MEMORIA DE PRÁCTICAS DE LABORATORIO

UNIVERSIDAD DE SEVILLA

DIMAS MARTIN GONZALEZ

Imagen que contiene electrónica, circuito

Descripción generada automáticamente

# PRACTICA 1

## OBJETIVOS.

### OBJETIVOS ACADEMICOS

El principal objetivo académico de esta práctica es introducir el uso de los sistemas de desarrollo de los microcontroladores actuales. Además de saber realizar la Instalación del sistema de desarrollo STM32CubeIDE y del simulador QEMU. Aprenderemos los conceptos básicos del sistema de desarrollo y del simulador.

### OBJETIVOS PRACTICOS

Los objetivos de implementación y fases de esta práctica son:

Fase 1: Instalación del sistema de desarrollo STM32CubeIDE y del simulador QEMU

Fase 2: Probar la instalación encendiendo un LED al pulsar un botón.

Fase 3: Utilizar la interrupción de pulsar botón para cambiar el estado del Led.

Fase 4: Hacer que le LED se encienda y apague a un determinado ritmo.

## INTRODUCCION

En esta práctica se empleará el entorno de desarrollo STM32CubeIDE, ofrecido de manera gratuita por STmicroelectronics. Este entorno, basado en ECLIPSE, amalgama varias herramientas de desarrollo que se han ido fusionando y evolucionando en un solo entorno: SW4STM32 + Atollic TrueStudio + STM32CubeMX. Mientras que los primeros son entornos de desarrollo integrados (IDE), el último es un generador automático de código de configuración para microcontroladores STM. Este generador no solo puede producir código para Atollic, sino también para otros entornos de desarrollo como KEIL, IAR, entre otros.

En esta práctica, procederemos a su instalación y exploraremos sus funciones fundamentales para familiarizarnos y desenvolvernos en el entorno.

En cuanto al simulador QEMU, dado que la primera sesión se realizará en línea y aún no se dispone del material de prácticas, se empleará un simulador de una placa de desarrollo básica para realizar pruebas.

La placa con la que se va a probar el sistema está basada en un STM32 Cortex M0, con muy pocos periféricos. Sin embargo, con la que se hará el resto de las prácticas en esta asignatura es una placa STM32 Cortex M4 con una gran cantidad de periféricos.

Este tipo de simuladores tiene una pequeña desventaja, su interacción con el exterior es bastante escasa, pero al menos son gratuitos y sí que podemos seguir el código que se ejecuta en el microcontrolador simulado con los depuradores integrados en el IDE.

## DESARROLLO DE LA PRACTICA

### FASE 1

Iniciamos descargando la versión 1.3.0 del enlace proporcionado, siguiendo los pasos sugeridos para realizar la instalación de manera adecuada. Para la instalación del simulador QEMU desde el entorno de desarrollo mismo, seguiremos las instrucciones proporcionadas en la práctica. Es importante tener en cuenta que el proceso de instalación del simulador puede presentar fallos y errores. En mi caso, no he experimentado ningún problema y todo funcionó correctamente en el primer intento de instalación. Posteriormente, será necesario descargar el paquete de simulación para los ARM Cortex a través del enlace sugerido y seguir los pasos proporcionados para instalarlo en nuestro ordenador.

### FASE 2

Inicialmente, creamos un proyecto vacío para verificar que todo funcione correctamente. Seleccionamos la familia de microcontroladores adecuada y permitimos que el programa descargue todo lo necesario, dado que es la primera vez que se ejecuta este proceso. Una vez que el proyecto está creado, seguimos las instrucciones para generar código y utilizar todos los elementos del microcontrolador.

Cada vez que modifiquemos algún parámetro, es necesario regenerar el código. En el archivo main.c, colocamos nuestro código, nuestros includes, etc.

Para llevar a cabo la tarea del LED, debemos incluir el siguiente código en el while():

Texto, Carta

Descripción generada automáticamente

Compilamos el proyecto. Dado que es la primera vez que realizamos este proceso, necesitamos especificar la ubicación del simulador, siguiendo los pasos indicados en la práctica. Una vez completado esto, cargamos el código en el simulador a través del debug QEMU, siguiendo nuevamente las instrucciones de la práctica. Aunque es un proceso meticuloso que requiere atención y seguir las instrucciones al pie de la letra, no es complicado.

Después de esto, pulsamos "Resume" en el IDE para ejecutar el código. Si observamos una ventana abierta con una imagen de la placa, podemos interactuar con el botón azul para encender y apagar el LED. Verificamos que el LED se enciende mientras mantenemos presionado el botón, asegurándonos de que todo funciona correctamente.

Al poner un breakpoint en la línea de código que hemos agregado y ejecutarlo, la ejecución se detiene ahí, lo que significa que el LED no cambiará incluso si presionamos el botón.

### FASE 3

Ahora procederemos a configurar el botón para encender el LED al pulsarlo y apagarlo al pulsarlo nuevamente, sin necesidad de mantenerlo presionado. En primer lugar, cambiaremos a la perspectiva MX para visualizar el chip. Desde aquí, nos dirigimos a "System View" y seleccionamos "GPIO" para verificar que la configuración de los pines sea la correcta.

Luego, nos dirigimos a la configuración del controlador de interrupciones, accediendo a "NVIC". En esta pestaña, verificamos que la configuración de las interrupciones sea la adecuada, conforme nos indica la práctica. Para localizar las rutinas de interrupción, navegamos al archivo STM32F1xx\_it.c, donde encontraremos dichas rutinas. La última de estas rutinas corresponde al pin externo que llama a otra subrutina. Para localizar esta subrutina, simplemente hacemos clic con Ctrl en la función correspondiente, lo que nos llevará al archivo donde se encuentra la subrutina, que en este caso es STM32F1xx\_hal\_gpio.c.

Para llevar a cabo la tarea propuesta, necesitamos utilizar la función de callback en el archivo main.c. También debemos comentar el código que habíamos añadido anteriormente. El resultado final sería algo similar a lo siguiente:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Volviendo a compilar y depurar como lo hicimos anteriormente, podemos verificar que ahora el LED responde como se espera. La variable “estado” se declara como “static”, lo que significa que solo se ejecutará la primera vez que el programa se ejecute. Si eliminamos el calificador “static”, la variable se inicializaría a 0 en cada ejecución, lo que provocaría un funcionamiento incorrecto.

### FASE 4

Para realizar una intermitencia con el LED, utilizaremos el temporizador del sistema Systick. Para ello, nos dirigimos directamente a la función de interrupción, ubicada en el archivo stm32f1xx\_it.c, justo antes de la función que mencionamos anteriormente.

Copiamos y pegamos el mismo código que utilizamos entre los dos primeros comentarios, quedando de la siguiente manera:

Texto

Descripción generada automáticamente

Compilamos y depuramos para observar su funcionamiento.

*¿Es necesario que comentemos el código anterior que controla el Led con el botón azul?*

No, no es necesario comentar el código anterior, dado que este no hace uso del botón y aquí el botón no es necesario en absoluto.

*¿Y si quisiéramos que parpadeara más lento el LED, por ejemplo 16 veces más lento, sin cambiar nada del código?*

Para disminuir la velocidad de parpadeo, reduciremos la frecuencia del reloj como indica la práctica. Una vez ajustado a /16, procedemos a compilar y depurar nuevamente para verificar que, efectivamente, se reduce la velocidad de parpadeo.

## CONTESTACION DE PREGUNTAS

*1.-Buscar en la Wikipedia que es el Qemu y hacer un resumen en cinco líneas reflexionando sobre qué es lo que hemos montado en esta práctica.*

QEMU es un emulador de procesadores que se basa en la traducción dinámica de binarios. Su objetivo principal es permitirnos emular hardware para el cual no disponemos en nuestro sistema operativo, todo esto sin necesidad de reparticionar el disco duro. De manera general, esto implica la capacidad de simular un entorno de hardware completo dentro de nuestro sistema operativo actual. Esto resulta particularmente útil en situaciones donde necesitamos probar o desarrollar software para plataformas específicas sin tener acceso físico a dicho hardware. En esta práctica, hemos podido ubicar el simulador en cualquier parte de nuestro disco duro, lo que ilustra la flexibilidad y la utilidad de esta herramienta.

*2.-¿Qué podríamos haber hecho en nuestro código para que el diodo parpadeara 16 veces más lento que al principio de la fase 4 sin tocar para nada la configuración del reloj del sistema?*

Modificar la función SysTick\_Handler, de forma que quedaría tal que así:

Texto

Descripción generada automáticamente

*3.-La intermitencia del LED se realiza con precisión, pero con valores un poco al azar. Calcule con los parámetros que se han usado cuánto tiempo está encendido el LED y cuánto tiempo está apagado. Y ahora, al contrario, busca una combinación de la configuración del reloj HCLK (con el CubeMX) y el valor de recarga del SYSTICK que haga que el Led esté con una intermitencia de 0,5 segundos encendido y 0,5 segundos apagado. En la figura del esquema del timer Systick hay una expresión en la que se muestra el cálculo del periodo de interrupción.*

Inicialmente, consideramos que el registro tiene una longitud de 24 bits, lo que nos proporciona 2^24 posibilidades.

Con cada ciclo de reloj, este valor se decrementa hasta llegar a 0, momento en el cual se genera la interrupción y se enciende el LED.

Dado que la frecuencia del reloj es de 64MHz, podemos calcular el tiempo necesario de la siguiente manera:

Texto

Descripción generada automáticamente

El resultado obtenido representa el tiempo necesario para que se active la interrupción, es decir, para encender o apagar el LED. En cuanto a la segunda parte de la pregunta, podemos observar en el archivo de configuración (".ioc") que en las "Configuraciones de Reloj" ("Clock Configurations") se encuentran los diferentes valores de frecuencia para HCLK. Al modificar el valor del preescalador AHB ("AHB Prescaler"), podemos dividir la frecuencia del reloj HCLK.

Es posible probar con distintos valores hasta encontrar aquel que sea adecuado para que la interrupción se produzca cada 0.5 segundos.

Imagen que contiene Texto

Descripción generada automáticamente

Con una frecuencia de 32MHz, el valor de recarga obtenido es de 15999999, lo cual es válido ya que se encuentra dentro del rango de 2^24.

## CONCLUSIONES

Esta práctica ha sido una experiencia nueva para mí. He adquirido conocimientos básicos sobre el microcontrolador y el entorno de desarrollo utilizado, lo que me permitirá manejarme con soltura al realizar tareas básicas. El simulador resulta fácil de entender y usar, pero es verdad que la interacción con la placa física en prácticas futuras será más interesante.

Aunque la práctica inicial ha sido positiva y novedosa en comparación con otras asignaturas, el proceso de adaptación/aprendizaje podrá llevar tiempo y esfuerzo adicional para completar los objetivos de manera satisfactoria en prácticas futuras.

En resumen, aunque hay aspectos positivos en la experiencia de aprendizaje, se espera que el desafío y la curva de aprendizaje sean mayores en las prácticas siguientes.

# PRACTICA 2

## OBJETIVOS

### OBJETIVOS ACADEMICOS

En esta práctica vamos a estudiar y entender el funcionamiento de los displays tradicionales que se usan con los microcontroladores, así como 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 (HAL: librería de abstracción de hardware; LL: librería de nivel bajo).

### OBJETIVOS PRACTICOS

Para conseguir estos objetivos académicos se nos ha planteado los siguientes objetivos prácticos:

1.- Mandar texto a cualquier posición en un display tipo HD44780.

2.-Definir un nuevo dibujo en la memoria del display y visualizarlo en pantalla.

3.- Manejar los LEDs y el botón de la placa al igual que se hizo en la práctica 1.

4.- 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.

## INTRODUCCION

La mayoría de los sistemas empotrados precisan de alguna interfaz con el usuario, ya sea una lavadora, un horno, un ascensor… es necesario algún tipo de elemento de manipulación. Las interfaces más simples son los botones y las luces (LEDs), pero hay veces que se requiere mostrar más información, y en estos casos las pantallas tipo display son la solución. En esta práctica veremos cómo funcionan los displays más sencillos que se suelen usar con los microcontroladores, nos referimos a dispositivos universales y de fácil adquisición (hay otros tipos de displays, que se parecen mucho a los que vamos a explicar, pero que están hechos para adaptarse a un determinado aparato, con unos determinados mensajes, este tipo de displays no serán tratados en esta práctica). Por otra parte, vamos a introducir la placa de desarrollo con la que vamos a realizar casi todas las prácticas este curso y el que viene. También se pretende aclarar las diferencias entre las librerías para el uso de periféricos que proporciona STMicroelectronic, en realidad estas diferencias sólo se pueden apreciar cuando se utiliza una y otra librería, y cuando surge el problema y comprobamos que una no proporciona la funcionalidad que la otra tiene.

Tal y como hemos visto en clase, nuestra placa tiene principalmente dos chips (además de los pertinentes puertos micro-USB que sirven para dar alimentación a la placa, LEDs, botones, etc.), que se ven a simple vista, los cuales son los más importantes, estos son el debugger y el propio microcontrolador STM32. El chip que está situado en el centro de la placa es el microcontrolador, y el que está situado a la derecha es el debugger (como he comentado, los puertos micro-USB alimentan a la placa, pero dependiendo cual se conecta, pasar por el debugger o no).

Aunque realmente el PC ve esta placa de desarrollo como 3 dispositivos:

1. Un depurador (debugger), y parte de este se encuentra dentro del microcontrolador.
2. El propio micro.
3. Un disco duro (pero virtual, claro, ya que no almacena nada, solo el código del microcontrolador).

Para el correcto funcionamiento de nuestra placa será necesaria la instalación del driver STLINK.

Como nota, una vez instalado, nuestro PC lo vera como un COM virtual.

Nuestro display LCD se compone principalmente de tres elementos:

1. La pantalla (donde se mostrarán los mensajes).
2. El controlador (el cual se comunica con el microcontrolador para recibir datos e instrucciones y luego controlar la visualización de los pixeles en el panel LCD).
3. Las interfaces (utilizadas por el controlador para realizar dichas comunicaciones. Pueden variar).

Además, nuestro LCD dispone de tres memorias, una que se le asocia la posición del LCD, otra memoria de caracteres ROM y una memoria RAM para la definición de caracteres.

El funcionamiento es el siguiente:

En la memoria del LCD (en la posición 0, p.e.) están las posiciones de memoria de caracteres, dependiendo de lo que se ponga (0x31, p.e., que es un 1), se mostrara en la posición 0 del LCD un 1. En la RAM simplemente el puntero apunta al del display para que se muestre en el LCD.

## DESARROLLO DE LA PRACTICA

### FASE 1

#### RENOMBRAMIENTO DE PINES DEL MICROCONTROLADOR

Así quedarían los pines una vez modificados. Tabla

Descripción generada automáticamente

Imagen que contiene Tabla

Descripción generada automáticamente

#### SELECCIÓN DE LIBRERIAS

Hay que escoger en esta primera fase LL en GPIO.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente con confianza media

#### CONFIGURACION DEL PROYECTO

Añadiendo el hd44780.h y el hd44780.c, en Inc y Src, respectivamente.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

#### CODIGO MODIFICADO EN MAIN

El include del header del hd44780.

Forma, Rectángulo

Descripción generada automáticamente

Código para mostrar el texto ‘Hola Dimas’ y el muñeco haciendo gimnasia.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Código dentro del while para ver la ‘animacion’.

Texto, Carta

Descripción generada automáticamente

### FASE 2

Si ponemos el código propuesto en el while(1), podemos ver que si pulsamos el botón azul de la placa, el led se apaga pero no es inmediato, debido a los HAL\_Delay(1000) que usamos para mover el dibujo. Si quitamos estos HAL\_Delay se apagará instantáneo pero los dibujos se moverán demasiado rápido. Para arreglar este problema debemos copiar el código propuesto en otro lugar. Concretamente en el fichero stm32l4xx\_it.c, en la funcion del SysTick\_Handler. Así se resolverá este problema y podremos controlar el LED de forma instantánea sin afectar a la velocidad de movimiento del dibujo.

El código quedaría tal que así:

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza baja

Para realizar la fase 4 de la práctica 1 debemos de hacer uso del mismo código que usamos en dicha práctica, pero esta vez usando la librería LL.

El código también iría en la función SysTick\_Handler del fichero stm32l4xx\_it.c (podemos comentar el código que hemos usado antes para el botón).

Quedaría de la siguiente forma:

Texto

Descripción generada automáticamente

Si compilamos y ejecutamos podremos ver ahora como el LED funciona de forma intermitente si necesidad de usar el botón.

### FASE 3

#### SELECCIÓN DE LIBRERIAS

Ahora en vez de LL usaremos HAL, en GPIO.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

#### CODIGO MODIFICADO EN MAIN

\*Importante\* Cambia la forma de escribirse para HAL.

Texto

Descripción generada automáticamente

#### CODIGO MODIFICADO EN LA LIBRERÍA

Código del Reloj:

Texto

Descripción generada automáticamente

Código del Reset:

Texto

Descripción generada automáticamente

Código del Lcd\_write:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## CONTESTACION DE PREGUNTAS

*1.-Queremos hacer dos tareas en paralelo, que los muñecos se muevan al ritmo que se movían antes y que cuando pulsemos el botón, el LED responda de forma inmediata (aparentemente inmediata…). Hay varias soluciones para esto: la expuesta en el tema 1 de teoría (explicarla en la memoria de esta práctica).*

Solución dada:

int contador = 0;

int estado\_mu = 0;

while (true) { // TAREA A

contador++;

if (contador == 10 && estado\_mu == 0) {

moveToXY(1, 7);

lcd\_write(0, 1);

moveToXY(1, 8);

lcd\_write(0, 1);

contador = 0;

estado\_mu = 1;

} if (contador == 10 && estado\_mu == 1) {

moveToXY(1, 7);

lcd\_write(1, 1);

moveToXY(1, 8);

lcd\_write(1, 1);

contador = 0;

estado\_mu = 0;

}

// TAREA B

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); // TEMPORIZACION

}

El ejercicio propone dos tareas que deben ejecutarse en paralelo:

Tarea A: Controlar el movimiento de los "muñecos".

Tarea B: Controlar la respuesta del LED al pulsar un botón.

Para garantizar que estas tareas se ejecuten en paralelo, es necesario tener en cuenta la forma en que se estructura el código y cómo se gestionan los eventos.

En el primer código proporcionado, el bucle principal se encarga de ejecutar ambas tareas de manera secuencial. Esto significa que primero se ejecuta la Tarea A y luego la Tarea B en cada iteración del bucle. Sin embargo, esto no es ideal para asegurar una respuesta inmediata al pulsar el botón.

Para mejorar la respuesta al botón, se puede utilizar un sistema de interrupciones. Las interrupciones son eventos que pueden ocurrir de manera asíncrona con respecto al flujo normal del programa. Cuando ocurre una interrupción, el procesador detiene temporalmente la ejecución del código principal para atender la interrupción. Esto permite manejar eventos externos de manera más rápida y eficiente.

Esta configuración garantiza que las dos tareas se ejecuten en paralelo y que el LED responda de manera inmediata al pulsar el botón, sin afectar el funcionamiento de la Tarea A.

## CONCLUSIONES

# PRACTICA 3

## OBJETIVOS

Los objetivos de esta práctica consisten en:

1. Ilustrar el conversor Analógico – Digital en STM32.
2. Digitalizar la botonera analógica del LCD-Shield.
3. Mostar el resultado en el LCD del shield.
4. Ilustrar el puerto serie en STM32.
5. Enviar la información de la botonera al PC a través de un puerto serie virtual.
6. Monitorizar el estado de la botonera en un terminal en el PC.

Siguiendo los siguientes pasos:

1. Crear un proyecto y configurar el STM32L475.
2. Comprobar el funcionamiento del display.
3. Leer el ADC y mostrar el resultado en pantalla.
4. Escalar las lecturas para la posterior clasificación de los botones.
5. Leer sensor de presión LPS22HB.
6. Leer sensor de temperatura/humedad HTS221.
7. Representar información de ambos de sensores.
8. Cálculo de valores máximos y mínimos.
9. Cambio dinámico de pantallas a partir del botón pulsado.
10. Configurar el puerto serie.
11. Envío básico de información al PC.
12. Redireccionar printf y mostrar la información en el PC.

## INTRODUCCION

En esta práctica nos dan a conocer 2 conceptos fundamentales del mundo de los sistemas empotrados:

1. La conversión Analógica-Digital.
2. La comunicación serie.

La conversión Analógica-Digital (ADC) es un proceso fundamental en la electrónica y la ingeniería de señales que implica la conversión de una señal analógica, que es continua en el tiempo y en amplitud, en una secuencia de números digitales discretos.

Algunos puntos adicionales:

**Proceso de Conversión Bidireccional:** Además de la conversión de analógico a digital (ADC), también existe la conversión de digital a analógico (DAC). Mientras que ADC convierte una señal analógica en una secuencia de números digitales, DAC hace lo contrario, convierte una secuencia de números digitales en una señal analógica.

**Resultados de la Conversión:** La salida de un proceso de conversión ADC se llama muestra o sample. Esta muestra es una representación digital de la señal analógica en un instante de tiempo específico.

**Resolución:** La resolución de ADC se refiere al número de bits utilizados para representar cada muestra. Por ejemplo, una ADC de 12 bits puede representar la señal analógica en 4096 niveles discretos (2^12).

**Frecuencia de Muestreo:** Es la cantidad de muestras tomadas por segundo y se mide en muestras por segundo (Samples per second, SPS) o Hz. La frecuencia de muestreo determina la cantidad de información capturada de la señal analógica. Un concepto relacionado es el periodo de muestreo, que es el tiempo transcurrido entre cada muestra.

**Ejemplo de Tarjeta de Sonido:** En el contexto de una tarjeta de sonido, las muestras se toman a una frecuencia de muestreo de 96kSamples/sec, lo que significa que se toman 96,000 muestras por segundo. Además, se menciona que estas muestras son de 24 bits, lo que indica que cada muestra se representa utilizando 24 bits de información digital. Esto proporciona una alta resolución para capturar detalles finos en la señal de audio.

La comunicación serie es un método ampliamente utilizado para transmitir datos entre dispositivos electrónicos.

Algunos puntos clave:

**Puertos Serie Asíncronos:** Estos puertos son muy comunes en una variedad de dispositivos, desde GPS y sensores digitales hasta modems telefónicos y dispositivos Arduino. Proporcionan una forma simple de comunicar microcontroladores entre sí.

**Puertos Serie en PCs:** En los PCs, los puertos serie se denominan puertos COM y se enumeran como COMx. Aunque el uso de puertos COM físicos ha disminuido con la popularidad de USB, los puertos COM virtuales han surgido como puentes USB-Serie, como los proporcionados por los chips FTDI y Prolific.

**Transmisión Serie Asíncrona:** Este método de transmisión utiliza dos líneas de datos, Tx (transmitir) y Rx (recibir). No hay una línea de reloj dedicada; en su lugar, el transmisor y el receptor deben tener configurado previamente el tiempo de bit (baudrate). La transmisión comienza con un bit de start y termina con un bit de stop, y el receptor muestrea la línea de recepción periódicamente para reconstruir la información.

**Interfaz con el PC:** El ST-Link, que es un depurador y programador de microcontroladores STM32, implementa un perfil USB llamado Virtual COM Port (VCP). Esto permite que el PC detecte el ST-Link como un puerto COM adicional, lo que facilita la comunicación directa entre el microcontrolador STM32 y el PC a través de una consola serie.

**USART en el STM32:** El microcontrolador STM32L475 tiene 5 USARTs (Universal Synchronous/Asynchronous Receiver/Transmitter) independientes que admiten operación full-duplex. Esto significa que pueden transmitir y recibir datos simultáneamente. Además, estos USARTs permiten la transmisión y recepción de datos mediante polling, interrupciones o DMA (Acceso Directo a Memoria).

## DESARROLLO DE LA PRACTICA

### FASE 1

Creamos un nuevo proyecto y configuramos el entorno siguiendo la guía de la práctica.

Una vez que hemos guardado las configuraciones hechas y hemos generado código, incluimos las librerías del LCD. A continuación, vamos a comprobar que el funcionamiento del LCD sea el correcto:

Texto, Carta

Descripción generada automáticamente

#### Leyendo el ADC

La lectura del ADC se realiza en 3 fases. Las librerías HAL nos proporcionan una función específica para cada uno de ellos:

1. Iniciar una conversión por software (Software Trigger):

• HAL\_ADC\_Start(&hadc1);

2. Esperar a que finalice la conversión:

• HAL\_ADC\_PollForConversion(&hadc1, timeOut);

3. Lectura del registro de datos:

• sample = HAL\_ADC\_GetValue(&hadc1);

#### Mostrando el resultado de la conversión

Texto, Carta

Descripción generada automáticamente

#### Convirtiendo el valor de la muestra a voltios

Diagrama

Descripción generada automáticamente con confianza media

Para obtener el voltaje de entrada, debemos despejar Ain de la ecuación.

Para evitar los números en coma flotante, podemos representar las muestras como miliVoltios, tomando Vref como 3300miliVoltios.

Imagen que contiene Texto

Descripción generada automáticamente

#### Mostar el voltaje en el display

Texto

Descripción generada automáticamente

#### Determinar el botón pulsado

Escribir una función que reciba el valor de la muestra y devuelva el id del botón pulsado:

Texto

Descripción generada automáticamente con confianza media

#### Mostar el id del botón en el display

Texto

Descripción generada automáticamente con confianza media

### FASE 2

La librería HAL nos proporciona 2 funciones para hacer transferencias I2C.

• Lectura: HAL\_I2C\_Mem\_Read(&hi2c2, slave\_dir, sub\_dir, I2C\_MEMADD\_SIZE\_8BIT, buffer, nBytes, timeOut)

• Escritura: HAL\_I2C\_Mem\_Write(&hi2c2, slave\_dir, sub\_dir, I2C\_MEMADD\_SIZE\_8BIT,buffer, nBytes, timeOut)

Previamente se deben almacenar en un buffer los datos a escribir.

Las transferencias se pueden hacer de nBytes, comenzando por la subdirección e incrementándose automáticamente byte a byte.

A partir de este punto vamos a generar nuevas configuraciones en el proyecto para poder activar el I2C.

#### Creando una librería

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

#### Leyendo la presión

Texto, Carta

Descripción generada automáticamente

### FASE 3 (FASE 4 en la guía)

Al igual que el LPS22H, tiene una dirección en el I2C propia y un mapa de registros internos.

Como diferencia, hay que habilitarle explícitamente el autoincremento de la subdirección para permitir lecturas de múltiples direcciones consecutivas.

#### Creando una librería

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

#### Leyendo el HTS211: Mostrando T/H en el display

En el bucle principal lanzar lecturas y escribirlas en el display.

Texto, Carta

Descripción generada automáticamente

#### Ampliación para casa

Modificar el proyecto para que:

• Cuando se pulse el botón “UP” se muestren los valores máximos registrados.

• Cuando se pulse el botón “DOWN” se muestren los valores mínimos registrados.

• Cuando se pulse el botón “LEFT” se muestren los valores medios calculados.

• Cuando se pulse el botón “RIGHT” se reseteen los máximos/mínimos y media.

Texto, Carta

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Imagen que contiene Tabla

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

### FASE 4

Volvemos a la herramienta de configuración. Vamos a configurar primero los pines, a continuación, la USART.

Guardamos y generamos código.

#### Transmitiendo datos

Para transmitir datos por el puerto serie tenemos la función:

• HAL\_UART\_Transmit(uart, pBytes, bytesLength, timeOut);

#### Abriendo el terminal

Vamos a usar el terminal serie integrado en el STM32CubeIDE.

#### Redireccionando printf

Los microcontroladores suelen proporcionar mecanismos para redireccionar la salida de datos de printf. Es común redireccionarlos hacia el puerto serie. De manera que podemos llamar a printf en cualquier parte de nuestro programa y enviar datos con formato de forma fácil.

La función: \_\_io\_putchar(int ch), recibe el carácter que debemos imprimir. Utilizamos la función HAL\_UART\_Transmit para enviar el carácter ch al puerto serie huart1.

Texto

Descripción generada automáticamente

Nombre de la empresa

Descripción generada automáticamente con confianza media

## CONCLUSIONES

# PRACTICA 4