Design a band pass filter using any method that meets the following specifications:

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fs</td>
<td>22050 Hz</td>
</tr>
<tr>
<td>stop band 1</td>
<td>0 to 1.5 KHz</td>
</tr>
<tr>
<td>pass band 1</td>
<td>3.0 KHz to 4.0 KHz</td>
</tr>
<tr>
<td>stop band 2</td>
<td>5.5 KHz to fs/2</td>
</tr>
<tr>
<td>pass band ripple</td>
<td>0.05</td>
</tr>
<tr>
<td>stop band ripple</td>
<td>0.02</td>
</tr>
<tr>
<td>filter order</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

Find the coefficients for the transfer function and implement your filter in C on the ARM Cortex STM32F4 Discovery board. Demonstrate your results and get the attached verification sheet signed.

Turn in the following:
1. Cover page with your name, the date turned in, the assignment number, and a table listing the filter coefficients.
2. The commented MATLAB® code which does the filter design and produces a plot of the filter frequency response.
3. The plots from your MATLAB® code with appropriate labels. These should include a) The frequency response from 0 to fs/2; b) a blow up of the pass band showing that it meets specifications; c) a blow up of each of the two stop bands showing that they meet specifications; d) a pole/zero plot of the transfer function.
4. A listing of your c-code for the filter.
5. A signed verification sheet that your modified filter worked. For the verification sheet you will need to use a signal generator input sinusoids. Note that the sinusoidal input must be in the range of 0 < v < 3.3 to avoid damage to your chip.

Note: This is a band pass filter and does not pass dc. Your output will be centered around 0 volts for a sinusoidal input. The D/A converter cannot output a negative signal and the negative portion of this will wrap around due to the twos complement number system. It will be necessary to add a dc offset to the output to see it properly since the d/a produces outputs from 0 to 3.3 volts. You may also need an amplification factor. A typical output statement in c might look like this, where y is the calculated output variable.

```c
yInt = (int)(3000*(y+1)); //Data to D/A
DAC_DHR12R1 = yInt & 0xFFF; //Converted number to D/A
```

You will also have to change the value loaded into TIM6 to get a sample frequency of 22050 Hz. See the comments in the line `TIM6_ARR = 7619;` of FIR30.c for an equation to calculate the correct value.
Student _____________________ has demonstrated a working digital filter running on the ARM Cortex board that produces an output signal indicating a band pass filter with a pass band centered between 3.0 KHz and 4.0 KHz.

Instructor______________________  Date ______________________
Blandford, Cron, or Randall
//FIR30Array.c
/*
This is the same program as FIR30 but uses arrays for coefficients and input variables.

This program implements a filter using integer arithmetic
This filter was designed in Matlab as a 31th order FIR filter with
%FIR30.m
N = 30;
f_c = 2000; f_s = 11025; % cutoff and sample frequency
[num den] = fir1(N,f_c/(f_s/2),hamming(N+1));

PA5 is analog input
PA4 is analog output
PA7 is digital output
*/
#include "stm32f407vg.h"
extern void SystemCoreClockUpdate(void);
const float b[] = {-0.001671740, -0.000505838, 0.002282468, 0.003995449,
  -0.000191671, -0.009090046, -0.010429167, 0.005909405,
  0.026566393,  0.019112863, -0.027066681, -0.066756728,
  -0.026576262,  0.116155203,  0.286640849, 0.363251007};
void InitializeClock(void);
int main()
{int i, xInt, yInt;
  float x[31];  //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
  GPIOA_OSPEEDER |= 0xC000; //Bits 15-14 = 11 for high speed on PA7
  //DAC bits
  DAC_CR |= 0x3E;           //Bits 3, 4, 5 = 111 for software trigger ch1
  DAC_CR |= 1;              //Bit 0 = 1 for Ch 1 enabled
  DAC_CR &= 0xFFFFF0FF;     //Channel is set to 5 which corresponds to PA5
  //ADC bits
  ADC1_CR2 |= 1;            //Bit 0 turn ADC on
  ADC1_CR2 |= 0x400;        //Bit 10 allows EOC to be set after conversion
  ADC1_SQR3 |= 0x5;         //Bits 4-0 are channel number for first conversion
  //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 = 7619;         //168 MHz/2)/7619 = 11025 Hz
  TIM6_CR1 |= 1;           //Enable Timer 6
  TIM6_EGR |= 1;
for(i=0;i<31;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[15]*x[15];
    for(i=0;i<15;i++)
        y += b[i]*(x[i] + x[30-i]);
    yInt = (int)(2048*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=30;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 168 MHz.
void InitializeClock()
{
    RCC_CFGR = 0x00000000;         //Reset Clock Configuration Register
    RCC_CR &= 0xEF06FFFF;          //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 = 0x27405408;      //Set PLLP = 0, PLLN = 336, PLLM = 8,
                                   //PLLQ = 7, PLL Src = HSE
    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
    FLASH_ACR &= 0xFFFFFFF8;       //Set flash wait states to 5
    FLASH_ACR |= 0x5;
    SystemCoreClockUpdate();
}