Departament d'Enginyeria Electrònica
CAMPUS DE TERRASSA

Projecte final de carrera

Títol PFC:

Estudio y desarrollo de un sistema de detección de fallos en actuadores electromecánicos mediante el análisis de emisiones acústicas.

Contingut:

Annex

Titulació:

2n Cicle Enginyeria Automàtica i electrònica

Assignatura:

Projecte final de Carrera

Alumne:

Daniel Zurita Millán

Professor/a:

Tutor: Jose Luis Romeral Martinez

In this annex you can found the code of the embedded application.

Main program

```
/* Codi per indicar la presència de fallades

Alumne: Daniel Zurita MIllán

-----------------------------------*/
/* Includes ------------------ ---------------------*/

#include "stm32f4xx.h"
#define ARM_MATH_CM4
#include "arm_math.h"
#include <stdio.h>
#include "tools.h"
#include "interrupts_config.h"
#include "fsmc_ram.h"

/* Variables ----------------------------------------------------
----*/

USART_InitTypeDef USART_InitStructure;
USART_InitTypeDef USART_InitStructure;
GPIO_InitTypeDef GPIO_InitStructure;

GPIO_InitTypeDef  Led_GPIO_InitStructure;

/* Private functions ---------------------------------------------
------------*/

/**
  * @brief  Main program
  * @param  None
  * @retval None
  */
int main(void)
{
  /*!< At this stage the microcontroller clock setting is already
configured,
       this is done through SystemInit() function which is called
from startup
       file (startup_stm32f4xx.s) before to branch to application
main.
       To reconfigure the default setting of SystemInit()
function, refer to
       system_stm32f4xx.c file
     */


  /* USARTx configured as follow:
        - BaudRate = 57600 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled
  */
```

```
      //USART Configuration


      RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);//(1)
      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC,ENABLE);//(2)


      GPIO_PinAFConfig(GPIOC,GPIO_PinSource10,GPIO_AF_USART3);//(3
.1) TX
      GPIO_PinAFConfig(GPIOC,GPIO_PinSource11,GPIO_AF_USART3);

/* Configure USART Tx as alternate function  */
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;//COM_TX_PIN[COM];
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* Configure USART Rx as alternate function  */
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;//COM_RX_PIN[COM];
  GPIO_Init(GPIOC, &GPIO_InitStructure);



      //RS232 CONFIG
  USART_InitStructure.USART_BaudRate = 57600;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;




// FUNCTIONS TO ENABLE USART
      USART_Init(USART3, &USART_InitStructure);
      USART_Cmd  ( USART3,  ENABLE  );


//external memory configuration
  SRAM_Init();//


// ADC + DMA configuration
       ADC3_CH7_DMA_Config();



          while (1)
    {


//recoge datos ADC y los guarda en memoria RAM externa
```

```
        Save_DataAQC();

printf("              DATA PROCESSING \n \n \r");

printf("Here the Results:\n \r");

                  // Función que realiza los filtros y parametros
+ las comparaciones logicas(RMS and filters)
                   Feature_analizing();

    }
}

#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source
line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name
and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n",
file, line) */

  /* Infinite loop */
  while (1)
  {
  }
}
#endif
```

Other Code

```
void SRAM_ReadBuffer(float32_t* pBuffer, uint32_t ReadAddr,
uint32_t NumHalfwordToRead)
{
  //for (; NumHalfwordToRead != 0; NumHalfwordToRead--) /* while
there is data to read */

      uint32_t i=0;

      for (i=0; i < NumHalfwordToRead; i++)
  {
    /* Read a half-word from the memory */
    *pBuffer++ = *(__IO float32_t*) (Bank1_SRAM3_ADDR  +
ReadAddr);

    /* Increment the address*/
    ReadAddr += 4;
  }
}
```

```
void Fill_Buffer(float32_t *pBuffer, uint32_t BufferLenght,
uint32_t Offset)
{
  uint32_t IndexTmp = 0;

  /* Put in global buffer different values */
  for (IndexTmp = 0; IndexTmp < BufferLenght; IndexTmp++ )
  {
    //pBuffer[IndexTmp] = IndexTmp + Offset;

            pBuffer[IndexTmp] = IndexTmp + Offset;



  }
}

float32_t ADC_GetSampleAvgN(uint32_t N)
{



float32_t avg_sample =0.0f;
//uint16_t adc_sample[10]={0};
      //uint16_t adc_sample=0;
uint32_t index=0;
      //uint32_t WRITE_READ_ADDR =    0;
      uint32_t ReadAddr =    0;
/* Get the N ADC samples */


//for (index=1; index<N; index++)
//{
/* ADC start conv */
//ADC_SoftwareStartConv(ADC3);
/* Wait end of conversion */
//while(ADC_GetFlagStatus(ADC3,ADC_FLAG_EOC) == RESET);
/* Store ADC samples*/
      //adc_sample=ADC_GetConversionValue(ADC3);
//*(uint16_t *) (Bank1_SRAM3_ADDR + WRITE_READ_ADDR
)=(float32_t)ADC_GetConversionValue(ADC3);

      //*(float32_t *)Bank1_NORFLASH1_ADDR= (float32_t)*(__IO
uint16_t*) (Bank1_SRAM3_ADDR  + WRITE_READ_ADDR);
//    *(float32_t *)(Bank1_NORFLASH1_ADDR + WRITE_READ_ADDR)=
ADC_GetConversionValue(ADC3);
      //*(float32_t *) (Bank1_SRAM3_ADDR + WRITE_READ_ADDR
)=ADC_GetConversionValue(ADC3);
      //SRAM_WriteBuffer(&adc_sample, WRITE_READ_ADDR , 1);
//    WRITE_READ_ADDR   += 4;
//adc_sample[index] = ADC_GetConversionValue(ADC3);
//    *(uint16_t *) (Bank1_SRAM3_ADDR + index ) =
ADC_GetConversionValue(ADC3);
//}


/* Add the N ADC samples */
//SRAM_WriteBuffer(adc_sample, 0, 10);
```

```
for (index=0; index<N; index++)
{

avg_sample += *( float32_t*) (Bank1_NORFLASH1_ADDR  + ReadAddr);



      //avg_sample = *(__IO uint16_t*) (Bank1_SRAM3_ADDR  +
index);
      ReadAddr+= 4;
}
/* Compute the average of N ADC samples */
avg_sample /= (float32_t)N;
/* Return average value*/
return avg_sample;
}


/**
  * @}
  */

void  Save_DataAQC(void){


      uint32_t index=0;

      WRITE_READ_ADDR=0;

      ADC_SoftwareStartConv(ADC3);// empieza conversion y DMA se
activa


       while (DMA_GetCmdStatus(DMA_STREAM) != DISABLE)

       // mientras DMA esta transfiriendo datos a "testInput2"  y
esperate en el bucle
       {

       }





       for (index=0; index<N_samples; index++)
{

      //<conversion a float de los datos recogidos del adc ( ya
estamos en memoria externa)
      testInput_f32[index]= (testInput2_f32[index]-
2046)*8.05/10000;// Subimos el 0 (-2046) y convertimos valores
digitales a voltios (8.05/10000)

}
}
/**
```

```
   * @}
  */

void   Feature_analizing(void){

//Definició de paràmetres
float32_t mean=0.0f;
float32_t RMS=0.0f;
float32_t max=0.0f;
float32_t   min=0.0f;


float32_t f1mean=0.0f;
float32_t f1RMS=0.0f;
float32_t f1max=0.0f;
float32_t   f1min=0.0f;

float32_t f2mean=0.0f;
float32_t f2RMS=0.0f;
float32_t f2max=0.0f;
float32_t   f2min=0.0f;

float32_t f3mean=0.0f;
float32_t f3RMS=0.0f;
float32_t f3max=0.0f;
float32_t   f3min=0.0f;

//Thresholds
float32_t f2TH = 0.6;
float32_t f3TH = 0.6;

uint32_t item_index=0;

uint32_t    itemmin_index=0;
      arm_fir_instance_f32 S;
//Sortides dels filtres
       float32_t   *outputF32F1;

       float32_t   *outputF32F2;

       float32_t   *outputF32F3;

 outputF32F1 = &testOutput[0];

 outputF32F2 = &testOutput[0];

 outputF32F3 = &testOutput[0];

 inputF32 = &testInput_f32[0];// inputF32 es un puntero a la señal
de entrada



// calculo estadisticos para los valores de los filtros

arm_mean_f32 ( inputF32  , N_samples, &mean);
arm_rms_f32 ( inputF32, N_samples, &RMS);
arm_max_f32 ( inputF32, N_samples,&max , &item_index);
arm_min_f32 ( inputF32, N_samples, &min, &itemmin_index);
```

```
      //bandpass FIR1

       // outputF32 es un puntero a la señal filtrada
  arm_fir_init_f32(&S, NUM_TAPS, (float32_t
*)&firCoeffs100_150Khz[0], &firStateF32[0], blockSize);



  for(i=0; i < numBlocks; i++)
    {
      arm_fir_f32(&S, testInput_f32 + (i * blockSize), outputF32F1
+ (i * blockSize), blockSize);
    }


            arm_mean_f32 (outputF32F1  , N_samples, &f1mean);
arm_rms_f32 (outputF32F1, N_samples, &f1RMS);
arm_max_f32 (outputF32F1, N_samples,&f1max , &item_index);
arm_min_f32 (outputF32F1, N_samples, &f1min, &itemmin_index);


//BANDPASS FIR2

             // outputF32 es un puntero a la señal filtrada
  arm_fir_init_f32(&S, NUM_TAPS, (float32_t
*)&firCoeffs150_200Khz[0], &firStateF32[0], blockSize);



  for(i=0; i < numBlocks; i++)
    {
      arm_fir_f32(&S, testInput_f32 + (i * blockSize), outputF32F2
+ (i * blockSize), blockSize);
    }


            arm_mean_f32 (outputF32F2  , N_samples, &f2mean);
arm_rms_f32 (outputF32F2, N_samples, &f2RMS);
arm_max_f32 (outputF32F2, N_samples,&f2max , &item_index);
arm_min_f32 (outputF32F2, N_samples, &f2min, &itemmin_index);

//BANDPASS FIR3

             // outputF32 es un puntero a la señal filtrada
  arm_fir_init_f32(&S, NUM_TAPS, (float32_t
*)&firCoeffs200_250Khz[0], &firStateF32[0], blockSize);



  for(i=0; i < numBlocks; i++)
    {
      arm_fir_f32(&S, testInput_f32 + (i * blockSize), outputF32F3
+ (i * blockSize), blockSize);
    }
```

```
            arm_mean_f32 (outputF32F3   , N_samples, &f3mean);
arm_rms_f32 (outputF32F3, N_samples, &f3RMS);
arm_max_f32 (outputF32F3, N_samples,&f3max , &item_index);
arm_min_f32 (outputF32F3, N_samples, &f3min, &itemmin_index);


            //f1RMS= 0.65;
            //f1RMS= 0.73029;
            f1RMS= 0.7189;
            Feature_F2= f2RMS/f1RMS;
            Feature_F3= f3RMS/f1RMS;


            if(Feature_F3 > f3TH ||Feature_F2 > f2TH  ){

                  condition='S';


            }
            else{

                        condition='H';

                  }

                  if(Feature_F3>Feature_F2){

                        condition='F';

                  }


                  printf( "-- Original Signal--  \n \r"
                                                "              RMS: %.5f
        \n \n \r "
                                                "-- Filtered Signal1
From 100KHz to 150KHz --  \n \n \r"
                                                "                 RMS:
%.5f    \n \r\n \r "
                        "-- Filtered Signal2 From 150KHz to 200KHz
--   \n \r"
                                                "                 RMS:
%.5f    \n \r \n \r"
                        "-- Filtered Signal3 From 200KHz to 250KHz
--  \n \n \r"
                                                "                 RMS:
%.5f    \n \r \n \r"
                      "-- Fault Diagnosis --  \n \r \n \r"
                                                "-- Fature Calculated
From 150KHz to 200KHz Band--  \n \r"
                                                "
Programmed Threshold: %.5f      \n \r"
                                                "
Feature Actual Value: %.5f      \n \n \r"
                                          "-- Fature Calculated From
200KHz to 250KHz Band--  \n \r"
```

```
                                        "
Programmed Threshold: %.5f      \n \r"
                                        "           Feature
Actual Value: %.5f        \n \r\n \r"
                                        "                   System
Condition: %c        \n \n \r"

                                        "Where: \n \r "
                                        "                   'H'
is healthy condition      \n \r"
                                        "                   'S'
is suspicious condition        \n \r"
                                        "                   'F'
is faulty condition       \n \r"    ,
 (RMS ),
(f1RMS ),
(f2RMS ),
(f3RMS ),
(f2TH ),
(Feature_F2),
(f3TH ),
(Feature_F3),
(condition)
);

/*
for(i=0; i <  N_samples; i++)
    {

      printf("    Mean: %.2f   \n \r ",  array2[i]       );

            }*/

}

/**
  * @}
  */

/**
  * @}
  */

/**
  * @brief  ADC3 channel07 with DMA configuration
  * @param  None
  * @retval None
  */
void ADC3_CH7_DMA_Config(void)
{
      // structures declaration
  ADC_InitTypeDef       ADC_InitStructure;
  ADC_CommonInitTypeDef ADC_CommonInitStructure;
  DMA_InitTypeDef       DMA_InitStructure;
  GPIO_InitTypeDef      GPIO_InitStructure;

  /* Enable ADC3, DMA2 and GPIO clocks
****************************************/
```

```
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2 |
RCC_AHB1Periph_GPIOF, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);

  /* DMA2 Stream0 channel2 configuration
************************************/
  DMA_InitStructure.DMA_Channel = DMA_Channel_2;
  DMA_InitStructure.DMA_PeripheralBaseAddr =
(uint32_t)ADC3_DR_ADDRESS;//source adress
  DMA_InitStructure.DMA_Memory0BaseAddr =
(uint32_t)&testInput2_f32; // destination adress
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;//modo de
funcionamiento (periferico a memoria RAM INTERNA!!!! aun no hemos
convertido a float)
  DMA_InitStructure.DMA_BufferSize = 20000;// numero de datos a
enviar (DMA is the flow controller)
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;// Los
datos recogidos se ponen en un array por lo tanto hay un
incremento unitario en la direccion para cada dato del ADC
  DMA_InitStructure.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord;//Formato de los datos a
transferir (periferico)
  DMA_InitStructure.DMA_MemoryDataSize =
DMA_MemoryDataSize_HalfWord;//Formato de los datos a transferir
(memoria) Half word 16 bits
  DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;// Modo normal o
circular
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;
  DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable; // No
espera a que se llene la FIFO envia el dato directamente al llegar
a la FIFO
  DMA_InitStructure.DMA_FIFOThreshold =
DMA_FIFOThreshold_HalfFull;
  DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;//
Numero de paquetes de 16 bits que envias, en este caso 1
  DMA_InitStructure.DMA_PeripheralBurst =
DMA_PeripheralBurst_Single;// Lo msimo pero para el periferico
  DMA_Init(DMA2_Stream0, &DMA_InitStructure);// Una vez
inicializada la estructura la cargamos y hacemos un enable
  DMA_Cmd(DMA2_Stream0, ENABLE);

  /* Configure ADC3 Channel7 pin as analog input
******************************/


      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;//PF6(CHANNEL 4)
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN ;// CHANNEL 4 IN
ANALOGUE MODE
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;// Configuracion
del pin del ADC
  GPIO_Init(GPIOF, &GPIO_InitStructure);//Cargar la configuracion
de la estructura


  /* ADC Common Init
*********************************************************/
  ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;// Adc
channels used for the signal acquisition
```

```
  ADC_CommonInitStructure.ADC_Prescaler =
ADC_Prescaler_Div2;//HCLK/2  the clock preescaler 60MHz del reloj
del micro /2 = 30 Mhz ( preescaler de 2)
  ADC_CommonInitStructure.ADC_DMAAccessMode =
ADC_DMAAccessMode_Disabled;//Si no usamos la FIFO ( llena entera
para enviar) esto tampoco.
  ADC_CommonInitStructure.ADC_TwoSamplingDelay =
ADC_TwoSamplingDelay_5Cycles;//Antes de empezar se tiene que
esperar cinco ciclos, sino las primeras acq son erroneas. Necesita
llegar al regimen permanente.
  ADC_CommonInit(&ADC_CommonInitStructure);//Cargamos la
configuracion en el adc

  /* ADC3 Init
*****************************************************************/
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;//
Resolucion de 12 bits (se puede 8 10 12 bits)
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;//Escanear
diferentes canales ( 1 solo adc escanea diferentes canales)
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;// Acquiere N
samples o continues acquisition. Es continua pq el dma ya coge las
20.000 da igual que siga cogiendo, cuando el dma vuelva a requerir
muestras las coge del buffer del adc
  ADC_InitStructure.ADC_ExternalTrigConvEdge =
ADC_ExternalTrigConvEdge_None;//Activado por evento externo ( a la
entrada de un pin que cuadno se pone en 1... etc
  ADC_InitStructure.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_T1_CC1;
  ADC_InitStructure.ADC_DataAlign =
ADC_DataAlign_Right;//Alineación de los datos, como son dos
registros, los puede alinear de formas diferentes
  ADC_InitStructure.ADC_NbrOfConversion = 1;// cantidad de
conversiones
  ADC_Init(ADC3, &ADC_InitStructure); // Cargar la configuracion
realizada

  /* ADC3 regular channel7 configuration
*************************************/

ADC_RegularChannelConfig(ADC3, ADC_Channel_4, 1,
ADC_SampleTime_3Cycles);//3 ciclos mas los 12 bits = formula del
reloj (15)

 /* Enable DMA request after last transfer (Single-ADC mode) */
  ADC_DMARequestAfterLastTransferCmd(ADC3, ENABLE);//Cuando acabes
una transferencia, empieza otra

  /* Enable ADC3 DMA */
  ADC_DMACmd(ADC3, ENABLE);//Habilita el ADC (activas el clock)

  /* Enable ADC3 */
  ADC_Cmd(ADC3, ENABLE);// lo mismo que arriba
}
/*****END OF FILE****/
```

Filter design fucntion

```
/*Funció per generar els filtres amb els coeficients
 * Filter Coefficients (C Source) generated by the Filter Design
and Analysis Tool
 *
 * Generated by MATLAB(R) 7.14 and the Signal Processing Toolbox
6.17.
 *
 * Generated on: 23-Oct-2012 11:19:58
 *
 */

/*
 * Discrete-Time FIR Filter (real)
 * -------------------------------
 * Filter Structure  : Direct-Form FIR
 * Filter Length     : 81
 * Stable            : Yes
 * Linear Phase      : Yes (Type 1)
 */

/* General type conversion for MATLAB generated C-code  */
//#include "tmwtypes.h"
/*
 * Expected path to tmwtypes.h
 * C:\Program Files\MATLAB\R2012a\extern\include\tmwtypes.h
 */
/*
 * Warning - Filter coefficients were truncated to fit specified
data type.
 *   The resulting response may not match generated theoretical
response.
 *   Use the Filter Design & Analysis Tool to design accurate
 *   single-precision filter coefficients.
 */
 #ifndef __fdacoefs_H
#define __fdacoefs_H



#define NUM_TAPS           80// 11
#define BLOCK_SIZE          160//12
#define TEST_LENGTH_SAMPLES 20000//620//120
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1];//150
static float32_t testOutput[TEST_LENGTH_SAMPLES];

const float32_t firCoeffs100_150Khz [NUM_TAPS] = {
  -0.007330250461,-0.008449853398, 0.006846241653,-0.002020535758,
0.001741481014,
  -0.002436416224,-0.003162794979,-0.006198007148,-0.007415499073,
-0.00797412917,
  -0.006331112236,-0.003000860801, 0.001798114972, 0.006887868512,
0.01110295579,
    0.01326324791,  0.01281652134, 0.009950355627, 0.005698841065,
0.001663408941,
  -0.0004606954171,0.0005138901761, 0.004744122736,
0.01113205403,   0.0173912067,
    0.02065248415,  0.01822424307, 0.008519558236,-0.008213017136,
-0.02961835079,
```

```
   -0.05150202662, -0.06868047267, -0.07615797222, -0.07049331069,
-0.0508318767,
   -0.01938140392,  0.01881279796,  0.05683692917,  0.08736464381,
0.1043399051,
     0.1043399051,  0.08736464381,  0.05683692917,  0.01881279796,
-0.01938140392,
    -0.0508318767, -0.07049331069, -0.07615797222, -0.06868047267,
-0.05150202662,
   -0.02961835079,-0.008213017136, 0.008519558236,  0.01822424307,
0.02065248415,
     0.0173912067,  0.01113205403,
0.004744122736,0.0005138901761,-0.0004606954171,
   0.001663408941, 0.005698841065, 0.009950355627,  0.01281652134,
0.01326324791,
    0.01110295579, 0.006887868512, 0.001798114972,-
0.003000860801,-0.006331112236,
   -0.00797412917,-0.007415499073,-0.006198007148,-
0.003162794979,-0.002436416224,
   0.001741481014,-0.002020535758, 0.006846241653,-
0.008449853398,-0.007330250461
};

const float32_t firCoeffs150_200Khz[NUM_TAPS] = {
  -0.008169984445,-0.006895589177,  0.00705046393,-
0.003107855096,0.0007259324775,
  -0.004652773961,-0.003666206263,-0.004341358319,-
0.0005558090634, 0.003695374122,
   0.008943026885,  0.01170191541,  0.01111036167,
0.006340683904,-0.0008265665965,
  -0.007852448151, -0.01185920462, -0.01144369412,-
0.007322463673,-0.002188335871,
  0.0007112480234,-0.0006745941937,-0.005669266917, -
0.01082510222, -0.01131590735,
  -0.003535905387,  0.01251648366,  0.03209616616,  0.04693885148,
0.04842497036,
    0.03163244203,-0.001594171859, -0.04206489399, -0.07590957731,
-0.08958625793,
   -0.07538210601, -0.03513024375,  0.01959820278,  0.07120864838,
0.1023571417,
     0.1023571417,  0.07120864838,  0.01959820278, -0.03513024375,
-0.07538210601,
   -0.08958625793, -0.07590957731, -0.04206489399,-0.001594171859,
0.03163244203,
    0.04842497036,  0.04693885148,  0.03209616616,
0.01251648366,-0.003535905387,
   -0.01131590735, -0.01082510222,-0.005669266917,-
0.0006745941937,0.0007112480234,
  -0.002188335871,-0.007322463673, -0.01144369412, -
0.01185920462,-0.007852448151,
  -0.0008265665965, 0.006340683904,  0.01111036167,
0.01170191541, 0.008943026885,
   0.003695374122,-0.0005558090634,-0.004341358319,-
0.003666206263,-0.004652773961,
  0.0007259324775,-0.003107855096,  0.00705046393,-
0.006895589177,-0.008169984445
};

const float32_t firCoeffs200_250Khz[NUM_TAPS] ={
```

```
   -0.01008437667,-0.003269691253, 0.005598692689,-
0.003909661435,0.0004079719074,
  -0.004923087545,0.0005912245251, 0.001684408169, 0.008161524311,
0.007740872446,
   0.005651022308,-0.002944818465,-0.009820912033, -
0.01379609387,-0.009734123945,
  -0.001475890633, 0.007581821643,  0.01110809669,  0.00864396058,
0.002616353566,
  -0.0009106203215,  0.00074476999, 0.005759826396,
0.007305501495,-0.0001687859476,
   -0.01610352844, -0.03133751824, -0.03316399828, -0.01377823483,
0.02193515375,
    0.05641343817,   0.0679121092,  0.04397584125,-0.008807535283,
-0.06549733877,
   -0.09522501379, -0.07868025452, -0.02059105411,  0.05114633963,
0.09977574646,
    0.09977574646,  0.05114633963, -0.02059105411, -0.07868025452,
-0.09522501379,
   -0.06549733877,-0.008807535283,  0.04397584125,   0.0679121092,
0.05641343817,
    0.02193515375, -0.01377823483, -0.03316399828, -0.03133751824,
-0.01610352844,
  -0.0001687859476, 0.007305501495, 0.005759826396,
0.00074476999,-0.0009106203215,
   0.002616353566,  0.00864396058,  0.01110809669,
0.007581821643,-0.001475890633,
  -0.009734123945, -0.01379609387,-0.009820912033,-0.002944818465,
0.005651022308,
   0.007740872446, 0.008161524311,
0.001684408169,0.0005912245251,-0.004923087545,
  0.0004079719074,-0.003909661435, 0.005598692689,-0.003269691253,
-0.01008437667
};

uint32_t numBlocks = TEST_LENGTH_SAMPLES/BLOCK_SIZE;

#endif /* __fdacoefs_H*/
```