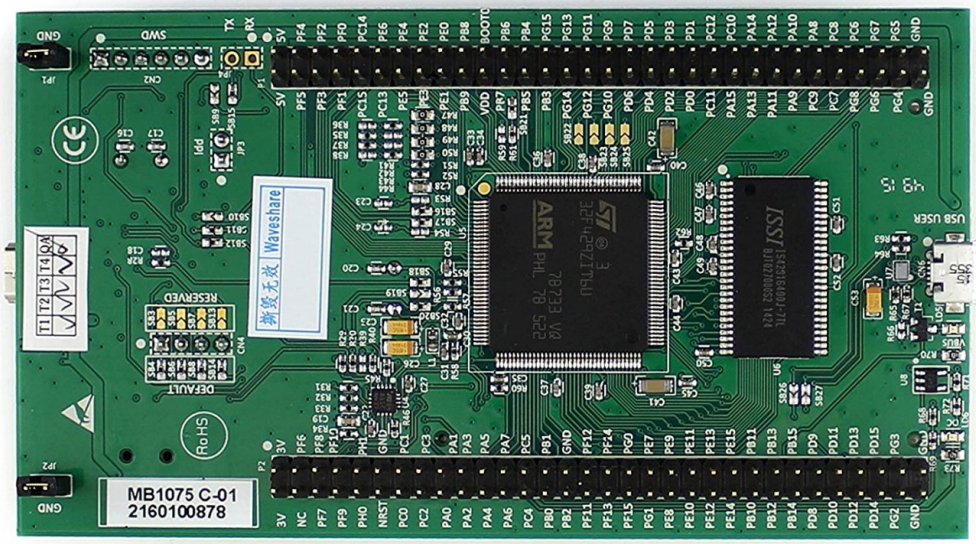
Sistemas de Control en Tiempo Real
Práctica 1

En esta práctica elaboraremos código en lenguaje C para manejo de una pantalla táctil en una placa STM32F429I-DISC1



La placa STM32F492I-DISC1 dispone de:

- un microcontrolador STM32F429ZIT6 de 32 bits con
  - núcleo ARM Cortex-M4 con unidad FPU de cálculo en punto flotante
  - reloj de hasta 180 MHz
  - 2 MiBytes de memoria Flash
  - 256 KiBytes de memoria RAM
  - interfaz con pantallas LCD
  - acelerador gráfico Chrom-ART para transferencias con una pantalla independientes de la CPU
  - hasta 3 ADCs de 12 bits para 16 entradas analógicas y 2 DAC de 12 bits
  - hasta 17 temporizadores con interfaz para encóders incrementales

  - interfaces SWD y JTAG para depuración de aplicaciones

  - hasta 168 E/S digitales con capacidad de interrupción

  - hasta 21 canales de comunicación I^2C, SPI, USART, CAN, SAI, SDIO.

  - controlador USB 2.0 para actuar como host o cliente

  - controlador Ethernet 100 Mbit/s

  - interfaz paralela de 8 a 14 bits con cámara, hasta 54 Mbytes/s

- una RAM externa de 8 MiBytes

- otro microcontrolador de 32 bits para implantar un depurador hardware en tiempo real con interfaz ST-LINK con conexión USB para descarga y depuración de aplicaciones, puerto serie virtual y almacenamiento de sistema de archivos

- pantalla LCD TFT táctil resistiva de 2.4" resolución QVGA de 240x320 puntos con controlador ILI9341

- giróscopo 3D I3G4250D

- alimentación a través de la conexión USB de la interfaz ST-LINK o mediante fuente externa de 5V

Sistema de desarrollo STM32CubeIDE:

- Programación de aplicaciones para la familia de microcontroladores STM32 de la firma STMicroelectronics con núcleo Cortex ARM.

- Sistema de desarrollo gratuito disponible para Windows, Linux y Mac OS.

- Programación en lenguajes C y C++.

- IDE (*Integrated Development Environment*) basado en Eclipse CDT (*C/C++ Development Tooling*).

- Interfaz gráfica para configuración inicial del microcontrolador y generación automática de código.

- Herramientas para análisis de memoria, periféricos y CPU durante la ejecución de aplicaciones.

Paquete de herramientas STM32CubeF4 para la programación de aplicaciones en microcontroladores de 32 bits de la familia STM32F4. Contiene:

- STM32F4 HAL (*Hardware Abstraction Layer*) con una API (*Application Programming Interface*) en lenguaje C en forma de macros, variables, tipos de dato, funciones, etc. comunes para múltiples familias de microcontroladores para una mayor portabilidad del código.

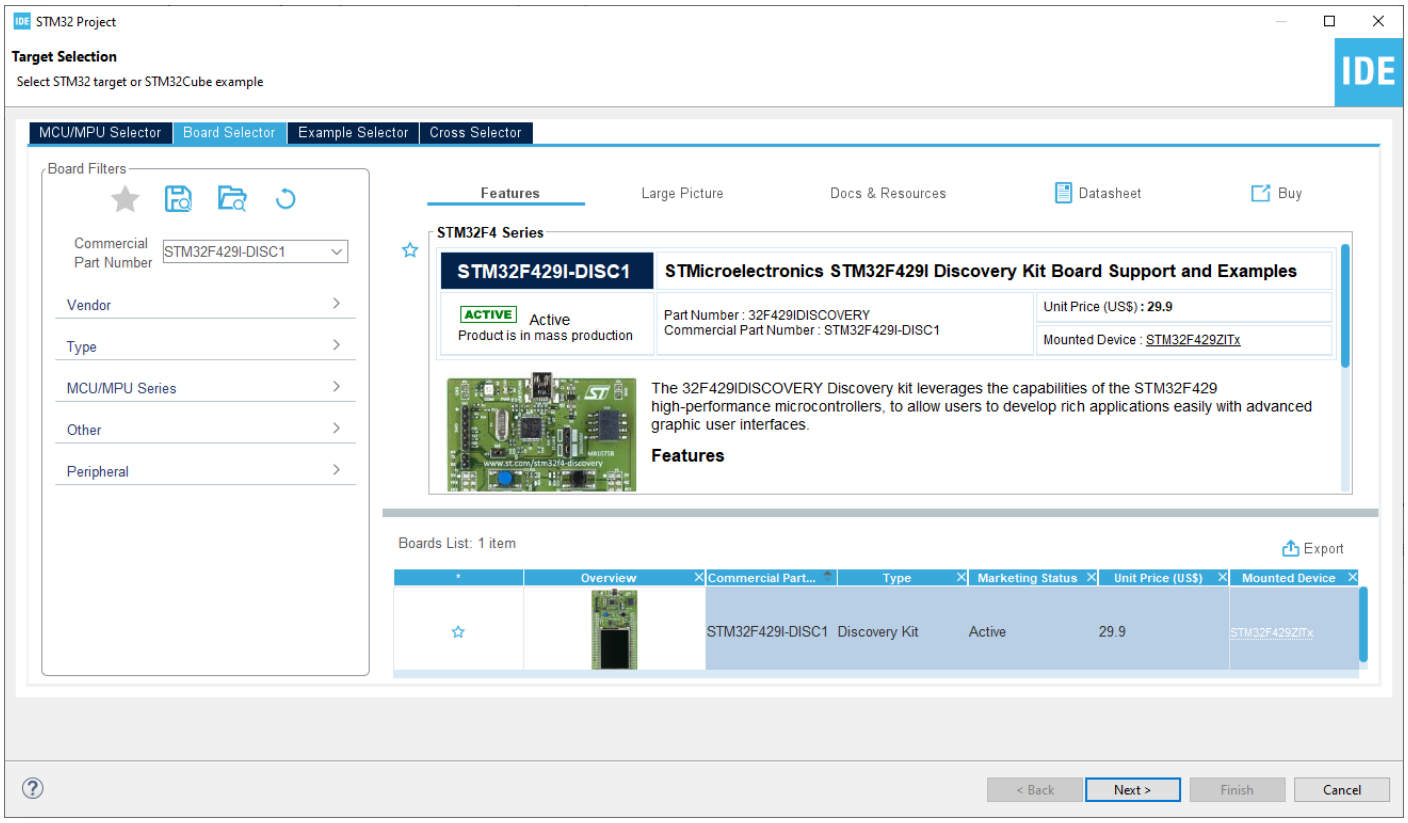
- STM32F4 LL (*Low Layer*) API para la programación a bajo nivel del manejo de los periféricos de cada modelo de microcontrolador, con un código más optimizado pero requiriendo un conocimiento más profundo del hardware.

- Un conjunto de bibliotecas de funciones a más alto nivel para manejo de comunicaciones USB, TCP/IP, etc y para poder disponer de un RTOS (*Real-Time Operating System*).

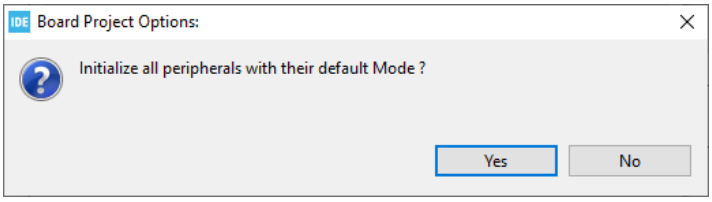
- Documentación: STM32F4 HAL and low-layer drivers

Creación de un nuevo proyecto en ST32CubeIDE: en menú *File - New - STM32 Project.*

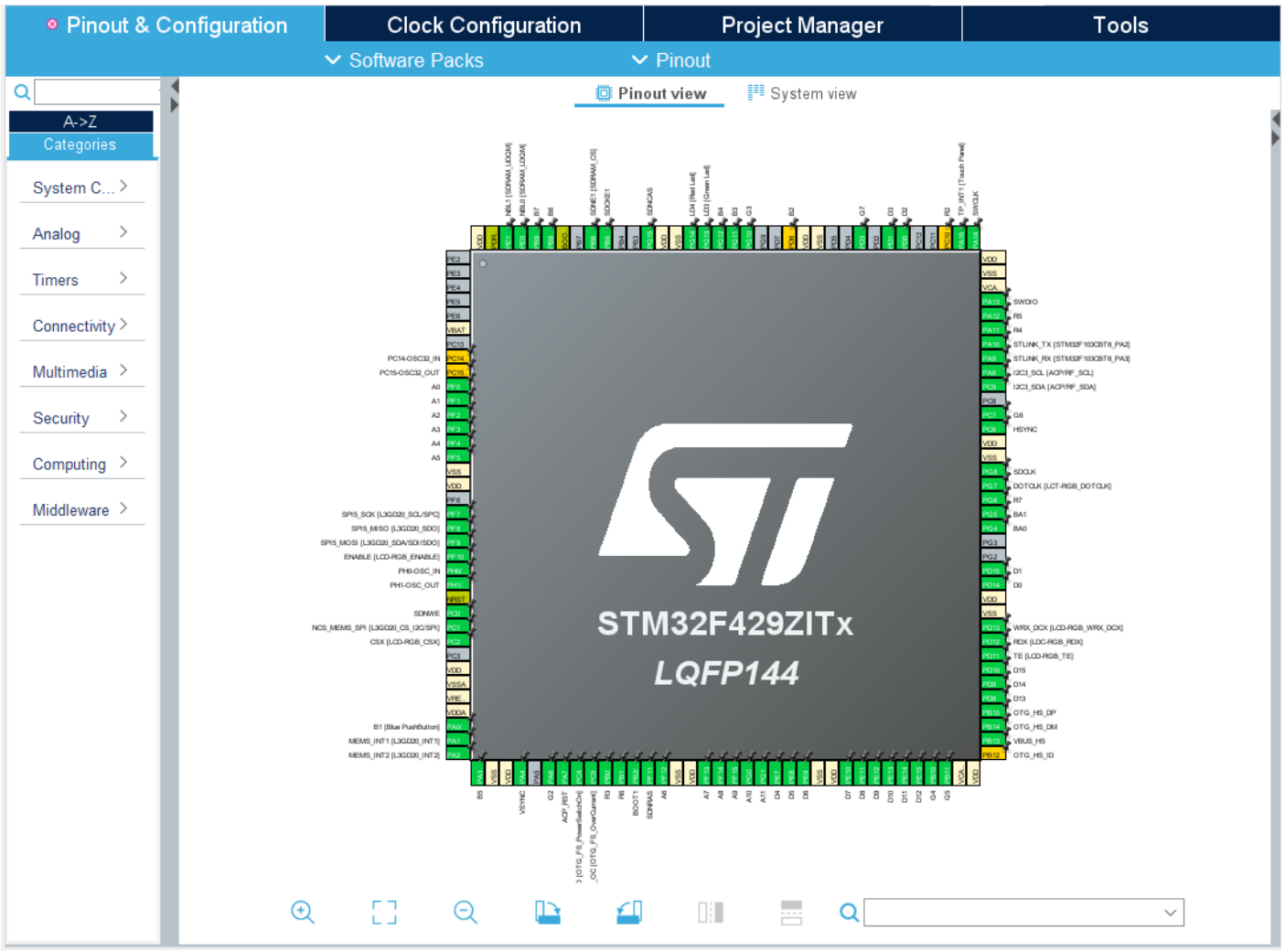
En la ventana *Target Selection* hay que indicar para qué microcontrolador o placa se va a preparar la aplicación. En la pestaña *Board Selector* indicaremos STM32F429I-DISC1 en la casilla *Commercial Part Number* y seleccionaremos esta placa abajo a la derecha.



Pulsando en *Next* podemos indicar el nombre del proyecto, el directorio donde se va a almacenar y el lenguaje de programación utilizado. En la siguiente ventana pulsaremos *Yes* para que se genere automáticamente código de inicialización de todos los periféricos disponibles en la placa.

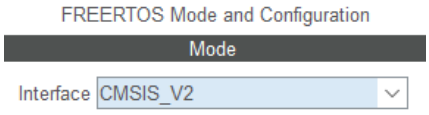


Haciendo doble click sobre el archivo con extensión `.ioc` existente en el proyecto, se abre una herramienta gráfica de configuración del microcontrolador que permite elegir en qué modo va a trabajar cada pin del micro y cada dispositivo interno. Según la configuración elegida, esta herramienta genera automáticamente código en lenguaje C en varios archivos del proyecto que programa la inicialización del micro.

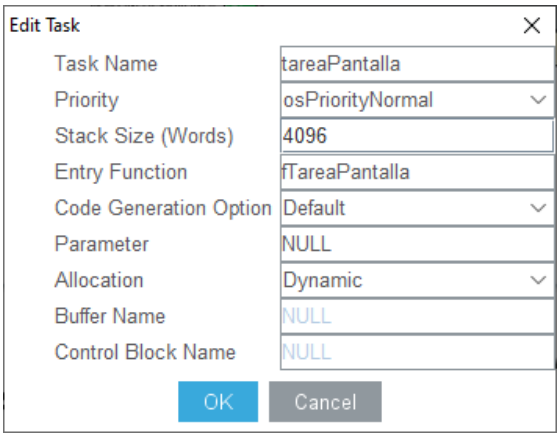


Yendo en el panel de la izquierda por *Middleware* - *FREERTOS*, entramos en la configuración de este sistema operativo.

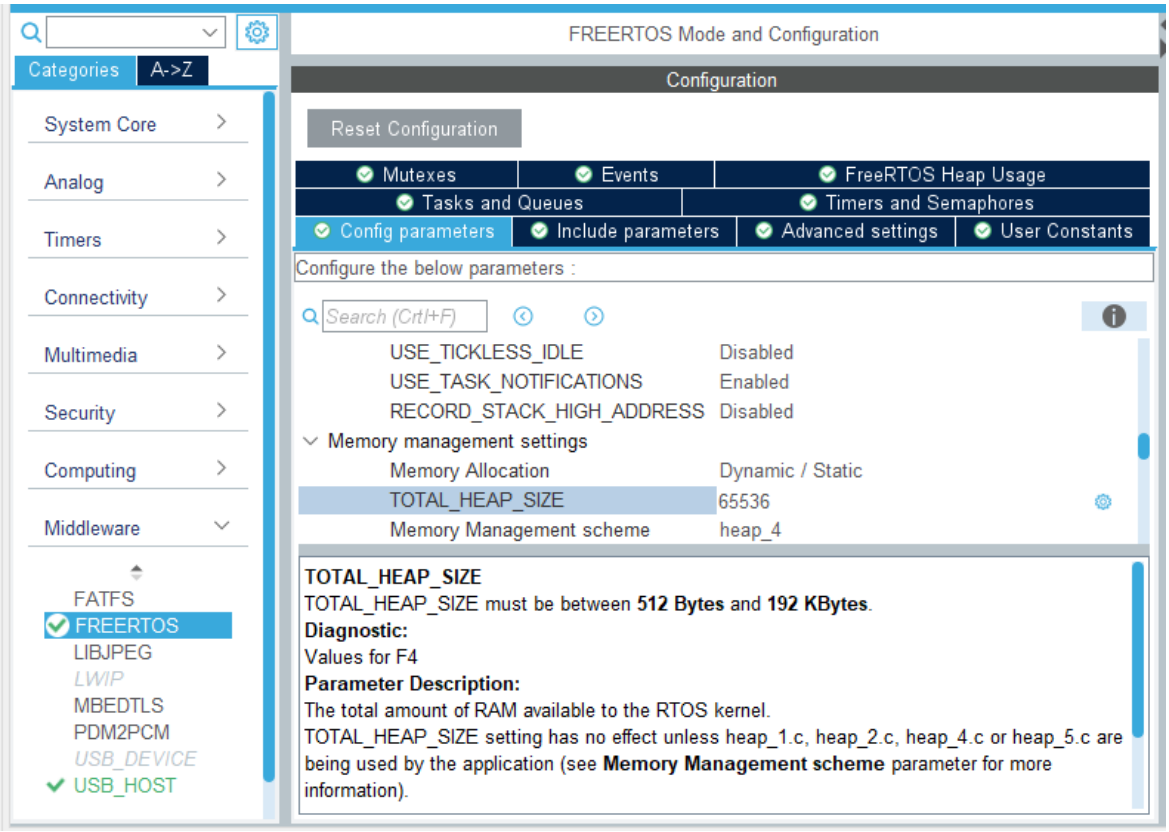
En Mode - Interface eligiremos la opción CMSIS_V2 para poder utilizar la versión 2 del estándar de programación CMSIS.



En la pestaña *Tasks and Queues* podemos indicar la creación de una nueva tarea ó hilo de ejecución. Pulsando el botón *Add*, indicamos que:
- el identificador utilizado en el programa para referirnos a esta tarea, por ejemplo **`tareaPantalla`**
- se ejecuta con una prioridad normal
- se le asigna una pila de 4096 words (cada word corresponde a 32 bits)
- esta tarea ejecuta el código de la función **`fTareaPantalla()`**
- seguidamente se indican opciones para la generación automática de código, entre las que se indica que se crea una estructura de datos para el manejo de esta tarera, que se reserva dinámicamente.



En la pestaña *Config parameters* es necesario en determinados proyectos aumentar la cantidad de memoria asignada a este sistema operativo para reservar bloques de memoria dinámica para el manejo de ciertas estructuras de datos. Para ello se puede aumentar el valor por defecto de 32 KBytes (32768 bytes) hasta 64 KBytes (65536 bytes), indicáncola en el parámetro **`TOTAL_HEAP_SIZE`**.



Para generar automáticamente el código de inicialización del microcontrolador y del sistema operativo hay que pulsar el icono y para compilar la aplicación hay que pulsar en el icono en forma de martillo de la barra de iconos superior o bien hay que elegir la opción de menú *Project - Build project*.

Se genera el archivo **`Core/Startup/startup_stm32f429zitx.s`** escrito en el lenguaje ensamblador del micocontrolador, donde se realizan llamadas a funciones de inicialización del micro y que llama a la función **`main()`** de nuestro programa:

```
/* Call the clock system intitialization function.*/
  bl  SystemInit
/* Call static constructors */
  bl __libc_init_array
/* Call the application's entry point.*/
  bl  main
```

En el directorio **`Drivers/STM32F4xx_HAL_Driver`** se encuentra el código de la capa HAL (*Hardware Abstraction Layer*) para microcontroladores de la familia STM32F4. Define estructuras de datos y funciones comunes para varios modelos de microcontroladores de la firma ST Microelectronics.

En el directorio **`Middlewares/ST/STM32_USB_Host_Library`** y en el directorio **`USB_HOST`** se encuentra código para implantar una interfaz USB Host con dispositivos externos.

En el directorio **`Middlewares/Third_Party/FreeRTOS`** se encuentra el código del sistema operativo en tiempo real FreeRTOS, utilizado para crear varias tareas ó hilos.

En **`Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS_V2`** se encuentran los recursos de programación que implantan el estándar CMSIS (*Common Microcontroller Software Interface Standard*) para microcontroladores con núcleo ARM. Utilizando este estándar, múltiples modelos de microcontroladores de diferentes fabricantes utilizando diferentes sistemas operativos se pueden programar de la misma forma, aumentando la portabilidad de las aplicaciones.

En el directorio **`Drivers/CMSIS`** se definen macros para la utilización de periféricos del microcontrolador según el estándar CMSIS.

En **`Core/Inc/FreeRTOSConfig.h`** se definen macros con ciertos valores para activar o desactivar determinadas funcionalidades del sistema operativo FreeRTOS y para asignar recursos para ese sistema.

En **`Core/Src/freertos.c`** existen funciones que se ejecutan cuando surge algún problema en la ejecución del sistema operativo FreeRTOS y donde el programador puede añadir código en zonas delimitadas con macros, por ejemplo:

```
/* USER CODE BEGIN 4 */
__weak void vApplicationStackOverflowHook(xTaskHandle xTask, signed char *pcTaskName)
{
   /* Run time stack overflow checking is performed if
   configCHECK_FOR_STACK_OVERFLOW is defined to 1 or 2. This hook function is
   called if a stack overflow is detected. */
}
/* USER CODE END 4 */
```

Se generan automáticamente los archivos **`Core/Inc/main.h`** y **`Core/Src/main.c`** para el programa principal donde aparecen secciones enmarcadas entre comentarios como

```
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
```

para que el programador añada código sólamente en esas secciones y con el cometido indicado en los comentarios. Cuando se realiza alguna modificación en la configuración del proyecto, estos archivos se generan de nuevo de forma automática y en ese proceso sólo se mantiene el código incluido entre esos comentarios.

En el directorio **`Core`** el programador puede añadir archivos de declaraciones con extensión **`.h`** en **`Core/Inc`** y archivos fuente con extensión **`.c`** en **`Core/Src`**.

Junto a este enunciado se proporciona un archivo **`ficheros.zip`** con los siguientes archivos que hay añadir al proyecto:

- **`io.h`**, **`ts.h`** y **`lcd.h`**: definición de tipos de datos y enumerados para E/S digital y manejo de la pantalla y su interfaz táctil.

- **`stm32f429i_discovery.h`** y **`stm32f429i_discovery.c`**: macros, enumerados y funciones para de varios recursos en la placa: LEDs, pulsadores, EEPROM. También macros para indicar en qué pines del microcontrolador se encuentran las líneas de canales de comunicaciones I2C, SPI, con la pantalla LCD y con el giróscopo.

- **stm32f429i_discovery_io.h** y **stm32f429i_discovery_io.c**: tipos de datos, enumerados y funciones para E/S digital.

- **stm32f429i_discovery_lcd.h** y **stm32f429i_discovery_lcd.c**: tipos de datos, enumerados y funciones para manejo de la pantalla.

- **stm32f429i_discovery_ts.h** y **stm32f429i_discovery_ts.c**: tipos de datos, enumerados y funciones para manejo de la interfaz táctil resistiva de la pantalla.

- **stm32f429i_discovery_sdram.h** y **stm32f429i_discovery_sdram.c**: tipos de datos, enumerados y funciones para manejo de un chip de memoria RAM estática externo presente en la placa.

- **ili9341.h** e **ili9341.c**: para la comunicación con el chip ILI9341 que controla la pantalla de cristal líquido presente en la placa.

- **stmpe811.h** y **stmpe811.c**: recursos para la comunicación con el chip ST STMPE811 que es el controlador de la interfaz táctil de la pantalla.

- **juegoCaractere8x11.h, juegoCaracteres8x11.c, juegoCaracteres11x16.h** y **juegoCaracteres11x16.c**: definición de dos juegos de caracteres para mostrar texto en la pantalla. En el primero se utilizan 8x11 puntos por cada carácter y en el segundo se utilizan 11x16 puntos. Para cada juego se define una matriz de bytes para indicar cómo se visualizan los caracteres de la tabla ASCII desde el código 32 hasta el 126. En el juego de 8x11 puntos cada carácter se representa con 11 bytes. Cada byte representa a una fila de puntos, total 8 puntos representados en 8 bits. Por ejemplo, el carácter 'A' se representa con los bytes:

```
0x3C, // . . @ @ @ @ . .
0x66, // . @ @ . . @ @ .
0xC3, // @ @ . . . . @ @
0xC3, // @ @ . . . . @ @
0xC3, // @ @ . . . . @ @
0xFF, // @ @ @ @ @ @ @ @
0xC3, // @ @ . . . . @ @
0xC3, // @ @ . . . . @ @
0xC3, // @ @ . . . . @ @
0xC3, // @ @ . . . . @ @
0x00, // . . . . . . . .
```

En el juego de 11x16 puntos cada carácter se representa con 32 bytes. Cada fila de 11 puntos se representa con los 8 bytes de un byte y los 3 bits más significativos del siguiente byte. Por ejemplo, para el carácter 'A':

```
0x1F, 0x00,  // . . . @ @ @ @ @ . . .
0x31, 0x80,  // . . @ @ . . . @ @ . .
0x20, 0x80,  // . . @ . . . . . @ . .
0x60, 0xC0,  // . @ @ . . . . . @ @ .
0x60, 0xC0,  // . @ @ . . . . . @ @ .
0x60, 0xC0,  // . @ @ . . . . . @ @ .
0x60, 0xC0,  // . @ @ . . . . . @ @ .
0x7F, 0xC0,  // . @ @ @ @ @ @ @ @ @ .
0x60, 0xC0,  // . @ @ . . . . . @ @ .
0x60, 0xC0,  // . @ @ . . . . . @ @ .
0x60, 0xC0,  // . @ @ . . . . . @ @ .
0x60, 0xC0,  // . @ @ . . . . . @ @ .
0x60, 0xC0,  // . @ @ . . . . . @ @ .
0x60, 0xC0,  // . @ @ . . . . . @ @ .
0x00, 0x00,  // . . . . . . . . . . .
0x00, 0x00,  // . . . . . . . . . . .
```

- **pantalla.h** y **pantalla.c**: archivos donde se dispone de funciones y estructuras de datos para la inicialización de la pantalla LCD, manejo de la interfaz táctil y dibujado de texto, líneas, círculos, rectángulos e imágenes. En estos archivos programaremos más funciones en esta práctica. A continuación se muestra en el siguiente ejemplo cómo se puede inicializar la pantalla y llamadas a estas funciones para la visualización de texto, imágenes, etc utilizando los dos frame buffers:

```
#define COLOR_FONDO_PANTALLA 0xFF000030  // Color azul para el fondo de la pantalla
#define COLOR_TEXTO 0xFFFFFF00  // Color amarillo para el texto
#define COLOR_DIBUJO 0xFFFF0000  // Color rojo para dibujar

PantallaLCD pantallaLCD;  // Estructura con datos de manejo de la pantalla

inicializaPantalla2Buffers(COLOR_FONDO_PANTALLA, &pantallaLCD);
// Inicializa los dos frame buffers de la pantalla y pone todos sus puntos al color 'COLOR_FONDO_PANTALLA'

JuegoCaracteres juegoCaracteres;
inicializaJuegoCaracteres(8, 11, 2, COLOR_TEXTO, juegoCaracteres8x11, &juegoCaracteres);
// Inicializa la estructura 'juegoCaracteres' para representar al juego de caracteres descrito
// en la matriz 'juegoCaracteres8x11' donde cada carácter ocupa 8x11 puntos, hay una separación de 2 puntos
// entre caracteres consecutivos y se visualizan en el color indicado en la macro 'COLOR_TEXTO'

dibujaImagen(0, 0, 240, 320, imagenCircuito240x320, &pantallaLCD);
// Dibuja una imagen que ocupa toda la pantalla, con esquina superior izquierda situada en (0, 0),
// de 240 puntos de ancho por 320 puntos de alto y descrita en la matriz 'imagenCircuito240x320'

dibujaCadenaCaracteres(10, 10, "Malet\'in auxilios", &juegoCaracteres, &pantallaLCD);
// Dibuja la cadena de caracteres "Maletín auxilios" en la posición (10, 10) utilizando el
// juego de caracteres definido anteriormente

dibujaImagen(10, 40, 50, 50, imagenMaletin50x50, &pantallaLCD);
// Dibuja en (10, 40) la imagen de 50 puntos de ancho y 50 de alto descrita en la matriz 'imagenMaletin50x50'

dibujaRectangulo(100, 100, 100, 50, 0xFFFFFFFF, &pantallaLCD);
// Dibuja un rectángulo con su esquina superior izquierda en (100, 100) y con 100 puntos de
// ancho y 50 puntos de alto y de color blanco

dibujaLinea(100, 100, 150, 150, COLOR_DIBUJO, &pantallaLCD);
// Dibuja una línea desde el punto (100, 100) al (150, 150) con el color indicado en 'COLOR_DIBUJO'

dibujaCirculo(100, 100, 30, COLOR_DIBUJO, &pantallaLCD);
// Dibuja círculo con centro en (100, 100), radio 20 y con un color indicado en 'COLOR_DIBUJO'

intercambiaBuffersLCD(&pantallaLCD);
// Intercambia los frame buffers para mostrar en la pantalla lo que se ha dibujado anteriormente

dibujaImagen(0, 0, 240, 320, imagenCircuito240x320, &pantallaLCD);
// Dibuja una imagen que ocupa toda la pantalla

dibujaCadenaCaracteres(10, 10, "Laboratorio", &juegoCaracteres, &pantallaLCD);
// Dibuja el texto "Laboratorio" en la posición (10, 10)

dibujaImagen(10, 40, 50, 50, imagenMicroscopio50x50, &pantallaLCD);
// Dibuja en (10, 40) la imagen de 50 puntos de ancho y 50 de alto descrita en 'imagenMicroscopio50x50'

while(1) {
    intercambiaBuffersLCD(&pantallaLCD);
    osDelay(2000);
}
// A cada 2 segundos intercambia los dos frame buffers
```

El resultado de ejecutar este código se puede comprobar en el vídeo 1 adjunto con este enunciado.

Las matrices **`imagenCircuito240x320`**, **`imagenMaletin50x50`** y **`imagenMicroscopio50x50`** contienen el color de los puntos de imágenes. Estas matrices se proporcionan en los archivos **`imagenCircuito240x320.c`**, **`imagenMaletin50x50.c`** e **`imagenMicroscopio50x50.c`**, con sus correspondientes archivos de declaraciones.

Para poder utilizar imágenes y para poder representarlas en el programa en matrices de bytes, se puede utilizar el conversor online disponible en https://lvgl.io/tools/imageconverter
En el formulario en Image file se elige el archivo de imagen que se desea convertir.
En **`Color format`** se indicará **`CF_TRUE_COLOR_ALPHA`** y en **`Output format`** se indicará **`C array`** para que se genere una matriz en lenguaje C con los colores expresados en formato ARGB en 32 bits.

El microcontrolador utiliza dos frame buffers. Cada uno de ellos corresponde a una zona de memoria de 240 * 320 * 4 = 307200 bytes = 300 KiB debido a que en cada uno de ellos se guarda el color de cada punto de la pantalla, de 240 puntos de ancho por 320 puntos de alto, y donde cada punto se representa en 4 bytes en formato ARGB.

En el formato ARGB la componente A es canal *alpha,* indicado en un byte, que expresa el grado de transparencia, desde 0=transparente hasta 255=opaco. En el frame buffer todos los puntos deberían de ser completamente opacos. Cada componente R, G y B indica con un valor entre 0 y 255 la cantidad de color rojo, verde y azul que se mezclan para establecer un color, cada componente en un byte. De esta forma, el color amarillo opaco se guarda con el valor de 32 bits 0xFFFFFF00, con componentes A=0xFF=255 (opaco), R=0xFF (a tope de rojo), G=0xFF (a tope de verde) y B=0x00 (nada de azul).

Los microcontroladores de la serie STM32F4 utilizan una organización *little-endian* para datos que ocupan varios bytes en memoria.
En el caso de los 4 bytes utilizados para expresar el color de cada punto en la pantalla, las componentes ARGB se guardan en memoria de forma que el primer byte es la componente B, a continuación la componente G, luego la componente R y finalmente el valor de transparencia Alpha.
De todas formas, cuando se expresa un valor de color en el programa en hexadecimal, por ejemplo 0xFF390DE4, las componentes son A=0xFF, R=0x39, G=0x0D y B=0xE4, luego en memoria se guarda en *little-endian*.

Las funciones a programar en esta práctica tiene que permitir:

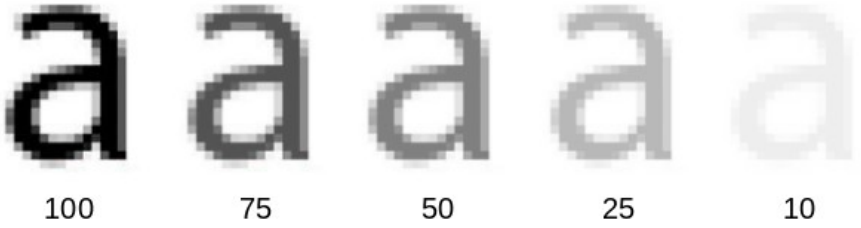
- inicializar la pantalla de forma que se le pueda asignar como fondo un color sólido o una imagen.

- dibujar una imagen donde puede haber puntos con canal Alpha diferente de 0xFF, de forma que puede disponer de zonas transparentes o semitransparentes que se funden con el fondo. La imagen tiene que poder dibujarse sobre lo que existe en el frame buffer o sobre lo que se haya definido como fondo en la pantalla.

- manejar juegos de caracteres que se representan mediante imágenes y donde se utilizan técnicas de antialiasing para poder suavizar su dibujado. Se proporcionan los archivos **`Consolas9x18Amarillo.h`** y **`Consolas9x18Amarillo.c`** donde se incluyen imágenes de los caracteres de la tabla ASCII, desde el código 32 al 126, utilizando un juego de caracteres Consolas de 9x18 puntos dibujado en amarillo. También se proporcionan archivos **`SansSerif8x13Amarillo.h`** y **`SansSerif8x13Amarillo.c`** para un juego de caracteres SansSerif de tamaño 8x13 puntos. En las imágenes de los caracteres se utilizan puntos semitransparentes en los bordes para suavizarlos.

  Se definirá una función para dibujar un carácter y otra para dibujar una cadena de caracteres. En el dibujado se utilizará el canal Alpha de los puntos de los caracteres para fundir el carácter con el fondo, que podrá ser lo que exista en el frame buffer o bien el fondo definido para la pantalla.

- En cualquier operación de dibujo (de puntos, líneas, círculos, rectángulos, caracteres o imágenes) se podrá indicar si se está dibujando sobre el mismo frame buffer donde aparecerá el resultado o si se toma como fondo el definido para la pantalla (un color sólido o una imagen de fondo). También se ha de poder aplicar un cierto nivel de transparencia desde el 0=transparente a 100=colores originales, para indicar cómo se mezcla lo que se dibuja con el fondo donde se dibuja. Y también si se dibuja la imagen convirtiéndola a blanco y negro, utilizando niveles de gris.



En el vídeo 2 adjunto a esta práctica se puede comprobar el resultado de ejecutar el siguiente código:

```
/* USER CODE BEGIN 4 */

uint32_t contador = 0;  // Contador a mostrar en la pantalla
uint8_t numResets = 0;  // Número de resets realizados sobre el contador
Boton botonReset;  // Estructura donde se guarda información para el manejo del botón

void funcionBotonReset() {  // Función a ejecutar cuando se pulse el botón
    contador = 0;  // Cuando se pulsa el botón, se resetea el contador
    numResets ++;  // Se contabilizan el número de resets
    if (numResets == 2)  // Si se reseteó 2 veces ...
        setHabilitacionBoton(0, &botonReset);  // Deshabilita el botón
}

/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartDefaultTask */
/**
  * @brief  Function implementing the defaultTask thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void *argument) {
  /* init code for USB_HOST */
  MX_USB_HOST_Init();
  /* USER CODE BEGIN 5 */

    inicializaPantalla2Buffers(0, COLOR_FONDO_PANTALLA, industria3_240x320, &pantallaLCD);
    // Inicializa la pantalla para utilizar como fondo la imagen descrita en la matriz 'industria3_240x320'

    JuegoCaracteres juego8x11, juego11x16;
    inicializaJuegoCaracteres(8, 11, 2, 0xFFFFFFFF, juegoCaracteres8x11, &juego8x11);
    inicializaJuegoCaracteres(11, 16, 1, 0xFFFFFFFF, juegoCaracteres11x16, &juego11x16);
    // Inicializa dos juegos de caracteres para dibujar en blanco:
    // - caracteres de 8x11 puntos descritos en la matriz 'juegoCaracteres8x11', con 2 puntos de
    //   separación entre caracteres consecutivos
    // - caracteres de 11x16 puntos descritos en la matriz 'juegoCaracteres11x16', con 1 punto de
    //   separación entre caracteres consecutivos

    JuegoCaracteresAlpha juegoSansSerif8x13, juegoConsolas9x18;
    inicializaJuegoCaracteresAlpha(8, 13, 0, SansSerif8x13Amarillo, &juegoSansSerif8x13);
    inicializaJuegoCaracteresAlpha(9, 18, 0, Consolas9x18Amarillo, &juegoConsolas9x18);
    // Inicializa dos juegos de caracteres creados a partir de imágenes:
    // - caracteres de 8x13 puntos descritos en la matriz 'SansSerif8x13Amarillo', sin separación
    //   entre caracteres consecutivos
    // - caracteres de 9x18 puntos descritos en la matriz 'Consolas9x18Amarillo', sin separación
    //   entre caracteres consecutivos

    inicializaBoton(170, 50, 60, 60, botonReset60x60, funcionBotonReset, 1, 1, &botonReset, &pantallaLCD);
    // Inicializa un botón con los siguientes parámetros:
    // '170, 50' son las coordenadas de su posición en pantalla.
    // '60, 60' son su ancho y alto en puntos.
    // 'botonReset60x60' es la matriz de uint8_t que describe la imagen que representa al botón.
    // 'funcionBotonReset' es una función sin parámetros y que no devuelve nada que se ejecutará
    // cuando se pulse el botón.
    // A continuación se pasan dos buleanos (en esta llamada dos unos) para indicar con el primero
    // si el botón va a ser visible en pantalla y el segundo para indicar si está habilitado (si
    // no está habilitado se mostrará semitransparente y en niveles de gris).
    // '&botonReset' es la dirección de una estructura donde se guarda la información necesaria
    // para gestionar el botón.
    // '&pantalla' es la dirección de la estructura de tipo PantallaLCD que contiene información
    // sobre la pantalla.

    int transparencia =  0;  // Para modificar la transparencia de una imagen
    int incremento = 10;  // La transparencia se va a modificar según este incremento
    char cadena[100];  // Cadena auxiliar para montar mensajes a mostrar en la pantalla
```

```
    while(1) {  // Repite contínuamente ...

        contador++;  // Incrementa un contador

        sprintf(cadena, "Contador: %-7ld", contador);
        // Añade el valor del contador a una cadena preparándolo en 7 caracteres alineado a la izquierda

        dibujaCadenaCaracteres(10, 10, cadena, &juego11x16, 0, 100, 1, &pantallaLCD);
        dibujaCadenaCaracteres(10, 30, cadena, &juego8x11, 0, 100, 1, &pantallaLCD);
        dibujaCadenaCaracteresAlpha(10, 50, cadena, &juegoConsolas9x18, 0, 100, 1, &pantallaLCD);
        dibujaCadenaCaracteresAlpha(10, 70, cadena, &juegoSansSerif8x13, 0, 100, 1, &pantallaLCD);
        // Dibuja la cadena utilizando para ello varios juegos de caracteres. A estas funciones se
        // pasa por parámetro, por orden:
        // - coordenada x donde se visualiza la cadena
        // - coordenada y donde se visualiza la cadena
        // - la cadena de caracteres
        // - la dirección de la estructura donde se describe el juego de caracteres utilizado
        // - un buleano que, si es cierto, la cadena se dibuja en niveles de gris
        // - un valor de 0 a 100 para indicar el grado de transparencia con el que se visualiza la cadena
        // - un buleano que si es cierto indica que la dibuja sobre el fondo de la pantalla, si es falso
        //   indica que se dibuja sobre lo que ya había en el frame buffer
        // - la dirección de la estructura donde hay información sobre la pantalla

        if (transparencia + incremento > 100 || transparencia + incremento < 0)
            incremento = - incremento;
        transparencia += incremento;
        // Actualiza el nivel de transparencia de 0 a 100, creciendo o decreciendo de 10 en 10

        dibujaImagen(140, 200, 70, 70, imagenAlerta1_70x70, 0, transparencia, 1, &pantallaLCD);
        // Dibuja en la posición (140, 200) una imagen de 70x70 puntos descrita en 'imagenAlerta1_70x70'.
        // En el sexto parámetro se puede indicar si se quiere dibujar la imagen en niveles de gris,
        // en este caso se está pasando un buleano falso, por lo que se dibujará con sus colores originales.
        // A continuación se indica un posible grado de transparencia (0=transparente hasta 100=opaca)
        // de la imagen.
        // Luego un buleano que si es cierto, hace que la imagen se dibuje sobre el fondo definido para
        // la pantalla, y si es falso, se deibuje sobre lo que tenga el frame buffer.

        intercambiaBuffersLCD(&pantallaLCD);
        // Intercambia los frame buffers para actualizar lo que se visualiza en la pantalla

        uint16_t xClick, yClick;
        int pulsada = pantallaPulsada(&pantallaLCD, &xClick, &yClick);
        // 'pulsada' es un buleano cierto si se está pulsando sobre la pantalla. Si es así, en 'xClick',
        // 'yClick' están las coordenadas del punto donde se está pulsando.

        atiendeBoton(xClick, yClick, pulsada, &botonReset);
        // Atiende a una posible pulsación sobre el botón indicado en 'botonReset'. Si se pulsó el botón,
        // se ejecuta la función que se asoció al mismo.

        osDelay(5);  // Retardo de 5 ms
    }

  /* USER CODE END 5 */
}
```