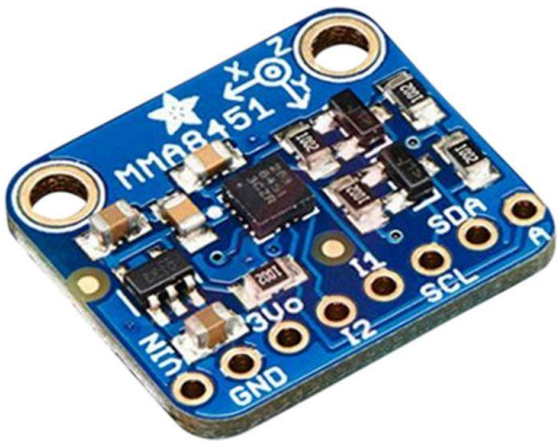
E.T.S. INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE MÁLAGA

PRÁCTICA FINAL – ACELEROMETRO ADAFRUIT MMA8451



ÓSCAR DIAZ SALDAÑA

Diseño con microcontroladores

Índice de actividades

Objetivo. 2
¿Qué es I2C? 2
Información acerca de Adafruit MMA8451. 2
Implementación. 4
Conclusión. 8
Bibliografía. 9

Objetivo.

Se va a implementar un controlador en la placa NUCLEO-L476RG para el dispositivo Adafruit MMA8451. Se utilizan principalmente en dispositivos portátiles, equipos móviles y productos electrónicos de consumo ampliamente conocidos.

El acelerómetro es un instrumento que mide la vibración o aceleración del movimiento de un dispositivo. La fuerza generada por la vibración hace que la masa comprima al material piezoeléctrico, generando una carga eléctrica proporcional a la fuerza ejercida. Con este controlador vamos a obtener la posición y orientación del dispositivo, permitiendo tener información de la ubicación del dispositivo. Es un dispositivo que proporciona la capacidad de medir y analizar la aceleración lineal y angular.

Para manipular este dispositivo tenemos que hacer uso de la comunicación I2C.

¿Qué es I2C?

I2C es un puerto y protocolo de comunicación serial, define la trama de datos y las conexiones físicas para transferir bits entre 2 dispositivos digitales. El puerto incluye dos cables de comunicación, SDA y SCL. Además, el protocolo permite conectar hasta 127 dispositivos esclavos con esas dos líneas, con hasta velocidades de 100, 400 y 1000 kbits/s.

El protocolo I2C es uno de los más utilizados para comunicarse con sensores digitales, ya que a diferencia del puerto Serial, su arquitectura permite tener una confirmación de los datos recibidos, dentro de la misma trama, entre otras ventajas.

Los modos de comunicación en I2C se refieren a las distintas tramas que pueden formarse en el bus.

- Maestro-Transmisor y Esclavo-Receptor. Este modo se usa cuando se desea configurar un registro del esclavo I2C.
- Maestro-Receptor Y Esclavo-Transmisor. Se usa cuando queremos leer información del sensor I2C.

Información acerca de Adafruit MMA8451.

Se va a describir la información de los registros utilizados para la configuración del dispositivo.

- **DEV_ADDR (0x1D<<1)**: este registro contiene la dirección del dispositivo, con la cual podemos direccionarlo en el bus I2C para confirmar que este es el dispositivo que queremos utilizar.
- **WHO_AM_I (0x0D)**: este registro sirve de identificación. El valor por defecto es 0x1A. Sirve para preguntarle al dispositivo su “nombre” y comprobar que estamos trabajando con el correcto.

0x0D: WHO_AM_I Device ID register (read only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	1	1	0	1	0

- **REG_OUT_X_MSB(0x01) | REG_OUT_X_LSB(0x02)**: este registro nos proporciona la magnitud del eje X en una salida de 14 bits expresados como complemento 2.

0x01: OUT_X_MSB: X_MSB register (read only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
XD13	XD12	XD11	XD10	XD9	XD8	XD7	XD6

0x02: OUT_X_LSB: X_LSB register (read only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
XD5	XD4	XD3	XD2	XD1	XD0	0	0

- **REG_OUT_Y_MSB(0x03) | REG_OUT_Y_LSB(0x04)**: este registro nos proporciona la magnitud del eje Y en una salida de 14 bits expresados como complemento 2.

0x03: OUT_Y_MSB: Y_MSB register (read only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
YD13	YD12	YD11	YD10	YD9	YD8	YD7	YD6

0x04: OUT_Y_LSB: Y_LSB register (read only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
YD5	YD4	YD3	YD2	YD1	YD0	0	0

- **REG_OUT_Z_MSB(0x05) | REG_OUT_Z_LSB(0x06)**: este registro nos proporciona la magnitud del eje Z en una salida de 14 bits expresados como complemento 2.

0x05: OUT_Z_MSB: Z_MSB register (read only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ZD13	ZD12	ZD11	ZD10	ZD9	ZD8	ZD7	ZD6

0x06: OUT_Z_LSB: Z_LSB register (read only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ZD5	ZD4	ZD3	ZD2	ZD1	ZD0	0	0

- **REG_PL_STATUS(0x10)**: este registro puede ser leído para obtener los cambios de orientación del dispositivo. Vamos a utilizar solo los 3 bits bajos para obtener las orientaciones.

0x10: PL_STATUS register (read only)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
NEWLP	LO	—	—	—	LAPO[1]	LAPO[0]	BAFRO

LAPO[1:0](1)	Landscape/portrait orientation. Default value: 00 00: Portrait up: Equipment standing vertically in the normal orientation 01: Portrait down: Equipment standing vertically in the inverted orientation 10: Landscape right: Equipment is in landscape mode to the right 11: Landscape left: Equipment is in landscape mode to the left.
BAFRO	Back or front orientation. default value: 0 0: Front: Equipment is in the front facing orientation. 1: Back: Equipment is in the back facing orientation.

- **REG_XYZ_DATA_CFG(0x0E)**: este registro establece el rango y filtro de paso alto para la salida. Lo utilizamos al inicializar el dispositivo poniendo la configuración de filtro por defecto y el escalado de 2g. Enviamos el comando de escritura 0x00.

0x0E: XYZ_DATA_CFG (read/write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	HPF_OUT	0	0	FS1	FS0

Table 20. XYZ data configuration descriptions

Field	Description
HPF_OUT	Enable high-pass output data 1 = output data high-pass filtered. Default value: 0.
FS[1:0]	Output buffer data format full scale. Default value: 00 (2 *g*).

- **REG_CONTROL_REG1(0x2A)**: este registro lo utilizamos para configurar la frecuencia de muestreo del aparato, velocidad de lectura, reducción de ruido y activar el dispositivo. Comenzamos poniéndolo en modo “standby” escribiendo 0x00 y después mandamos el comando 0x09 para poner la frecuencia de modo sleep a 50Hz, muestreo de datos a 400Hz, modo normal de reducción de ruido, modo normal de lectura y activamos el escalado.

0x2A: CTRL_REG1 register (read/write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ASLP_RATE1	ASLP_RATE0	DR2	DR1	DR0	LNOISE	F_READ	ACTIVE

Table 61. CTRL_REG1 description

Field	Description
ASLP_RATE[1:0]	Configures the auto-wake sample frequency when the device is in sleep mode. Default value: 00. See Table 62 for more information.
DR[2:0]	Data-rate selection. Default value: 000. See Table 63 for more information.
LNOISE	Reduced noise reduced maximum range mode. Default value: 0. (0: Normal mode; 1: Reduced noise mode)
F_READ	Fast read mode: Data format limited to single byte default value: 0. (0: Normal mode 1: Fast-read mode)
ACTIVE	Full-scale selection. Default value: 00. (0: Standby mode; 1: Active mode)

Table 62. Sleep mode rate description

ASLP_RATE1	ASLP_RATE0	Frequency (Hz)
0	0	50
0	1	12.5
1	0	6.25
1	1	1.56

Table 63. System output data rate selection

DR2	DR1	DR0	ODR	Period
0	0	0	800 Hz	1.25 ms
0	0	1	400 Hz	2.5 ms
0	1	0	200 Hz	5 ms
0	1	1	100 Hz	10 ms
1	0	0	50 Hz	20 ms
1	0	1	12.5 Hz	80 ms
1	1	0	6.25 Hz	160 ms
1	1	1	1.56 Hz	640 ms

- **REG_CONTROL_REG2(0x2B)**: este registro lo utilizamos para establecer el modo de muestreo normal y de alimentación, también se puede configurar modo de test, reset y auto-sleep. Enviamos el comando 0x00.

0x2B: CTRL_REG2 register (read/write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ST	RST	0	SMODS1	SMODS0	SLPE	MODS1	MODS0

Table 66. MODS oversampling modes

(S)MODS1	(S)MODS0	Power mode
0	0	Normal
0	1	Low Noise Low Power
1	0	High Resolution
1	1	Low Power

- **REG_CONTROL_REG3(0x2C)**: este registro sirve para configurar los modos de interrupción del dispositivo. Nosotros no hacemos uso de las interrupciones por lo que vamos a enviar 0x00 para tener todos los modos desactivados.

0x2C: CTRL_REG3 register (read/write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FIFO_GATE	WAKE_TRANS	WAKE_LNDPRT	WAKE_PULSE	WAKE_FF_MT	—	IPOL	PP_OD

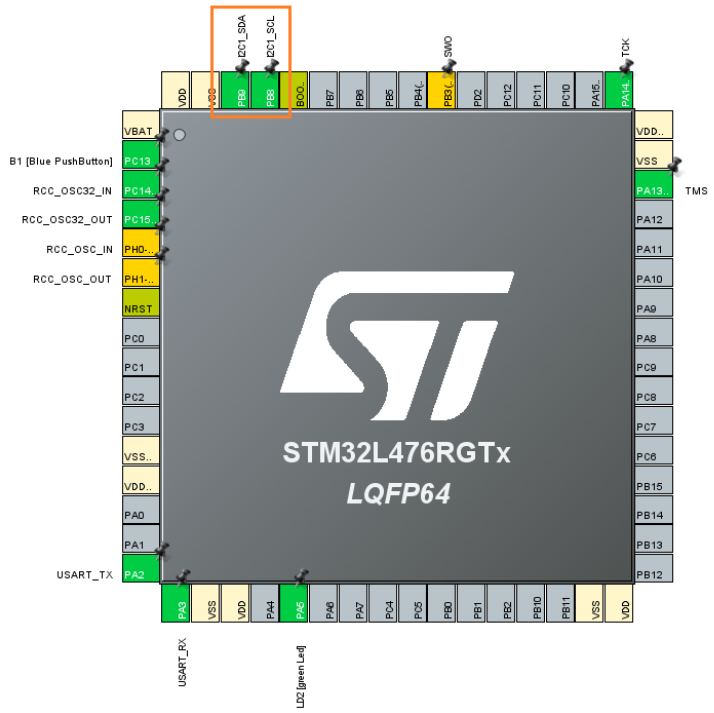
Table 68. CTRL_REG3 description

Field	Description
FIFO_GATE	0: FIFO gate is bypassed. FIFO is flushed upon the system mode transitioning from wake to sleep mode or from sleep to wake mode. Default value: 0. 1: The FIFO input buffer is blocked when transitioning from wake to sleep mode or from sleep to wake mode until the FIFO is flushed. Although the system transitions from wake to sleep or from sleep to wake the contents of the FIFO buffer are preserved, new data samples are ignored until the FIFO is emptied by the host application. If the FIFO_GATE bit is set to logic '1' and the FIFO buffer is not emptied before the arrival of the next sample, then the FGERR bit in the SYS_MOD register (0x0B) will be asserted. The FGERR bit remains asserted as long as the FIFO buffer remains un-emptied. Emptying the FIFO buffer clears the FGERR bit in the SYS_MOD register.
WAKE_TRANS	0: Transient function is bypassed in sleep mode. Default value: 0. 1: Transient function interrupt can wake up system
WAKE_LNDPRT	0: Orientation function is bypassed in sleep mode. Default value: 0. 1: Orientation function interrupt can wake up system
WAKE_PULSE	0: Pulse function is bypassed in sleep mode. Default value: 0. 1: Pulse function interrupt can wake up system
WAKE_FF_MT	0: Freefall/motion function is bypassed in sleep mode. Default value: 0. 1: Freefall/motion function interrupt can wake up
IPOL	Interrupt polarity active high, or active low. Default value: 0. 0: Active low; 1: Active high
PP_OD	Push-pull/open drain selection on interrupt pad. Default value: 0. 0: Push-pull; 1: Open drain

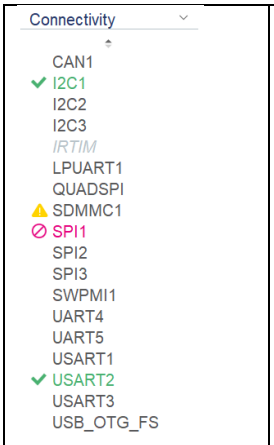
Implementación.

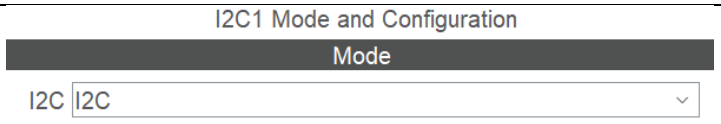
Para comenzar a desarrollar el proyecto tenemos que configurar los pines y las frecuencias para las cuales vamos a trabajar.

Utilizamos el software STM32CubeMX. Comenzamos configurando los pines que vamos a utilizar para dar señal a SCL y SDA.

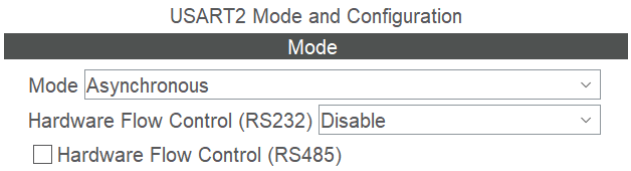


Utilizamos los pines PB9 para datos y PB8 para la señal de reloj. Estos pines están marcados en la placa para la comunicación I2C, teniendo hasta 3 pares para poder comunicarte en modo maestro esclavo. También, el pin PA2 y PA3 para poder realizar la comunicación USART para enviar los datos y poder verlos de forma remota.



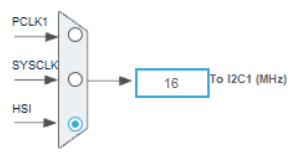


En conectividad tenemos que activar el I2C1 que corresponde con los pines que hemos utilizado en modo I2C.



Verificar también que tenemos la USART activada para trabajar.

Debemos configurar también el reloj del sistema para establecer la frecuencia a la que se va a comunicar el sistema.

Reloj I2C1. HSI = Oscilador interno.	Reloj USART2. PCLK1 = Oscilador sistema
	

Con esto tendríamos configurada la configuración de los pines y relojes que vamos a utilizar.

En Eclipse los métodos que vamos a utilizar son:

- HAL_I2C_Master_Transmit: Maestro se hace con el bus y envía información a esclavo.
- HAL_I2C_Master_Receive: Maestro se hace con el bus y recibe información de esclavo.
- HAL_I2C_Mem_Read: Lee una cantidad de datos en modo de bloqueo desde una dirección de memoria específica.

Utilizamos las instrucciones de Master debido a que nuestro microcontrolador NUCLEO-L476RG actúa como maestro sobre el acelerómetro MMA8451.

Para realizar la implementación, nos hemos basado en la librería de Adafruit en C++ para ver como se configura los registros y como utilizarlos.

Método begin para iniciar el periférico. Creamos un array como buffer para poder escribir caracteres y poder mostrarlos por la salida estándar. Creamos también, un array de dos posiciones para guardar en la primera posición el registro y en la segunda el valor a escribir. Configuramos los diferentes registros y finalmente comprobamos si está bien inicializado.

```
void begin(){
	char buffer[64];

	//standby
	uint8_t datos[2] = {REG_CONTROL_REG1, 0x00};
	HAL_I2C_Master_Transmit(&hi2c1, DEV_ADDR, datos, 2, 500);

	datos[0] = REG_PL_CFG;
	datos[1] = 0x40;
	HAL_I2C_Master_Transmit(&hi2c1, DEV_ADDR, datos, 2, 500);

	datos[0] = REG_XYZ_DATA_CFG;
	datos[1] = 0x00;
	HAL_I2C_Master_Transmit(&hi2c1, DEV_ADDR, datos, 2, 500);

	//activamos modo normal
	datos[0] = REG_CONTROL_REG2;
	datos[1] = 0x00;
	HAL_I2C_Master_Transmit(&hi2c1, DEV_ADDR, datos, 2, 500);

	//modo no interrupciones
	datos[0] = REG_CONTROL_REG3;
	datos[1] = 0x00;
	HAL_I2C_Master_Transmit(&hi2c1, DEV_ADDR, datos, 2, 500);

	//data rate on + activo on
	datos[0] = REG_CONTROL_REG1;
	datos[1] = 0x09;
	HAL_I2C_Master_Transmit(&hi2c1, DEV_ADDR, datos, 2, 500);

	HAL_StatusTypeDef estado;
	estado = HAL_I2C_IsDeviceReady(&hi2c1, DEV_ADDR, 1, 500);
	if(estado != HAL_OK){
		strcpy(buffer, "ERROR\n\r");
```

```
	}else{
		strcpy(buffer, "Inicializando...\n\r\n\r");
	}

	HAL_UART_Transmit(&huart2, (uint8_t *)buffer, strlen(buffer),
HAL_MAX_DELAY);
}
```

Método whoami. Para recibir el valor de identificación del periférico.

```
int whoami(){
	uint8_t valor;
	HAL_I2C_Mem_Read(&hi2c1, DEV_ADDR, REG_WHO_AM_I, 1, &valor, 1, 500);
	return valor;
}
```

Método getAccel. Para obtener los valores de aceleración de los diferentes registros. Obtenemos la parte baja y alta, la desplazamos y la guardamos en su posicion cada una.

```
void getAccel(float *accel){
	uint8_t aux[6];

	HAL_I2C_Mem_Read(&hi2c1, DEV_ADDR, REG_OUT_X_MSB, 1, &aux[0], 1, 500);
	HAL_I2C_Mem_Read(&hi2c1, DEV_ADDR, REG_OUT_X_LSB, 1, &aux[1], 1, 500);
	accel[0] = ((int16_t)aux[0]<<6) | (aux[1]>>2);

	HAL_I2C_Mem_Read(&hi2c1, DEV_ADDR, REG_OUT_Y_MSB, 1, &aux[2], 1, 500);
	HAL_I2C_Mem_Read(&hi2c1, DEV_ADDR, REG_OUT_Y_LSB, 1, &aux[3], 1, 500);
	accel[1] = ((int16_t)aux[2]<<6) | (aux[3]>>2);

	HAL_I2C_Mem_Read(&hi2c1, DEV_ADDR, REG_OUT_Z_MSB, 1, &aux[4], 1, 500);
	HAL_I2C_Mem_Read(&hi2c1, DEV_ADDR, REG_OUT_Z_LSB, 1, &aux[5], 1, 500);
	accel[2] = ((int16_t)aux[4]<<6) | (aux[5]>>2);

	accel[0] = (accel[0]/4096);
	accel[1] = (accel[1]/4096);
	accel[2] = (accel[2]/4096);
}
```

Método getOri. Para obtener la orientación del periférico. Leo el registro y cojo los bits del valor para ver la orientación desplazando. Compruebo si es mayor o igual que 128 para comprobar el bit si está correcto.

```
void getOri(int *orientacion){
	uint8_t aux;

	HAL_I2C_Mem_Read(&hi2c1, DEV_ADDR, REG_PL_STATUS, 1, &aux, 1, 500);

	if(aux >= 128){
		orientacion[0] = (aux >> 1)&1;
		orientacion[1] = (aux >> 2)&1;
		orientacion[2] = (aux >> 0)&1;
	}

}
```

Método main. Método principal con la ejecución del programa. Comenzamos ejecutando begin para configurar el periférico y compruebo si es el que quiero utilizar.

```
float accel[3];
  int orientacion[3];
  char buffer[128];

  strcpy(buffer, "\n\r\n\r\n\rBienvenido a MMA8451 - ADAFRUIT \n\r\n\r");
  HAL_UART_Transmit(&huart2, (uint8_t *)buffer, strlen(buffer),
HAL_MAX_DELAY);
  begin();
  HAL_Delay(500);

  if( whoami()==0x1a){
        strcpy(buffer, "Dispositivo inicializado correctamente \n\r\n\r");
  } else{
        strcpy(buffer, "Dispositivo NO inicializado correctamente \n\r");
        return -1;
  }
  HAL_UART_Transmit(&huart2, (uint8_t *)buffer, strlen(buffer),
HAL_MAX_DELAY);
```

En el while, que se ejecuta continuamente, recogemos los valores y los mostramos.

```
while (1)
  {
        getAccel(accel);
        sprintf(buffer, "Posicion: X = %.3f Y = %.3f Z = %.3f \n\r",
accel[0], accel[1], accel[2]);
        HAL_UART_Transmit(&huart2, (uint8_t *)buffer, strlen(buffer),
HAL_MAX_DELAY);

        getOri(orientacion);
        sprintf(buffer, "Orientacion: X = %i Y = %i Z = %i \n\r\n\r",
orientacion[0], orientacion[1], orientacion[2]);
        HAL_UART_Transmit(&huart2, (uint8_t *)buffer, strlen(buffer),
HAL_MAX_DELAY);

        HAL_Delay(500);
  }
```

Conclusión.

Este dispositivo es muy útil para dispositivos que se muevan, pudiendo obtener información de donde se encuentra y su orientación. También, he aprendido a trabajar con I2C, para hacer comunicaciones maestro-esclavo las cuales son muy utilizadas en la actualidad.

Finalmente, se incluye un video de muestra de ejecución.

https://drive.google.com/file/d/1fFymeTYjkWv_5B5afSQvjqma8uwYFtOX/view?usp=sharing

Bibliografía.

https://learn.adafruit.com/adafruit-mma8451-accelerometer-breakout

https://pdf1.alldatasheet.es/datasheet-pdf/view/392303/FREESCALE/MMA8451Q.html

https://es.omega.com/prodinfo/acelerometro.html

https://es.wikipedia.org/wiki/Aceler%C3%B3metro

https://github.com/adafruit/Adafruit_MMA8451_Library

https://hetpro-store.com/TUTORIALES/i2c/