EE 311 **February 12, 2018**

Creating a Low Pass Filter

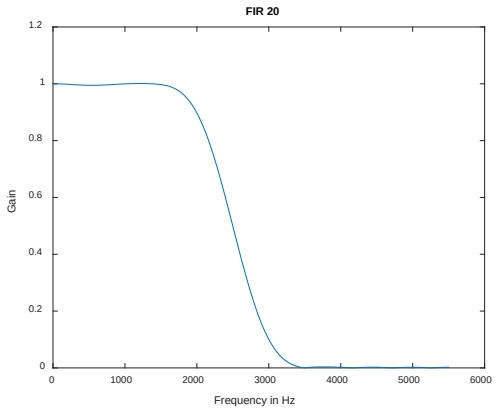
Use fir1 to create a low pass FIR filter with a Hamming window of order 20 that has a cutoff frequency of 2500 Hz and a sample frequency of 11025 Hz.

We begin with MATLAB and design the filter:

```
%FIR20A.m
N = 20;
fc = 2500; fs = 11025;  % cutoff and sample frequency
num = fir1(N,fc/(fs/2),hamming(N+1));
for i=1:length(num);
    fprintf(1, '%3.12f\n', num(i));
end
[H f] = freqz(num, 1, 1024, fs);
figure(1);clf;
plot(f, abs(H));
```

This program creates the following figure and prints the coefficients shown.

```
0.002524244820
0.000917229061
-0.006128585069
-0.006375978798
0.016173804879
0.025544812165
-0.029857375998
-0.077656305614
0.041690040532
0.307013374458
0.452309479124
0.307013374458
0.041690040532
-0.077656305614
-0.029857375998
0.025544812165
0.016173804879
-0.006375978798
-0.006128585069
0.000917229061
0.002524244820
```



The response is symmetric so we create a coefficient array in C with half of the coefficients:

```
const float b[] = {0.002524244820, 0.000917229061, -0.006128585069, -0.006375978798,
                   0.016173804879, 0.025544812165, -0.029857375998, -0.077656305614,
                   0.041690040532, 0.307013374458, 0.452309479124
                  };
```

Note that the coefficients have been declared constant and go into program memory. The difference equation can be implemented as a loop:

```
    y = b[10]*x[10];   //The center coefficient
    for(i=0;i<10;i++)
      y += b[i]*(x[i] + x[20-i]);
```

We can shift the variables with another loop.

```
  for(i=20;i>0;i--)
        x[i] = x[i-1];
```

The complete program is given below:

```
#include "stm32f446.h"
/*stm446Template.c              December 28, 2017
  This program toggles a bit on PA7 at about 100 KHz.
*/
void InitializeClock(void);
//FIR20Array.c
/*
This filter was designed in MatLab as a 20th order FIR filter with
%FIR20A.m
N = 20;
fc = 2500; fs = 11025;  % cutoff and sample frequency
num = fir1(N,fc/(fs/2),hamming(N+1));
for i=1:length(num);
    fprintf(1, '%3.12f\n', num(i));
end
[H f] = freqz(num, 1, 1024, fs);
figure(1);clf;
plot(f, abs(H));


PA5 is analog input
PA4 is analog output
PA7 is digital output
*/
const float b[] = {0.002524244820, 0.000917229061, -0.006128585069, -0.006375978798,
                   0.016173804879, 0.025544812165, -0.029857375998, -0.077656305614,
                   0.041690040532, 0.307013374458, 0.452309479124
                  };
void InitializeClock(void);
int main()
  {int i, xInt, yInt;
   float x[21];  //x is input and y is output
   float y;
   //Clock bits
   InitializeClock();        //Set clock to 168 MHz
   RCC_AHB1ENR |= 1;         //Bit 0 is GPIOA clock enable bit
   RCC_APB1ENR |= (1 << 29); //Bit 29 is DAC clock enable bit
   RCC_APB2ENR |= 0x100;     //Bit 8 is ADC 1 clock enable bit
   RCC_APB1ENR |= (1 << 4);  //Enable peripheral timer for timer 6
   //I/O bits
   GPIOA_MODER |= 0x4000;    //Bits 15-14 = 01 for digital output on PA7
   //OTYPER register resets to 0 so it is push/pull by default
   GPIOA_OSPEEDER |= 0xC000;  //Bits 15-14 = 11 for high speed on PA7
   //PUPDR defaults to no pull up no pull down
   GPIOA_MODER |= 0xF00;     //PA4-PA5 are analog
   GPIOA_PUPDR &= 0xFFFFF0FF;//Pins PA4 PA5 are no pull up and no pull down
   //DAC bits
   DAC_CR |= 0x3E;           //Bits 3, 4, 5 = 111 for software trigger ch1
                             //Bit 2 = 1 for Ch 1 trigger enabled
                             //Bit 1 = 1 for Ch 1 output buffer enabled
   DAC_CR |= 1;              //Bit 0 = 1 for Ch 1 enabled
   //ADC bits
   ADC1_CR2 |= 1;             //Bit 0 turn ADC on
   ADC1_CR2 |= 0x400;         //Bit 10 allows EOC to be set after conversion
   ADC_CCR |= 0x30000;       //Bits 16 and 17 = 11 so clock divided by 8
   ADC1_SQR3 |= 0x5;          //Bits 4:0 are channel number for first conversion
                             // Channel is set to 5 which corresponds to PA5
   //Timer 6 bits
   TIM6_CR1 |= (1 << 7);   //Auto reload is buffered
   TIM6_CR1 |= (1 << 3);   //One pulse mode is on.
   TIM6_PSC = 0;           //Don't use prescaling
```

```
 TIM6_ARR = 8163;        //(180 MHz/2)/8163 = 11025 Hz
 TIM6_CR1 |= 1;          //Enable Timer 6
 TIM6_EGR |= 1;
 for(i=0;i<21;i++)       //Initialize x to 0
    x[i] = 0;
 //Main program loop
 while(1)
   {GPIOA_ODR |= (1 << 7);      //Set bit 7 to 1
    ADC1_CR2 |= 0x40000000;      //Bit 30 does software start of A/D conversion
    while((ADC1_SR & 0x2) == 0); //Bit 1 is End of Conversion
       xInt = ADC1_DR;
    x[0] = ((float)(xInt & 0xFFF))/(float)4095.0;
    //This loop does the difference equation
    y = b[10]*x[10];
    for(i=0;i<10;i++)
             y += b[i]*(x[i] + x[20-i]);
       yInt = (int)(1000*y); //Data to D/A
    DAC_DHR12R1 = yInt & 0xFFF; //Converted number to D/A
    DAC_SWTRIGR |= 0x1;         //Start the D/A conversion
    //This loop does shifting of variable for next loop
    for(i=20;i>0;i--)
       x[i] = x[i-1];
    GPIOA_ODR &= ~(1 << 7);     //Set bit 7 to 0
    while((TIM6_CR1 & 1) != 0); //Wait here until timer runs out
    TIM6_CR1 |= 1;              //Restart timer
   }
}

    //This function resets the system clock to 180 MHz.
void InitializeClock()
    {RCC_CFGR = 0x00000000;         //Reset Clock Configuration Register
     RCC_CR &= 0xFEF6FFFF;          //Reset HSEON, CSSON and PLLON Bits
     RCC_CR |= (1 << 16);           //Turn on HSE clock
     while((RCC_CR & (1 << 17)) == 0); //Wait until HSE is ready
     RCC_CR |= (1 << 19);
     RCC_PLLCFGR = 0x27405A08;      //Set PLLP = 0, PLLN = 360, PLLM = 8,
                                    //PLLQ = 7, PLL Src = HSE
     FLASH_ACR &= 0xFFFFFFF8;       //Set flash wait states to 5
     FLASH_ACR |= 0x5;
     RCC_CR |= (1 << 24);           //Enable PLL on
     while((RCC_CR & (1 << 25)) == 0); //Wait for PLL to lock on
     RCC_CFGR = 0x9402;             // APB2/2, APB1/4, AHB/1
    }
```