PROYECTO TD II UTN-FRA 2024
INCLINÓMETRO DIGITAL MÁS ODÓMETRO

Hernández, Alexis de la Cruz
e-mail: alexisdelacruzhernandez@gmail.com

RESUMEN: *El proyecto realizado consiste en un inclinómetro digital realizado a partir de un acelerómetro y en un odómetro realizado a partir de un encoder rotativo junto a una rueda. El mismo cuenta con alarma auditiva y visual configurable por el usuario.*

PALABRAS CLAVE: Acelerómetro, Encoder rotativo, Inclinómetro, Odómetro.

1 INTRODUCCIÓN

La función del inclinómetro es llevada a cabo por un acelerómetro que sensa las aceleraciones en los ejes X, Y y Z, al medir estas aceleraciones en reposo obtenemos el efecto de la gravedad, pudiendo calcular la inclinación.

El odómetro utiliza un encoder rotativo solidario a una rueda con perímetro conocido, al girar la rueda recorremos su perímetro y, contando los pulsos enviados por el encoder, se calcula la distancia equivalente medida.

Para visualizar los menús se emplea un display OLED y para seleccionarlos se utiliza un teclado matricial de membrana.

El mismo cuenta con indicación visual, por medio de un led, y auditiva, por medio de un buzzer pasivo, habrá indicaciones cuando se utilice el teclado, se guarden datos en memoria, y cuando se utilicen los modos de medición con alarma.

2 DIAGRAMA EN BLOQUES

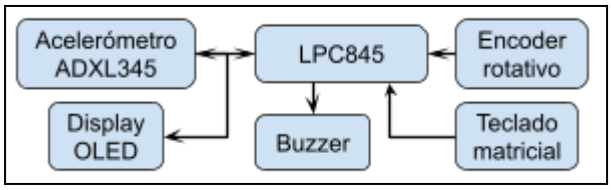

Figura 1. Diagrama en bloques.

3 DESCRIPCIÓN DE BLOQUES

3.1 LPC845

El microcontrolador utilizado para el desarrollo del proyecto es el LPC845, más específicamente se usó la placa de desarrollo LPC845-Breakout (imagen 1).

Como características del microcontrolador podemos mencionar su arquitectura Arm Cortex M0+ de 32 bits, la cual es de bajo consumo.

Los periféricos utilizados en el desarrollo de este proyecto son:

- Systick.
- SC Timer.
- DAC (Digital-to-Analog Converter).
- FLASH.
- I^2C (Inter-Integrated Circuit).
- GPIO.

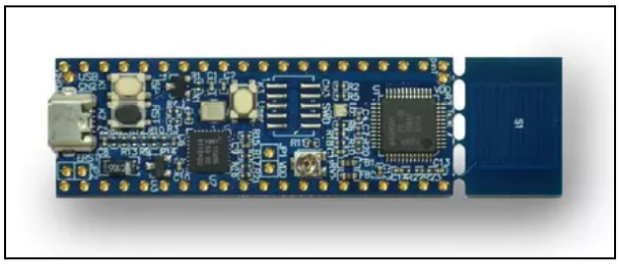

Imagen1. LPC845-Breakout

3.2 TECLADO MATRICIAL

Para desplazarse por los menús se eligió un teclado matricial 4x4 de membrana (imagen 2).

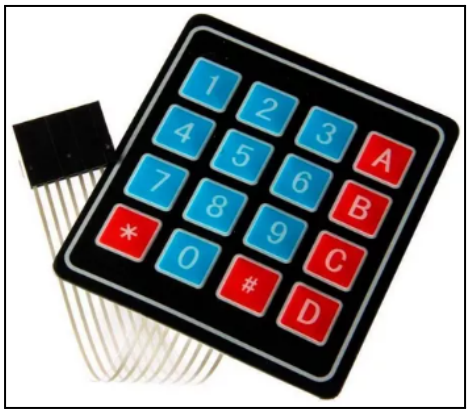

Imagen 2. Teclado matricial

Este tipo de configuración nos permite tener 16 pulsadores funcionales con 8 conexiones, ya que utiliza una matriz de 4 filas y 4 columnas como se muestra en la figura 2.

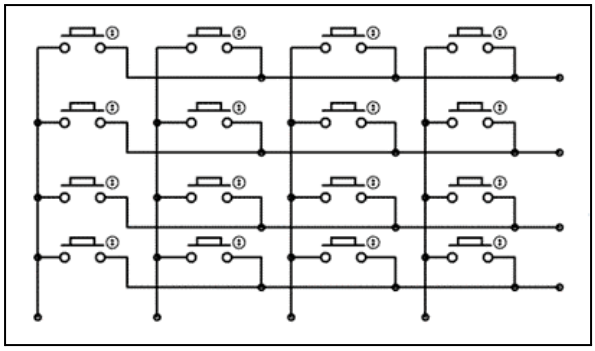

Figura 2. Esquemático teclado

Se utilizaron las columnas como salidas y las filas como entradas, las salidas no utilizadas manteniéndose en 3,3V y la que se está leyendo en 0V, las entradas cuentan con resistencias pull-up para mantener un estado lógico alto mientras no se presionan.

3.3 ENCODER ROTATIVO

Para realizar la medición de distancia se implementa un encoder rotativo de 30 pulsos por vuelta (imagen 3), el mismo es de tipo incremental y permite la detección del sentido de rotación.

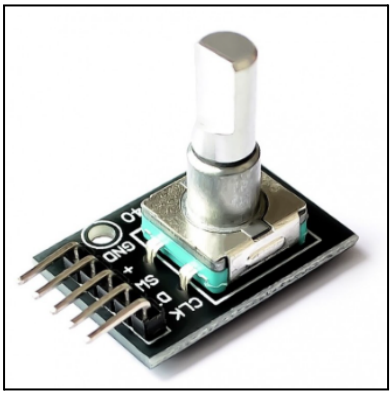

Imagen 3. Encoder rotativo Ky-040

Las señales enviadas por este tipo de encoder son las mostradas en la figura 3, para su implementación se usó una interrupción externa para detectar los cambios y poder medir la rotación, luego distancia.

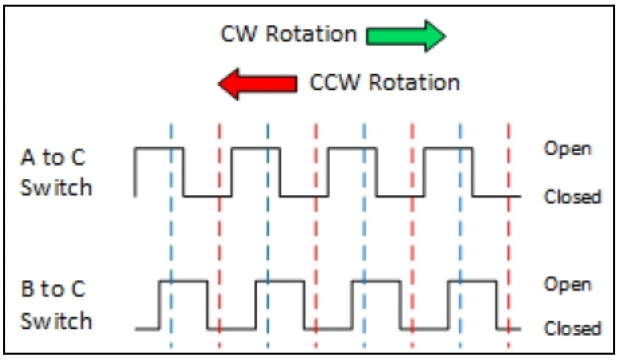

Figura 3. Diagrama de tiempos encoder

Solidario al eje del encoder se coloca una rueda de perímetro conocido para conocer la distancia recorrida. Se eligió como resolución 0,5 centímetros, por lo tanto un pulso del encoder equivale a esa distancia, como se cuenta con 30 pulsos el perímetro total es de 15 centímetros, dándonos un radio para la circunferencia de 2,387 centímetros.

3.4 BUZZER

La indicación auditiva se implementó con un buzzer pasivo (imagen 4), el mismo requiere para funcionar una señal cuadrada de determinada frecuencia.

Imagen 4. Buzzer pasivo

Si bien las frecuencias de audio reproducibles son amplias se eligió la frecuencia de 1kHz.

El volumen del buzzer es configurable por el usuario, para lograr este ajuste se utilizó el DAC para controlar la corriente que atraviesa al buzzer junto al SC Timer para generar un PWM. La señal del DAC sigue la frecuencia del PWM por medio de su interrupción.

Debido al límite de corriente del DAC (350µA) y la corriente requerida por el buzzer (60mA), se utilizó un par de transistores en configuración darlington (figura 4) con una resistencia en emisor para estabilizar la ganancia frente a cambios del hfe de los transistores.

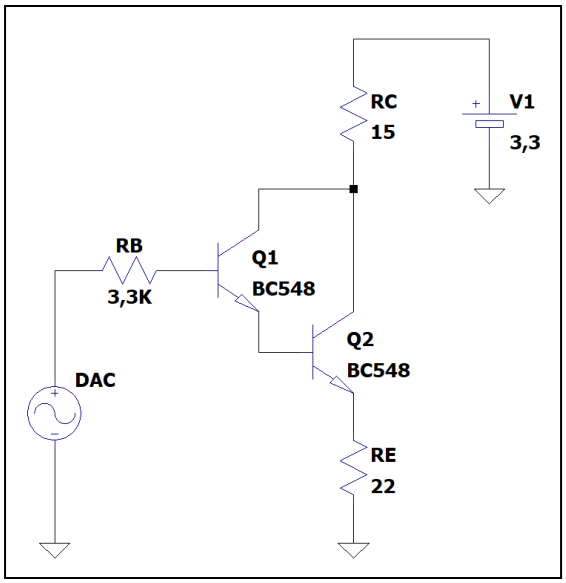

Figura 4. Control del buzzer

La resistencia Rc es la resistencia equivalente del buzzer; tanto Rc como su corriente fueron obtenidas experimentalmente ya que no se contaba con la hoja de datos del mismo. El cálculo de RB y RE se realiza más adelante en el informe.

3.5 DISPLAY OLED

Para la visualización de los distintos menús y de las mediciones se seleccionó un display OLED de 128x64 pixeles (imagen 5).

Imagen 5. Display OLED 128x64

Este display cuenta con un controlador SSD1306, el cual se comunica con el microcontrolador por medio del protocolo I²C, pudiendo recibir tanto comandos como escrituras en RAM, lo cual es equivalente a manejar los píxeles de la pantalla.

El display tiene un tamaño de 0,96 pulgadas con una resolución de 128x64 píxeles, la tecnología OLED se caracteriza por su gran contraste, buena calidad de imagen y bajo consumo de energía, ideal para aplicaciones portátiles.

3.6 ACELERÓMETRO ADXL345

El inclinómetro se implementó con el acelerómetro ADXL345 (imagen 6), el mismo se encarga de medir las aceleraciones en los ejes X, Y y Z. Con estos valores de aceleración logramos calcular su inclinación respecto al plano con nivel 0°.

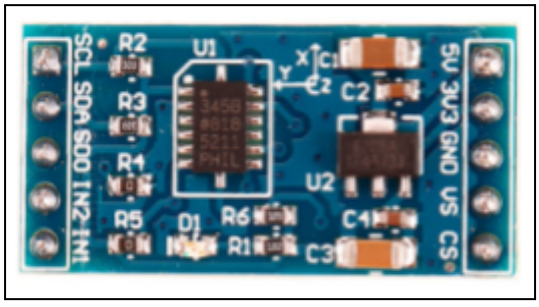

Imagen 6. Acelerómetro ADXL345

Este sensor tiene la posibilidad de comunicarse tanto con protocolo I²C como SPI, ya que el display se comunica con I²C se decidió continuar con este protocolo para con la misma línea de comunicación controlar los dos módulos.

Al mismo se le pueden configurar cuatro rangos de medición: ±2/4/8/16g, para nuestra aplicación, que se basa en medir la aceleración de la gravedad, utilizamos el rango de ±2g. Este rango cuenta con una resolución de 10 bits, dándonos aproximadamente una resolución de 0,004g por bit.

Para utilizar el protocolo I²C el datasheet nos especifica que el pin CS (chip select del protocolo SPI) debe estar en 1 lógico y el pin SDO determina el addres que tendrá, si está en alto su address será 0x1D y si está en bajo será 0x53.

Este módulo cuenta con pines de interrupción configurables, en este caso se configuró una interrupción de DATA READY para saber cuándo están listos los datos para leerse.

Se seleccionó este módulo frente a otros del mercado por dos razones, su error es menor aunque su resolución también, y su consumo de energía es mucho menor a otros ampliamente utilizados como el MPU6050.

3.7 ALIMENTACIÓN

Para energizar el circuito se optó por usar un módulo de fuente DC-DC step down basada en el integrado LM2596 (imagen 7).

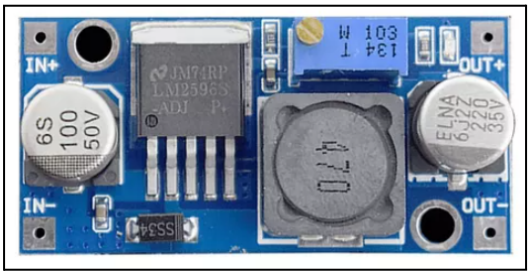

Imagen 7. Fuente DC-DC step down

Se eligió este tipo de fuente en vez de una regulación lineal ya que es más eficiente energéticamente, y, al alimentarlo con baterías, su eficiencia es importante.

La tensión de entrada permitida es de 4V a 40V, el voltaje de salida es de 1,25V a 37V, siendo este valor menor al de entrada, y la corriente máxima entregada es de 3A.

4 PROTOCOLO I²C

Como se mencionó anteriormente tanto el display como el acelerómetro utilizan el protocolo I²C, por lo tanto en el presente ítem explicaremos su funcionamiento y los comandos propios de cada módulo.

4.1 CARACTERÍSTICAS

Es una interfaz de comunicación half duplex sincronica. Half duplex significa que el envío de datos es en ambas direcciones (del dispositivo A al B y del B al A) pero no de forma simultánea (mientras A envía B no puede enviar). Sincrónica hace referencia a que los datos enviados están sincronizados con un reloj.

Está basado en el sistema maestro-esclavo, donde el maestro sincroniza el bus, direcciona los esclavos y escribe o lee datos hacia y desde los registros de los esclavos.

Utiliza dos líneas de control: SDA, donde se envían los datos, y SCL, donde se envía el sincronismo con el reloj. Los dispositivos tienen salidas colector abierto, por lo que, en estados de inactividad, las líneas del bus no tienen un estado definido, para solventar esto se colocan resistencias de pull-up para mantener el estado en alto.

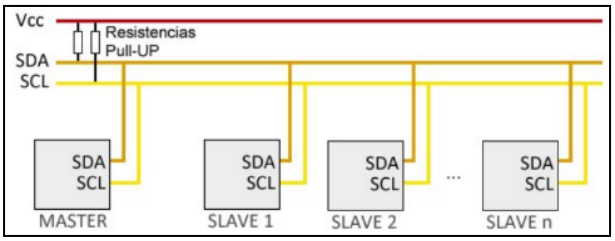

Figura 5. Diagrama conexión I²C

4.2 TRANSMISIÓN Y RECEPCIÓN

En forma genérica la secuencia a seguir en el protocolo es la indicada en la figura 6.

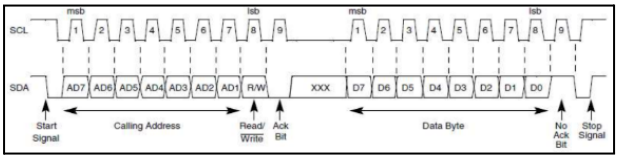

Figura 6. Comunicación I²C

La comunicación se realiza con paquetes de 8 bits, se inicia la comunicación cuando el canal SDA pasa de alto a bajo, luego se envían los 7 bits del address del esclavo junto con un bit que indica lectura (1) o escritura (0). Luego se envían o reciben paquetes de 8 bits dependiendo del driver del esclavo. Cada paquete de datos finaliza con la recepción de un bit de reconocimiento (ACK) o no reconocimiento (NACK).

4.3 DISPLAY OLED

Para la comunicación con el display OLED solo se escriben datos. Tiene dos posibilidades de escritura, de comandos o de datos.

Para la escritura de comandos se debe enviar el address (0x3C) seguido de 0x00, esto identifica al comando, una vez especificado esto se envía el registro a modificar y luego el nuevo valor del registro.

Para la escritura de datos (pixeles) se debe enviar el address (0x3C) seguido de 0x40, esto identifica al dato, luego se envían paquetes de datos que identifican los estados de los pixeles, siendo 1 encendido y 0 apagado.

4.4 ACELERÓMETRO

En el caso del acelerómetro se realizan más lecturas que escrituras, las escrituras se realizan para configurar y encender el módulo, las lecturas para leer los valores de aceleración.

Para escribir en los registros se debe envíar el address (0x1D), envíar el registro a escribir, y luego enviar los bytes a guardar.

Para leer los datos se optó por realizar una lectura múltiple ya que todos los datos importantes están en registros consecutivos. Para realizar esto debemos envíar el address (0x1D) indicando escritura, enviar el registro inicial a leer, enviar nuevamente el address (0x1D) indicando lectura y leer todos los bytes deseados.

5 OBTENCIÓN DE DISTANCIA

Previamente se mencionó el diámetro de la rueda asociada al encoder, en esta sección vamos a desarrollar su obtención y cómo se confeccionó.

La resolución mínima deseada es de 0,5 centímetros como ya se dijo. Como el encoder tiene 30 pasos por vuelta el perímetro total es:

$$P = 30\,.\,Resolución = 15\,cm \quad (1)$$

Con el perímetro calculado podemos calcular el radio de la circunferencia para luego diseñar la rueda.

$$P = 2\pi R \quad (2)$$

$$R = P\,/\,2\pi = 2,3873\,cm \quad (3)$$

Con el radio obtenido ya podemos diseñar la rueda.

Para diseñar la rueda se utilizó el software de diseño 3D fusión 360 ya que la misma va a estar impresa en 3D.

El primer diseño desarrollado es el que se muestra en la figura 7, este modelo cumple con el requisito del radio y tiene la muesca para la introducción del eje del encoder.

Este modelo fue mejorado al de la figura 8, donde se separó la rueda en dos partes, la rueda central, rígida, y la cubierta exterior, flexible.

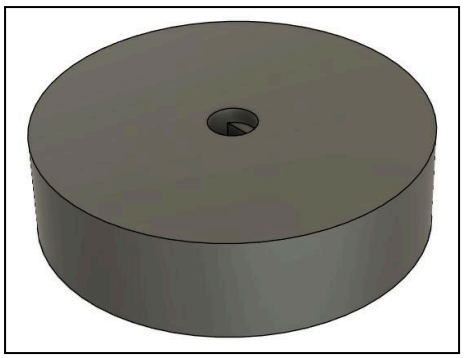

Figura 7. Primer diseño de la rueda

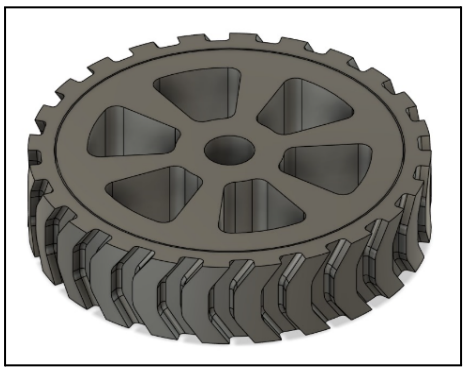

Figura 8. Diseño final de la rueda

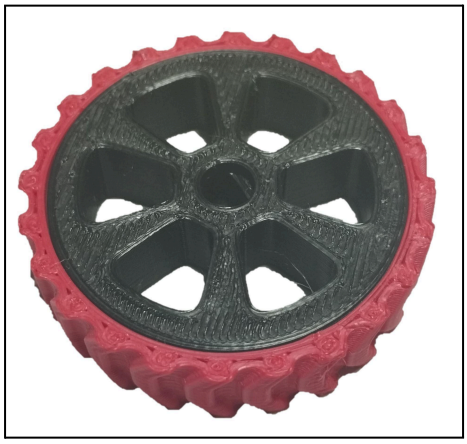

Imagen 8. Rueda impresa en 3D

6 CONTROL DE VOLUMEN

En la sección 3.4 se mostró el circuito utilizado, para su diseño se partió de los siguientes datos:

- $I_B = I_{DAC\,MAX} = 200\mu A$
- $I_C = I_{BUZZER} = 60mA$
- $V_{CC} = 3,3V$
- $R_C = R_{BUZZER} = 15\Omega$
- $Transistores\ BC548C \rightarrow hfe = 300$

Analizando la malla de entrada tenemos:

$$V_{DAC} - I_B . R_B - 2 . V_{BE} - I_C . R_C = 0 \qquad (4)$$

Donde V_{DAC} se tomó como 3,3V y V_{BE} como 0,7V, despejando el valor de I_C reemplazando I_B por I_C / hfe:

$$I_C = \left(V_{DAC} - 2 . V_{BE}\right) / \left(R_E + R_B / hfe^2\right) \qquad (5)$$

Como aproximación se consideró el hfe muy elevado al ser darlington, pudiendo simplificar el término de R_B / hfe^2 y despejar R_E:

$$R_E = \left(V_{DAC} - 2 . V_{BE}\right) / I_C = 31,7\Omega \qquad (6)$$

Como valor comercial se tomó 22Ω para compensar la aproximación tomada. Para el cálculo de R_B despejamos de la ecuación 4:

$$R_B = \left(V_{DAC} - 2 . V_{BE} - I_C . R_E\right) / I_B = 2,9k\Omega \qquad (7)$$

Como valor comercial se tomó 3,3kΩ.

7 OBTENCIÓN DE INCLINACIÓN

Como se mencionó anteriormente la inclinación se calcula a partir de la aceleración en los ejes X, Y y Z del acelerómetro debida a la gravedad. En la figura 9 se muestra un esquema con los vectores obtenidos por el acelerómetro.

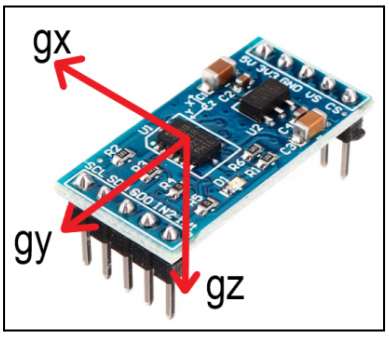

Figura 9. Vectores obtenidos

Aplicando el teorema de pitágoras podemos calcular el módulo del vector resultante de la suma de las tres gravedades.

$$gt = \sqrt{gx^2 + gy^2 + gz^2} \qquad (8)$$

Como el vector gz es perpendicular al plano a nivel podemos calcular la inclinación entre gz y gt para saber la inclinación respecto al nivel. Cómo gt es la hipotenusa y gz es el cateto adyacente del triángulo formado podemos calcular la inclinación usando el arco coseno.

$$\Theta = arc\,cos\,(gz\ /\ gt) \qquad (9)$$

Para darle signo al ángulo obtenido se optó por tomar la gravedad en el eje x como referencia, si es negativa tomamos la inclinación como negativa, y si es positiva la dejamos positiva.

Para reducir el ruido de las mediciones se usó un filtro de promedio móvil con un número de 10 datos.

Este filtro consiste en tomar 10 datos y calcular su promedio, el nuevo dato leído reemplazará al dato más viejo de la lista.

8 MEMORIA FLASH

Para guardar las configuraciones realizadas por el usuario se optó por usar parte de la memoria flash del microcontrolador.

8.1 SEGMENTACIÓN

La memoría flash del LPC845 tiene una capacidad de 64kB, está dividida en 64 sectores de 1kB (1024 bytes) cada uno, en la tabla 1 se muestran los primeros 6 sectores. Para almacenar los datos se utilizó el sector 63 (último) para asegurarnos no pisar el código.

Tabla 1

Sector number	Sector size [KB]	Page number	Address range
0	1	0 - 15	0x0000 0000 - 0x0000 03FF
1	1	16 - 31	0x0000 0400 - 0x0000 07FF
2	1	32 - 47	0x0000 0800 - 0x0000 0BFF
3	1	48 - 63	0x0000 0C00 - 0x0000 0FFF
4	1	64 - 79	0x0000 1000 - 0x0000 13FF
5	1	80 - 95	0x0000 1400 - 0x0000 17FF

8.2 DATOS ALMACENADOS

Los datos a guardar en la memoria FLASH son todas las configuraciones del usuario: contraste de la pantalla OLED (1 byte), volumen del buzzer (1 byte), alarma para el modo inclinómetro (2 bytes), alarma para el modo odómetro (4 bytes). Por lo tanto el total de datos a guardar es de 8 bytes.

La IAP (In Application Programming) nos proveé de funciones dedicadas al manejo de la memoria FLASH. Al estudiar sus funciones concluimos dos cosas: el borrado de la memoria se hace por sector (1024 bytes), y el mínimo de bytes a guardar es de 32.

Para prolongar la vida útil de la memoria se optó por agregar un byte extra de datos que nos indique la versión de los datos guardados, de esta manera podremos guardar datos 16 veces antes de borrar el sector 63.

9 MENÚS

Los menús se hicieron en un editor de imágenes y luego con la ayuda de una página se tradujeron los píxeles a cadenas de bytes para ser cargadas en memoria y mostrarlas. a continuación se mostrarán los diferentes menús con su funcionalidad.

9.1 INICIO

En este menú (imagen 9) la tecla A selecciona el modo odómetro, la tecla B el modo inclinómetro, la tecla D nos lleva a la configuración general y la tecla # guarda las configuraciones en memoria (solo se guardan si se realizó algún cambio en las configuraciones).

Imagen 9. Menú de inicio

9.2 SELECCIÓN DE ALARMA

Luego de seleccionar el modo en el menú de inicio en este (imagen 10) seleccionamos si lo queremos con alarma (tecla A), o sin alarma (tecla B), la tecla C nos envía nuevamente al menú de inicio.

Imagen 10. Menú de selección de alarma

9.3 CONFIGURACIÓN GENERAL

En este menú (imagen 11) se configura el contraste (tecla A) y el volumen (tecla B), yendo del 0% al 99%, la tecla C nos envía al menú de inicio.

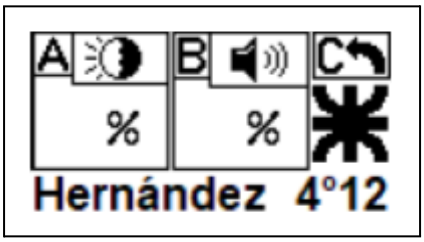

Imagen 11. Menú de configuración general

9.4 CONFIGURACIÓN DE LA ALARMA DEL INCLINÓMETRO

La tecla A nos permite modificar el ángulo de alarma, desde un valor de 0° a 180°, la tecla B modifica el signo de la alarma, ampliando el rango desde -180° a 180°, la tecla D confirma la configuración y la tecla C nos envía al menú de selección de alarma.

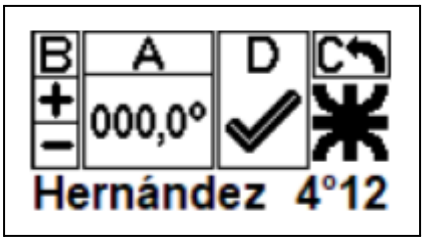

Imagen 12. Menú de configuración de la alarma del inclinómetro

9.5 MEDICIÓN DEL INCLINÓMETRO

En este menú se visualiza la inclinación del instrumento junto con su respectivo signo, la tecla B setea en 0° la actual medición, permitiendo mediciones relativas a determinado ángulo, la tecla D mantiene fijada la última medición hasta que se presione nuevamente la tecla, la tecla C nos devuelve al menú anterior, ya sea el de configuración de la alarma o el de selección de alarma.

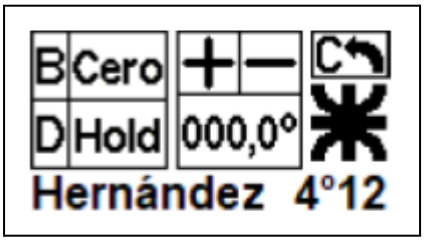

Imagen 13. Menú de medición del inclinómetro

9.6 CONFIGURACIÓN DE LA ALARMA DEL ODÓMETRO

Este menú es equivalente al menú de configuración de la alarma del inclinómetro, excepto que nos permite configurar si la indicación es en metros o centímetros (tecla B) y el rango de valores de la tecla A es de 0 hasta 999,9, ya sean metros o centímetros.

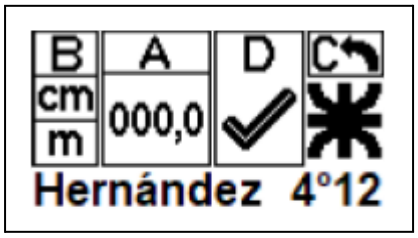

Imagen 14. Menú de configuración de la alarma del odómetro

9.7 MEDICIÓN DEL ODÓMETRO

En este menú se visualiza la distancia medida junto con su respectivo signo y unidad, la tecla B setea en 0 centímetros la actual medición, permitiendo mediciones relativas a determinada distancia, la tecla D mantiene fijada la última medición hasta que se presione nuevamente la tecla, la tecla C nos devuelve al menú anterior, ya sea el de configuración de la alarma o el de selección de alarma.

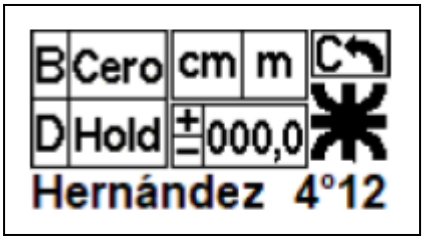

Imagen 15. Menú de medición del odómetro

10 BIBLIOGRAFÍA

[1] Datasheet LPC845 [En línea]. Disponible en: https://www.nxp.com/docs/en/data-sheet/LPC84x.pdf

[2] Datasheet teclado matricial de membrana 4x4 [En línea]. Disponible en: https://agelectronica.lat/pdfs/textos/O/OKY0272.PDF

[3] Datasheet encoder rotativo ky-040 [En línea]. Disponible en: https://www.rcscomponents.kiev.ua/datasheets/ky-040-datasheet.pdf?srsltid=AfmBOorRnsCTJoi_xenRCBnXKT4ObjUHna_cqFSUPBoE69l3MWEk8xTK

[4] Datasheet BC548C [En línea]. Disponible en: https://www.openhacks.com/uploadsproductos/bc548.pdf

[5] Datasheet SSD1306 (OLED) [En línea]. Disponible en: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf

[6] Display OLED por I^2C [En línea]. Disponible en: https://community.nxp.com/t5/LPC-Microcontrollers/LPC845-SSD1306-IIC-I2C-APPLICATION-EXAMPLE/m-p/1209656

[7] Datasheet ADXL345 [En línea]. Disponible en: https://www.analog.com/media/en/technical-documentation/data-sheets/adxl345.pdf

[8] Datasheet LM2596 [En línea]. Disponible en: https://www.ti.com/lit/ds/symlink/lm2596.pdf?ts=1733789428200&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fes-mx%252FLM2596

[9] Documentación SDK [En línea]. Disponible en: https://mcuxpresso.nxp.com/api_doc/dev/116/modules.html

[10] Conversor de imagen a bytes [En línea]. Disponible en: https://javl.github.io/image2cpp/

ANEXO I: ESQUEMÁTICO

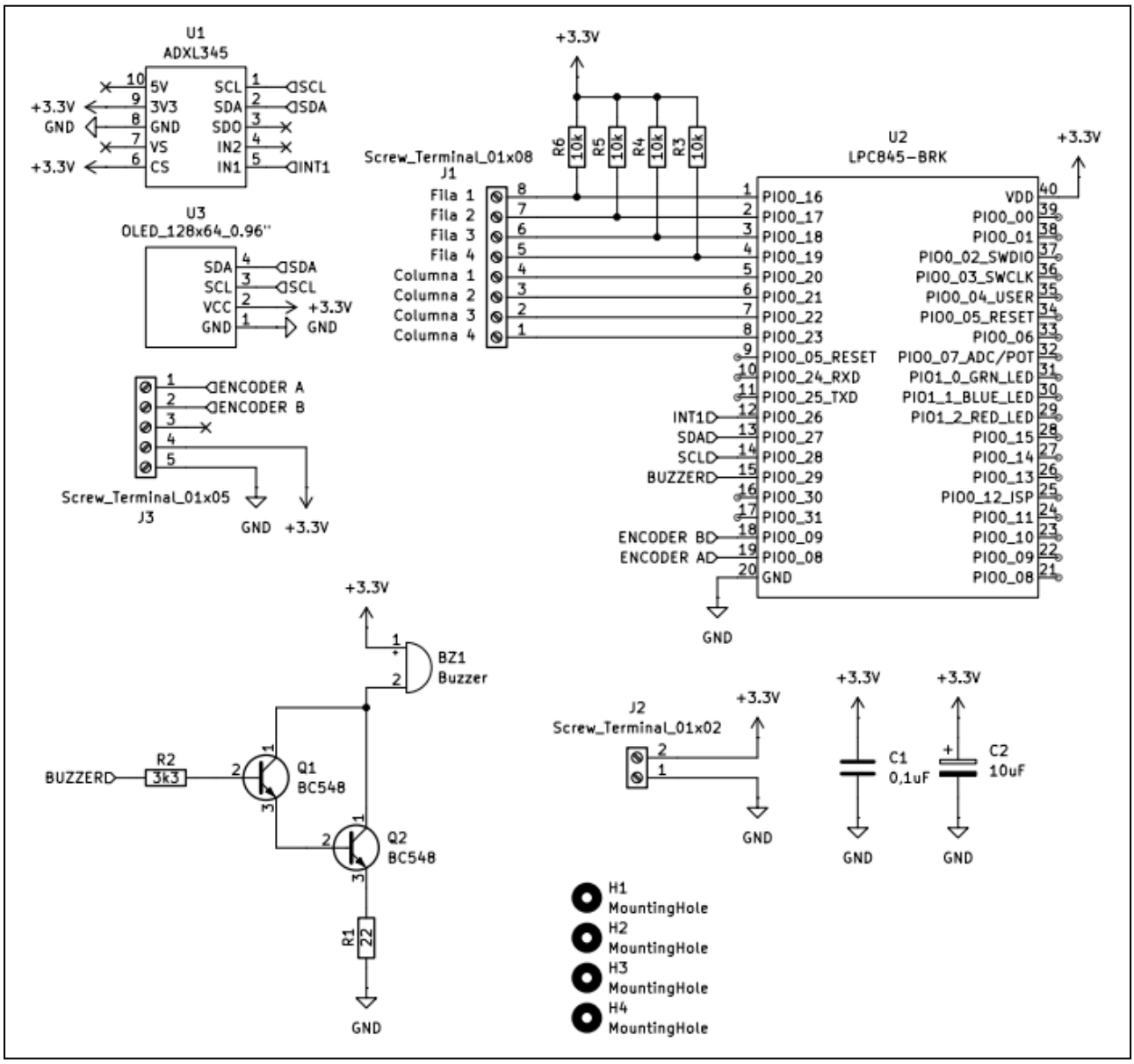

Figura 10. Esquemático del proyecto

ANEXO II: PCB IMPRESO

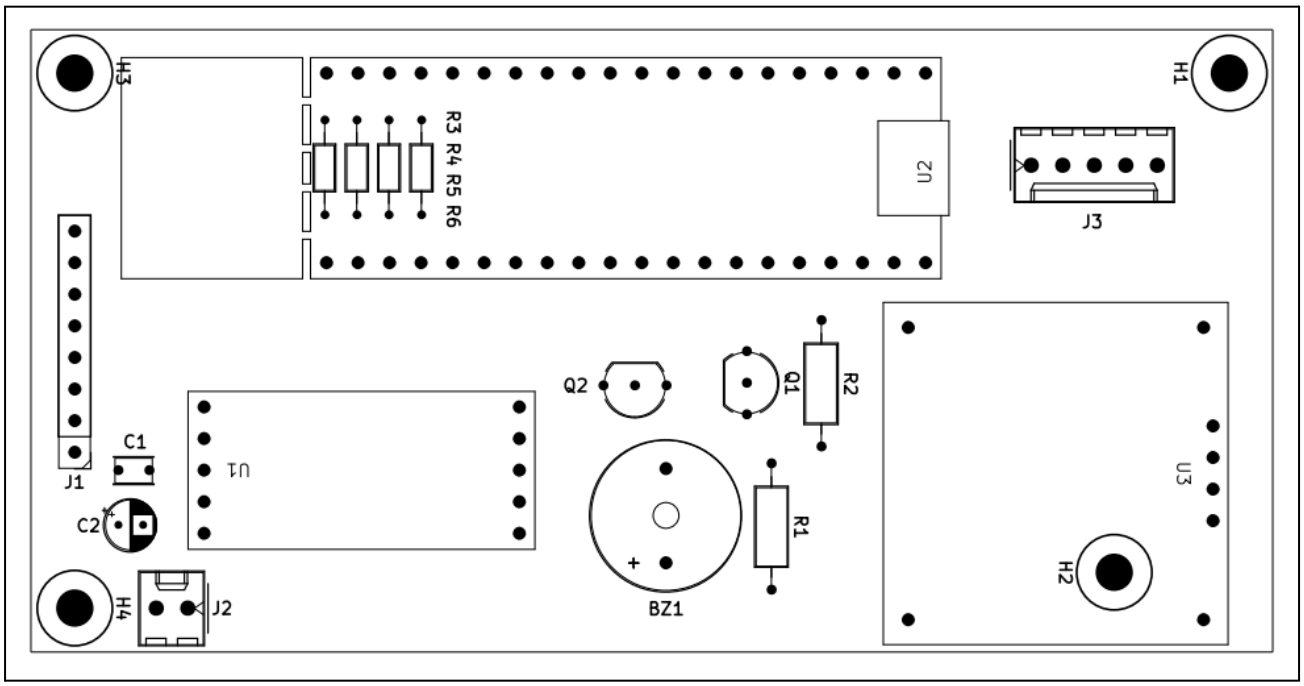

Figura 11. Disposición de componentes en la placa

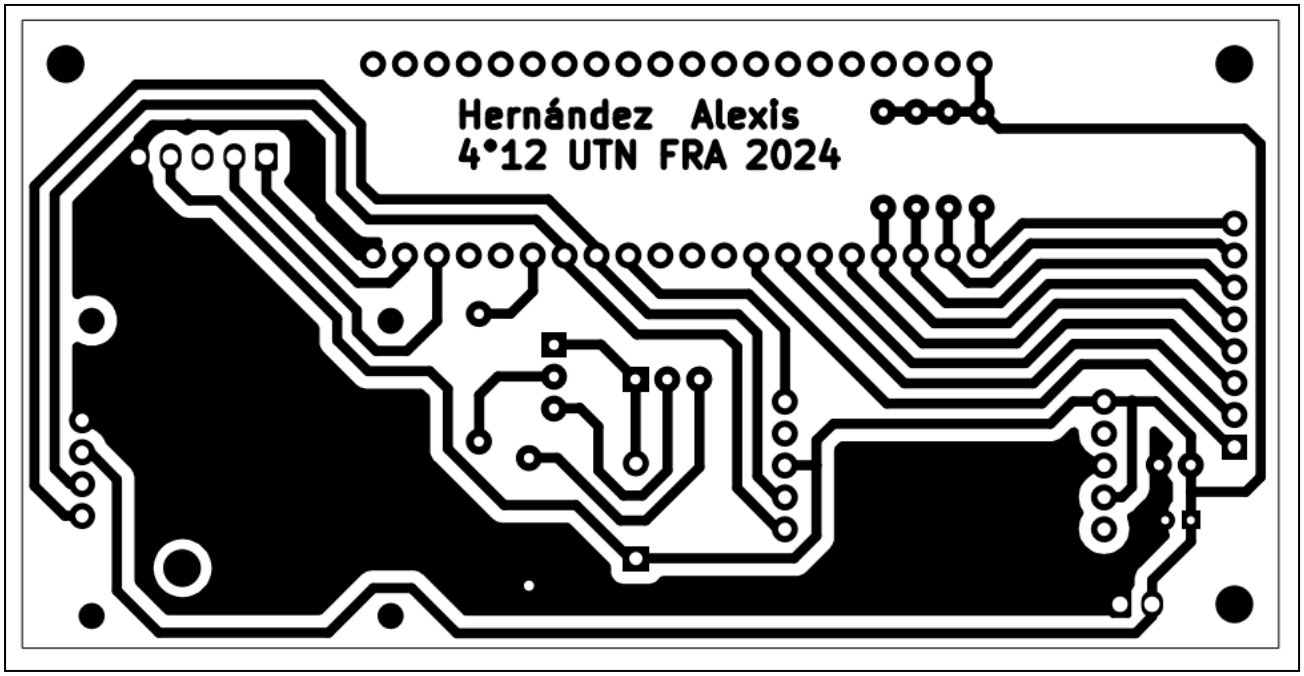

Figura 12. Disposición de las pistas

ANEXO III: PLACA TERMINADA

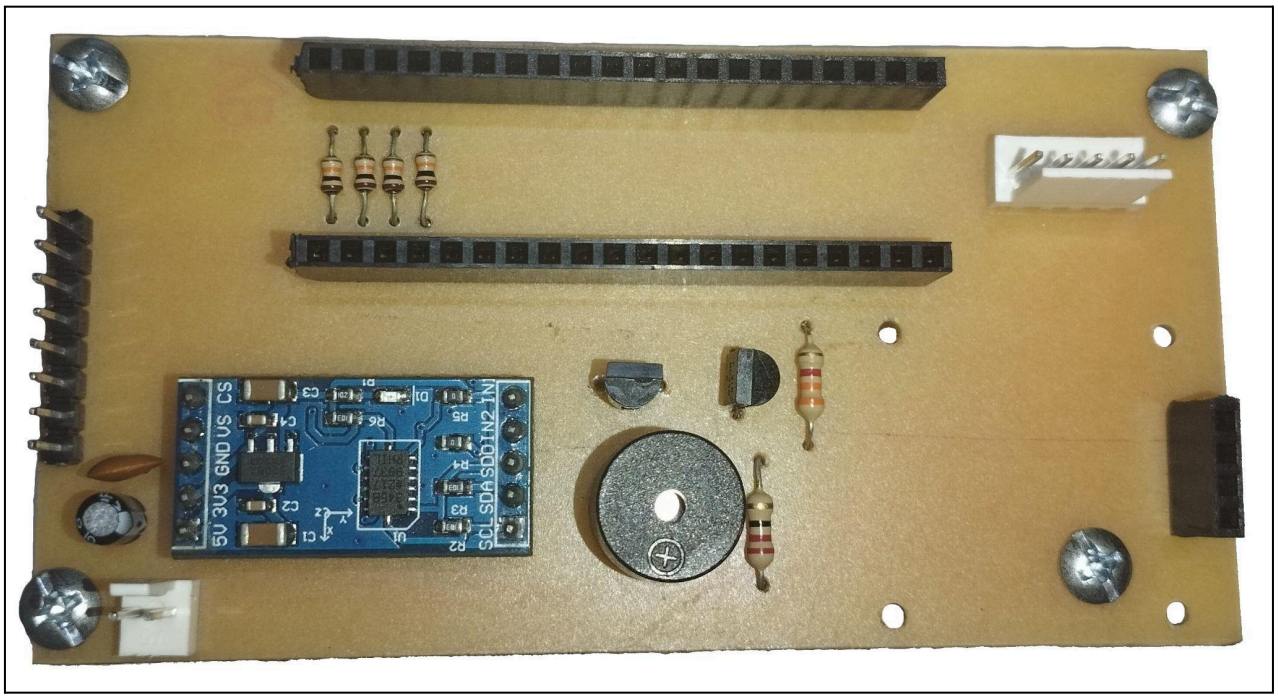

Imagen 16. Placa terminada sin módulos

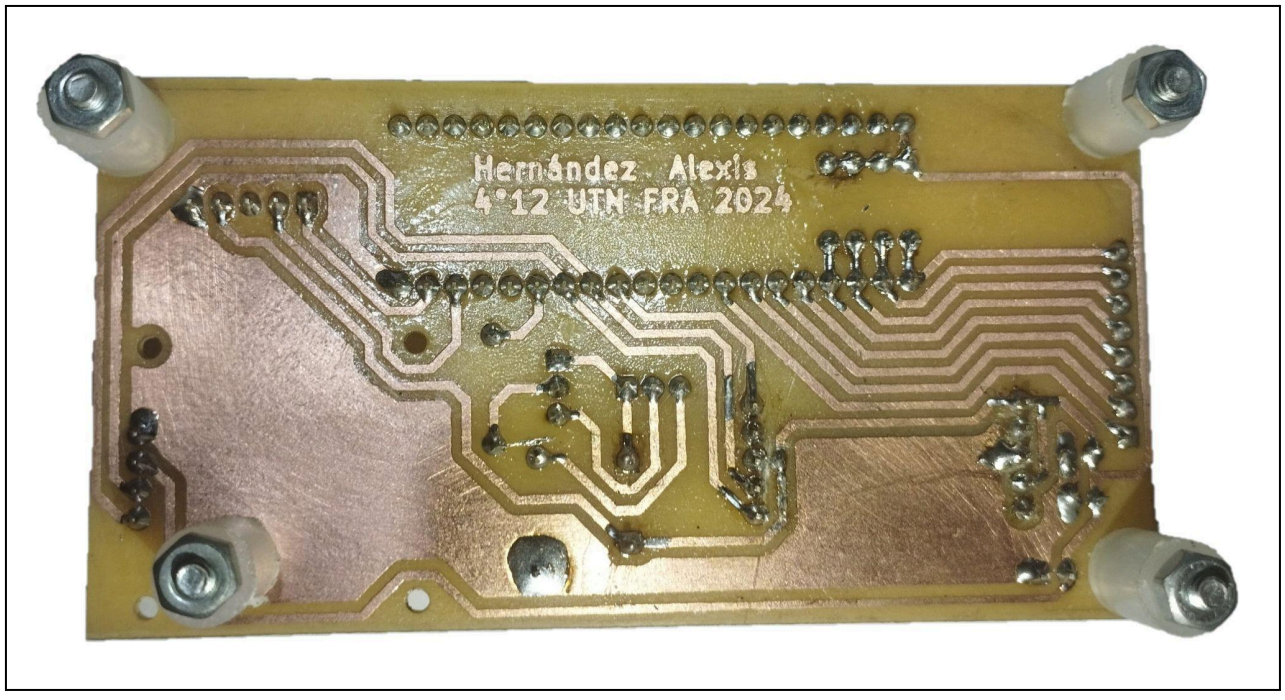

Imagen 17. Placa terminada lado del cobre

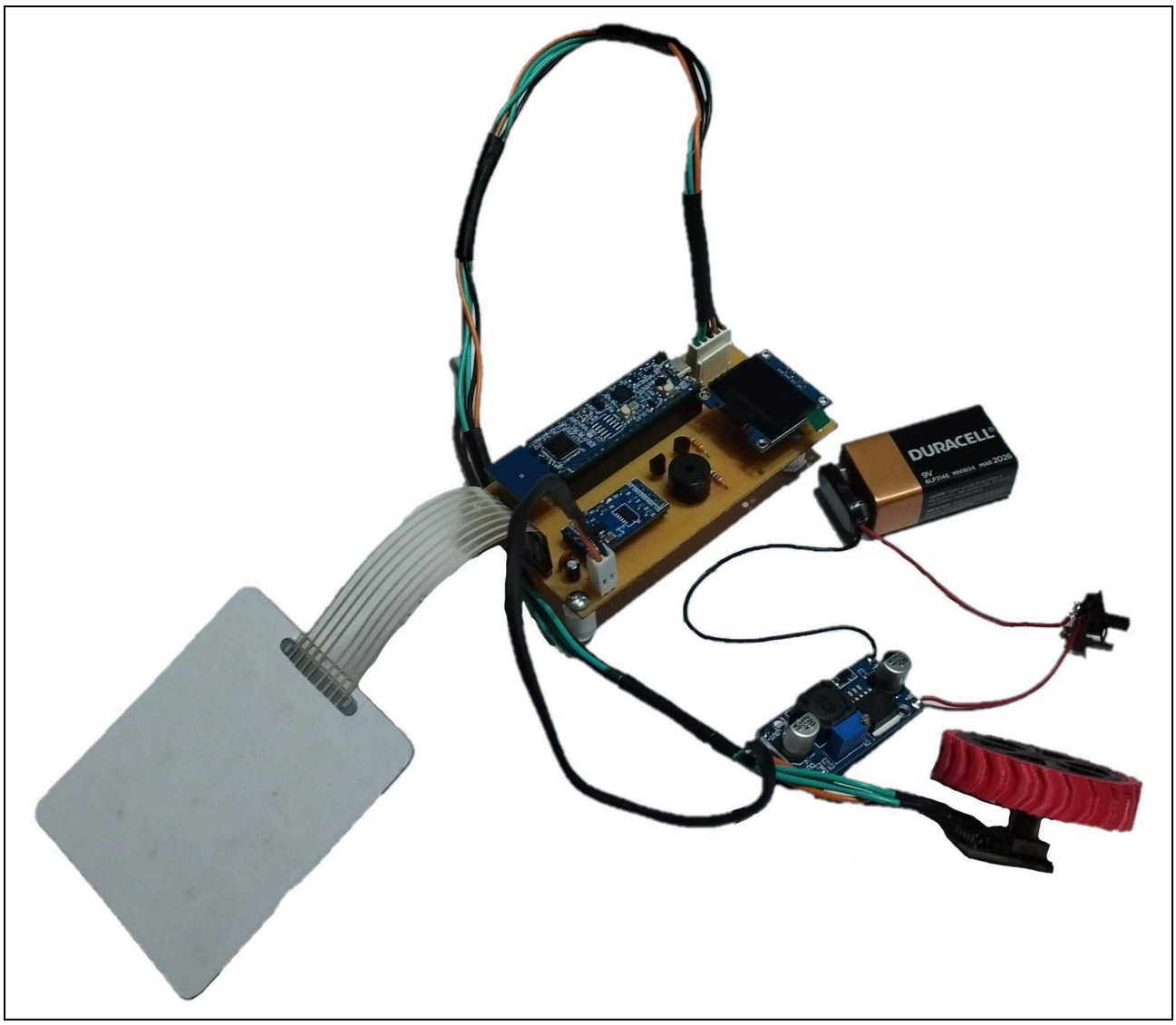

Imagen 18. Proyecto completo

ANEXO IV: DIAGRAMA DE FLUJO

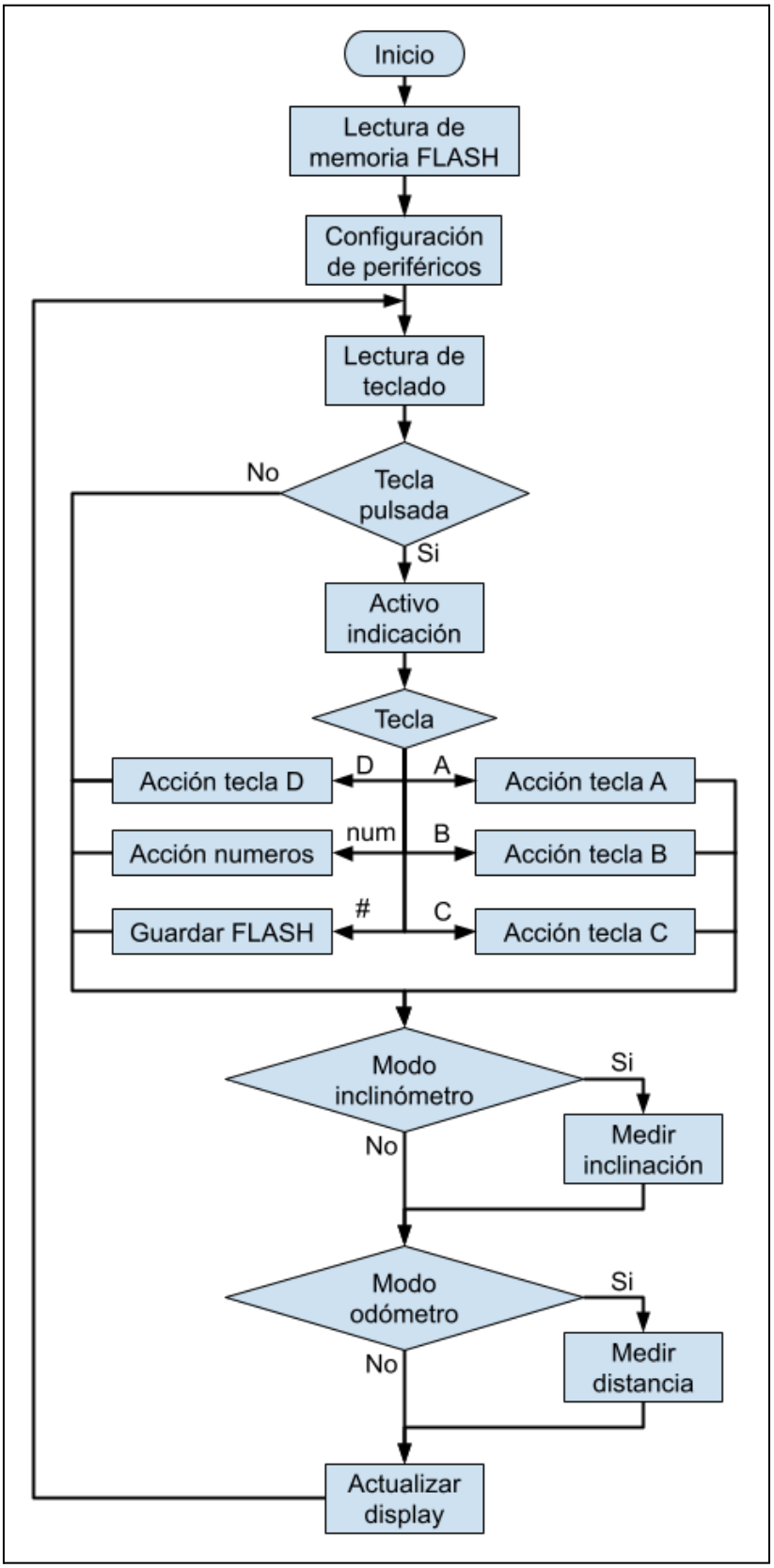

Figura 13. Diagrama de flujo

ANEXO V: CÓDIGO

El directorio del proyecto es el siguiente. Se utilizará para enumerar los distintos códigos.

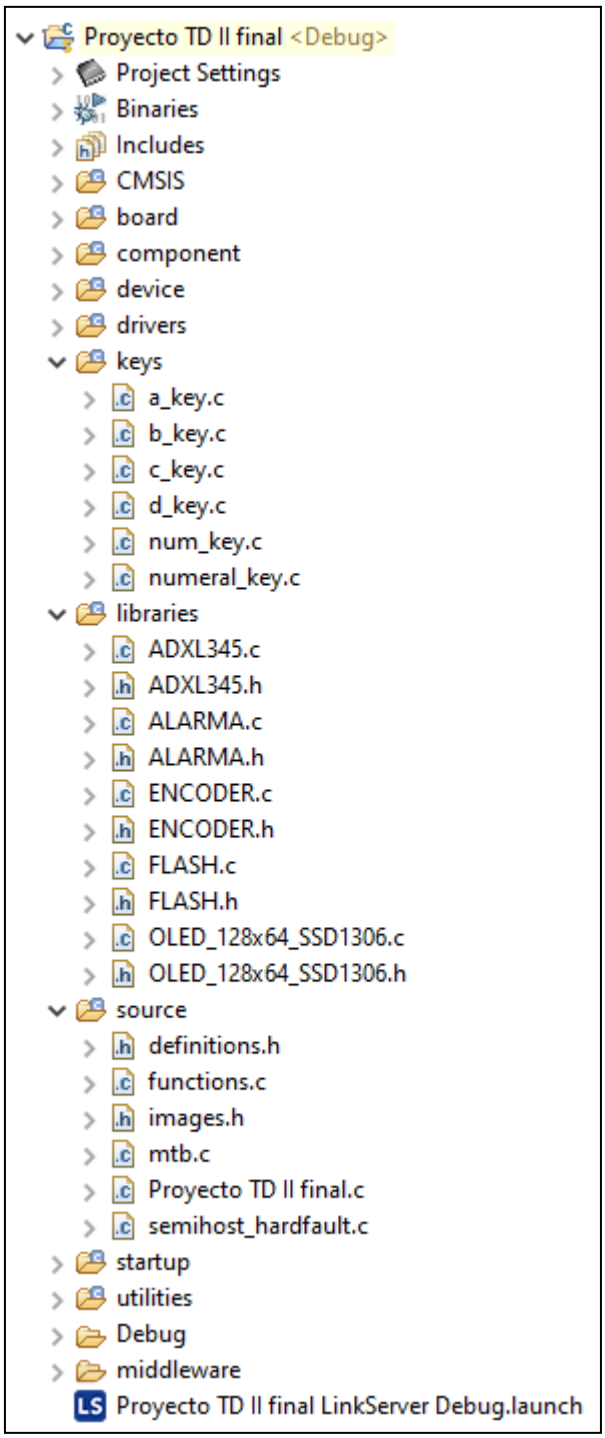

Figura 14. Directorio del proyecto

CÓDIGO ADXL345

ADXL345.h

```
#ifndef ADXL345_H_
#define ADXL345_H_

#include "fsl_gpio.h"
#include "fsl_i2c.h"
#include <math.h>

#define PI 3.14159265359

#define ADDRESS_ADXL345 0x53 // El pin SDO está en bajo

#define INT_ENABLE 0x2E
#define BW_RATE 0x2C
#define DATA_FORMAT 0x31
#define OFSZ 0x20
#define POWER_CTL 0x2D

#define ADXL345_ON 0x08
#define ADXL345_OFF 0x00

#define PUERTO_0 0 // Puerto donde esta el pin de interrupción
#define INT_ADXL345 26 // Pin INT0 del ADXL345

typedef struct { // Estructura que almacena los datos del acelerometro
	float gravedades_x[10];
	float gravedades_y[10];
	float gravedades_z[10];
	float suma_gx, suma_gy , suma_gz;
	uint8_t indice;
} gravedad_t;

void ADXL345_Comando (uint8_t direccionRegistro, uint8_t valor);
void ADXL345_Configuracion (void);
void ADXL345_Encender (gravedad_t *gravedad);
void ADXL345_Apagar (void);
int16_t ADXL345_Angulo (gravedad_t *gravedad);

#endif /* ADXL345_H_ */
```

ADXL345.c

```
#include "ADXL345.h"

void ADXL345_Comando (uint8_t direccionRegistro, uint8_t valor) {
	i2c_master_transfer_t masterXfer;

	uint8_t data[2];
	data[0] = direccionRegistro; // Dirección del registro
	data[1] = valor;		// Valor a escribir

	masterXfer.slaveAddress = ADDRESS_ADXL345;
	masterXfer.direction = kI2C_Write;
	masterXfer.subaddress = 0;
	masterXfer.subaddressSize = 0;
	masterXfer.data = data;
	masterXfer.dataSize = sizeof(data);
	masterXfer.flags = kI2C_TransferDefaultFlag;
```

```
	I2C_MasterTransferBlocking(I2C1, &masterXfer);
}

void ADXL345_Configuracion (void) {
	ADXL345_Comando (INT_ENABLE, 0x80); // Interrupción data ready activada en INT1
	ADXL345_Comando (BW_RATE, 0x09); // 50Hz
	ADXL345_Comando (DATA_FORMAT, 0x00); // Formato
	ADXL345_Comando (OFSZ, 50); // Offset eje Z

	gpio_pin_config_t in_config = {kGPIO_DigitalInput};

	//GPIO_PortInit (GPIO, PUERTO_0); // Ya inicializado en otro lado
	GPIO_PinInit (GPIO, PUERTO_0, INT_ADXL345, &in_config);
}

void ADXL345_Encender (gravedad_t *gravedad) {
	ADXL345_Comando (POWER_CTL, ADXL345_ON);
	for (uint8_t i = 0; i < 10; i ++) {
		gravedad->gravedades_x[i] = 0;
		gravedad->gravedades_y[i] = 0;
		gravedad->gravedades_z[i] = 0;
	}
	gravedad->suma_gx = 0;
	gravedad->suma_gy = 0;
	gravedad->suma_gz = 0;
	gravedad->indice = 0;
}

void ADXL345_Apagar (void) {
	ADXL345_Comando (POWER_CTL, ADXL345_OFF);
}

int16_t ADXL345_Angulo (gravedad_t *gravedad) {
	int16_t x, y, z, angulo;
	float gx, gy, gz;
	uint8_t lecturas[] = {0, 0, 0, 0, 0, 0};
	uint8_t REGISTRO_DATAX0 = 0x32;

	I2C_MasterStart (I2C1, ADDRESS_ADXL345, kI2C_Write);
	I2C_MasterWriteBlocking (I2C1, &REGISTRO_DATAX0, 1, 0);
	I2C_MasterRepeatedStart (I2C1, ADDRESS_ADXL345, kI2C_Read);
	I2C_MasterReadBlocking (I2C1, &lecturas[0], 6, 0);

	x = (((int16_t)lecturas[1]) << 8 | lecturas[0]);
	gx = (float) x / 256;
	y = (((int16_t)lecturas[3]) << 8 | lecturas[2]);
	gy = (float) y / 256;
	z = (((int16_t)lecturas[5]) << 8 | lecturas[4]);
	gz = (float) z / 256;
	gravedad->suma_gx -= gravedad->gravedades_x[gravedad->indice];
	gravedad->gravedades_x[gravedad->indice] = gx;
	gravedad->suma_gx += gravedad->gravedades_x[gravedad->indice];
	gravedad->suma_gy -= gravedad->gravedades_y[gravedad->indice];
	gravedad->gravedades_y[gravedad->indice] = gy;
	gravedad->suma_gy += gravedad->gravedades_y[gravedad->indice];
	gravedad->suma_gz -= gravedad->gravedades_z[gravedad->indice];
	gravedad->gravedades_z[gravedad->indice] = gz;
	gravedad->suma_gz += gravedad->gravedades_z[gravedad->indice];
	gravedad->indice += 1;
	if (gravedad->indice == 10) gravedad->indice = 0;

	angulo = (int16_t) (acos ((gravedad->suma_gz / 10) /
			sqrt (pow (gravedad->suma_gx / 10, 2) + pow (gravedad->suma_gy / 10, 2) +
			pow (gravedad->suma_gz / 10, 2))) * 10 * (180 / PI));
```

```
        if (gravedad->suma_gx / 10 < 0) return -angulo;
        else return angulo;
}
```

CÓDIGO ALARMA

ALARMA.h

```
#ifndef ALARMA_H_
#define ALARMA_H_

#include "fsl_sctimer.h"
#include "fsl_dac.h"
#include "fsl_power.h"
#include "fsl_swm.h"

#define TIEMPO_INDICACION 50 // Tiempo en ms de la indicación

void SCTIMER_Inicializacion (void); // Inicializa el SCTimer
void DAC_Inicializacion (void); // Inicializa el DAC
void INDICADOR_Enciendo (uint8_t mseg, uint8_t brillo); // Enciende la indicación mseg ms
void INDICADOR_Alarma (uint8_t mseg_on, uint16_t mseg_off, uint8_t brillo); // Configuración de la alarma

extern uint32_t evento_pwm, evento_periodo; // Guardan el numero de los eventos
extern volatile uint16_t flag_alarma; // Flag para el tiempo de la alarma
extern volatile uint8_t DAC_on, flag_buzzer; // Guarda el estado del DAC, 0: apagado, 1: encendido, 2: desactivado

/*
Codigo en el main para que funcione:

        ...
        INDICADOR_Enciendo (TIEMPO_INDICACION, 99); // Para las teclas
        ...
        INDICADOR_Alarma (TIEMPO_INDICACION, TIEMPO_INDICACION * 11, 99) // Para alarma muy cerca
        ...

void SCT0_IRQHandler(void) { // Interrupción del SCTimer
        if (SCTIMER_GetStatusFlags (SCT0) & (1 << evento_periodo)) {
                SCTIMER_ClearStatusFlags (SCT0, 1 << evento_periodo);
                if (DAC_on) {
                        DAC_SetBufferValue (DAC1, 0);
                        DAC_on = 0;
                }
                else {
                        DAC_SetBufferValue (DAC1, (volumen == 0) ? 0 : (410 + volumen * 31 / 14));
                        DAC_on = 1;
                }
        }
}
*/

#endif /* ALARMA_H_ */
```

ALARMA.c

```
#include "ALARMA.h"

uint32_t evento_pwm, evento_periodo;
volatile uint16_t flag_alarma = 0;
volatile uint8_t DAC_on = 2, flag_buzzer = 0;

void SCTIMER_Inicializacion (void) {
```

```
	sctimer_pwm_signal_param_t pwmParam;
	sctimer_config_t sctimerInfo;
	SCTIMER_GetDefaultConfig (&sctimerInfo);
	SCTIMER_Init (SCT0, &sctimerInfo);
	pwmParam.output = kSCTIMER_Out_4;
	pwmParam.level = kSCTIMER_LowTrue;
	pwmParam.dutyCyclePercent = 99;
	SCTIMER_SetupPwm (SCT0, &pwmParam, kSCTIMER_CenterAlignedPwm, 1600, CLOCK_GetFreq
			(kCLOCK_Fro), &evento_pwm);
	SCTIMER_StopTimer (SCT0, kSCTIMER_Counter_U);
	SCT0->OUTPUT |= (1 << 4);
	CLOCK_EnableClock (kCLOCK_Swm);
	SWM_SetMovablePinSelect (SWM0, kSWM_SCT_OUT4, kSWM_PortPin_P1_0);
	CLOCK_DisableClock (kCLOCK_Swm);
	SCTIMER_CreateAndScheduleEvent (SCT0, kSCTIMER_OutputRiseEvent, 0, 4, kSCTIMER_Counter_U,
			&evento_periodo);
	SCTIMER_EnableInterrupts (SCT0, 1 << evento_periodo);
	EnableIRQ(SCT0_IRQn);
}

void DAC_Inicializacion (void) {
	dac_config_t configuracion_dac;
	CLOCK_EnableClock (kCLOCK_Swm);
	SWM_SetFixedPinSelect (SWM0, kSWM_DAC_OUT1, true);
	CLOCK_DisableClock (kCLOCK_Swm);
	POWER_DisablePD (kPDRUNCFG_PD_DAC1);
	configuracion_dac.settlingTime = kDAC_SettlingTimeIs25us;
	DAC_Init (DAC1, &configuracion_dac);
}

void INDICADOR_Enciendo (uint8_t mseg, uint8_t brillo) {
	SCTIMER_UpdatePwmDutycycle (SCT0, kSCTIMER_Out_4, brillo, evento_pwm);
	flag_buzzer = mseg;
	DAC_on = 0;
}

void INDICADOR_Alarma (uint8_t mseg_on, uint16_t mseg_off, uint8_t brillo) {
	if (flag_alarma == 0) {
		INDICADOR_Enciendo (TIEMPO_INDICACION, brillo); // Enciendo 50ms
		flag_alarma = mseg_off;
	}
}
```

CÓDIGO ENCODER

ENCODER.h

```
#ifndef ENCODER_H_
#define ENCODER_H_

#include "fsl_gpio.h"
#include "fsl_pint.h"
#include "fsl_syscon.h"

#define PUERTO_0 0 // Puerto donde estan los pines del encoder
#define ENCODER_A 9 // Pin encoder A, en la placa es el clk
#define ENCODER_B 8 // Pin encoder B, en la placa es el dt

#define UNIDAD_DISTANCIA 5 // Cuanta distancia equivale un pulso

void ENCODER_Inicializacion (void); // Inicializa los pines y configura las interrupciones
void ENCODER_Activo (void); // Habilito la interrupción
void ENCODER_Desactivo (void); // Desactivo la interrupción
```

```
void ENCODER_Interrupcion (pint_pin_int_t pintr, uint32_t pmatch_status); // Interrupción del pin A
int8_t ENCODER_Lectura (void); // Devuelve el numero que suma o resta a la distancia

extern volatile uint8_t antirrebote_encoder; // Flag para el antirrebote
extern volatile uint8_t B; // Valor del pin B

/*
Codigo en el main para que funcione:

		...
		if (antirrebote_encoder == 1) {
			distancia += ENCODER_Lectura;
		}
		...

void SysTick_Handler (void) { // Interrupción del SysTick
		if (antirrebote_encoder != 0) antirrebote_encoder --;
}
*/

#endif /* ENCODER_H_ */
```

ENCODER.c

```
#include "ENCODER.h"

volatile uint8_t antirrebote_encoder = 0, B = 0;

void ENCODER_Inicializacion (void) {
		gpio_pin_config_t in_config = {kGPIO_DigitalInput};

		//GPIO_PortInit (GPIO, PUERTO_0); // Ya inicializado en otro lado
		GPIO_PinInit (GPIO, PUERTO_0, ENCODER_A, &in_config);
		GPIO_PinInit (GPIO, PUERTO_0, ENCODER_B, &in_config);

		SYSCON_AttachSignal (SYSCON, kPINT_PinInt0, kSYSCON_GpioPort0Pin9ToPintsel);
		PINT_Init (PINT);
		PINT_PinInterruptConfig (PINT, kPINT_PinInt0, kPINT_PinIntEnableBothEdges, ENCODER_Interrupcion);
		PINT_DisableCallback (PINT); // Desactiva interrupciones para encoder
}

void ENCODER_Activo (void) {
		PINT_EnableCallback (PINT);
}

void ENCODER_Desactivo (void) {
		PINT_DisableCallback (PINT);
}

void ENCODER_Interrupcion (pint_pin_int_t pintr, uint32_t pmatch_status) {
		if (antirrebote_encoder == 0) {
			antirrebote_encoder = 6;
			B = GPIO_PinRead (GPIO, PUERTO_0, ENCODER_B);
		}
}

int8_t ENCODER_Lectura (void) {
		uint8_t A;
		A = GPIO_PinRead (GPIO, PUERTO_0, ENCODER_A);
		antirrebote_encoder = 0;
		if (A == B) return -UNIDAD_DISTANCIA;
		else return UNIDAD_DISTANCIA;
}
```

CÓDIGO FLASH

FLASH.h

```
#ifndef FLASH_H_
#define FLASH_H_

#include "fsl_iap.h"

#define SECTOR 63
#define FLASH_ADDRESS_BASE 0x0000FC00
#define LONGITUD_DATO 64

int8_t FLASH_Busqueda (void);
void FLASH_Lectura (uint8_t *contraste, uint8_t *volumen, uint8_t *version,
                    int16_t *angulo_alarma, int32_t *distancia_alarma);
void FLASH_Guardado (uint8_t contraste, uint8_t volumen, uint8_t *version,
                     int16_t angulo_alarma, int32_t distancia_alarma);

#endif /* FLASH_H_ */
```

FLASH.c

```
#include "FLASH.h"

int8_t FLASH_Busqueda (void) {
	uint8_t n = 0, version;
	uint32_t *memoria;
	while (n < 16) {
		memoria = (uint32_t *) (FLASH_ADDRESS_BASE + LONGITUD_DATO * n + 0x4 * 4);
		version = *memoria;
		if (version == 0xFF) break;
		n ++;
	}
	return n - 1;
}

void FLASH_Lectura (uint8_t *contraste, uint8_t *volumen, uint8_t *version,
                    int16_t *angulo_alarma, int32_t *distancia_alarma) {
	uint32_t *memoria;
	if (FLASH_Busqueda () != -1) {
		memoria = (uint32_t *) (FLASH_ADDRESS_BASE + LONGITUD_DATO * FLASH_Busqueda ());
		for (uint8_t i = 0; i < 5; i ++) {
			switch (i) {
			case 0: *contraste = *memoria;
				break;
			case 1: *volumen = *memoria;
				break;
			case 2: *angulo_alarma = *memoria;
				break;
			case 3: *distancia_alarma = *memoria;
				break;
			case 4: *version = *memoria;
				break;
			}
			memoria ++;
		}
	}
	else {
		FLASH_Guardado (*contraste, *volumen, &(*version), *angulo_alarma, *distancia_alarma);
	}
}
```

```
void FLASH_Guardado (uint8_t contraste, uint8_t volumen, uint8_t *version,
                     int16_t angulo_alarma, int32_t distancia_alarma) {
        uint32_t datos[5];
        datos[0] = contraste;
        datos[1] = volumen;
        datos[2] = angulo_alarma;
        datos[3] = distancia_alarma;
        (*version) ++;
        if (*version == 16) {
                *version = 0;
                // Borrado flash
                if (IAP_PrepareSectorForWrite (SECTOR, SECTOR) != kStatus_IAP_Success) {
                        // Error
                }
                if (IAP_EraseSector (SECTOR, SECTOR, 12000000) != kStatus_IAP_Success) {
                        // Error
                }
        }
        datos[4] = *version;
        // Escritura flash
        if (IAP_PrepareSectorForWrite (SECTOR, SECTOR) != kStatus_IAP_Success) {
                // Errpr
        }
        status_t resp = IAP_CopyRamToFlash (FLASH_ADDRESS_BASE + LONGITUD_DATO * datos[4], &datos, 64,
                                            12000000);
        if (resp != kStatus_IAP_Success) {
                // Error
        }
}
```

CÓDIGO OLED_128x64_SSD1306

OLED_128x64_SSD1306.h

```
/*
* Se definen todos los comandos, el addres, el ancho y alto y los prototipos de las funciones
* del display OLED 128x64 con SSD1306
* Se incluye el archivo con la fuente de 7x7 pixeles
* Inspirado en fsl_SSD1306_I2C.h de
* https://community.nxp.com/t5/LPC-Microcontrollers/LPC845-SSD1306-IIC-I2C-APPLICATION-EXAMPLE/m-p/1209656
*/

#ifndef OLED_128X64_SSD1306_H_
#define OLED_128X64_SSD1306_H_

#include <math.h>
#include "fsl_common.h"

/*******************************************************************************************************************************
* Comandos del SSD1306
*******************************************************************************************************************************/
#define OLED_SETCONTRAST                                0x81
#define OLED_ENTIREDISPLAYON_OFF                        0xA4
#define OLED_ENTIREDISPLAYON_ON                         0xA5
#define OLED_SETNORMALDISPLAY                           0xA6
#define OLED_SETINVERSEDISPLAY                          0xA7
#define OLED_SETDISPLAYOFF                              0xAE
#define OLED_SETDISPLAYON                               0xAF
#define OLED_SETDISPLAYOFFSET                           0xD3
#define OLED_SETCOMPINS                                 0xDA
#define OLED_SETVCOMLEVEL                               0xDB
#define OLED_SETDISPLAYCLOCKDIVOSCILLATORFREQUENCY      0xD5
#define OLED_SETPRECHARGEPERIOD                         0xD9
```

```
#define OLED_SETMULTIPLEXRATIO                  0xA8
#define OLED_SETLOWERCOLUMN                     0x00
#define OLED_SETHIGHERCOLUMN                    0x10
#define OLED_SETDISPLAYSTARTLINE                0x40
#define OLED_SETMEMORYADDRESSINGMODE            0x20
#define OLED_SETCOLUMNADDRESS                   0x21
#define OLED_SETPAGEADDRESS                     0x22
#define OLED_COMSCANTOPLOW                      0xC0
#define OLED_COMSCANLOWTOP                      0xC8
#define OLED_SETSEGMENTREMAP                    0xA0
#define OLED_SETCHARGEPUMP                      0x8D
#define OLED_ACTIVATE_SCROLL                    0x2F
#define OLED_DEACTIVATE_SCROLL                  0x2E
#define OLED_SET_VERTICAL_SCROLL_AREA           0xA3
#define OLED_RIGHT_HORIZONTAL_SCROLL            0x26
#define OLED_LEFT_HORIZONTAL_SCROLL             0x27
#define OLED_VERTICAL_AND_RIGHT_HORIZONTAL_SCROLL 0x29
#define OLED_VERTICAL_AND_LEFT_HORIZONTAL_SCROLL  0x2A
/*****************************************************************************************************************************
* Address del SSD1306
*****************************************************************************************************************************/
#define SSD1306_ADDRESS                         0x3C
/*****************************************************************************************************************************
* Ancho y alto del display
*****************************************************************************************************************************/
#define OLED_WIDTH 128
#define OLED_HEIGHT 64
/*****************************************************************************************************************************
* Selección de enceder/apagar display
*****************************************************************************************************************************/
enum _OLED_Pixel {
        Apagar_Pixel,
        Encender_Pixel,
};
/*****************************************************************************************************************************
* Prototipos de las funciones
*****************************************************************************************************************************/

/*
* Inicialización del display
* No tiene parametros
*/
void OLED_Inicio (void);

/*
* Cambia el contraste del display
* Tiene 1 parametro:
* Contraste: valor a cambiar
*/
void OLED_Contraste (uint8_t contraste);

/*
* Refresco del display
* Envia a pantalla el buffer de datos
* No tiene parametros
*/
void OLED_Refresco (void);

/*
* Borra el buffer
* Llena el buffer con todos ceros
* No tiene parametros
*/
void OLED_BorrarBuffer (void);
```

```
/*
* Setea un pixel en especifico
* Tiene 3 parametros:
* X: coordenada en eje X
* Y: coordenada en eje Y
* Estado: encender o apagar pixel
*/
void OLED_SetPixel (uint8_t X, uint8_t Y, uint8_t Estado);

/*
* Cambia el estado de un pixel
* Tiene 2 parametros:
* X: coordenada en eje X
* Y: coordenada en eje Y
*/
void OLED_TogglePixel (uint8_t X, uint8_t Y);

/*
* Escribe un numero en el buffer
* Tiene 4 parametros:
* X: coordenada en el eje X
* Y: coordenada en el eje Y
* numero: numero a escribir
* Estado: encender o apagar pixeles
*/
void OLED_EscribirNumero (uint8_t X, uint8_t Y, uint8_t numero, uint8_t Estado);

/*
* Escribe en un rectangulo relleno del Buffer
* Tiene 5 parametros:
* X: coordenada en eje X
* Y: coordenada en eje Y
* Ancho: ancho del rectangulo
* Alto: alto del rectangulo
* Estado: encender o apagar pixeles
*/
void OLED_RectanguloRelleno (uint8_t X, uint8_t Y, uint8_t Ancho, uint8_t Alto, uint8_t Estado);

/*
* Escribe en un rectangulo vacio del Buffer
* Tiene 5 parametros:
* X: coordenada en eje X
* Y: coordenada en eje Y
* Ancho: ancho del rectangulo
* Alto: alto del rectangulo
* Estado: encender o apagar pixeles
*/
void OLED_RectanguloVacio (uint8_t X, uint8_t Y, uint8_t Ancho, uint8_t Alto, uint8_t Estado);

/*
* Cambia el estado de los pixeles de un rectangulo
* Tiene 4 parametros:
* X_Ini: punto inicial del rectangulo en X
* Y_Ini: punto inicial del rectangulo en Y
* X_Fin: punto final del rectangulo en X
* Y_Fin: punto final del rectangulo en Y
*/
void OLED_ToggleRectangulo (uint8_t X_Ini, uint8_t Y_Ini, uint8_t X_Fin, uint8_t Y_Fin);

/*
* Dibuja una recta en el Buffer
* Tiene 5 parametros:
* X_Ini: posición inicial en X
* Y_Ini: posición inicial en Y
```

```
* X_Fin: posición final en X
* Y_Fin: posición final en Y
* Estado: encender o apagar pixeles
*/
void OLED_DibujarRecta (uint8_t X_Ini, uint8_t Y_Ini, uint8_t X_Fin, uint8_t Y_Fin, uint8_t Estado);

/*
* Dibuja un archo de circunferencia en el Buffer
* Tiene 6 parametros:
* X: centro de arco en eje x
* Y: centro de arco en eje y
* Radio: radio del arco
* Ang_Ini: angulo inicial
* Ang_Fin: angulo final
* Estado: encender o apagar pixeles
*/
void OLED_DibujarArco (uint8_t X, uint8_t Y, uint8_t Radio, int16_t Ang_Ini_Por_Diez, int16_t Ang_Fin_Por_Diez,
                       uint8_t Estado);

/*
* Copia una "imagen" en el buffer
* Guarda en el buffer los datos del buffer de la imagen
* Tiene 2 parametros:
* Imagen: contiene la información de todos los pixeles de la imagen
* Tamaño: tamaño de la imagen
*/
void OLED_CopiarImagen (const uint8_t *Imagen, uint16_t Tamaño);

#endif /* OLED_128X64_SSD1306_H_ */
```

OLED_128x64_SSD1306.c

```
#include "../libraries/OLED_128x64_SSD1306.h"

#include "fsl_i2c.h"
#include "LPC845.h"

/*****************************************************************************************************************************
* Buffer del display
* En nuestro caso 128 x 64 = 8192 pixeles, cada 8 bits son
* 8192 / 8 = 1024 bytes
*****************************************************************************************************************************/
static uint8_t OLED_Buffer[(OLED_WIDTH * OLED_HEIGHT) / 8];
/*****************************************************************************************************************************
* Funciones
*****************************************************************************************************************************/
static void OLED_Comando (uint8_t comando) {
	i2c_master_transfer_t xfer = {0};

	xfer.data = (uint8_t *)&comando;
	xfer.dataSize = sizeof(comando);
	xfer.flags = kI2C_TransferDefaultFlag;
	xfer.slaveAddress = SSD1306_ADDRESS;
	xfer.direction = kI2C_Write;
	xfer.subaddress = 0x0; // Indica que es comando
	xfer.subaddressSize = 1;
	I2C_MasterTransferBlocking (I2C1, &xfer);
}

static void OLED_Data (uint8_t *data) {
	i2c_master_transfer_t xfer = {0};

	xfer.data = data;
```

```
	xfer.dataSize = sizeof(OLED_Buffer);
	xfer.flags = kI2C_TransferDefaultFlag;
	xfer.slaveAddress = SSD1306_ADDRESS;
	xfer.direction = kI2C_Write;
	xfer.subaddress = 0x40; // Indica que escribe en RAM
	xfer.subaddressSize = 1;
	I2C_MasterTransferBlocking (I2C1, &xfer);
}

void OLED_Inicio (void) {
	OLED_Comando (OLED_SETDISPLAYOFF); // Apagar display
	OLED_Comando (OLED_SETDISPLAYCLOCKDIVOSCILLATORFREQUENCY); // Seleccionar frecuencia
	OLED_Comando (0x80); // Frecuencia recomendada
	OLED_Comando (OLED_SETMULTIPLEXRATIO); // Elegir multiplexación
	OLED_Comando (OLED_HEIGHT - 1);		// Multiplexación para las 64 lineas
	OLED_Comando (OLED_SETDISPLAYOFFSET); // Configuramos el offset
	OLED_Comando (0x0); // Sin offset
	OLED_Comando (OLED_SETDISPLAYSTARTLINE | 0x0); // Elegimos la linea de inicio (0)
	OLED_Comando (OLED_SETCHARGEPUMP); // Configuramos la alta tensión
	OLED_Comando (0x14); // Usar 3,3V para generar el alto voltaje
	OLED_Comando (OLED_SETMEMORYADDRESSINGMODE); // Elegimos modo de direccionamiento
	OLED_Comando (0x00); // Direccionamiento horizontal
	OLED_Comando (OLED_SETSEGMENTREMAP | 0x1);
	OLED_Comando (OLED_COMSCANLOWTOP);
	OLED_Comando (OLED_SETCOMPINS);
	OLED_Comando (0x12);
	OLED_Comando (OLED_SETCONTRAST); // Ajustamos contraste
	OLED_Comando (206); // Contraste en 206
	OLED_Comando (OLED_SETPRECHARGEPERIOD);
	OLED_Comando (0xF1);
	OLED_Comando (OLED_SETVCOMLEVEL);
	OLED_Comando (0x40);
	OLED_Comando (OLED_ENTIREDISPLAYON_OFF);
	OLED_Comando (OLED_SETNORMALDISPLAY);
	OLED_Comando (OLED_DEACTIVATE_SCROLL);
	OLED_Comando (OLED_SETCOLUMNADDRESS);
	OLED_Comando (0);
	OLED_Comando (OLED_WIDTH - 1);
	OLED_Comando (OLED_SETPAGEADDRESS);
	OLED_Comando (0);
	OLED_Comando (OLED_HEIGHT / 8 - 1);
	OLED_Comando (OLED_SETDISPLAYON);
	OLED_Comando (OLED_SETLOWERCOLUMN | 0x0);
	OLED_Comando (OLED_SETHIGHERCOLUMN | 0x0);
	OLED_Comando (OLED_SETDISPLAYSTARTLINE | 0x0);
	OLED_BorrarBuffer (); // Vaciamos el buffer
}

void OLED_Contraste (uint8_t contraste) {
	OLED_Comando (OLED_SETCONTRAST); // Ajustamos contraste
	OLED_Comando (contraste);
}

void OLED_Refresco (void) {
	OLED_Comando (0xB0); // Pagina 0
	OLED_Comando (((0 & 0xF0) >> 4) | 0x10); // Parte alta de la columna
	OLED_Comando ((0 & 0x0f) | 0x00); // Parte baja de la columna
	OLED_Data (&OLED_Buffer[0]); // Mostramos el buffer en pantalla
}

void OLED_BorrarBuffer (void) {
	memset (OLED_Buffer, 0, sizeof(OLED_Buffer)); // 0 en el buffer
}
```

```
void OLED_SetPixel (uint8_t X, uint8_t Y, uint8_t Estado) {
	// Verificamos que se encuentra en el rango del display
	if(X >= OLED_WIDTH || Y >= OLED_HEIGHT) {
		// No hacer nada
	}
	else {
		switch(Estado) {
			case Apagar_Pixel:
				// Colocamos 0 en la posición del buffer
				OLED_Buffer [X + (Y / 8) * OLED_WIDTH] &= ~(1 << (Y & 7));
				break;
			case Encender_Pixel:
				// Colocamos 1 en la posición del buffer
				OLED_Buffer [X + (Y / 8) * OLED_WIDTH] |= (1 << (Y & 7));
				break;
		}
	}
}

void OLED_TogglePixel (uint8_t X, uint8_t Y) {
	if(X >= OLED_WIDTH || Y >= OLED_HEIGHT) {
		// No hacer nada
	}
	else {
		OLED_Buffer[X + (Y / 8) * OLED_WIDTH] ^= (1 << (Y % 8));
	}
}

void OLED_EscribirNumero (uint8_t X, uint8_t Y, uint8_t numero, uint8_t Estado) {
	if (X < OLED_WIDTH - 6 && Y < OLED_HEIGHT - 11 && numero < 10) {
		switch (numero) {
			case 0:
				OLED_DibujarRecta (X + 0, Y + 3, X + 0, Y + 8, Estado);
				OLED_DibujarRecta (X + 1, Y + 1, X + 1, Y + 10, Estado);
				OLED_DibujarRecta (X + 2, Y + 0, X + 2, Y + 2, Estado);
				OLED_DibujarRecta (X + 2, Y + 9, X + 2, Y + 11, Estado);
				OLED_DibujarRecta (X + 3, Y + 0, X + 3, Y + 1, Estado);
				OLED_DibujarRecta (X + 3, Y + 10, X + 3, Y + 11, Estado);
				OLED_DibujarRecta (X + 4, Y + 0, X + 4, Y + 2, Estado);
				OLED_DibujarRecta (X + 4, Y + 9, X + 4, Y + 11, Estado);
				OLED_DibujarRecta (X + 5, Y + 1, X + 5, Y + 10, Estado);
				OLED_DibujarRecta (X + 6, Y + 3, X + 6, Y + 8, Estado);
				break;
			case 1:
				OLED_DibujarRecta (X + 0, Y + 4, X + 0, Y + 5, Estado);
				OLED_DibujarRecta (X + 1, Y + 3, X + 1, Y + 5, Estado);
				OLED_DibujarRecta (X + 2, Y + 2, X + 2, Y + 4, Estado);
				OLED_DibujarRecta (X + 3, Y + 1, X + 3, Y + 3, Estado);
				OLED_DibujarRecta (X + 4, Y + 0, X + 4, Y + 11, Estado);
				OLED_DibujarRecta (X + 5, Y + 0, X + 5, Y + 11, Estado);
				OLED_DibujarRecta (X + 6, Y + 0, X + 6, Y + 11, Estado);
				break;
			case 2:
				OLED_DibujarRecta (X + 0, Y + 2, X + 0, Y + 3, Estado);
				OLED_DibujarRecta (X + 0, Y + 9, X + 0, Y + 11, Estado);
				OLED_DibujarRecta (X + 1, Y + 1, X + 1, Y + 3, Estado);
				OLED_DibujarRecta (X + 1, Y + 8, X + 1, Y + 11, Estado);
				OLED_DibujarRecta (X + 2, Y + 0, X + 2, Y + 2, Estado);
				OLED_DibujarRecta (X + 2, Y + 7, X + 2, Y + 11, Estado);
				OLED_DibujarRecta (X + 3, Y + 0, X + 3, Y + 1, Estado);
				OLED_DibujarRecta (X + 3, Y + 6, X + 3, Y + 8, Estado);
				OLED_DibujarRecta (X + 3, Y + 10, X + 3, Y + 11, Estado);
				OLED_DibujarRecta (X + 4, Y + 0, X + 4, Y + 2, Estado);
				OLED_DibujarRecta (X + 4, Y + 5, X + 4, Y + 7, Estado);
```

```
            OLED_DibujarRecta (X + 4, Y + 10, X + 4, Y + 11, Estado);
            OLED_DibujarRecta (X + 5, Y + 1, X + 5, Y + 6, Estado);
            OLED_DibujarRecta (X + 5, Y + 10, X + 5, Y + 11, Estado);
            OLED_DibujarRecta (X + 6, Y + 2, X + 6, Y + 5, Estado);
            OLED_DibujarRecta (X + 6, Y + 10, X + 6, Y + 11, Estado);
            break;
        case 3:
            OLED_DibujarRecta (X + 0, Y + 2, X + 0, Y + 3, Estado);
            OLED_DibujarRecta (X + 0, Y + 8, X + 0, Y + 9, Estado);
            OLED_DibujarRecta (X + 1, Y + 1, X + 1, Y + 3, Estado);
            OLED_DibujarRecta (X + 1, Y + 8, X + 1, Y + 10, Estado);
            OLED_DibujarRecta (X + 2, Y + 0, X + 2, Y + 2, Estado);
            OLED_DibujarRecta (X + 2, Y + 9, X + 2, Y + 11, Estado);
            OLED_DibujarRecta (X + 3, Y + 0, X + 3, Y + 1, Estado);
            OLED_DibujarRecta (X + 3, Y + 5, X + 3, Y + 5, Estado);
            OLED_DibujarRecta (X + 3, Y + 10, X + 3, Y + 11, Estado);
            OLED_DibujarRecta (X + 4, Y + 0, X + 4, Y + 2, Estado);
            OLED_DibujarRecta (X + 4, Y + 4, X + 4, Y + 6, Estado);
            OLED_DibujarRecta (X + 4, Y + 9, X + 4, Y + 11, Estado);
            OLED_DibujarRecta (X + 5, Y + 1, X + 5, Y + 10, Estado);
            OLED_DibujarRecta (X + 6, Y + 2, X + 6, Y + 4, Estado);
            OLED_DibujarRecta (X + 6, Y + 6, X + 6, Y + 9, Estado);
            break;
        case 4:
            OLED_DibujarRecta (X + 0, Y + 6, X + 0, Y + 9, Estado);
            OLED_DibujarRecta (X + 1, Y + 4, X + 1, Y + 9, Estado);
            OLED_DibujarRecta (X + 2, Y + 3, X + 2, Y + 6, Estado);
            OLED_DibujarRecta (X + 2, Y + 8, X + 2, Y + 9, Estado);
            OLED_DibujarRecta (X + 3, Y + 2, X + 3, Y + 4, Estado);
            OLED_DibujarRecta (X + 3, Y + 8, X + 3, Y + 9, Estado);
            OLED_DibujarRecta (X + 4, Y + 1, X + 4, Y + 11, Estado);
            OLED_DibujarRecta (X + 5, Y + 0, X + 5, Y + 11, Estado);
            OLED_DibujarRecta (X + 6, Y + 8, X + 6, Y + 9, Estado);
            break;
        case 5:
            OLED_DibujarRecta (X + 0, Y + 0, X + 0, Y + 5, Estado);
            OLED_DibujarRecta (X + 0, Y + 8, X + 0, Y + 9, Estado);
            OLED_DibujarRecta (X + 1, Y + 0, X + 1, Y + 5, Estado);
            OLED_DibujarRecta (X + 1, Y + 8, X + 1, Y + 10, Estado);
            OLED_DibujarRecta (X + 2, Y + 0, X + 2, Y + 1, Estado);
            OLED_DibujarRecta (X + 2, Y + 4, X + 2, Y + 5, Estado);
            OLED_DibujarRecta (X + 2, Y + 9, X + 2, Y + 11, Estado);
            OLED_DibujarRecta (X + 3, Y + 0, X + 3, Y + 1, Estado);
            OLED_DibujarRecta (X + 3, Y + 4, X + 3, Y + 5, Estado);
            OLED_DibujarRecta (X + 3, Y + 10, X + 3, Y + 11, Estado);
            OLED_DibujarRecta (X + 4, Y + 0, X + 4, Y + 1, Estado);
            OLED_DibujarRecta (X + 4, Y + 4, X + 4, Y + 6, Estado);
            OLED_DibujarRecta (X + 4, Y + 9, X + 4, Y + 11, Estado);
            OLED_DibujarRecta (X + 5, Y + 0, X + 5, Y + 1, Estado);
            OLED_DibujarRecta (X + 5, Y + 5, X + 5, Y + 10, Estado);
            OLED_DibujarRecta (X + 6, Y + 0, X + 6, Y + 1, Estado);
            OLED_DibujarRecta (X + 6, Y + 6, X + 6, Y + 9, Estado);
            break;
        case 6:
            OLED_DibujarRecta (X + 0, Y + 2, X + 0, Y + 9, Estado);
            OLED_DibujarRecta (X + 1, Y + 1, X + 1, Y + 10, Estado);
            OLED_DibujarRecta (X + 2, Y + 0, X + 2, Y + 2, Estado);
            OLED_DibujarRecta (X + 2, Y + 4, X + 2, Y + 6, Estado);
            OLED_DibujarRecta (X + 2, Y + 9, X + 2, Y + 11, Estado);
            OLED_DibujarRecta (X + 3, Y + 0, X + 3, Y + 1, Estado);
            OLED_DibujarRecta (X + 3, Y + 4, X + 3, Y + 5, Estado);
            OLED_DibujarRecta (X + 3, Y + 10, X + 3, Y + 11, Estado);
            OLED_DibujarRecta (X + 4, Y + 0, X + 4, Y + 1, Estado);
            OLED_DibujarRecta (X + 4, Y + 4, X + 4, Y + 6, Estado);
```

```
                                OLED_DibujarRecta (X + 4, Y + 9, X + 4, Y + 11, Estado);
                                OLED_DibujarRecta (X + 5, Y + 0, X + 5, Y + 2, Estado);
                                OLED_DibujarRecta (X + 5, Y + 5, X + 5, Y + 10, Estado);
                                OLED_DibujarRecta (X + 6, Y + 1, X + 6, Y + 2, Estado);
                                OLED_DibujarRecta (X + 6, Y + 6, X + 6, Y + 9, Estado);
                                break;
                        case 7:
                                OLED_DibujarRecta (X + 0, Y + 0, X + 0, Y + 1, Estado);
                                OLED_DibujarRecta (X + 1, Y + 0, X + 1, Y + 1, Estado);
                                OLED_DibujarRecta (X + 1, Y + 9, X + 1, Y + 11, Estado);
                                OLED_DibujarRecta (X + 2, Y + 0, X + 2, Y + 1, Estado);
                                OLED_DibujarRecta (X + 2, Y + 7, X + 2, Y + 11, Estado);
                                OLED_DibujarRecta (X + 3, Y + 0, X + 3, Y + 1, Estado);
                                OLED_DibujarRecta (X + 3, Y + 5, X + 3, Y + 9, Estado);
                                OLED_DibujarRecta (X + 4, Y + 0, X + 4, Y + 1, Estado);
                                OLED_DibujarRecta (X + 4, Y + 3, X + 4, Y + 7, Estado);
                                OLED_DibujarRecta (X + 5, Y + 0, X + 5, Y + 5, Estado);
                                OLED_DibujarRecta (X + 6, Y + 0, X + 6, Y + 3, Estado);
                                break;
                        case 8:
                                OLED_DibujarRecta (X + 0, Y + 2, X + 0, Y + 4, Estado);
                                OLED_DibujarRecta (X + 0, Y + 7, X + 0, Y + 9, Estado);
                                OLED_DibujarRecta (X + 1, Y + 1, X + 1, Y + 10, Estado);
                                OLED_DibujarRecta (X + 2, Y + 0, X + 2, Y + 1, Estado);
                                OLED_DibujarRecta (X + 2, Y + 5, X + 2, Y + 6, Estado);
                                OLED_DibujarRecta (X + 2, Y + 10, X + 2, Y + 11, Estado);
                                OLED_DibujarRecta (X + 3, Y + 0, X + 3, Y + 1, Estado);
                                OLED_DibujarRecta (X + 3, Y + 5, X + 3, Y + 6, Estado);
                                OLED_DibujarRecta (X + 3, Y + 10, X + 3, Y + 11, Estado);
                                OLED_DibujarRecta (X + 4, Y + 0, X + 4, Y + 1, Estado);
                                OLED_DibujarRecta (X + 4, Y + 5, X + 4, Y + 6, Estado);
                                OLED_DibujarRecta (X + 4, Y + 10, X + 4, Y + 11, Estado);
                                OLED_DibujarRecta (X + 5, Y + 1, X + 5, Y + 10, Estado);
                                OLED_DibujarRecta (X + 6, Y + 2, X + 6, Y + 4, Estado);
                                OLED_DibujarRecta (X + 6, Y + 7, X + 6, Y + 9, Estado);
                                break;
                        case 9:
                                OLED_DibujarRecta (X + 0, Y + 2, X + 0, Y + 5, Estado);
                                OLED_DibujarRecta (X + 0, Y + 9, X + 0, Y + 10, Estado);
                                OLED_DibujarRecta (X + 1, Y + 1, X + 1, Y + 6, Estado);
                                OLED_DibujarRecta (X + 1, Y + 9, X + 1, Y + 11, Estado);
                                OLED_DibujarRecta (X + 2, Y + 0, X + 2, Y + 2, Estado);
                                OLED_DibujarRecta (X + 2, Y + 5, X + 2, Y + 7, Estado);
                                OLED_DibujarRecta (X + 2, Y + 10, X + 2, Y + 11, Estado);
                                OLED_DibujarRecta (X + 3, Y + 0, X + 3, Y + 1, Estado);
                                OLED_DibujarRecta (X + 3, Y + 6, X + 3, Y + 7, Estado);
                                OLED_DibujarRecta (X + 3, Y + 10, X + 3, Y + 11, Estado);
                                OLED_DibujarRecta (X + 4, Y + 0, X + 4, Y + 2, Estado);
                                OLED_DibujarRecta (X + 4, Y + 5, X + 4, Y + 7, Estado);
                                OLED_DibujarRecta (X + 4, Y + 9, X + 4, Y + 11, Estado);
                                OLED_DibujarRecta (X + 5, Y + 1, X + 5, Y + 10, Estado);
                                OLED_DibujarRecta (X + 6, Y + 2, X + 6, Y + 9, Estado);
                                break;
                }
        }
}

void OLED_RectanguloRelleno (uint8_t X, uint8_t Y, uint8_t Ancho, uint8_t Alto, uint8_t Estado) {
        if ((X + Ancho) >= OLED_WIDTH || (Y + Alto) >= OLED_HEIGHT){
                // No hacer nada
        }
        else {
                for (uint8_t i = X; i <= X + Ancho; i ++) {
                        for (uint8_t j = Y; j <= Y + Alto; j ++) {
```

```
                                        OLED_SetPixel (i, j, Estado);
                                }
                        }
                }
        }

void OLED_RectanguloVacio (uint8_t X, uint8_t Y, uint8_t Ancho, uint8_t Alto, uint8_t Estado) {
        if ((X + Ancho) > OLED_WIDTH || (Y + Alto) > OLED_HEIGHT){
                // No hacer nada
        }
        else {
                // Dibujar la línea superior
                for (uint8_t i = 0; i < Ancho; i++) {
                        OLED_SetPixel(X + i, Y, Estado);
                }
                // Dibujar la línea inferior
                for (uint8_t i = 0; i < Ancho; i++) {
                        OLED_SetPixel(X + i, Y + Alto - 1, Estado);
                }
                // Dibujar la línea izquierda
                for (uint8_t i = 0; i < Alto; i++) {
                        OLED_SetPixel(X, Y + i, Estado);
                }
                // Dibujar la línea derecha
                for (uint8_t i = 0; i < Alto; i++) {
                        OLED_SetPixel(X + Ancho - 1, Y + i, Estado);
                }
        }
}

void OLED_ToggleRectangulo (uint8_t X_Ini, uint8_t Y_Ini, uint8_t X_Fin, uint8_t Y_Fin) {
        uint8_t X_menor = X_Ini, X_mayor = X_Fin, Y_menor = Y_Ini, Y_mayor = Y_Fin;

        if (X_Ini >= OLED_WIDTH || Y_Ini >= OLED_HEIGHT || X_Fin >= OLED_WIDTH || Y_Fin >= OLED_HEIGHT) {
                        // No hacer nada
        }
        else {
                if (X_Ini > X_Fin) {
                        X_mayor = X_Ini;
                        X_menor = X_Fin;
                }
                if (Y_Ini > Y_Fin) {
                        Y_mayor = Y_Ini;
                        Y_menor = Y_Fin;
                }
                for (uint8_t x = X_menor; x <= X_mayor; x ++) {
                        for (uint8_t y = Y_menor; y <= Y_mayor; y ++) {
                                OLED_TogglePixel (x, y);
                        }
                }
        }
}

void OLED_DibujarRecta (uint8_t X_Ini, uint8_t Y_Ini, uint8_t X_Fin, uint8_t Y_Fin, uint8_t Estado) {
        int16_t dx = abs (X_Fin - X_Ini);
        int16_t dy = abs (Y_Fin - Y_Ini);
        int8_t sx = (X_Ini < X_Fin) ? 1 : -1;
        int8_t sy = (Y_Ini < Y_Fin) ? 1 : -1;
        int16_t err = dx - dy;
        int16_t e2;

        if (X_Ini >= OLED_WIDTH || Y_Ini >= OLED_HEIGHT || X_Fin >= OLED_WIDTH || Y_Fin >= OLED_HEIGHT) {
                                // No hacer nada
        }
```

```
	else {
		while (1) {
			OLED_SetPixel (X_Ini, Y_Ini, Estado);
			if (X_Ini == X_Fin && Y_Ini == Y_Fin) break;
			e2 = 2 * err;
			if (e2 > -dy) {
				err -= dy;
				X_Ini += sx;
			}
			if (e2 < dx) {
				err += dx;
				Y_Ini += sy;
			}
		}
	}
}

void OLED_DibujarArco (uint8_t X, uint8_t Y, uint8_t Radio, int16_t Ang_Ini_Por_Diez, int16_t Ang_Fin_Por_Diez,
			uint8_t Estado) {
	uint8_t x, y;
	int16_t angulo_menor = Ang_Ini_Por_Diez, angulo_mayor = Ang_Fin_Por_Diez;
	if (Ang_Ini_Por_Diez > Ang_Fin_Por_Diez) {
		angulo_menor = Ang_Fin_Por_Diez;
		angulo_mayor = Ang_Ini_Por_Diez;
	}
	while (angulo_menor <= angulo_mayor) {
		x = X + (int16_t)(Radio * cos(angulo_menor * (3.141592654 / 180.0) / 10));
		y = Y - (int16_t)(Radio * sin(angulo_menor * (3.141592654 / 180.0) / 10));
		OLED_SetPixel (x, y, Estado);
		angulo_menor += 5;
	}
}

void OLED_CopiarImagen (const uint8_t *Imagen, uint16_t Tamaño) {
	uint16_t CpyBuffer;

	OLED_BorrarBuffer ();
	for(CpyBuffer = 0; CpyBuffer < Tamaño; CpyBuffer ++) {
		OLED_Buffer[CpyBuffer] = *(Imagen + CpyBuffer);
	}
}
```

images.h

En este código se copiaron y pegaron los mapas de bits de las imágenes mostradas anteriormente, debido a la repetitividad del código y a su nula relevancia no lo colocamos en este apartado, en caso de querer verlo más adelante se adjunta un repositorio en github con el proyecto completo.

definitions.h

```
#ifndef DEFINITIONS_H_
#define DEFINITIONS_H_

#include "OLED_128x64_SSD1306.h"
#include "ADXL345.h"
#include "ENCODER.h"
#include "ALARMA.h"
#include "FLASH.h"
#include "images.h"

#include <stdio.h>
#include "board.h"
#include "peripherals.h"
```

```
#include "pin_mux.h"
#include "clock_config.h"
#include "LPC845.h"
#include "fsl_debug_console.h"

#define PUERTO_0 0
#define FILA_1 16
#define FILA_2 17
#define FILA_3 18
#define FILA_4 19
#define COLUMNA_1 20
#define COLUMNA_2 21
#define COLUMNA_3 22
#define COLUMNA_4 23

typedef enum {
	ninguno = 0,
	op_a = 1,
	op_b = 2,
	op_c = 3,
	op_d = 4,
	op_numeral = 5,
} opcion_t;

typedef enum {
	distancias = 1,
	grados = 2,
} modo_t;

typedef enum {
	inicio = 1,
	alarma = 2,
	config_general = 3,
	config_grados = 4,
	mostrar_grados = 5,
	config_distancia = 6,
	mostrar_distancia = 7,
} menu_t;

void Delay_ms (uint8_t mseg);
void GPIO_Inicializacion (void);
void SysTick_Inicializacion (void);
void I2C_Inicializacion (void);
uint8_t TECLADO_Lectura (uint8_t Fila_Ini, uint8_t Columna_Ini, uint8_t Fila_Fin, uint8_t Columna_Fin);
void OLED_BarraEscritura (uint8_t X, uint8_t Y);

void toggle_op_a (menu_t menu);
void toggle_op_b (menu_t menu);
void toggle_op_c (menu_t menu);
void toggle_op_d (menu_t menu);
void mostrar_menu (menu_t menu, uint8_t *positivo, uint8_t *negativo, uint8_t *centimetros, uint8_t *metros);
void menu_independiente (menu_t menu, opcion_t op, uint8_t numeros_configuracion, uint8_t con_alarma,
                         gravedad_t *gravedad, int16_t *angulo, int16_t *angulo_resultante,
                         int16_t angulo_cero, int32_t *distancia, uint8_t *positivo, uint8_t *negativo,
                         uint8_t *metros, uint8_t *centimetros);

/*
* op: opcion que está activa en el menu
* menu: menu que se está mostrando
* con_alarma: indica si se usa la alarma o no
* modo: indica si es medicion de distancia o de angulo
* numeros_configuracion: indica el numero que se está inigresando por teclado
*/
uint8_t tecla_a (opcion_t *op, menu_t *menu, uint8_t *con_alarma, modo_t *modo, uint8_t *numeros_configuracion);
```

```
uint8_t tecla_b (opcion_t *op, menu_t *menu, uint8_t *con_alarma, modo_t *modo, uint8_t *numeros_configuracion,
                 int16_t *angulo_cero, int16_t angulo, int32_t *distancia, gravedad_t *gravedad);
uint8_t tecla_c (opcion_t *op, menu_t *menu, uint8_t con_alarma, int16_t *angulo_cero, int32_t *distancia);
uint8_t tecla_d (opcion_t *op, menu_t *menu, gravedad_t *gravedad);
void tecla_num (opcion_t *op, menu_t *menu, uint8_t *numeros_configuracion, uint8_t *boton);
void tecla_numeral (opcion_t *op, menu_t menu);

extern volatile uint8_t flag_tick;
extern volatile uint16_t flag_tick_barra;
extern uint8_t teclas[];
extern uint8_t volumen, volumen_config, contraste, version, cambio_de_config;
extern int32_t distancia_alarma;
extern uint8_t unidad_distancia_alarma;
extern int16_t angulo_alarma;
extern int8_t signo_angulo_alarma;

#endif /* DEFINITIONS_H_ */
```

functions.c

```
#include "definitions.h"

void Delay_ms (uint8_t mseg) {
        flag_tick = 0;
        while (flag_tick <= mseg);
}

void GPIO_Inicializacion (void) {
        gpio_pin_config_t out_config = {kGPIO_DigitalOutput, 1};
        gpio_pin_config_t in_config = {kGPIO_DigitalInput};
        GPIO_PortInit (GPIO, PUERTO_0);
        GPIO_PinInit (GPIO, PUERTO_0, FILA_1, &in_config);
        GPIO_PinInit (GPIO, PUERTO_0, FILA_2, &in_config);
        GPIO_PinInit (GPIO, PUERTO_0, FILA_3, &in_config);
        GPIO_PinInit (GPIO, PUERTO_0, FILA_4, &in_config);
        GPIO_PinInit (GPIO, PUERTO_0, COLUMNA_1, &out_config);
        GPIO_PinInit (GPIO, PUERTO_0, COLUMNA_2, &out_config);
        GPIO_PinInit (GPIO, PUERTO_0, COLUMNA_3, &out_config);
        GPIO_PinInit (GPIO, PUERTO_0, COLUMNA_4, &out_config);
}

void SysTick_Inicializacion (void) {
        (void) SysTick_Config (SystemCoreClock / 1000);
}

void I2C_Inicializacion (void) {
        uint32_t baudRate = 400000;
        uint32_t frecuencia = 12000000;
        i2c_master_config_t  i2config;
        CLOCK_Select (kI2C1_Clk_From_MainClk);
        CLOCK_EnableClock (kCLOCK_Swm);
        SWM_SetMovablePinSelect (SWM0, kSWM_I2C1_SDA, kSWM_PortPin_P0_27);
        SWM_SetMovablePinSelect (SWM0, kSWM_I2C1_SCL, kSWM_PortPin_P0_28);
        CLOCK_DisableClock (kCLOCK_Swm);
        I2C_MasterGetDefaultConfig (&i2config);
        i2config.baudRate_Bps = baudRate;
        I2C_MasterInit (I2C1, &i2config, frecuencia);
}

uint8_t TECLADO_Lectura (uint8_t Fila_Ini, uint8_t Columna_Ini, uint8_t Fila_Fin, uint8_t Columna_Fin) {
        for (uint8_t columna = Columna_Ini; columna <= Columna_Fin; columna ++) {
                GPIO_PinWrite (GPIO, PUERTO_0, COLUMNA_1 + columna, 0);
                for (uint8_t fila = Fila_Ini; fila <= Fila_Fin; fila ++) {
```

```
                        if (GPIO_PinRead (GPIO, PUERTO_0, FILA_1 + fila) == 0) {
                                Delay_ms (50);
                                if (GPIO_PinRead (GPIO, PUERTO_0, FILA_1 + fila) == 0) {
                                        GPIO_PinWrite (GPIO, PUERTO_0, COLUMNA_1 + columna, 1);
                                        return teclas[4 * fila + columna];
                                }
                        }
                }
                GPIO_PinWrite (GPIO, PUERTO_0, COLUMNA_1 + columna, 1);;
        }
        return '\0';
}

void OLED_BarraEscritura (uint8_t X, uint8_t Y) {
        if (flag_tick_barra >= 500) {
                OLED_ToggleRectangulo (X, Y, X + 1, Y + 11);
                flag_tick_barra = 0;
                OLED_Refresco ();
        }
}

void toggle_op_a (menu_t menu) {
        switch (menu) {
        case inicio:
        case alarma:
                OLED_ToggleRectangulo (1, 1, 47, 47);
                break;
        case config_general:
                OLED_ToggleRectangulo (1, 1, 13, 14);
                break;
        case config_grados:
                OLED_ToggleRectangulo (19, 1, 63, 14);
                break;
        case mostrar_grados:
                // Nada
                break;
        case config_distancia:
                OLED_ToggleRectangulo (26, 1, 63, 14);
                break;
        case mostrar_distancia:
                // Nada
                break;
        }
}

void toggle_op_b (menu_t menu) {
        switch (menu) {
        case inicio:
        case alarma:
                OLED_ToggleRectangulo (51, 1, 97, 47);
                break;
        case config_general:
                OLED_ToggleRectangulo (51, 1, 62, 15);
                break;
        case config_grados:
                OLED_ToggleRectangulo (1, 17, 15, 31);
                OLED_ToggleRectangulo (1, 33, 15, 47);
                break;
        case mostrar_grados:
                OLED_ToggleRectangulo (1, 1, 12, 23);
                break;
        case config_distancia:
                OLED_ToggleRectangulo (1, 17, 22, 31);
                OLED_ToggleRectangulo (1, 33, 22, 47);
```

```
                break;
        case mostrar_distancia:
                OLED_ToggleRectangulo (1, 1, 12, 23);
                break;
        }
}

void toggle_op_c (menu_t menu) {
        if (menu != inicio) OLED_ToggleRectangulo (101, 1, 126, 15);
}

void toggle_op_d (menu_t menu) {
        switch (menu) {
        case inicio:
                OLED_ToggleRectangulo (101, 1, 126, 15);
                break;
        case alarma:
        case config_general:
                // Nada
                break;
        case config_grados:
        case config_distancia:
                OLED_ToggleRectangulo (67, 1, 97, 47);
                break;
        case mostrar_grados:
        case mostrar_distancia:
                OLED_ToggleRectangulo (1, 25, 12, 47);
                break;
        }
}

void mostrar_menu (menu_t menu, uint8_t *positivo, uint8_t *negativo, uint8_t *centimetros, uint8_t *metros) {
        switch (menu) {
        case inicio:
           OLED_CopiarImagen (inicio_1, sizeof (inicio_1));
                break;
        case alarma:
           OLED_CopiarImagen (alarma_2, sizeof (alarma_2));
                break;
        case config_general:
                OLED_CopiarImagen (config_general_3, sizeof (config_general_3));
                OLED_EscribirNumero (11, 28, contraste / 10, Encender_Pixel);
                OLED_EscribirNumero (19, 28, contraste % 10, Encender_Pixel);
                OLED_EscribirNumero (61, 28, volumen / 10, Encender_Pixel);
                OLED_EscribirNumero (69, 28, volumen % 10, Encender_Pixel);
                break;
        case config_grados:
                OLED_CopiarImagen (config_grados_4, sizeof (config_grados_4));
                OLED_EscribirNumero (21, 26, abs (angulo_alarma) / 1000, Encender_Pixel);
                OLED_EscribirNumero (29, 26, abs (angulo_alarma) / 100 - (abs (angulo_alarma) / 1000) * 10,
                                     Encender_Pixel);
                OLED_EscribirNumero (37, 26, abs (angulo_alarma) / 10 - (abs (angulo_alarma) / 100) * 10,
                                     Encender_Pixel);
                OLED_EscribirNumero (48, 26, abs (angulo_alarma) - (abs (angulo_alarma) / 10) * 10,
                                     Encender_Pixel);
                if (signo_angulo_alarma == 1) OLED_ToggleRectangulo (1, 17, 15, 31);
                else OLED_ToggleRectangulo (1, 33, 15, 47);
                break;
        case mostrar_grados:
                OLED_CopiarImagen (mostrar_grados_5, sizeof (mostrar_grados_5));
                for (uint8_t i = 0; i < 3; i ++) OLED_EscribirNumero (54 + 8 * i, 30, 0, Encender_Pixel);
                OLED_EscribirNumero (81, 30, 0, Encender_Pixel);
                OLED_ToggleRectangulo (53, 1, 73, 21);
                *positivo = 1;
```

```
                *negativo = 0;
                break;
        case config_distancia:
                OLED_CopiarImagen (config_distancia_6, sizeof (config_distancia_6));
                // Si unidad_alarma_distancia es 1 ya esta guardado en cm y se muestra en cm
                // Si unidad_alarma_distancia es 100 esta guardado en cm pero se muestra en m, se divide por 100
                distancia_alarma = distancia_alarma / unidad_distancia_alarma; // Si es cm se deja igual y si es m
                                                                               //sacamos los 0
                OLED_EscribirNumero (28, 26, distancia_alarma / 1000, Encender_Pixel);
                OLED_EscribirNumero (36, 26, distancia_alarma / 100 - (distancia_alarma / 1000) * 10,
                                     Encender_Pixel);
                OLED_EscribirNumero (44, 26, distancia_alarma / 10 - (distancia_alarma / 100) * 10, Encender_Pixel);
                OLED_EscribirNumero (55, 26, distancia_alarma - (distancia_alarma / 10) * 10, Encender_Pixel);
                distancia_alarma = distancia_alarma * unidad_distancia_alarma; // Devolvemos los 0 si estaba en m
                if (unidad_distancia_alarma == 1) OLED_ToggleRectangulo (1, 17, 22, 31);
                else OLED_ToggleRectangulo (1, 33, 22, 47);
                break;
        case mostrar_distancia:
                OLED_CopiarImagen (mostrar_distancia_7, sizeof (mostrar_distancia_7));
                for (uint8_t i = 0; i < 3; i ++) OLED_EscribirNumero (63 + 8 * i, 32, 0, Encender_Pixel);
                OLED_EscribirNumero (90, 32, 0, Encender_Pixel);
                OLED_ToggleRectangulo (51, 27, 60, 36);
                *positivo = 1;
                *negativo = 0;
                OLED_ToggleRectangulo (51, 1, 74, 23);
                *centimetros = 1;
                *metros = 0;
                break;
        }
}

void menu_independiente (menu_t menu, opcion_t op, uint8_t numeros_configuracion, uint8_t con_alarma,
                         gravedad_t *gravedad, int16_t *angulo, int16_t *angulo_resultante,
                         int16_t angulo_cero, int32_t *distancia, uint8_t *positivo, uint8_t *negativo,
                         uint8_t *metros, uint8_t *centimetros) {
        switch (menu) {
        case inicio: case alarma: break;
        case config_general:
                if (op == op_a) {
                        if (numeros_configuracion == 1) OLED_BarraEscritura (11, 28);
                        else OLED_BarraEscritura (19,28);
                }
                if (op == op_b) {
                        if (numeros_configuracion == 1) OLED_BarraEscritura (61, 28);
                        else OLED_BarraEscritura (69,28);
                }
                break;
        case config_grados:
                if (op == op_a) {
                        if (numeros_configuracion == 1) OLED_BarraEscritura (21, 26);
                        else if (numeros_configuracion == 2) OLED_BarraEscritura (29, 26);
                        else if (numeros_configuracion == 3) OLED_BarraEscritura (37, 26);
                        else OLED_BarraEscritura (48, 26);
                }
                break;
        case mostrar_grados:
                if (con_alarma == 1 && op != op_d) {
                        if (abs (*angulo_resultante - angulo_alarma) <= 10) {
                                INDICADOR_Alarma (TIEMPO_INDICACION, TIEMPO_INDICACION * 11, 99);
                        }
                        else if (abs (*angulo_resultante - angulo_alarma) <= 50) {
                                INDICADOR_Alarma (TIEMPO_INDICACION, TIEMPO_INDICACION * 21, 70);
                        }
                        else if (abs (*angulo_resultante - angulo_alarma) <= 100) {
```

```
                                INDICADOR_Alarma (TIEMPO_INDICACION, TIEMPO_INDICACION * 41, 40);
                        }
                        else {
                                INDICADOR_Alarma (TIEMPO_INDICACION, TIEMPO_INDICACION * 61, 10);
                        }
                }
                if (GPIO_PinRead (GPIO, PUERTO_0, INT_ADXL345) == 1) {
                        *angulo = ADXL345_Angulo (&(*gravedad));
                        if (gravedad->indice == 0 && op != op_d) {
                                *angulo_resultante = *angulo - angulo_cero;
                                if (*angulo_resultante >= 0 && *positivo == 0) {
                                OLED_ToggleRectangulo (53, 1, 73, 21);
                                OLED_ToggleRectangulo (75, 1, 95, 21);
                                *positivo = 1;
                                *negativo = 0;
                                }
                        if (*angulo_resultante < 0 && *negativo == 0) {
                                OLED_ToggleRectangulo (53, 1, 73, 21);
                                OLED_ToggleRectangulo (75, 1, 95, 21);
                                *positivo = 0;
                                *negativo = 1;
                        }
                        OLED_RectanguloRelleno (54, 30, 22, 11, Apagar_Pixel);
                        OLED_RectanguloRelleno (81, 30, 6, 11, Apagar_Pixel);
                        *angulo_resultante = abs (*angulo_resultante);
                        OLED_EscribirNumero (54, 30, *angulo_resultante / 1000, Encender_Pixel);
                        OLED_EscribirNumero (62, 30, *angulo_resultante / 100 - (*angulo_resultante / 1000) * 10,
                                        Encender_Pixel);
                        OLED_EscribirNumero (70, 30, *angulo_resultante / 10 - (*angulo_resultante / 100) * 10,
                                        Encender_Pixel);
                        OLED_EscribirNumero (81, 30, *angulo_resultante - (*angulo_resultante / 10) * 10,
                                        Encender_Pixel);
                        OLED_Refresco ();
                        }
                }
        case config_distancia:
                if (op == op_a) {
                        if (numeros_configuracion == 1) OLED_BarraEscritura (28, 26);
                        else if (numeros_configuracion == 2) OLED_BarraEscritura (36, 26);
                        else if (numeros_configuracion == 3) OLED_BarraEscritura (44, 26);
                        else OLED_BarraEscritura (55, 26);
                }
                break;
        case mostrar_distancia:
                if (op != op_d) {
                        if (antirrebote_encoder == 1) {
                                *distancia += ENCODER_Lectura ();
                                int32_t distancia_resultante = *distancia;
                                if (distancia_resultante >= 0 && *positivo == 0) {
                                        OLED_ToggleRectangulo (51, 27, 60, 36);
                                        OLED_ToggleRectangulo (51, 38, 60, 47);
                                        *positivo = 1;
                                        *negativo = 0;
                                }
                                if (distancia_resultante < 0 && *negativo == 0) {
                                        OLED_ToggleRectangulo (51, 27, 60, 36);
                                        OLED_ToggleRectangulo (51, 38, 60, 47);
                                        *positivo = 0;
                                        *negativo = 1;
                                }
                                OLED_RectanguloRelleno (63, 32, 22, 11, Apagar_Pixel);
                                OLED_RectanguloRelleno (90, 32, 6, 11, Apagar_Pixel);
                                distancia_resultante = abs (distancia_resultante);
                                if (distancia_resultante <= 9999 && *centimetros == 0) { // En cm mostramos
```

```
                                        OLED_ToggleRectangulo (51, 1, 74, 23);
                                        OLED_ToggleRectangulo (76, 1, 97, 23);
                                        *centimetros = 1;
                                        *metros = 0;
                                }
                                if (distancia_resultante > 9999 && *metros == 0) { // En m mostramos
                                        OLED_ToggleRectangulo (51, 1, 74, 23);
                                        OLED_ToggleRectangulo (76, 1, 97, 23);
                                        *centimetros = 0;
                                        *metros = 1;
                                }
                                if (distancia_resultante > 9999) distancia_resultante = distancia_resultante / 100;
                                OLED_EscribirNumero (63, 32, distancia_resultante / 1000, Encender_Pixel);
                                OLED_EscribirNumero (71, 32, distancia_resultante / 100 - (distancia_resultante /
                                                1000) * 10, Encender_Pixel);
                                OLED_EscribirNumero (79, 32, distancia_resultante / 10 - (distancia_resultante /
                                                100) * 10, Encender_Pixel);
                                OLED_EscribirNumero (90, 32, distancia_resultante - (distancia_resultante / 10) * 10,
                                                 Encender_Pixel);
                                OLED_Refresco ();
                        }
                        if (con_alarma == 1) {
                                if (abs (*distancia - distancia_alarma) <= 10 * unidad_distancia_alarma) {
                                        INDICADOR_Alarma (TIEMPO_INDICACION, TIEMPO_INDICACION * 11,
                                                99);
                                }
                                else if (abs (*distancia - distancia_alarma) <= 20 * unidad_distancia_alarma) {
                                        INDICADOR_Alarma (TIEMPO_INDICACION, TIEMPO_INDICACION * 21,
                                                70);
                                }
                                else if (abs (*distancia - distancia_alarma) <= 50 * unidad_distancia_alarma) {
                                        INDICADOR_Alarma (TIEMPO_INDICACION, TIEMPO_INDICACION * 41,
                                                40);
                                }
                                else {
                                        INDICADOR_Alarma (TIEMPO_INDICACION, TIEMPO_INDICACION * 61,
                                                10);
                                }
                        }
                }
        }
}
```

a_key.c

```
#include "definitions.h"

uint8_t tecla_a (opcion_t *op, menu_t *menu, uint8_t *con_alarma, modo_t *modo, uint8_t *numeros_configuracion) {
        if (*menu != mostrar_grados && *menu != mostrar_distancia) {
                switch (*op) {
                case ninguno: break;
                case op_a:
                        if (*menu == inicio) {
                                *menu = alarma;
                                *modo = distancias;
                                return 1;
                        }
                        else if (*menu == alarma) {
                                if (*modo == distancias) *menu = config_distancia;
                                else *menu = config_grados;
                                *con_alarma = 1;
                                return 1;
                        }
```

```
                        else return 0;
                case op_b:
                        if (*menu == inicio || *menu == alarma) toggle_op_b (*menu);
                        else return 0;
                        break;
                case op_c:
                        if (*menu != inicio) toggle_op_c (*menu);
                        else return 0;
                        break;
                case op_d:
                        if (*menu != alarma && *menu != config_general) toggle_op_d (*menu);
                        else return 0;
                        break;
                case op_numeral:
                        OLED_ToggleRectangulo (101, 19, 126, 47);
                        break;
                }
                toggle_op_a (*menu);
                *op = op_a;
                if (*menu == config_general) {
                        OLED_RectanguloRelleno (11, 28, 14, 11, Apagar_Pixel);
                        (*numeros_configuracion) ++;
                        contraste = 0;
                        flag_tick_barra = 0;
                }
                else if (*menu == config_grados) {
                        angulo_alarma = 0;
                        OLED_RectanguloRelleno (21, 26, 23, 11, Apagar_Pixel);
                        OLED_RectanguloRelleno (48, 26, 6, 11, Apagar_Pixel);
                        (*numeros_configuracion) ++;
                        flag_tick_barra = 0;
                }
                else if (*menu == config_distancia) {
                        distancia_alarma = 0;
                        OLED_RectanguloRelleno (28, 26, 23, 11, Apagar_Pixel);
                        OLED_RectanguloRelleno (55, 26, 6, 11, Apagar_Pixel);
                        (*numeros_configuracion) ++;
                        flag_tick_barra = 0;
                }
        }
        return 0;
}
```

b_key.c

```
#include "definitions.h"

uint8_t tecla_b (opcion_t *op, menu_t *menu, uint8_t *con_alarma, modo_t *modo, uint8_t *numeros_configuracion,
                int16_t *angulo_cero, int16_t angulo, int32_t *distancia, gravedad_t *gravedad) {
        switch (*op) {
        case ninguno: break;
        case op_a:
                if (*menu == inicio || *menu == alarma) toggle_op_a (*menu);
                else return 0;
                break;
        case op_b:
                if (*menu == inicio) {
                        *menu = alarma;
                        *modo = grados;
                        return 1;
                }
                else if (*menu == alarma) {
                        if (*modo == distancias) {
```

```
                                *menu = mostrar_distancia;
                                ENCODER_Activo ();
                        }
                        else {
                                *menu = mostrar_grados;
                                ADXL345_Encender (&(*gravedad));
                        }
                        *con_alarma = 0;
                        return 1;
                }
                else return 0;
                break;
        case op_c:
                if (*menu != inicio && *menu != config_grados && *menu != config_distancia) toggle_op_c (*menu);
                break;
        case op_d:
                if (*menu == inicio) toggle_op_d (*menu);
                if (*menu == mostrar_distancia || *menu == mostrar_grados) return 0;
                break;
        case op_numeral:
                OLED_ToggleRectangulo (101, 19, 126, 47);
                break;
        }
        toggle_op_b (*menu);
        if (*menu == inicio || *menu == alarma || *menu == config_general) {
                *op = op_b;
        }
        switch (*menu) {
        case inicio: case alarma: break;
        case config_general:
                OLED_RectanguloRelleno (61, 28, 14, 11, Apagar_Pixel);
                (*numeros_configuracion) ++;
                volumen_config = 0;
                flag_tick_barra = 0;
                break;
        case config_grados:
                signo_angulo_alarma = signo_angulo_alarma * (-1);
                angulo_alarma = angulo_alarma * signo_angulo_alarma;
                cambio_de_config = 1;
                break;
        case mostrar_grados:
                *angulo_cero = angulo;
        case mostrar_distancia:
                *distancia = 0;
                OLED_Refresco ();
                Delay_ms (100);
                toggle_op_b (*menu);
                *op = ninguno;
                return 1;
        case config_distancia:
                if (unidad_distancia_alarma == 1) {
                        unidad_distancia_alarma = 100;
                        distancia_alarma = distancia_alarma * 100;
                }
                else {
                        unidad_distancia_alarma = 1;
                        distancia_alarma = distancia_alarma / 100;
                }
                cambio_de_config = 1;
                break;
        }
        return 0;
}
```

c_key.c

```
#include "definitions.h"

uint8_t tecla_c (opcion_t *op, menu_t *menu, uint8_t con_alarma, int16_t *angulo_cero, int32_t *distancia) {
	if (*menu != inicio) {
		switch (*op) {
		case ninguno: break;
		case op_a:
			if (*menu != config_general && *menu != config_grados && *menu != config_distancia)
				toggle_op_a (*menu);
			break;
		case op_b:
			if (*menu != config_general && *menu != config_grados && *menu != config_distancia)
				toggle_op_b (*menu);
			break;
		case op_c:
			switch (*menu) {
			case inicio: break;
			case alarma: case config_general:
				*menu = inicio;
				break;
			case config_grados: case config_distancia:
				*menu = alarma;
				break;
			case mostrar_grados:
				if (con_alarma) *menu = config_grados;
				else *menu = alarma;
				*angulo_cero = 0;
				ADXL345_Apagar ();
				break;
			case mostrar_distancia:
				if (con_alarma) *menu = config_distancia;
				else *menu = alarma;
				*distancia = 0;
				ENCODER_Desactivo ();
				break;
			}
			return 1;
		case op_d:
			if (*menu == config_grados || *menu == config_distancia) toggle_op_d (*menu);
			else return 0;
			break;
		case op_numeral: break;
		}
		toggle_op_c (*menu);
		*op = op_c;
	}
	return 0;
}
```

d_key.c

```
#include "definitions.h"

uint8_t tecla_d (opcion_t *op, menu_t *menu, gravedad_t *gravedad) {
	if (*menu != alarma && *menu != config_general) {
		switch (*op) {
		case ninguno: break;
		case op_a:
			if (*menu == inicio) toggle_op_a (*menu);
			else return 0;
			break;
```

```
                case op_b:
                        if (*menu == inicio) toggle_op_b (*menu);
                        else return 0;
                        break;
                case op_c:
                        if (*menu != inicio) toggle_op_c (*menu);
                        else return 0;
                        break;
                case op_d:
                        switch (*menu) {
                        case alarma: case config_general: break;
                        case inicio:
                                *menu = config_general;
                                break;
                        case config_grados:
                                *menu = mostrar_grados;
                                ADXL345_Encender (&(*gravedad));
                                break;
                        case config_distancia:
                                *menu = mostrar_distancia;
                                ENCODER_Activo ();
                                PINT_EnableCallback (PINT);
                                break;
                        case mostrar_grados: case mostrar_distancia:
                                toggle_op_d (*menu);
                                *op = ninguno;
                                return 0;
                        }
                        return 1;
                        case op_numeral:
                                OLED_ToggleRectangulo (101, 19, 126, 47);
                                break;
                }
                toggle_op_d (*menu);
                *op = op_d;
        }
        return 0;
}
```

num_key.c

```
#include "definitions.h"

void tecla_num (opcion_t *op, menu_t *menu, uint8_t *numeros_configuracion, uint8_t *boton) {
        switch (*menu) {
        case inicio: case alarma: case mostrar_grados: case mostrar_distancia: break;
        case config_general:
                if (*numeros_configuracion == 1) {
                        if (*op == op_a) contraste += (*boton - 48) * 10;
                        else volumen_config += (*boton - 48) * 10;
                        OLED_RectanguloRelleno (50 * (*op) - 39, 28, 1, 11, Apagar_Pixel);
                        OLED_EscribirNumero (50 * (*op) - 39, 28, *boton - 48, Encender_Pixel);
                }
                else {
                        if (*op == op_a) contraste += *boton - 48;
                        else volumen_config += *boton - 48;
                        OLED_RectanguloRelleno (50 * (*op) - 31, 28, 1, 11, Apagar_Pixel);
                        OLED_EscribirNumero (50 * (*op) - 31, 28, *boton - 48, Encender_Pixel);
                }
                (*numeros_configuracion) ++;
                if (*numeros_configuracion == 3) {
                        *numeros_configuracion = 0;
                        if (*op == op_a) {
```

```
                        toggle_op_a (*menu);
                        OLED_Contraste (contraste * 255 / 99);
                }
                else {
                        toggle_op_b (*menu);
                        volumen = volumen_config;
                }
                cambio_de_config = 1;
                *op = ninguno;
        }
        break;
case config_grados:
        switch (*numeros_configuracion) {
        case 1:
                if (*boton == '0' || *boton == '1') {
                        angulo_alarma += (*boton - 48) * 1000;
                        OLED_RectanguloRelleno (21, 26, 1, 11, Apagar_Pixel);
                        OLED_EscribirNumero (21, 26, *boton - 48, Encender_Pixel);
                        (*numeros_configuracion) ++;
                }
                break;
        case 2:
                if ((angulo_alarma == 1000 && (*boton - 48) <= 8) || angulo_alarma == 0) {
                        angulo_alarma += (*boton - 48) * 100;
                        OLED_RectanguloRelleno (29, 26, 1, 11, Apagar_Pixel);
                        OLED_EscribirNumero (29, 26, *boton - 48, Encender_Pixel);
                        (*numeros_configuracion) ++;
                }
                break;
        case 3:
                if ((angulo_alarma == 1800 && (*boton - 48) == 0) || angulo_alarma < 1800) {
                        angulo_alarma += (*boton - 48) * 10;
                        OLED_RectanguloRelleno (37, 26, 1, 11, Apagar_Pixel);
                        OLED_EscribirNumero (37, 26, *boton - 48, Encender_Pixel);
                        (*numeros_configuracion) ++;
                }
                break;
        case 4:
                if ((angulo_alarma == 1800 && (*boton - 48) == 0) || angulo_alarma < 1800) {
                        angulo_alarma += *boton - 48;
                        OLED_RectanguloRelleno (48, 26, 1, 11, Apagar_Pixel);
                        OLED_EscribirNumero (48, 26, *boton - 48, Encender_Pixel);
                        (*numeros_configuracion) ++;
                }
                break;
        }
        if (*numeros_configuracion == 5) {
                *numeros_configuracion = 0;
                angulo_alarma = angulo_alarma * signo_angulo_alarma;
                toggle_op_a (*menu);
                cambio_de_config = 1;
                *op = ninguno;
        }
        break;
case config_distancia:
        if (*numeros_configuracion < 4) {
                distancia_alarma += (*boton - 48) * pow (10, 4 - (*numeros_configuracion));
                OLED_RectanguloRelleno (20 + 8 * (*numeros_configuracion), 26, 1, 11, Apagar_Pixel);
                OLED_EscribirNumero (20 + 8 * (*numeros_configuracion), 26, *boton - 48, Encender_Pixel);
        }
        else {
                distancia_alarma += *boton - 48;
                OLED_RectanguloRelleno (55, 26, 1, 11, Apagar_Pixel);
                OLED_EscribirNumero (55, 26, *boton - 48, Encender_Pixel);
```

```
                }
                (*numeros_configuracion) ++;
                if (*numeros_configuracion == 5) {
                        *numeros_configuracion = 0;
                        if (unidad_distancia_alarma == 100) {
                                distancia_alarma = distancia_alarma * unidad_distancia_alarma;
                                // Si está en metros se guarda en centimetros la alarma
                        }
                        toggle_op_a (*menu);
                        cambio_de_config = 1;
                        *op = ninguno;
                }
                break;
        }
}
```

numeral_key.c

```
#include "definitions.h"

void tecla_numeral (opcion_t *op, menu_t menu) {
        if (menu == inicio) {
                switch (*op) {
                case ninguno: break;
                case op_a:
                        toggle_op_a (menu);
                        break;
                case op_b:
                        toggle_op_b (menu);
                        break;
                case op_c: break;
                case op_d:
                        toggle_op_d (menu);
                        break;
                case op_numeral:
                        *op = ninguno;
                        if (cambio_de_config == 1) {
                                FLASH_Guardado (contraste, volumen, &version, angulo_alarma, distancia_alarma);
                                cambio_de_config = 0;
                                for (uint8_t i = 0; i < 5; i ++) {
                                        OLED_ToggleRectangulo (101, 19, 126, 47);
                                        OLED_Refresco ();
                                        Delay_ms (250);
                                        if (i != 4) INDICADOR_Enciendo (TIEMPO_INDICACION, 99);
                                }
                                return;
                        }
                }
                OLED_ToggleRectangulo (101, 19, 126, 47);
                *op = op_numeral;
        }
}
```

Proyecto TD II final.c

```
#include "definitions.h"

volatile uint8_t flag_tick = 0;
volatile uint16_t flag_tick_barra = 0;
uint8_t teclas[] = { // Valores de las teclas
        '1', '2', '3', 'A',
        '4', '5', '6', 'B',
        '7', '8', '9', 'C',
```

```
		'*', '0', '#', 'D',
};
uint8_t volumen = 10, volumen_config = 0, contraste = 80, version = 15, cambio_de_config = 0;
int32_t distancia_alarma = 0;
uint8_t unidad_distancia_alarma = 1;
int16_t angulo_alarma = 0;
int8_t signo_angulo_alarma = 1;

int main(void) {
		gravedad_t gravedad;
		int16_t angulo = 0, angulo_resultante = 0, angulo_cero = 0;
		int32_t distancia = 0;
		opcion_t opcion = ninguno;
		menu_t menu = inicio;
		uint8_t boton = '\0', pulsado = 0;
		uint8_t numeros_configuracion = 0;
		uint8_t alarma = 0;
		modo_t modo;
		uint8_t actualizar_menu = 0;

		uint8_t positivo = 0, negativo = 0; // Flags para marcar el signo en menu de mostrar
		uint8_t metros = 0, centimetros = 0;

		FLASH_Lectura (&contraste, &volumen, &version, &angulo_alarma, &distancia_alarma);

		if (angulo_alarma < 0) signo_angulo_alarma = -1;
		if (distancia_alarma > 9999) unidad_distancia_alarma = 100;

		GPIO_Inicializacion ();
		ENCODER_Inicializacion ();
		SCTIMER_Inicializacion ();
		DAC_Inicializacion ();
		SysTick_Inicializacion ();
		I2C_Inicializacion ();
		ADXL345_Configuracion ();
		ADXL345_Apagar ();
		OLED_Inicio ();
		OLED_Contraste (contraste * 255 / 99);
		OLED_CopiarImagen (inicio_1, sizeof (inicio_1));
		OLED_Refresco ();

		while(1) {
			if (numeros_configuracion == 0) {
				boton = TECLADO_Lectura (0, 3, 3, 3);
				// Lectura de la tecla #
				GPIO_PinWrite (GPIO, PUERTO_0, COLUMNA_3, 0);
				if (GPIO_PinRead (GPIO, PUERTO_0, FILA_4) == 0) {
					Delay_ms (50);
					if (GPIO_PinRead (GPIO, PUERTO_0, FILA_4) == 0) {
						boton = teclas[4 * 3 + 2];
					}
				}
				GPIO_PinWrite (GPIO, PUERTO_0, COLUMNA_3, 1);
			}
			else {
				boton = TECLADO_Lectura (0, 0, 3, 2);
			}
			if (boton == '\0') pulsado = 0;
			if (boton != '\0' && pulsado == 0) {
				pulsado = 1;
				INDICADOR_Enciendo (TIEMPO_INDICACION, 99);
				switch (boton) {
				case 'A':
					actualizar_menu = tecla_a (&opcion, &menu, &alarma, &modo,
```

```
                                                    &numeros_configuracion);
                        break;
                case 'B':
                        actualizar_menu = tecla_b (&opcion, &menu, &alarma, &modo,
                                                    &numeros_configuracion, &angulo_cero, angulo,
                                                    &distancia, &gravedad);
                        break;
                case 'C':
                        actualizar_menu = tecla_c (&opcion, &menu, alarma, &angulo_cero, &distancia);
                        break;
                case 'D':
                        actualizar_menu = tecla_d (&opcion, &menu, &gravedad);
                        break;
                case '0': case '1': case '2': case '3': case '4':
                case '5': case '6': case '7': case '8': case '9':
                        tecla_num (&opcion, &menu, &numeros_configuracion, &boton);
                        break;
                case '*': break;
                case '#':
                        tecla_numeral (&opcion, menu);
                        break;
                }
                if (actualizar_menu) {
                        mostrar_menu (menu, &positivo, &negativo, &centimetros, &metros);
                        opcion = ninguno;
                        actualizar_menu = 0;
                }
                OLED_Refresco ();
        }
        menu_independiente (menu, opcion, numeros_configuracion, alarma, &gravedad, &angulo,
                        &angulo_resultante, angulo_cero, &distancia, &positivo, &negativo,
                        &metros, &centimetros);
    }
    return 0 ;
}

void SysTick_Handler (void) {
    flag_tick ++;
    flag_tick_barra ++;
    if (antirrebote_encoder != 0) antirrebote_encoder --;
    if (flag_alarma != 0) flag_alarma --;
    if (flag_buzzer != 0) flag_buzzer --;
    if (flag_buzzer == 0 && DAC_on != 2) {
        DAC_SetBufferValue (DAC1, 0);
        SCTIMER_StopTimer (SCT0, kSCTIMER_Counter_U);
        SCT0->OUTPUT |= (1 << 4);
        DAC_on = 2;
    }
}

void SCT0_IRQHandler(void) {
    if (SCTIMER_GetStatusFlags (SCT0) & (1 << evento_periodo)) {
        SCTIMER_ClearStatusFlags (SCT0, 1 << evento_periodo);
        if (DAC_on) {
            DAC_SetBufferValue (DAC1, 0);
            DAC_on = 0;
        }
        else {
            DAC_SetBufferValue (DAC1, (volumen == 0) ? 0 : (410 + volumen * 31 / 14));
            DAC_on = 1;
        }
    }
}
```

ANEXO VI: REPOSITORIO DE GITHUB

Se adjunta el repositorio de Github en caso de querer visualizar el código de forma más limpia y/o ver los datos de las imágenes.

https://github.com/AlexisDeLaCruzHernandez/Proyecto_Digitales_II