The c-program on the following pages implements a 30th order FIR low pass filter for the ARM Cortex STM32F4 Discovery board. The sample frequency is about 11025 Hz and a cutoff frequency of 2 KHz. The Keil project file for the program is on the website as FIR30Array.zip.

A) Download the zipped file, compile it, and load it into the ARM Cortex board. Verify that it runs as a low pass filter, with a sample frequency of 11025 Hz, and a cutoff frequency of 2 KHz.

B) The filter was designed from MATLAB® using the following code:

```
%FIR30.m
N = 30;
fc = 2000; fs = 11025;  % cutoff and sample frequency
num = fir1(N,fc/(fs/2),hamming(N+1));
```

Alter the MATLAB® code so that the cutoff frequency is 1 KHz and the window is a Blackman window. Implement your redesigned filter on the ARM Cortex board and have it verified by your instructor.

Turn in the following:
1. Cover page with your name, the date turned in, and the assignment number.
2. The MATLAB® code which produces a plot of the original filter frequency response and your modified filters frequency response on the same axis. Use two different colors. You should plot the magnitude for both filters on one plot and the phase for the two on a second plot.
3. The plots from your MATLAB® code with appropriate labels.
4. A listing of your modified c-code for the new filter.
5. A table which lists the coefficients for the original filter and your modified filter side-by-side.
6. 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 $0 < v < 3.0$ to avoid damage to your chip. Set the amplitude to 1.2 volts and set the offset to 1.3 volts.
Verification sheet
EE 311 STM Assignment 5

Student _____________________ has demonstrated a working digital filter running on the
ARM Cortex board that produces an output signal indicating a low pass filter with a cutoff
frequency of 1 KHz

Instructor____________________________   Date ______________________

Blandford, Cron, or Randall
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

```matlab
N = 30;
f = 2000; fs = 11025;  % cutoff and sample frequency
[num den] = fir1(N,f/(fs/2),hamming(N+1));
```

PA5 is analog input
PA4 is analog output
PA7 is digital output

```c
#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
    //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 |= 0xF000;  //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 = 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 &= 0xEF6FFFFF;          //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);           //Enable PLL on
    RCC_PLLCFGR = 0x27405408;      //Set PLLP = 0, PLLN = 336, PLLM = 8,
                                 //PLLQ = 7, PLL Src = HSE
    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
    SystemCoreClockUpdate();
}