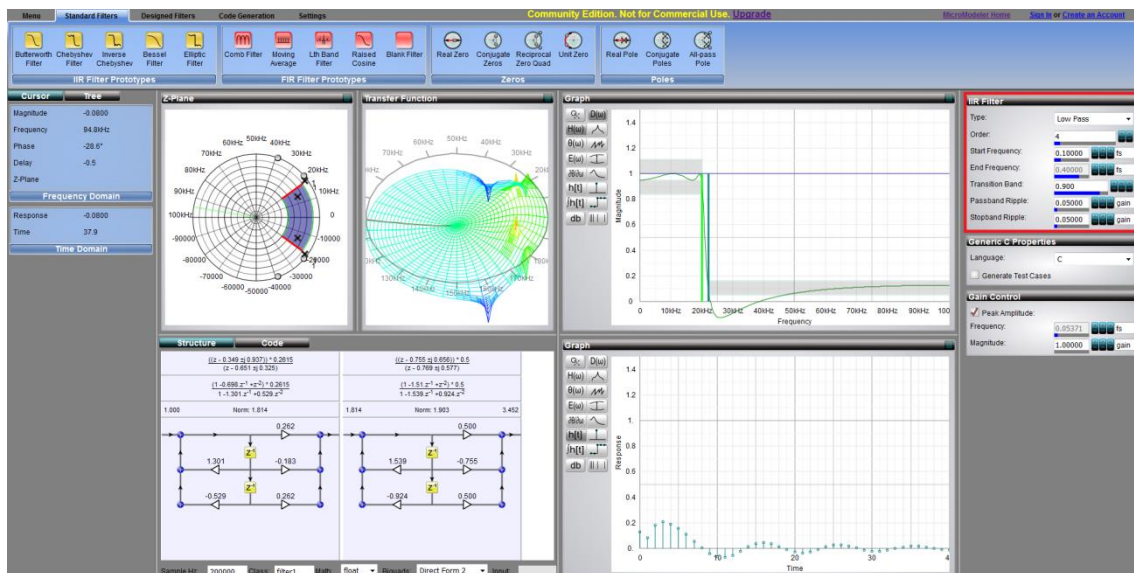
O primeiro passo é definir o tipo do filtro. Para isto, é necessário ir à aba “Standard Filters”, selecionar a opção “Elliptic Filter” e arrastá-la para a área de trabalho do site.



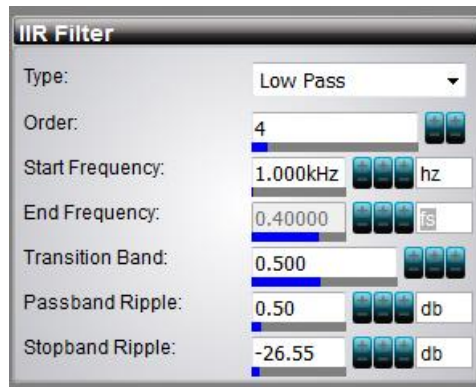
A próxima etapa consiste em alterar a frequência de amostragem para 200kHz. Para isto, alterar o valor “1” contido na opção “Sample HZ” para “200000”. Com esta alteração, as frequências do gráfico se alteram, facilitando o projeto do filtro.



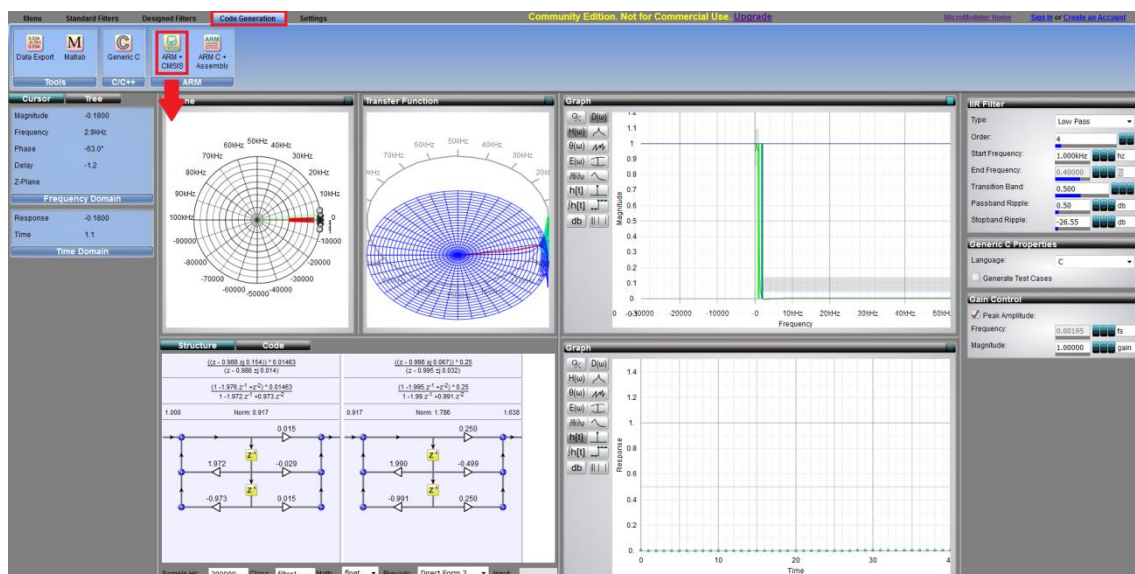
Uma vez com as frequências do filtro na escala correta, chega o momento de configurar o tipo do filtro, a frequência de corte e o ripple na banda passante. Todos esses parâmetros podem ser alterados na caixa denominada “IIR Filter”, localizada à direita da tela. Em muitos casos é conveniente mudar o tipo da unidade mostrada (ex.: ao invés de mostrar em “gain”, mostrar em “dB”). Para isto, basta clicar com o mouse sobre a caixa que indica a unidade.



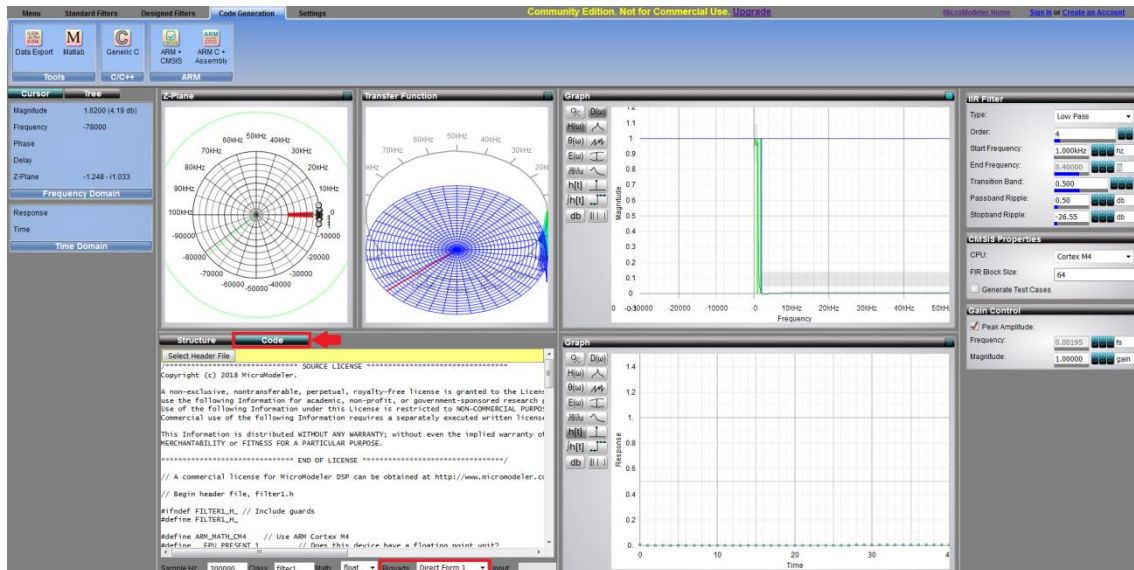
Para contemplar as especificações do filtro necessárias, as seguintes alterações foram feitas nos parâmetros do filtro.



Uma vez que o filtro foi projetado corretamente, chega o momento de gerar o código para o ARM. Para isto, ir à aba “Code Generation”, selecionar a opção “ARM + CMSIS” e arrastá-la para a área de trabalho do site.



Com isto feito, o site se encarrega de gerar o código para ser utilizado no microcontrolador. Para visualizar o código gerado, basta ir à aba “Code” e mudar a opção “Biquads” de “Direct Form 2” para “Direct Form 1”. O site gera um arquivo “.h” e um arquivo “.c” com as funções para a utilização do filtro.



O próximo passo consiste em importar o código gerado pelo site para o projeto no TrueSTUDIO – Atollic. Para isto, é necessário criar um arquivo “.h” e um “.c” com o nome da classe indicado pelo site (pode ser alterado), no caso deste exemplo é “filter1”.

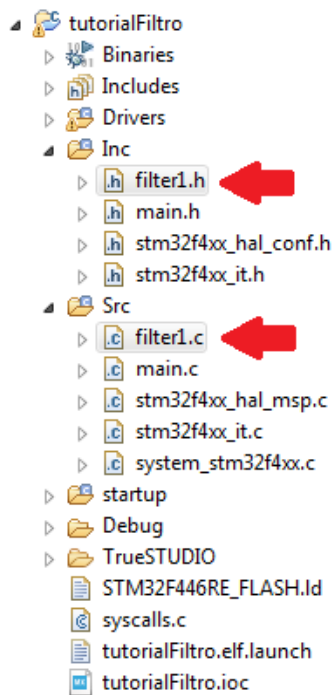
The screenshot shows the TrueSTUDIO code editor with the 'Code' tab selected. The editor displays the generated header file 'filter1.h'. The code includes a license agreement for MicroModeler DSP, dated 2018. Below the license, the code generation settings are visible: 'Sample Hz' is set to 200000, 'Class' is 'filter1', 'Math' is 'float', 'Biquads' is 'Direct Form 1', and 'Input' is 'Input'. The code includes the following preprocessor directives:

```

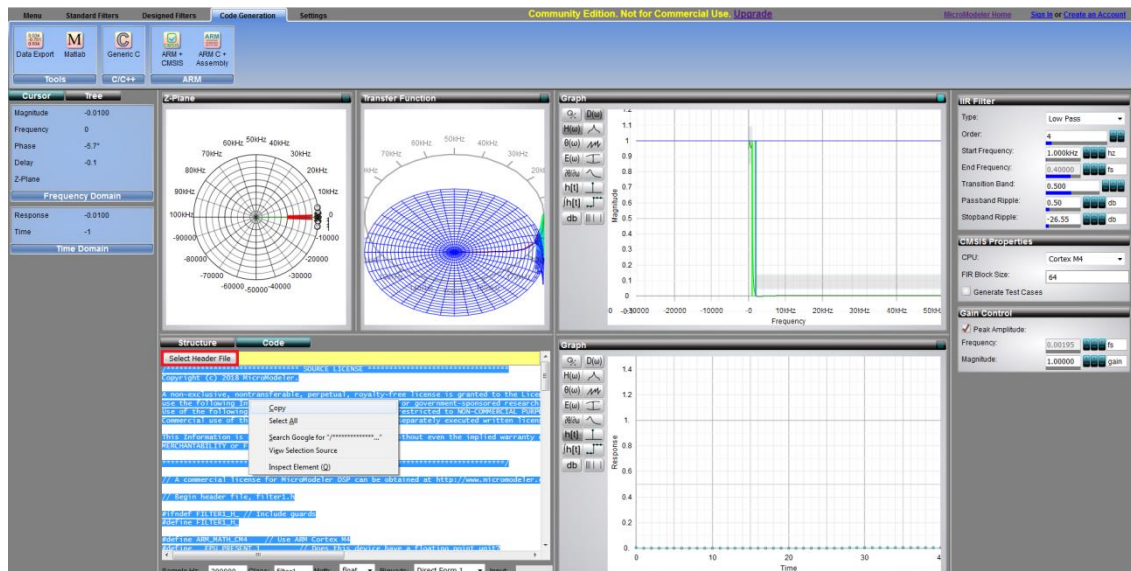
#ifndef FILTER1_H // Include guards
#define FILTER1_H

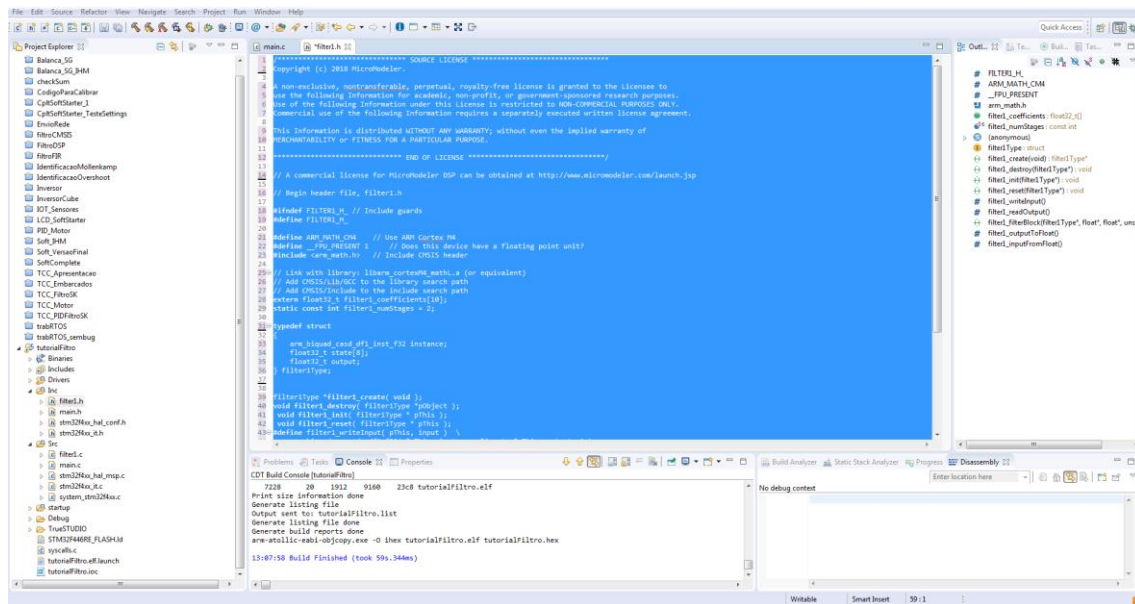
#define ARM_MATH_CM4 // Use ARM Cortex M4
#define FPU_PRESENT 1 // Does this device have a floating point unit?

```

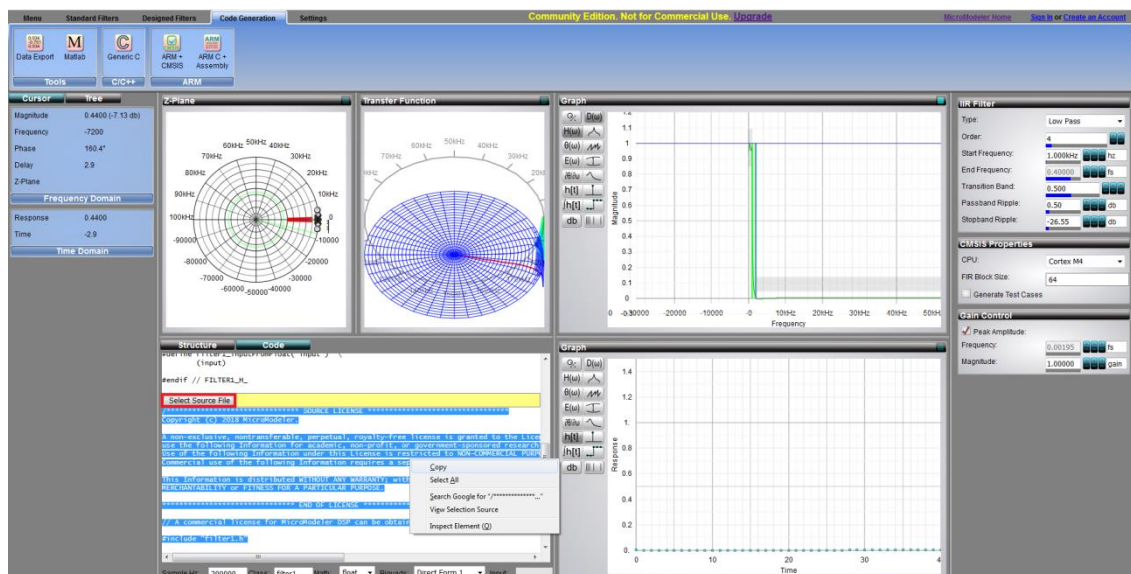



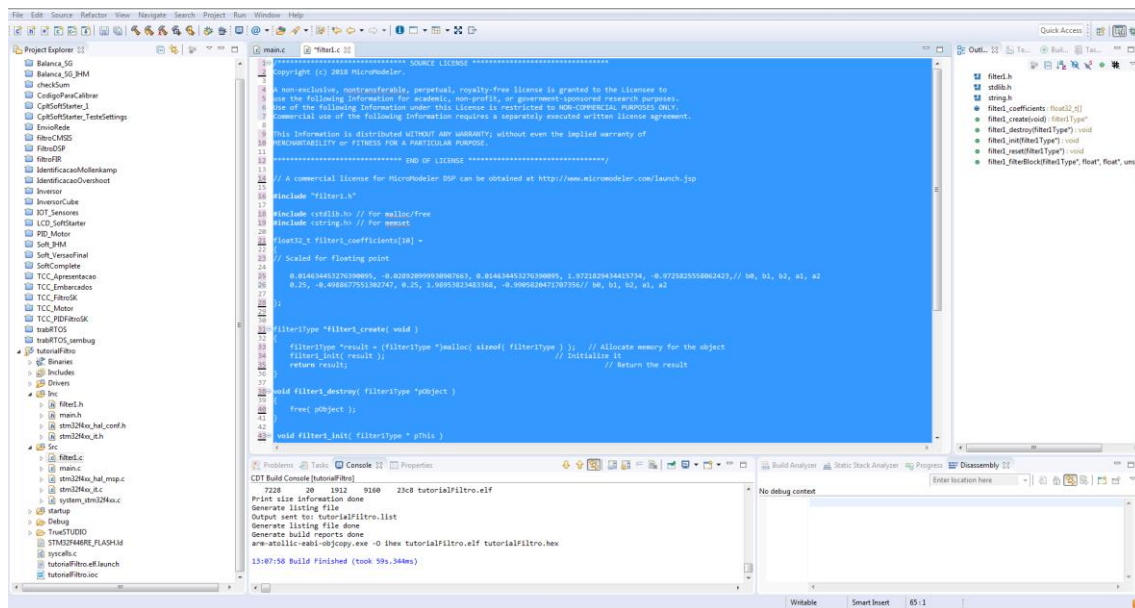
Agora basta copiar o código do “.h” gerado pelo site e colar dentro do arquivo “filter1.h” na pasta do projeto. Para selecionar o código pertinente ao “.h”, clicar sobre “Select Header File” e então copiar o código utilizando o botão direito do mouse (não funciona utilizando as teclas de atalho).



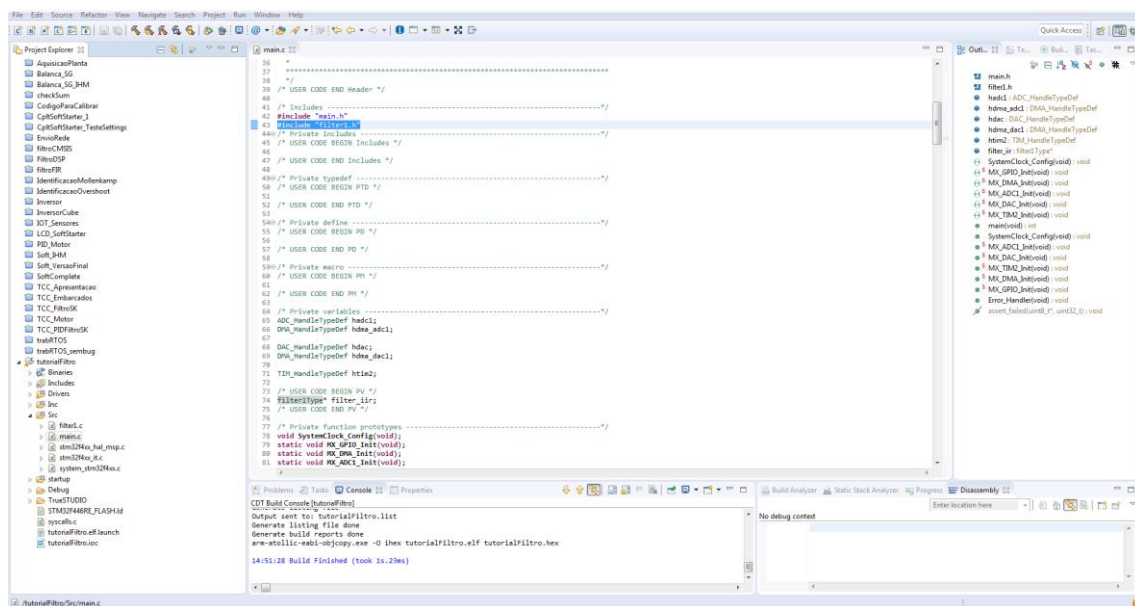


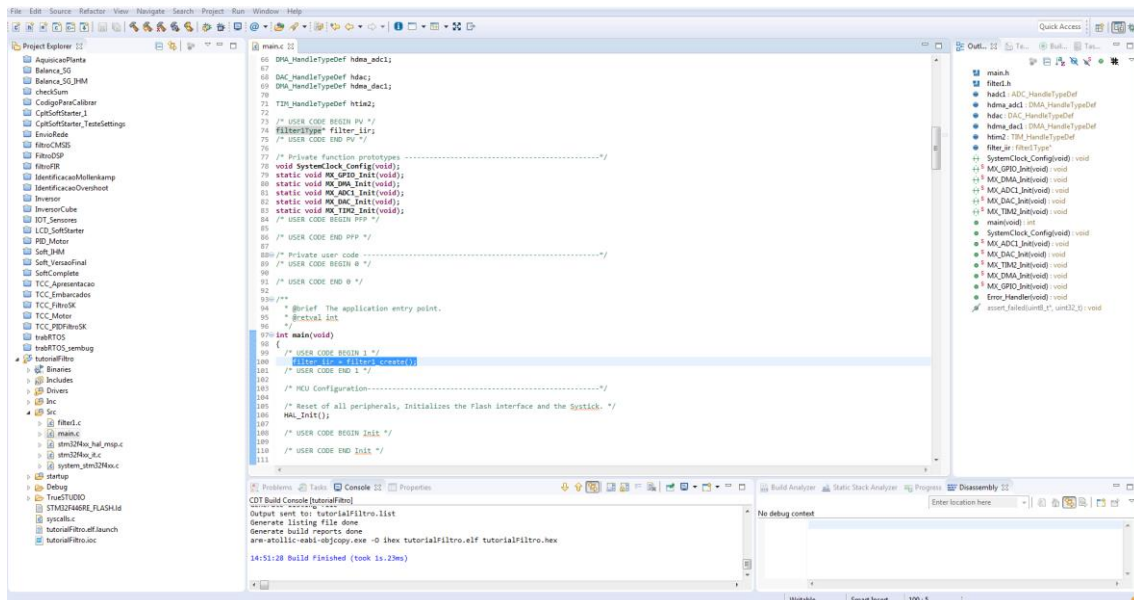
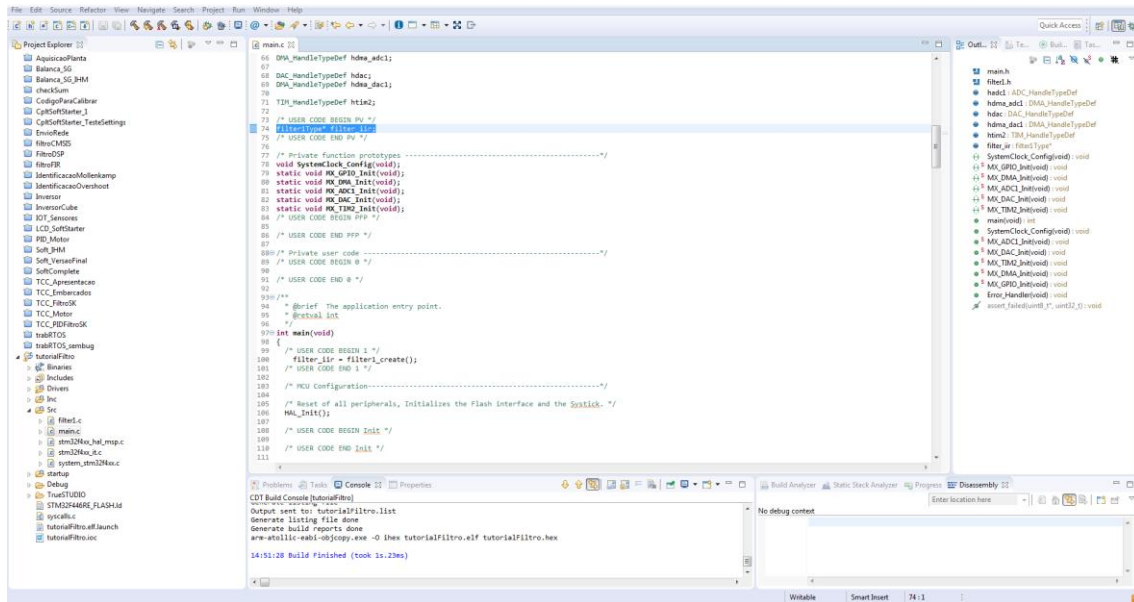
O mesmo procedimento deve ser adotado para transferir o código pertinente ao “.c” do site para a pasta do projeto. Desta vez, clicar na opção “Select Source File”.





Após esta última etapa, o filtro está quase pronto para o uso. Basta incluir o arquivo “filter1.h” no arquivo “main.c” e criar uma instância do mesmo como no exemplo abaixo.



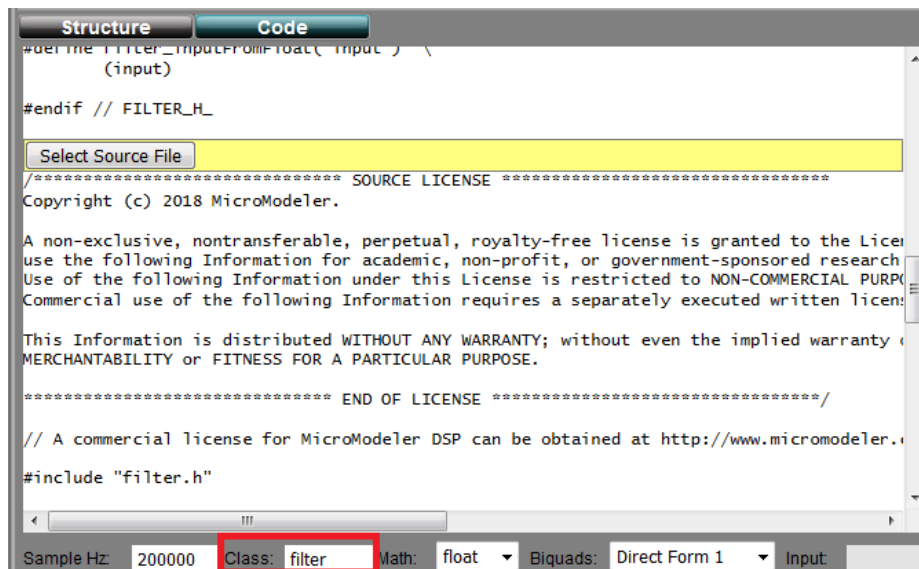


A seguinte função deve ser chamada para realizar a filtragem dos dados.

```
int filter1_filterBlock( filter1Type * pThis, float * pInput, float * pOutput, unsigned int count )
{
    arm_biquad_cascade_df1_f32( &pThis->instance, pInput, pOutput, count );
    return count;
}
```

É recomendado a filtragem de blocos de medidas para otimizar o processamento do microcontrolador. O código exemplo, disponibilizado no GitHub do autor, pode ser facilmente adaptado a outros tipos de filtros apenas

substituindo os arquivos “filter.h” e “filter.c” com o código gerado pelo site. É importante lembrar da necessidade de mudar o nome da classe do site para “filter” ao invés de “filter1”. Só assim o código exemplo funcionará corretamente após substituir os arquivos “.c” e “.h”.



PROJETO A PARTIR DO MATLAB / OCTAVE

O projeto através de CADs matemáticos permite a construção de filtros de ordem elevada. Como no caso anterior, é possível projetar tanto filtros IIR como filtros FIR. Encontrar os coeficientes do filtro através de um CAD matemático é uma tarefa bem menos intuitiva comparada à utilização do site. O programa utilizado neste tutorial será o OCTAVE, uma vez que sua licença é distribuída gratuitamente. O download do software OCTAVE pode ser feito através do site <https://www.gnu.org/software/octave/>. Serão apresentados alguns métodos de projeto de filtros IIR e de filtros FIR. No final, será mostrado um breve passo a passo de como implementar os filtros projetados na placa de desenvolvimento STM32F446RE utilizando a CMSIS.

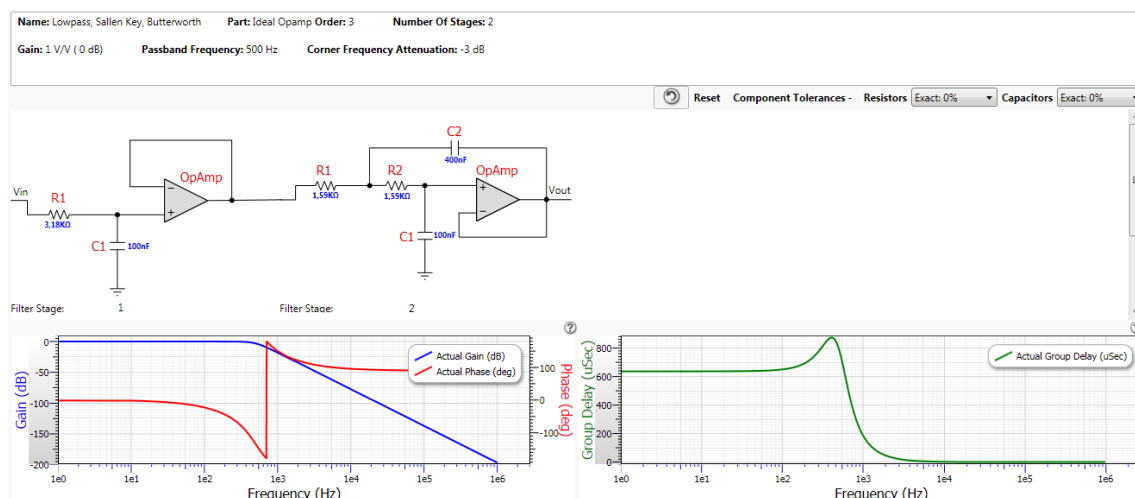
a) Filtro IIR – Discretização da Função de Transferência:

O processo de discretização da função de transferência permite criar um filtro digital que imita o comportamento de um filtro analógico. Para poder aplicar esse método, é necessário conhecer previamente a função de transferência desejada. A função de transferência relaciona algebricamente a saída de um sistema à sua entrada. É representada pela razão de dois polinômios no domínio Laplace.

Filtros analógicos de ordem elevada frequentemente são implementados através de sucessivos estágios de segunda ordem em cascata. As funções da CMSIS permitem que o mesmo procedimento seja adotado para a construção de filtros digitais. Desta forma, se torna possível discretizar a função de transferência de cada etapa individualmente, o que facilita bastante no momento da concepção do filtro.

Como exemplo, será projetado um filtro passa-baixas Butterworth de 3º ordem com uma frequência de corte de 500Hz. A frequência de amostragem adotada será de 200kHz (deve ser configurada no microcontrolador). A primeira etapa consiste em projetar um filtro analógico para então realizar o levantamento da função de transferência do mesmo. Para isto, será utilizado o software FilterPro da Texas Instruments. Esse programa é distribuído gratuitamente e pode ser baixado diretamente do site da fabricante de componentes.

O projeto do filtro no FiltroPro é fácil e intuitivo, assim o passo a passo não será abordado neste tutorial. Uma vez configurado os parâmetros para atender aos requisitos do exemplo, o seguinte circuito foi gerado:



O próximo passo consiste em realizar o levantamento da função de transferência do circuito usando como base os componentes calculados. O primeiro estágio (estágio de 1º ordem) tem seu comportamento regido pela seguinte função de transferência:

$$\frac{V_o}{V_{in}} = \frac{1}{RCs + 1} = \frac{1}{3180 \times 100 \times 10^{-9} \times s + 1} = \frac{1}{0.000318s + 1}$$

O segundo estágio (estágio de 2º ordem) pode ser descrito pela seguinte função de transferência:

$$\frac{Vo}{Vin} = \frac{\omega_n^2}{s^2 + \frac{\omega_n}{Q}s + \omega_n^2}$$

Onde:

$$\omega_n = \frac{1}{\sqrt{R_1 C_1 R_2 C_2}} = \frac{1}{\sqrt{1590 \times 100 \times 10^{-9} \times 1590 \times 400 \times 10^{-9}}} = 3144.65 \text{ rad/s}$$

$$\frac{1}{Q} = \sqrt{\frac{R_2 C_1}{R_1 C_2}} + \sqrt{\frac{R_1 C_1}{R_2 C_2}} = \sqrt{\frac{1590 \times 100 \times 10^{-9}}{1590 \times 400 \times 10^{-9}}} + \sqrt{\frac{1590 \times 100 \times 10^{-9}}{1590 \times 400 \times 10^{-9}}} = 1$$

Assim:

$$\frac{Vo}{Vin} = \frac{3144.65^2}{s^2 + 3144.65s + 3144.65^2}$$

Uma vez de posse das funções de transferência, é possível aplicar uma transformada bilinear para a obtenção das funções de transferência no domínio discreto. Para isto, iniciar o software OCTAVE e inserir no console as seguintes linhas de código:

```
>> format long;
>> [numd1, dend1] = bilinear(1, [0.000318, 1], 1/200E3);
>> [numd2, dend2] = bilinear(3144.65^2, [1, 3144.65, 3144.65^2], 1/200E3);
>> dtf1 = tf(numd1, dend1, 1/200e3, 'variable', 'z^-1')
>> dtf2 = tf(numd2, dend2, 1/200e3, 'variable', 'z^-1')
```

Após inserir as linhas de comando, o OCTAVE deve mostrar as funções de transferência no domínio discreto (domínio Z , onde Z^{-1} simboliza um operador de atraso de tempo).

```
Janela de Comandos
>> format long;
>> [numd1, dend1] = bilinear(1, [0.000318, 1], 1/200E3);
>> [numd2, dend2] = bilinear(3144.65^2, [1, 3144.65, 3144.65^2], 1/200E3);
>> dtf1 = tf(numd1, dend1, 1/200e3, 'variable', 'z^-1')

Transfer function 'dtf1' from input 'u1' to output ...

      0.0078 + 0.0078 z^-1
y1:  -----
      1 - 0.9844 z^-1

Sampling time: 5e-06 s
Discrete-time model.
>> dtf2 = tf(numd2, dend2, 1/200e3, 'variable', 'z^-1')

Transfer function 'dtf2' from input 'u1' to output ...

      6.132e-05 + 0.0001226 z^-1 + 6.132e-05 z^-2
y1:  -----
      1 - 1.984 z^-1 + 0.9844 z^-2

Sampling time: 5e-06 s
Discrete-time model.
>> |
```

Para descobrir os coeficientes necessários para a implementação do filtro digital, é preciso adequar a função de transferência discreta ao seu formato genérico. É importante lembrar que o operador Z^{-1} representa atraso na amostra e pode ser substituído por $[n - 1]$. Assim, $Vo[n - 1]$ representa o valor anterior da saída e $Vo[n]$ representa o valor atual (no instante n). Uma função de segunda ordem discreta pode ser generalizada no seguinte formato.

$$\frac{Vo}{Vin} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}}$$

$$a_0 Vo[n] + a_1 Vo[n - 1] + a_2 Vo[n - 2] = b_0 Vin[n] + b_1 Vin[n - 1] + b_2 Vin[n - 2]$$

$$a_0Vo[n] = b_0Vin[n] + b_1Vin[n - 1] + b_2Vin[n - 2] - a_1Vo[n - 1] - a_2Vo[n - 2]$$

A equação acima pode ser utilizada como alternativa ao uso das funções da CMSIS. É relevante ressaltar que a transformada bilinear é feita de uma forma que o coeficiente a_0 sempre será igual a um. Assim, $Vo[n]$ equivale ao sinal $Vin[n]$ após passar pelo filtro.

Para a utilização da CMSIS, é necessário organizar os coeficientes calculados em um vetor da seguinte forma.

```
float32_t coeficientes[2][5] = {{b0, b1, b2, -a1, -a2}, // Primeiro estágio
                                {b0, b1, b2, -a1, -a2}}; // Segundo estágio
```

No caso do exemplo, teremos a seguinte situação.

1º Estágio:

$$\frac{0.0078 + 0.0078z^{-1}}{1 - 0.9844z^{-1}} = \frac{0.0078 + 0.0078z^{-1} + 0z^{-2}}{1 - 0.9844z^{-1} + 0z^{-2}} = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{a_0 + a_1z^{-1} + a_2z^{-2}}$$

$$b_0 = 0.0078 \quad b_1 = 0.0078 \quad b_2 = 0$$

$$a_0 = 1 \quad a_1 = -0.9844 \quad a_2 = 0$$

2º Estágio:

$$\frac{6.132E - 5 + 0.0001226z^{-1} + 6.132E - 5z^{-2}}{1 - 1.984z^{-1} + 0.9844z^{-2}} = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{a_0 + a_1z^{-1} + a_2z^{-2}}$$

$$b_0 = 6.132E - 5 \quad b_1 = 0.0001226 \quad b_2 = 6.132E - 5$$

$$a_0 = 1 \quad a_1 = -1.984 \quad a_2 = 0.9844$$

```
float32_t coeficientes[2][5] = {{7.8E-3, 7.8E-3, 0, 9.844E-1, 0},  
                                {6.132E-5, 1.226E-4, 6.132E-5, 1.984, -9.844E-1}};
```

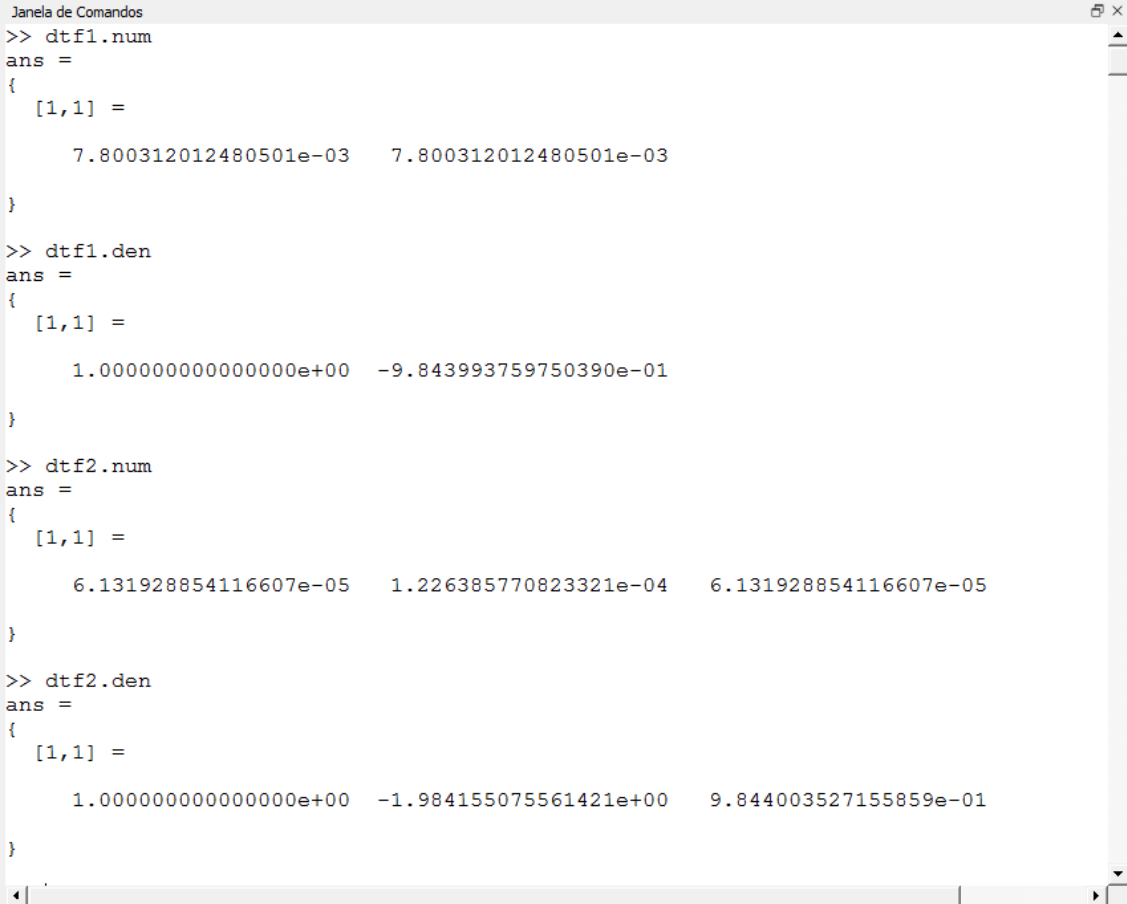
Caso uma maior precisão dos coeficientes seja necessária, utilizar as seguintes instruções para mostrar os números com um maior número de algarismos significativos.

```
>> dtf1.num
```

```
>> dtf1.den
```

```
>> dtf2.num
```

```
>> dtf2.den
```



```
Janela de Comandos
>> dtf1.num
ans =
{
    [1,1] =
        7.800312012480501e-03    7.800312012480501e-03
}
>> dtf1.den
ans =
{
    [1,1] =
        1.000000000000000e+00    -9.843993759750390e-01
}
>> dtf2.num
ans =
{
    [1,1] =
        6.131928854116607e-05    1.226385770823321e-04    6.131928854116607e-05
}
>> dtf2.den
ans =
{
    [1,1] =
        1.000000000000000e+00    -1.984155075561421e+00    9.844003527155859e-01
}
}
```

Observação: para cada tipo de filtro / topologia diferente da adotada no exemplo, é necessário seguir um procedimento diferente para realizar o levantamento da função de transferência. Os cálculos apresentados funcionam apenas para filtros passa-baixas topologia Sallen-Key.

b) Filtro IIR – Funções Dedicadas:

O software OCTAVE conta com funções dedicadas para o projeto de filtros digitais. Existe uma função específica para o projeto de cada aproximação de filtro. Na sequência estão listadas as funções disponíveis.

```
>> [z,p,k] = besself(n,Wn,ftype)      % Bessel
>> [z,p,k] = butter(n,Wn,ftype)       % Butterworth
>> [z,p,k] = cheby1(n,Rp,Wn,ftype)    % Chebyshev Tipo I
>> [z,p,k] = cheby2(n,Rs,Wn,ftype)    % Chebyshev Tipo II
>> [z,p,k] = ellip(n,Rp,Rs,Wn,ftype)  % Elíptico
```

Onde:

- n: ordem do filtro
- Rp: ripple na banda passante
- Rs: ripple na banda de rejeição
- Wn: frequência de corte em relação à frequência de amostragem

$$Wn = \text{freq_corte} / \text{freq_amost} * 2$$

- ftype: tipo do filtro (passa-baixas, passa-altas, ...)

‘low’ = filtro passa-baixas

‘high’ = filtro passa-altas

‘pass’ = filtro passa-banda

‘stop’ = filtro rejeita-banda

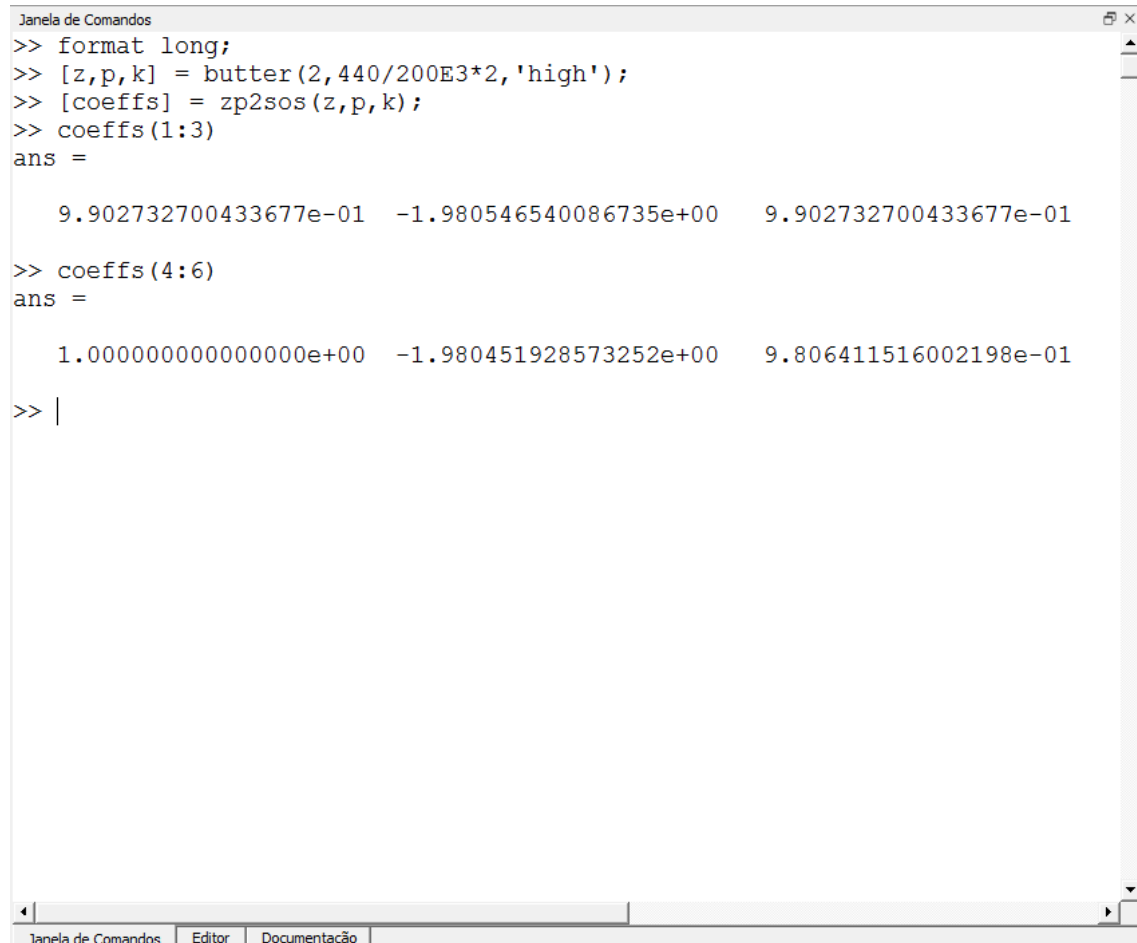
Como exemplo, será projetado um filtro passa-altas Butterworth de 2º ordem com frequência de corte de 440Hz. Será utilizado 200kHz de frequência de amostragem como nos exemplos anteriores. As seguintes linhas de código devem ser inseridas no console do OCTAVE.

```
>> format long;
>> [z,p,k] = butter(2,440/200E3*2,‘high');
```

```
>> [coeffs] = zp2sos(z,p,k);

>> coeffs(1:3)

>> coeffs(4:6)
```



```
Janela de Comandos
>> format long;
>> [z,p,k] = butter(2,440/200E3*2,'high');
>> [coeffs] = zp2sos(z,p,k);
>> coeffs(1:3)
ans =
    9.902732700433677e-01   -1.980546540086735e+00    9.902732700433677e-01

>> coeffs(4:6)
ans =
    1.000000000000000e+00   -1.980451928573252e+00    9.806411516002198e-01

>> |
```

$$b_0 = 9.902E-1 \quad b_1 = -1.9805 \quad b_2 = 9.902E-1$$

$$a_0 = 1 \quad a_1 = -1.9804 \quad a_2 = 0.9806$$

Como no método anterior, é necessário adequar os coeficientes calculados ao formato compreendido pela CMSIS. Para isto, os coeficientes devem ser organizados em um vetor na seguinte forma.

```
float32_t coeficientes[5] = {b0, b1, b2, -a1, -a2};
```

```
float32_t coeficientes[5] = {9.902E-1, -1.9805, 9.902E-1, 1.9804, -0.9806};
```


É possível visualizar a resposta esperada do filtro recém projetado ao inserir no console as seguintes linhas de código.

```
>> [b,a] = zp2tf(z,p,k);  
>> freqz(b,a);
```

No projeto de filtros passa-banda e rejeita-banda, a ordem do filtro obtido será de duas vezes a ordem especificada. Assim, para o projeto de um filtro passa-banda de décima ordem com uma banda passante de 400Hz a 600Hz e uma frequência de amostragem de 200kHz, a seguinte instrução deve ser utilizada.

```
>> [z,p,k] = butter(5,[400,600]/200E3*2,'pass');
```

É importante notar que a frequência de corte inferior e a frequência de corte superior devem ser informadas entre colchetes e separadas por vírgula no espaço destinado à frequência de corte. Não é possível implementar filtros de ordem ímpar, uma vez que a função disponível suporta apenas números inteiros como argumento para a ordem do filtro.

Observação: o OCTAVE não possui todas as funções adequadamente adaptadas. Sendo assim, o usuário pode vir a se deparar com problemas em algumas instruções. Nestes casos, recomenda-se o uso do MATLAB.

c) Filtro FIR – Método da Janela

Um filtro FIR pode ser facilmente projetado no software OCTAVE através do método da janela. O programa conta com uma função dedicada que facilita o processo. Na sequência está o protótipo da função juntamente com uma breve explicação a respeito dos seus argumentos.

```
>> b = fir1(n,Wn,ftype)
```

Onde:

- n: ordem do filtro
- Wn: frequência de corte em relação à frequência de amostragem

$$Wn = \text{freq_corte} / \text{freq_amost} * 2$$

- ftype: tipo do filtro (passa-baixas, passa-altas, ...)

‘low’ = filtro passa-baixas

‘high’ = filtro passa-altas

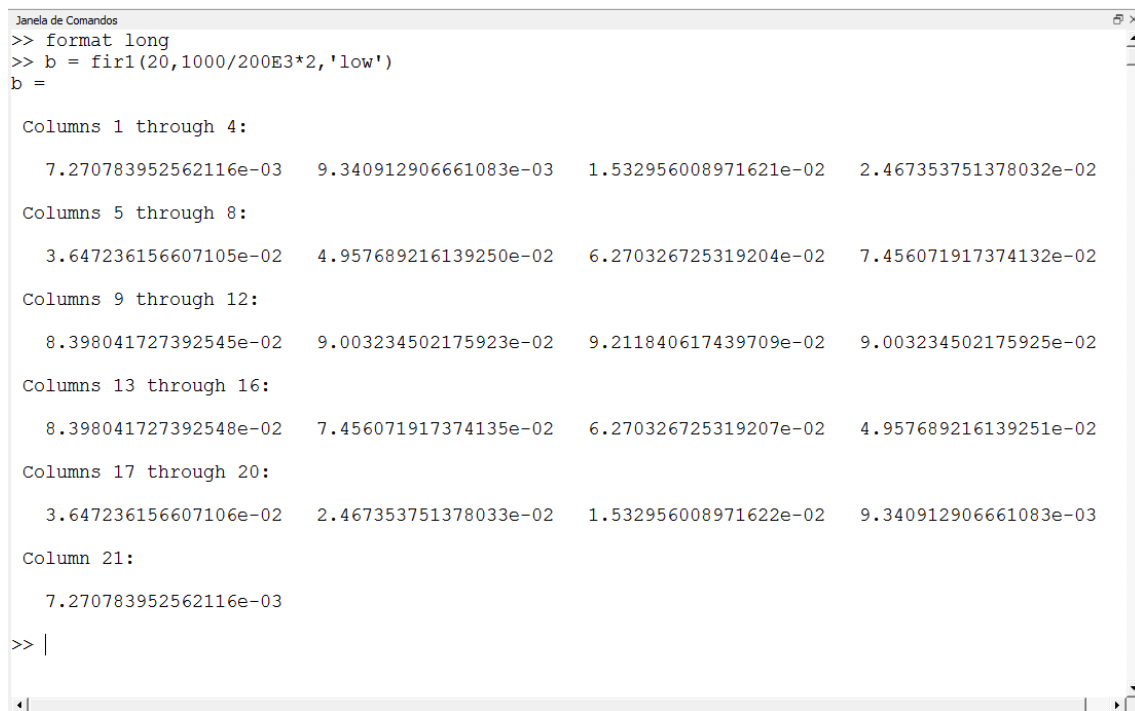
‘pass’ = filtro passa-banda

‘stop’ = filtro rejeita-banda

Será projetado, como exemplo, um filtro passa-baixas de 20^o ordem com frequência de corte de 1000Hz. Será utilizado 200kHz de frequência de amostragem. Para realizar o projeto, é necessário inserir as seguintes linhas de código no console do OCTAVE.

```
>> format long;
```

```
>> b = fir1(20,1000/200E3*2,'low')
```



```
Janela de Comandos
>> format long
>> b = fir1(20,1000/200E3*2,'low')
b =

Columns 1 through 4:
    7.270783952562116e-03    9.340912906661083e-03    1.532956008971621e-02    2.467353751378032e-02
Columns 5 through 8:
    3.647236156607105e-02    4.957689216139250e-02    6.270326725319204e-02    7.456071917374132e-02
Columns 9 through 12:
    8.398041727392545e-02    9.003234502175923e-02    9.211840617439709e-02    9.003234502175925e-02
Columns 13 through 16:
    8.398041727392548e-02    7.456071917374135e-02    6.270326725319207e-02    4.957689216139251e-02
Columns 17 through 20:
    3.647236156607106e-02    2.467353751378033e-02    1.532956008971622e-02    9.340912906661083e-03
Column 21:
    7.270783952562116e-03
>> |
```

Os valores encontrados são os coeficientes do filtro FIR. É necessário organizar os coeficientes em forma de um vetor para a utilização da CMSIS. O vetor deve ser construído na seguinte forma.

```
float32_t coeficientes[21] = {b0, b1, ... , b19, b20, b21};
```

```
float32_t coeficientes[21] = {7.27E-3, 9.34E-3, ... , 1.53E-2, 9.34E-3, 7.27E-3};
```

A resposta em frequência e fase do filtro FIR projetado pode ser vista ao inserir o seguinte comando no console.

```
>> freqz(b);
```

Para o projeto de um filtro FIR passa-banda / rejeita-banda é necessário informar ambas as frequências de corte entre colchetes e separadas por vírgula como no exemplo abaixo.

```
>> b = fir1(70,[fc1,fc2]/200E3*2,'pass');
```

Observação: o software OCTAVE, assim como o MATLAB, conta com outras funções para o projeto de filtros FIR. Um material de apoio para a utilização destas outras funções pode ser encontrado no próprio site do MATLAB (<https://www.mathworks.com/help/signal/ug/fir-filter-design.html>).

d) Filtro Digital – Utilizando a CMSIS:

Para a utilização das funções da CMSIS, é necessário incluir algumas bibliotecas. Para isto, as seguintes linhas de código devem ser inseridas no início do programa.

```
#define ARM_MATH_CM4           // Define o CPU como ARM Cortex M4
#define __FPU_PRESENT 1       // Dispositivo trabalha com ponto flutuante
#include <arm_math.h>          // Inclui o cabeçalho da CMSIS
```

O segundo passo se baseia em criar as estruturas necessárias para o funcionamento do filtro. Como exemplo, será implementado o filtro IIR projetado através da transformada bilinear. Esse filtro é um passa-baixas com uma frequência de corte de 500Hz e é constituído por dois estágios de segunda ordem em cascata. No espaço dedicado à declaração das variáveis, inserir as seguintes instruções.

```
#define nEstagios (2)                // Número de estágios de segunda ordem

float32_t coeficientes[nEstagios][5] =
{{7.8E-3, 7.8E-3, 0, 9.844E-1, 0},    // b0, b1, b2, a1, a2
 {6.132E-5, 1.226E-4, 6.132E-5, 1.984, -9.844E-1}}; // b0, b1, b2, a1, a2

float32_t retentores[nEstagios][4] = {0};

arm_biquad_casd_df1_inst_f32 instancia; // Instância do filtro
```

Uma vez com as estruturas criadas, é necessário inicializar a estrutura do filtro. Para isso, o seguinte comando deve ser inserido dentro do “main” do código.

```
int main(void){

arm_biquad_cascade_df1_init_f32(&instancia, nEstagios, coeficientes, retentores);

...

<código>

...

}
```

Com as etapas anteriores completas, o filtro encontra-se pronto para o uso. A seguinte função deve ser chamada para o processamento dos dados.

```
arm_biquad_cascade_df1_f32(&instancia, pMedidas, pSaida, nMedidas);
```

Onde “pMedidas” é um ponteiro para o vetor que contém as medidas a serem processadas; “pSaida” é um ponteiro para o vetor que receberá os dados filtrados e “nMedidas” o número de medidas a serem processadas. O arquivo “main.c” desse exemplo pode ser encontrado no GitHub do autor com o nome “main_CMSIS.c”.