Escuela Técnica Superior de Ingeniería Informática

Ingeniería Informática. Ingeniería de Computadores.

MEMORIA PRÁCTICA II

Asignatura: Sistemas Empotrados y de Tiempo Real I
Profesor: Gabriel Jiménez Moreno

Alumno: Álvaro José Gullón Vega

Índice

1. Objetivos

En esta segunda práctica distinguimos dos objetivos:

1.1. Académico

Los objetivo académicos en este caso, constan de estudiar y entender el funcionamiento de los Displays LCD tradicionales que se usan con los microcontroladores, conocer los aspectos más básicos de la placa de desarrollo de la asignatura y entender las diferencias entre las diversas librerías que proporcionan los fabricantes para el manejo de los periféricos en los microcontroladores tipo Cortex.

1.2. Práctico

Para conseguir estos objetivos académicos, se han propuestos estos objetivos prácticos:

- Mandar texto a cualquier posición en un display tipo HD44780.
- Definir un nuevo dibujo en la memoria del Display y visualizarlo en pantalla
- Manejar los LEDs y el botón de la placa al igual que se hizo en la práctica 1. correspondientes a ese pin y conectadas a VCC o a GND.
- Sustituir en la librería del display LCD, que se proporciona en esta práctica, las llamadas a las funciones GPIO LL por funciones equivalentes GPIO HAL.

2. Introducción

Como hemos mencionado en los objetivos, esta segunda práctica nos acercará a conocer los Display LCD al igual que nos ofrecerá una primera toma de contacto con la placa que usaremos en la asignatura.

Las interfaces visuales más primitivas están basadas en LEDs, un ejemplo de ellos los displays de siete segmentos, los que si agrupamos varios de ellos podemos componer una interfaz visual más completa con un problema que es la cantidad de pines que se requieren para su control. Por otra parte contamos con los displays LCD u OLED, dos tecnologías que han evolucionado mucho los últimos años con las que se construyen un gran número de dispositivos de visualización destinados a su uso con microcontroladores.

Nosotros, en SETR1, nos vamos a centrar en el uso de displays más modestos y primitivos que suelen disponer de dos tipos de interfaces:

- **Paralela:** Consiste en un bus de ocho bits *(D0-D7)* y tres líneas de control *(RS, E, R/W)*. El bus de datos es bidireccional de forma que podemos escribir en el display o podemos leer de él como por el ejemplo el bit de estado.

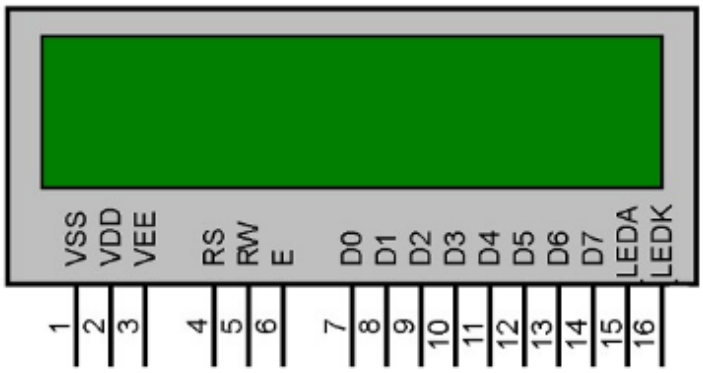

Figura 1: Pinout típico del Display LCD.

- **Serie síncrona:** Usan normalmente puertos *I2C* o *SPI*. Se utilizan dos o tres líneas de conexión con el microcontrolador.

El circuito de control es el cerebro del display, por un lado tiene los pines que sirven de interfaz con el usuario y por otra controla la pantalla de visualización. Casi siempre suelen tener una memoria RAM, excepto los más sofisticados que incluyen de dos en adelante para por ejemplo en la GameBoy organizarlas como memoria principal, memoria de background, memoria de tejas, etc...

La pantalla donde se muestra la información en ocasiones su tamaño es independiente de la capacidad de direccionamiento de la memoria del controlador, y no tienen por qué coincidir. Otro aspecto a tener en cuenta es la resolución de cada carácter, el nuestro tiene 5 columnas por 8 filas de pixel, con 2x16 caracteres y con una anchura total de 6,5 cm.

Para usar el HD44780 se realiza una conexión pin a pin con los GPIO del microcontrolador. Se suele colocar un potenciómetro que lo hace es regular en contraste de la pantalla. Para transferir datos, podemos usar *D0-D7* o *D4-D7*, en este último caso necesitamos dos accesos, primero 4 bits y después los otros 4 bits restantes. La librería software que vamos a usar en esta práctica para el manejo del HD44780 utiliza la interfaz de 4 bits y esperas entre transferencias, sin preguntar por el estado del dispositivo (si está Busy), y por tanto sin utilizar la línea R/W, sólo se hacen escrituras al LCD. Hay muchas librerías en internet para estos LCD y para múltiples plataformas y microcontroladores, ésta que vamos a usar es una de tantas, sin ninguna característica especial.

El software para usar el display HD44780 es muy simple, primero hay una fase de inicialización (función *lcd_ reset()* + función *lcd_ display_ settings(1,0,0)* en la que se configura una serie de características, las más importante son si se va a usar el bus de conexión completo o sólo la parte alta (D0-D7 ó D4-D7) y si la pantalla es de una o dos líneas. En nuestro caso sabremos si la inicialización ha ido bien cuando observemos que se "ven" activas las dos líneas de la pantalla, y no sólo una.

En el HD44780 se pueden hacer las siguientes operaciones:

- **Escritura (función lcd_write(byte, rs):** se transfiere un byte y se indica si es un dato a la memoria o un comando (parámetro rs).
- **Apuntar CGRAM (lcd_cgram_address(uint8_t address)):** cuando se usa esta función se manda un comando que sitúa el puntero de dirección en la CGRAM y apuntando a una determinada dirección.

- **Apuntar a la DDRAM (lcd_display_address(uint8_t address)):** si queremos que vuelvan a visualizarse los datos en la pantalla tenemos que hacer que el apuntador vuelva a apuntar a una dirección de la DDRAM.
- **moveToXY(unsigned char row, unsigned char column):** con ella situamos el cursor en la fila/columna de la pantalla que queramos, como es natural hay que usar esa función primero y después mandar el texto que se quiera ver en la pantalla.
- **lcd_print(char string[]):** sirve para visualizar una cadena de caracteres.
- **writeIntegerToLCD(int integer): se utiliza para mostar un entero.**
- **lcd_clock():** es la encargada de subir y bajar la línea de habilitación E.
- **lcd_clear():** nos permite limpiar el contenido de la pantalla de nuestro display.

En microcontroladores complejos, como los ARM Cortex, el uso de librerías es esencial para desarrollar rápidamente. Aunque el núcleo ARM está estandarizado por la empresa ARM, los periféricos y el hardware adicional varían según el fabricante. ARM proporciona una interfaz estándar llamada CMSIS, útil pero limitada, ya que no incluye funciones prácticas para manejar el hardware específico de cada fabricante.

Para facilitar el desarrollo, cada fabricante ofrece sus propias librerías. Por ejemplo, STMicroelectronics proporciona dos:

- **HAL (Hardware Abstraction Layer):** de alto nivel, fáciles de usar y portables, aunque menos eficientes.
- **LL (Low Layer):** de bajo nivel, más cercanas al hardware, optimizadas y potentes, pero más complejas.

Las HAL intentan ser universales dentro de cada fabricante y son comúnmente utilizadas por principiantes. En cambio, las LL ofrecen mayor control y rendimiento, ideales para desarrolladores con más experiencia. En esta práctica se estudiarán ambas aplicadas a los periféricos GPIO.

Los pines GPIO se agrupan en puertos de 16 bits (GPIOA, GPIOB, etc.). Cada pin se identifica por su puerto y número (por ejemplo, PA1 o PC2). Para manipular estos pines se utilizan registros de 32 bits y funciones específicas. Así, tenemos el siguiente grupo de funciones LL para manejar los GPIO:

```
1        LL_GPIO_IsOutputPinSet(GPIO_TypeDef *GPIOx, uint32_t PinMask)
            // lee uno o varios pines de un
2        puerto de salida
3        LL_GPIO_IsInputPinSet(GPIO_TypeDef *GPIOx, uint32_t PinMask) //
            lee uno o varios pines de un
4        puerto de entrada
5        LL_GPIO_SetOutputPin(GPIO_TypeDef *GPIOx, uint32_t PinMask) //
            pone a uno pin/pines
6        LL_GPIO_ResetOutputPin(GPIO_TypeDef *GPIOx, uint32_t PinMask)
            // pone a cero pin/pines
7        LL_GPIO_TogglePin(GPIO_TypeDef *GPIOx, uint32_t PinMask) //
            complementa pin/pines
```

```
1        LL_GPIO_ReadOutputPort(GPIO_TypeDef *GPIOx) // lee un puerto de
             salida entero
2        LL_GPIO_WriteOutputPort(GPIO_TypeDef *GPIOx, uint32_t
             PortValue) // escribe un puerto entero
```

Por último, vamos a tratar con la placa **B-L475E-IOT01A** con el microcontrolador **STM32L475**:

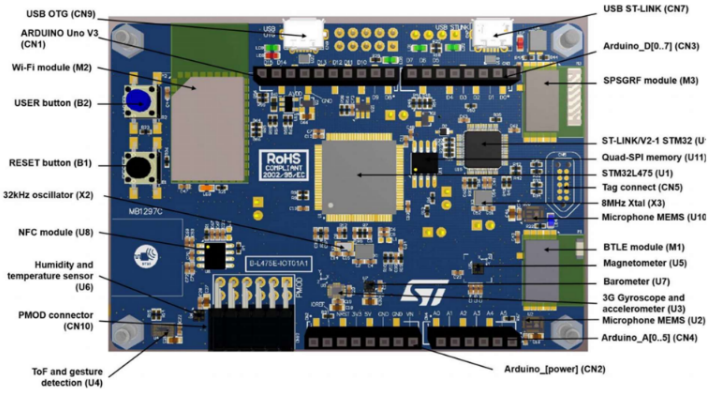

Figura 2: Placa **B-L475E-IOT01A** y elementos que la componen.

Los aspectos más destacables de esta placa que nos conviene conocer es que el depurador ST-LINK vía USB sobre la misma placa que nos permite tres tipos de dispostivos: depurador, unidad de almacenamiento ficticio "masivaz puerto virtual COM. También contiene un conector tipo Arduino.

3. Desarrollo de la práctica

3.1. Fase 1: Usar el display HD44780 y empezar a manejar la placa de la asignatura.

Lo primero que haremos será colocar el shield del display sobre la placa, teniendo en cuenta la orientación de esta para que los pines vayan en su sitio, para que sea más fácil, podemos retirar el .[em]bellecedor"del botón azul para que no haga de escalón y no esté la placa puesta de aquella manera. Una vez colocada, prepararemos el material necesario para tras realizar la fase uno, conectar la placa a nuestro equipo usando el USB y la entrada *USB ST-LINK*. Procederemos a crear un nuevo proyecto y cuando nos salga la ventana de *Target Selection* seleccionamos *Board Selector* y buscamos nuestra placa, colocamos el nombre de nuestro proyecto y cuando nos pregunte si queremos inicializar los periféricos diremos que sí. Cuando se cree, se llevará un proceso que puede tardar bastante pues se está descargando las librerías del generador de código.

Una vez finalizado el proceso anterior, lo primero que vamos a hacer es renombrar los pines del microcontrolador que están conectados al display LCD del shield de Arduino para que su denominación corresponda con el que se pone en la librería del LCD que más adelante copiaremos a nuestro proyecto, por lo que cambiaremos la funcionalidad de los pines a *GPIO_ OUTPUT*. Para saber qué pines cambiar y qué nombres colocarles debemos seguir la siguiente tabla:

Nombre actual	Arduino	Pin MCU	Tipo de puerto		Nombre nuevo que hay que poner
ARD_D10	D10	PA2	Output Push Pull	Low	Led_LCD
ARD_D9	D9	PA15	Output Push Pull	Low	E_LCD
ARD_D6	D6	PB1	Output Push Pull	Low	D6_LCD
ARD_D5	D5	PB4	Output Push Pull	Low	D5_LCD
ARD_D8	D8	PB2	Output Push Pull	Low	RS_LCD
ARD_D7	D7	PA4	Output Push Pull	Low	D7_LCD
ARD_D4	D4	PA3	Output Push Pull	Low	D4_LCD

Figura 3: Pines para usar la librería del display LCD.

A continuación, vamos a seleccionar las librerías LL para manejar los GPIO, esto es necesario pues el fichero que vamos a insertar que maneja el LCD utiliza las funciones de la librería LL para los GPIO. Seleccionamos *Advanced Settings* y después pulsamos sobre la columna HAL en la fila de GPIO y lo cambiamos a LL.

Una vez hecho esto, generamos el código e importamos el fichero .c (código) y .h (cabecera) de la librería del LCD a nuestro proyecto, simplemente arrastrando el fichero hasta la carpeta en el explorador de archivos de nuestro IDE. Debemos importar a nuestro **main.c** la cabecera de la librería usando la siguiente sentencia:

```
#include "hd44780.h"
```

Ahora es turno de probar si funciona correctamente el display, para ello debemos usar el siguiente código antes del *while(1)* para que se inicialice el LCD y se escriba un mensaje:

```
lcd_reset();
lcd_display_settings(1,0,0);
lcd_clear();
lcd_print("Hola mundo");
```

Ahora es momento de conectar nuestra placa al ordenador. Tras conectarla nos pedirá que instalemos el driver del depurador de STM a nuestro PC. Una vez instalado compilaremos nuestro proyecto pulsando sobre *Project* → *Build Project* y posteriormente en *Run* → *Debug Configurations* → *Doble click sobre STM32 Cortex-M C/C++ Application* → *Debug* y se ejecutará el código, pero... sale ennegrecido. ¿Cómo lo solucionamos? Simplemente debemos colocar la instrucción:

```
LL_GPIO_SetOutputPin(GPIOA, Led_LCD_Pin);
```

En mi caso, mi display LCD era nuevo a estrenar, entonces he tenido que ajustar el potenciómetro del contraste para que se vea correctamente.

A continuación, vamos a definir un nuevo carácter que se pueda visualizar en el LCD siguiendo este código:

```
lcd_cgram_address(0); se selecciona la direccion 0 de la CGRAM,
   patron con codigo 0
lcd_write(0x15, 1); se va cargando con los bytes del patron de cada
   fila de la CGRAM
lcd_write(0x15, 1);
lcd_write(0xe, 1);
lcd_write(0x4, 1);
lcd_write(0x4, 1);
lcd_write(0xa, 1);
lcd_write(0x11, 1);
lcd_write(0x00, 1);
```

```
lcd_cgram_address(8); se selecciona la direccion 8 de la CGRAM, patron con codigo 1
lcd_write(0x4, 1); se va cargando con los bytes del patron de cada fila de la CGRAM
lcd_write(0x4, 1);
lcd_write(0xe, 1);
lcd_write(0x15, 1);
lcd_write(0x15, 1);
lcd_write(0xa, 1);
lcd_write(0xa, 1);
lcd_write(0x00, 1);
lcd_display_address(0); //Vuelve a direccionar la DDRAM para visualizar datos en pantalla
```

En esas líneas de programa se definen dos nuevos patrones o caracteres, cuyo código serían el 0 y el 1 (son los punteros a dichos patrones), es decir, como si fueran dos nuevos caracteres con códigos ASCII 0 y 1. Aquí cabe señalar que para muchos displays LCD gráficos (no para este HD44780) existe software de ayuda que permite el diseño de patrones y de pantallas. Vamos a insertar esas líneas de código en el proyecto actual, debajo de la última línea en la que hayamos escrito en el LCD, y antes del while(1). Insertemos ahora, dentro del while(1), las siguientes líneas de código:

```
moveToXY(1,7); //situa el cursor en la posicion fila 1 columna 7, primer muneco
lcd_write(0,1); //escribe el nuevo patron 0 en la DDRAM
moveToXY(1,8); //situa el cursor en la posicion fila 1 columna 7, segundo muneco
lcd_write(0,1);
HAL_Delay(1000); //retraso de 1 segundo para dejar ver la figura en pantalla
moveToXY(1,7);
lcd_write(1,1); //escribe el nuevo patron 0 en la DDRAM
moveToXY(1,8);
lcd_write(1,1);
HAL_Delay(1000);
```

Tras estos pasos, hemos dibujado en pantalla a dos monigotes haciendo gimnasia con los brazos y las piernas y acabado la fase 1.

3.2. Fase 2: Repetir la Fase 2 y 4 de la práctica 1 con la nueva placa.

Para llevar a cabo esta fase dos de la práctica uno en la que se enciende y se apaga un LED (como por ejemplo el que podemos identificar como *LED2*) de la placa mediante la pulsación de un botón (en nuestro caso el azul que viene incluido en la práctica y que podemos identificar como *BUTTON_EXTI13*) como en la práctica 1, simplemente debemos incluir este código dentro del while(1) en nuestro fichero **main.c**:

```
if (LL_GPIO_IsInputPinSet(BUTTON_EXTI13_GPIO_Port, BUTTON_EXTI13_Pin))
    LL_GPIO_SetOutputPin(LED2_GPIO_Port, LED2_Pin);
else
    LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);
```

Pero ocurre un problema, debido a los *HAL_Delay(1000)* impiden que cuando pulsamos el botón el LED se apague inmediatamente. Para ello debemos tratar con la interrupción del botón o la del Systick, o en nuestro caso implementar algo como esto:

```
    int contador=0, estado_mu=0;
    while (1) {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    //TAREA A
    contador++;
    if(contador==10 && estado_mu==0){
        moveToXY(1,7); //situa el cursor en la posicion fila 1 columna
           7, primer muneco
        lcd_write(0,1); //escribe el nuevo patron 0 en la DDRAM
        moveToXY(1,8); //situa el cursor en la posicion fila 1 columna
           8, segundo muneco
        lcd_write(0,1);
        contador = 0;
        estado_mu = 1;
    }

    if(contador==10 && estado_mu==1){
        moveToXY(1,7);
        lcd_write(1,1); //escribe el nuevo patron 0 en la DDRAM
        moveToXY(1,8);
        lcd_write(1,1);
        contador = 0;
        estado_mu = 0;
    }

    if (LL_GPIO_IsInputPinSet(BUTTON_EXTI13_GPIO_Port,
       BUTTON_EXTI13_Pin))
    LL_GPIO_SetOutputPin(LED2_GPIO_Port, LED2_Pin);
    else
    LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);
    HAL_Delay(100);
}
```

Para repetir la fase cuatro de la práctica uno, nos basta con copiar el código proporcionado en el boletín de la práctica para escribirlo en el fichero *stm32l4xx_it.c* en la función *SysTick_Handler(void)* y con esto quedaría acabada esta fase.

3.3. Fase 3: Repetir la fase 1 de esta práctica, pero usando librerías HAL.

Esta fase es sencilla, ya que nos proporcionan las instrucciones para tratar con la librería HAL, que son estas:

```
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, 0); pone a cero el pin 5 del
       puerto A
    HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13); devuelve el valor del pin 13
       del puerto C
    HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
       complementa un pin
```

Un ejemplo de cómo se tratan sería el siguiente en el que están estas dos instrucciones que son **equivalentes**, solo que estamos tratando con instrucciones HAL:

```
        LL_GPIO_SetOutputPin(LED2_GPIO_Port, LED2_Pin);

        HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
```

Con esto, ya estaría acabada esta fase y por ende esta práctica dos.

4. Conclusiones

En mi opinión he disfrutado con esta práctica ya que he conseguido los objetivos propuestos y he aprendido mediante la realización de todas las fases a tratar con un display LCD para desempeñarlo en futuros proyectos puesto que es un elemento muy usado para mostrar información. Esta práctica ayuda también porque es muy visual y proporciona una sensación de satisfacción cuando ves que salen las cosas.