INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

MAESTRÍA EN DISEÑO ELECTRÓNICO

REPORTE DE FORMACIÓN COMPLEMENTARIA EN ÁREA DE CONCENTRACIÓN EN SISTEMAS EMBEBIDOS Y TELECOMUNICACIONES

Reporte técnico que para obtener el grado de

MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: Américo Lorenzana Gutiérrez

Asesor: Dr. Omar Longoria Gandara
Asesor: Dr. Zabdiel Brito Brito

Tlaquepaque, Jalisco. Agosto de 2016.

MAESTRO EN INGENIERÍA (2016) ITESO

Maestría en Diseño Electrónico Tlaquepaque, Jal., México

ÁREA DE CONCENTRACIÓN: **“Sistemas embebidos y Telecomunicaciones”**

AUTOR: Américo Lorenzana Gutiérrez

Ingeniero en Mecatrónica (Universidad Panamericana, Zapopan, Jalisco)

REVISORES: Dr. Omar H. Longoria Gandara

Dr. Zabdiel Brito Brito

NÚMERO DE PÁGINAS: 14

Contenido

Introducción

Los Sistemas embebidos son pequeños, rápidos y herramientas muy poderosas que ayudan a realizar tareas fundamentales en nuestras vidas, estos sistemas no están diseñados para propósitos generales como lo son las computadoras personales, en su lugar son sistemas diseñados con un rango de funciones limitadas para una tarea en específico, típicamente estos sistemas tienen fuertes restricciones en costo, consumo de potencia y confiabilidad.

Este reporte lo he destinado a presentar algunos proyectos desarrollados durante el curso de la maestría en diseño electrónico, el área de concentración elegida es *Sistemas Embebidos y Telecomunicaciones*, los proyectos enlistados a continuación fueron seleccionados por la cantidad de aportación a la formación académica.

De la asignatura de Procesamiento Digital de Señales

- Diseño de un reproductor de audio usando un MCU de 32bits

De la asignatura de Sistema Embebidos avanzados

- Diseño de Software para un vehículo aéreo no tripulado

De la asignatura de Diseño e implementación de Sistemas Operativos

- Diseño de un sistema de archivos

Se seleccionó un proyecto de asignaturas diferentes todas relacionadas al diseño de sistemas embebidos. Los proyectos desarrollados me facilitaron la generación de conocimiento sobre el diseño y funcionamiento de los componentes fundamentales dales como: Procesadores, Memorias, Múltiples periféricos, Compiladores, Sistemas Operativos de tiempo Real, etc.

En todos los proyectos se utilizó un lenguaje de programación basado en “C”. Los dos primeros proyectos mencionados tuvieron limitaciones en utilización de memoria y capacidad de procesamiento, esto me ayudo a entender las complicaciones que se pueden presentar y las razones para elegir diferentes componentes para maximizar su confiabilidad, siendo que uno de sus

requerimientos es el funcionamiento continuo sin errores. El tercer proyecto se realizó en ambiente Linux, con recursos suficientes para cumplir los objetivos planteados, el objetivo fue el diseño desde cero del sistema de archivos.

Los proyectos fueron desarrollados cumpliendo sus requerimientos alcanzando los objetivos propuestos, se utilizaron diferentes tarjetas de evaluación, al inicio desconocía las tecnologías que se iban a utilizar a lo largo de la maestría, esto despertó gran interés en mí.

La influencia que tuve por parte de mi trabajo (Ingeniero de Software en sistemas embebidos) fue bastante, me ayudó a dirigir mi formación en el área relacionada. Los proyectos presentados en este reporte fueron la apertura para seguir investigando en el área de sistemas embebidos utilizando sistemas operativos en tiempo real y técnicas de programación para la optimización del uso de recursos en sistemas de bajo costo.

1. Resumen de los proyectos realizados

A continuación se describen los tres proyectos realizados, sin profundizar en detalles específicos (para esto dirigirse a la sección de apéndices).

1.1. Diseño de un reproductor de audio usando un MCU de 32 bits

1.1.1 Introducción

El objetivo del proyecto es realizar un "*Playback and recording*" basado en un microcontrolador STM32F407VG6 y una tarjeta de evacuación *STM32F4 DISCOVERY*, dicha tarjeta contiene un sensor de audio y un DAC para audio con un controlador para altavoces.

El audio se grabara en un archivo *WAVE* y será almacenado en una memoria USB Flash utilizando un controlador *USB OTG HOST*, al momento de grabar el audio se podrá filtrar digitalmente utilizando tres diferentes filtros.

1.1.2 Antecedentes

El conocimiento necesario para desarrollar este proyecto se basa en desarrollo de código en lenguaje C, el cual ya contaba con suficiente experiencia. Por otra parte, el diseño de filtros digitales fue un tema que se tomó durante el semestre en la materia de Procesamiento digital de Señales, otro conocimiento necesario para este proyecto fue librerías de *Pulse Density Modulated* para la grabación de audio.

1.1.3 Solución desarrollada

El proyecto se realizó utilizando librerías proporcionadas por *STMicroelectronics*, se usaron librerías para el uso del micrófono embebido, sistema de archivos para montar la memoria USB, para generar el archivo WAVE, entre otros.

En el proyecto se incluyeron tres filtros digitales, Pasa altas, Pasa Bajas y Pasa banda, estos filtros se codificaron utilizando lenguaje C, el reporte del diseño de los filtros de respuesta finita al impulso se incluye en el anexo.

1.1.4 Análisis de resultados

El resultado satisfizo en plenitud los requerimientos del proyecto. Se logró de manera exitosa, con el uso del hardware proporcionado, el grabando audio y almacenándolo en formado *WAVE* en la memoria externa USB. El audio se pudo acceder para ser reproducido por el CODEC embebido, una vez en reproducción, se podían apreciar los cambios realizados por los filtros digitales correspondientes. El proyecto fue exitoso.

1.1.5 Conclusiones

Este proyecto demandó mucho tiempo de investigación por mi cuenta, debido a que la tarjeta de evaluación tenía muchos drivers y debí investigar cómo usarlos. Comprendí el

funcionamiento de micrófonos embebidos y la aplicación de filtros digitales, este proyecto lo realice de manera individual aumentando la carga de trabajo pero al final fue muy gratificante.

1.2. Diseño de Software para un vehículo aéreo no tripulado

1.2.1 Introducción

En el proyecto se considera en general un módulo para el algoritmo de navegación de un vehículo aéreo no tripulado, este módulo consiste en simular un sensor cuya respuesta está definida por una formula.

El objetivo de proyecto es resolver utilizando librerías de punto fijo y punto flotante, estas librerías deben ser hechas por el estudiante. Significa que el rendimiento del algoritmo se verá afectado directamente por el manejo numérico que se obtendrá en cada librería referentemente.

1.2.2 Antecedentes

Anteriormente en clase se había abordado el tema de optimización de código para sistemas de recursos reducidos, esto implicaba en realizar análisis matemáticos para saber los rangos numéricos de los sistemas, con el objetivo de maximizar los recursos usados. Esto fue algo prioritario debido a que este proyecto se realizó en un microcontrolador de 16bit de *Texas Instruments* MSP430 con recursos bastante limitados.

1.2.3 Solución desarrollada

La solución desarrollada puede manejar tanto manejo numérico en punto fijo o punto flotante, pero no puede utilizarse los dos al mismo tiempo, también se realizó un análisis matemático el cual sugirió que se deberían usar amenos 10bits para las variables, ya que 8 bits era insuficiente y podría desencadenar un *OverFlow* del sistema.

Las librerías desarrolladas para punto flotante se hicieron en base de número de 16bits, la resolución era mayor pero el espacio utilizado para código era mucho mayor también.

1.2.4 Análisis de resultados

Algunas desventajas de los resultados obtenidos:

- Las funciones desarrolladas solo funcionan para ESTE proyecto, el código no es modular.
- Se tuvo que utilizar optimizaciones al momento de compilar el código de las librerías de punto flotante para que pudiera ser programado en el microcontrolador.
- Debido a la metodología utilizada en el código, era muy difícil agregar mejoras en los algoritmos, un código de difícil mantenimiento.

Las ventajas de los resultados fueron:

- La precisión obtenida por ambas librerías eran muy altas, superaron las expectativas.
- Los cálculos eran realizados en tiempos muy cortos, alcanzando una tasa de actualizaciones cercanas a las de 1Mhz.

1.2.5 Conclusiones

Las operaciones en punto fijo fueron realmente muy sencillas de entender y realizar, no se requirió mucho tiempo de análisis.

Por otra parte, las operaciones de punto flotante me tomo mucho tiempo de análisis para entender cómo funcionan y mucho más tiempo para poder implementar las librerías de una manera óptima, encontré muy poca documentación en internet que explicaba cómo realizar las operaciones en punto flotante con bases binarias. Las funciones para calcular valores en punto flotante ocupo mucha más memoria flash que las librerías en punto fijo. Esto me hizo entender la importancia de selección de librerías matemáticas al momento en empezar un proyecto en un ambiente embebido con recursos limitados.

1.3. Diseño de un sistema de archivos

1.3.1 Introducción

El objetivo de este proyecto fue realizar un sistema de archivos, el cual tuviera la capacidad de escritura/lectura de archivos, manejo de espacio en memoria física para determinar dónde y cómo ser guaran los archivos.

1.3.2 Antecedentes

En proyectores anteriores había tenido la oportunidad de utilizar Sistemas de Archivos para microcontroladores, nunca había tenido la oportunidad de introducirme más en el funcionamiento, solo había tenido la oportunidad de usar los sistemas de archivos.

1.3.3 Solución desarrollada

El sistema de archivos fue creado en C, se utilizaron librerías estándar. La arquitectura del sistema de archivos fue bastante sencilla, pero abarco desde creación, edición y eliminación de archivos de manera dinámica en la memoria disponibles.

Se desarrolló un *Shell* para tener una interface con el sistema de archivos, de esta manera se podía ingresar comandos para la manipulación de archivos.

1.3.4 Análisis de resultados

Los resultados fueron favorables, el sistema de archivos funciono con éxito, más sin embargo la arquitectura fue muy pobre y de muy difícil mantenimiento, esto género que varios errores surgieran, pero lo el objetivo del proyecto se cumplió.

1.3.5 Conclusiones

Este proyecto fue complicado, debido a que muchas funciones requeridas para el desarrollo d este proyecto no las conocía, muchos conceptos eran nuevos. El proyecto cumplió con el cometido de enriquecerme enormemente con nuevos términos y conocimiento. Añadiendo este proyecto se desarrolló en Linux, por lo tanto fue todo una experiencia totalmente nueva y grata.

2. Conclusiones

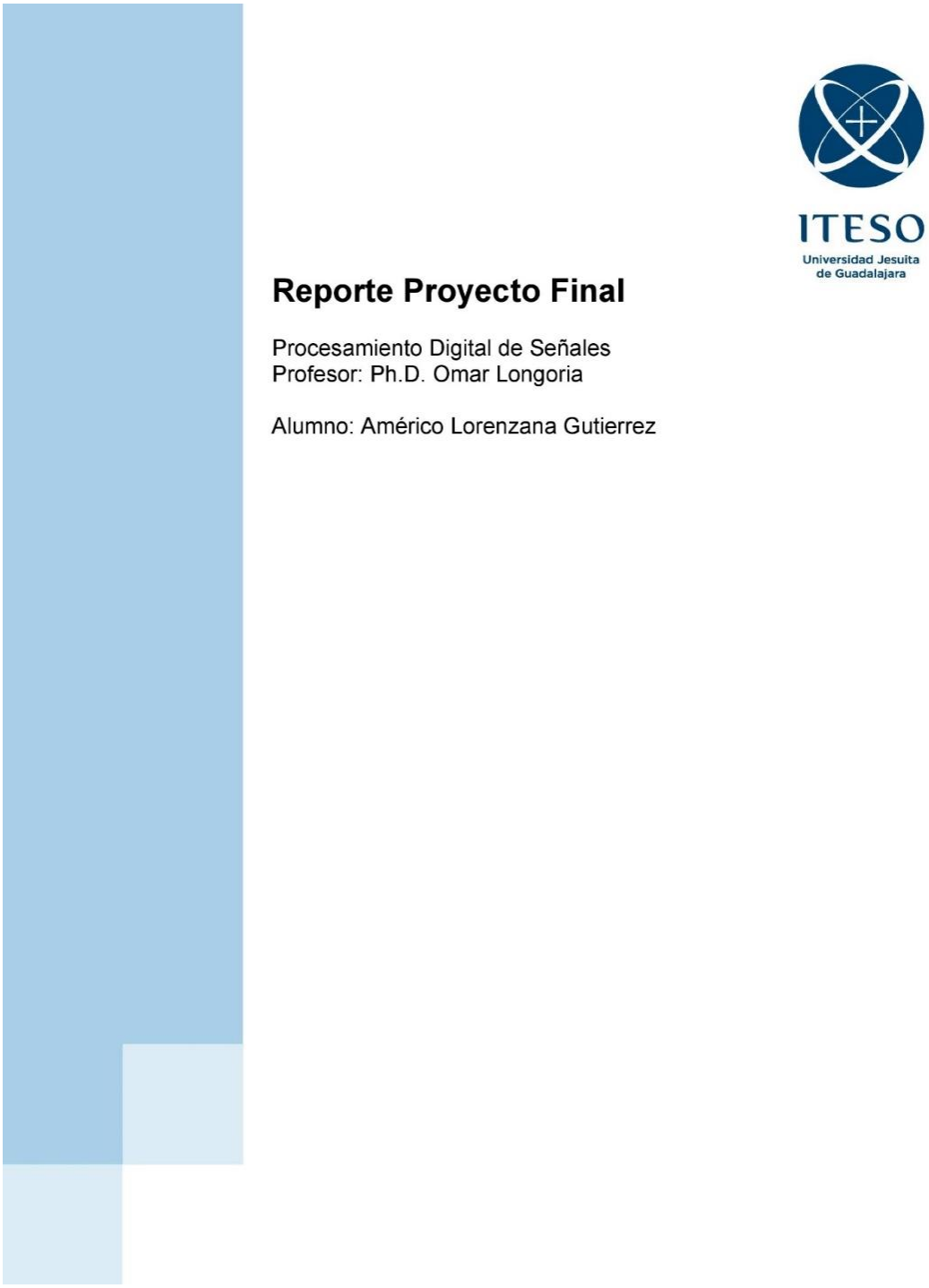
La realización de los proyectos expuestos en este reporte facilito la generación de nuevo conocimiento para ser usado en mi vida laboral, hago mención que las herramientas adquiridas en el transcurso de diferentes asignaturas las utilizo día con día para desempeñar mi trabajo.

Además, por la experiencia obtenida me siento más capaz de desarrollar proyectos donde no tenga mucho conocimiento del tema, debido a que desarrolle una capacidad sistemática para llevar acabo estrategias para descubrir algo, esto es transcendente ya es la única manera para poder aprender a utilizar nuevas tecnologías la cual pienso que es lo más apreciado que aprendí en el trascurso de la MDE.

3. Apéndices

3.1. Apéndice A.

Reporte “Diseño de un reproductor de audio usando un MCU de 32 bits”



Introducción

El objetivo del proyecto es realizar un “Playback and recording” basado en un microcontrolador STM32F407VG6 y una tarjeta de evacuación con el siguiente hardware:

-MP45DT02 sensor de audio MEMS omnidireccional.
-CS43L22 DAC para audio con un controlador para altavoces de clase D.
-LEDs de propósito general
-Un botón de propósito general

La reproducción de audio se hace desde un archivo WAVE que puede ser leído a partir de una memoria Flash utilizando un controlador FileSystem y un controlador USB OTG HOST.

El proceso de grabación utiliza un micrófono MP45DT02 con una librería de PDM (Pulse Density Modulated) que toma los datos del micrófono y los convierte a un formato de 16-bits pulse-code modulation (PCM).

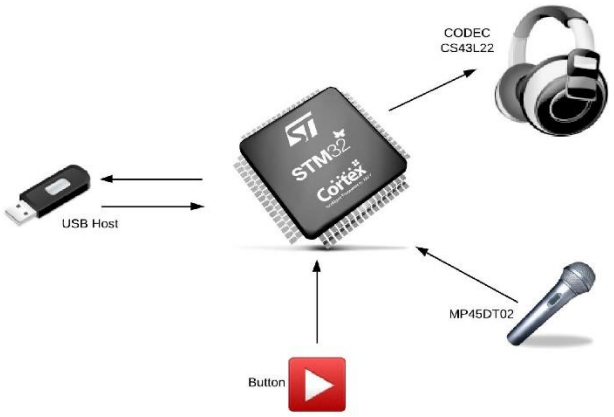
Los datos obtenidos de 16-bits son procesados por un filtro digital IIR, se implemento 3 variantes de filtros:

-Filtro pasa bajas, Fc = 100Hz
-Filtro pasa bandas, Fc = 700Hz a 1,500Hz
-Filtro pasa altas, Fc = 2,000Hz

La señal de audio generada es guardada en formato .wave, con la finalidad que pueda ser reproducido.

Descripción de funcionamiento

La aplicación “Playback and recording” puede reproducir y grabar audio desde una memoria Flash externa en formato .wave, se puede cambiar la acción de reproducción a grabado y viceversa utilización un botón de propósito general.



La aplicación basada en un microcontrolador STM32F407VG6 tiene las siguientes funcionalidades:

-Micrófono MEMS
-CODEC de audio
-Audífonos
-Memoria Flash Externa

Los principales módulos son los siguientes:

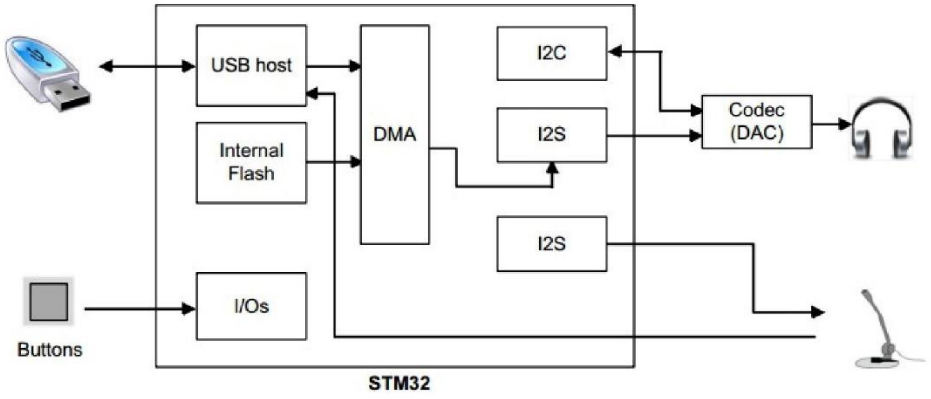
-USB peripheral: configura en modo HOST, se implemento MSC (mass storage class) para transmitir y recibir datos de la memoria Flash.

-I2C peripheral: configura en modo maestro para transmitir datos al codec de audio, también se utiliza para recibir datos del micrófono.

-DMA: se utiliza para transmitir/recibir datos de un buffer al I2S, reduce significativamente la carga al CPU.

-I2C peripheral: se utiliza para controlar varios dispositivos externos como el codec de audio.

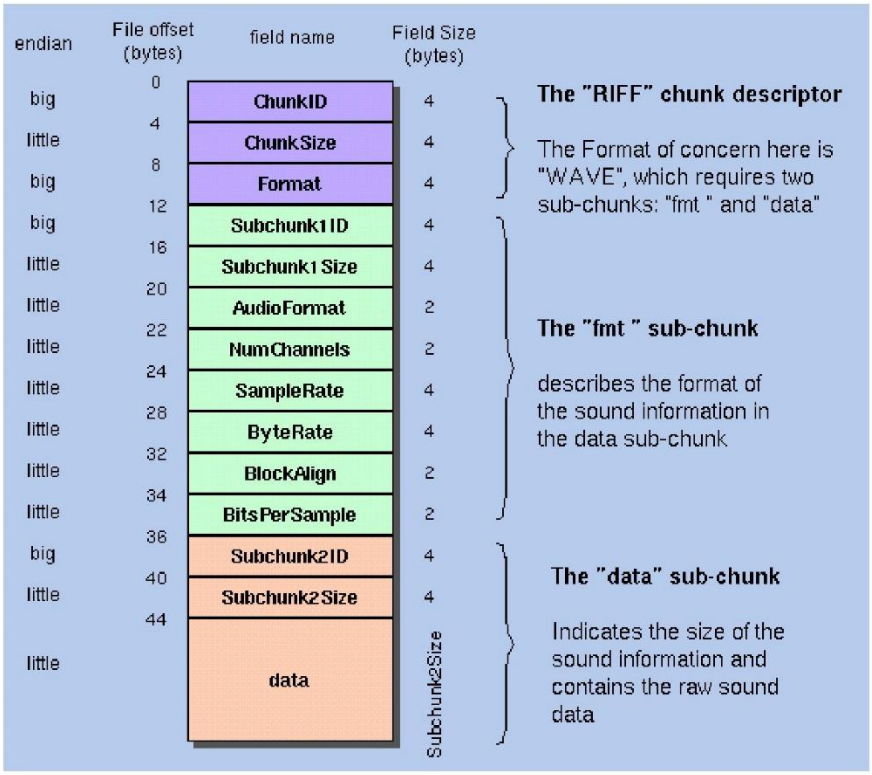
-User Botton: se utiliza para la aplicación.



Reproducción desde USB flash

Cualquier archivo wave puede ser abirto desde la USB flash utilizando un sistema de ficheros FatFs, siendo tranferido a un bloque de SRAM utilizando el DMA para reducir carga de CPU.

El encabezado del archivo wave contiene la siguiente información:



-Frecuencia de muestreo = 8,000Hz
-Formato de audio = PCM (formato sin compreción)
-Bit por muestra = 16bits
-Numero de Canales = 2

La configuración del Codec de audio es obtenida del encabezado del archivo wave, al finalizar la configuración un LED color azul empieza a parpadear para indicar configuración exitosa.

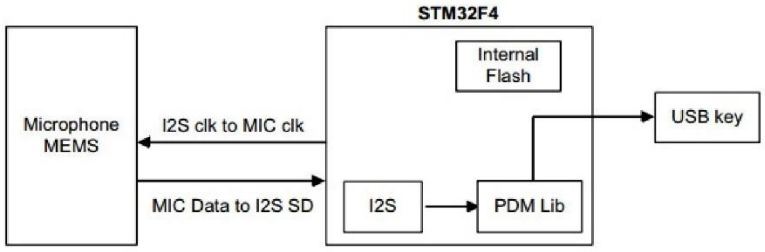
La reproducción maneja dos buffers de memoria, el primer buffer se usa para almacenar datos del archivo wave leídos desde la USB Flash utilizando el sistema de archivos FatFs, cuando el buffer esta completamente lleno el DMA manda la información del buffer al Codec de audio, en el mismo momento los datos leído del USB flash se almacenan en el segundo buffer.

Los dos buffers son intercalados infinitamente para poder reproducir sin ningún retardo.

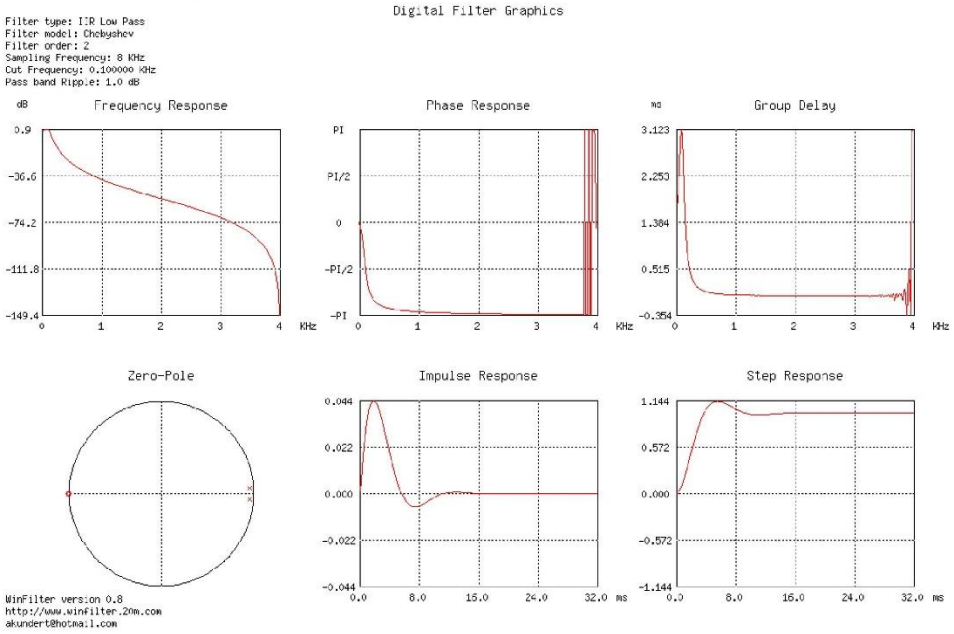
Grabado de audio y filtrado

La aplicación solo empieza a grabar audio cuando detecta que esta conectado un USB Flash, después configura el periférico I2S como master para generar una señal de reloj de 1.024Mhz para alimentar el micrófono digital, La señal de 1,024Mhz se calculo partiendo de la salida de audio requerida de 16Khz y el factor de decimación (64) (16K x 64 = 1.024Mhz)

El proceso de filtrado se utiliza la biblioteca de software de decodificación de audio PDM. Esta biblioteca implementa varios filtros para la salida de la señal de alta frecuencia de 1 bit PDM desde un micrófono digital y la transforma en una de 16 bits PCM con una frecuencia de audio señalada (8Khz).



Al obtener los datos depues del filtrado de decimación que impleta 3 clases de filtros IIR:
-Filtro Pasa Bajas



Filter type: Low Pass
Filter model: Chebyshev
Filter order: 2
Sampling Frequency: 8 KHz
Cut Frequency: 0.100000 KHz
Pass band Ripple: 1.000000 dB
Coefficents Quantization: 16-bit

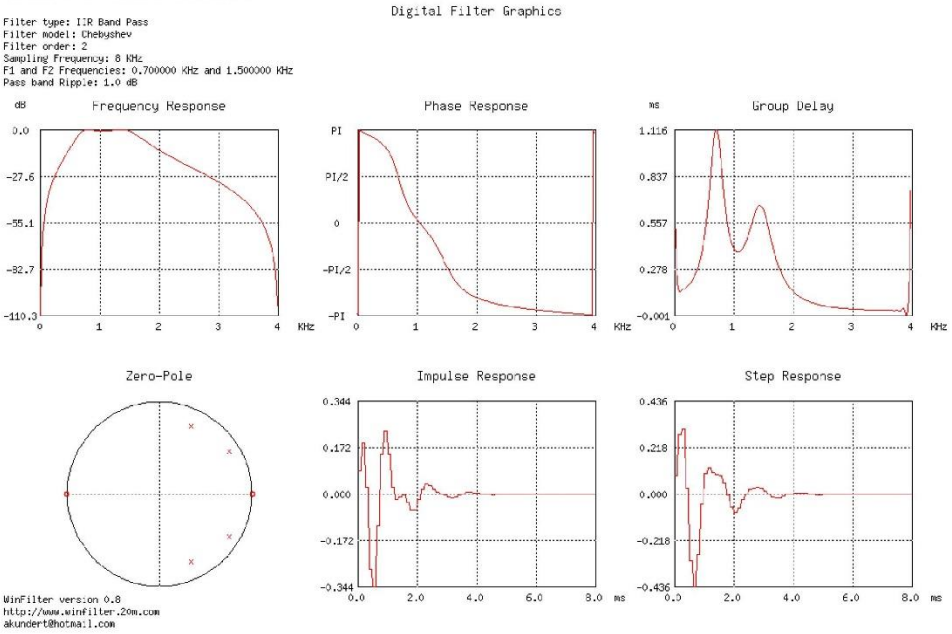
Z domain Zeros
z = -1.000000 + j 0.000000
z = -1.000000 + j 0.000000

Z domain Poles
z = 0.955444 + j -0.067421
z = 0.955444 + j 0.067421

A = [0.83398 1.667968 0.83398];

B = [1 -1.910088 0.91741];

-Filtro Pasa Bandas



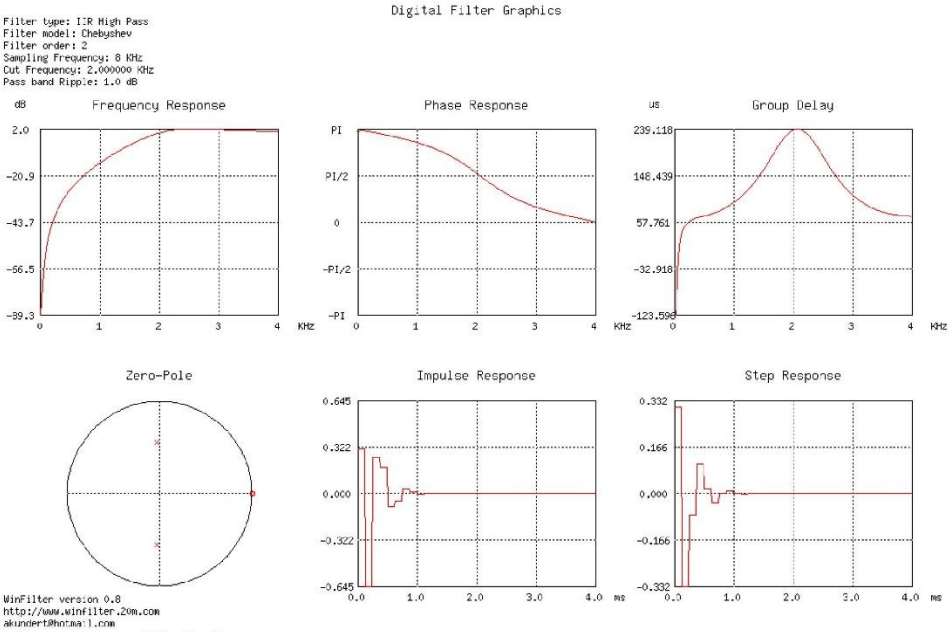
Filter type: Band Pass
Filter model: Chebyshev
Filter order: 2
Sampling Frequency: 8 KHz
Fc1 and Fc2 Frequencies: 0.700000 KHz and 1.500000 KHz
Coefficents Quantization: 16-bit

Z domain Zeros
z = -1.000000 + j 0.000000
z = -1.000000 + j 0.000000
z = 1.000000 + j 0.000000
z = 1.000000 + j 0.000000

Z domain Poles
z = 0.340560 + j -0.736519
z = 0.340560 + j 0.736519
z = 0.751908 + j -0.466619
z = 0.751908 + j 0.466619
A = [1.402832 0 -2.805786 0 1.4028];

B = [1 -2.18493 2.465820 -1.5235595 0.51562];

-Filtro Pasa Altas



Filter type: High Pass
Filter model: Chebyshev
Filter order: 2
Sampling Frequency: 8 KHz
Cut Frequency: 2.000000 KHz
Pass band Ripple: 1.000000 dB
Coefficents Quantization: 16-bit

Z domain Zeros
z = 1.000000 + j 0.000000
z = 1.000000 + j 0.000000

Z domain Poles
z = -0.032028 + j -0.559410
z = -0.032028 + j 0.559410

A = [0.312469 -0.62493 0.3124694];

B = [1 .064056 0.139648];

Anexo

Main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/** @addtogroup STM32F4-Discovery_Audio_Player_Recorder
  * @{
  */

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
#if defined MEDIA_USB_KEY
 USB_OTG_CORE_HANDLE          USB_OTG_Core;
 USBH_HOST                    USB_Host;
#endif

RCC_ClocksTypeDef RCC_Clocks;
__IO uint8_t RepeatState = 0;
__IO uint16_t CCR_Val = 16826;
extern __IO uint8_t LED_Toggle;

/* Private function prototypes -----------------------------------------------*/
static void TIM_LED_Config(void);
/* Private functions ---------------------------------------------------------*/

/**
  * @brief  Main program.
  * @param  None
  * @retval None
*/
int main(void)
{
  /* Initialize LEDS */
  STM_EVAL_LEDInit(LED3);
  STM_EVAL_LEDInit(LED4);
  STM_EVAL_LEDInit(LED5);
  STM_EVAL_LEDInit(LED6);

  /* Green Led On: start of application */
  STM_EVAL_LEDOn(LED4);

  /* SysTick end of count event each 10ms */
  RCC_GetClocksFreq(&RCC_Clocks);
  SysTick_Config(RCC_Clocks.HCLK_Frequency / 100);

  /* Configure TIM4 Peripheral to manage LEDs lighting */
  TIM_LED_Config();

  /* Initialize the repeat status */
  RepeatState = 0;
  LED_Toggle = 7;

#if defined MEDIA_IntFLASH
```

```
  WavePlayBack(I2S_AudioFreq_48k);
  while (1);

#elif defined MEDIA_USB_KEY

  /* Initialize User Button */
  STM_EVAL_PBInit(BUTTON_USER, BUTTON_MODE_EXTI);

  /* Init Host Library */
  USBH_Init(&USB_OTG_Core, USB_OTG_FS_CORE_ID, &USB_Host, &USBH_MSC_cb,
&USR_Callbacks);

  while (1)
  {
    /* Host Task handler */
    USBH_Process(&USB_OTG_Core, &USB_Host);
  }

#endif

}

/**
  * @brief  Configures the TIM Peripheral for Led toggling.
  * @param  None
  * @retval None
  */
static void TIM_LED_Config(void)
{
  TIM_OCInitTypeDef  TIM_OCInitStructure;
  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
  NVIC_InitTypeDef NVIC_InitStructure;
  uint16_t prescalervalue = 0;

  /* TIM4 clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

  /* Enable the TIM3 gloabal Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  /* Initialize Leds mounted on STM324F4-EVAL board */
  STM_EVAL_LEDInit(LED3);
  STM_EVAL_LEDInit(LED4);
  STM_EVAL_LEDInit(LED6);

  /* Compute the prescaler value */
  prescalervalue = (uint16_t) ((SystemCoreClock ) / 550000) - 1;

  /* Time base configuration */
```

```
  TIM_TimeBaseStructure.TIM_Period = 65535;
  TIM_TimeBaseStructure.TIM_Prescaler = prescalervalue;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);

  /* Enable TIM4 Preload register on ARR */
  TIM_ARRPreloadConfig(TIM4, ENABLE);

  /* TIM PWM1 Mode configuration: Channel */
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = CCR_Val;
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

  /* Output Compare PWM1 Mode configuration: Channel2 */
  TIM_OC1Init(TIM4, &TIM_OCInitStructure);
  TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Disable);

  /* TIM Interrupts enable */
  TIM_ITConfig(TIM4, TIM_IT_CC1 , ENABLE);

  /* TIM4 enable counter */
  TIM_Cmd(TIM4, ENABLE);
}

#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
```

Main.h

```
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx.h"
#include "stm32f4_discovery.h"
#include "stm32f4_discovery_audio_codec.h"
#include "stm32f4_discovery_lis302dl.h"
#include <stdio.h>
#include "stm32f4xx_it.h"
#include "waveplayer.h"

#ifdef MEDIA_USB_KEY
 #include "waverecorder.h"
 #include "usb_hcd_int.h"
 #include "usbh_usr.h"
 #include "usbh_core.h"
 #include "usbh_msc_core.h"
 #include "pdm_filter.h"
#endif

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/

#if defined MEDIA_USB_KEY
  /* You can change the Wave file name as you need, but do not exceed 11 characters
*/
  #define WAVE_NAME "0:audio.wav"
  #define REC_WAVE_NAME "0:rec.wav"

  /* Defines for the Audio recording process */
  #define RAM_BUFFER_SIZE         1500  /* 3Kbytes (1500 x 16 bit) as a RAM buffer
size.
                                           More the size is higher, the recorded
quality is better */
  #define TIME_REC                3000 /* Recording time in millisecond(Systick Time
Base*TIME_REC= 10ms*3000)
                                       (default: 30s) */
#endif /* MEDIA_USB_KEY */

/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void TimingDelay_Decrement(void);
void Delay(__IO uint32_t nTime);

#endif /* __MAIN_H */
```

Waverecorder.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "pdm_filter.h"
#include "waverecorder.h"

/** @addtogroup STM32F4-Discovery_Audio_Player_Recorder
* @{
*/

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* SPI Configuration defines */
#define SPI_SCK_PIN                     GPIO_Pin_10
#define SPI_SCK_GPIO_PORT               GPIOB
#define SPI_SCK_GPIO_CLK                RCC_AHB1Periph_GPIOB
#define SPI_SCK_SOURCE                  GPIO_PinSource10
#define SPI_SCK_AF                      GPIO_AF_SPI2

#define SPI_MOSI_PIN                    GPIO_Pin_3
#define SPI_MOSI_GPIO_PORT              GPIOC
#define SPI_MOSI_GPIO_CLK               RCC_AHB1Periph_GPIOC
#define SPI_MOSI_SOURCE                 GPIO_PinSource3
#define SPI_MOSI_AF                     GPIO_AF_SPI2

#define AUDIO_REC_SPI_IRQHANDLER        SPI2_IRQHandler

/* Audio recording frequency in Hz */
#define REC_FREQ                        8000

/* PDM buffer input size */
#define INTERNAL_BUFF_SIZE      64

/* PCM buffer output size */
#define PCM_OUT_SIZE            16

#define HIGH_PASS

#if defined(LOW_PASS)

      #define NCoef             2
      #define DCgain            512

#elif defined(BAND_PASS)

      #define NCoef 4
      #define DCgain 16

#elif defined(HIGH_PASS)

      #define NCoef 2
      #define DCgain 1

#endif
```

```c
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
extern __IO uint16_t Time_Rec_Base;
extern __IO uint8_t Command_index;
extern USB_OTG_CORE_HANDLE  USB_OTG_Core;
extern __IO uint32_t WaveCounter;
extern FIL file;
extern __IO uint8_t LED_Toggle;
uint16_t RAM_Buf[RAM_BUFFER_SIZE];
uint16_t RAM_Buf1 [RAM_BUFFER_SIZE];
uint16_t buf_idx = 0, buf_idx1 =0;
uint16_t *writebuffer;
uint16_t counter = 0;
uint8_t WaveRecStatus = 0;
/* Current state of the audio recorder interface intialization */
static uint32_t AudioRecInited = 0;
PDMFilter_InitStruct Filter;

/* Audio recording Samples format (from 8 to 16 bits) */
uint32_t AudioRecBitRes = 16;
uint16_t RecBuf[PCM_OUT_SIZE], RecBuf1[PCM_OUT_SIZE];
uint8_t RecBufHeader[512], Switch = 0;
__IO uint32_t Data_Status =0;
/* Audio recording number of channels (1 for Mono or 2 for Stereo) */
uint32_t AudioRecChnlNbr = 1;
/* Main buffer pointer for the recorded data storing */
uint16_t* pAudioRecBuf;
/* Current size of the recorded buffer */
uint32_t AudioRecCurrSize = 0;
uint16_t bytesWritten;
/* Temporary data sample */
static uint16_t InternalBuffer[INTERNAL_BUFF_SIZE];

static uint32_t InternalBufferSize = 0;

/* Private function prototypes -----------------------------------------------*/
static void WaveRecorder_GPIO_Init(void);
static void WaveRecorder_SPI_Init(uint32_t Freq);
static void WaveRecorder_NVIC_Init(void);
int16_t iir(int16_t NewSample);

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

/**
  * @brief  Initialize wave recording
  * @param  AudioFreq: Sampling frequency
  *         BitRes: Audio recording Samples format (from 8 to 16 bits)
  *         ChnlNbr: Number of input microphone channel
  * @retval None
  */
uint32_t WaveRecorderInit(uint32_t AudioFreq, uint32_t BitRes, uint32_t ChnlNbr)
{
  /* Check if the interface is already initialized */
  if (AudioRecInited)
  {
    /* No need for initialization */
    return 0;
```

```
  }
  else
  {
    /* Enable CRC module */
    RCC->AHB1ENR |= RCC_AHB1ENR_CRCEN;

    /* Filter LP & HP Init */
    Filter.LP_HZ = 10000;
    Filter.HP_HZ = 10;
    Filter.Fs = 16000;
    Filter.Out_MicChannels = 1;
    Filter.In_MicChannels = 1;

    PDM_Filter_Init((PDMFilter_InitStruct *)&Filter);

    /* Configure the GPIOs */
    WaveRecorder_GPIO_Init();

    /* Configure the interrupts (for timer) */
    WaveRecorder_NVIC_Init();

    /* Configure the SPI */
    WaveRecorder_SPI_Init(AudioFreq);

    /* Set the local parameters */
    AudioRecBitRes = BitRes;
    AudioRecChnlNbr = ChnlNbr;

    /* Set state of the audio recorder to initialized */
    AudioRecInited = 1;

    /* Return 0 if all operations are OK */
    return 0;
  }
}

/**
  * @brief  Start audio recording
  * @param  pbuf: pointer to a buffer
  *         size: Buffer size
  * @retval None
  */
uint8_t WaveRecorderStart(uint16_t* pbuf, uint32_t size)
{
/* Check if the interface has already been initialized */
  if (AudioRecInited)
  {
    /* Store the location and size of the audio buffer */
    pAudioRecBuf = pbuf;
    AudioRecCurrSize = size;

    /* Enable the Rx buffer not empty interrupt */
    SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_RXNE, ENABLE);
    /* The Data transfer is performed in the SPI interrupt routine */
    /* Enable the SPI peripheral */
    I2S_Cmd(SPI2, ENABLE);
```

```
    /* Return 0 if all operations are OK */
    return 0;
  }
  else
  {
    /* Cannot perform operation */
    return 1;
  }
}

/**
  * @brief  Stop audio recording
  * @param  None
  * @retval None
  */
uint32_t WaveRecorderStop(void)
{
  /* Check if the interface has already been initialized */
  if (AudioRecInited)
  {

    /* Stop conversion */
    I2S_Cmd(SPI2, DISABLE);

    /* Return 0 if all operations are OK */
    return 0;
  }
  else
  {
    /* Cannot perform operation */
    return 1;
  }
}

/**
  * @brief  This function handles AUDIO_REC_SPI global interrupt request.
  * @param  None
  * @retval None
*/

void AUDIO_REC_SPI_IRQHANDLER(void)
{
   u16 volume;
   u16 app;



  /* Check if data are available in SPI Data register */
  if (SPI_GetITStatus(SPI2, SPI_I2S_IT_RXNE) != RESET)
  {
    app = SPI_I2S_ReceiveData(SPI2);
    InternalBuffer[InternalBufferSize++] = HTONS(app);


    /* Check to prevent overflow condition */
    if (InternalBufferSize >= INTERNAL_BUFF_SIZE)
    {
```

```
      volume = 10;

      PDM_Filter_64_LSB((uint8_t *)InternalBuffer, (uint16_t *)pAudioRecBuf,
volume , (PDMFilter_InitStruct *)&Filter);

      InternalBufferSize = 0;


      Data_Status = 1;
    }
  }
}

/**
  * @brief  Initialize the wave header file
  * @param  pHeadBuf:Pointer to a buffer
  * @retval None
  */
uint32_t WavaRecorderHeaderInit(uint8_t* pHeadBuf)
{
  uint16_t count = 0;

  /* write chunkID, must be 'RIFF'  ------------------------------------------*/
  pHeadBuf[0] = 'R';
  pHeadBuf[1] = 'I';
  pHeadBuf[2] = 'F';
  pHeadBuf[3] = 'F';

  /* Write the file length */
  /* The sampling time 8000 Hz
   To recorde 10s we need 8000 x 10 x 2 (16-Bit data) */
  pHeadBuf[4] = 0x00;
  pHeadBuf[5] = 0xE2;
  pHeadBuf[6] = 0x04;
  pHeadBuf[7] = 0x00;


  /* Write the file format, must be 'WAVE' */
  pHeadBuf[8]  = 'W';
  pHeadBuf[9]  = 'A';
  pHeadBuf[10] = 'V';
  pHeadBuf[11] = 'E';

  /* Write the format chunk, must be'fmt ' */
  pHeadBuf[12]  = 'f';
  pHeadBuf[13]  = 'm';
  pHeadBuf[14]  = 't';
  pHeadBuf[15]  = ' ';

  /* Write the length of the 'fmt' data, must be 0x10 */
  pHeadBuf[16]  = 0x10;
  pHeadBuf[17]  = 0x00;
  pHeadBuf[18]  = 0x00;
  pHeadBuf[19]  = 0x00;

  /* Write the audio format, must be 0x01 (PCM) */
```

```
  pHeadBuf[20]  = 0x01;
  pHeadBuf[21]  = 0x00;

  /* Write the number of channels, must be 0x01 (Mono) or 0x02 (Stereo) */
  pHeadBuf[22]  = 0x02;
  pHeadBuf[23]  = 0x00;

  /* Write the Sample Rate 8000 Hz */
  pHeadBuf[24]  = (uint8_t)((REC_FREQ & 0xFF));
  pHeadBuf[25]  = (uint8_t)((REC_FREQ >> 8) & 0xFF);
  pHeadBuf[26]  = (uint8_t)((REC_FREQ >> 16) & 0xFF);
  pHeadBuf[27]  = (uint8_t)((REC_FREQ >> 24) & 0xFF);

  /* Write the Byte Rate */
  pHeadBuf[28]  = (uint8_t)((REC_FREQ & 0xFF));
  pHeadBuf[29]  = (uint8_t)((REC_FREQ >> 8) & 0xFF);
  pHeadBuf[30]  = (uint8_t)((REC_FREQ >> 16) & 0xFF);
  pHeadBuf[31]  = (uint8_t)((REC_FREQ >> 24) & 0xFF);

  /* Write the block alignment */
  pHeadBuf[32]  = 0x02;/*0x02*/
  pHeadBuf[33]  = 0x00;

  /* Write the number of bits per sample */
  pHeadBuf[34]  = 0x10; /*0x08*/
  pHeadBuf[35]  = 0x00;

  /* Write the Data chunk, must be 'data' */
  pHeadBuf[36]  = 'd';
  pHeadBuf[37]  = 'a';
  pHeadBuf[38]  = 't';
  pHeadBuf[39]  = 'a';

  /* Write the number of sample data */
  pHeadBuf[40] = 0x00;
  pHeadBuf[41] = 0xE2;
  pHeadBuf[42] = 0x04;
  pHeadBuf[43] = 0x00;

  /* Fill the missing bytes in Buffer with 0x80 */
  for (count = 44; count < 512 ; count ++)
  {
    pHeadBuf[count] = 0x80;
  }

  /* Return 0 if all operations are OK */
  return 0;
}

/**
  * @brief  Update the recorded data
  * @param  None
  * @retval None
  */
void WaveRecorderUpdate(void)
{
  uint8_t bIndex;
```

```
  WaveRecorderInit(32000,16, 1);
  WaveCounter = 0;
  LED_Toggle = 7;


  /* Remove Wave file if exist on flash disk */
  f_unlink (REC_WAVE_NAME);

  /* Open the file to write on it */
  if ((HCD_IsDeviceConnected(&USB_OTG_Core) != 1) || (f_open(&file, REC_WAVE_NAME,
FA_CREATE_ALWAYS | FA_WRITE) != FR_OK))
  {
    /* Set ON Red LED */
    while(1)
    {
      STM_EVAL_LEDToggle(LED5);
    }
  }
  else
  {
    WaveRecStatus = 1;
  }
  /* Initialize the Header Wave */
  WavaRecorderHeaderInit(RecBufHeader);

  /* Write the Header wave */
  f_write (&file, RecBufHeader, 512, (void *)&bytesWritten);

  /* Increment tne wave counter */
  WaveCounter += 512;

  /* Start the record */
  WaveRecorderStart(RecBuf, PCM_OUT_SIZE);

  /* Reset the time base variable */
  Time_Rec_Base = 0;
  Switch = 0;

  while(HCD_IsDeviceConnected(&USB_OTG_Core))
  {
    /* Wait for the recording time */
    if (Time_Rec_Base <= TIME_REC)
    {
      /* Wait for the data to be ready with PCM form */
      while((Data_Status == 0)&& HCD_IsDeviceConnected(&USB_OTG_Core));
      Data_Status =0;

      /* Switch the buffers*/
      if (Switch ==1)
     {
       pAudioRecBuf = RecBuf;
       writebuffer = RecBuf1;
       WaveCounter += 32;
       Switch = 0;
     }
      else
      {
```

```
        pAudioRecBuf = RecBuf1;
        writebuffer = RecBuf;
        WaveCounter += 32;
        Switch = 1;
      }

      //TODO: add digital distortion filter here

      for(bIndex = 0 ; bIndex<16 ; bIndex++)
      {
         writebuffer[bIndex] = (uint16_t)iir( (int16_t)writebuffer[bIndex]     );
      }



      for (counter=0; counter<16; counter++)
      {
        LED_Toggle = 3;
        if (buf_idx< RAM_BUFFER_SIZE)
        {
          /* Store Data in RAM buffer */
          RAM_Buf[buf_idx++]= *(writebuffer + counter);
          if (buf_idx1 == RAM_BUFFER_SIZE)
          {
            buf_idx1 = 0;
            /* Write the stored data in the RAm to the USB Key */
            f_write (&file, (uint16_t*)RAM_Buf1, RAM_BUFFER_SIZE*2 , (void
*)&bytesWritten);
          }
        }
        else if (buf_idx1< RAM_BUFFER_SIZE)
        {
          /* Store Data in RAM buffer */
          RAM_Buf1[buf_idx1++]= *(writebuffer + counter);
          if (buf_idx == RAM_BUFFER_SIZE)
          {
            buf_idx = 0;
            /* Write the stored data in the RAM to the USB Key */
            f_write (&file, (uint16_t*)RAM_Buf, RAM_BUFFER_SIZE*2 , (void
*)&bytesWritten);
          }
        }
      }

      /* User button pressed */
      if ( Command_index != 1)
      {
        /* Stop recording */
        WaveRecorderStop();
        Command_index = 0;
        LED_Toggle = 6;
        break;
      }
    }
    else /* End of Recording time  */
    {
      WaveRecorderStop();
```

```
      LED_Toggle = 4;
      Command_index = 2;
      Data_Status =0;
      break;
    }
  }

  /* Update the data length in the header of the recorded wave */
  f_lseek(&file, 0);

  RecBufHeader[4] = (uint8_t)(WaveCounter + 512) ;
  RecBufHeader[5] = (uint8_t)(((WaveCounter+512)  >> 8) & 0xFF);
  RecBufHeader[6] = (uint8_t)(((WaveCounter+512)  >> 16) & 0xFF);
  RecBufHeader[7] = (uint8_t)(((WaveCounter+512)  >> 24) & 0xFF);

  RecBufHeader[40] = (uint8_t)(WaveCounter);
  RecBufHeader[41] = (uint8_t)((WaveCounter >> 8) & 0xFF);
  RecBufHeader[42] = (uint8_t)((WaveCounter >> 16) & 0xFF);
  RecBufHeader[43] = (uint8_t)((WaveCounter >> 24) & 0xFF);

  /* Write the updated header wave */
  f_write (&file, RecBufHeader, 512, (void *)&bytesWritten);

  /* Close file and filesystem */
  f_close (&file);
  f_mount(0, NULL);

}

/**
  * @brief  Initialize GPIO for wave recorder.
  * @param  None
  * @retval None
  */
static void WaveRecorder_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* Enable GPIO clocks */
  RCC_AHB1PeriphClockCmd(SPI_SCK_GPIO_CLK | SPI_MOSI_GPIO_CLK, ENABLE);

  /* Enable GPIO clocks */
  RCC_AHB1PeriphClockCmd(SPI_SCK_GPIO_CLK | SPI_MOSI_GPIO_CLK, ENABLE);

  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

  /* SPI SCK pin configuration */
  GPIO_InitStructure.GPIO_Pin = SPI_SCK_PIN;
  GPIO_Init(SPI_SCK_GPIO_PORT, &GPIO_InitStructure);

  /* Connect SPI pins to AF5 */
  GPIO_PinAFConfig(SPI_SCK_GPIO_PORT, SPI_SCK_SOURCE, SPI_SCK_AF);

  /* SPI MOSI pin configuration */
```

```
  GPIO_InitStructure.GPIO_Pin =  SPI_MOSI_PIN;
  GPIO_Init(SPI_MOSI_GPIO_PORT, &GPIO_InitStructure);
  GPIO_PinAFConfig(SPI_MOSI_GPIO_PORT, SPI_MOSI_SOURCE, SPI_MOSI_AF);
}

/**
  * @brief  Initialize SPI peripheral.
  * @param  Freq :Audio frequency
  * @retval None
  */
static void WaveRecorder_SPI_Init(uint32_t Freq)
{
  I2S_InitTypeDef I2S_InitStructure;

  /* Enable the SPI clock */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2,ENABLE);

  /* SPI configuration */
  SPI_I2S_DeInit(SPI2);
  I2S_InitStructure.I2S_AudioFreq = 32000;
  I2S_InitStructure.I2S_Standard = I2S_Standard_LSB;
  I2S_InitStructure.I2S_DataFormat = I2S_DataFormat_16b;
  I2S_InitStructure.I2S_CPOL = I2S_CPOL_High;
  I2S_InitStructure.I2S_Mode = I2S_Mode_MasterRx;
  I2S_InitStructure.I2S_MCLKOutput = I2S_MCLKOutput_Disable;
  /* Initialize the I2S peripheral with the structure above */
  I2S_Init(SPI2, &I2S_InitStructure);

  /* Enable the Rx buffer not empty interrupt */
  SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_RXNE, ENABLE);
}


/**
  * @brief  Initialize the NVIC.
  * @param  None
  * @retval None
  */
static void WaveRecorder_NVIC_Init(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;

  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);
  /* Configure the SPI interrupt priority */
  NVIC_InitStructure.NVIC_IRQChannel = SPI2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}

int16_t iir(int16_t NewSample)
{
#if defined(LOW_PASS)
      int16_t ACoef[NCoef+1] =
      {
              13664,
              27328,
```

```
		13664
	};


	int16_t BCoef[NCoef+1] =
	{
		16384,
		-31308,
		15031
	};
#elif defined(BAND_PASS)
	int16_t ACoef[NCoef+1] =
	{
		11492,
		    0,
		-22985,
		    0,
		11492
	};


	int16_t BCoef[NCoef+1] =
	{
		 8192,
		-17899,
		20200,
		-12481,
		 4224
	};

#elif defined(HIGH_PASS)
	int16_t ACoef[NCoef+1] =
	{
		10239,
		-20478,
		10239
	};


	int16_t BCoef[NCoef+1] =
	{
		32768,
		 2099,
		10288
	};


#endif

	static int32_t y[NCoef+1]; //output samples
	//Warning!!!!!! This variable should be signed (input sample width + Coefs
width + 2 )-bit width to avoid saturation.

	static int16_t x[NCoef+1]; //input samples
	int n;

	//shift the old samples
	for(n=NCoef; n>0; n--)
```

```
	{
	   x[n] = x[n-1];
	   y[n] = y[n-1];
	}

	//Calculate the new output
	x[0] = NewSample;
	y[0] = ACoef[0] * x[0];

	for(n=1; n<=NCoef; n++)
	{
		y[0] += ((ACoef[n] * x[n]) - (BCoef[n] * y[n]));
	}

	y[0] /= BCoef[0];

	return (y[0] / DCgain);
}


#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

/**
* @}
*/
```

Waverecorder.h

```
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __I2S_AUDIO_H
#define __I2S_AUDIO_H

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx.h"
#include "main.h"


/* Exported types ------------------------------------------------------------*/
/* Exported Defines ----------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void AUDIO_REC_SPI_IRQHANDLER(void);
uint32_t WaveRecorderInit(uint32_t AudioFreq, uint32_t BitRes, uint32_t ChnlNbr);
uint8_t WaveRecorderStart(uint16_t* pbuf, uint32_t size);
uint32_t WaveRecorderStop(void);
uint32_t WavaRecorderHeaderInit(uint8_t* pHeadBuf);
void Delay(__IO uint32_t nTime);
void WaveRecorderUpdate(void);
extern uint32_t ReadUnit(uint8_t *buffer, uint8_t idx, uint8_t NbrOfBytes,
Endianness BytesFormat);

#endif /* __WAVE_RECORDER_H */
```

Waveplayer.h

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/** @addtogroup STM32F4-Discovery_Audio_Player_Recorder
* @{
*/

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#if defined MEDIA_IntFLASH
 /* This is an audio file stored in the Flash memory as a constant table of 16-bit
data.
    The audio format should be WAV (raw / PCM) 16-bits, Stereo (sampling rate may be
modified) */
extern uint16_t AUDIO_SAMPLE[];
/* Audio file size and start address are defined here since the audio file is
    stored in Flash memory as a constant table of 16-bit data */
#define AUDIO_FILE_SZE          990000
#define AUIDO_START_ADDRESS     58 /* Offset relative to audio file header size */
#endif

/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
#if defined MEDIA_USB_KEY
 extern __IO uint8_t Command_index;
 static uint32_t wavelen = 0;
 static char* WaveFileName ;
 static __IO uint32_t SpeechDataOffset = 0x00;
 __IO ErrorCode WaveFileStatus = Unvalid_RIFF_ID;
 UINT BytesRead;
 WAVE_FormatTypeDef WAVE_Format;
 uint16_t buffer1[_MAX_SS] ={0x00};
 uint16_t buffer2[_MAX_SS] ={0x00};
 uint8_t buffer_switch = 1;
 extern FATFS fatfs;
 extern FIL file;
 extern FIL fileR;
 extern DIR dir;
 extern FILINFO fno;
 extern uint16_t *CurrentPos;
 extern USB_OTG_CORE_HANDLE USB_OTG_Core;
 extern uint8_t WaveRecStatus;
#endif

__IO uint32_t XferCplt = 0;
__IO uint8_t volume = 100, AudioPlayStart = 0;
__IO uint32_t WaveCounter;
uint8_t Buffer[6];
__IO uint32_t WaveDataLength = 0;
extern __IO uint8_t Count;
extern __IO uint8_t RepeatState ;
extern __IO uint8_t LED_Toggle;
extern __IO uint8_t PauseResumeStatus ;
extern uint32_t AudioRemSize;
static __IO uint32_t TimingDelay;
```

```
/* Private function prototypes -----------------------------------------------*/
static void Mems_Config(void);
static void EXTILine_Config(void);
#if defined MEDIA_USB_KEY
 static ErrorCode WavePlayer_WaveParsing(uint32_t *FileLen);
#endif

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

/**
  * @brief  Play wave from a mass storge
  * @param  AudioFreq: Audio Sampling Frequency
  * @retval None
*/

void WavePlayBack(uint32_t AudioFreq)
{
  /*
  Normal mode description:
  Start playing the audio file (using DMA stream) .
  Using this mode, the application can run other tasks in parallel since
  the DMA is handling the Audio Transfer instead of the CPU.
  The only task remaining for the CPU will be the management of the DMA
  Transfer Complete interrupt or the Half Transfer Complete interrupt in
  order to load again the buffer and to calculate the remaining data.
  Circular mode description:
  Start playing the file from a circular buffer, once the DMA is enabled it
  always run. User has to fill periodically the buffer with the audio data
  using Transfer complete and/or half transfer complete interrupts callbacks
  (EVAL_AUDIO_TransferComplete_CallBack() or EVAL_AUDIO_HalfTransfer_CallBack()...
  In this case the audio data file is smaller than the DMA max buffer
  size 65535 so there is no need to load buffer continuously or manage the
  transfer complete or Half transfer interrupts callbacks. */

  /* Start playing */
  AudioPlayStart = 1;
  RepeatState =0;
#if defined MEDIA_IntFLASH

  /* Initialize wave player (Codec, DMA, I2C) */
  WavePlayerInit(AudioFreq);

  /* Play on */
  AudioFlashPlay((uint16_t*)(AUDIO_SAMPLE +
AUIDO_START_ADDRESS),AUDIO_FILE_SZE,AUIDO_START_ADDRESS);

  /* LED Blue Start toggling */
  LED_Toggle = 6;

  /* Infinite loop */
  while(1)
  {
    /* check on the repeate status */
    if (RepeatState == 0)
    {
      if (PauseResumeStatus == 0)
```

```
      {
        /* LED Blue Stop Toggling */
        LED_Toggle = 0;
        /* Pause playing */
        WavePlayerPauseResume(PauseResumeStatus);
        PauseResumeStatus = 2;
      }
      else if (PauseResumeStatus == 1)
      {
        /* LED Blue Toggling */
        LED_Toggle = 6;
        /* Resume playing */
        WavePlayerPauseResume(PauseResumeStatus);
        PauseResumeStatus = 2;
      }
    }
    else
    {
      /* Stop playing */
      WavePlayerStop();
      /* Green LED toggling */
      LED_Toggle = 4;
    }
  }

#elif defined MEDIA_USB_KEY
  /* Initialize wave player (Codec, DMA, I2C) */
  WavePlayerInit(AudioFreq);
  AudioRemSize   = 0;

  /* Get Data from USB Key */
  f_lseek(&fileR, WaveCounter);
  f_read (&fileR, buffer1, _MAX_SS, &BytesRead);
  f_read (&fileR, buffer2, _MAX_SS, &BytesRead);

  /* Start playing wave */
  Audio_MAL_Play((uint32_t)buffer1, _MAX_SS);
  buffer_switch = 1;
  XferCplt = 0;
  LED_Toggle = 6;
  PauseResumeStatus = 1;
  Count = 0;

  while((WaveDataLength != 0) &&  HCD_IsDeviceConnected(&USB_OTG_Core))
  {
    /* Test on the command: Playing */
    if (Command_index == 0)
    {
      /* wait for DMA transfert complete */
      while((XferCplt == 0) &&  HCD_IsDeviceConnected(&USB_OTG_Core))
      {
        if (PauseResumeStatus == 0)
        {
          /* Pause Playing wave */
          LED_Toggle = 0;
          WavePlayerPauseResume(PauseResumeStatus);
          PauseResumeStatus = 2;
        }
```

```
        else if (PauseResumeStatus == 1)
        {
          LED_Toggle = 6;
          /* Resume Playing wave */
          WavePlayerPauseResume(PauseResumeStatus);
          PauseResumeStatus = 2;
        }
      }
      XferCplt = 0;

      if(buffer_switch == 0)
      {
        /* Play data from buffer1 */
        Audio_MAL_Play((uint32_t)buffer1, _MAX_SS);
        /* Store data in buffer2 */
        f_read (&fileR, buffer2, _MAX_SS, &BytesRead);
        buffer_switch = 1;
      }
      else
      {
        /* Play data from buffer2 */
        Audio_MAL_Play((uint32_t)buffer2, _MAX_SS);
        /* Store data in buffer1 */
        f_read (&fileR, buffer1, _MAX_SS, &BytesRead);
        buffer_switch = 0;
      }
    }
    else
    {
      WavePlayerStop();
      WaveDataLength = 0;
      RepeatState =0;
      break;
    }
  }
#if defined PLAY_REPEAT_OFF
  RepeatState = 1;
  WavePlayerStop();
  if (Command_index == 0)
    LED_Toggle = 4;
#else
  LED_Toggle = 7;
  RepeatState = 0;
  AudioPlayStart = 0;
  WavePlayerStop();
#endif
#endif

}

/**
  * @brief  Pause or Resume a played wave
  * @param  state: if it is equal to 0 pause Playing else resume playing
  * @retval None
  */
void WavePlayerPauseResume(uint8_t state)
{
  EVAL_AUDIO_PauseResume(state);
```

```
}

/**
  * @brief  Configure the volune
  * @param  vol: volume value
  * @retval None
  */
uint8_t WaveplayerCtrlVolume(uint8_t vol)
{
  EVAL_AUDIO_VolumeCtl(vol);
  return 0;
}


/**
  * @brief  Stop playing wave
  * @param  None
  * @retval None
  */
void WavePlayerStop(void)
{
  EVAL_AUDIO_Stop(CODEC_PDWN_SW);
}

/**
* @brief  Initializes the wave player
* @param  AudioFreq: Audio sampling frequency
* @retval None
*/
int WavePlayerInit(uint32_t AudioFreq)
{

  /* MEMS Accelerometre configure to manage PAUSE, RESUME and Controle Volume
operation */
  Mems_Config();

  /* EXTI configue to detect interrupts on Z axis click and on Y axis high event */
  EXTILine_Config();

  /* Initialize I2S interface */
  EVAL_AUDIO_SetAudioInterface(AUDIO_INTERFACE_I2S);

  /* Initialize the Audio codec and all related peripherals (I2S, I2C, IOExpander,
IOs...) */
  EVAL_AUDIO_Init(OUTPUT_DEVICE_AUTO, volume, AudioFreq );

  return 0;
}

/**
  * @brief  MEMS accelerometre management of the timeout situation.
  * @param  None.
  * @retval None.
  */
uint32_t LIS302DL_TIMEOUT_UserCallback(void)
{
  /* MEMS Accelerometer Timeout error occured */
```

```
  while (1)
  {
  }
}

/**
* @brief  Play wave file from internal Flash
* @param  None
* @retval None
*/
uint32_t AudioFlashPlay(uint16_t* pBuffer, uint32_t FullSize, uint32_t StartAdd)
{
  EVAL_AUDIO_Play((uint16_t*)pBuffer, (FullSize - StartAdd));
  return 0;
}

/*----------------------------------
Callbacks implementation:
the callbacks prototypes are defined in the stm324xg_eval_audio_codec.h file
and their implementation should be done in the user code if they are needed.
Below some examples of callback implementations.
----------------------------------------------------------*/
/**
* @brief  Calculates the remaining file size and new position of the pointer.
* @param  None
* @retval None
*/
void EVAL_AUDIO_TransferComplete_CallBack(uint32_t pBuffer, uint32_t Size)
{
  /* Calculate the remaining audio data in the file and the new size
  for the DMA transfer. If the Audio files size is less than the DMA max
  data transfer size, so there is no calculation to be done, just restart
  from the beginning of the file ... */
  /* Check if the end of file has been reached */

#ifdef AUDIO_MAL_MODE_NORMAL

#if defined MEDIA_IntFLASH

#if defined PLAY_REPEAT_OFF
  LED_Toggle = 4;
  RepeatState = 1;
  EVAL_AUDIO_Stop(CODEC_PDWN_HW);
#else
  /* Replay from the beginning */
  AudioFlashPlay((uint16_t*)(AUDIO_SAMPLE +
AUIDO_START_ADDRESS),AUDIO_FILE_SZE,AUIDO_START_ADDRESS);
#endif

#elif defined MEDIA_USB_KEY
  XferCplt = 1;
  if (WaveDataLength) WaveDataLength -= _MAX_SS;
  if (WaveDataLength < _MAX_SS) WaveDataLength = 0;

#endif

#else /* #ifdef AUDIO_MAL_MODE_CIRCULAR */
```

```
#endif /* AUDIO_MAL_MODE_CIRCULAR */
}

/**
* @brief  Manages the DMA Half Transfer complete interrupt.
* @param  None
* @retval None
*/
void EVAL_AUDIO_HalfTransfer_CallBack(uint32_t pBuffer, uint32_t Size)
{
#ifdef AUDIO_MAL_MODE_CIRCULAR

#endif /* AUDIO_MAL_MODE_CIRCULAR */

  /* Generally this interrupt routine is used to load the buffer when
  a streaming scheme is used: When first Half buffer is already transferred load
  the new data to the first half of buffer while DMA is transferring data from
  the second half. And when Transfer complete occurs, load the second half of
  the buffer while the DMA is transferring from the first half ... */
  /*
  ...........
  */
}

/**
* @brief  Manages the DMA FIFO error interrupt.
* @param  None
* @retval None
*/
void EVAL_AUDIO_Error_CallBack(void* pData)
{
  /* Stop the program with an infinite loop */
  while (1)
  {}

  /* could also generate a system reset to recover from the error */
  /* .... */
}

/**
* @brief  Get next data sample callback
* @param  None
* @retval Next data sample to be sent
*/
uint16_t EVAL_AUDIO_GetSampleCallBack(void)
{
  return 0;
}


#ifndef USE_DEFAULT_TIMEOUT_CALLBACK
/**
  * @brief  Basic management of the timeout situation.
  * @param  None.
  * @retval None.
```

```
  */
uint32_t Codec_TIMEOUT_UserCallback(void)
{
  return (0);
}
#endif /* USE_DEFAULT_TIMEOUT_CALLBACK */
/*-----------------------------------------------------------------------------*/

/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 10 ms.
  * @retval None
  */
void Delay(__IO uint32_t nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);
}

/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }
}

#if defined MEDIA_USB_KEY

/**
  * @brief  Start wave player
  * @param  None
  * @retval None
  */
void WavePlayerStart(void)
{
  char path[] = "0:/";

  buffer_switch = 1;

  /* Get the read out protection status */
  if (f_opendir(&dir, path)!= FR_OK)
  {
    while(1)
    {
      STM_EVAL_LEDToggle(LED5);
      Delay(10);
    }
  }
  else
  {
    if (WaveRecStatus == 1)
```

```
    {
      WaveFileName = REC_WAVE_NAME;
    }
    else
    {
      WaveFileName = WAVE_NAME;
    }
    /* Open the wave file to be played */
    if (f_open(&fileR, WaveFileName , FA_READ) != FR_OK)
    {
      STM_EVAL_LEDOn(LED5);
      Command_index = 1;
    }
    else
    {
      /* Read data(_MAX_SS byte) from the selected file */
      f_read (&fileR, buffer1, _MAX_SS, &BytesRead);

      WaveFileStatus = WavePlayer_WaveParsing(&wavelen);

      if (WaveFileStatus == Valid_WAVE_File)  /* the .WAV file is valid */
      {
        /* Set WaveDataLenght to the Speech wave length */
        WaveDataLength = WAVE_Format.DataSize;
      }
      else /* Unvalid wave file */
      {
        /* Led Red Toggles in infinite loop */
        while(1)
        {
          STM_EVAL_LEDToggle(LED5);
          Delay(10);
        }
      }
      /* Play the wave */
      WavePlayBack(WAVE_Format.SampleRate);
    }
  }
}

/**
  * @brief  Reset the wave player
  * @param  None
  * @retval None
  */
void WavePlayer_CallBack(void)
{
  /* Reset the wave player variables */
  RepeatState = 0;
  AudioPlayStart =0;
  LED_Toggle = 7;
  PauseResumeStatus = 1;
  WaveDataLength =0;
  Count = 0;

  /* Stops the codec */
  EVAL_AUDIO_Stop(CODEC_PDWN_HW);
  /* LED off */
```

```
  STM_EVAL_LEDOff(LED3);
  STM_EVAL_LEDOff(LED4);
  STM_EVAL_LEDOff(LED6);

  /* TIM Interrupts disable */
  TIM_ITConfig(TIM4, TIM_IT_CC1, DISABLE);
  f_mount(0, NULL);
}

/**
  * @brief  Checks the format of the .WAV file and gets information about
  *   the audio format. This is done by reading the value of a
  *   number of parameters stored in the file header and comparing
  *   these to the values expected authenticates the format of a
  *   standard .WAV  file (44 bytes will be read). If  it is a valid
  *   .WAV file format, it continues reading the header to determine
  *   the  audio format such as the sample rate and the sampled data
  *   size. If the audio format is supported by this application,
  *   it retrieves the audio format in WAVE_Format structure and
  *   returns a zero value. Otherwise the function fails and the
  *   return value is nonzero.In this case, the return value specifies
  *   the cause of  the function fails. The error codes that can be
  *   returned by this function are declared in the header file.
  * @param  None
  * @retval Zero value if the function succeed, otherwise it return
  *         a nonzero value which specifies the error code.
  */
static ErrorCode WavePlayer_WaveParsing(uint32_t *FileLen)
{
  uint32_t temp = 0x00;
  uint32_t extraformatbytes = 0;

  /* Read chunkID, must be 'RIFF' */
  temp = ReadUnit((uint8_t*)buffer1, 0, 4, BigEndian);
  if (temp != CHUNK_ID)
  {
    return(Unvalid_RIFF_ID);
  }

  /* Read the file length */
  WAVE_Format.RIFFchunksize = ReadUnit((uint8_t*)buffer1, 4, 4, LittleEndian);

  /* Read the file format, must be 'WAVE' */
  temp = ReadUnit((uint8_t*)buffer1, 8, 4, BigEndian);
  if (temp != FILE_FORMAT)
  {
    return(Unvalid_WAVE_Format);
  }

  /* Read the format chunk, must be'fmt ' */
  temp = ReadUnit((uint8_t*)buffer1, 12, 4, BigEndian);
  if (temp != FORMAT_ID)
  {
    return(Unvalid_FormatChunk_ID);
  }
  /* Read the length of the 'fmt' data, must be 0x10 -------------------------*/
  temp = ReadUnit((uint8_t*)buffer1, 16, 4, LittleEndian);
```

```
if (temp != 0x10)
{
  extraformatbytes = 1;
}
/* Read the audio format, must be 0x01 (PCM) */
WAVE_Format.FormatTag = ReadUnit((uint8_t*)buffer1, 20, 2, LittleEndian);
if (WAVE_Format.FormatTag != WAVE_FORMAT_PCM)
{
  return(Unsupporetd_FormatTag);
}

/* Read the number of channels, must be 0x01 (Mono) or 0x02 (Stereo) */
WAVE_Format.NumChannels = ReadUnit((uint8_t*)buffer1, 22, 2, LittleEndian);

/* Read the Sample Rate */
WAVE_Format.SampleRate = ReadUnit((uint8_t*)buffer1, 24, 4, LittleEndian);

/* Read the Byte Rate */
WAVE_Format.ByteRate = ReadUnit((uint8_t*)buffer1, 28, 4, LittleEndian);

/* Read the block alignment */
WAVE_Format.BlockAlign = ReadUnit((uint8_t*)buffer1, 32, 2, LittleEndian);

/* Read the number of bits per sample */
WAVE_Format.BitsPerSample = ReadUnit((uint8_t*)buffer1, 34, 2, LittleEndian);
if (WAVE_Format.BitsPerSample != BITS_PER_SAMPLE_16)
{
  return(Unsupporetd_Bits_Per_Sample);
}
SpeechDataOffset = 36;
/* If there is Extra format bytes, these bytes will be defined in "Fact Chunk" */
if (extraformatbytes == 1)
{
  /* Read th Extra format bytes, must be 0x00 */
  temp = ReadUnit((uint8_t*)buffer1, 36, 2, LittleEndian);
  if (temp != 0x00)
  {
    return(Unsupporetd_ExtraFormatBytes);
  }
  /* Read the Fact chunk, must be 'fact' */
  temp = ReadUnit((uint8_t*)buffer1, 38, 4, BigEndian);
  if (temp != FACT_ID)
  {
    return(Unvalid_FactChunk_ID);
  }
  /* Read Fact chunk data Size */
  temp = ReadUnit((uint8_t*)buffer1, 42, 4, LittleEndian);

  SpeechDataOffset += 10 + temp;
}
/* Read the Data chunk, must be 'data' */
temp = ReadUnit((uint8_t*)buffer1, SpeechDataOffset, 4, BigEndian);
SpeechDataOffset += 4;
if (temp != DATA_ID)
{
  return(Unvalid_DataChunk_ID);
}
```

```
  /* Read the number of sample data */
  WAVE_Format.DataSize = ReadUnit((uint8_t*)buffer1, SpeechDataOffset, 4,
LittleEndian);
  SpeechDataOffset += 4;
  WaveCounter =  SpeechDataOffset;
  return(Valid_WAVE_File);
}

/**
* @brief  Reads a number of bytes from the SPI Flash and reorder them in Big
*         or little endian.
* @param  NbrOfBytes: number of bytes to read.
*         This parameter must be a number between 1 and 4.
* @param  ReadAddr: external memory address to read from.
* @param  Endians: specifies the bytes endianness.
*         This parameter can be one of the following values:
*             - LittleEndian
*             - BigEndian
* @retval Bytes read from the SPI Flash.
*/
uint32_t ReadUnit(uint8_t *buffer, uint8_t idx, uint8_t NbrOfBytes, Endianness
BytesFormat)
{
  uint32_t index = 0;
  uint32_t temp = 0;

  for (index = 0; index < NbrOfBytes; index++)
  {
    temp |= buffer[idx + index] << (index * 8);
  }

  if (BytesFormat == BigEndian)
  {
    temp = __REV(temp);
  }
  return temp;
}
#endif

/**
  * @brief  Configures EXTI Line0 (connected to PA0 pin) in interrupt mode
  * @param  None
  * @retval None
  */
static void EXTILine_Config(void)
{
  GPIO_InitTypeDef   GPIO_InitStructure;
  NVIC_InitTypeDef   NVIC_InitStructure;
  EXTI_InitTypeDef   EXTI_InitStructure;
  /* Enable GPIOA clock */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);
  /* Enable SYSCFG clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
  /* Configure PE0 and PE1 pins as input floating */
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1;
```

```
  GPIO_Init(GPIOE, &GPIO_InitStructure);

  /* Connect EXTI Line to PE1 pins */
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE, EXTI_PinSource1);

  /* Configure EXTI Line1 */
  EXTI_InitStructure.EXTI_Line = EXTI_Line1;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);

  /* Enable and set EXTI Line0 Interrupt to the lowest priority */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI1_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}

#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

/**
* @}
*/
```

usbh_usr.c

```c
/* Includes ------------------------------------------------------------------*/
#include "usbh_usr.h"
#include "stm32f4xx_it.h"

/** @addtogroup STM32F4-Discovery_Audio_Player_Recorder
  * @{
  */

/* External variables --------------------------------------------------------*/
/* Private typedef -----------------------------------------------------------*/
/* Private defines -----------------------------------------------------------*/
/* Private macros ------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
__IO uint8_t Command_index = 0;
/*  Points to the DEVICE_PROP structure of current device */
/*  The purpose of this register is to speed up the execution */
FATFS fatfs;
FIL file;
FIL fileR;
DIR dir;
FILINFO fno;

USBH_Usr_cb_TypeDef USR_Callbacks =
{
  USBH_USR_Init,
  USBH_USR_DeInit,
  USBH_USR_DeviceAttached,
  USBH_USR_ResetDevice,
  USBH_USR_DeviceDisconnected,
  USBH_USR_OverCurrentDetected,
  USBH_USR_DeviceSpeedDetected,
  USBH_USR_Device_DescAvailable,
  USBH_USR_DeviceAddressAssigned,
  USBH_USR_Configuration_DescAvailable,
  USBH_USR_Manufacturer_String,
  USBH_USR_Product_String,
  USBH_USR_SerialNum_String,
  USBH_USR_EnumerationDone,
  USBH_USR_UserInput,
  USBH_USR_MSC_Application,
  USBH_USR_DeviceNotSupported,
  USBH_USR_UnrecoveredError
};

extern USB_OTG_CORE_HANDLE          USB_OTG_Core;
extern __IO uint8_t AudioPlayStart ;
uint8_t joystick_use = 0x00;
uint8_t lcdLineNo = 0x00;
extern __IO uint8_t RepeatState ;
extern __IO uint8_t LED_Toggle;
static uint8_t USBH_USR_ApplicationState = USH_USR_FS_INIT;
extern __IO uint32_t WaveDataLength ;
extern __IO uint16_t Time_Rec_Base;

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
```

```
/**
  * @brief  USBH_USR_Init
  * @param  None
  * @retval None
  */
void USBH_USR_Init(void)
{
}

/**
  * @brief  USBH_USR_DeviceAttached
  * @param  None
  * @retval None
  */
void USBH_USR_DeviceAttached(void)
{
  RepeatState = 0;

  LED_Toggle = 7;
  /* Red LED off when device attached */
  STM_EVAL_LEDOff(LED5);
  /* Green LED on */
  STM_EVAL_LEDOn(LED4);
  /* TIM Interrupts enable */
  TIM_ITConfig(TIM4, TIM_IT_CC1, ENABLE);
}

/**
  * @brief  USBH_USR_UnrecoveredError
  * @param  None
  * @retval None
  */
void USBH_USR_UnrecoveredError (void)
{
}

/**
  * @brief  USBH_DisconnectEvent
  *         Device disconnect event
  * @param  None
  * @retval Staus
  */
void USBH_USR_DeviceDisconnected (void)
{
  /* Red Led on if the USB Key is removed */
  STM_EVAL_LEDOn(LED5);
  /* Disable the Timer */
  TIM_ITConfig(TIM4, TIM_IT_CC1 , DISABLE);

  /* If USB key Removed when playing a wave */
  if( (WaveDataLength!=0)&& (Command_index != 1))
  {
    WavePlayer_CallBack();
    Command_index = 0;
  }
```

```
  /* If USB key Removed when recording a wave */
  if(Command_index == 1)
  {
    WaveRecorderStop();
    STM_EVAL_LEDOff(LED3);
    Command_index = 1;
    Time_Rec_Base = 0;
    LED_Toggle = 7;
  }
}

/**
  * @brief  USBH_USR_ResetUSBDevice
  * @param  None
  * @retval None
  */
void USBH_USR_ResetDevice(void)
{
  /* callback for USB-Reset */
}


/**
  * @brief  USBH_USR_DeviceSpeedDetected
  *         Displays the message on LCD for device speed
  * @param  Device speed:
  * @retval None
  */
void USBH_USR_DeviceSpeedDetected(uint8_t DeviceSpeed)
{
}

/**
  * @brief  USBH_USR_Device_DescAvailable
  * @param  device descriptor
  * @retval None
  */
void USBH_USR_Device_DescAvailable(void *DeviceDesc)
{
  /* callback for device descriptor */
}

/**
  * @brief  USBH_USR_DeviceAddressAssigned
  *         USB device is successfully assigned the Address
  * @param  None
  * @retval None
  */
void USBH_USR_DeviceAddressAssigned(void)
{
  /* callback for device successfully assigned the Address */
}

/**
  * @brief  USBH_USR_Conf_Desc
  * @param  Configuration descriptor
  * @retval None
  */
```

```c
void USBH_USR_Configuration_DescAvailable(USBH_CfgDesc_TypeDef * cfgDesc,
    USBH_InterfaceDesc_TypeDef *itfDesc,
    USBH_EpDesc_TypeDef *epDesc)
{
  /* callback for configuration descriptor */
}

/**
  * @brief  USBH_USR_Manufacturer_String
  * @param  Manufacturer String
  * @retval None
  */
void USBH_USR_Manufacturer_String(void *ManufacturerString)
{
  /* callback for  Manufacturer String */
}

/**
  * @brief  USBH_USR_Product_String
  * @param  Product String
  * @retval None
  */
void USBH_USR_Product_String(void *ProductString)
{
  /* callback for Product String */
}

/**
  * @brief  USBH_USR_SerialNum_String
  * @param  SerialNum_String
  * @retval None
  */
void USBH_USR_SerialNum_String(void *SerialNumString)
{
  /* callback for SerialNum_String */
}

/**
  * @brief  EnumerationDone
  *         User response request is displayed to ask application jump to class
  * @param  None
  * @retval None
  */
void USBH_USR_EnumerationDone(void)
{
  /* 0.5 seconds delay */
  USB_OTG_BSP_mDelay(500);

  USBH_USR_MSC_Application();
}

/**
  * @brief  USBH_USR_DeviceNotSupported
  *         Device is not supported
  * @param  None
  * @retval None
  */
void USBH_USR_DeviceNotSupported(void)
```

```
{
}


/**
  * @brief  USBH_USR_UserInput
  *         User Action for application state entry
  * @param  None
  * @retval USBH_USR_Status : User response for key button
  */
USBH_USR_Status USBH_USR_UserInput(void)
{
  /* callback for Key botton: set by software in this case */
  return USBH_USR_RESP_OK;
}

/**
  * @brief  USBH_USR_OverCurrentDetected
  *         Over Current Detected on VBUS
  * @param  None
  * @retval None
  */
void USBH_USR_OverCurrentDetected (void)
{
}

/**
  * @brief  USBH_USR_MSC_Application
  * @param  None
  * @retval Staus
  */
int USBH_USR_MSC_Application(void)
{

  switch (USBH_USR_ApplicationState)
  {
    case USH_USR_FS_INIT:

      /* Initialises the File System*/
      if (f_mount( 0, &fatfs ) != FR_OK )
      {
        /* efs initialisation fails*/
        return(-1);
      }

      /* Flash Disk is write protected */
      if (USBH_MSC_Param.MSWriteProtect == DISK_WRITE_PROTECTED)
      {
        while(1)
        {
          /* Red LED On */
          STM_EVAL_LEDOn(LED5);
        }
      }
      /* Go to menu */
      USBH_USR_ApplicationState = USH_USR_AUDIO;
      break;
```

```
    case USH_USR_AUDIO:

      /* Go to Audio menu */
      COMMAND_AudioExecuteApplication();

      /* Set user initialization flag */
      USBH_USR_ApplicationState = USH_USR_FS_INIT;
      break;

    default:
      break;
  }
  return(0);
}

/**
  * @brief  COMMAND_AudioExecuteApplication
  * @param  None
  * @retval None
  */
void COMMAND_AudioExecuteApplication(void)
{
  /* Execute the command switch the command index */
  switch (Command_index)
  {
  /* Start Playing from USB Flash memory */
  case CMD_PLAY:
    if (RepeatState == 0)
      WavePlayerStart();
    break;
    /* Start Recording in USB Flash memory */
  case CMD_RECORD:
    RepeatState = 0;
    WaveRecorderUpdate();
    break;
  default:
    break;
  }
}

/**
  * @brief  USBH_USR_DeInit
  *         Deint User state and associated variables
  * @param  None
  * @retval None
  */
void USBH_USR_DeInit(void)
{
  USBH_USR_ApplicationState = USH_USR_FS_INIT;
}

/**
  * @}
  */
```

usbh_usr.h

```
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __USH_USR_H__
#define __USH_USR_H__

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "ff.h"
#include "stm32f4_discovery.h"
#include "usbh_core.h"
#include <stdio.h>
#include "usbh_msc_core.h"

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* State Machine for the USBH_USR_ApplicationState */
#define USH_USR_FS_INIT    ((uint8_t)0x00)
#define USH_USR_AUDIO      ((uint8_t)0x01)

#define CMD_PLAY           ((uint8_t)0x00)
#define CMD_RECORD         ((uint8_t)0x01)

/* Exported macros -----------------------------------------------------------*/
/* Exported variables --------------------------------------------------------*/
extern  USBH_Usr_cb_TypeDef USR_Callbacks;

/* Exported functions ------------------------------------------------------- */
void USBH_USR_Init(void);
void USBH_USR_DeviceAttached(void);
void USBH_USR_ResetDevice(void);
void USBH_USR_DeviceDisconnected (void);
void USBH_USR_OverCurrentDetected (void);
void USBH_USR_DeviceSpeedDetected(uint8_t DeviceSpeed);
void USBH_USR_Device_DescAvailable(void *);
void USBH_USR_DeviceAddressAssigned(void);
void USBH_USR_Configuration_DescAvailable(USBH_CfgDesc_TypeDef * cfgDesc,
                                          USBH_InterfaceDesc_TypeDef *itfDesc,
                                          USBH_EpDesc_TypeDef *epDesc);
void USBH_USR_Manufacturer_String(void *);
void USBH_USR_Product_String(void *);
void USBH_USR_SerialNum_String(void *);
void USBH_USR_EnumerationDone(void);
USBH_USR_Status USBH_USR_UserInput(void);
int USBH_USR_MSC_Application(void);
void USBH_USR_DeInit(void);
void USBH_USR_DeviceNotSupported(void);
void USBH_USR_UnrecoveredError(void);
void COMMAND_AudioExecuteApplication(void);
extern void WavePlayerStart(void);
extern void WaveRecorderUpdate(void);
extern void Delay(__IO uint32_t nTime);
extern void WavePlayer_CallBack(void);
```

```
extern uint32_t WaveRecorderStop(void);
#ifdef __cplusplus
}
#endif

#endif /*__USH_USR_H__*/
```

usb_bsp.c

```c
/* Includes ------------------------------------------------------------------*/
#include "usb_bsp.h"
#include "stm32f4_discovery.h"

/** @addtogroup STM32F4-Discovery_Audio_Player_Recorder
  * @{
  */

/* External variables --------------------------------------------------------*/
/* Private typedef -----------------------------------------------------------*/
/* Private defines -----------------------------------------------------------*/

#define USE_ACCURATE_TIME
#define TIM_MSEC_DELAY                     0x01
#define TIM_USEC_DELAY                     0x02
#define HOST_OVRCURR_PORT                  GPIOD
#define HOST_OVRCURR_LINE                  GPIO_Pin_5
#define HOST_OVRCURR_PORT_SOURCE           GPIO_PortSourceGPIOD
#define HOST_OVRCURR_PIN_SOURCE            GPIO_PinSourceD
#define HOST_OVRCURR_PORT_RCC              RCC_APB2Periph_GPIOD
#define HOST_OVRCURR_EXTI_LINE             EXTI_Line5
#define HOST_OVRCURR_IRQn                  EXTI9_5_IRQn


#define HOST_POWERSW_PORT_RCC              RCC_AHB1Periph_GPIOC
#define HOST_POWERSW_PORT                  GPIOC
#define HOST_POWERSW_VBUS                  GPIO_Pin_0

/* Private macros ------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
ErrorStatus HSEStartUpStatus;
#ifdef USE_ACCURATE_TIME
  __IO uint32_t BSP_delay = 0;
#endif

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
#ifdef USE_ACCURATE_TIME
 static void BSP_SetTime(uint8_t Unit);
 static void BSP_Delay(uint32_t nTime, uint8_t Unit);
 static void USB_OTG_BSP_TimeInit ( void );
#endif

/**
  * @brief  BSP_Init
  *         board user initializations
  * @param  None
  * @retval None
  */
void BSP_Init(void)
{
  /* Configure PA0 pin: User Key pin */
  STM_EVAL_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
}
```

```
/**
  * @brief  USB_OTG_BSP_Init
  *         Initilizes BSP configurations
  * @param  None
  * @retval None
  */
void USB_OTG_BSP_Init(USB_OTG_CORE_HANDLE *pdev)
{
 /* Note: On STM32F4-Discovery board only USB OTG FS core is supported. */

  GPIO_InitTypeDef GPIO_InitStructure;
 #ifdef USE_USB_OTG_FS

  RCC_AHB1PeriphClockCmd( RCC_AHB1Periph_GPIOA , ENABLE);

  /* Configure SOF VBUS ID DM DP Pins */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9  |
      GPIO_Pin_11 |
        GPIO_Pin_12;

  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_OTG1_FS) ;
  GPIO_PinAFConfig(GPIOA,GPIO_PinSource11,GPIO_AF_OTG1_FS) ;
  GPIO_PinAFConfig(GPIOA,GPIO_PinSource12,GPIO_AF_OTG1_FS) ;

  /* this for ID line debug */


  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_10;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
  GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_OTG1_FS) ;


  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
  RCC_AHB2PeriphClockCmd(RCC_AHB2Periph_OTG_FS, ENABLE) ;

  /* Intialize Timer for delay function */
  USB_OTG_BSP_TimeInit();
}

/**
  * @brief  USB_OTG_BSP_EnableInterrupt
  *         Configures USB Global interrupt
  * @param  None
  * @retval None
  */
void USB_OTG_BSP_EnableInterrupt(USB_OTG_CORE_HANDLE *pdev)
```

```
{

  NVIC_InitTypeDef NVIC_InitStructure;
  /* Enable USB Interrupt */
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

  NVIC_InitStructure.NVIC_IRQChannel = OTG_FS_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  /* Enable the Overcurrent Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = HOST_OVRCURR_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

  NVIC_Init(&NVIC_InitStructure);
}

/**
  * @brief  BSP_Drive_VBUS
  *         Drives the Vbus signal through IO
  * @param  state : VBUS states
  * @retval None
  */
void USB_OTG_BSP_DriveVBUS(USB_OTG_CORE_HANDLE *pdev, uint8_t state)
{
  /*
  On-chip 5 V VBUS generation is not supported. For this reason, a charge pump
  or, if 5 V are available on the application board, a basic power switch, must
  be added externally to drive the 5 V VBUS line. The external charge pump can
  be driven by any GPIO output. When the application decides to power on VBUS
  using the chosen GPIO, it must also set the port power bit in the host port
  control and status register (PPWR bit in OTG_FS_HPRT).

  Bit 12 PPWR: Port power
  The application uses this field to control power to this port, and the core
  clears this bit on an overcurrent condition.
  */
  if (0 == state)
  {
    /* DISABLE is needed on output of the Power Switch */
    GPIO_SetBits(HOST_POWERSW_PORT, HOST_POWERSW_VBUS);
  }
  else
  {
    /*ENABLE the Power Switch by driving the Enable LOW */
    GPIO_ResetBits(HOST_POWERSW_PORT, HOST_POWERSW_VBUS);
  }
}

/**
  * @brief  USB_OTG_BSP_ConfigVBUS
  *         Configures the IO for the Vbus and OverCurrent
  * @param  None
  * @retval None
```

```
  */
void  USB_OTG_BSP_ConfigVBUS(USB_OTG_CORE_HANDLE *pdev)
{
  GPIO_InitTypeDef GPIO_InitStructure;


  RCC_AHB1PeriphClockCmd(HOST_POWERSW_PORT_RCC, ENABLE);


  /* Configure Power Switch Vbus Pin */
  GPIO_InitStructure.GPIO_Pin = HOST_POWERSW_VBUS;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

  GPIO_Init(HOST_POWERSW_PORT, &GPIO_InitStructure);

  /* By Default, DISABLE is needed on output of the Power Switch */
  GPIO_SetBits(HOST_POWERSW_PORT, HOST_POWERSW_VBUS);

  USB_OTG_BSP_mDelay(200);   /* Delay is need for stabilising the Vbus Low
      in Reset Condition, when Vbus=1 and Reset-button is pressed by user */

}

/**
  * @brief  USB_OTG_BSP_TimeInit
  *         Initializes delay unit using Timer2
  * @param  None
  * @retval None
  */
static void USB_OTG_BSP_TimeInit (void)
{
#ifdef USE_ACCURATE_TIME
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Set the Vector Table base address at 0x08000000 */
  NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x00);

  /* Configure the Priority Group to 2 bits */
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

  /* Enable the TIM2 gloabal Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

  NVIC_Init(&NVIC_InitStructure);

  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
#endif
}

/**
  * @brief  USB_OTG_BSP_uDelay
  *         This function provides delay time in micro sec
```

```
  * @param  usec : Value of delay required in micro sec
  * @retval None
  */
void USB_OTG_BSP_uDelay (const uint32_t usec)
{

#ifdef USE_ACCURATE_TIME
  BSP_Delay(usec, TIM_USEC_DELAY);
#else
  __IO uint32_t count = 0;
  const uint32_t utime = (120 * usec / 7);
  do
  {
    if ( ++count > utime )
    {
      return ;
    }
  }
  while (1);
#endif

}

/**
  * @brief  USB_OTG_BSP_mDelay
  *          This function provides delay time in milli sec
  * @param  msec : Value of delay required in milli sec
  * @retval None
  */
void USB_OTG_BSP_mDelay (const uint32_t msec)
{
#ifdef USE_ACCURATE_TIME
  BSP_Delay(msec, TIM_MSEC_DELAY);
#else
  USB_OTG_BSP_uDelay(msec * 1000);
#endif

}

/**
  * @brief  USB_OTG_BSP_TimerIRQ
  *         Time base IRQ
  * @param  None
  * @retval None
  */
void USB_OTG_BSP_TimerIRQ (void)
{
#ifdef USE_ACCURATE_TIME
  if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
  {
    TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    if (BSP_delay > 0x00)
    {
      BSP_delay--;
    }
    else
    {
      TIM_Cmd(TIM2, DISABLE);
```

```
    }
  }
#endif
}

#ifdef USE_ACCURATE_TIME

/**
  * @brief  BSP_Delay
  *         Delay routine based on TIM2
  * @param  nTime : Delay Time
  * @param  unit : Delay Time unit : mili sec / micro sec
  * @retval None
  */
static void BSP_Delay(uint32_t nTime, uint8_t unit)
{

  BSP_delay = nTime;
  BSP_SetTime(unit);
  while (BSP_delay != 0);
  TIM_Cmd(TIM2, DISABLE);
}

/**
  * @brief  BSP_SetTime
  *         Configures TIM2 for delay routine based on TIM2
  * @param  unit : msec /usec
  * @retval None
  */
static void BSP_SetTime(uint8_t unit)
{
  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

  TIM_Cmd(TIM2, DISABLE);
  TIM_ITConfig(TIM2, TIM_IT_Update, DISABLE);


  if (unit == TIM_USEC_DELAY)
  {
    TIM_TimeBaseStructure.TIM_Period = 11;
  }
  else if (unit == TIM_MSEC_DELAY)
  {
    TIM_TimeBaseStructure.TIM_Period = 11999;
  }
  TIM_TimeBaseStructure.TIM_Prescaler = 5;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
  TIM_ClearITPendingBit(TIM2, TIM_IT_Update);

  TIM_ARRPreloadConfig(TIM2, ENABLE);

  /* TIM IT enable */
  TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);

  /* TIM2 enable counter */
```

```
  TIM_Cmd(TIM2, ENABLE);
}

#endif

/**
* @}
*/
```

3.2. Apéndice B

Reporte “diseño de software para un vehículo aéreo no tripulado”

Instituto Tecnológico de Estudios Superiores de Occidente

Master in Electronic Circuits Design

Advanced Embedded Systems

Raymundo Magaña Gómez

Project: Unmanned Aerial Vehicle (UAV)

Final delivery

Team:

Américo Lorenzana Gutiérrez

Manglio González Carrasco

September 12th of 2012

This document contains the following information:

- Project description and requirements
- Functional schematic diagram
- Flow diagram of the main MCU program
- Describe the advantages of using a 10-bit variable for sensor 2 instead of an 8-bit one.
- Justification of the solutions implemented and their advantages and disadvantages.
- Master and Slave MCU's code.
- Matlab code to compare the results calculated in the MCU and the "ideal" ones.
- Graphs showing the error between fixed and floating point against the "ideal" values.
- Team conclusions
- References and Bibliography

Project description

It will be considered that you are helping in the design of an unmanned aerial vehicle (UAV), and that your part in the project consists on the solution of the formula 1 below to calculate the acceleration of the UAV.

$$x = sensor\_1 - 3 \cdot sin(\frac{sensor\_2}{16}) + \frac{(sensor\_1 + sensor\_2)}{2}$$

Where:

$sensor\_1$ = Provides as output a 20 bit data variable with a measuring range between $15.385u_1$ and $120.854u_1$, information which is sent through SPI to our microcontroller.

$sensor\_2$ = Provides a differential output voltage that is directly proportional to the sensed acceleration. It will be considered that this differential voltage has already passed a signal conditioning stage, and that after this stage it supplies a voltage to our microcontroller that will be saved in an 8 bit unsigned variable, having -7.9u_2 in decimal for 0 and 255 in decimal for $15.3u_2$ (linear scale).

The rest of the variables in formula (1) use u_1 as units and consequently also x should be given in u_1 units.

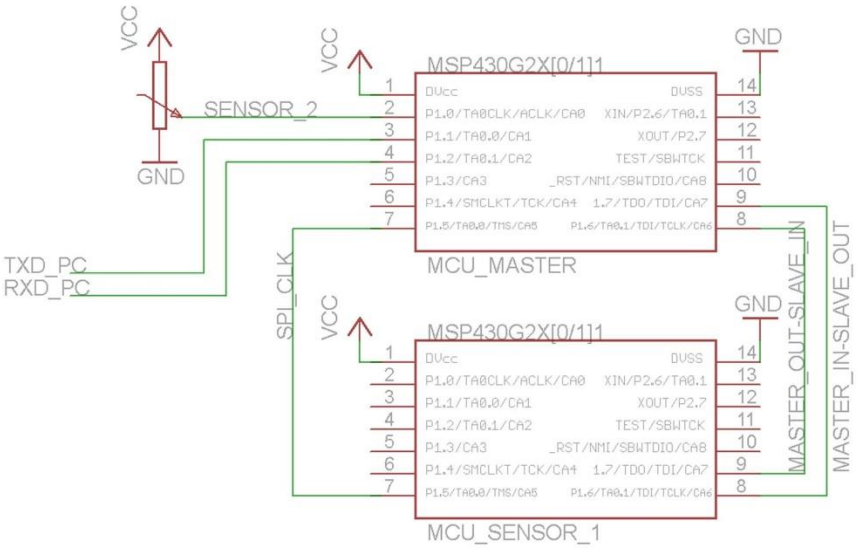
We have that $1u_2 = 0.23502u_1$

Requirements

1. The x calculated from formula 1 shall be within $\pm 1/2\ u_1$ of the exact output (calculated for instance using MatLab).
2. This assignment will be solved using fixed point and floating point operations.

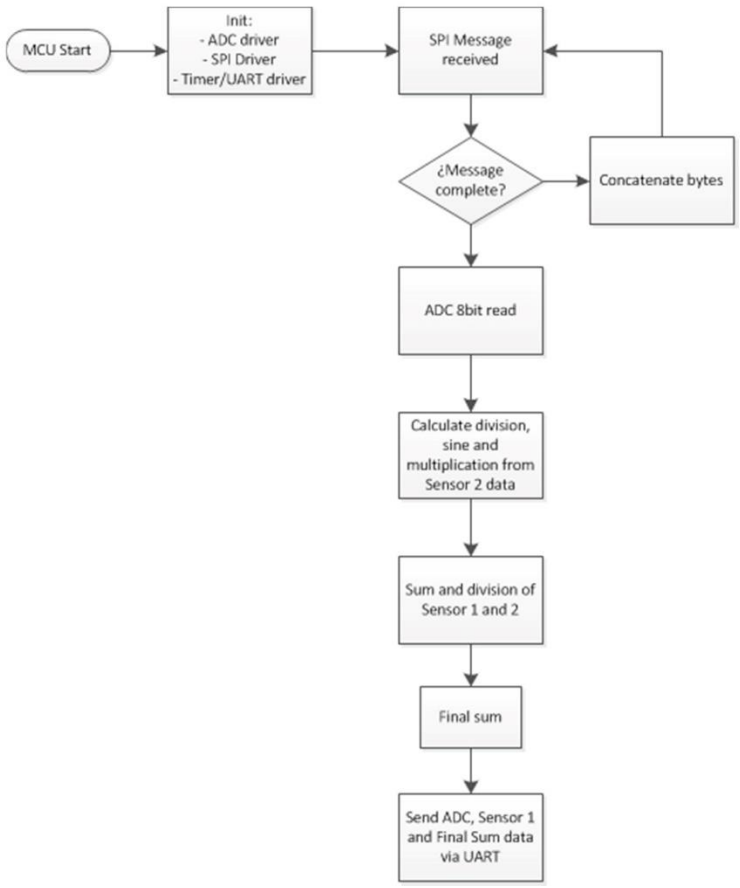
Functional schematic diagram

This is the functional schematic diagram of the system. As it can be seen, we use two MCU's (one is simulating a sensor through SPI) and a potentiometer connected to the ADC inputs of the main MCU.



Flow diagram of the main MCU program

This is the flow diagram of the main MCU code. We have an INIT state where the required drives are initialized. Then we jump into the data acquisition from the SPI driver. Once the SPI message is complete, the ADC is turned on and read. After that the calculation of the values is executed and then the data is sent through the UART to the PC. The cycle repeats itself endlessly.



Advantages of using a 10-bit variable instead of an 8-bit variable

Although the problem determines that an 8-bit variable should be used, the ranges that a variable of that size (0 to 255) have are too few. This means that little changes in the read signal are discarded. Since the project was not a real one, having an 8-bit or 10-bit is irrelevant. However should the sensor's changes be very little, but relevant, truncating its data could be potentially dangerous. We should have in mind that the resolution is calculated as follows:

$$Resolution = \frac{VOLTAGE\_REFERENCE}{ADC\_MAXIMUM\_VALUE}$$

With the above formula we can determine the minimum step. That value is really important when reading signals that have very little changes, but affect the system a lot. So again, depending on the situation having a bigger resolution is just not helpful, but mandatory.

Justification of the solution implemented and its advantages/disadvantages

The solution implemented can only handle either fixed or floating point operations. Both options cannot fit into the selected MCU. There are some disadvantages:

- Some functions are designed to work ONLY for this project.
- It was needed to use the compiler highest optimization configuration so that the floating point implementation could fit. Some level of optimization doesn't produce abnormal behavior on the final code, however if the highest optimization level is selected some routines are discarded or don't do what they're expected to do.
- Because of the way the solution was implemented, further improvement or additions to the code is almost impossible.

Nonetheless, several advantages were also accomplished:

- The precision of the final and intermediate results was not compromised and even most of the available memory was used, the error between the real vs. ideal values was well under the requirements.
- The values are calculated pretty fast at 1 MHz, if the MCU is configured to use the external oscillator they could be performed even faster.

The two options were designed differently:

- Fixed point operations were pretty straight forward and didn't require too much analysis. All the operations were stored in signed 32-bit long variables to have a wider precision. The final value of X was also stored in a signed 32-bit long variable (although it could be inferred from the formula that only positive values will be produced). An structure was defined to store the following values:
    - 32-bit long signed value
    - 32-bit long beta from the value

  This was used in all the values calculated because the beta was necessary so that arithmetic operations could be performed. Also when the data was sent to the PC, the PC program needed to know where the decimal point began.

- Floating point operations were a complete different animal. A lot of time was spent to understand how to perform them (a lot of documentation on the internet didn't explain how to do binary operations). Furthermore, the routines to calculate floating point values consume a lot more flash memory that fixed point and that was a big problem. When the operations were finally understood and designed, it was determined that a union of two variable types was needed:
    - 32-bit unsigned long value
    - Single precision (32-bit) float value

In the float part of the union, the floating point value is loaded, but it's not used. The 32-bit unsigned long part was a mirror of the float part, having the: sign, exponent and mantissa. This was needed so that integer operations could be performed. The mantissas were stored in 32-bit unsigned long variables so that precision would not be lost. It was decided that the division would not be implemented; instead it was handled as a multiplication. All the calculations are performed with 32-bit long unsigned variables and that is the type that is assigned to X (since all the values are handled as if they were float, X holds the corresponding format for a float variable). The PC receives the data in float format and it interprets it that way.

This is the code that the master MCU implemented:

```
/*****************************************************************/
/*!
      \file   UAV.c
      \author Manglio González Carrasco
      \date   August 25, 2012
      \brief  This is the main file. Here the core task is performed. A scheduler is used
                  to separate the tasks according to priority.
*/
/*****************************************************************/
#warning READ THIS COMMENT
/*
 * NOTE: FlOAT and FIXED are def/undef in "Project_Types.h"
 */

/*****************************************************************/
/* Includes */
/*****************************************************************/
#include "SPI_Handler.h"
#include "Project_Types.h"
#include "ADC_Driver.h"
#include "Sine.h"
#include "UART_Driver.h"
#include "Other.h"

#ifdef FIXED
      #include "MathFixed.h"
#else
      #ifdef FLOAT
            #include "Float_Math.h"
      #endif
#endif

/*****************************************************************/
/* Variables */
/*****************************************************************/









#ifdef FIXED
      Fixed s32FinalResult;
      Fixed u32SumResult;
      Fixed u32MultResult;
      Fixed SineResult;
      Fixed Sensor1;
      Fixed Sensor2;
#else
            #ifdef FLOAT
            uFloat fSensor1;
            uFloat fSensor2;
            uFloat fSumResult;
            uFloat fMultResult;
            uFloat fFinalResult;
            uFloat fSineResult;
            uFloat fMultCnt;
            uFloat fDivCnt;
      #endif
#endif

u08 u8ADCValue = 0;

/*****************************************************************/
/* Functions */
/*****************************************************************/
void Main_Task(void);

void main(void)
{
      WDTCTL = WDTPW + WDTHOLD;  // Stop WDT

      /*
       *     This initializes the oscillator at 1 MHz
       */
      DCOCTL = 0;
      BCSCTL1 = CALBC1_1MHZ;
      DCOCTL = CALDCO_1MHZ;
```

```
	/*
	 * Initialize the float part of the union with
	 * their respective value, in this case 3 for the
	 * multiplication and 0.5 for the division
	 */
	#ifdef FLOAT
		fMultCnt.fNumber = MUL_CNT;
		fDivCnt.fNumber = DIV_CNT;
	#endif

	SPI_Init();
	ADC_Init();

	TimerA_UART_init();
	__enable_interrupt();

	for(;;)
	{
		SPI_PeriodicTask();
		Main_Task();
	}
}


/**-------------------------------------------------------------------------------------------
* \fn		void Main_Task(void
* \brief	This function calculates the value once data from sensors has been acquired.
* \autor	Manglio González Carrasco
* \param	None
* \return	None
* \warning	None
-------------------------------------------------------------------------------------------*/
void Main_Task(void)
{
	if (SPI_u8CheckFlag())
	{









		SPI_ClearFlag();

	#ifdef FIXED
		/*
		 * The following happens here:
		 * This code is reachable only after a full SPI message is received.
		 * After that, the received value is cleared of the SOF. Its beta is stored.
		 * Then the ADC is read and the sine value is calculated accordingly. The Sensor2
		 * value is also calculated from this. The betas are known and are stored in the variables.
		 * Then the multiplication is performed and the addition. Then the addition, division and final
		 * addition. The ADC value, Sensor 1 value and its beta and the final value and its beta are sent
		 * through the virtual com.
		 */
		Sensor1.dwNum = (u32Sensor1 & SENSOR1_VALUE);
		Sensor1.dwBeta = SENSOR1_BETA;

		u8ADCValue = ADC_u8Read();

		SineResult.dwNum = Sine_Function(u8ADCValue);
		SineResult.dwBeta = SINE_BETA;

		Sensor2.dwNum = s32Convert_2U2_U1(u8ADCValue);
		Sensor2.dwBeta = SENSOR2_BETA;

		u32MultResult.dwNum = Fixed_MultCnt(&SineResult, 3u);
		u32MultResult.dwBeta = SineResult.dwBeta;
		u32MultResult = Fixed_add_sub(&Sensor1, &u32MultResult, FIXED_SUB);

		u32SumResult = Fixed_add_sub(&Sensor1, &Sensor2, FIXED_SUM);
		u32SumResult.dwNum = Fixed_div(&u32SumResult, 1);

		s32FinalResult = Fixed_add_sub(&u32SumResult, &u32MultResult,FIXED_SUM);

		TimerA_UART_print((u08*)&u8ADCValue,1);

		TimerA_UART_print((u08*)&Sensor1.dwBeta,4);
```

```
            TimerA_UART_print((u08*)&Sensor1.dwNum,4);

            TimerA_UART_print((u08*)&s32FinalResult.dwBeta,4);

            TimerA_UART_print((u08*)&s32FinalResult.dwNum,4);
      #else
            #ifdef FLOAT
                  /*
                   * Here happens the following:
                   * The ADC is read and its value is stored in an 8 bit variable.
                   * If the code has reached this part it's because the SPI has received a full message,
                   * therefore the received value is stored in the long part of the union of the Sensor1.
                   * Then the sine value is calculated accordingly to the value read from the ADC (Sensor2).
                   * The Sine value is multiplied and then added to the value of the Sensor1. Afterwards the
                   * value of the ADC is converted so that the Sensor2 value is in U1 units.
                   * Depending on the value of the ADC the value is either added or subtracted, that's the reason
                   * for the IF condition. Finally that value is multiplied by .5 (divided by 2) and added.
                   * After that, the value of the Sensor 1, ADC and Final Result are sent through the emulated
                   * RS232 port.
                   */
                  u8ADCValue = ADC_u8Read();

                  fSensor1.u32Number = u32Sensor1;

                  fSineResult.u32Number = Sine_Function(u8ADCValue);

                  fMultResult = Float_AddSubMul(&fSineResult,&fMultCnt, FLOAT_MUL);

                  fMultResult = Float_AddSubMul(&fSensor1, &fMultResult, FLOAT_SUM);

                  fSensor2.u32Number =  u32Convert_2U1_U1(u8ADCValue);

                  if (u8ADCValue > 86)
                  {
                        fSumResult = Float_AddSubMul(&fSensor1, &fSensor2, FLOAT_SUM);
                  }
                  else









                  {
                        fSumResult = Float_AddSubMul(&fSensor1, &fSensor2, FLOAT_SUB);
                  }

                  fSumResult = Float_AddSubMul(&fSumResult, &fDivCnt, FLOAT_MUL);

                  fFinalResult = Float_AddSubMul(&fMultResult, &fSumResult, FLOAT_SUM);

                  u32Sensor1 = 0;

                  TimerA_UART_print((u08*)&u8ADCValue,1);

                  TimerA_UART_print((u08*)&fSensor1.u32Number,4);

                  TimerA_UART_print((u08*)&fFinalResult.u32Number,4);
            #endif
      #endif
      }
      else
      {

      }
}


/*****************************************************************/
/*!
      \file   ADC_Driver.c
      \author Manglio González Carrasco
      \date   August 29, 2012
      \brief  This is the header file for the ADC driver.
*/
/*****************************************************************/

#ifndef ADC_DRIVER_H_
#define ADC_DRIVER_H_
```

```
	/*****************************************************************/
	/* Includes */
	/*****************************************************************/
	#include "Project_Types.h"
	#include <msp430g2231.h>


	/**-------------------------------------------------------------------------------------------
	* \fn		void ADC_Init(void)
	* \brief	This function initializes the ADC. Since polling will be used the ISR is not enabled.
	* \autor	Manglio González Carrasco
	* \param	 None
	* \return	None
	* \warning	 None
	-------------------------------------------------------------------------------------------*/
	void ADC_Init(void);

	/**-------------------------------------------------------------------------------------------
	* \fn		void ADC_u8Read(void)
	* \brief	This function starts a conversion and waits until it's done. Once it's done
	*			the value is returned.
	* \autor	Manglio González Carrasco
	* \param	 None
	* \return	None
	* \warning	 None
	-------------------------------------------------------------------------------------------*/
	u08 ADC_u8Read(void);

#endif /* __ADC_DRIVER_H__ */



/*****************************************************************/
/*!
	\file	ADC_Driver.c
	\author Manglio González Carrasco
	\date	August 29, 2012









	\brief  This is the ADC driver that will obtain the information from a voltage
			source.
*/
/*****************************************************************/

/*****************************************************************/
/* Includes */
/*****************************************************************/
#include "ADC_Driver.h"


void ADC_Init(void)
{
	ADC10CTL0 = SREF_1 + ADC10SHT_2 + ADC10ON + REFON;
	ADC10CTL1 = INCH_3;                       // A1
	ADC10AE0 |= 0x08;                         // PA.1 ADC option select
}

u08 ADC_u8Read(void)
{
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start

    while (ADC10CTL1 & ADC10BUSY);          // ADC10BUSY?

    ADC10CTL0 &= ~ENC;               // Sampling and conversion start

    return (ADC10MEM>>2u); //since only 8bits are used, the two less significant bits are ignored.
}



/*****************************************************************/
/*!
	\file	SPI_Driver.h
	\author Américo Lorenzana Gutierrez
	\date	June 12, 2012
	\brief  This is the header file for the SPI driver.
```

```
*/
/*****************************************************************/
#ifndef SPI_H_
#define SPI_H_

      /*****************************************************************/
      /* Includes */
      /*****************************************************************/
      #include <msp430g2231.h>
    #include "Project_Types.h"

      /*****************************************************************/
      /* Defines */
      /*****************************************************************/
      #define      SPI_8BITS          8u
      #define      SLAVE_DELAY        0x00FF

      /*****************************************************************/
      /* Public functions */
      /*****************************************************************/

      /**-------------------------------------------------------------------------------------------
      * \fn         void SPI_u8Init(void)
      * \brief      This function initializes the USI module in its SPI operation 8bit mode.
      * \autor      Américo Lorenzana Gutierrez
      * \param       None
      * \return     None
      * \warning     None
      -------------------------------------------------------------------------------------------*/
      void SPI_u8Init(void);

      /**-------------------------------------------------------------------------------------------
      * \fn         u08 SPI_u8WriteRead(u08 u8Data)
      * \brief      This function writes a byte in the SPI transmission bus and then reads it.
      * \autor      Américo Lorenzana Gutierrez
      * \param       u8Data
      * \return     USISRL









      * \warning     The delay for the slave device should be considered when performing timing analysis.
      -------------------------------------------------------------------------------------------*/
      u08 SPI_u8WriteRead(u08 u8Data);

#endif /*__SPI_DRIVER_H__*/


/*****************************************************************/
/*!
      \file   SPI_Driver.c
      \author Américo Lorenzana Gutierrez
      \date   June 12, 2012
      \brief  This driver initializes the SPI mode of the USI module to operate as a Master device. Also there is a function
              of read/write for sending or receiving data.
*/
/*****************************************************************/

/*****************************************************************/
/* Includes */
/*****************************************************************/
#include "SPI_Driver.h"

/*****************************************************************/
/* Functions */
/*****************************************************************/


void SPI_u8Init(void)
{
       USICTL0 =    (1*USIPE7)        /* USI  Port Enable Px.7 */
                        |(1*USIPE6)  /* USI  Port Enable Px.6 */
                        |(1*USIPE5)        /* USI  Port Enable Px.5 */
                        |(0*USILSB)        /* USI  LSB first  1:LSB / 0:MSB */
                        |(1*USIMST)        /* USI  Master Select / 1:Master*/
                        |(0*USIGE)         /* USI  General Output Enable Latch */
                  |(1*USIOE)          /* USI  Output Enable */
                        |(1*USISWRST);     /* USI  Software Reset */
```

```
    USICTL1 =      (0*USICKPH) /* USI  Sync. Mode: Clock Phase */
                         |(0*USII2C) /* USI  I2C Mode */
                         |(0*USISTTIE)/* USI  START Condition interrupt enable */
                         |(0*USIIE)   /* USI  Counter Interrupt enable */
                         |(0*USIAL)   /* USI  Arbitration Lost */
                         |(0*USISTP)  /* USI  STOP Condition received */
                         |(0*USISTTIFG)/* USI  START Condition interrupt Flag */
                         |(0*USIIFG);  /* USI  Counter Interrupt Flag */


      USICKCTL =   (USIDIV_4)|(USISSEL_2); /*Sets the speed of the SPI */

      USICTL0 &= ~USISWRST; /*Resets the module*/
}

u08 SPI_u8WriteRead(u08 u8Data)
{
      volatile u16 u8Delay = SLAVE_DELAY;

      USISRL = u8Data;
      USICNT = SPI_8BITS;

      do
      {
            u8Delay--;
      }while (u8Delay != 0);

      return(USISRL);
}



/*
 * uart.h
 *
 *  Created on: 31/08/2012









 *      Author: Americo
 */

#ifndef UART_H_
#define UART_H_

#include <msp430g2231.h>
#include "Project_Types.h"
//-------------------------------------------------------------------------------
// Hardware-related definitions
//-------------------------------------------------------------------------------
#define UART_TXD   0x02                     // TXD on P1.1 (Timer0_A.OUT0)
#define UART_RXD   0x04                     // RXD on P1.2 (Timer0_A.CCI1A)

//-------------------------------------------------------------------------------
// Conditions for 9600 Baud SW UART, SMCLK = 1MHz
//-------------------------------------------------------------------------------
#define UART_TBIT_DIV_2     (1000000 / (9600 * 2))
#define UART_TBIT           (1000000 / 9600)


//-------------------------------------------------------------------------------
// Function prototypes
//-------------------------------------------------------------------------------
void TimerA_UART_init(void);
void TimerA_UART_tx(unsigned char byte);
void TimerA_UART_print(u08* dwPrint, u08 bSize);


#endif /* UART_H_ */

/*
 * uart.c
 *
 *  Created on: 31/08/2012
 *      Author: Americo
 */
```

```
#include "UART_Driver.h"

unsigned int txData;                        // UART internal variable for TX
//unsigned char rxBuffer;                     // Received UART character

//-------------------------------------------------------------------------------
// Function configures Timer_A for full-duplex UART operation
//-------------------------------------------------------------------------------
void TimerA_UART_init(void)
{

    P1OUT = 0x00;                           // Initialize all GPIO
    P1SEL = UART_TXD + UART_RXD;            // Timer function for TXD/RXD pins
    P1DIR = 0xFF & ~UART_RXD;               // Set all pins but RXD to output

    TACCTL0 = OUT;                          // Set TXD Idle as Mark = '1'
    TACCTL1 = SCS + CM1 + CAP + CCIE;       // Sync, Neg Edge, Capture, Int
    TACTL = TASSEL_2 + MC_2;                // SMCLK, start in continuous mode
}
//-------------------------------------------------------------------------------
// Outputs one byte using the Timer_A UART
//-------------------------------------------------------------------------------
void TimerA_UART_tx(unsigned char byte)
{
    while (TACCTL0 & CCIE);                 // Ensure last char got TX'd
    TACCR0 = TAR;                           // Current state of TA counter
    TACCR0 += UART_TBIT;                    // One bit time till first bit
    TACCTL0 = OUTMOD0 + CCIE;               // Set TXD on EQU0, Int
    txData = byte;                          // Load global variable
    txData |= 0x100;                        // Add mark stop bit to TXData
    txData <<= 1;                           // Add space start bit
}

//-------------------------------------------------------------------------------
// Prints a string over using the Timer_A UART
//-------------------------------------------------------------------------------
```

```
void TimerA_UART_print(u08* dwPrint, u08 bSize)
{
      u08 bIndex = 0;
      dwPrint += (bSize-1);
      for(bIndex = 0;bIndex < bSize; bIndex++)
      {
      TimerA_UART_tx(*dwPrint);
      dwPrint--;
      }
}
//-------------------------------------------------------------------------------
// Timer_A UART - Transmit Interrupt Handler
//-------------------------------------------------------------------------------
#pragma vector = TIMERA0_VECTOR
__interrupt void Timer_A0_ISR(void)
{
    static unsigned char txBitCnt = 10;

    TACCR0 += UART_TBIT;                    // Add Offset to CCRx
    if (txBitCnt == 0)
    {                       // All bits TXed?
        TACCTL0 &= ~CCIE;                   // All bits TXed, disable interrupt
        txBitCnt = 10;                      // Re-load bit counter
    }
    else {
        if (txData & 0x01)
        {
          TACCTL0 &= ~OUTMOD2;              // TX Mark '1'
        }
        else
        {
          TACCTL0 |= OUTMOD2;               // TX Space '0'
        }
        txData >>= 1;
        txBitCnt--;
    }
}
```

```
/*****************************************************************/
/*!
      \file   UAV.c
      \author Americo Lorenzana Gutierrez
      \date   September 1, 2012
      \brief  This is the header file for the fixed point operations file.
*/
/*****************************************************************/
#ifndef MATHFIXED_H_
#define MATHFIXED_H_
      /*****************************************************************/
      /* Includes */
      /*****************************************************************/
      #include "Project_Types.h"

      /*****************************************************************/
      /* Defines */
      /*****************************************************************/
      #define FIXED_SUM  1
      #define FIXED_SUB  0

      /*****************************************************************/
      /* Typedefs */
      /*****************************************************************/
      typedef struct
      {
            s32 dwNum;
            s32 dwBeta;
      }Fixed;

      /*****************************************************************/
      /* Defines */
      /*****************************************************************/
      #define SINE_BETA         1000000
      #define SENSOR1_BETA      1000
      #define SENSOR2_BETA      1000000









      /*****************************************************************/
      /* Functions */
      /*****************************************************************/
      Fixed Fixed_add_sub(Fixed* fA, Fixed* fB, u08 Op);
      s32 Fixed_MultCnt(Fixed* fA,u08 Cnt);
      s32 Fixed_div(Fixed* fA,u08 Cnt);
#endif /* MATHFIXED_H_ */

/*****************************************************************/
/*!
      \file   UAV.c
      \author Americo Lorenzana Gutierrez
      \date   September 1, 2012
      \brief  Here the fixed point operations are performed.
*/
/*****************************************************************/

/*****************************************************************/
/* Includes */
/*****************************************************************/
#include "MathFixed.h"

/*****************************************************************************/
/*!
      \var   Fixed_add(Fixed fA, Fixed fB, u08 Op)
      \brief if Op 1 = Add two fixed point, else subtract fA to fB, return a signed
                fixed point with greatest beta.
*/
/*****************************************************************************/
Fixed Fixed_add_sub(Fixed* fA, Fixed* fB, u08 Op)
{
      Fixed FResult;
      u16 u16FactorA = 1;
      u16 u16FactorB = 1;

            if(fB->dwBeta > fA->dwBeta)
```

```
            {
                  u16FactorA = (fB->dwBeta/fA->dwBeta);
                  FResult.dwBeta = fB->dwBeta;
            }
            else if(fB->dwBeta < fA->dwBeta)
            {
                  u16FactorB = (fA->dwBeta/fB->dwBeta);
                  FResult.dwBeta = fA->dwBeta;
            }
            if (Op)
            {
                  FResult.dwNum = (fA->dwNum*u16FactorA) + (fB->dwNum*u16FactorB);
            }
            else
            {
                  FResult.dwNum = (fA->dwNum*u16FactorA) - (fB->dwNum*u16FactorB);
            }
      return(FResult);
}

/*****************************************************************************/
/*!
      \var   Fixed_MultCnt(Fixed * fA,u08 Cnt)
      \brief Multiply a fixed point with a constant, return a signed fixed point.
*/
/*****************************************************************************/
s32 Fixed_MultCnt(Fixed* fA,u08 Cnt)
{
      s32 FResult = 0;

      FResult = fA->dwNum * Cnt;

      return(FResult);
}
/*****************************************************************************/
/*!
      \var   Fixed_div(Fixed *fA, u08 Cnt)









      \brief Divide a fixed point number by a power of 2 number, return a signed fixed point.
*/
/*****************************************************************************/
s32 Fixed_div(Fixed* fA, u08 Cnt)
{
      s32 FResult = 0;

      FResult = fA->dwNum>>Cnt;

      return(FResult);
}

/*****************************************************************/
/*!
      \file   Float_Math.c
      \author Manglio González Carrasco
      \date   September 10, 2012
      \brief  This function add/subtract/multiply two single precision float numbers and
              returns its value in the corresponding format. The function was decided to
              merge because most of the code is repetitive.
*/
/*****************************************************************/
#ifndef FLOAT_MATH_H_
#define FLOAT_MATH_H_

      /*****************************************************************/
      /* Includes */
      /*****************************************************************/
      #include "Project_Types.h"

      /*****************************************************************/
      /* Defines */
      /*****************************************************************/
      #define FLOAT_SUM 1
      #define FLOAT_SUB 2
      #define FLOAT_MUL 3
```

```
#define MUL_CNT           3 //obtained from problem's equation
#define DIV_CNT       .5 //obtained from problem's equation

#define IMPLICIT_ONE            0x00800000
#define MANTISSA                0x007FFFFF
#define EXPONENT                0x7F800000
#define BIAS                    127
#define INCREASE_EXP            0x00800000
#define DECREASE_EXP            0x00800000
#define IS_IMPLICIT_ONE   0x7F800000
#define RIGHT_NORMALIZED  0x7F000000
#define LEFT_NORMALIZED         0x00800000


/*****************************************************************/
/* Definitions */
/*****************************************************************/
typedef union      uFloat
{
      u32 u32Number;
      float fNumber;
}uFloat;


/*****************************************************************/
/* Functions */
/*****************************************************************/

/**-------------------------------------------------------------------------------------------
* \fn         uFloat Float_AddSubMul(uFloat * A, uFloat * B, u08 Op)
* \brief      This function adds/subtracts/multiplies two float single precision numbers
*                   and returns a uFloat value (as defined by the typedef).
* \autor      Manglio González Carrasco
* \param       None
* \return     uFloat
* \warning     None
-------------------------------------------------------------------------------------------*/
```

```
      uFloat Float_AddSubMul(uFloat * A, uFloat * B, u08 Op);
#endif /* FLOAT_MATH_H_ */

/*****************************************************************/
/*!
      \file   Float_Math.c
      \author Manglio González Carrasco
      \date   September 10, 2012
      \brief  This function add/subtract/multiply two single precision float numbers and
                  returns its value in the corresponding format. The function was decided to
                  merge because most of the code is repetitive.
*/
/*****************************************************************/

/*****************************************************************/
/* Includes */
/*****************************************************************/
#include "Float_Math.h"

uFloat Float_AddSubMul(uFloat * A, uFloat * B, u08 Op)
{
      uFloat fResult;
      u08 u8Carries = 0;
      u32 u32A = 0;
      u32 u32B = 0;
      u08 ExpA = 0;
      u08 ExpB = 0;
      u32 Result = 0;
      u08 u8ImplicitOne = 0;

      /*
       * Obtain the mantissa and set the "implicit" 1 so that the operations
       * are performed correctly.
       */
      u32A = ((A -> u32Number & MANTISSA) | (IMPLICIT_ONE));
      u32B = ((B -> u32Number & MANTISSA) | (IMPLICIT_ONE));
```

```
/*
 * Obtain the exponent of each number and stored in an 8-bit variable.
 * That's why it's right-shifted 23 positions.
 */
ExpA = ((A -> u32Number & EXPONENT) >> 23);
ExpB = ((B -> u32Number & EXPONENT) >> 23);

if (Op == FLOAT_SUM || Op == FLOAT_SUM)
{
	/*
	 * If the exponent of A is bigger than B then the
	 * mantissa of B is right-shifted the difference
	 * between the exponents and the exponent of A is
	 * stored as the exponent of the result
	 */
	if (ExpA > ExpB)
	{
		u8Carries = (ExpA - ExpB);
		fResult.u32Number = (((u32)ExpA) << 23);
		u32B >>= u8Carries;
	}
	else
	{
		if (ExpA == ExpB)
		{
			/*
			 * This is a special case and either
			 * exponent could be stored. In this
			 * case the exponent of B was chose.
			 */
			fResult.u32Number = (((u32)ExpB) << 23);
		}
		else
		{
			/*
			 * This case is when the exponent of B is bigger
			 * than A's.
			 */
			fResult.u32Number = (((u32)ExpB) << 23);
			u8Carries = ((u08)ExpB - (u08)ExpA);
			u32A >>= u8Carries;
		}
	}

	if (Op == FLOAT_SUM)
	{
		Result = (u32A + u32B);
	}
	else if (Op == FLOAT_SUB)
	{
		Result = (u32A - u32B);
	}
}
else if (Op == FLOAT_MUL)
{
	/*
	 * Since the result of the multiplication is stored on a 32 bit
	 * variable, we decided to right-shift both values 8 positions, so that the
	 * most significant bits are conserved. The value is later right-shifted
	 * 16 positions so that the value is aligned.
	 */
	Result = (((u32A >> 8) * ((u32B >> 8))) >> 16);

	/*
	 * The value is left-shifted 9 positions so that in the upper 9
	 * bits is left the "implicit" 1, that's required to perform the
	 * normalization of the value.
	 */
	Result <<= 9;

	/*
	 * The exponent is stored in the final result. Since it's a
	 * multiplication both exponents are added, and the BIAS is
	 * subtracted. Since exponents can be positive or negative
```

```
		 * a cast is performed to maintain this property.
		 */
		fResult.u32Number = (u32)((u32)((s08)ExpA + (s08)ExpB - 127) << 23);
	}

	/*
	 * Should a 1 appear in the upper 8 bits, it means the value must
	 * be normalized. In the other case if the first bit of the mantissa
	 * is a 0, it also needs to be normalized.
	 */
	if (Result & IS_IMPLICIT_ONE)
	{
		/*
		 * In this case the exponent is increased by
		 * 1, every time a 0 is found and the mantissa
		 * is right-shifted by 1. The condition breaks
		 * once it finds a 1 in the last position of the 8
		 * bit exponent. Hence * the "implicit" 1 has been reached.
		 */
		for (u8ImplicitOne = 0; u8ImplicitOne < 8; u8ImplicitOne++)
		{
			if (!(Result & RIGHT_NORMALIZED))
			{
				/*
				 * Control variable is set to maximum to exit
				 */
				u8ImplicitOne = 8;
			}
			else
			{
				fResult.u32Number += INCREASE_EXP;
				Result >>= 1;
			}
		}
	}
	else
	{









		/*
		 * Same as above, but in this case the
		 * exponent is decreased by 1 and the mantissa
		 * is right-shifted by 1 every time a 0 is found.
		 * When a 1 appears in the last place of the 8-bit exponent
		 * the cycle exits.
		 */
		for (u8ImplicitOne = 0; u8ImplicitOne < 8; u8ImplicitOne++)
		{
			if (Result & LEFT_NORMALIZED)
			{
				/*
				 * Control variable is set to maximum to exit
				 */
				u8ImplicitOne = 8;
			}
			else
			{
				fResult.u32Number -= DECREASE_EXP;
				Result <<= 1;
			}
		}
	}

	/*
	 * Finally the normalized mantissa is added
	 * to the final value. An and operation is performed
	 * to eliminate the "implicit" 1.
	 */
	fResult.u32Number |= (Result & MANTISSA);

	return (fResult);
}

/**************************************************************************/
/*!
	\file   Sine.h
```

```
      \author Manglio González Carrasco
      \date   August 31, 2012
      \brief  This is the header file for the sine function.
*/
/*****************************************************************/

#ifndef SINE_H_
#define SINE_H_
      /*****************************************************/
      /* Includes */
      /*****************************************************************/
      #include "Project_Types.h"
      #include "Float_Math.h"
      /*****************************************************************/
      /* Defines */
      /*****************************************************************/
      #define MIN_SINE_VALUE    2025u //calculated from excel
      #define MIN_SINE_STEP     23u   //calculated from excel

      #define MIN_FLOAT_NEGATIVE_SINE_VALUE  -0.00202530305041 51 //calculated from excel
      #define MIN_FLOAT_SINE_STEP                          0.0000232332711759 //calculated from excel
      #define MIN_FLOAT_POSITIVE_SINE_VALUE  0.000019227574     //calculated from excel

      /**-------------------------------------------------------------------------------------------
      * \fn         s32 Sine_Function(u08 u8Angle)
      * \brief      This function calculates the possible value for the input. Since the ranges are known
      *                   and fixed, the function has been rigged to perform only on those ranges The value returned is
      *                   already in U1 units and divided by 16.
      * \autor      Manglio González Carrasco
      * \param       None
      * \return     s32
      * \warning     None
      -------------------------------------------------------------------------------------------*/
      #ifdef FIXED
            s32 Sine_Function(u08 u8Angle);
      #else
            #ifdef FLOAT









                  /**-------------------------------------------------------------------------------------------
                  * \fn         u32 Sine_Function(u08 u8Angle)
                  * \brief      This function calculates the possible value for the input. Since the ranges are known
                  *                   and fixed, the function has been rigged to perform only on those ranges The value returned is
                  *                   already in U1 units and divided by 16.
                  * \autor      Manglio González Carrasco
                  * \param       None
                  * \return     u32
                  * \warning     None
                  -------------------------------------------------------------------------------------------*/
                  u32 Sine_Function(u08 u8Angle);
            #endif
      #endif

#endif /* SINE_H_ */

/*****************************************************************/
/*!
      \file   Sine.c
      \author Manglio González Carrasco
      \date   August 31, 2012
      \brief  This function calculates the sine for a given value. The function has been
                  linearized and therefore it is implemented as a y = mx+b equation. With a
                  correction factor to minimize the error.
*/
/*****************************************************************/

/*****************************************************************/
/* Includes */
/*****************************************************************/
#include "Sine.h"

#ifdef FIXED
      s32 Sine_Function(u08 u8Angle)
      {
            s32 s32Result = 0u;
            u08 u8CorrectionFactor = 0u;
```

```
		/*
		 * Since this is a linear approximation, error is accumulated
		 * therefore a correction value is added to the final result so that
		 * the deviation is no bigger than .0001
		 */
		if ((u8Angle >= 12u) && (u8Angle <= 71u))
		{
			u8CorrectionFactor = (u08)((u8Angle/4u) - 1u);
		}
		else if ((u8Angle >= 72u) && (u8Angle <= 131u))
		{
			u8CorrectionFactor = (u08)((u8Angle/4u) - 2u);
		}
		else if ((u8Angle >= 132u) && (u8Angle <= 191u))
		{
			u8CorrectionFactor = (u08)((u8Angle/4u) - 3u);
		}
		else if (u8Angle >= 192u)
		{
			u8CorrectionFactor = (u08)((u8Angle/4u) - 4u);
		}

		s32Result = ((u8Angle*MIN_SINE_STEP) + u8CorrectionFactor) - MIN_SINE_VALUE;

		return (s32Result);
	}
#else
	#ifdef FLOAT

	u32 Sine_Function(u08 u8Angle)
	{
		uFloat fResult;
		uFloat fCnt;
		uFloat fAddValue;

		/*
		 * Here happens the following:









		 * The add value is loaded with its minimum and then it is multiplied by the ADC value.
		 * Since the float library doesn't handle negative numbers, it was calculated that after
		 * the 87h the value changes sign therefore the u8Angle is subtracted this bias so that the
		 * multiplication result is correct. Afterwards the value is added to the minimum sine value and the
		 * result is returned.
		 */

		fAddValue.fNumber = MIN_FLOAT_SINE_STEP;

		if (u8Angle > 87)
		{
			fCnt.fNumber = u8Angle - 88;

			fAddValue = Float_AddSubMul(&fAddValue, &fCnt, FLOAT_MUL);

			fResult.fNumber = MIN_FLOAT_POSITIVE_SINE_VALUE;

			fResult = Float_AddSubMul(&fResult, &fAddValue, FLOAT_SUM);
		}
		else
		{
			fCnt.fNumber = u8Angle;

			fAddValue = Float_AddSubMul(&fAddValue, &fCnt, FLOAT_MUL);

			fResult.fNumber = MIN_FLOAT_NEGATIVE_SINE_VALUE;

			if (u8Angle != 0)
			{
				fResult = Float_AddSubMul(&fResult, &fAddValue, FLOAT_SUB);
			}
		}

		return (fResult.u32Number);
	}
	#endif
#endif
```

```
/******      ***************************************************************/
/*!
      \file   SPI_Handler.h
      \author Manglio González Carrasco
      \date   August 25, 2012
      \brief  This is the header file for the SPI Handler.
*/
/*****************************************************************/
#ifndef SPI_HANDLER_H_
#define SPI_HANDLER_H_

      /*****************************************************************/
      /* Includes */
      /*****************************************************************/
      #include "Project_Types.h"
      #include "SPI_Driver.h"

      /*****************************************************************/
      /* Defines */
      /*****************************************************************/
      #define SOF                     0xFA
      #define DUMMY                   0x00
      #define MESSAGE_SIZE      0x03
      #define FIRST_BYTE        32u
      #define BYTE_SIZE         8u
      #define ASK_FOR_DATA    0xDD
      #define SENSOR1_VALUE     0x000FFFFF

      /*****************************************************************/
      /* Enums */
      /*****************************************************************/
      typedef enum
      {
            GET_DATA = 0x00,
            RISE_FLAG
      }eSPI_States;









      /*****************************************************************/
      /* Extern variables */
      /*****************************************************************/
      extern u32 u32Sensor1;
      /*****************************************************************/
      /* Public functions */
      /*****************************************************************/

      /**-------------------------------------------------------------------------------------------
      * \fn          void SPI_Init(void)
      * \brief       This function initializes the USI module in its SPI configuration.
      * \autor       Manglio González Carrasco
      * \param        None
      * \return      None
      * \warning      None
      -------------------------------------------------------------------------------------------*/
      void SPI_Init(void);

      /**-------------------------------------------------------------------------------------------
      * \fn          void SPI_PeriodicTask(void)
      * \brief       This is the periodic task of the SPI handler, it executes the next state.
      * \autor       Manglio González Carrasco.
      * \param        None
      * \return      None
      * \warning      None
      -------------------------------------------------------------------------------------------*/
      void SPI_PeriodicTask(void);

      /**-------------------------------------------------------------------------------------------
      * \fn          u08 SPI_u8CheckFlag(void)
      * \brief       This function checks if the data received flag has been set.
      * \autor       Manglio González Carrasco.
      * \param        None
      * \return      u8DataReceivedFlag
      * \warning      None
      -------------------------------------------------------------------------------------------*/
```

```
	u08 SPI_u8CheckFlag(void);

	/**-----------------------------------------------------------------------------------------------
	* \fn		void SPI_ClearFlag(void)
	* \brief	This function clears the data received flag.
	* \autor	Manglio González Carrasco.
	* \param	 None
	* \return	None
	* \warning	 None
	-----------------------------------------------------------------------------------------------*/
	void SPI_ClearFlag(void);

#endif /* __SPI_HANDLER_H__ */

/*****************************************************************/
/*!
	\file	SPI_Handler.c
	\author Manglio González Carrasco
	\date	August 25, 2012
	\brief	This is the HIL layer driver of the SPI module. Here the data is requested, formatted
		and a flag is raised when a complete message is received.
*/
/*****************************************************************/

/*****************************************************************/
/* Includes */
/*****************************************************************/
#include "SPI_Handler.h"

/*****************************************************************/
/* Functions */
/*****************************************************************/
void SPI_GetData(void);
void SPI_RiseFlag(void);

/*****************************************************************/
/* Variables */









/*****************************************************************/
u32 u32Sensor1 = 0x00000000;

static u08 u8DataReceivedFlag = FALSE;
#ifdef FIXED
	static u08 u8NextState = GET_DATA;
#endif

static u08 u8ByteCounter = 0x00;

#ifdef FIXED
	funct_p vfnp_SPIFunction[]=
	{
		SPI_GetData,
		SPI_RiseFlag,
	};
#endif

void SPI_Init(void)
{
	SPI_u8Init();
}

void SPI_PeriodicTask(void)
{
	#ifdef FIXED
		vfnp_SPIFunction[u8NextState]();
	#else
		#ifdef FLOAT
			/*
			 * The SPI handler had to be reduced so that the Float library could fit. Therefore the code does the following:
			 * Sends an "ASK_FOR_DATA" to the slave MCU (which it validates) and then takes the value and loads it
			 * in the long variable. It assumes the data received is correct (no header validation could be performed). Once it
			 * receives 4 bytes of data it raises the flag and waits to repeat.
			 */
			u08 u8SPIRead = SPI_u8WriteRead(ASK_FOR_DATA);
```

```
                        u8ByteCounter++;
                        u32Sensor1 |= (u32)((u32)u8SPIRead << (FIRST_BYTE - (BYTE_SIZE*u8ByteCounter))); /*the SOF byte is placed in the MSB*/

                        if (u8ByteCounter > MESSAGE_SIZE)
                        {
                              u8DataReceivedFlag = TRUE;
                              u32Sensor1 |= u8SPIRead;
                              u8ByteCounter = 0u;
                        }
                  #endif
            #endif
}

/**-------------------------------------------------------------------------------------------
* \fn          void SPI_GetData(void)
* \brief       This function writes on the SPI module and reads it to retrieve it. Once that is done,
*                 it checks if the byte read is a valid byte and then proceeds to format it.
* \autor       Manglio González Carrasco
* \param        None
* \return      None
* \warning      None
-------------------------------------------------------------------------------------------*/
void SPI_GetData(void)
{
      #ifdef FIXED
            u08 u8SPIRead = SPI_u8WriteRead(DUMMY);

            if (SOF == u8SPIRead)
            {
                  u8ByteCounter++;
                  u32Sensor1 |= (u32)((u32)u8SPIRead << (FIRST_BYTE - (BYTE_SIZE*u8ByteCounter))); /*the SOF byte is placed in the MSB*/
                  u8NextState = GET_DATA;
            }
            else
            {
                  if (0u < u8ByteCounter) /*if the SOF byte has been received*/
                  {









                        if (MESSAGE_SIZE == u8ByteCounter)
                        {
                              u8NextState = RISE_FLAG;
                              u32Sensor1 |= (u32)(u8SPIRead);
                              u8ByteCounter = 0u;
                        }
                        else
                        {
                              u8NextState = GET_DATA;
                              u8ByteCounter++;
                              u32Sensor1 |= (u32)((u32)u8SPIRead << (FIRST_BYTE - (BYTE_SIZE*u8ByteCounter))); /*the received byte is placed in the
corresponding space*/
                        }
                  }
                  else
                  {
                        u8NextState = GET_DATA;
                  }
            }
      #else

      #endif
}

/**-------------------------------------------------------------------------------------------
* \fn          void SPI_RiseFlag(void)
* \brief       This function sets the data received flag and clears the byte counter
* \autor       Manglio González Carrasco
* \param        None
* \return      None
* \warning      None
-------------------------------------------------------------------------------------------*/
#ifdef FIXED
      void SPI_RiseFlag(void)
      {
            u8DataReceivedFlag = TRUE;
            u8ByteCounter = 0u;
```

```
	}
#endif

void SPI_ClearFlag(void)
{
	u8DataReceivedFlag = FALSE;
	#ifdef FIXED
		u8NextState = GET_DATA;
	#endif
}

u08 SPI_u8CheckFlag(void)
{
	return u8DataReceivedFlag;
}

/*****************************************************************/
/*!
	\file   Sine.c
	\author Manglio González Carrasco
	\date   August 31, 2012
	\brief  This function calculates the sine for a given value. The function has been
		linearized and therefore it is implemented as a y = mx+b equation. With a
		correction factor to minimize the error.
*/
/*****************************************************************/
#ifndef OTHER_H_
#define OTHER_H_

	/*****************************************************************/
	/* Includes */
	/*****************************************************************/
	#include "Project_Types.h"
	#include "Float_Math.h"

	/*****************************************************************/
	/* Defines */
```

```
	/*****************************************************************/
	#define MIN_S2_VALUE	1856658 //value obtained from excel
	#define MIN_S2_STEP	21382 //value obtained from excel

	#define MIN_FLOAT_S2_STEP		0.021382212 //value obtained from excel
	#define MIN_FLOAT_S2_NEGATIVE_VALUE	-1.856658	//value obtained from excel
	#define	MIN_FLOAT_S2_POSITIVE_VALUE	0.003594424 //value obtained from excel


	/*****************************************************************/
	/* Public Functions */
	/*****************************************************************/

	/**-------------------------------------------------------------------------------------
	* \fn		s32 s32Convert_2U2_U1(u08 u8Data)
	* \brief	This function converts the values read from the ADC which
	*		are in U2 units to U1 units. The values have been multiplied
	*		by 1M so that at least the first 3 decimals are correct.
	* \autor	Manglio González Carrasco
	* \param	None
	* \return	s32
	* \warning	None
	-------------------------------------------------------------------------------------*/
	#ifdef FIXED
		s32 s32Convert_2U2_U1(u08 u8Data);
	#else
		#ifdef FLOAT
			/**-----------------------------------------------------------------------------------------
			* \fn		u32 u32Convert_2U2_U1(u08 u8Data)
			* \brief	This function converts the values read from the ADC which
			*		are in U2 units to U1 units. The values have been multiplied
			*		by 1M so that at least the first 3 decimals are correct.
			* \autor	Manglio González Carrasco
			* \param	None
			* \return	u32
			* \warning	None
			-----------------------------------------------------------------------------------------*/
```

```
                u32 u32Convert_2U1_U1(u08 u8Data);
            #endif
      #endif
#endif /* OTHER_H_ */

/*****************************************************************/
/*!
      \file   Sine.c
      \author Manglio González Carrasco
      \date   August 31, 2012
      \brief  This function converts the ADC read value that it's in U2 units
                to U1 units. Obviously the values have been escalated accordingly.
*/
/*****************************************************************/

/*****************************************************************/
/* Includes */
/*****************************************************************/
#include "Other.h"

#ifdef FIXED
      s32 s32Convert_2U2_U1(u08 u8Data)
      {
            /*
             * The value is merely multiplied and its value returned.
             */
            volatile s32 s32Result;

            s32Result = (s32)(((s32)u8Data * MIN_S2_STEP) - MIN_S2_VALUE);

            return (s32Result);
      }
#else
      #ifdef FLOAT
            u32 u32Convert_2U1_U1(u08 u8Data)
            {
                  uFloat fResult;









                  uFloat fCnt;
                  uFloat fAddValue;

                  /*
                   * The same operation as the sine calculation is applied here. Check Sine.h
                   * for further explanation.
                   */
                  fAddValue.fNumber = MIN_FLOAT_S2_STEP;;

                  if (u8Data > 86)
                  {
                        fCnt.fNumber = u8Data - 87;

                        fAddValue = Float_AddSubMul(&fAddValue, &fCnt, FLOAT_MUL);

                        fResult.fNumber = MIN_FLOAT_S2_POSITIVE_VALUE;

                        fResult = Float_AddSubMul(&fResult, &fAddValue, FLOAT_SUM);
                  }
                  else
                  {
                        fCnt.fNumber = u8Data;

                        fAddValue = Float_AddSubMul(&fAddValue, &fCnt, FLOAT_MUL);

                        fResult.fNumber = MIN_FLOAT_S2_NEGATIVE_VALUE;

                        if (u8Data != 0)
                        {
                              fResult = Float_AddSubMul(&fResult, &fAddValue, FLOAT_SUB);
                        }
                  }

                  return (fResult.u32Number);
            }
      #endif
#endif
```

```
/*****************************************************************/
/*!
      \file   Project_Types.h
      \author Manglio González Carrasco
      \date   August 25, 2012
      \brief  This header file contains all the definitions of variables required
                  in the project.
*/
/*****************************************************************/
#ifndef PROJECT_TYPES_H_
#define PROJECT_TYPES_H_

    typedef unsigned char u08;
    typedef unsigned short u16;
    typedef unsigned long u32;
    typedef unsigned long long u64;
    typedef signed char s08;
    typedef signed short s16;
    typedef signed long s32;
    typedef signed long long s64;
    typedef float f32;
    typedef double f64;

    #ifndef TRUE
      #define TRUE              (1)
    #endif

    #ifndef FALSE
      #define FALSE      (0)
    #endif

    typedef void (*const funct_p)(void);

    #define FIXED
    //#undef FIXED









    //#define FLOAT
    #undef FLOAT

#endif /* PROJECT_TYPES_H_ */
```

This is the code for the slave MCU:

```
**********************************/
/*
       \file       MainSensor.c
       \author     Américo Lorenzana
       \date       August 25, 2012
       \brief            This is the main file of the Sensor slave, just wait for an interrupt to send the current data.
 */
#include <msp430g2231.h>
#include "SPI.h"


//#define FIXED  //Define FIXED to specified use fixed point, if not use float point

#ifdef FIXED

      #define START_OF_MSG                  (u08)(0xFA)                                                    //Start of message
      #define SHIFTS_BYTE_NUMBER            (((u08)8*(bByteNumber-1)))                                   //Number of shifts to get to current byte
      #define CURRENT_BYTE_TO_SEND          (u08)0xFF&((dwSensedData)>>SHIFTS_BYTE_NUMBER)  //Get current byte to send by SPI
      #define MIN_DATA                      (u32)(15385)                                                 //Fixed point number, B = 1000
      #define MAX_DATA                      (u32)(120854)                                                  //Fixed point number, B = 1000
      u32 dwSensedData = MIN_DATA;//15.385

#else

      typedef union             //to handler float point number
      {
            float FloatNumber;
            u32     U32Number;
      }uFloat;
```

```
      #define SHIFTS_BYTE_NUMBER            (((u08)8*(bByteNumber-1)))            //Number of shifts to get to current byte
      #define CURRENT_BYTE_TO_SEND                (u08)0xFF&((dwSensedData.U32Number)>>SHIFTS_BYTE_NUMBER)  //Get current byte to send by SPI
      #define MIN_DATA_FLOAT                      (float)(15.385)                 //float point minimum value
      #define MAX_DATA_FLOAT                      (float)(120.854)                //float point maximum value

      uFloat dwSensedData = {MIN_DATA_FLOAT};                                     //15.385

#endif

u08 bByteNumber = 0;


void main(void)
{
      WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer

      vfnSPIInit();
      __enable_interrupt();
      _BIS_SR(LPM0_bits + GIE);             // Enter LPM0 w/ interrupt

      for(;;)
      {
      }

}

// USI interrupt service routine
#pragma vector=USI_VECTOR
__interrupt void universal_serial_interface(void)
{
      USICTL1 &= ~USIIFG;//Clear interrupt flag
      #ifdef FIXED
            if(!bByteNumber)//Update data to send
            {
                  dwSensedData &= 0x00ffffff;                         //Remove Start of message
                  dwSensedData += 1000;;                              //Add 1000/1000









                  dwSensedData %= MAX_DATA;                           //Check if SensedData reached maximum value
                  dwSensedData |= ((u32)START_OF_MSG<<24);            //Add Start of message
                  bByteNumber = 4;                                    //Send four bytes
            }

            bfnSPIPutData(CURRENT_BYTE_TO_SEND);              //Send the current byte, begin with the most significant byte
            bByteNumber--;

      #else
            if(0xDD ==  USISRL)
            {
                  if(!bByteNumber)                      //Update data to send
                  {
                        dwSensedData.FloatNumber = dwSensedData.FloatNumber + (float)1;       //Add 1.0
                        if(MAX_DATA_FLOAT <= dwSensedData.FloatNumber )                       //check if sensedData reached maximum value
                        {
                              dwSensedData.FloatNumber = MIN_DATA_FLOAT;                      //SensedData begin again
                        }
                              bByteNumber = 4;                                                //Send four bytes
                  }
                  bfnSPIPutData((u08)0xFF&((dwSensedData.U32Number)>>SHIFTS_BYTE_NUMBER));     //Send the current byte, begin with the most significant byte
                  bByteNumber--;
                  }

      #endif
}


#ifndef SPI_H_
#define SPI_H_
     #include <msp430g2231.h>
    #include "MyTypes.h"


      void vfnSPIInit(void);
      u08 bfnSPIPutData(u08 bData);
```

```
      u08 bfnSPIGetData(void);
#endif /*SPI_H_*/


//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
/*!
      \file   SPI.c
      \author Américo Lorenzana Gutierrez
      \date   June 12, 2012
      \brief
*/
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
//
//-------------------------------------------------------------------------------
#include "SPI.h"

//-------------------------------------------------------------------------------
// Variables
//-------------------------------------------------------------------------------



void vfnSPIInit(void)
{
       USICTL0 =   (1*USIPE7)         /* USI  Port Enable Px.7 */
                   |(1*USIPE6)        /* USI  Port Enable Px.6 */
                   |(1*USIPE5)        /* USI  Port Enable Px.5 */
                   |(0*USILSB)        /* USI  LSB first  1:LSB / 0:MSB */
                   #ifdef SPI_MASTER
                         |(1*USIMST)  /* USI  Master Select / 1:Master*/
                   #else
                         |(0*USIMST)  /* USI  Master Select  0:Slave*/
                   #endif
                         |(0*USIGE)   /* USI  General Output Enable Latch */
                   |(1*USIOE)         /* USI  Output Enable */









                   |(1*USISWRST);     /* USI  Software Reset */

    USICTL1 =      (0*USICKPH)        /* USI  Sync. Mode: Clock Phase */
                   |(0*USII2C)        /* USI  I2C Mode */
                   |(0*USISTTIE)      /* USI  START Condition interrupt enable */
                   |(1*USIIE)         /* USI  Counter Interrupt enable */
                   |(0*USIAL)         /* USI  Arbitration Lost */
                   |(0*USISTP)        /* USI  STOP Condition received */
                   |(0*USISTTIFG)     /* USI  START Condition interrupt Flag */
                   |(0*USIIFG);        /* USI  Counter Interrupt Flag */


      USICKCTL =   (USIDIV_4)
                   |(USISSEL_2);

      USICTL0 &= ~USISWRST;
}

u08 bfnSPIPutData(u08 bData)
{
      USISRL = bData;
      USICNT = 8;
//    while(!(USIIFG&USICTL1));
      return(TRUE);
}

u08 bfnSPIGetData(void)
{
      USICNT = 8;
      while(!(USIIFG&USICTL1));
      return(USISRL);
}
```

```
#ifndef __MY_TYPES_H__
#define __MY_TYPES_H__
    typedef unsigned char u08;
    typedef unsigned short u16;
    typedef unsigned long u32;
    typedef unsigned long long u64;
    typedef signed char s08;
    typedef signed short s16;
    typedef signed long s32;
    typedef signed long long s64;
    #ifndef TRUE
      #define TRUE              (1)
      #endif
      #ifndef FALSE
            #define FALSE       (0)
      #endif

    typedef union
      {
      u64    dl;
            u32 dw[2];
            u16 w[4];
            u08 b[8];
      }uReg64;
      typedef union
      {
            u32 dw;
            u16 w[2];
            u08 b[4];
      }uReg32;
      typedef union
      {
            u16 w;
            u08 b[2];
      }uReg16;









      typedef union
      {
            float f;
            u32 dw;
            u16 w[2];
            u08 b[4];
      }ufReg;

#endif /*__MY_TYPES_H__*/
```

This code is used to handle the fixed point data received from the MCU:

```
%
%     \author Americo Lorenzana


clc;
clear all;
format('longG');

s = serial('COM26','BaudRate',9600);
set(s,'InputBufferSize',4096);
fopen(s);

Xideal = zeros(10,1);
Xreal = zeros(10,1);
Xadc = zeros(10,1);
Xsensor1 = zeros(10,1);
Xsensor2 = zeros(10,1);
Xbs = zeros(10,1);
Xbreal = zeros(10,1);
for i=1:200
    InputData=fread(s,17);

    ADC = InputData(1);
```

```
    %Beta del sensor 1B1
    B1 = InputData(2:5);
    B1 = dec2hex(B1);
    B1 = [B1(1) B1(5) B1(2) B1(6) B1(3) B1(7) B1(4) B1(8)];
    B1 = hex2dec(B1);

    %Valor del sensor 1S1
    S1 = InputData(6:9);
    S1 = dec2hex(S1);
    S1 = [S1(1) S1(5) S1(2) S1(6) S1(3) S1(7) S1(4) S1(8)];
    S1 = hex2dec(S1);

    %Beta del resultadoBx
    Bx = InputData(10:13);
    Bx = dec2hex(Bx);
    Bx = [Bx(1) Bx(5) Bx(2) Bx(6) Bx(3) Bx(7) Bx(4) Bx(8)];
    Bx = hex2dec(Bx);

    %Valor del resultado
    X = InputData(14:17);
    X = dec2hex(X);
    X = [X(1) X(5) X(2) X(6) X(3) X(7) X(4) X(8)];
    X = hex2dec(X);

    Sensor1 = S1/B1;

    [Xideal(i) , Xsensor2(i) ] = uav(ADC,Sensor1);

    Xreal(i) = X/Bx;
    Xbreal(i) = Bx;
    Xsensor1(i) = Sensor1;
    Xbs(i) = B1;
    Xadc(i) = ADC;
end

figure;
hold on;









plot(Xideal,'b');
plot(Xreal,'r');

fclose(s);
```

This code is used to handle the floating point data received from the MCU:

```
%
%     \author Americo Lorenzanaclc;
clear all;
format('longG');

s = serial('COM26','BaudRate',9600);
set(s,'InputBufferSize',1024);
fopen(s);

Xideal = zeros(10,1);
Xreal = zeros(10,1);
Xadc = zeros(10,1);
Xsensor1 = zeros(10,1);
Xsensor2 = zeros(10,1);


for i=1:50
    InputData=fread(s,9);

    ADC = InputData(1);

    %Valor del sensor 1S1
    S1 = InputData(2:5);
    S1 = dec2hex(S1);
    S1 = [S1(1) S1(5) S1(2) S1(6) S1(3) S1(7) S1(4) S1(8)];
    S1 = hex2dec(S1);
    Sensor1 = typecast(uint32(S1),'single');

    %Valor del resultado
```

```
    X = InputData(6:9);
    X = dec2hex(X);
    X = [X(1) X(5) X(2) X(6) X(3) X(7) X(4) X(8)];
    X = hex2dec(X);
    X = typecast(uint32(X),'single');

    [Xideal(i) , Xsensor2(i) ] = uav(ADC,Sensor1);

    Xreal(i) = X;
    Xsensor1(i) = Sensor1;
    Xadc(i) = ADC;
end

figure;
hold on;
plot(Xideal,'b');
plot(Xreal,'r');

fclose(s);
```

This function is used to calculate the "ideal" value from the data received:

```
%
%     \author Americo Lorenzana
function [X,Sensor2]=uav(ADC,S1)

% S1 = input('S1:');
% ADC_Output = input('Valor ADC:');

S2 = ((ADC*.09080)-7.9)*.23502;  %Convierte el valor digital a valores S2 en unidaes U1
x1 = (S1+S2)/2;
x2 = 3*sind(S2/16);
x3 = S1;
X= x3-(x2)+(x1);
Sensor2 = S2;
end
```

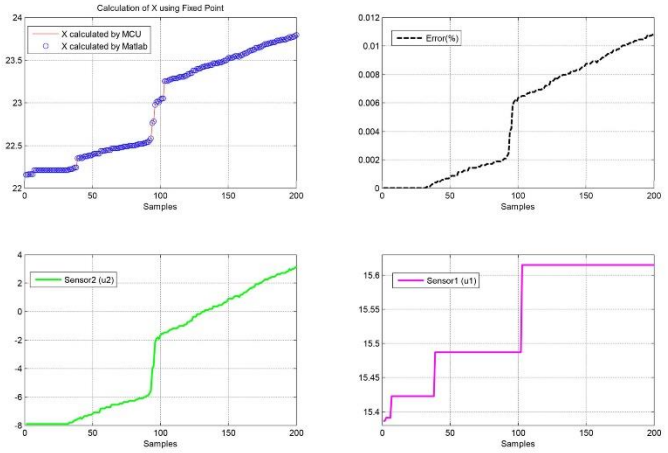
```
% fprintf('Input Sine = %.10f\n',S1);
% fprintf('X = %.10f\n',X);
```

This code is used to generate the graphs from the calculated data

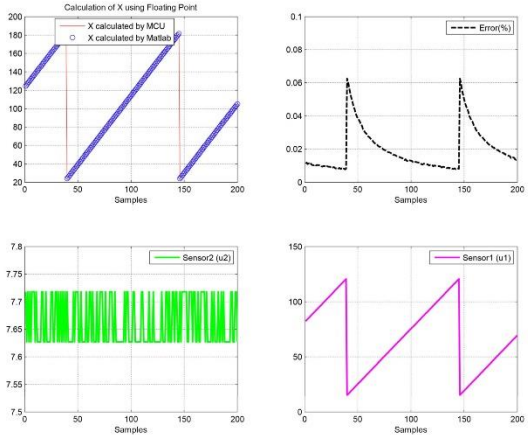
```
%
%     \author Americo Lorenzanaclose;
difference = abs((Xideal-Xreal))./Xideal;
error = difference*100;


[n,m] = size(Xreal);
n = 1:1:n;
figure;
hold on;

title('X calculado por el MCU')

subplot(2,2,1);
%plot(Xreal,'r','LineWidth',2);
plot(n,Xreal,'r-',n,Xideal,'bo');

legend('X calculated by MCU','X calculated by Matlab');
xlabel('Samples')
title('Calculation of X using Floating Point')
grid;


subplot(2,2,2);
plot(error,'k--','LineWidth',2);
legend('Error(%)');
xlabel('Samples')
axis([0,200,0,.1]);
grid;
```

```
subplot(2,2,3);
plot((Xsensor2./.23502),'g','LineWidth',2);
axis([0,200,6,8]);
xlabel('Samples')
legend('Sensor2 (u2)')
axis([0,200,7.5,7.8]);
grid;


subplot(2,2,4);
plot(Xsensor1,'m','LineWidth',2);
legend('Sensor1 (u1)')
xlabel('Samples')
axis([0,200,0,150]);

grid;
```

These are the graphs for the fixed point calculations, as it can be seen the error is well under the requirements:



These are the graphs for the floating point calculations, floating point has a bigger error than fixed ,but that is actually correct as the operations performed consecutively lose precision each time:



Team conclusions

This project was very time demanding and required a lot of reasoning (primarily to design the floating point library). All in all, I think that it helped me to have more in mind, if I hadn't by now, that MCU's memory is limited. I've never had to perform floating point operations before; so far I've always used fixed point operations for two reasons:

- Easier to implement
- Easier to debug

However I can see the benefits of having floating point operations, but the time required to design and code a "custom" library for MCU's that don't have an FPU renders it useless. Again it depends on the project, because sometimes fixed point operations fall short to the problem's needs and having 64-bit operations is out of the question.

As a final note I must say that projects like these can be good for developing coding skills such as:

- Using pointers instead of variables directly to consume less memory.
- Using operations of the size of the MCU (8, 16, 32, etc.) so that they're performed faster and consume less memory.

Manglio González Carrasco.

With this project I learned:

That the use of embedded systems for applications requiring numerical handling, offers a very economical and cost effective solution taking into account the complexity in development. To solve arithmetic problems have various forms of numerical representation, two of them are floating point and fixed point.

The experience I gained in developing the UAV Project, I could detect some advantages and disadvantages of using floating point and fixed point.

Using fixed point was very easy and fast, the fixed point library developed for the project not spending a lot of resources, the final accuracy is maintained in a good range, a disadvantage of the fixed point was the dynamic range, we are limited to certain decimal and the only way to increase the range is moving the point to the left or right.

Using floating point was complicated, library spend many resources for all the routines to perform the operation of addition, subtraction and multiplication, floating point allow much higher dynamic range than when using fixed point, but the accuracy is not improved, adding large number with very small numbers lost accuracy, which did not happen in fixed point if everything was in range.

In conclusion, to develop an application using arithmetic handling for an embedded system the optimal way to do it, is to analyze the numerical ranges that will be used in the App and use fixed point numbers, this becomes the solution that spend less resources and have more accuracy.

Américo Lorenzana Gutiérrez

References and Bibliography

- **MSP430G2x21, MSP430G2x31 Mixed Signal Microcontroller (Rev. I)**
  http://www.ti.com/lit/ds/symlink/msp430g2231.pdf
- **MSP430x2xx Family User's Guide (Rev. I)**
  http://www.ti.com/lit/ug/slau144i/slau144i.pdf
- **IEEE 754 CONVERTER**
  http://www.h-schmidt.net/FloatConverter/IEEE754.html

3.3. Apéndice C

Código fuente “diseño de un sistema de archivos”

3.3.1 Makefile.mak

```
all: datehandler vdisk createvd vdformat dumpsec block_node_handler
logical_sector filehandling shell1

shell1: shell1.c tiposdatos.h tiposdatos.c block_node_handler.h
logical_sector.h datehandler.h filehandling.h block_node_handler.o
logical_sector.o vdisk.o filehandling.o datehandler.o
      gcc -o shell1 shell1.c tiposdatos.c filehandling.o block_node_handler.o
logical_sector.o vdisk.o datehandler.o

filehandling: filehandling.c  tiposdatos.h block_node_handler.h
logical_sector.h block_node_handler.o logical_sector.o vdisk.o tiposdatos.c
      gcc -c filehandling.c tiposdatos.c

createvd: createvd.c vdisk.h vdisk.o
      gcc -o createvd createvd.c vdisk.o

vdformat: vdformat.c vdisk.h tiposdatos.h vdisk.o
      gcc -o vdformat vdformat.c vdisk.o

dumpsec: dumpsec.c vdisk.h vdisk.o
      gcc -o dumpsec dumpsec.c vdisk.o

block_node_handler: block_node_handler.c block_node_handler.h vdisk.h vdisk.o
tiposdatos.h datehandler.h datehandler.o tiposdatos.h tiposdatos.c
      gcc -c block_node_handler.c tiposdatos.c

logical_sector: logical_sector.c logical_sector.h tiposdatos.h vdisk.h
vdisk.o tiposdatos.h tiposdatos.c
      gcc -c logical_sector.c tiposdatos.c

datehandler: datehandler.c datehandler.h
      gcc -c datehandler.c

vdisk: vdisk.c vdisk.h
      gcc -c vdisk.c
```

3.3.2 Block_node_hanlder.h

```
#ifndef __BLOCK_NODE_HANDLER_H__
#     define       __BLOCK_NODE_HANDLER__H__

int isblockfree(int block);
int nextfreeblock(void);
int assignblock(int block);
int unassignblock(int block);
int writeblock(int block,char *buffer);
int readblock(int block,char *buffer);
int searchinode(char *filename);
int removeinode(int numinode);

int isinodefree(int inode);
int nextfreeinode();
int assigninode(int inode);
int unassigninode(int inode);



#endif
```

3.3.3 datatypes.h

```
#ifndef ___DATA_TYPES_H__

      #define ___DATA_TYPES_H__

      struct SECBOOT {
            char jump[4];
            char nombre_disco[8];
            unsigned char sec_res;        // 1 sector de arranque
            unsigned char sec_mapa_bits_nodos_i;  // 1 sector
            unsigned char sec_mapa_bits_bloques;  // 4 sectores
            unsigned short sec_tabla_nodos_i;   // 1 sector
            unsigned short sec_log_unidad;      // 54400 sectores
            unsigned char sec_x_bloque;         // 4 sectores por bloque
            unsigned char heads;                // 20 superficies
            unsigned char cyls;                 // 160 cilindros
            unsigned char secfis;               // 17 sectores
            char restante[487];
      };

      struct INODE {
            char name[20];
            unsigned short uid;
            unsigned short gid;
            unsigned short perms;
            unsigned int datetimecreat;
```

```
            unsigned int datetimemodif;
            unsigned int size;
            unsigned short blocks[10];
            unsigned short indirect;
            unsigned short indirect2;
      };

      struct OPENFILES {
            int inuse;
            unsigned short inode;
            int currpos;
            int currbloqueenmemoria;
            char buffer[2048];
            unsigned short buffindirect[1024];
      };

      SECBOOT secboot;


#endif
```

3.3.4 datehandler.h

```
#ifndef __DATEHANDLER_H__
#     define      __DATEHANDLER_H__

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <time.h>

typedef union DATE
{
      struct
      {
            unsigned char year            : 6;
            unsigned char month     : 4;
            unsigned char day             : 5;
            unsigned char hour            : 5;
            unsigned char min       : 6;
            unsigned char sec       : 6;
      }sdate;
      unsigned int uDate;
}DATE;

unsigned int datetoint(DATE date);
int inttodate(DATE *date,unsigned int val);
unsigned int currdatetimetoint();


#endif
```

3.3.5 filehandling.h

```
vddirent *vdreaddir(VDDIR *dirdesc);
VDDIR *vdopendir(char *path);

int vdseek(int fd, int offset, int whence);
int vdwrite(int fd, char *buffer, int bytes);
int vdread(int fd, char *buffer, int bytes);
int vdcreat(char* filename, unsigned short permissions);
int vdopen(char* filename, unsigned char mode);
int vdunlink(char * filename);
int vdclose(int fd);

int getsizeof(char * filename);
```

3.3.6 logical_sector.h

```
#ifndef __LOGICAL_SECTOR_H__

#     define      __LOGICAL_SECTOR_H__

#     define      SUCCESS 1
#     define      FAILED      -1

#     include     "vdisk.h"
#include "tiposdatos.h"
      /*
            Esta función lee el(los) sector(es) indicado(s) por los parametros
y retorna como fallido o exitoso
      */
      int vdreadseclog(int sector, char * buffer);

      /*
            Esta función escribe el(los) sector(es) indicado(s) por los
parametros y retorna como fallido o exitoso
      */
      int vdwriteseclog(int sector, char * buffer);

#endif
```

3.3.7 tiposdatos.h

```
#ifndef ___DATA_TYPES_H__

	#define ___DATA_TYPES_H__

	typedef struct SECBOOT {
		char jump[4];
		char nombre_disco[8];
		unsigned char sec_res;		// 1 sector de arranque
		unsigned char sec_mapa_bits_nodos_i;  // 1 sector
		unsigned char sec_mapa_bits_bloques;  // 4 sectores
		unsigned char paddingbyte;
		unsigned short sec_tabla_nodos_i;   // 1 sector
		unsigned short sec_log_unidad;      // 54400 sectores
		unsigned char sec_x_bloque;         // 4 sectores por bloque
		unsigned char heads;                // 20 superficies
		unsigned char cyls;                 // 160 cilindros
		unsigned char secfis;               // 17 sectores
		char restante[488];
	}SECBOOT;

	typedef struct INODE {
		char name[20];
		unsigned short uid;
		unsigned short gid;
		unsigned short perms;
		unsigned int datetimecreat;
		unsigned int datetimemodif;
		unsigned int size;
		unsigned short blocks[10];
		unsigned short indirect;
		unsigned short indirect2;
		unsigned short PaddingShort;
	}INODE;

	typedef struct OPENFILES {
		int inuse;
		unsigned short inode;
		int currpos;
		int currbloqueenmemoria;
		char buffer[2048];
		unsigned short buffindirect[1024];
	}OPENFILES;

	/*************************************************************
	/* Tipos de datos para el manejo de directorios
	 *************************************************************/

	typedef int VDDIR;


	typedef struct vddirent
	{
		char *d_name;
```

```
}vddirent;


extern unsigned char inicio_nodos_i;
extern OPENFILES openfiles[8];
extern SECBOOT secboot;
extern INODE inode[8];
extern unsigned char secboot_en_memoria;
extern unsigned char inodesmap_en_memoria;
extern unsigned char nodos_i_en_memoria;
extern unsigned char inodesmap[2];
extern int mapa_bits_bloques;
extern unsigned char blocksmap_en_memoria;
extern unsigned short inicio_area_datos;
extern unsigned char blocksmap[16384];

#endif
```

3.3.8 vdisk.h

```
#define HEADS 20
#define SECTORS 17
#define CYLINDERS 160

int vdwritesector(int drive, int head, int cylinder, int sector, int nsecs,
char *buffer);

int vdreadsector(int drive, int head, int cylinder, int sector, int nsecs,
char *buffer);
```

3.3.9 block_node_handler.c

```
#include "block_node_handler.h"
#include "vdisk.h"
#include "tiposdatos.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


//
*****************************************************************************
*
// Para el mapa de bits del área de de datos
//
*****************************************************************************
*
int isblockfree(int block)
```

```
{
	int offset=block/8;  // Obtener en que byte
	int shift=block%8;   // y bit, que indica si el bloque está libte
	int result;
	int i;

	// Es importante tenerlo en memoria porque ahí es donde tenemos
	// la información del disco, cuáles son los sectores donde hay qué
	if(!secboot_en_memoria)
	{
		result=vdreadsector(0,0,0,1,1,(char *) &secboot);
		secboot_en_memoria=1;
	}

	// Calculo el sector donde está el mapa de bits para los bloques
	mapa_bits_bloques= secboot.sec_res+secboot.sec_mapa_bits_nodos_i;

// ¿Está en memoria el mapa de bits de bloques?, si no, cargarlo a // memoria
	if(!blocksmap_en_memoria)
	{
		// Leer todos los sectores del mapa de bits a memoria
		for(i=0;i<secboot.sec_mapa_bits_bloques;i++)

	result=vdreadseclog(mapa_bits_bloques+i,blocksmap+i*512);
		blocksmap_en_memoria=1;
	}


	if(blocksmap[offset] & (1<<shift))
		return(0);
	else
		return(1);
}

// Buscar en el mapa de bits cuál es el siguiente bloque libre
int nextfreeblock()
{
	int i,j;
	int result;

	// Ya quedó explicado arriba
	if(!secboot_en_memoria)
	{
		result=vdreadsector(0,0,0,1,1,(char *) &secboot);
		secboot_en_memoria=1;
	}

	mapa_bits_bloques = secboot.sec_res+secboot.sec_mapa_bits_nodos_i;

	if(!blocksmap_en_memoria)
	{
		for(i=0;i<secboot.sec_mapa_bits_bloques;i++)
			result=vdreadseclog(mapa_bits_bloques+i,blocksmap+i*512);
		blocksmap_en_memoria=1;
	}

	// Buscar el primer byte en el mapa de bloques donde hay al menos un
```

```
	// bloque libre
	i=0;
	while(blocksmap[i]==0xFF && i<secboot.sec_mapa_bits_bloques*512)
		i++;

	// Si no llegamos al final
	if(i<secboot.sec_mapa_bits_bloques*512)
	{
		j=0;
		while(blocksmap[i] & (1<<j) && j<8)
			j++;

		return(i*8+j);  // Regresando cual es el primer bloque libre
encontrado
	}
	else
		return(-1);  // Llegamos al final del mapa y no encontramos
				// un bloque libre

}

//Poner un bloque como asignado
int assignblock(int block)
{
	int offset=block/8;
	int shift=block%8;
	int result;
	int i;



	if(!secboot_en_memoria)
	{
		result=vdreadsector(0,0,0,1,1,(char *) &secboot);
		secboot_en_memoria=1;
	}

	mapa_bits_bloques= secboot.sec_res+secboot.sec_mapa_bits_nodos_i;

	if(!blocksmap_en_memoria)
	{
		for(i=0;i<secboot.sec_mapa_bits_bloques;i++)
			result=vdreadseclog(mapa_bits_bloques+i,blocksmap+i*512);
		blocksmap_en_memoria=1;
	}
	//printf("asignando bloque\n");
	// Poner en 1 el bit en el byte que corresponde al número de bloque
	blocksmap[offset]|=(1<<shift);

	// Escribir ese sector en el disco
	offset=(offset/512)*512;
		// 0 .. 511 = 0
		// 512 .. 1023 = 512
		// 1024 .. 1535 = 1024
		// offset=offset-(offset%512);
```

```
	vdwriteseclog(mapa_bits_bloques+offset,blocksmap);

	return(1);
}

//Liberar un bloque
int unassignblock(int block)
{
	int offset=block/8;
	int shift=block%8;
	int result;
	char mask;
	int sector;
	int i;

	if(!secboot_en_memoria)
	{
		result=vdreadsector(0,0,0,1,1,(char *) &secboot);
		secboot_en_memoria=1;
	}

	mapa_bits_bloques= secboot.sec_res+secboot.sec_mapa_bits_nodos_i;

	if(!blocksmap_en_memoria)
	{
		for(i=0;i<secboot.sec_mapa_bits_bloques;i++)
			result=vdreadseclog(mapa_bits_bloques+i,blocksmap+i*512);
		blocksmap_en_memoria=1;
	}

	blocksmap[offset]&=(char) ~(1<<shift);

	sector=(offset/512)*512;
	vdwriteseclog(mapa_bits_bloques+sector,blocksmap+sector*512);

	return(1);
}

//
**************************************************************************
*****
// Lectura y escritura de bloques
//
**************************************************************************
*****
int writeblock(int block,char *buffer)
{
	int result;
	int i;

	// Verificar si el sector de boot está en memoria y si no, cárgalo
	if(!secboot_en_memoria)
	{
		result=vdreadsector(0,0,0,1,1,(char *) &secboot);
		secboot_en_memoria=1;
	}
```

```
	// Calcula cuál es el sector donde inician los bloques
	inicio_area_datos=secboot.sec_res+secboot.sec_mapa_bits_nodos_i
+secboot.sec_mapa_bits_bloques+secboot.sec_tabla_nodos_i;

	// Escribir todos los sectores lógicos que corresponden al bloque
	for(i=0;i<secboot.sec_x_bloque;i++)
		vdwriteseclog(inicio_area_datos+(block-
1)*secboot.sec_x_bloque+i,buffer+512*i);
		// Escribimos cada uno de los sectores que corresponden al bloque
		// buffer+512*i hacemos que cada iteración del for un subbuffer de
512
		// bytes.
	return(1);
}

//Lee un bloque
int readblock(int block,char *buffer)
{
	int result;
	int i;

	if(!secboot_en_memoria)
	{
		result=vdreadsector(0,0,0,1,1,(char *) &secboot);
		secboot_en_memoria=1;
	}

	inicio_area_datos=secboot.sec_res+secboot.sec_mapa_bits_nodos_i+secboot.
sec_mapa_bits_bloques+secboot.sec_tabla_nodos_i;

	for(i=0;i<secboot.sec_x_bloque;i++)
		vdreadseclog(inicio_area_datos+(block-
1)*secboot.sec_x_bloque+i,buffer+512*i);
	return(1);
}



// A partir de una posición en el archivo determina la dirección de memoria
// donde está el apuntador en el nodo i que está cargado en memoria.
unsigned short *postoptr(int fd,int pos)
{
	int currinode;
	unsigned short *currptr;
	unsigned short indirect1;

	// El número de inodo actual lo obtenemos de la tabla de archivos
abiertos
	currinode=openfiles[fd].inode;

	// Está en los primeros 10 K
	if((pos/1024)<10)
		// Está entre los 10 apuntadores directos
		currptr=&inode[currinode].blocks[pos/1024];
	else if((pos/1024)<1034)
```

```
	{
		// Si el apuntador a bloque indirecto está vacío, asígnale un
bloque
		if(inode[currinode].indirect==0)
		{
			// El siguiente bloque que disponible de acuerdo al mapa de
bits
			indirect1=nextfreeblock();
			assignblock(indirect1); // Asígnalo
			inode[currinode].indirect=indirect1;
		}
		// En la tabla de archivos abiertos tenemos el buffer que almacena
		// el bloque de apuntadores
		currptr=&openfiles[fd].buffindirect[pos/1024-10];
	}
	else
		return((unsigned short*)-1 /*NULL*/);

	return(currptr);
}

unsigned short *currpostoptr(int fd)
{
	unsigned short *currptr;

	currptr=postoptr(fd,openfiles[fd].currpos);

	return(currptr);
}

// En un inodo específico escribe el nombre, atributos y usuario y dueño
// del nuevo archivo.
// Usada por la función vdcreate
int setninode(int num, char *filename, unsigned short atribs, int uid, int
gid)
{
	int i;

	int result;

	if (num >= 0)
	{
		if(!secboot_en_memoria)
		{
			result=vdreadsector(0,0,0,1,1,(char *) &secboot);
			secboot_en_memoria=1;
		}

	inicio_nodos_i=secboot.sec_res+secboot.sec_mapa_bits_nodos_i+secboot.sec
_mapa_bits_bloques;

		if(!nodos_i_en_memoria)
		{
			result=vdreadseclog(inicio_nodos_i,(char*)&inode);
			nodos_i_en_memoria=1;
		}
```

```
            strncpy(inode[num].name,(char*)filename,20);

            if(strlen(inode[num].name)>19)
                  inode[num].name[19]='\0';

            inode[num].datetimecreat=currdatetimetoint();
            inode[num].datetimemodif=currdatetimetoint();
            inode[num].uid=uid;
            inode[num].gid=gid;
            inode[num].perms=atribs;
            inode[num].size=0;

            for(i=0;i<10;i++)
                  inode[num].blocks[i]=0;

            inode[num].indirect=0;
            inode[num].indirect2=0;

            // Optimizar la escritura escribiendo solo el sector lógico que
            // corresponde al inodo que estamos asignando.
            // i=num/8;
            // result=vdwriteseclog(inicio_nodos_i+i,&inode[i*8]);

            result=vdwriteseclog(inicio_nodos_i,(char *)&inode);
      }

      return(num);
}


// Buscar a partir del nombre del archivo el número de inodo correspondiente
int searchinode(char *filename)
{
      int i;
      int free;
      int result;

      if(!secboot_en_memoria)
      {
            result=vdreadsector(0,0,0,1,1,(char *) &secboot);
            secboot_en_memoria=1;
      }
      inicio_nodos_i=secboot.sec_res+secboot.sec_mapa_bits_nodos_i+secboot.sec
_mapa_bits_bloques;

      if(!nodos_i_en_memoria)
      {
            result=vdreadseclog(inicio_nodos_i,(char*)&inode);

            nodos_i_en_memoria=1;
      }

      if(strlen(filename)>19)
            filename[19]='\0';
```

```
	i=0;
	while(strcmp(inode[i].name,filename) && i<8)
		i++;

	if(i>=8)
		return(-1);
	else
		return(i);
}


// Elimina un inodo de la tabla de nodos i
// Util para borrar un archivo
int removeinode(int numinode)
{
	int i;

	unsigned short temp[2048];

	for(i=0;i<10;i++)
		if(inode[numinode].blocks[i]!=0)
			unassignblock(inode[numinode].blocks[i]);

	if(inode[numinode].indirect!=0)
	{
		// Leer el bloque
		readblock(inode[numinode].indirect,(char *) temp);
		for(i=0;i<2048;i++)
			if(temp[i]!=0)
				unassignblock(temp[i]);
		unassignblock(inode[numinode].indirect);
		inode[numinode].indirect=0;
	}
	unassigninode(numinode);
	return(1);
}

// ************************************************************************
// Para el mapa de bits del área de nodos i
// ************************************************************************
int isinodefree(int inode)
{
	int offset=inode/8;
	int shift=inode%8;
	int result;
	int mapa_bits_nodos_i;

	// Checar si el sector de boot está en memoria
	if(!secboot_en_memoria)
	{
		// Si no está en memoria, cárgalo
		result=vdreadsector(0,0,0,1,1,(char *) &secboot);
		secboot_en_memoria=1;
	}
	mapa_bits_nodos_i=secboot.sec_res;  //Usamos la información del sector
de boot para
					//determinar en que sector inicia el
```

```
                                    // mapa de bits de nodos i

      // Está el mapa está en memoria
      if(!inodesmap_en_memoria)
      {
            // Si no está en memoria, hay que leerlo del disco
            result=vdreadseclog(mapa_bits_nodos_i,inodesmap);
            inodesmap_en_memoria=1;
      }


      if(inodesmap[offset] & (1<<shift))
            return(0);
      else
            return(1);
}

//Obtiene el siguiente i-nodo libre
int nextfreeinode()
{
      int i,j;
      int result;
      int mapa_bits_nodos_i;

      if(!secboot_en_memoria)
      {
            result=vdreadsector(0,0,0,1,1,(char *) &secboot);
            secboot_en_memoria=1;
      }
      mapa_bits_nodos_i= secboot.sec_res;

      if(!inodesmap_en_memoria)
      {
            result=vdreadseclog(mapa_bits_nodos_i,inodesmap);
            inodesmap_en_memoria=1;
      }

      // Recorrer byte por byte mientras sea 0xFF sigo recorriendo
      i=0;
      while(inodesmap[i]==0xFF && i<secboot.sec_mapa_bits_nodos_i)
            i++;

      if(i<secboot.sec_mapa_bits_nodos_i)
      {
            j=0;
            while(inodesmap[i] & (1<<j) && j<8)
                  j++;

            return(i*8+j);
      }
      else
      {
            return(-1);
      }
}
```

```
//Asigna el i-nodo
int assigninode(int inode)
{
      int offset=inode/8;
      int shift=inode%8;
      int result;
      int mapa_bits_nodos_i;

      if(!secboot_en_memoria)
      {
            result=vdreadsector(0,0,0,1,1,(char *) &secboot);
            secboot_en_memoria=1;
      }

      mapa_bits_nodos_i =     secboot.sec_res;

      if(!inodesmap_en_memoria)
      {
            result=vdreadseclog(mapa_bits_nodos_i,inodesmap);
            inodesmap_en_memoria=1;
      }

      inodesmap[offset]|=(1<<shift);
      vdwriteseclog(mapa_bits_nodos_i,(char*)inodesmap);
      return(1);
}

//Libera el i-nodo
int unassigninode(int inode)
{
      int offset=inode/8;
      int shift=inode%8;
      int result;
      char mask;
      int mapa_bits_nodos_i;

      if(!secboot_en_memoria)
      {
            result=vdreadsector(0,0,0,1,1,(char *) &secboot);
            secboot_en_memoria=1;
      }

      mapa_bits_nodos_i= secboot.sec_res;

      if(!inodesmap_en_memoria)
      {
            result=vdreadseclog(mapa_bits_nodos_i,inodesmap);
            inodesmap_en_memoria=1;
      }


      inodesmap[offset]&=(char) ~(1<<shift);
      vdwriteseclog(mapa_bits_nodos_i,inodesmap);
      return(1);
}
```

3.3.10 createvd.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "vdisk.h"

#define NSECS HEADS*SECTORS*CYLINDERS

int main(int argc,char **argv)
{
    int fp;
    int i;
    char nombre[20];
    char buffer[512];
    if(argc!=2)
    {
        fprintf(stderr,"Debe indicar el número de disco virtual a
crear\n\n");
        exit(2);
    }
    if(argv[1][0]<'0' || argv[1][0]>'3')
    {
        fprintf(stderr,"Los números válidos para la creación del disco son
entre 0 y 3\n\n");
        exit(2);
    }
    sprintf(nombre,"disco%c.vd\0",argv[1][0]);
    fp=open(nombre,0);
    if(fp!=-1)
    {
        fprintf(stderr,"Este disco virtual ya existe, para volver a crearlo
es necesario borrarlo\n\n");
        exit(2);
    }
    fp=creat(nombre,0644);
    memset(buffer,0,512);
    for(i=0;i<NSECS;i++)
        write(fp,buffer,512);
    close(fp);
}
```

3.3.11 datehandler.c

```
#include "datehandler.h"
```

```
//
************************************************************************
******
// Funciones de manejo de fecha y hora
//
************************************************************************
******

// Convierte la fecha que está en una estructura fecha a un entero de 32 bits
unsigned int datetoint(DATE date)
{
      unsigned int val=0;

      val=date.sdate.year-1970;
      val<<=4;
      val|=date.sdate.month;
      val<<=5;
      val|=date.sdate.day;
      val<<=5;
      val|=date.sdate.hour;
      val<<=6;
      val|=date.sdate.min;
      val<<=6;
      val|=date.sdate.sec;

      return(val);
}

// Extraer la fecha y hora que está empaquetada en un entero de 32 bits
// Almacena los resultados en una estructura
int inttodate(DATE *date,unsigned int val)
{
      date->sdate.sec=val&0x3F;
      val>>=6;
      date->sdate.min=val&0x3F;
      val>>=6;
      date->sdate.hour=val&0x1F;
      val>>=5;
      date->sdate.day=val&0x1F;
      val>>=5;
      date->sdate.month=val&0x0F;
      val>>=4;
      date->sdate.year=(val&0x3F) + 1970;
      return(1);
}

// Fecha y hora actual empaquetada en un entero de 32 bits
// Las funciones de creación y escritura de archivos usan esta función.
unsigned int currdatetimetoint()
{
      char i = 0;

      struct tm *tm_ptr;
      time_t the_time;
```

```
	DATE now;

	(void) time(&the_time);
	tm_ptr=gmtime(&the_time);

	now.sdate.year=tm_ptr->tm_year-70;
	now.sdate.month=tm_ptr->tm_mon+1;
	now.sdate.day=tm_ptr->tm_mday;
	now.sdate.hour=tm_ptr->tm_hour;
	now.sdate.min=tm_ptr->tm_min;
	now.sdate.sec=tm_ptr->tm_sec;

	return(datetoint(now));
}
```

3.3.12 dumpsec.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "vdisk.h"

#define LINESIZE 16
#define SECSIZE 512

int main(int argc,char *argv[])
{
	int drive;
	int ncyl,nhead,nsec;
	int fd;
	unsigned char buffer[SECSIZE];
	int offset;
	int i,j,r;
	unsigned char c;

	if(argc==5)
	{
		drive=atoi(argv[1]);
		ncyl=atoi(argv[2]);
		nhead=atoi(argv[3]);
		nsec=atoi(argv[4]);
		if(drive<0 || drive> 3 || ncyl>CYLINDERS || nhead > HEADS || nsec
> SECTORS || ncyl<0 || nhead<0 || nsec<1)
		{
			fprintf(stderr,"Posición invalida\n");
			exit(1);
		}
		printf("Desplegando de disco%d.vd Cil=%d, Sup=%d,
Sec=%d\n",drive,ncyl,nhead,nsec);

	}
```

```
	else
	{
		fprintf(stderr,"Error en los argumentos\n");
		exit(1);
	}

	if(vdreadsector(drive,nhead,ncyl,nsec,1,buffer)==-1)
	{
		fprintf(stderr,"Error al abrir disco virtual\n");
		exit(1);
	}

	for(i=0;i<SECSIZE/LINESIZE;i++)
	{
		printf("\n %3X -->",i*LINESIZE);
		for(j=0;j<LINESIZE;j++)
		{
			c=buffer[i*LINESIZE+j];
			printf("%2X ",c);
		}
		printf("  |  ");
		for(j=0;j<LINESIZE;j++)
		{
			c=buffer[i*LINESIZE+j]%256;
			if(c>0x1F && c<127)
				printf("%c",c);
			else
				printf(".");
		}
	}
	printf("\n");
}
```

3.3.13 filehandling.c

```
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>
#include "block_node_handler.h"
#include "logical_sector.h"
#include "tiposdatos.h"
#include "filehandling.h"


VDDIR dirs[2]={-1,-1};
vddirent current;



// fd = Descriptor del archivo
```

```
// Offset = Cuanto hay que mover el apuntador
// Whence = A partir de dond
//    0 = A partir del inicio.
//    1 = A partir de la posición actual del puntero.
//    2 = A partir del final del archivo.
//
// devuelve la posición donde queda el puntero del archivo después de
moverlo.
int vdseek(int fd, int offset, int whence)
{
      unsigned short oldblock,newblock;

      // Si no está abierto regresa error
      if(openfiles[fd].inuse==0)
            return(-1);

      oldblock= currpostoptr(fd);//(unsigned short*)currpostoptr(fd);
            // A partir de la posición actual del archivo, me regresa un
apuntador
            // a una dirección de memoria que contiene el bloque actual.
            //    Puede ser el inodo o un bloque de apuntadores.


      if(whence==0) // A partir del inicio
      {
            // Si el offset es negativo o excede el tamaño del archivo regresa
            // error
            if(offset<0 ||
               openfiles[fd].currpos+offset>inode[openfiles[fd].inode].size)
                  return(-1);
            openfiles[fd].currpos=offset;

      } else if(whence==1) // A partir de la posición actual
      {
            // Validar que posición actual - offset no vaya antes del
principio
            // también que posición actual + offset no exceda el tamaño del
archivo
            if(openfiles[fd].currpos+offset>inode[openfiles[fd].inode].size ||
               openfiles[fd].currpos+offset<0)
                  return(-1);
            openfiles[fd].currpos+=offset;

      } else if(whence==2) // A partir del final
      {
            // Validar que no sea positivo
            // Si es negativo, que el valor absoluto no exceda el
            // tamaño del archivo
            if(offset>inode[openfiles[fd].inode].size ||
               openfiles[fd].currpos-offset<0)
                  return(-1);
            openfiles[fd].currpos=inode[openfiles[fd].inode].size-offset;
      } else
            return(-1);
      // Una vez cambiada la posición, obtenemos el nuevo bloque actual
```

```
	newblock= currpostoptr(fd); //(unsigned short*)currpostoptr(fd);

	// Si después del movimiento hay un cambio de bloque
	if(newblock!=oldblock)
	{
		// Escribir el bloque que ya no estoy usando
		// Si quieren en la tabla de archivos abiertos poner una bandera
que
		// indica si el bloque fue modificado y si es así escríbelo
		// Una vez que se escribe, poner esa bandera en 0.
		writeblock(oldblock,openfiles[fd].buffer);

		// Cargar el nuevo bloque a memoria
		readblock(newblock,openfiles[fd].buffer);
		openfiles[fd].currbloqueenmemoria=newblock;
	}

	return(openfiles[fd].currpos);
}

// fd es el identificador en la tabla de archivos abiertos
// buffer un apuntador al area de memoria donde está lo que vamos a escribir
// bytes cuántos bytes vamos a escribir
int vdwrite(int fd, char *buffer, int bytes)
{
	int currblock;
	int currinode;
	int cont=0;
	int sector;
	int i;
	int result;
	unsigned short *currptr;

	// Si no está abierto, regresa error
	if(openfiles[fd].inuse == 0)
	{
		printf("El archivo no está abierto\n");
		return(-1);
	}
	else
	{
		//printf("El archivo sí está disponible y su nodo es: %d\n",
openfiles[fd].inode);
	}
	// Determinar cuál es el inodo de la tabla de nodos i del archivo que
	// vamos a escribir
	currinode=openfiles[fd].inode;

	// Ciclo para recorrer byte por byte del buffer a escribir
	while(cont<bytes)
	{
		// Obtener la dirección de donde está el bloque que corresponde
		// a la posición actual, si no hay bloque asignado a la posición
		// actual del archivo, regresamos error
		currptr = (unsigned short *)currpostoptr(fd);//(unsigned
short)currpostoptr(fd);
		if(currptr==NULL)
```

```
		{
			//printf("Aquí tronó\n");
			return(-1);
		}
		else
		{

		}

		// Obtener el número de bloque actual
		currblock=*currptr;

		//printf("Escribiendo el bloque numero: %d\n", currblock);

		// Si el bloque está en 0, aún no hay bloque para
		// escribir este carácter del archivo
		// hay que darle uno
		if(currblock==0)
		{
			// Busca un bloque libre en el mapa de bits
			// si no hay bloques libres, regresa -1
			currblock=nextfreeblock();
			if(currblock==-1)
				return(-1);
			else
				//printf("El bloque estaba vacío, el nuevo bloque es:
%d\n", currblock);
			// El bloque encontrado ponerlo en donde
			// apunta el apuntador al bloque actual
			*currptr=currblock; // aquí ya lo estoy escribiendo en el
nodo i.
			assignblock(currblock); // Poner ese bloque como asignado

			// Los cambios se hicieron en el nodo i en memoria, ahora
			// hay que escribirlos en el disco
			// Escribir el sector de la tabla de nodos i
			// En el disco

			// Recordar que los nodos i son de 64 bytes y caben 8 en
			// un sector
			sector=currinode;
			result=vdwriteseclog(inicio_nodos_i,(char*)&inode);
		}

		// Si el bloque de la posición actual no está en memoria
		// Lee el bloque al buffer del archivo
		if(openfiles[fd].currbloqueenmemoria!=currblock)
		{
			// Leer el bloque actual hacia el buffer que
			// está en la tabla de archivos abiertos
			readblock(currblock,openfiles[fd].buffer);
			openfiles[fd].currbloqueenmemoria=currblock;
		}

		// Copia el caracter del buffer que recibe la función
```

```
            // vdwrite, al buffer donde tenemos el bloque actual
            openfiles[fd].buffer[openfiles[fd].currpos%2048]=buffer[cont];

            // Incrementa posición actual del archivo
            openfiles[fd].currpos++;

            // Si la posición es mayor que el tamaño, modifica el tamaño
            if(openfiles[fd].currpos>inode[currinode].size)
                  inode[openfiles[fd].inode].size=openfiles[fd].currpos;

            // Incrementa el contador
            cont++;

            // Si se llena el buffer, escríbelo
            if(openfiles[fd].currpos%2048==0 ||
(inode[openfiles[fd].inode].size - openfiles[fd].currpos < 2048))
            {
                  writeblock(currblock,openfiles[fd].buffer);
                  //printf("Escribiendo el bloque: %d\n", currblock);
            }
      }
      return(cont);
}

// fd es el identificador en la tabla de archivos abiertos
// buffer un apuntador al area de memoria donde está lo que vamos a escribir
// bytes cuántos bytes vamos a escribir
int vdread(int fd, char *buffer, int bytes)
{
      int currblock;
      int currinode;
      int cont=0;
      int sector;
      int i;
      int result;
      unsigned short *currptr;

      // Si no está abierto, regresa error
      if(openfiles[fd].inuse==0)
      {
            printf("el archivo no está abierto\n");
            return(-1);
      }
      else
            //printf("El archivo sí está abierto\n");
      // Determinar cuál es el inodo de la tabla de nodos i del archivo que
      // vamos a escribir
      currinode=openfiles[fd].inode;

      // Ciclo para recorrer byte por byte del buffer a escribir
      while(cont<bytes)
      {
            //printf("atoradote\n");
            // Obtener la dirección de donde está el bloque que corresponde
            // a la posición actual, si no hay bloque asignado a la posición
            // actual del archivo, regresamos error
```

```
            currptr = (unsigned short *)currpostoptr(fd);  //(unsigned
short)currpostoptr(fd);
            if(currptr==NULL)
            {
                  //printf("Aquí trono \n");
                  return(-1);
            }

            // Obtener el número de bloque actual
            currblock=*currptr;

            // Si el bloque está en 0, aún no hay bloque para
            // escribir este carácter del archivo
            // hay que darle uno
            /*
            if(currblock==0)
            {
                  // Busca un bloque libre en el mapa de bits
                  // si no hay bloques libres, regresa -1
                  currblock=nextfreeblock();
                  if(currblock==-1)
                        return(-1);

                  // El bloque encontrado ponerlo en donde
                  // apunta el apuntador al bloque actual
                  *currptr=currblock; // aquí ya lo estoy escribiendo en el
nodo i.
                  assignblock(currblock); // Poner ese bloque como asignado

                  // Los cambios se hicieron en el nodo i en memoria, ahora
                  // hay que escribirlos en el disco
                  // Escribir el sector de la tabla de nodos i
                  // En el disco

                  // Recordar que los nodos i son de 64 bytes y caben 8 en
                  // un sector
                  sector=currinode;
                  result=vdreadseclog(inicio_nodos_i,(char *)&inode);
            }
*/

            // Si el bloque de la posición actual no está en memoria
            // Lee el bloque al buffer del archivo
            if(openfiles[fd].currbloqueenmemoria!=currblock)
            {
                  // Leer el bloque actual hacia el buffer que
                  // está en la tabla de archivos abiertos
                  readblock(currblock,openfiles[fd].buffer);
                  openfiles[fd].currbloqueenmemoria=currblock;
            }

            // Copia el caracter del buffer que recibe la función
            // vdwrite, al buffer donde tenemos el bloque actual
            buffer[cont] = openfiles[fd].buffer[openfiles[fd].currpos%2048];
```

```
            // Incrementa posición actual del archivo
            openfiles[fd].currpos++;

            /*// Si la posición es mayor que el tamaño, modifica el tamaño
            if(openfiles[fd].currpos>inode[currinode].size)
                  inode[openfiles[fd].inode].size=openfiles[fd].currpos;
            */
            // Incrementa el contador
            cont++;
      }
      return(cont);
}

int vdcreat(char* filename, unsigned short permissions)
{
      int inodo = 0;
      int fd = 0;

      if (searchinode(filename) == -1)
            inodo = setninode(nextfreeinode(), filename, permissions,
(int)getuid(), (int)getgid());
      else
            return -1;

      if (inodo >= 0)
      {
            if (assigninode(inodo) == 0)
                  return -1;
            else
            {
                  //printf("Abriendo el archivo\n");
                  fd = vdopen(filename,0);
                  openfiles[fd].currpos = 0;
                  //printf("El fd del archivo es: %d\n", fd);

                  return fd;
            }
      }
      else
      {
            printf("Ya no hay inodos disponibles\n");
      }
}

int vdopen(char* filename, unsigned char position)
{
       int inode = searchinode(filename);
       int i;

       for(i = 0; i < 8; i++)
       {
              if (openfiles[i].inuse == 0)
              {
                      openfiles[i].inuse = 1;
                      openfiles[i].inode = inode;
                               openfiles[i].currpos = vdseek(i,position,0);
                               openfiles[i].currbloqueenmemoria = 0;
```

```
                        break;
                }
        }

        if (i < 8)
        {
        }
        else
        {
                return (-1);
        }

        return (i);
}

int vdunlink(char * filename)
{
      int inodo = 0;

      inodo = searchinode(filename);

      unassigninode(inodo);

      removeinode(inodo);

      return (1);
}

int vdclose(int fd)
{
      char * clear = (char*)&openfiles[fd];
      int j = 0;

      if (openfiles[fd].inuse == 0)
            return (-1);
      else
      {
            for (j = 0; j < sizeof(OPENFILES); j++)
                  *clear++ = 0;
      }

      return (1);
}

/*************************************************************
/* Manejo de directorios
 *************************************************************/

VDDIR *vdopendir(char *path)
{
      int i=0;
      int result;

      if(!secboot_en_memoria)
      {
```

```
            result=vdreadsector(0,0,0,1,1,(char *) &secboot);
            secboot_en_memoria=1;
      }
      inicio_nodos_i=secboot.sec_res+secboot.sec_mapa_bits_nodos_i+secboot.sec
_mapa_bits_bloques;

      if(!nodos_i_en_memoria)
      {
            for(i=0;i<secboot.sec_tabla_nodos_i;i++)
                  result=vdreadseclog(inicio_nodos_i+i,(char*)&inode[i*8]);

            nodos_i_en_memoria=1;
      }

      if(strcmp(path,".")!=0)
            return(NULL);

      i=0;
      while(dirs[i]!=-1 && i<2)
            i++;

      if(i==2)
            return(NULL);

      dirs[i]=0;

      return(&dirs[i]);
}

vddirent *vdreaddir(VDDIR *dirdesc)
{
      int i;

      int result;
      if(!nodos_i_en_memoria)
      {
            result=vdreadseclog(inicio_nodos_i+i,(char*)&inode[i]);
            nodos_i_en_memoria=1;
      }

      // Mientras no haya nodo i, avanza
      while(isinodefree(*dirdesc) && *dirdesc < 8)
            (*dirdesc)++;


      // Apunta a donde está el nombre en el inodo
      current.d_name=inode[*dirdesc].name;

      (*dirdesc)++;

      if(*dirdesc>= 8)
            return(NULL);
      return( &current);
}

int vdclosedir(VDDIR *dirdesc)
{
```

```
	(*dirdesc)=-1;
}


int getsizeof(char * filename)
{
	return(inode[searchinode((char * )filename)].size);
}
```

3.3.14 logical_sector.c

```
#include "logical_sector.h"

int vdreadseclog(int sector, char * buffer)
{
	int secfis;
	int superficie;
	int cilindro;

	if (!secboot_en_memoria)
	{
		(void)vdreadsector(0,0,0,1,1,(char *) &secboot);
		secboot_en_memoria = 1;
	}

	if (sector > secboot.sec_log_unidad)
		return FAILED;

	secfis = (sector % secboot.secfis) + 1;
    superficie = (sector / secboot.secfis) % secboot.heads;
	cilindro = (sector /(secboot.secfis * secboot.heads));

	if (vdreadsector(0, superficie, cilindro, secfis, 1, buffer) > 0)
		return SUCCESS;
	else
		return FAILED;
}

int vdwriteseclog(int sector, char * buffer)
{
	int secfis;
	int superficie;
	int cilindro;

	if (!secboot_en_memoria)
	{
```

```
            (void)vdreadsector(0,0,0,1,1,(char *) &secboot);
            secboot_en_memoria = 1;
      }

      if (sector > secboot.sec_log_unidad)
            return FAILED;

      secfis = (sector % secboot.secfis) + 1;
    superficie = (sector / secboot.secfis) % secboot.heads;
      cilindro = (sector /(secboot.secfis * secboot.heads));

      if (vdwritesector(0, superficie, cilindro, secfis, 1, buffer) > 0)
            return SUCCESS;
      else
            return FAILED;
}
```

3.3.15 shell1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "tiposdatos.h"
#include "filehandling.h"


#define MAXLEN 80
#define BUFFERSIZE 512

#define TYPEBUFFERSIZE 2048

#define LINESIZE 16
#define SECSIZE 512

void locateend(char *cmd);
int executecmd(char *cmd);

void typev(char *arg1)
{
            int fd;
            int i,j,r;
            char buffer[512];
            unsigned char c;
            int size, ncars;
            int nBlocks = 0;


            printf("abriendo\n");
```

```
            fd = vdopen(arg1, 0);
            printf("file descriptor = %d\n",fd);

            size = getsizeof((char* )&arg1[0]);
            printf("size = %d\n",size);

            nBlocks = (size / BUFFERSIZE) + ((size%BUFFERSIZE > 0)? 1 : 0);

            do
            {
                  ncars = vdread(fd,(char*)&buffer , BUFFERSIZE);
                  for(i=0;i<SECSIZE/LINESIZE    ;i++)
                  {
                        for(j=0;j<LINESIZE;j++)
                        {
                              c=buffer[i*LINESIZE+j]%256;
                              if(c>0x1F && c<127)
                                    printf("%c",c);
                              else
                                    if(c == '\n')
                                          printf("\n");
                                    if(c == '\t')
                                          printf("\t");
                        }
                  }
            }while(nBlocks--);

            printf("\n");
            if(vdclose(fd))
            {
                  printf("Archivo cerrado exitosamente\n\n");
            }
}

void typeu(char *arg1)
{
            int sfile;
            char buffer[BUFFERSIZE];
            int ncars;
            unsigned char c;
            int i,j,r;

            sfile = open(arg1, 0);
            do
            {
                  memset((void *)&buffer,0,sizeof(buffer));
                  ncars = read(sfile,buffer, BUFFERSIZE);
                  for(i=0;i<SECSIZE/LINESIZE    ;i++)
                  {
                        for(j=0;j<LINESIZE;j++)
                        {
                              c=buffer[i*LINESIZE+j]%256;
                              if(c>0x1F && c<127)
                                    printf("%c",c);
                              else
```

```
                                    if(c == '\n')
                                          printf("\n");
                                    if(c == '\t')
                                          printf("\t");
                              }
                        }
                }while(ncars == BUFFERSIZE);

                printf("\n");
                if(close(sfile))
                {
                      printf("Archivo cerrado exitosamente\n\n");
                }
}

int main()
{
      char linea[MAXLEN];
      int result=1;
      while(result)
      {
            printf("vshell > ");
            fflush(stdout);
            read(0,linea,80);
            locateend(linea);
            result=executecmd(linea);
      }
}

void locateend(char *linea)
{
      // Localiza el fin de la cadena para poner el fin
      int i=0;
      while(i<MAXLEN && linea[i]!='\n')
            i++;
      linea[i]='\0';
}

int executecmd(char *linea)
{
      char *cmd;
      char *arg1;
      char *arg2;
      char *search=" ";

      // Separa el comando y los dos posibles argumentos
      cmd=strtok(linea," ");
      arg1=strtok(NULL," ");
      arg2=strtok(NULL," ");


      // No hay comando
      if(cmd==NULL)
            return(1);

      // comando "exit"
      if(strcmp(cmd,"exit")==0)
```

```
	return(0);

// comando "copy"
if(strcmp(cmd,"copy")==0)
{
	if(arg1==NULL && arg2==NULL)
	{
		fprintf(stderr,"Error en los argumentos\n");
		return(1);
	}
	if(!isinvd(arg1) && !isinvd(arg2))
		copyuu(&arg1[2],&arg2[2]);

	else if(!isinvd(arg1) && isinvd(arg2))
		copyuv(&arg1[2],arg2);

	else if(isinvd(arg1) && !isinvd(arg2))
		copyvu(arg1,&arg2[2]);

	else if(isinvd(arg1) && isinvd(arg2))
		copyvv(arg1,arg2);


}

// comando "cat"
if(strcmp(cmd,"cat")==0)
{
	if(isinvd(arg1))
		catv(arg1);
	else
		catu(&arg1[2]);
}

// comando dir
if(strcmp(cmd,"dir")==0)
{
	if(arg1==NULL)
		dirv();
	else if(!isinvd(arg1))
		diru(&arg1[2]);
}

// comando create
if(strcmp(cmd,"create") == 0)
{
	if(arg1 == NULL)
	{
		return(1);
	}
	//int vdcreat(char* filename, unsigned short permissions)
	if( vdcreat((char*)&arg1[0], 0777 ))
	{
		printf("Archivo creado exitosamente.\n");
	}
```

```
            else
            {
                  printf("El archivo no pudo crearse. Quizás ya existe uno con
el mismo nombre\n");
            }
      }

      // comando delete
      if(strcmp(cmd,"delete") == 0)
      {
            if(arg1 == NULL)
            {
                  printf("no hay argumentos\n");
                  return(1);
            }
            if(vdunlink((char*)&arg1[0]))
            {
                  printf("Archivo %s exitosamente borrado.\n", (char*)arg1);
            }
      }

      // comando type
      if(strcmp(cmd,"type") == 0)
      {
            if(arg1 == NULL)
            {
                  printf("no hay argumentos\n");
                  return(1);
            }

            if(!isinvd(arg1))
                  typeu((char*)&arg1[2]);
            else
                  typev(arg1);
      }

      return 1;
}
/* Regresa verdadero si el nombre del archivo no comienza con // y por lo
   tanto es un archivo que está en el disco virtual */

int isinvd(char *arg)
{
      if(strncmp(arg,"//",2)!=0)
            return(1);
      else
            return(0);
}


/* Copia un archivo del sistema de archivos de UNIX a un archivo destino
   en el mismo sistema de archivos de UNIX */

int copyuu(char *arg1,char *arg2)
{
      int sfile,dfile;
      char buffer[BUFFERSIZE];
```

```
      int ncars;

      sfile=open(arg1,0);
      dfile=creat(arg2,0640);
      do {
            ncars=read(sfile,buffer,BUFFERSIZE);
            write(dfile,buffer,ncars);
      } while(ncars==BUFFERSIZE);
      close(sfile);
      close(dfile);
      return(1);
}



/* Copia un archivo del sistema de archivos de UNIX a un archivo destino
   en el el disco virtual */

int copyuv(char *arg1,char *arg2)
{
      int sfile,dfile;
      char buffer[BUFFERSIZE];
      int ncars;

      sfile=open(arg1,0);
      dfile=vdcreat(arg2,0777);
      if (dfile == -1)
      {
            printf("Ya existe un archivo con ese nombre\n");
            return 1;
      }

      do {
            ncars=read(sfile,buffer,BUFFERSIZE);
            if (vdwrite(dfile,buffer,ncars) == -1)
                  printf("El archivo no se copió correctamente.\n");
      } while(ncars==BUFFERSIZE);
      close(sfile);
      vdclose(dfile);
      return(1);
}


/* Copia un archivo del disco virtual a un archivo destino
   en el sistema de archivos de UNIX */

int copyvu(char *arg1,char *arg2)
{
      int sfile,dfile;
      char buffer[BUFFERSIZE];
      int ncars;
      int size, nBlocks;


      sfile=vdopen(arg1,0);
```

```
	size = getsizeof(arg1);
	nBlocks = size/BUFFERSIZE;

	dfile=creat(arg2,0640);
	do {
		ncars=vdread(sfile,buffer,BUFFERSIZE);
		write(dfile,buffer,ncars);
	} while(nBlocks--);
	vdclose(sfile);
	close(dfile);
	return(1);
}



/* Copia un archivo del disco virtual a un archivo destino
   en el mismo disco virtual */

int copyvv(char *arg1,char *arg2)
{
	int sfile,dfile;
	char buffer[BUFFERSIZE];
	int ncars;
	int size;
	int nBlocks;
	sfile=vdopen(arg1,0);

	size = getsizeof(arg1);
	nBlocks = size/BUFFERSIZE;

	dfile=vdcreat(arg2,0777);
	if (dfile == -1)
	{
		printf("Ya existe un archivo con ese nombre\n");
		return 1;
	}
	do {
		ncars=vdread(sfile,buffer,BUFFERSIZE);
		vdwrite(dfile,buffer,ncars);
	} while(nBlocks--);
	vdclose(sfile);
	vdclose(dfile);
	return(1);
}


/* Despliega un archivo del disco virtual a pantalla */

int catv(char *arg1)
{
	int sfile,dfile;
	char buffer[BUFFERSIZE];
	int ncars;

	sfile=vdopen(arg1,0);
	do {
		ncars=vdread(sfile,buffer,BUFFERSIZE);
```

```
            write(1,buffer,ncars);  // Escribe en el archivo de salida
estandard
      } while(ncars==BUFFERSIZE);
      vdclose(sfile);
      return(1);
}


/* Despliega un archivo del sistema de archivos de UNIX a pantalla */

int catu(char *arg1)
{
      int sfile,dfile;
      char buffer[BUFFERSIZE];
      int ncars;

      sfile=open(arg1,0);
      do {
            ncars=read(sfile,buffer,BUFFERSIZE);
            write(1,buffer,ncars);  // Escribe en el archivo de salida
estandard
      } while(ncars==BUFFERSIZE);
      close(sfile);
      return(1);
}


/* Muestra el directorio en el sistema de archivosd de UNIX */

int diru(char *arg1)
{
      DIR *dd;
      struct dirent *entry;

      if(arg1[0]=='\0')
            strcpy(arg1,".");

      printf("Directorio %s\n",arg1);

      dd=opendir(arg1);
      if(dd==NULL)
      {
            fprintf(stderr,"Error al abrir directorio\n");
            return(-1);
      }

      while((entry=readdir(dd))!=NULL)
            printf("%s\n",entry->d_name);

      closedir(dd);
}

/* Muestra el directorio en el sistema de archivos en el disco virtual */

int dirv()
```

```
{
	VDDIR *dd;
	struct vddirent *entry;

	printf("Directorio del disco virtual\n");

	dd=vdopendir(".");
	if(dd==NULL)
	{
		fprintf(stderr,"Error al abrir directorio\n");
		return(-1);
	}

	while((entry=vdreaddir(dd))!=NULL)
		printf("%s\n",entry->d_name);

	vdclosedir(dd);
}
```

3.3.16 tiposdatos.c

```
#include "tiposdatos.h"

unsigned char inicio_nodos_i = 0;
OPENFILES openfiles[8];
SECBOOT secboot;
INODE inode[8];
unsigned char secboot_en_memoria = 0;
unsigned char inodesmap_en_memoria = 0;
unsigned char nodos_i_en_memoria = 0;
unsigned char inodesmap[2];
int mapa_bits_bloques = 0;
unsigned char blocksmap_en_memoria = 0;
unsigned short inicio_area_datos = 0;
unsigned char blocksmap[16384];
```

3.3.17 vdformat.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "vdisk.h"
#include "tiposdatos.h"


int main(int argc, char ** argv)
{
	int i;
```

```
      char * nodo;
      char buffer[512];

    SECBOOT secboot;

      for(i = 0; i < sizeof(buffer); i++)
      {
            buffer[i] = 0;
      }

      printf("Disco numero %c\n", argv[1][0]);
    secboot.jump[0] = 0;
      secboot.jump[1] = 0;
      secboot.jump[2] = 0;
      secboot.jump[3] = 0;
    secboot.nombre_disco[0] = argv[1][0];
    secboot.nombre_disco[1] = 0;
    secboot.nombre_disco[2] = 0;
    secboot.nombre_disco[3] = 0;
    secboot.nombre_disco[4] = 0;
    secboot.nombre_disco[5] = 0;
    secboot.nombre_disco[6] = 0;
    secboot.nombre_disco[7] = 0;
    secboot.sec_res = 1;                              // 1 sector de arranque
    secboot.sec_mapa_bits_nodos_i = 1;          // 1 sector
      secboot.sec_mapa_bits_bloques = 4;  // 4 sectores
      secboot.paddingbyte = 0;
    secboot.sec_tabla_nodos_i = 1;        // 1 sector
    secboot.sec_log_unidad = 54400;             // 54400 sectores
    secboot.sec_x_bloque = 4;                   // 4 sectores por bloque
    secboot.heads = 20;                               // 20 superficies
    secboot.cyls = 160;                               // 160 cilindros
    secboot.secfis = 17;                              // 17 sectores

      for (i = 0; i < sizeof(secboot.restante); i++)
            secboot.restante[i] = 0;

    if (vdwritesector((int)(argv[1][0] - '0'), 0, 0, 1, 1, (char *) &secboot)
> 0)
      {
            for (i = 1; i < 7; i++)
            {
                  vdwritesector((int)(argv[1][0] - '0'), 0, 0, 1 + i, 1,
buffer);
            }
            printf("El disco %c se formateo exitosamente\n",
secboot.nombre_disco[0]);
      }
      else
            printf("El disco no se pudo formatear. ¿Estás seguro de que
existe?\n");


}
```

3.3.18 vdisk.c

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include "vdisk.h"

int currentcyl[4]={0,0,0,0};
int currentsec[4]={0,0,0,0};

int vdwritesector(int drive, int head, int cylinder, int sector, int nsecs,
char *buffer)
{
    char filename[20];
    int fp;
    int timecyl,timesec;
    int sl,offset;
    sprintf(filename,"disco%c.vd",(char) drive+'0');
    fp=open(filename,O_WRONLY);
    if(fp==-1)
        return(-1);

    // Valida parámetros
    if(drive<0 || drive>3)
        return(-1);

    if(head<0 || head>=HEADS)
        return(-1);

    if(cylinder<0 || cylinder>=CYLINDERS)
        return(-1);

    if(sector<1 || sector>SECTORS)
        return(-1);

    if(sector+nsecs-1>SECTORS)
        return(-1);

    // Hace el retardo
    /*timesec=sector-currentsec[drive];
    if(timesec<0)
        timesec+=SECTORS;
    usleep(timesec*1000);
    currentsec[drive]=sector;

    timecyl=abs(currentcyl[drive]-cylinder);
    usleep(timecyl*1000);
    currentcyl[drive]=cylinder;
*/
    // Calcula la posición en el archivo
    sl=cylinder*SECTORS*HEADS+head*SECTORS+(sector-1);
    offset=sl*512;
    lseek(fp,offset,SEEK_SET);
    write(fp,buffer,512*nsecs);
```

```
    close(fp);
    return(nsecs);
}

int vdreadsector(int drive, int head, int cylinder, int sector, int nsecs,
char *buffer)
{
    char filename[20];
    int fp;
    int timecyl,timesec;
    int sl,offset;
    sprintf(filename,"disco%c.vd",(char) drive+'0');
    fp=open(filename,O_RDONLY);
    if(fp==-1)
        return(-1);

    // Valida parámetros
    if(drive<0 || drive>3)
        return(-1);

    if(head<0 || head>=HEADS)
        return(-1);

    if(cylinder<0 || cylinder>=CYLINDERS)
        return(-1);

    if(sector<1 || sector>SECTORS)
        return(-1);

    if(sector+nsecs-1>SECTORS)
        return(-1);

    // Hace el retardo
   /* timesec=sector-currentsec[drive];
    if(timesec<0)
        timesec+=SECTORS;
    usleep(timesec*1000);
    currentsec[drive]=sector;

    timecyl=abs(currentcyl[drive]-cylinder);
    usleep(timecyl*1000);
    currentcyl[drive]=cylinder;
     */
    // Calcula la posición en el archivo
    sl=cylinder*SECTORS*HEADS+head*SECTORS+(sector-1);
    offset=sl*512;
    lseek(fp,offset,SEEK SET);
    read(fp,buffer,512*nsecs);
    close(fp);
    return(nsecs);
}
```