TRABAJO
SISTEMAS ELECTRÓNICOS DIGITALES MICROS

SISMÓMETRO

Grupo: A404
Realizado por:
Marcos Espinoza Pino (56351)
Mario García López (56379)
Manuel Hidalgo (56416)

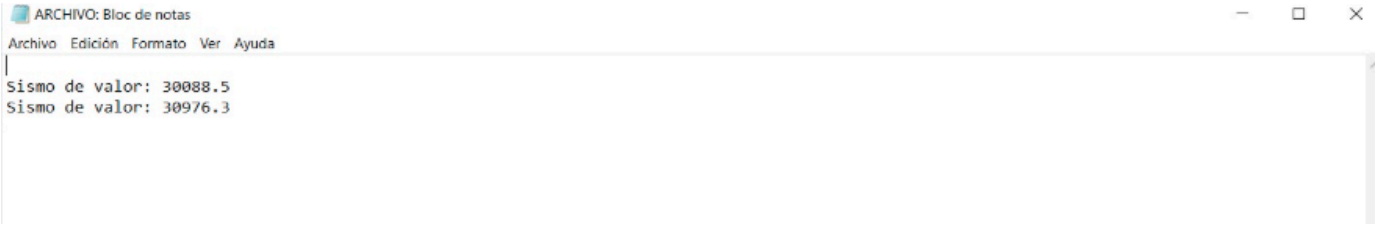
Índice

Introducción

En este trabajo se ha desarrollado un sistema sismográfico basado en un microcontrolador STM32,

Capaz de detectar eventos sísmicos mediante un acelerómetro y registrar los valores correspondientes en un archivo de texto, dicho archivo se encuentra en un pendrive que colocaremos en nuestra placa. Cuando se detecte un evento que supere el umbral establecido, se registrará el valor en nuestro archivo. Este proyecto tiene como objetivo simular un sistema de monitoreo de aceleración en tiempo real, enfocándose en los conceptos de detección, registro y análisis de datos.

Una vez se registre un evento sísmico, en nuestro archivo aparecerá lo siguiente:



Se ha utilizado el sensor LSM303DLHC para medir la aceleración en los tres ejes (x,y,z) en un solo dispositivo, configurando un umbral que permite determinar cuándo ocurre un evento sísmico significativo. Además, se implementa un sistema de registro en una unidad USB utilizando el sistema de archivos FATFS.

Las herramientas empleadas incluyen STM32CubeIDE, HAL (Hardware Abstraction Layer) y bibliotecas adicionales para la lectura del sensor.

Se ha desarrollado una maqueta de un volcán ya que se considera que una de las posibles utilidades del trabajo es detectar catástrofes naturales como tsunamis, erupciones volcánicas o terremotos…

1.1 Descripción del Código

El proyecto desarrollado utiliza un microcontrolador STM32F411E-DISCO para implementar un sistema de detección de eventos sísmicos. A continuación, se describen los principales bloques de funcionamiento del código, organizados en diferentes etapas:

Estructura General:

El código se organiza en diferentes secciones para la inicialización del hardware y la ejecución de las tareas principales del sistema. Esto incluye la configuración de periféricos clave, la interacción con sensores y dispositivos externos, y la gestión de los eventos sísmicos detectados.

- Inicialización del hardware: Configuración de periféricos como GPIO, I2C, SPI y USB.

Configura periféricos esenciales como GPIO para el control de LEDs, I2C para la comunicación con el acelerómetro LSM303DLHC, SPI para posibles ampliaciones futuras y USB OTG en modo Host para conectar dispositivos de almacenamiento USB y registrar datos detectados cuando se supere el umbral.

- Lectura de datos del acelerómetro: Obtención de valores de aceleración en tiempo real.

El acelerómetro se configura para operar a 100 Hz, con un rango de ±2g. Los datos de

aceleración de los ejes X, Y y Z se obtienen mediante I2C, y se calcula la magnitud total de aceleración ajustando el eje Z para compensar la gravedad.

- Detección de eventos sísmicos: Verificación de si la aceleración supera un umbral definido.

La detección se basa en comparar la magnitud de aceleración calculada con un umbral predefinido (2500 mg). Si se supera el umbral, se registra la aceleración máxima durante el evento y, si dura más de 10 segundos, se almacena en un archivo .

- Registro de datos: Escritura de los valores detectados en un archivo de texto.

Los valores de aceleración máxima detectados se almacenan en un archivo de texto en un dispositivo USB utilizando FATFS, lo que permite su análisis posterior en sistemas operativos como Windows o Linux.

- Indicadores LED: Activación de LEDs según la detección de eventos.

Se utilizan LEDs para indicar eventos sísmicos y el estado de conexión USB. Un LED se enciende cuando se detecta un evento sísmico, mientras que otro indica si el dispositivo USB está correctamente conectado y montado.

1.2 Librerías

`"fatfs.h":` Maneja el sistema de archivos de una memoria USB.

`"usb_host.h":` Para manejar la interfaz USB.

`<lsm303dlhc.h> :` Para interactuar con el acelerómetro

`"math.h":` Necesaria para cálculos matemáticos

1.3 Constantes

`#define SEISMIC_THRESHOLD 250` //Umbral SEISMIC_THRESHOLD que determinará si la aceleración detectada es suficientemente alta para ser considerada como un evento sísmico.

```
#define SEISMIC_LED_PIN GPIO_PIN_15
#define SEISMIC_LED_PORT GPIOD
#define led_test GPIO_PIN_12
#define port_test GPIOD
```

1.4 Variables globales

`FATFS fs;` Maneja el sistema de archivos en la memoria USB

`int seismicEventDetected = 0;` Indica si hay evento sísmico o no.

`int16_t acc_data[3];` Almacena los datos de aceleración en x, y, z.

`float accelerationgravedad = 16000;` Factor de corrección de la aceleración gravitatoria.

`float accelerationMagnitude;` Magnitud total de la aceleración.

`float maxAccelerationMagnitude;` Guarda la aceleración máxima.

1. 5 Módulos y Funcionalidades

1. **5. 1 Configuración del Hardware.**

- Los pines GPIO se configuran para controlar LEDs, leer datos de sensores y manejar señales de entrada/salida.
- Ejemplo de configuración:

```
static void MX_GPIO_Init(void)
{
GPIO_InitTypeDef GPIO_InitStruct = {0};
```

/* Habilitación del reloj para los puertos GPIO utilizados , en este caso A,B,C,D,E y H, necesario porque en sistemas embebidos están apagados por defecto*/

```
__HAL_RCC_GPIOE_CLK_ENABLE();
__HAL_RCC_GPIOC_CLK_ENABLE();
__HAL_RCC_GPIOH_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();
__HAL_RCC_GPIOD_CLK_ENABLE();
```

/*Configuramos las salidas */

```
HAL_GPIO_WritePin(CS_I2C_SPI_GPIO_Port, CS_I2C_SPI_Pin, GPIO_PIN_RESET);


HAL_GPIO_WritePin(OTG_FS_PowerSwitchOn_GPIO_Port,OTG_FS_PowerSwitchOn_Pin,
GPIO_PIN_SET);


HAL_GPIO_WritePin(GPIOD, LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                        |Audio_RST_Pin, GPIO_PIN_RESET);



/*Configure GPIO pin : DATA_Ready_Pin */
GPIO_InitStruct.Pin = DATA_Ready_Pin; //Pin-DATA_Ready_Pin
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;//Configuramos como entrada
GPIO_InitStruct.Pull = GPIO_NOPULL;//No usa resistencia interna
HAL_GPIO_Init(DATA_Ready_GPIO_Port, &GPIO_InitStruct);//Inicializamos el pin



/*Configure GPIO pin : CS_I2C_SPI_Pin */
GPIO_InitStruct.Pin = CS_I2C_SPI_Pin; //Seleccionamos el pin `CS_I2C_SPI_Pin`
  GPIO_InitStruct.Mode  =  GPIO_MODE_OUTPUT_PP;  //Configuramos el pin como salida push-pull.
GPIO_InitStruct.Pull = GPIO_NOPULL;//No usa resistencia interna
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;// Establece una velocidad de conmutación baja.
```

```
HAL_GPIO_Init(CS_I2C_SPI_GPIO_Port, &GPIO_InitStruct);//Inicializamos el pin


/*Configure GPIO pins : INT1_Pin INT2_Pin MEMS_INT2_Pin */
GPIO_InitStruct.Pin = INT1_Pin|INT2_Pin|MEMS_INT2_Pin;//Selecciona los pines.
GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;// Configuramos para generar un evento en flanco de subida.
GPIO_InitStruct.Pull = GPIO_NOPULL;// No se usa resistencia pull-up ni pull-down.
HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);// Inicializamos los pines en el puerto `GPIOE`.


/*Configure GPIO pin : OTG_FS_PowerSwitchOn_Pin */
GPIO_InitStruct.Pin = OTG_FS_PowerSwitchOn_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;// Configuramos el pin como salida push-pull.
GPIO_InitStruct.Pull = GPIO_NOPULL;// No se usa resistencia pull-up ni pull-down.
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;// Establece una velocidad de conmutación baja.
HAL_GPIO_Init(OTG_FS_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);
/*Configure GPIO pin : PA0 */
GPIO_InitStruct.Pin = GPIO_PIN_0;// Selecciona el pin 0 del puerto A
GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;// Configuramos para generar un evento en flanco de subida.
GPIO_InitStruct.Pull = GPIO_NOPULL;// No se usa resistencia pull-up ni pull-down.
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);// Inicializamos los pines en el puerto `GPIOA`
/*Configure GPIO pins : LD4_Pin LD3_Pin LD5_Pin LD6_Pin
                         Audio_RST_Pin */
GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                      |Audio_RST_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;/ Configuramos los pines como salida push-pull.
GPIO_InitStruct.Pull = GPIO_NOPULL;// No se usa resistencia pull-up ni pull-down.
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;// Establece una velocidad de conmutación baja.
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);//Inicializamos
/*Configure GPIO pin : OTG_FS_OverCurrent_Pin */
GPIO_InitStruct.Pin = OTG_FS_OverCurrent_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(OTG_FS_OverCurrent_GPIO_Port, &GPIO_InitStruct);
```

}

- **Propósito en el proyecto:**
  - LEDs se utilizan para indicar eventos sísmicos.
  - Pines configurados como entrada podrían manejar sensores o interrupciones externas.

1. **5. 2 Configuración de I2C (Inter-Integrated Circuit):**

- El protocolo I2C se utiliza para comunicarse con el acelerómetro digital (LSM303DLHC) y otros posibles dispositivos que compartan el bus I2C.
- Ejemplo de configuración**:** Se explicará I2C1 pero también se tienen I2S2 y I2S3.

```
static void MX_I2C1_Init(void)
{
 hi2c1.Instance = I2C1;
 hi2c1.Init.ClockSpeed = 100000;
 hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
 hi2c1.Init.OwnAddress1 = 0;
 hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
 hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
 hi2c1.Init.OwnAddress2 = 0;
 hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
 hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
 if (HAL_I2C_Init(&hi2c1) != HAL_OK)
 {
   Error_Handler();
 }
}
```

- **Propósito en el proyecto:**
  - Leer datos de aceleración de los ejes X, Y, Z del acelerómetro.
  - Configurar parámetros del sensor, como frecuencia de muestreo y escala.

1. **5 . 3 Configuración de SPI (Serial Peripheral Interface):**

- El protocolo SPI se usa para dispositivos que requieren transferencia de datos rápida y sincrónica.

  Ejemplo de configuración:

```
static void MX_SPI1_Init(void)
{
 hspi1.Instance = SPI1;
 hspi1.Init.Mode = SPI_MODE_MASTER;
 hspi1.Init.Direction = SPI_DIRECTION_2LINES;
 hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
 hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
```

```
 hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
 hspi1.Init.NSS = SPI_NSS_SOFT;
 hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
 hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
 hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
 hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
 hspi1.Init.CRCPolynomial = 10;
 if (HAL_SPI_Init(&hspi1) != HAL_OK)
 {
   Error_Handler();
 }
}
```

- **Propósito en el proyecto:**
  - SPI está configurado para permitir la integración de otros sensores o periféricos en el futuro.

1 . 5 . 4 Configuración de USB (Host):

Explicaremos a continuación las funcionalidades relacionadas con el USB.

- Se configura el USB en modo Host para permitir la conexión de dispositivos de almacenamiento como pendrives.
- Ejemplo de configuración**:**

```
void MX_USB_HOST_Init();
{
  MX_USB_HOST_Process();
}
```

- El proyecto utiliza FATFS para leer y escribir datos en un dispositivo de almacenamiento USB:
  - Montar el sistema de archivos FAT en

```
void onHostUserClassActiveCallback(void)
 f_mount(&fs, "", 0);
HAL_GPIO_WritePin(port_test, led_test, GPIO_PIN_SET); //Notifica USB enchufado
```

  - Desmontar el sistema en `void onHostUserDisconnectionCallback(void):`

```
 f_mount(NULL, "", 0);
 HAL_GPIO_WritePin(port_test, led_test, GPIO_PIN_RESET);
```

  - Guardar datos sísmicos en un archivo:

```
 saveSismicData();
```

- **Propósito en el proyecto:**
  - Guardar los datos de aceleración máxima detectada en un archivo para análisis posterior.

1.6 Lectura del Acelerómetro

El proyecto utiliza el acelerómetro digital LSM303DLHC para obtener lecturas de aceleración en los ejes X, Y y Z en tiempo real. Este acelerómetro se comunica con el microcontrolador STM32 a través del protocolo I2C visto en las clases de teoría.

Proceso de lectura de datos:

1. **Inicialización del acelerómetro:** Antes de poder leer datos, el acelerómetro debe ser configurado, a una frecuencia de 100 Hz. Esto se hace en la función:

```
LSM303DLHC_AccInit(
   (LSM303DLHC_ODR_100_HZ | LSM303DLHC_AXES_ENABLE) +
    256 * (LSM303DLHC_FULLSCALE_2G | LSM303DLHC_BlockUpdate_Single)
    );
```

   - **Frecuencia de muestreo:** Se establece en 100 Hz (LSM303DLHC_ODR_100_HZ).
   - **Ejes habilitados:** X, Y, Z (LSM303DLHC_AXES_ENABLE).
   - **Rango de medida:** ±2g (LSM303DLHC_FULLSCALE_2G).
   - **Bloqueo de actualización:** Configurado para evitar lecturas inconsistentes

2. **Lectura de datos de aceleración:** Durante el funcionamiento principal del programa, los valores de aceleración se obtienen mediante la función:

```
LSM303DLHC_AccReadXYZ(acc_data);
```

   - **acc_data[3]:** Es un vector de tres elementos que almacena las lecturas de los tres ejes X, Y y Z.
   - **Datos proporcionados:** Los valores son proporcionales a la aceleración en miligravedad (mg), donde cada unidad representa 1 mg.

3. **Cálculo de la magnitud de la aceleración:** Los valores de aceleración obtenidos en los tres ejes se combinan para calcular la magnitud total del vector de aceleración, y almacenarla en la variable accelerationMagnitude:

```
accelerationMagnitude = sqrt((float)(acc_data[0] * acc_data[0] + acc_data[1] *
acc_data[1] + (acc_data[2] - accelerationgravedad) * (acc_data[2] -
accelerationgravedad)));
```

   Empleamos la librería math. h por sqrt.

   - **acc_data[0] (x), acc_data[1](y), acc_data[2](z):** Representan las lecturas de los ejes X, Y y Z.
   - **Corrección por gravedad:** Se resta la aceleración gravitacional estándar (16000 mg en este caso) del eje Z para obtener lecturas precisas.
   - **Resultado:** La magnitud calculada representa el movimiento total en 3D.

4. **Uso de los datos de aceleración:**
   - **Comparación con un umbral:** Se verifica si la magnitud de la aceleración excede un valor predeterminado (SEISMIC_THRESHOLD = 2500 mg) para detectar un evento sísmico:

// Detectar si se supera el umbral, en caso de superarse, notificaremos encendiendo un LED.

```
HAL_GPIO_WritePin(SEISMIC_LED_PORT,    SEISMIC_LED_PIN,    accelerationMagnitude    >
SEISMIC_THRESHOLD ? GPIO_PIN_SET : GPIO_PIN_RESET
```

El código de arriba equivale al siguiente if, pero de forma más compacta.

```
    if (accelerationMagnitude > SEISMIC_THRESHOLD) {
            HAL_GPIO_WritePin(SEISMIC_LED_PORT, SEISMIC_LED_PIN, GPIO_PIN_SET);
        }
   else {
            HAL_GPIO_WritePin(SEISMIC_LED_PORT, SEISMIC_LED_PIN,GPIO_PIN_RESET);
        }
```

- **Almacenamiento de datos:** Si se detecta un evento, los valores máximos de aceleración se guardan en un archivo USB mediante FATFS para análisis posterior.

Resumen del flujo de lectura

1. Configuración inicial del acelerómetro.
2. Lectura periódica de los valores de aceleración (X, Y, Z).
3. Cálculo de la magnitud de aceleración total.
4. Verificación de si el valor excede un umbral para detectar eventos sísmicos.
5. Almacenamiento de los datos relevantes para análisis futuro.

Propósito en el proyecto

La lectura de datos del acelerómetro permite:

1. **Detección en tiempo real:** Identificar eventos sísmicos mediante la magnitud de aceleración.
2. **Indicadores visuales:** Activar un LED como señal de detección para así poder informar al usuario.
3. **Almacenamiento y análisis:** Guardar datos relevantes para análisis posterior, contribuyendo a la funcionalidad del sistema. Un valor muy alto detectado por nuestro micro, informaría de un evento sísmico importante, pudiendo informar a la población de la catástrofe natural que va a ocurrir.

1.7 Registro de Eventos Sísmicos en nuestro USB

Registro del evento

Si se detecta un evento sísmico:

- Se registra el momento del inicio del evento con HAL_GetTick() para medir la duración del evento:

```
startTime = HAL_GetTick(); //Guardamos en startime, el inicio del evento.
seismicEventDetected = 1; //Ponemos a 1 , evento sísmico detectado.
maxAccelerationMagnitude = accelerationMagnitude;
```

- Durante el evento y hasta que se cumpla un tiempo de 10 segundos el cuál está almacenado en la variable duration, el sistema actualiza la magnitud máxima de aceleración detectada.

```
if (accelerationMagnitude > valormax) {
                valormax = accelerationMagnitude;
        }
```

- Cuando se excedan los 10 segundos, reiniciaremos la variable de detección de evento sísmico y guardaremos en nuestro archivo el valor máximo de aceleración que se encuentra en maxAccelerationMagnitude:

```
if (HAL_GetTick() - startTime > 10000)
```
//Comparamos el tiempo actual que nos da Hal_GetTick() con el valor que almacenamos al principio del evento, si da más de 10000 ms guardamos el valor de aceleración correspondiente.

```
      {
      seismicEventDetected = 0;
      saveSismicData(maxAccelerationMagnitude);
      }
```

Propósito en el proyecto

El mecanismo de detección de eventos sísmicos permite:

- **Identificación en tiempo real:** Detectar movimientos que superen un umbral crítico.
- **Registro de datos:** Almacenar información para análisis posterior.
- **Indicación visual:** Proveer retroalimentación inmediata mediante LED.

1.8 Registro de Datos

El proyecto implementa un sistema para registrar los valores de aceleración máxima detectados durante un evento sísmico. Estos datos se escriben en un archivo de texto en un dispositivo de almacenamiento USB utilizando el sistema de archivos FATFS.

Proceso de registro de datos

1. Montaje del sistema de archivos FAT

Antes de escribir en el archivo, el dispositivo de almacenamiento USB debe ser montado.

Se informará al usuario de USB conectado con un LED.

Esto se realiza en las funciones de inicialización y callbacks de conexión USB:

```
void onHostUserClassActiveCallback(void)//conexion USB
{
 f_mount(&fs, "", 0); // Montar el sistema de archivos en la raíz
 HAL_GPIO_WritePin(port_test, led_test, GPIO_PIN_SET); // Indicar conexión con un LED
}
void onHostUserDisconnectionCallback(void)//desconexion
{
 f_mount(NULL, "", 0); // Desmontar el sistema de archivos
 HAL_GPIO_WritePin(port_test, led_test, GPIO_PIN_RESET);  // Informar desconexión
}
```

2. Función para guardar los datos sísmicos

Cuando se detecta un evento sísmico, la magnitud máxima de aceleración se registra en un archivo de texto utilizando la función **saveSismicData()**.

```
void saveSismicData() {
    FIL archivo;
    FRESULT resultado;
    resultado = f_open(&archivo, "archivo.txt", FA_WRITE,FA_ALWAYS); //abrimos el archivo para escribir, en caso de que no exista lo crea.
if (resultado == FR_OK) { //Si conseguimos abrirlo
    f_lseek(&archivo, f_size(&archivo)); //Nos colocamos al final del archivo
    const char* texto = "\nSismo de valor: "; //Frase que se imprimirá en en el archivo junto con el valor.
    UINT bytesEscritos;
    f_write(&archivo, texto, strlen(texto), &bytesEscritos); //Escribe texto en el archivo.
    char buffer[20];
    snprintf(buffer, sizeof(buffer), "%.1f", maxAccelerationMagnitude);//convertimos la magnitud máxima a texto.
    f_write(&archivo, buffer, strlen(buffer), &bytesEscritos);//Escribimos magnitud.
    f_close(&archivo);
    }
```

Detalles del funcionamiento de la escritura:

1. **Apertura del archivo:**
   - El archivo "archivo.txt" se abre en modo escritura utilizando f_open.
   - Si el archivo no existe, se crea automáticamente debido al uso de la bandera FA_OPEN_ALWAYS
2. **Apertura del archivo:**
   - El archivo "archivo.txt" se abre en modo escritura utilizando f_open.
   - Si el archivo no existe, se crea automáticamente debido al uso de la bandera FA_OPEN_ALWAYS.
3. **Posicionamiento del puntero al final del archivo:**
   - El puntero se mueve al final del archivo con f_lseek para que los nuevos datos se agreguen al final sin sobrescribir los existentes.
4. **Escritura de datos:**
   - Se escribe un texto descriptivo para identificar el evento:

     const char* texto = "\nSismo de valor: ";
   - Luego, se convierte la magnitud máxima de aceleración (tipo float) en una cadena de caracteres usando snprintf:

     snprintf(buffer, sizeof(buffer), "%.1f", maxAccelerationMagnitude);
   - Ambas cadenas (texto y valor) se escriben en el archivo utilizando f_write.
5. **Cierre del archivo:**

- Finalmente, el archivo se cierra con f_close para garantizar que los datos se guarden correctamente.

Fragmento del flujo de registro

El registro se realiza durante un evento sísmico si este supera el umbral y dura más de 10 segundos como hemos mencionado antes.

Ejemplo de archivo generado

El archivo "archivo.txt" contendrá una lista de eventos sísmicos registrados con sus valores máximos de aceleración. Ejemplo:

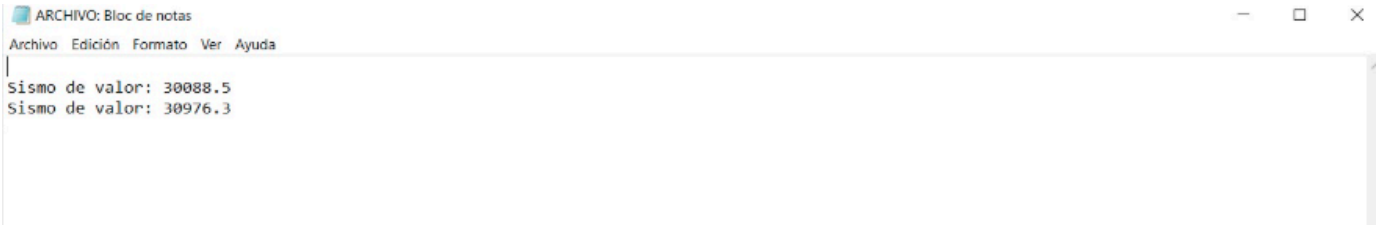
Sismo de valor: 2567.8

Sismo de valor: 3124.5

Sismo de valor: 2987.2

Siendo 2567.8, 3124.5, 2987.2 los valores máximos

Tras haber realizado alguna prueba con el micro y se han obtenido los siguientes resultados:



Como se puede ver se han detectado dos eventos sísmicos, que tras pasar los 10 segundos se ha registrado el valor máximo de cada evento acompañados de la frase: Sismo de valor.

Estos eventos sísmicos que ha detectado han sido golpes que se han dado a la mesa. Los valores máximos han sido: 30088.5 y 30976..

Propósito del registro de datos

1. **Análisis posterior:** Permite estudiar la magnitud de los eventos detectados.
2. **Persistencia:** Almacena datos en un dispositivo USB para que no se pierdan al apagar el sistema.
3. **Documentación:** Proporciona un historial de eventos en un formato legible.
4. **Notificación:** Gracias a que se guardan los datos, en un futuro se podría desarrollar un sistema que fuera capaz de notificar a todos los ciudadanos a través de sus teléfonos móviles.

Resumen

El sistema de registro de datos utiliza el sistema de archivos FATFS para guardar los valores de aceleración máxima detectados en un archivo de texto en un dispositivo USB. Esto permite mantener un registro persistente de los eventos sísmicos, proporcionando una herramienta útil para el análisis y monitoreo.

1.9 Indicadores LED

En este proyecto, los LEDs se utilizan como indicadores visuales para señalar la detección de eventos sísmicos, conexión USB... Esto proporciona retroalimentación en tiempo real sobre el estado del sistema, especialmente cuando se detectan vibraciones significativas que superan el umbral definido.

Configuración de los LEDs

Los LEDs están configurados como salidas digitales en los pines GPIO. La configuración inicial se realiza en la función **MX_GPIO_Init()** vista previamente:

```
GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                      |Audio_RST_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;/ Configuramos los pinez como salida push-pull.
GPIO_InitStruct.Pull = GPIO_NOPULL;// No se usa resistencia pull-up ni pull-down.
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;// Establece una velocidad de conmutación baja.
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);//Inicializamos
```

Activación de LEDs según la detección

El LED principal se activa o desactiva según la detección de un evento sísmico. Esto se gestiona en el bucle principal y depende de si la magnitud de la aceleración supera el umbral definido (SEISMIC_THRESHOLD).

Código de activación/desactivación:

```
HAL_GPIO_WritePin(SEISMIC_LED_PORT, SEISMIC_LED_PIN, accelerationMagnitude > SEISMIC_THRESHOLD ? GPIO_PIN_SET : GPIO_PIN_RESET
```

- **Condición:** El LED se enciende si la magnitud de la aceleración (accelerationMagnitude) excede el umbral definido en SEISMIC_THRESHOLD (2500 mg en nuestro caso).
- **Explicacion funciones usadas:**

  Si se cumple esta condición(`accelerationMagnitude > SEISMIC_THRESHOLD)` se ejecutará lo primero que se encuentre despues de la ? (*GPIO_PIN_SET*) en caso contrario se ejecuatrá lo siguiente (*GPIO_PIN_RESET*)

Estrategia de uso de LEDs.

1. **LED para eventos sísmicos:**
   - Un LED específico se enciende cuando se detecta un evento sísmico que supera el umbral.
   - Este LED proporciona una indicación visual inmediata al usuario.
2. **LED para conexión USB:**
   - Otro LED indica si un dispositivo USB está conectado y el sistema de archivos FAT está montado.
   - Esto se maneja en los callbacks de conexión y desconexión USB vistas previamente:

     ```
     void onHostUserClassActiveCallback(void)
     void onHostUserDisconnectionCallback(void)
     ```

Ejemplo de flujo de activación de LEDs

1. **En el bucle principal:**
   - El sistema lee datos del acelerómetro y calcula la magnitud de la aceleración.

- Si la magnitud supera el umbral, se enciende el LED asociado al evento sísmico.
- Cuando la magnitud vuelve a estar por debajo del umbral, el LED se apaga.

2. **Durante la conexión USB:**
   - Al conectar un dispositivo USB, se monta el sistema de archivos y se enciende un LED para confirmar la conexión
   - Al desconectar el USB, se desmonta el sistema de archivos y el LED se apaga.

Resumen:

1. **Configuración inicial:** Los LEDs están configurados como salidas digitales en los pines GPIO.
2. **Indicador de eventos sísmicos:**
   - Un LED se enciende cuando se detecta un evento sísmico (aceleración que supera el umbral).
3. **Indicador de estado USB:**
   - Un LED separado se utiliza para mostrar el estado de conexión de un dispositivo USB.
4. **Propósito:** Los LEDs proporcionan una forma simple y efectiva de visualizar el estado del sistema y los eventos detectados en tiempo real.

Implementación del Sistema

1.10 Flujo de Trabajo—Resumido

1. Inicio: Inicialización de todos los periféricos y configuraciones básicas.
2. Lectura de Datos: Captura continua de datos del acelerómetro.
3. Cálculo de Aceleración.
4. Detección: Comparación de la aceleración con el umbral definido.
5. Registro: Escritura de datos en un archivo si se detecta un evento.
6. Indicadores: Activación de LEDs según los eventos detectados.

1.11 Pruebas y Resultados

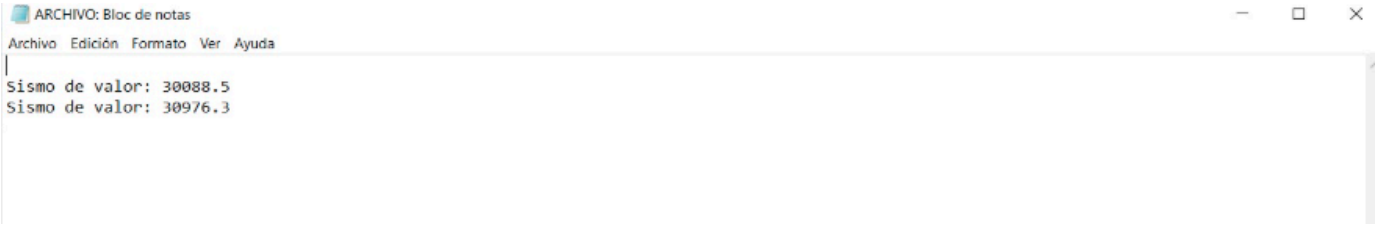
Estrategias de Prueba:

- Para comprobar que funcionaba el sismómetro se realizaron varias pruebas con el micro, la idea es simular las vibraciones del seísmo golpeando una mesa sobre la que se encuentra nuestra placa:

Primero no se dió ningún golpe, como era de esperar no se encendió en ningún momento el led que detecta el evento sísmico y no se registró nada en nuestro USB.

A continuación, se empezó a dar golpes suaves. Se encendía nuestro led, ya que detectaba dichos golpes como eventos sísmicos. La aceleración máxima se registraba en el pendrive.

Por último, se incrementó la intensidad de los golpes. Se pudo observar como la aceleración máxima que se registraba en nuestro archivo era mucho mayor que en la simulación previa.



Además, al conectar el pendrive al micro se encendía el LED correspondiente. Por lo tanto, todo funcionaba como era de esperar.