

Universidad de Alcalá

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA  
INDUSTRIAL



**TRABAJO FIN DE GRADO**

ESTUDIO Y DESARROLLO DE UN SISTEMA OPERATIVO PARA  
MICROCONTROLADOR BASADO EN CORTEX-M3

**Autor:** Vladislav Kravchenko

**Tutor/es:** Eliseo García García

ESCUELA POLITÉCNICA  
SUPERIOR

2019



UNIVERSIDAD DE ALCALA

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

**TRABAJO DE FIN DE GRADO**

ESTUDIO Y DESARROLLO DE UN SISTEMA OPERATIVO PARA  
MICROCONTROLADOR BASADO EN CORTEX-M3

**Autor:** Vladislav Kravchenko

**Tutor:** Eliseo García García

**TRIBUNAL:**

**Presidente:** José Raúl Durán Díaz

**Vocal 1º:** Juan Ignacio Pérez Sanz

**Vocal 2º:** Eliseo García García

**FECHA:**



# Índice

<b>RESUMEN .....</b>	<b>VIII</b>
<b>ABSTRACT.....</b>	<b>X</b>
<b>LISTA DE ABREVIATURAS Y ACRÓNIMOS.....</b>	<b>XII</b>
<b>LISTA DE FIGURAS .....</b>	<b>XIV</b>
<b>INTRODUCCIÓN.....</b>	<b>1</b>
<b>BASE TEÓRICA .....</b>	<b>3</b>
<b>1. UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER.....</b>	<b>6</b>
1.1. INTRODUCCIÓN .....	6
1.2. FUNCIÓN SVC – CONFIGURACIÓN UART.....	7
1.3. FUNCIÓN SVC – ENVIAR POR LA UART .....	8
1.4. FUNCIÓN SVC – RECIBIR POR LA UART .....	9
1.5. FUNCIÓN SVC – CLOSE.....	10
<b>2. CONEXIÓN ETHERNET .....</b>	<b>11</b>
2.1. INTRODUCCIÓN A LA COMUNICACIÓN VÍA ETHERNET .....	11
2.2. INTRODUCCIÓN COMUNICACIÓN ETHERNET EN S.O. RTX.....	14
2.2.1. Comunicación Ethernet: Servidor – Cliente .....	18
2.2.2. Comunicación Ethernet: Cliente – Servidor .....	19
2.2.3. Comunicación Ethernet – dos microcontroladores. ....	20
<b>3. PANTALLA LCD .....</b>	<b>22</b>
3.1. FUNCIÓN SVC – CONFIGURACIÓN PANTALLA LCD .....	24
3.2. FUNCIÓN SVC – REPRESENTACIÓN EN LA PANTALLA LCD .....	26
3.3. FUNCIÓN SVC – LIBERAR MEMORIA.....	28
<b>4. PANTALLA TÁCTIL – TOUCH SCREEN .....</b>	<b>29</b>
4.1. FUNCIÓN SVC – OPEN DE LA PANTALLA TÁCTIL .....	29
4.2. FUNCIÓN SVC – WRITE DE LA PANTALLA TÁCTIL.....	30
4.3. FUNCIÓN SVC – CLOSE.....	30
<b>5. FICHEROS .....</b>	<b>31</b>
5.1. INTRODUCCIÓN .....	31
5.2. DESARROLLO DE LAS FUNCIONES SVC .....	33
5.2.1. Función SVC – Open.....	35
5.2.2. Función SVC – Write.....	36
5.2.3. Función SVC – Read.....	37
5.2.4. Función SVC – Close.....	38
<b>6. CONTROL DE ACCESO.....</b>	<b>40</b>

<b>7. FUTURAS LÍNEAS DE INVESTIGACIÓN.....</b>	<b>43</b>
<b>CONCLUSIÓN.....</b>	<b>44</b>
<b>ANEXOS.....</b>	<b>45</b>
ANEXO 1. CÓDIGO APLICACIÓN SERVIDOR EN SO WINDOWS.....	45
ANEXO 2. CÓDIGO APLICACIÓN CLIENTE EN SO WINDOWS.....	46
ANEXO 3. CODIFICACIÓN MANCHESTER.....	47
ANEXO 4. LPC1768.....	48
ANEXO 5. FUNCIONES UTILIZADAS EN EL DESARROLLO DEL PROYECTO .....	49
<b>BIBLIOGRAFÍA.....</b>	<b>59</b>



## Resumen

Este proyecto pretende crear un sistema operativo para microcontroladores basados en la arquitectura Cortex-M3 que proporcione tanto al usuario, como al mismo hardware, de unas garantías de funcionamiento que mantenga la seguridad y una óptima eficiencia a la hora de gestionar los recursos del microcontrolador.

Para ello, se parte de la base de un sistema operativo ya existente para estos microcontroladores como es **RTX**, un RTOS(Real-Time Operating System) desarrollado para dispositivos ARM y Cortex – M.

El objetivo será la programación de las características de la placa con la que trabajaremos, dentro de este sistema operativo; dotándole de la capacidad de manejo de los recursos dentro de las aplicaciones con las que el usuario vaya a trabajar.

### **Palabras clave:**

Sistema operativo

Microcontrolador

Recursos

Seguridad

Funcionalidad





## Abstract

This project aims to create an operating system for microcontrollers based on the Cortex-M3 architecture that provides the user, as well as the hardware itself, with operational guarantees that maintain security and optimum efficiency when managing microcontroller resources.

To do this, we start from the base of an existing operating system for these microcontrollers, such as **RTX**, an RTOS (Real – Time Operating System) developed for ARM and Cortex – M devices.

The objective will be the programming of the characteristics of the board with which we will work, within this operating system; giving it the ability to manage resources within the applications with which the user goes to work.

### **Keywords:**

Operative System

Microcontroller

Resources

Security

Functionality



## Lista de abreviaturas y acrónimos

<b>Abreviatura</b>	<b>Término</b>
<b>SO</b>	<i>Sistema Operativo</i>
<b>RTOS</b>	<i>Real-Time Operating System</i>
<b>USB</b>	<i>Universal Serial Bus</i>
<b>ADC</b>	<i>Analog-to-Digital Conversion</i>
<b>DAC</b>	<i>Digital-to-Analog Conversion</i>
<b>I2C</b>	<i>Inter-Integrated Circuit</i>
<b>UART</b>	<i>Universal Asynchronous Receiver-Transmitter</i>
<b>RS232</b>	<i>Recommended Standard 232</i>
<b>I2S</b>	<i>Integrated Interchip Sound</i>
<b>SVC</b>	<i>Supervisor Call Instruction</i>
<b>THR</b>	<i>Transmit Holding Register</i>
<b>TSR</b>	<i>Transmit Shift Register</i>
<b>RSR</b>	<i>Receive Shift Register</i>
<b>RBR</b>	<i>Receive Holding Register</i>
<b>LCR</b>	<i>Line Control Register</i>
<b>FDR</b>	<i>Fractional Divider Register</i>
<b>IER</b>	<i>Interrupt Enable Register</i>
<b>LAN</b>	<i>Local Area Network</i>
<b>UDP</b>	<i>User Datagram Protocol</i>
<b>TCP</b>	<i>Transport Control Protocol</i>
<b>IP</b>	<i>Internet Protocol</i>
<b>MAC</b>	<i>Media Access Control</i>
<b>ISN</b>	<i>Initial Sequence Number</i>
<b>ARP</b>	<i>Address Resolution Protocol</i>
<b>LCD</b>	<i>Liquid Cristal Display</i>
<b>TFT</b>	<i>Thin Film Transistor</i>
<b>SD</b>	<i>Secure Digital</i>
<b>SPI</b>	<i>Serial Peripheral Interface</i>
<b>SSP</b>	<i>Synchronous Serial Port</i>
<b>MOSI</b>	<i>Master Output Slave Input</i>
<b>MISO</b>	<i>Master Input Slave Output</i>
<b>PWM</b>	<i>Pulse – Width Modulation</i>
<b>CPU</b>	<i>Central Processing Unit</i>
<b>CAN</b>	<i>Controller Area Network</i>
<b>DMA</b>	<i>Direct Memory Access</i>



## Lista de figuras

<b>Figura 1.</b> Jerarquía en un microcontrolador con SO .....	1
<b>Figura 2.</b> Esquema del organizador del SO RTX .....	3
<b>Figura 3.</b> Ejemplo de importación de la función __SVC_1 .....	4
<b>Figura 4.</b> Ejemplo de incorporación de una referencia a la función __SVC_1 .....	5
<b>Figura 5.</b> Esquema de comunicación a través de la UART .....	6
<b>Figura 6.</b> Registro PINSEL para configuración de pines UART2.....	7
<b>Figura 7.</b> Captura del programa Termite.exe con el contenido de un mensaje enviado por el microcontrolador.....	9
<b>Figura 8.</b> Programa Termite.exe con los mensajes enviados y recibidos por el microcontrolador. ....	10
<b>Figura 9.</b> Diferentes campos en un paquete de Ethernet (Fuente: Manual de usuario LPC1768).....	12
<b>Figura 10.</b> Campos existentes en un paquete Ethernet con protocolo TCP/IP.....	13
<b>Figura 11.</b> Página web del proyecto EasyWEB (Mini-DK2 for LAN8720).....	14
<b>Figura 12.</b> Tipo de estructura proporcionada por el SO.....	16
<b>Figura 13.</b> Dirección IP del microcontrolador.....	17
<b>Figura 14.</b> Dirección física del microcontrolador.....	17
<b>Figura 15.</b> Dirección IP de la puerta de enlace .....	17
<b>Figura 16.</b> Aplicación cliente en el sistema operativo Windows – estableciendo conexión con LPC1768 .....	18
<b>Figura 17.</b> Array que contiene la información que le llega al microcontrolador a través de la comunicación Ethernet .....	19
<b>Figura 18.</b> Aplicación servidor en el sistema operativo Windows – estableciendo conexión con LPC1768.....	19
<b>Figura 19.</b> Conexión comunicación Ethernet entre dos microcontroladores .....	20
<b>Figura 20.</b> Mensaje recibido en el microcontrolador servidor .....	21
<b>Figura 21.</b> Mensaje recibido en el microcontrolador cliente.....	21
<b>Figura 22.</b> Colores definidos en el proyecto ILI9325 .....	23
<b>Figura 23.</b> Estructura con los parámetros necesarios para la representación en la pantalla LCD.....	24
<b>Figura 24.</b> Resultados obtenidos en la pantalla tras ejecutar la aplicación con SO, con select desde 0 a 8, respectivamente. ....	27
<b>Figura 25.</b> Calibración de la pantalla táctil en el microcontrolador LPC1768.....	30
<b>Figura 26.</b> Tarjeta microSD empleada en el desarrollo y la comprobación del proyecto .....	31

<b>Figura 27.</b> Asignación de pines de la tarjeta microSD en el modo SPI.....	32
<b>Figura 28.</b> Diagrama de pines de la tarjeta microSD.....	32
<b>Figura 29.</b> Diagrama de la ranura para las tarjetas SD en la placa de desarrollo Mini – DK2 .....	33
<b>Figura 30.</b> El microcontrolador espera a que se detecte una conexión de una tarjeta SD .....	35
<b>Figura 31.</b> El microcontrolador detecta la conexión de la tarjeta SD.....	35
<b>Figura 32.</b> Opciones disponibles para indicar la opción elegida a la variable “option” en la función SVC write(); .....	36
<b>Figura 33.</b> Opciones disponibles para indicar la opción elegida a la variable “option” en la función SVC read(); .....	37
<b>Figura 34.</b> Opción SHOW_SIZE de la función SVC read(); indica la memoria de la tarjeta SD y la memoria disponible .....	37
<b>Figura 35.</b> Opcion SHOW_FILES de la función SVC read(); indica los ficheros existentes en la tarjeta SD .....	38
<b>Figura 36.</b> Opciones disponibles para indicar la opción elegida a la variable “option” en la función SVC close(); .....	38
<b>Figura 37.</b> Estructura de control de acceso, ejemplo considerado del Touch Panel .....	41
<b>Figura 38.</b> Ejemplo de codificación Manchester.....	47
<b>Figura 39.</b> Placa de desarrollo Mini – DK2 con microcontrolador LPC1768 basado en Cortex – M3 .....	48

*MEMORIA*



## Introducción

La historia de los microcontroladores empieza en la década de los 70 del siglo XX, más concretamente en el año 1971 fue inventado el primer microcontrolador TMS-1000. Los primeros microcontroladores eran microprocesadores con memoria RAM y ROM.

Con el paso de los años los microcontroladores fueron desarrollándose para diferentes tipos de aplicaciones tales como automóviles, electrodomésticos, etc. Debido a esto, apareció la complejidad de crear un sistema en el cual existían diversos subsistemas que también debían ser controlados; subsistemas tales como motores, sensores, etc.

De esta complejidad en cuanto a programación de la aplicación para un sistema complejo, en el cual se debe controlar todos los subsistemas, y, la temporización adecuada de la respuesta de estos sistemas apareció la necesidad de tener un “programa específico” que nos ayudara a manejar las capacidades de nuestro hardware. Este “programa específico” es el **sistema operativo (SO)**. La jerarquía en un microcontrolador con sistema operativo se ilustra en la Figura 1.



*Figura 1. Jerarquía en un microcontrolador con SO*

El SO nos ayuda en la temporización de las tareas, disminuye la complejidad de la programación debido a que el sistema operativo nos ofrece las funciones específicas para comunicarnos con el hardware, y, además, se incrementa la portabilidad de los programas desarrollados. Los SO también nos brindan mayor seguridad a la hora del funcionamiento del sistema y evita el uso incorrecto del sistema al no interactuar el usuario directamente con el hardware.

Nuestro objetivo, en este trabajo, es el desarrollo de un sistema operativo para los microcontroladores basados en Cortex-M3 que incluya dentro de sí las funciones para el control de algunas de las características que poseen estas placas de desarrollo, para

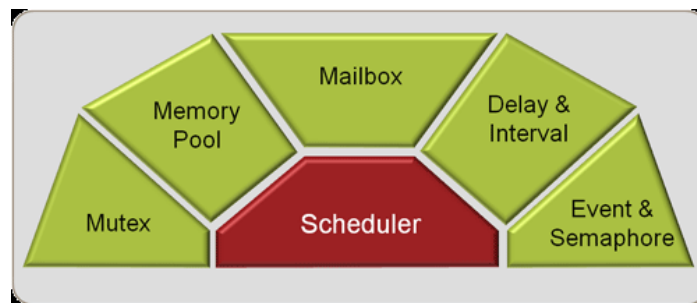
así brindar mayor seguridad a la hora de trabajar, poseer un buen manejo de recursos y una buena temporización en la ejecución de tareas.

Para llevar a cabo este proyecto partiremos de la base de un sistema operativo ya existente llamado RTX, un RTOS diseñado para dispositivos ARM y Cortex-M.

## Base teórica

El sistema operativo RTX permite realizar programas que realizan simultáneamente múltiples funciones, manejadas por el scheduler (organizador) que es quien decide que tarea se ejecuta, cuanto tiempo y cuando se va a pasar a ejecutar la siguiente tarea. Una tarea es un programa cargado en memoria desde donde es procesado por la CPU. Para tener este control, el usuario es el encargado de evaluar y asignar la prioridad de cada tarea a ejecutar. Con lo que el sistema operativo elige en función de esa prioridad cual será la siguiente tarea o proceso para ejecutar.

En la Figura 2 se presenta el esquema del scheduler del SO RTX, diseñado para permitir ejecutar tareas en tiempos predecibles.



*Figura 2. Esquema del organizador del SO RTX*

Las ventajas de utilizar un RTOS frente a la ejecución de cada función en un orden fijo son las siguientes:

- Las tareas se ejecutan cuando es necesario, para garantizar una mejor respuesta al flujo de eventos.
- Al garantizar una mejor respuesta al flujo de eventos, se da la ilusión de que se están ejecutando varias tareas a la vez.
- Los eventos e interrupciones son manejados dentro de un tiempo definido.
- Se gestiona el intercambio de datos, memoria y recursos de hardware entre las tareas.
- Se hace uso de un espacio definido de pila para cada tarea, lo que define un uso de memoria predecible.
- En cuanto a la administración de recursos, el sistema operativo nos permite enfocarnos más en el desarrollo de la aplicación.

Como habíamos dicho en la introducción, nuestro sistema operativo debe evitar que el usuario acceda directamente al hardware, y para eso haremos uso de las funciones SVC o llamadas de supervisor. Estas funciones son excepciones dirigidas al sistema operativo para generar llamadas al sistema. Cuando se llama a una función SVC se

ejecuta el controlador de excepciones de software en el sistema operativo y proporciona el servicio solicitado a la aplicación de usuario haciendo que el acceso al hardware esté bajo el control del SO.

Estas funciones SVC se utilizan de igual manera que otras funciones, pero se ejecutan en modo Privileged Handler del núcleo de Cortex-M.

Por lo tanto, partiendo del sistema operativo RTX de ARM Keil, y utilizando las funciones SVC, se propone como objetivo proceder a desarrollar este sistema operativo integrando en él las funcionalidades que poseen los microcontroladores basados en Cortex-M3, entre ellas se incluirán las siguientes:

- UART
- Ethernet 10/100Mbps
- LCD
- Pantalla táctil
- Trabajo con ficheros en una tarjeta SD

El proyecto planteado va a consistir en gran parte en la creación de las funciones llamada al sistema, por lo que se muestra imprescindible presentar los pasos que se necesitan seguir para una correcta utilización de estas funciones en el sistema operativo RTX.

- En primer lugar, se debe incluir el archivo **SVC\_Table.s** en el proyecto con el que se está trabajando.
- A continuación, se declara una función con el atributo **\_\_svc(x)** con x un numero empezando desde el 1. La función SVC 0 está reservada para el kernel RTX. También hay que tener en cuenta que no se deben dejar huecos al numerar las funciones SVC; estas deben ocupar un rango continuo de números a partir del 1.
- Ahora se deberá crear una función cuyo nombre será **\_\_SVC\_x** (con x el mismo numero utilizado para declarar la función). Este nombre posteriormente es referenciado por el linker del módulo **SVC\_Table.s**.
- Ahora que tenemos la función **\_\_SVC\_x**, se deberá añadir ésta en la tabla de funciones SVC en el archivo **SVC\_Table.s**. En primer lugar, se importa desde otros módulos (Figura 3).

```
; Import user SVC functions here.  
IMPORT __SVC_1
```

*Figura 3. Ejemplo de importación de la función \_\_SVC\_1*

Y a continuación, se debe agregar una referencia a esta función en la tabla (Figura 4).

```
; Insert user SVC functions here. SVC 0 used by RTL Kernel.  
DCD      __SVC_1                ; user SVC function
```

*Figura 4. Ejemplo de incorporación de una referencia a la función \_\_SVC\_1*

- Ahora, al haber seguido estos pasos correctamente, ya se podría utilizar la función **\_\_SVC\_1** presentada en estos ejemplos. Por lo que se podría desarrollar el código que deberá ejecutar esta función.

Una vez expuestos los objetivos de este proyecto y la base teórica de la que se parte para el desarrollo de este proyecto, se continua con la primera característica implementada dentro del sistema operativo: la UART.

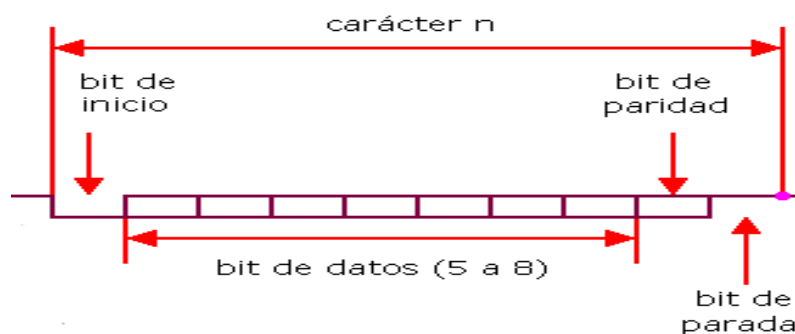
# 1. Universal Asynchronous Receiver-Transmitter

## 1.1. Introducción

El Universal Asynchronous Receiver-Transmitter abreviado en inglés como UART es un dispositivo de comunicación encargado tanto de recibir como enviar información convirtiendo los datos serie a paralelos (cuando se reciben datos) y viceversa (cuando se envían datos). La placa LPC1768 con la que trabajamos posee cuatro módulos UART (0,1,2 y 3), que son capaces de trabajar independientemente. La forma en la que se establece la comunicación serie es a través de dos líneas, Tx y Rx, una de transmisión de datos y otra de recepción. La transmisión de información se realiza a una velocidad de transmisión (o baudrate) configurable; aunque puede ser cualquier valor existen valores estándares, tales como: 110, 150, 300, 600, 900, 1200, 2400, 4800, 9600, 14400, 19200, etc. La comunicación puede ser configurada de la siguiente manera:

- Bit de inicio
- Longitud del dato 5,6,7 o 8 bits
- 1, 1.5 o 2 bits de parada
- Bit de paridad habilitado o deshabilitado

Al establecerse la comunicación e iniciar la transmisión, el propio hardware según lo configurado, enviará el bit de inicio, el bit de paridad y el bit de parada. En la recepción sucede lo mismo, el hardware recibirá el bit de inicio, que indica que se debe muestrear la línea de recepción de datos.



*Figura 5. Esquema de comunicación a través de la UART*

En LPC1768 la información es transmitida de la siguiente manera: Primero, los datos se almacenan en el registro THR bit a bit, a continuación el registro TSR lee los datos almacenados en THR y los ensambla para transmitirlos a través del pin de salida serial, Tx.

Por otro lado, en la recepción se controla la línea de entrada serie, para una entrada válida. El registro RSR acepta caracteres válidos, y una vez ensamblado un carácter este se pasa al registro RBR, donde esperará a ser leído.

Una vez realizada una mínima introducción de lo que es la UART, se explica como se ha integrado dentro del sistema operativo RTX.

## 1.2. Función SVC – configuración UART

Como primer paso se crea la función llamada al supervisor (*void \_\_svc(1)* *open\_uart(uint8\_t UARTn, uint32\_t baudrate, osThreadId ID)*) con la que se configuraran las cuatro posibles UART. A esta función se le podrá enviar como argumentos, la UART que se desea configurar, eligiendo entre cuatro diferentes macros "UART0", "UART1", "UART2" o "UART3", y, también, se le podrá enviar como argumento la velocidad de transmisión con la que se va a trabajar. Se ha definido por defecto la comunicación con longitud de dato de 8 bits, sin paridad y un bit de parada para las 4 UART.

Vamos a coger como ejemplo que se necesita definir la comunicación por la UART2 con una velocidad de transmisión de 9600 bps.

El primer paso es la configuración de los pines como RXD2 y TXD2 en el registro PINSEL0, por lo que escribe en los bits correspondientes de este registro.

**Table 80. Pin function select register 0 (PINSEL0 - address 0x4002 C000) bit description**

PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.0	GPIO Port 0.0	RD1	TXD3	SDA1	00
3:2	P0.1	GPIO Port 0.1	TD1	RXD3	SCL1	00
5:4	P0.2	GPIO Port 0.2	TXD0	AD0.7	Reserved	00
7:6	P0.3	GPIO Port 0.3	RXD0	AD0.6	Reserved	00
9:8	P0.4 <sup>[1]</sup>	GPIO Port 0.4	I2SRX_CLK	RD2	CAP2.0	00
11:10	P0.5 <sup>[1]</sup>	GPIO Port 0.5	I2SRX_WS	TD2	CAP2.1	00
13:12	P0.6	GPIO Port 0.6	I2SRX_SDA	SSEL1	MAT2.0	00
15:14	P0.7	GPIO Port 0.7	I2STX_CLK	SCK1	MAT2.1	00
17:16	P0.8	GPIO Port 0.8	I2STX_WS	MISO1	MAT2.2	00
19:18	P0.9	GPIO Port 0.9	I2STX_SDA	MOSI1	MAT2.3	00
21:20	P0.10	GPIO Port 0.10	TXD2	SDA2	MAT3.0	00
23:22	P0.11	GPIO Port 0.11	RXD2	SCL2	MAT3.1	00
29:24	-	Reserved	Reserved	Reserved	Reserved	0
31:30	P0.15	GPIO Port 0.15	TXD1	SCK0	SCK	00

*Figura 6. Registro PINSEL para configuración de pines UART2*

El siguiente paso, será la habilitación de la UART2 en el registro PCONP, por el contrario, la comunicación serie no será establecida. Esta es una de las diferencias

entre la configuración de la UART2 y la UART3 con respecto a UART0 y UART1, ya que estas últimas están habilitadas por defecto en el registro PCONP.

El establecimiento del tipo de comunicación (longitud de datos, paridad y bit de parada) se configura en el registro LCR, para ver cómo se realiza la configuración se puede consultar la tabla 280 del manual de usuario de LPC176x/5x.

Con el baudrate indicado anteriormente como argumento de la función, se continua con los cálculos para establecer los parámetros correctos. La expresión con la que se calcula el baudrate es la siguiente:

$$UART_{baudrate} = \frac{PCLK}{16 * (256 * UnDLM + UnDLL) * \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

Donde PCLK es el reloj del periférico; UnDLM y UnDLL forman parte de un mismo registro pero que se divide en dos partes, parte baja UnDLL bit [7:0] y parte alta UnDLM bit [15:8];  $1 \leq MulVal \leq 15$  y  $0 \leq DivAddVal \leq 14$  con  $DivAddVal \leq MulVal$ .

Tras calcular todos estos datos, con respecto al baudrate definido, se deben almacenar en sus respectivos registros. Los valores de UnDLM y UnDLL, se almacenan en los registros DLM y DLL, respectivamente, pero para ello, se debe habilitar el acceso para la configuración en el ya conocido registro LCR. Tras la escritura se debe deshabilitar el acceso. Las variables DivAddVal y MulVal se escriben en el registro FDR (Tabla 286 del manual de usuario).

Una vez calculados estos parámetros, lo último que se debe hacer dentro de esta función de configuración de las UART, es la habilitación de las interrupciones de las correspondientes UART con las que se trabaja. En el caso de nuestro ejemplo, la UART2. Para ello se debe escribir en el registro IER, habilitando las interrupciones de transmisión y las interrupciones de recepción.

Con todos estos pasos realizados, ya se habría creado nuestra primera función SVC en el sistema operativo RTX.

### 1.3. Función SVC – enviar por la UART

Cuando el usuario desee enviar información a través de la UART, tendrá a su disposición la función SVC `write_uart(uint8_t UARTn, char *datos, osThreadId ID)`. Como se observa, a esta función se le pasa como argumentos el número de la UART por la que se quiere enviar la información, y, la información en sí a través de `char *datos`.

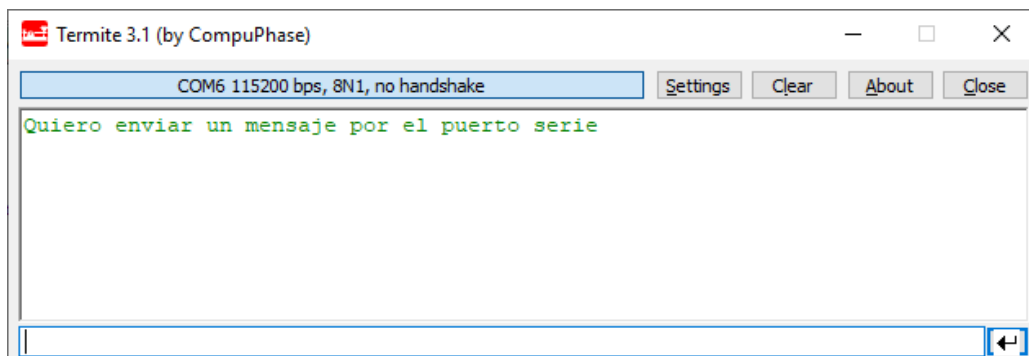


La configuración de la UART en el apartado anterior estableció que se producía una interrupción en el momento en el que se detectaba que hay información disponible para transmisión o para recepción. Según esta configuración, en cuanto el usuario transmita información, en la función se enviará el primer dato al registro THR para activar el flag de interrupción de la UART, para continuar realizando la transmisión dentro de la correspondiente función Handler (UARTx\_IRQHandler ()); con  $x = \{0,1,2,3\}$ .

En el momento en el que se haya terminado la transmisión de información, se activará un flag a '1' que nos indicará que es posible el envío de la siguiente ristra de datos. Esto se ha realizado para que no se produzca pérdida de información debido a la reescritura de datos.

Para comprobar el funcionamiento de esta función SVC, hemos utilizado un programa llamado Terminate.exe, el cual es un terminal RS232, con una ventana que contiene los datos recibidos y una línea para escribir cadenas para transmitir.

En la siguiente Figura 7, se puede observar como el microcontrolador ha enviado el mensaje *“Quiero enviar un mensaje por el puerto serie”*, y como se puede observar, el programa lo ha recibido correctamente. Por ello, podemos concluir que se ha realizado correctamente el desarrollo de esta función SVC de la UART.



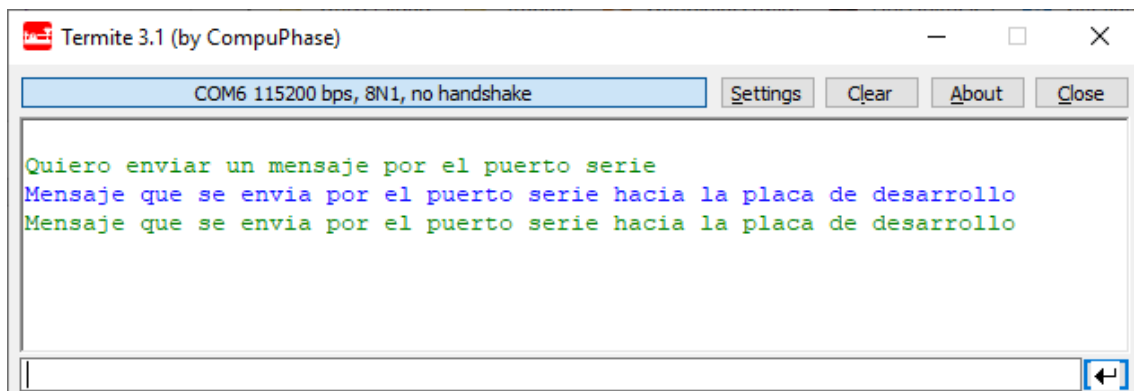
*Figura 7. Captura del programa Terminate.exe con el contenido de un mensaje enviado por el microcontrolador.*

### 1.4. Función SVC – recibir por la UART

La función llamada al supervisor para recibir los datos `void __svc(3) read_uart(uint8_t UARTn, char *datos_rx, osThreadId ID)` tiene los mismos argumentos que la función enviar, pero en este caso lo que queremos es almacenar o visualizar lo que nos llega por la UART. Como ya se ha dicho, cada vez que se detecta un dato en el registro RBR, registro de recepción, se activa la interrupción y se pasa a ejecutar la función de interrupción de la UART correspondiente; en esta función se ha escrito el código para que los datos que lleguen por RBR se indiquen a un puntero, que, tras acabar de recibir

los datos, se asigna a un array previamente inicializado para guardar la información recibida.

Para comprobar que todo funciona según se ha planeado, utilizamos el programa anteriormente mencionado, Termite.exe. En la Figura 8, podemos observar el primer mensaje que se envió en el apartado anterior con la función SVC `write_uart()` de la UART. Además, lo siguiente que hacemos es enviar un mensaje desde la línea disponible para transmisión en este programa. Los mensajes que se envían a través de esta línea se representan en color azul. El mensaje enviado es “*Mensaje que se envía por el puerto serie hacia la placa de desarrollo*”. Para comprobar la correcta recepción de este mensaje por el microcontrolador, le pedimos que reenvíe el mensaje por el puerto serie hacia el programa, y como se puede observar el mensaje en color verde, se ha recibido por el programa tal cual se ha enviado.



*Figura 8. Programa Termite.exe con los mensajes enviados y recibidos por el microcontrolador.*

Con todo esto comprobado, podemos concluir que se ha realizado satisfactoriamente la incorporación de esta función SVC en el sistema operativo, y que realiza su función correctamente.

### 1.5. Función SVC – Close

Se ha creado una función SVC `void __svc(19) close_uart(osThreadId ID)` que realiza la función de devolver la memoria reservada para la estructura de control de acceso, que se explica detalladamente en el apartado 6 de este documento. Por lo que no realiza ninguna función que esté relacionada con la UART propiamente dicha.

## 2. Conexión Ethernet

### 2.1. Introducción a la comunicación vía Ethernet

Ethernet es un estándar de comunicación en redes LAN (Local Area Network) que es utilizado para la comunicación entre dispositivos de un área no muy amplia, pudiendo ser oficinas, un campus de una universidad o en una casa particular. Para realizar esta comunicación existen varios protocolos, entre los que se encuentran TCP/IP y UDP.

El protocolo UDP no está orientado a conexión ya que el flujo de información es unidireccional, el destinatario de la información no confirma la recepción de los datos que le ha enviado el emisor. El destinatario conocerá únicamente la IP del emisor que transmite los paquetes de información.

Por el contrario, el protocolo TCP/IP, formado por el protocolo de transporte o TCP y el protocolo de internet o IP, está orientado a conexión debido a que el destinatario de la información es notificado de la llegada de ésta, y además confirma la recepción satisfactoria. También, si la recepción fuera errónea, el destinatario es capaz de solicitar un reenvío de estos datos.

El motivo de la explicación de estos dos protocolos es que Cortex-M3 puede llegar a implementar alguno de estos métodos de comunicación a través de la red Ethernet. En nuestro desarrollo del sistema operativo utilizaremos el protocolo TCP/IP debido a que estamos creando un sistema operativo fiable y TCP/IP es quien nos proporciona la seguridad de que nuestra información enviada ha sido recibida correctamente.

Cuando se envía un mensaje a través de Ethernet, los datos se ensamblan con cabeceras y colas para gestionar la transmisión y recepción del mensaje. Además, si el protocolo de comunicación es TCP/IP, como es nuestro caso, se añadirán las cabeceras del protocolo TCP y del protocolo IP, con la información pertinente.

El protocolo Ethernet está compuesto por:

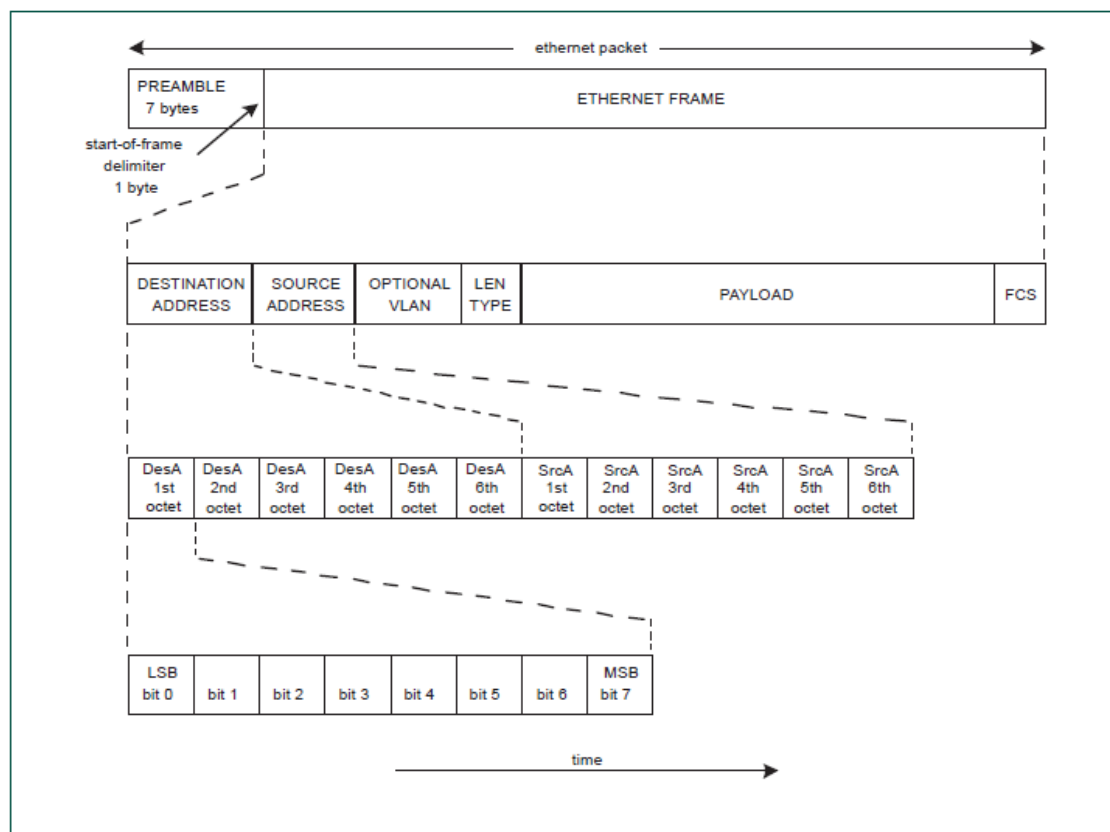
- Paquetes Ethernet.
- Una capa física (Physical Layer).
- Operación de control de acceso a los medios (*en inglés: MAC Operation*).

## 1. Paquete Ethernet

Un paquete Ethernet está compuesto por:

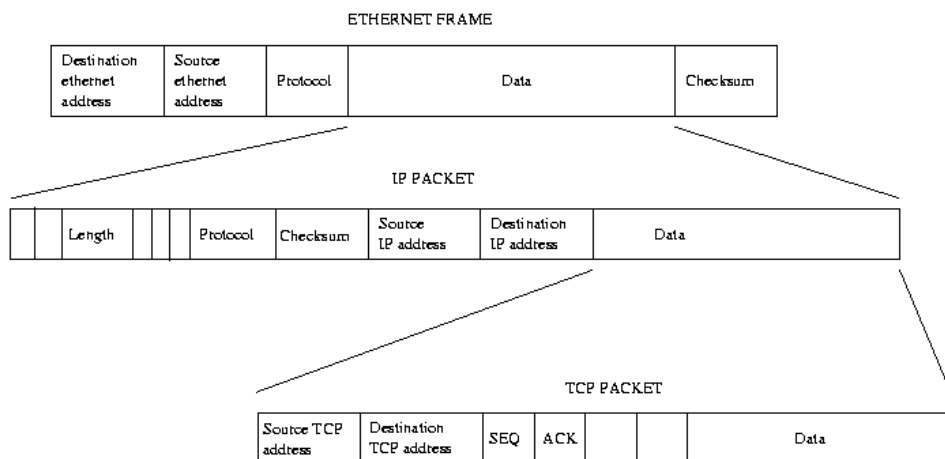
- Un preámbulo de la forma ("10101010"), un delimitador de inicio de trama y una trama de Ethernet
- La trama de Ethernet está compuesta por la dirección de destino, la dirección de origen, un campo VLAN (acrónimo de virtual LAN) opcional, el campo de longitud/tipo, el campo de los datos y por último el campo de verificación de trama.
- Cada dirección consta de 6 Bytes y cada Byte consta de 8 bits, los cuales se transmiten comenzando por el bit menos significativo.

El esquema descrito se puede observar en la Figura 9.



*Figura 9. Diferentes campos en un paquete de Ethernet (Fuente: Manual de usuario LPC1768)*

Además, como ya hemos dicho el protocolo TCP/IP añade sus propias cabeceras, resultando algo parecido a lo que se presenta en la Figura 10.



*Figura 10. Campos existentes en un paquete Ethernet con protocolo TCP/IP*

El empaquetado y desempaquetado de todos estos datos se realiza automáticamente, quedando para el destinatario únicamente la información que envió el emisor.

## 2. Physical Layer

La capa física se ocupa de la forma electrónica de bajo nivel en que se transmiten las señales. El módulo PHY es conectado a un transformador de señal, y este es conectado a la interfaz física RJ-45. En Ethernet las señales se transmiten utilizando la codificación Manchester (*ver Anexo 3*), para que los relojes de los diferentes dispositivos de envío y recepción estén sincronizados. Los niveles lógicos se transmiten a lo largo del medio utilizando niveles de tensión de  $\pm 0.85V$ .

## 3. Operación de control de acceso a los medios

MAC Operation se utiliza para resolver en un sistema de comunicación, el uso de solo un canal de comunicación para la transmisión de información. Sin la operación de control de acceso a los medios se producirían interferencias en la comunicación entre múltiples dispositivos. Una de las funciones de esta operación es la de añadir la dirección MAC de los dispositivos en cada trama que se transmite, para poder identificar a los dispositivos que se comunican. Hay que hacer hincapié en que las direcciones MAC deben ser únicas para cada dispositivo y son identificadores de 48 bits que tienen la siguiente forma: F0:E1:D2:C3:B4:A5. También se suele conocer como dirección física.

Otra de las funciones que se atribuyen a la operación de control de acceso al medio, es la de identificación y corrección de errores en la transmisión, y, si fuera el caso, se descartan esas tramas erróneas o duplicadas.

## 2.2. Introducción comunicación Ethernet en S.O. RTX

Una vez explicado lo que presenta la comunicación vía Ethernet, y los protocolos que se van a utilizar, se debe pasar a la explicación de cómo se ha integrado ésta dentro del sistema operativo RTX y las características que posee.

En primer lugar, se debe saber que se partió de la base de un proyecto existente llamado EasyWEB (Mini-DK2 for LAN8720) cuya funcionalidad es la siguiente: en el momento en el que se establece la conexión con la página web diseñada, se maneja una ésta y se insertan valores dinámicos en ella. Los valores dinámicos son voltajes (de valores arbitrarios) de las entradas analógicas que se representan en una barra horizontal en función del valor de estas. El aspecto que tiene esta página se puede apreciar en la Figura 11.

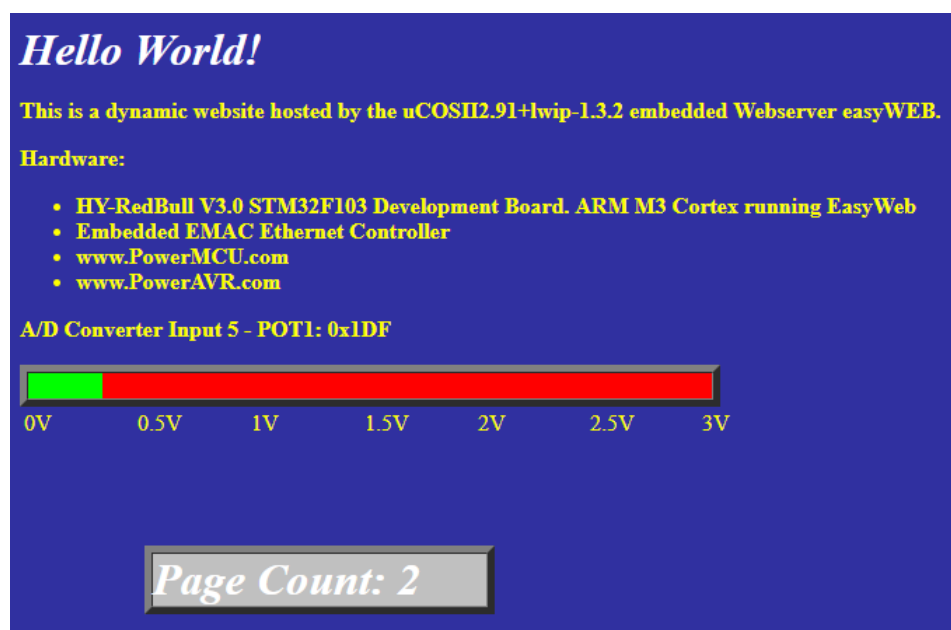


Figura 11. Página web del proyecto EasyWEB (Mini-DK2 for LAN8720)

En el proyecto EasyWEB existen funciones de las que haremos uso y por consiguiente las vamos a introducir brevemente.

- **void TCPLowLevelInit(void):** Esta función es la que se debe llamar en primer lugar, para inicializar la comunicación. Inicia el controlador Ethernet, el timer y varios flags que servirán de ayuda para la comunicación.
- **void DoNetworkStuff (void):** Función que sondea el estado del controlador Ethernet. Además, decodifica las tramas recibidas, mide los tiempos de espera de TCP y responde a las solicitudes de ARP.
- **void TCPPassiveOpen(void):** al llamar a esta función, el microcontrolador se configura en modo 'LISTEN', en el que espera conexiones entrantes.

- ***void TCPActiveOpen(void)***: al contrario que la función anterior, TCPActiveOpen intenta establecer conexión con un host remoto y una solicitud de ARP es enviada hacia este.
- ***void TCPClose(void)***: cierra la conexión TCP con el host remoto. Después de que se cierre la conexión actual, se puede establecer una nueva conexión.
- ***void TCPReleaseRxBuffer(void)***: función que indica que los datos en el búfer de recepción han sido leídos. Por lo tanto, se utiliza ese búfer para una nueva transmisión.
- ***void TCPTransmitTxBuffer(void)***: esta función se encarga de transmitir la información hacia el host remoto con un máximo de 512 Bytes. Si este número se excede la información se enviará en diferentes tramas.

Para que el usuario pueda establecer la comunicación Ethernet con su Cortex – M3, se han creado las siguientes funciones llamadas al supervisor (SVC functions). Estas funciones se encargarán de inicializar, transmitir, recibir y cerrar la conexión Ethernet.

- ***P\_TETH \_\_svc(4) open\_ethernet(osThreadId ID)***: esta función será la encargada de inicializar los registros pertinentes de las placas Cortex – M3 relacionados con Ethernet. También dará la señal a un flag para que se active el timer del protocolo TCP dentro del reloj del sistema, que será incrementado cada 210 milisegundos. Junto con el incremento de este timer se incrementará el ISN (Initial Sequence Number). Esta función *open\_ethernet*, también será la encargada de reservar memoria del sistema para las estructuras de conexión que deberá inicializar el usuario (de esto hablaremos más adelante), y devolverá la dirección de memoria en la que se haya reservado.
- ***void \_\_svc(5) write\_ethernet (char \*data\_tx, osThreadId ID)***: esta función tendrá como objetivo la transmisión de información. En el momento en el que se llame a esta función se comprobará que la conexión está establecida y, además, se liberará el buffer de recepción para poder recibir el ACK de la información que hayamos enviado. La longitud de la información, en Bytes, que transmitimos se guarda en la variable *BytesToSend*. Si el valor de esta variable fuera mayor a 512 Bytes no se podrá enviar de una sola vez toda la información, y habrá que dividirla en varios paquetes.
- ***void \_\_svc(6) read\_ethernet (unsigned char \*data\_rx, osThreadId ID)***: función para recibir los datos que nos llegan vía Ethernet. La variable que nos indica que nos han llegado datos al buffer de recepción es *TCPRxDataCount*, que además nos indica la longitud de los datos recibidos. Entonces únicamente debemos copiar del buffer *TCP\_RX\_BUF* en *data\_rx* justamente la longitud que se nos indica en *TCPRxDataCount*.

- **`void __svc(7) connect_ethernet(struct OS_TETH *add, osThreadId ID, uint8_t num_con)`**: esta función tiene varias tareas a realizar; la primera será añadir una nueva estructura de conexión al listado de todas las conexiones que tendrá la aplicación que requiera el usuario. Otra tarea será la de establecer la estructura de conexión que se va a ejecutar en este momento. En el caso de que se hayan creado dos o más estructuras en un mismo hilo, el usuario debe indicar a través de `num_con` la estructura que quiere elegir para la conexión. Con esto, una vez obtenida esa estructura, y en función de los parámetros que tenga, se configurará la comunicación vía ethernet en el microcontrolador en modo cliente o en modo servidor.
- **`void __svc(8) close_ethernet (osThreadId ID, struct OS_TETH *cls)`**: función que se encarga tanto de la liberación de memoria reservada para las estructuras de conexión, como del cierre de conexión totalmente en el momento en el que se hayan liberado todas las estructuras y el usuario ya no desee utilizar la conexión Ethernet.

Para que la comunicación Ethernet se ponga en marcha, el primer paso que el usuario debe realizar en su aplicación es rellenar una estructura a la que se le reservará memoria con la función `open_ethernet()`. El tipo de estructura lo proporciona el sistema operativo y es la que viene reflejada en la Figura 12 y tendrá los siguientes parámetros; algunos de ellos se deberán rellenar por el usuario y otros los rellenará el propio SO:

```
typedef struct OS_TETH {
    unsigned char ID;
    uint8_t IP_1;
    uint8_t IP_2;
    uint8_t IP_3;
    uint8_t IP_4;
    uint8_t S_C;
    uint16_t P_EXT;
    uint16_t P_INT;
    uint8_t n_con;
    struct OS_TETH *next;
}*P_TETH;
```

Figura 12. Tipo de estructura proporcionada por el SO

- **`unsigned char ID`**: este primer parámetro será rellenado por el sistema operativo, e indicará el ID del hilo en el que se haya configurado la estructura.
- **`uint8_t IP_1...IP_4`**: estos cuatro parámetros los deberá rellenar el usuario con los valores de la IP remota a la que se requerirá conectar cuando el microcontrolador funcione como cliente. Cuando el microcontrolador funcione como servidor estos cuatro parámetros no serán de utilidad.



- **uint8\_t S\_C**: campo a rellenar por el usuario en el cual se indicará en qué modo debe trabajar el microcontrolador para esta estructura. Existen dos macros CLIENT y SERVER que indicarán el modo de conexión cliente y servidor, respectivamente.
- **uint16\_t P\_EXT**: puerto externo utilizado en modo cliente, se debe rellenar por el usuario.
- **uint16\_t P\_INT**: puerto interno utilizado en modo servidor, se debe rellenar por el usuario.
- **uint8\_t n\_con**: parámetro que tiene la función de identificar a la estructura, en el caso de que se haya creado más de una estructura en un mismo hilo. Se debe rellenar por el usuario y tendrá que tener un identificador único para cada estructura en un mismo hilo.

También como primer paso, se deben especificar la dirección IP (Figura 13) que identificará al microcontrolador, la dirección física (Figura 14), y la dirección de la puerta de enlace de nuestro entorno (Figura 15). Además, se debe definir la subred (255.255.255.0). Tanto la dirección IP como la dirección del Gateway se pueden configurar en el archivo tcpip.h y la dirección física se debe configurar en EMAC.h.

```
#define MYIP_1      192
#define MYIP_2      168
#define MYIP_3      1
#define MYIP_4      170
```

*Figura 13. Dirección IP del microcontrolador*

```
#define MYMAC_1      0x1E
#define MYMAC_2      0x30
#define MYMAC_3      0x6c
#define MYMAC_4      0xa2
#define MYMAC_5      0x45
#define MYMAC_6      0x5e
```

*Figura 14. Dirección física del microcontrolador*

```
#define GWIP_1      192
#define GWIP_2      168
#define GWIP_3      1
#define GWIP_4      1
```

*Figura 15. Dirección IP de la puerta de enlace*

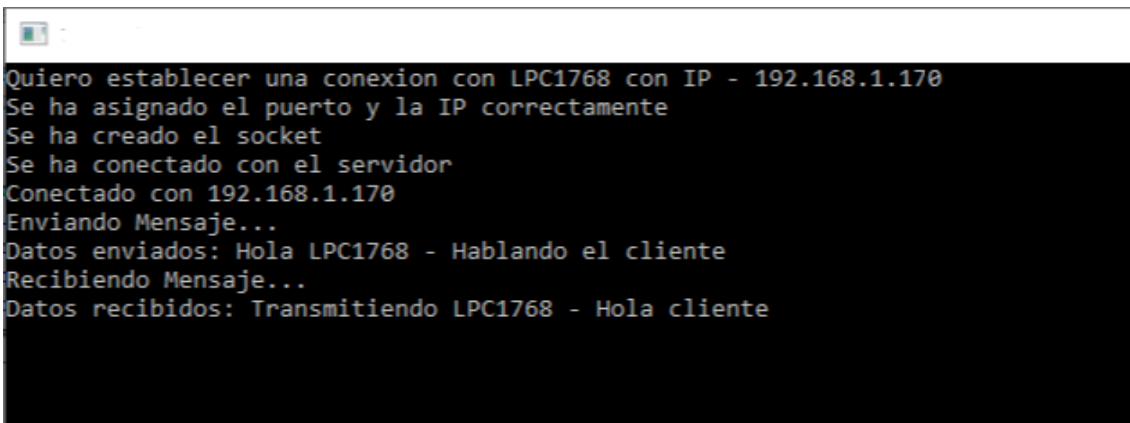
Una vez configurados todos estos parámetros, el usuario ya puede realizar su aplicación utilizando las funciones SVC desarrolladas, y empezar a comunicarse vía Ethernet a través del microcontrolador. Para demostrar el funcionamiento de la comunicación Ethernet, hemos decidido actuar como usuarios. Para ello se han programado dos aplicaciones que se ejecutaran en el terminal del sistema operativo

Windows, una aplicación será **servidor.exe** y otra aplicación será **cliente.exe**. Haremos uso de ellas en los respectivos apartados. Su programación se puede ver en el anexo (Anexo 1 y Anexo 2, respectivamente).

### 2.2.1. Comunicación Ethernet: Servidor – Cliente

Para que nuestro microcontrolador trabaje como servidor se debe llamar a la función `TCPPassiveOpen()` que establece al microcontrolador en modo LISTENING, en el que escucha las conexiones entrantes en un determinado puerto `TCPLocalPort`. En el momento en el que se establece la conexión, existe una variable(`SocketStatus`) que nos indica este hecho; desde este momento podemos tanto recibir como transmitir información con el cliente.

Se ha elegido la conexión del microcontrolador (servidor), con el ordenador que trabajaría como cliente. Ahora haremos uso del programa **cliente.exe** anteriormente programada en el SO Windows.



```
Quiero establecer una conexion con LPC1768 con IP - 192.168.1.170
Se ha asignado el puerto y la IP correctamente
Se ha creado el socket
Se ha conectado con el servidor
Conectado con 192.168.1.170
Enviando Mensaje...
Datos enviados: Hola LPC1768 - Hablando el cliente
Recibiendo Mensaje...
Datos recibidos: Transmitiendo LPC1768 - Hola cliente
```

*Figura 16. Aplicación cliente en el sistema operativo Windows – estableciendo conexión con LPC1768*

En la Figura 16 se puede observar que tanto el cliente como el servidor han establecido la conexión y se han mandado unos mensajes *“Hola LPC1768 – Hablando el cliente”* y *“Transmitiendo LPC1768 – Hola cliente”*. Para comprobar la recepción del mensaje en el microcontrolador se ha inicializado un array para guardar la información entrante, ilustrado en la Figura 17.

mensaje_x	0x10000398 "Hola LPC1768 - Hablando el clien..."	unsigned char[40]
[0]	0x48 'H'	unsigned char
[1]	0x6F 'o'	unsigned char
[2]	0x6C 'l'	unsigned char
[3]	0x61 'a'	unsigned char
[4]	0x20 ' '	unsigned char
[5]	0x4C 'L'	unsigned char
[6]	0x50 'P'	unsigned char

**Figura 17.** Array que contiene la información que le llega al microcontrolador a través de la comunicación Ethernet

Por lo tanto, podemos concluir que la conexión Servidor – Cliente ha sido satisfactoria, obteniendo los mensajes sin pérdida alguna en ambas partes.

### 2.2.2. Comunicación Ethernet: Cliente – Servidor

Nuestro microcontrolador también puede trabajar como cliente, para ello se llamará a la función llamada al supervisor *connect\_ethernet()* con la estructura con los parámetros adecuados para trabajar como cliente. En esta función llamada al supervisor, se utilizará la función *TCPActiveOpen()* que establecerá al microcontrolador en modo CLIENTE enviando un paquete inicial SYN al servidor como parte del establecimiento de conexión en tres pasos. Aquí utilizaremos la aplicación servidor que se programó en el SO Windows que esperará una conexión entrante, y nos indicará desde donde se ha realizado la conexión. Lo siguiente que se realiza es el intercambio de información, en este caso, la aplicación enviará el mensaje *“Hola Cliente, hablando el servidor”* y el microcontrolador una vez recibido el mensaje, transmitirá el mensaje *“Hola servidor – Hablando el Cliente”*.

El resultado de la comunicación entre el microcontrolador y el servidor se puede observar en la Figura 18.

```

Esperando conexiones entrantes...
Conexion entrante desde: 192.168.1.170 - Direccion Mini-DK2
Enviando Mensaje...
Datos enviados: Hola Cliente, hablando el servidor
Recibiendo Mensaje...
Datos recibidos: Hola servidor - Hablando el Cliente

```

**Figura 18.** Aplicación servidor en el sistema operativo Windows – estableciendo conexión con LPC1768

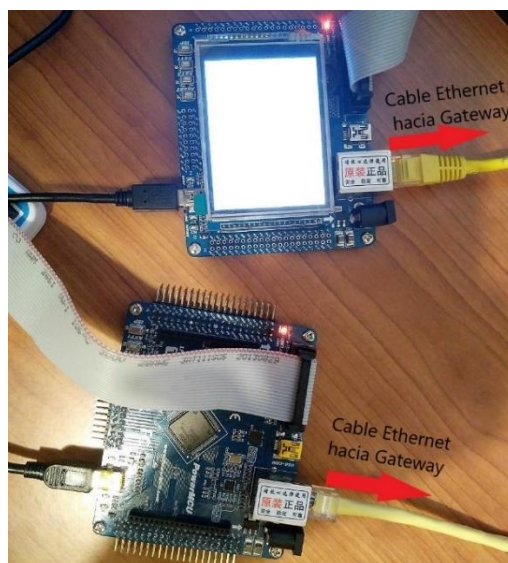
El mensaje recibido en el microcontrolador es el enviado por la aplicación, y se asemeja a lo ya visto en la Figura 17.

### 2.2.3. Comunicación Ethernet – dos microcontroladores.

Al igual que se puede establecer la comunicación con el ordenador, se puede conectar dos microcontroladores para que se comuniquen entre ellos. Para esto uno de ellos debe establecer la comunicación como cliente y el otro como servidor. Además, deben tener diferentes IPs y diferentes direcciones físicas, por este motivo se va a cambiar a uno de ellos estos parámetros en los archivos anteriormente indicados (tcpip.h y EMAC.h).

Una vez realizado el ajuste, editamos lo que se van a enviar estas dos placas entre sí: *“Conexión placa 1”* y *“Conexión placa 2”* es la información que se van a transmitir.

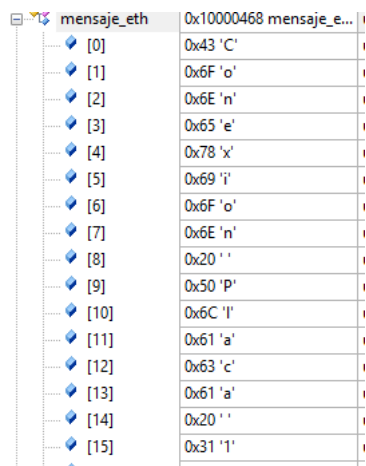
En la Figura 19 podemos visualizar el conexionado de los dos microcontroladores, conectados entre sí a través del Gateway del entorno en el que se ha trabajado.



*Figura 19. Conexionado comunicación Ethernet entre dos microcontroladores*

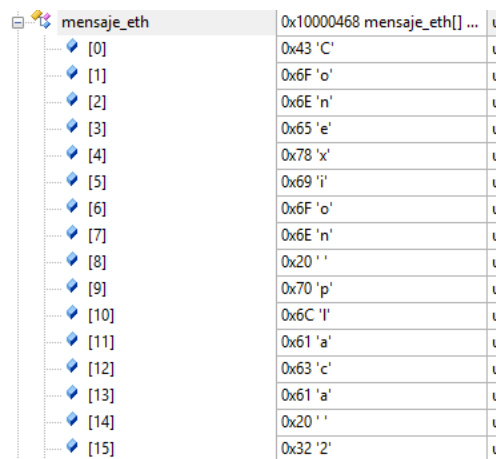
Tras la conexión y puesta en marcha de la aplicación en los ordenadores (se ha necesitado un ordenador adicional para poder realizar la simulación de la comunicación entre dos microcontroladores, en el cual se ha instalado y configurado la misma versión del programa Keil  $\mu$ vision 4.74, además de la instalación de los drivers necesarios), el microcontrolador servidor, encendido antes que el microcontrolador cliente, se dispone a escuchar las conexiones entrantes que le llegan a través del puerto Ethernet.

Una vez conectado el microcontrolador cliente, el cual envía el paquete SYN, se establece la conexión entre los dos microcontroladores servidor – cliente y lo que obtenemos a cada lado es lo que se puede observar en las Figuras 20 y 21.



	mensaje_eth	0x10000468 mensaje_e...
[0]	0x43	'C'
[1]	0x6F	'o'
[2]	0x6E	'n'
[3]	0x65	'e'
[4]	0x78	'x'
[5]	0x69	'i'
[6]	0x6F	'o'
[7]	0x6E	'n'
[8]	0x20	' '
[9]	0x50	'P'
[10]	0x6C	'l'
[11]	0x61	'a'
[12]	0x63	'c'
[13]	0x61	'a'
[14]	0x20	' '
[15]	0x31	'1'

*Figura 20. Mensaje recibido en el microcontrolador servidor*



	mensaje_eth	0x10000468 mensaje_eth[] ...
[0]	0x43	'C'
[1]	0x6F	'o'
[2]	0x6E	'n'
[3]	0x65	'e'
[4]	0x78	'x'
[5]	0x69	'i'
[6]	0x6F	'o'
[7]	0x6E	'n'
[8]	0x20	' '
[9]	0x70	'p'
[10]	0x6C	'l'
[11]	0x61	'a'
[12]	0x63	'c'
[13]	0x61	'a'
[14]	0x20	' '
[15]	0x32	'2'

*Figura 21. Mensaje recibido en el microcontrolador cliente*

Los mensajes, por lo que podemos ver, se han enviado y recibido correctamente por los dos microcontroladores, por lo que podemos concluir que en este apartado también funciona correctamente la conexión Ethernet.

En conclusión, se ha comprobado la conexión Ethernet en diferentes modos de trabajo y en todos ellos se han obtenido resultados satisfactorios, con lo que podemos dejar constancia de que la funcionalidad se ha integrado correctamente dentro del sistema operativo que estamos desarrollando.

### 3. Pantalla LCD

Como indica el título de este apartado, con lo siguiente que vamos a trabajar es con la pantalla TFT-LCD de la placa Mini – DK2. Al igual que con los apartados anteriores se van a crear funciones llamadas al supervisor, que mantendrán separado al usuario final con el acceso directo al hardware. Con lo que el usuario podrá hacer uso de estas funciones para realizar sus necesidades, esta vez con la pantalla TFT-LCD.

La pantalla TFT-LCD es un accesorio útil para la representación de información. Es capaz de representar cadenas de texto, letras, números y diferentes figuras (rectángulos, cuadrados, rectas, círculos) que en su conjunto transforman la pantalla TFT-LCD en un hardware de salida muy funcional y potente.

Para realizar el proyecto se ha hecho uso de las librerías proporcionadas en el proyecto “RTC\_ILI9325” el cual hace funcionar el TFT-LCD en modo 16 bits, y se van a utilizar las funciones existentes en el fichero “lcd driver.c”. Estas funciones van a ser explicadas a continuación para una mejor comprensión del proyecto.

- ***void lcdInitDisplay(void)***: función que inicializa el controlador LCD ILI9325 y configura los registros requeridos.
- ***void setRotation(uint8\_t dir)***: esta función establece la dirección de escritura. La variable dir puede valer 0,1,2 ó 3. Por defecto se establece dir = 3 para una escritura de izquierda a derecha y de arriba hasta abajo.
  - dir = 0 indicará que la escritura se realice de derecha a izquierda y de arriba hasta abajo.
  - dir = 1 indicará la escritura de abajo hacia arriba y de izquierda a derecha.
  - dir = 2 indicará la escritura de arriba hacia abajo y de derecha a izquierda.
- ***void getRotation(void)***: devuelve la dirección de escritura establecida.
- ***void fillScreen(uint16\_t color)***: función que rellena la pantalla con el color pasado como argumento en la variable “color”. Los posibles colores son un total de 65535, pero nosotros haremos uso de los representados en la figura 22 y son los colores negro, azul, rojo, verde, cian, magenta, amarillo y blanco.

```
#define BLACK      0x0000
#define BLUE      0x001F
#define RED       0xF800
#define GREEN     0x07E0
#define CYAN      0x07FF
#define MAGENTA   0xF81F
#define YELLOW    0xFFE0
#define WHITE     0xFFFF
```

*Figura 22. Colores definidos en el proyecto ILI9325*

- ***void drawChar(uint16\_t x, uint16\_t y, char c, uint16\_t fColor, uint16\_t bColor, uint8\_t s)***: representa un carácter “c” en el punto (x,y) de color “fColor” con fondo de color “bColor”, y de tamaño “s”.
- ***void drawString(uint16\_t x, uint16\_t y, char \*c, uint16\_t fColor, uint16\_t bColor, uint8\_t s)***: representa una cadena “c” acabada en null en el punto (x,y) de color “fColor” con fondo de color “bColor”, y de tamaño “s”.
- ***void drawCircle(uint16\_t x0, uint16\_t y0, uint16\_t r, uint16\_t color)***: función que dibuja un círculo en el punto (x0,y0) con radio “r” y de color “color”.
- ***void fillCircle(uint16\_t x0, uint16\_t y0, uint16\_t r, uint16\_t color)***: función que rellena un círculo de un color “color” en un punto (x0,y0) y de radio “r”.
- ***void drawRect(uint16\_t x, uint16\_t y, uint16\_t w, uint16\_t h, uint16\_t color)***: dibuja un rectángulo en un punto (x,y) con una anchura “w” y una altura “h” y de un color “color”.
- ***void fillRect(uint16\_t x0, uint16\_t y0, uint16\_t w, uint16\_t h, uint16\_t color)***: rellena un rectángulo en un punto (x0,y0) con una anchura “w” y una altura “h” y de un color “color”.
- ***void drawFastLine(uint16\_t x0, uint16\_t y0, uint16\_t length, uint16\_t color, uint8\_t rotflag)***: dibuja una línea desde una posición inicial (x0,y0) con una longitud “length”, de un color específico “color” y con una orientación especificada en la variable “rotflag” (‘0’ horizontal y ‘1’ vertical).
- ***void drawLine(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, uint16\_t color)***: dibuja una línea desde un punto inicial (x0,y0) hasta un punto final (x1,y1) de un color específico “color”.
- ***void drawPixel(uint16\_t x, uint16\_t y, uint16\_t color)***: representa un píxel en un punto determinado (x,y) y de un color específico “color”.

Todas estas funciones de configuración y representación se van a agrupar en diferentes funciones de llamada al supervisor que servirán para la misma causa, pero con una capa extra entre el usuario y el hardware de la pantalla TFT-LCD.

### 3.1. Función SVC – Configuración pantalla LCD

El usuario dispondrá de la función llamada al supervisor *P\_LCD\_\_svc(9)* *open\_lcd(uint16\_t color,uint8\_t rotation, osThreadId ID)*; la cual tendrá como finalidad servir como iniciador y configurador de los parámetros de la pantalla TFT-LCD. El usuario deberá utilizar esta función, para lo ya comentado, que es la configuración de la pantalla, pero además servirá para devolver un espacio de memoria reservada para una estructura que será utilizada por las demás funciones SVC.

Pero vayamos por partes. En primer lugar, se llamará a la función *lcdInitDisplay()*, que inicializará todos los registros necesarios para el funcionamiento de la pantalla. En segundo lugar, se procede a la reserva de memoria para la estructura. Para ello se define un puntero a un grupo de memoria con los Bytes aproximados que ocupa la estructura (en nuestro caso 220 Bytes), y se llama a la función *rt\_init\_box()*. Una vez hecho esto, ya se puede reservar memoria con *rt\_alloc\_box()* que devolverá la dirección de memoria habilitada. Por último, se limpia el espacio de memoria reservada con la función *memset()* escribiendo ceros en todas las posiciones de la estructura.

Para finalizar con esta función SVC, se rellena la pantalla con un color específico indicado en la variable “*uint16\_t color*” con la función *fillScreen()* y se ajusta la rotación con la función *setRotation()* mediante la variable “*uint8\_t rotation*”. Y para acabar la función SVC devuelve la dirección de memoria reservada para la estructura.

La estructura ya mencionada en varias ocasiones servirá al usuario para rellenar todos los parámetros necesarios para la representación en la pantalla TFT-LCD, con las funciones vistas anteriormente. La estructura tendrá la siguiente forma, vista en la Figura 23.

```
typedef struct OS_LCD {
    uint8_t select;
    uint16_t x;
    uint16_t y;
    uint16_t xl;
    uint16_t yl;
    char c;
    char *s;
    uint16_t color;
    uint16_t bcolor;
    uint8_t size;
    uint16_t radio;
    uint16_t width;
    uint16_t height;
    uint16_t length;
    uint8_t rotflag;
}*P_LCD;
```

**Figura 23.** Estructura con los parámetros necesarios para la representación en la pantalla LCD

La estructura tiene los siguientes parámetros, explicados a continuación:



- **uint8\_t select:** parámetro que tiene la función de indicar que acción quiere realizar el usuario en la pantalla TFT-LCD. Es un parámetro utilizado dentro de un *switch – case* y se debe rellenar por el usuario y podrá coger valores desde el '0' hasta el '9', cada uno indicando una acción diferente.
- **uint8\_t x:** valor de la coordenada X de la pantalla TFT-LCD que podrá tener un valor desde 0 hasta 239.
- **uint8\_t y:** valor de la coordenada Y de la pantalla TFT-LCD que podrá tener un valor desde 0 hasta 319.
- **uint8\_t x1:** valor de la coordenada X que servirá como punto de destino para algunas funciones de la pantalla TFT-LCD.
- **uint8\_t y1:** valor de la coordenada Y que servirá como punto de destino para algunas funciones de la pantalla TFT-LCD.
- **char c:** variable para representar una letra.
- **char \*s:** puntero para representar una cadena de texto en la pantalla TFT-LCD.
- **uint16\_t color:** variable para indicar el color de letras, figuras, etc.
- **uint16\_t bgcolor:** variable para indicar el color de fondo para letras(y/o números) y frases.
- **uint8\_t size:** representa el tamaño de lo que se va a representar en pantalla y puede ser: SMALL, MEDIUM y LARGE.
- **uint16\_t radio:** representa el radio de la circunferencia a dibujar en pantalla. El valor se introduce en píxeles.
- **uint16\_t width:** valor del ancho del rectángulo que se va a representar en pantalla, se introduce el valor en píxeles.
- **uint16\_t height:** valor de la altura del rectángulo que se va a representar en pantalla, se introduce el valor en píxeles.
- **uint16\_t length:** valor de la longitud utilizado para representar una recta en pantalla. El valor se debe indicar en píxeles.
- **uint8\_t rotflag:** variable que indica la orientación de la recta que se va a dibujar en pantalla. El valor "*rotflag*" = '1' representa una recta vertical y por el contrario el valor "*rotflag*" = '0' representa una recta horizontal.

### 3.2. Función SVC – representación en la pantalla LCD

Esta función SVC (*void \_\_svc(10) write\_lcd(P\_LCD lcd, osThreadId ID)*) tendrá la misión de representar en pantalla todo lo que indique el usuario a través de la estructura indicada en la Figura 23, denominada “OS\_LCD”. Por lo tanto, esta función tendrá un argumento que será la propia estructura y, por otro lado, tendrá un argumento “osThreadId ID” necesario para el control de acceso que será explicado en el apartado 6 de este documento.

Dentro de esta función se implementa un *switch – case* cuyo parámetro será el “select” de la estructura “OS\_LCD”. El valor de “select” puede tomar valores desde ‘0’ hasta ‘9’ ambos incluido. Cada uno de los valores del intervalo realizan una función diferente:

- *select* = ‘0’ → llama a la función *drawChar()* y le pasa los valores necesarios indicados en la estructura “OS\_LCD”.
- *select* = ‘1’ → llama a la función *drawString()* con los valores de la estructura “OS\_LCD”.
- *select* = ‘2’ → llama a la función *drawCircle()* con los valores de la estructura “OS\_LCD”.
- *select* = ‘3’ → llama a la función *fillcircle()* con los valores de la estructura “OS\_LCD”.
- *select* = ‘4’ → llama a la función *fillScreen()* con los valores de la estructura “OS\_LCD”.
- *select* = ‘5’ → llama a la función *drawRect()* con los valores de la estructura “OS\_LCD”.
- *select* = ‘6’ → llama a la función *fillRect()* con los valores de la estructura “OS\_LCD”.
- *select* = ‘7’ → llama a la función *drawFastLine()* con los valores de la estructura “OS\_LCD”.
- *select* = ‘8’ → llama a la función *drawLine()* con los valores de la estructura “OS\_LCD”.
- *select* = ‘9’ → llama a la función *drawPixel()* con los valores de la estructura “OS\_LCD”.

En la Figura 24 se ha realizado una especie de tabla ilustrativa con los resultados que se pueden obtener al ejecutar la aplicación introduciendo los parámetros adecuados y para cada posible valor de “select”. No se ha incluido la opción *select* = ‘9’ debido a que no se llega a apreciar la representación de un píxel en este documento.

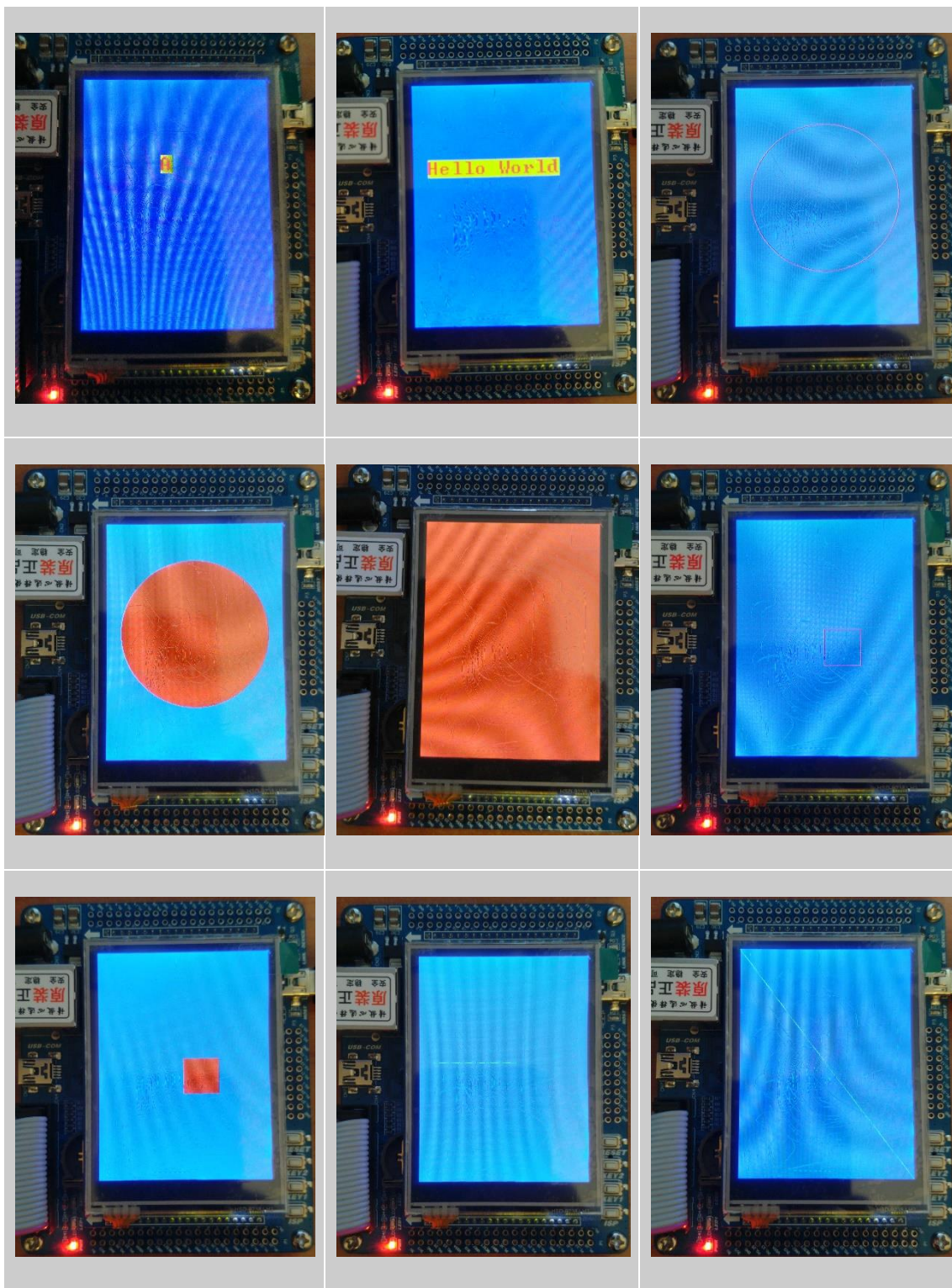


Figura 24. Resultados obtenidos en la pantalla tras ejecutar la aplicación con SO, con select desde 0 a 8, respectivamente.

### 3.3. Función SVC – liberar memoria

El objetivo de esta función llamada al supervisor (*void \_\_svc(11) close\_lcd(osThreadId ID)*) es devolver al sistema operativo la memoria anteriormente reservada en la función *SVC open\_lcd()* de la pantalla TFT-LCD. La memoria reservada para la estructura “*OS\_LCD*” debe ser liberada al terminar de utilizar la pantalla TFT-LCD para que el sistema operativo pueda volver a manejar ese espacio de memoria para sus necesidades. Para ello en primer lugar limpiamos el espacio de memoria correspondiente a la estructura con la función *memset()* escribiendo ceros en todas las posiciones. A continuación, liberamos la memoria con la función *rt\_free\_box(void \*box\_mem, void \*box)* que nos proporciona el sistema operativo RTX.

Además, también libera el espacio de memoria reservada para la estructura de control de acceso, siguiendo los mismos pasos indicados en el párrafo anterior.

Realizadas estas operaciones, el usuario dejaría de trabajar con la pantalla TFT-LCD de forma segura.

## 4. Pantalla táctil – Touch screen

La placa de desarrollo Mini-DK2 basada en el microcontrolador LPC1768 puede trabajar con una pantalla TFT-LCD táctil de 2,8 pulgadas; es la misma pantalla que se utilizó en el desarrollo del apartado anterior, por lo que es un dispositivo combinado que incluye una pantalla TFT-LCD y tecnología táctil en la pantalla. Este dispositivo tiene la posibilidad de mostrar contenido gráfico y además actuar como dispositivo de interfaz.

Las pantallas táctiles utilizan dos métodos para registrar la interacción con la pantalla, llamadas resistiva y capacitiva. En las pantallas resistivas lo que se hace es medir la diferencia de tensión causada por ejercer presión del dedo o un lápiz en la pantalla. Mientras que en las pantallas capacitivas se actúa en el momento en el que se interrumpe la corriente al interactuar con la pantalla. Cabe destacar que las pantallas táctiles no son un tipo de pantalla, sino son un componente que se puede agregar a una pantalla existente. En nuestro caso la pantalla TFT-LCD tiene la tecnología de pantalla resistiva.

Como en otras ocasiones, vamos a partir de un proyecto ya existente que nos servirá de base para integrar esta funcionalidad dentro del sistema operativo RTX.

### 4.1. Función SVC – Open de la pantalla táctil

Se ha creado una función llamada al supervisor *void \_\_svc(12) open\_touch(osThreadId ID)* para la configuración de los registros necesarios para el correcto funcionamiento tanto de la pantalla TFT-LCD como de la pantalla táctil. Para ello, dentro de esta función se han utilizado algunas funciones existentes del proyecto base. Estas son:

- ***void TP\_Init(void)***: función que configura la pantalla táctil con los valores adecuados para su funcionamiento. Se inicializa el periférico SSP.
- ***void LCD\_Initializtion(void)***: función que inicializa y configura la pantalla TFT-LCD con unos valores determinados en los registros correspondientes.
- ***void TouchPanel\_Calibrate(void)***: función necesaria para la calibración de la pantalla táctil. Muestra en la pantalla tres puntos en los que el usuario debe interactuar para que la pantalla táctil funcione (mostrado en la Figura 25; únicamente se visualiza el primer punto de los tres en total).



*Figura 25. Calibración de la pantalla táctil en el microcontrolador LPC1768*

Dentro de esta función llamada al supervisor se llama a las tres funciones explicadas anteriormente para configurar correctamente la pantalla y que ésta trabaje como pantalla táctil.

## 4.2. Función SVC – Write de la pantalla táctil

Función `void __svc(13) write_touch(osThreadId ID)` cuyo objetivo es la representación en la pantalla la entrada táctil sobre ésta. Para cumplir este objetivo se han utilizado las siguientes funciones desarrolladas en el proyecto base:

- **`uint8_t getDisplayPoint(Coordinate *displayPtr, Coordinate *screenPtr, Matrix *matrixPtr)`**: función que detecta la posición en la que se ha interactuado con la pantalla táctil.
- **`void TP_DrawPoint(uint16_t Xpos, uint16_t Ypos)`**: función que representa en la pantalla la posición detectada por la anterior función.

En conjunto con estas dos funciones tenemos una función que representa por pantalla lo que el usuario introduce a través de un lápiz o simplemente a través de uno de sus dedos en la propia pantalla.

## 4.3. Función SVC – Close

En este apartado la función `void __svc(18) close_touch(osThreadId ID)`; realiza la única función de liberación de memoria de la estructura que sirve como control de acceso del proyecto. El apartado de este control se explica detenidamente en el apartado 6 de este documento, por lo que aquí únicamente se indica que existe tal función SVC en la funcionalidad de la pantalla táctil y se expone su objetivo.



## 5. Ficheros

### 5.1. Introducción

En esta parte del documento se aborda la conexión de una tarjeta SD con la placa de desarrollo basada en el microcontrolador Cortex – M3. Las tarjetas SD son dispositivos en formato tarjeta de memoria para multitud de dispositivos tales como teléfonos móviles, tablets, cámaras de fotos, etc. Este estándar fue desarrollado en 1999 y presenta cuatro versiones de tarjetas:

1. Standard Capacity (SDSC)
2. High Capacity (SDHC)
3. Extended Capacity (SDXC)
4. Input/Output (SDIO)

En tres tamaños diferentes:

1. SD estándar original
2. miniSD
3. microSD

La placa de desarrollo Mini – DK2 contiene una ranura para las tarjetas microSD. En específico la tarjeta microSD con la que vamos a trabajar ha sido desarrollada por la compañía Samsung - MMAGR02GUECA-MB(Figura 26).



*Figura 26. Tarjeta microSD empleada en el desarrollo y la comprobación del proyecto*

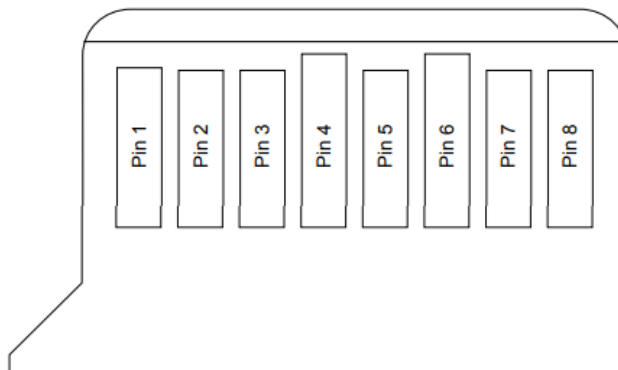
Las tarjetas SD constan de dos regiones: un “núcleo de memoria” en el que se guardan los archivos reales y donde existe el sistema de archivos; y, la segunda región es el “controlador de tarjeta SD” el cual ayuda a comunicar la tarjeta con dispositivos externos, como por ejemplo el microcontrolador.

En este proyecto se utiliza el bus SPI para comunicar la tarjeta microSD con el microcontrolador. El microcontrolador es quien inicia todas las transferencias de datos a través de los canales MOSI y MISO del bus SPI.

A continuación, se presenta la tabla de asignación de pines para el modo SPI de la tarjeta microSD (Figura 27) y el diagrama de pines de esta (Figura 28).

Pin #	SPI Mode		
	Name	Type <sup>1</sup>	Description
1	RSV		Reserved
2	CS	I <sup>3</sup>	Chip Select (neg true)
3	DI	I	Data In
4	VDD	S	Supply voltage
5	SCLK	I	Clock
6	VSS	S	Supply voltage ground
7	DO	O/PP	Data Out
8	RSV <sup>4</sup>		

*Figura 27. Asignación de pines de la tarjeta microSD en el modo SPI*



*Figura 28. Diagrama de pines de la tarjeta microSD*

Además, en la Figura 29 también se presenta el diagrama de la ranura para las tarjetas microSD de la placa de desarrollo Mini – DK2, en el que también se puede visualizar los pines para la comunicación a través del bus SPI, además de los componentes eléctricos utilizados.



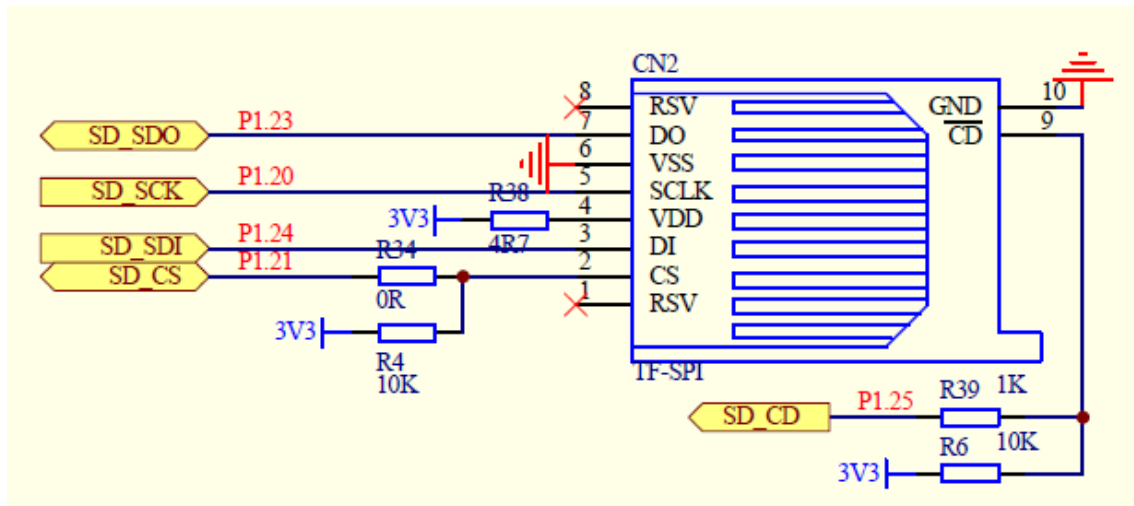


Figura 29. Diagrama de la ranura para las tarjetas SD en la placa de desarrollo Mini – DK2

Se puede comprobar a simple vista que tanto los pines de la tarjeta microSD como de la ranura de la placa de desarrollo coinciden.

## 5.2. Desarrollo de las funciones SVC

Este apartado se centra en la explicación de cómo se han desarrollado las diferentes funciones SVC en el sistema operativo RTX, con el fin de poder trabajar con la tarjeta SD con nuestro microcontrolador, leyendo y escribiendo ficheros dentro de dicha tarjeta.

Para llevar a cabo el objetivo, se ha utilizado un proyecto existente, exento de sistema operativo, cuya función también era la de leer y escribir ficheros en las tarjetas montadas en la placa de desarrollo. Por este motivo se han utilizado tanto funciones como archivos fuente de este proyecto. A continuación, se explican las funciones de este proyecto base que nos han servido para el desarrollo e integración dentro del sistema operativo RTX de la funcionalidad de trabajar con ficheros con nuestro microcontrolador.

- **`void MSD_SPI_Configuration(void)`**: función que configura la comunicación a través del bus SPI de la tarjeta SD.
- **`FRESULT f_mount (BYTE vol, FATFS *fs)`**: monta y desmonta una unidad lógica. La variable “vol” indica el numero de la unidad lógica para ser montada o desmontada. La variable “fs” es un puntero al nuevo objeto del sistema de ficheros (un cero si se quiere desmontar la unidad lógica).
- **`FRESULT f_open (FIL *fp, const TCHAR *path, BYTE mode)`**: función que sirve para abrir o crear ficheros. La variable “fp” es un puntero al objeto de fichero

en blanco. La variable *“path”* es un puntero al nombre del fichero. La variable *“mode”* indica el modo de acceso y los flags de modo de abrir el fichero.

- ***FRESULT f\_write (FIL \*fp, const void \*buff, UINT btw, UINT \*bw)***: función que se encarga de escribir dentro del fichero. La variable *“fp”* es un puntero al fichero. La variable *“buff”* es un puntero a un buffer de datos a escribir. La variable *“btw”* indica el número de bytes a escribir y, por último, la variable *“bw”* es un puntero al número de bytes escritos.
- ***FRESULT f\_close (FIL \*fp)***: función cuya función es cerrar el fichero abierto. Tiene como argumento de la función la variable *“fp”* que es un puntero al fichero que se quiere cerrar.
- ***FRESULT f\_mkdir (const TCHAR \*path)***: crea un nuevo directorio. Como argumento de la función tiene la variable *“path”* que es un puntero a la ruta del directorio.
- ***FRESULT f\_read (FIL \*fp, void \*buff, UINT btr, UINT \*br)***: función cuyo objetivo es leer el contenido de un fichero. La variable *“fp”* es un puntero al fichero. La variable *“buff”* es un puntero al buffer de datos, en el cual se van a guardar los datos leídos del fichero. La variable *“btr”* indica el número de bytes que se quieren leer del fichero. Por último, la variable *“br”* es un puntero al número de bytes leídos.
- ***FRESULT f\_unlink (const TCHAR \*path)***: función cuyo objetivo es la supresión de un fichero o directorio. Como argumento tiene la variable *“path”* que es un puntero a la ruta del fichero o directorio.
- ***FRESULT scan\_files (char\* path)***: función que escanea el directorio en busca de todos los ficheros presentes en el. La función tiene como argumento la variable *“path”*, un puntero al directorio.
- ***int SD\_TotalSize(void)***: función que indica al usuario la cantidad de memoria que posee la tarjeta SD y la cantidad de memoria disponible.

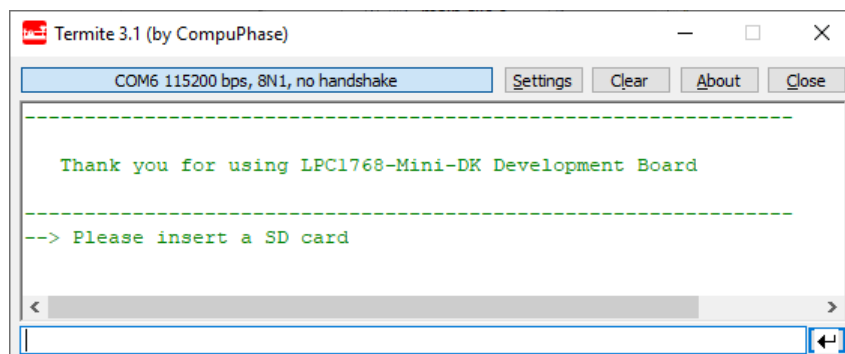
Todas estas funciones se van a utilizar en diferentes funciones llamada al supervisor, para que, en su conjunto, el usuario pueda trabajar con ficheros con el microcontrolador Cortex – M3, disponiendo de multitud de posibilidades.

### 5.2.1. Función SVC – Open

Esta es la primera función llamada al supervisor que se ha creado para poner en marcha el manejo de ficheros en la placa de desarrollo. Dentro de esta función se ha inicializado la comunicación a través del bus SPI con la función `MSD_SPI_Configuration()`.

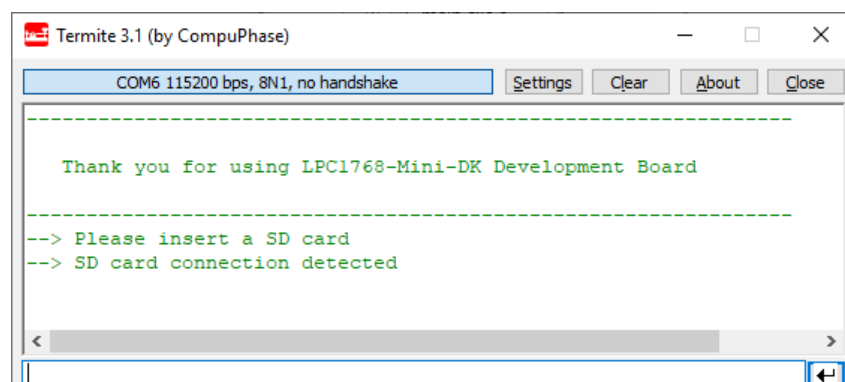
Además, se ha inicializado el puerto UART para poder visualizar todo lo que se va a realizar con los ficheros, tanto lectura de estos como escritura y algunos comentarios adicionales que se han añadido para seguir el correcto funcionamiento del microcontrolador con la tarjeta SD.

Esta función comprueba si hay alguna tarjeta SD conectada al microcontrolador, y en el caso de que no estuviera conectada ninguna tarjeta, espera hasta que detecte la conexión de una, mostrado en la Figura 30.



*Figura 30. El microcontrolador espera a que se detecte una conexión de una tarjeta SD*

Una vez conectada una tarjeta SD (Figura 31), el sistema se encarga de montar la tarjeta utilizando la función `f_mount()` anteriormente vista.



*Figura 31. El microcontrolador detecta la conexión de la tarjeta SD*

Con esto ya podríamos trabajar con la tarjeta SD, utilizando las funciones SVC que se explican más adelante.

### 5.2.2. Función SVC – Write

Función en la que se van a realizar varias tareas, con la opción de elegir cada una a través de un switch – case, indicando la opción a la función a través del argumento “option”. Las opciones son tres:

1. Creación de un fichero nuevo y escritura dentro de este, en un directorio determinado.
2. Creación de un directorio nuevo.
3. Escritura dentro de un archivo existente.

```
#define NEW_FILE    0
#define MKDIR      1
#define OPEN_FILE   2
```

*Figura 32. Opciones disponibles para indicar la opción elegida a la variable “option” en la función SVC write\_file();*

En el primer caso, el nuevo fichero se crea en un directorio determinado indicado por la variable “file” que es un puntero argumento de la función SVC write\_file(). Esta variable indicada tanto el directorio como el nombre del nuevo fichero que se va a crear. Para realizar esta acción se llama a la función f\_open() y se le pasan los argumentos correctos para terminar de crear correctamente el nuevo fichero. Tras esto, se llama a la función f\_write() la cual se va a encargar de escribir lo que el usuario indique dentro del fichero creado anteriormente. En el caso de que todo haya ido correctamente, el sistema imprime por pantalla “-->File successfully created”, en el caso contrario el sistema imprime “-->File created in the disk”, pudiendo significar que ya existe un fichero de idéntico nombre en este directorio.

La segunda opción crea un nuevo directorio dentro de la tarjea SD. Si el directorio se ha creado satisfactoriamente el sistema imprime “-->Directory successfully created”. Pero también puede pasar que el sistema imprima alguno de los siguientes errores:

- “-->There is a directory with the same name”: este error indica que no se puede crear un nuevo directorio en esta ruta, ya que ya existe un directorio con el mismo nombre.
- “-->No space to allocate a new cluster”: este error indica que no hay espacio suficiente para crear un nuevo directorio.

La tercera opción, OPEN\_FILE, se encarga de abrir un fichero ya existente dentro del directorio y escribir dentro de este fichero lo que indique el usuario. El nuevo contenido se añade a continuación de lo ya existente en el fichero. Esta opción devuelve por pantalla, si todo ha ido correctamente, “-->Successfully written in a existing file”. Y si, en el caso de que haya habido un error, imprime por pantalla “-->File doesn't written”.

Con esta función SVC, y las tres opciones que nos permite utilizar, ya podríamos manejar la escritura de diferentes ficheros y la creación de directorios. El siguiente paso será la lectura de ficheros existentes dentro de la tarjeta SD.

### 5.2.3. Función SVC – Read

Al igual que en la función SVC `write_file()`; existente varias opciones a elegir para obtener diferentes acciones del microcontrolador en el manejo de una tarjeta SD.

También son tres opciones:

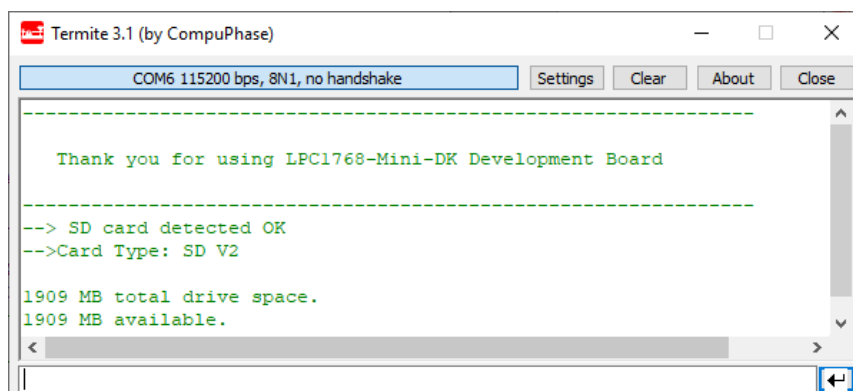
1. Leer el contenido de un fichero existente.
2. Obtener la memoria total y la memoria disponible de la tarjeta SD.
3. Mostrar todos los directorios y todos los ficheros existentes dentro de la tarjeta SD.

```
#define READ_FILE    0
#define SHOW_SIZE    1
#define SHOW_FILES   2
```

**Figura 33.** Opciones disponibles para indicar la opción elegida a la variable “option” en la función SVC `read_file()`;

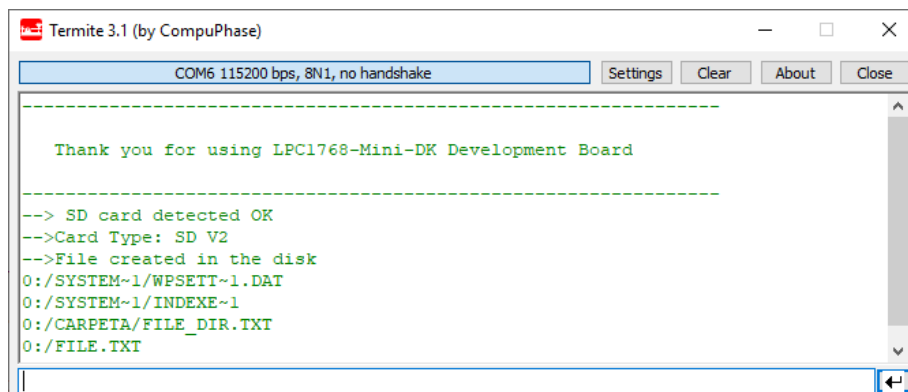
La primera opción nos permite obtener el contenido de un fichero en concreto. Para ello se debe indicar la ruta del fichero que se quiere leer. Lo primero que hace el microcontrolador, es la apertura a través de la función “`f_open`” del fichero indicado, y a continuación lee a través de la función “`f_read`” el contenido del fichero. Al acabar y guardar el contenido leído en un buffer, cierra el fichero utilizando la función “`f_close`”, e imprime por pantalla la información leída anteriormente. Con esto acabaría la ejecución de la primera opción, si esta fuera elegida por el usuario.

La segunda opción que es capaz de elegir el usuario nos devuelve tanto el tamaño completo de la tarjeta SD utilizada, como la memoria libre de la misma.



**Figura 34.** Opción `SHOW_SIZE` de la función SVC `read_file()`; indica la memoria de la tarjeta SD y la memoria disponible

La tercera opción es capaz de indicarnos todos los ficheros existentes en la memoria de la tarjeta SD indicándonos la ruta de estos. El único parámetro que se necesita proporcionar para que se ejecute correctamente es el directorio principal con el que se ha montado la tarjeta SD, que por defecto es "0:".



**Figura 35.** Opción `SHOW_FILES` de la función `SVC read_file()`; indica los ficheros existentes en la tarjeta SD

Con estas tres opciones el usuario ya puede leer y ver lo que esconde la tarjeta SD y además visualizar el tamaño hay disponible. Por último, nos queda explicar la función SVC encargada de eliminar ficheros y desmontar la tarjeta SD del microcontrolador.

#### 5.2.4. Función SVC – Close

Como se ha indicado en el último párrafo del apartado anterior, esta función SVC va a tratar de eliminar los ficheros existentes en la memoria de la tarjeta SD, esto, por un lado, y, por otro lado, también tendrá la opción de desmontar la tarjeta SD que se montó con la función `SVC open_file()`;

Además de lo comentado existe otra opción, a parte, que será utilizada en un apartado posterior de este documento. Dicho esto, vamos a explicar la labor de esta función SVC del sistema operativo. Las tres opciones comentadas anteriormente se exponen en la Figura 36.

```
#define REMOVE      0
#define UNMOUNT    1
#define FREE_MEM    2
```

**Figura 36.** Opciones disponibles para indicar la opción elegida a la variable "option" en la función `SVC close()`;

Cuando el usuario elige la opción "`REMOVE`", esto quiere decir que se desea eliminar algún elemento de la tarjeta SD, pudiendo ser un fichero o un directorio. Por lo tanto, el usuario debe indicar la ruta de este fichero o directorio que será eliminado por el microcontrolador de la memoria de la tarjeta SD. El microcontrolador tendrá

diferentes respuestas en función de si la acción se ha realizado correctamente o ha habido algún error:

- Si la operación de eliminación ha terminado satisfactoriamente se imprimirá por pantalla el siguiente mensaje *-->Removed successfully*.
- Si el nombre del directorio no coincide se imprimirá *-->The path name format is invalid*.
- En el caso de que el acceso sea denegado se imprimirá: *-->Acces denied due to prohibited access or directory full*.
- Y, por último, si hay un error de aserción se imprimirá: *-->Assertion failed*.

La opción *"UNMOUNT"*, será elegida por el usuario en el caso de que este haya terminado de trabajar con la tarjeta SD. Por lo tanto, el sistema operativo será el encargado de desmontar la tarjeta SD y ya no se podrá trabajar con esta, hasta que no se vuelva a montar con la función `open()`; vista anteriormente. El microcontrolador mostrará diferentes mensajes en función de si se ha completado la acción correctamente o no.

- *-->Unmounted successfully*, será el mensaje mostrado si se ha desmontado la tarjeta SD correctamente.
- *-->The logical drive number is invalid*, mensaje mostrado si no se ha podido desmontar la tarjeta SD debido a que se ha indicado incorrectamente el número del disco lógico.

Con esta función SVC explicada terminamos la explicación del funcionamiento del microcontrolador con los ficheros en una tarjeta SD.

## 6. Control de acceso

Este apartado surge como consecuencia de la necesidad de controlar el acceso y el uso, de las diferentes funcionalidades que se han desarrollado en este proyecto, por el usuario.

El problema aparece en el momento en el que el usuario, por desconocimiento o por falta de atención, intenta acceder y configurar a través de las funciones *SVC open()*; la misma funcionalidad del microcontrolador en diferentes hilos. Esto provoca problemas, que, por un lado, el microcontrolador nos puede redirigir directamente a una excepción Hardfault y el programa deje de funcionar totalmente; o, por otro lado, el funcionamiento del programa continuaría, pero con alta probabilidad de malfuncionamiento del proyecto total del usuario. Por este motivo, se procederá a añadir a las funcionalidades incorporadas al sistema operativo desarrollado, un sistema de control que funcionará de una forma muy básica.

El control de acceso que se ha pensado implementar dentro del SO tendrá el siguiente funcionamiento: Una vez que el usuario inicialice una de las características de la placa añadidas al SO(UART, Ethernet, pantalla LCD y pantalla LCD táctil o el manejo de ficheros) en un determinado hilo, desde ese mismo instante, este hilo es el único que será capaz de trabajar con aquella funcionalidad utilizada por el usuario. Todos los demás hilos que intenten acceder a la configuración a través de la función *open()*; de esta misma funcionalidad no serán tomados en cuenta. La única forma de que otro hilo sea capaz de manejar la funcionalidad utilizada es que el usuario llame a la función *close()*; de este modo liberando dicha funcionalidad.

Para llevar todo este planteamiento a cabo, se ha pensado crear una estructura, la cual guardará un identificador del hilo que ha llamado a la función *open()*; de unas de las funcionalidades disponibles del proyecto. Por lo que el acceso a las funciones *write()*, *read()* o *close()* solo se podrá acceder con ese identificador que se pasará como argumento de las funciones *SVC* comentadas(y, en el caso del módulo Ethernet, la función *connect\_ethernet()*).

Aquí se puede observar una excepción, la cual consiste en la posibilidad de utilizar todas las funciones anteriormente comentadas en otro hilo que no haya sido el que originó el comienzo del funcionamiento de una determinada característica del microcontrolador, únicamente proporcionando el identificador del hilo original, como argumento de la función en otro hilo distinto. Pero esto no se considera como un problema.



Continuando con la presentación de la estructura, que servirá como control de acceso, se debe decir que se creará una para cada una de las cinco diferentes funcionalidades proporcionadas en este proyecto y todas ellas tendrán la forma que se puede observar en la Figura 37.

```
typedef struct T_SYNC_OS{  
    unsigned char ID;  
}*P_TSINC;
```

*Figura 37. Estructura de control de acceso, ejemplo considerado del Touch Panel*

El control de acceso será similar a todas las funcionalidades vistas en el proyecto, por lo que se va a abordar como ejemplo de explicación una de ellas, y será posible la extrapolación de esta explicación a las demás funcionalidades. Como ejemplo se considerará la funcionalidad de la pantalla táctil, vista la forma de la estructura en la Figura 35.

Lo primero que se hace, para poder trabajar con una estructura dentro del sistema operativo, es la reserva de memoria para ésta. Por lo tanto, se procede a declarar un conjunto de bytes que se pueden considerar como un conjunto de memoria a través de la definición *\_declare\_box (pool, size, cnt)* donde *pool* indica el nombre de la variable del conjunto de memoria, *size* indica el número de bytes en cada bloque de memoria y *cnt* indica el número de bloques en el conjunto de memoria. También se inicializa una variable *unsigned int t\_reserved* que actuará como flag indicador de que ya se ha reservado la memoria necesaria para la estructura.

A continuación, una vez declarado el conjunto de memoria, se debe inicializar este conjunto de memoria a través de *rt\_init\_box(void \*box\_mem, unsigned int box\_size, unsigned int blk\_size);* cuyos parámetros son *box\_mem* que es el nombre de la variable utilizada para definir el conjunto de memoria (*pool*). Las variables *box\_size* y *blk\_size* indican el tamaño del conjunto de memoria y el tamaño de cada bloque, respectivamente. En nuestro caso, se debe indicar el tamaño del conjunto de memoria reservado y el tamaño de la estructura.

Por último, se asigna la memoria a través de la función *rt\_alloc\_box(void \*box\_mem)* en la que se debe utilizar la variable utilizada para declarar el conjunto de memoria. Una vez realizados todos estos pasos la variable *t\_reserved*, que funciona como flag, es cambiada de valor para que no se repita el mismo proceso de reserva de memoria en las sucesivas llamadas a la función *open()*. Cabe mencionar que este proceso se ha realizado dentro de un *if* con parámetro la variable *t\_reserved*.

Una vez reservada la memoria, se debe identificar el hilo que ha utilizado la función *open()* para ello, el usuario debe pasar como argumento de la función *open()* la ID del

hilo (para obtener la ID de un hilo puede utilizar la función *osThreadGetId()*). Una vez realizado esta operación, dentro de esta función se obtendrá un numero identificador del hilo por medio de la utilización de la función *P\_TCB rt\_tid2ptcb (osThreadId thread\_id)* que devuelve una estructura “*P\_TCB*” que contiene dentro de ella una variable que contiene el número entero identificador del hilo(en la variable *task\_id* de la estructura en cuestión), que es único para cada uno de ellos.

De este modo, se guarda, una única vez, este número identificador en nuestra propia estructura, y cada vez que se acceda a una función SVC desarrollada en este proyecto, se comprueban estos números y en el caso de que coincidan, se continua con la ejecución del código pertinente, y en el caso, de que no coincidan, se salta hasta el final de la función y por consiguiente se sale de la ejecución de la función.

Para obtener acceso a la ejecución desde otro hilo, se debe llamar a la función *close()* que liberará la memoria reservada para la estructura de control de acceso y limpiará el flag *t\_reserved*. Con esto ya se podría ejecutar la funcionalidad de la pantalla táctil desde otro hilo diferente.

En el caso del apartado 5 de este documento, en el que se aborda el manejo de ficheros con el microcontrolador, la función *close\_file()* se debe llamar indicando como argumento de la función la definición *FREE\_MEM* para liberar el espacio de memoria reservado para la estructura de control de acceso y así poder trabajar desde otro hilo o simplemente terminar de trabajar con el sistema de ficheros.

## 7. Futuras líneas de investigación

Con la incorporación de las funcionalidades de la placa de desarrollo Mini-DK2 vistas en este proyecto, se ha abordado una parte de todas las características de esta placa basada en Cortex - M3. Por lo que las líneas futuras de investigación y desarrollo pueden evolucionar en este aspecto, incorporando los restantes periféricos que poseen estos microcontroladores:

- Dos interfaces CAN.
- Convertidor ADC de 12 bits y un convertidor DAC de 10 bits
- Bus I2C, 3 buses SPI y un bus I2S
- 6 salidas PWM

Además, se puede seguir desarrollando las funcionalidades vistas en este proyecto, corrigiendo errores que puedan surgir en el uso; y optimizando el uso de la memoria del sistema operativo. Se pueden ampliar las funciones creadas dando mayor posibilidad de elección al usuario final de las placas de desarrollo, ya que se ha visto que algunos parámetros de este proyecto se seleccionaban por defecto.

Por otro lado, el mercado de los microcontroladores es muy amplio lo que nos da la posibilidad de realizar proyectos similares con microcontroladores diferentes y microprocesadores con muy amplia variedad a nuestra disposición. Por ejemplo, dentro de la familia Cortex de ARM, con perfil microcontrolador nos encontramos con

- Cortex – M35P
- Cortex – M33
- Cortex – M23
- Cortex – M7
- Cortex – M4
- Cortex – M1
- Cortex – M0+
- Cortex – M0

Una gran variedad de microprocesadores para microcontroladores que nos proporcionaría mucho trabajo para un periodo de tiempo extenso.

## Conclusión

En el desarrollo de este proyecto se han conseguido incorporar diferentes funcionalidades, de forma satisfactoria, de un microcontrolador basado en Cortex – M3 dentro de un sistema operativo existente para este tipo de microcontroladores. Con esto el usuario de una placa de desarrollo con un microcontrolador basado en Cortex – M3, tiene a su disposición un sistema operativo que le otorga la posibilidad de utilizar las funcionalidades incorporadas por nosotros en el sistema operativo, con tan solo la llamada a las funciones SVC desarrolladas durante este proyecto, proporcionando la facilidad de uso y la seguridad a la hora de utilizar los periféricos de este microcontrolador.

Hablamos de facilidad de uso, debido a que el usuario necesita utilizar un número reducido de funciones que le otorgan la posibilidad de trabajar con el periférico que necesite. Esto comparado con la necesidad de programar un periférico a mano y que realice la funcionalidad que se desea, lo que antes provocaba la escritura de muchas líneas de código, ahora se resume en una función a la que se le deben pasar como argumentos los parámetros requeridos.

Se habla de seguridad a la hora de trabajar con los periféricos añadidos al sistema operativo, debido a que el usuario no interactúa con el hardware, todo pasa a través del Kernel del sistema operativo.

Al haber conseguido que funcionen los periféricos en el sistema operativo, y proporcionando las ventajas anteriormente comentadas podemos decir que hemos cumplido nuestros objetivos principales que nos planteábamos al comienzo del desarrollo del trabajo de fin de grado.

Por otro lado, en el transcurso del desarrollo del proyecto, se han visto diferentes campos como son el Transmisor-Receptor Asíncrono Universal, la comunicación vía Ethernet, la programación de una pantalla LCD y la programación de una pantalla táctil, además de la incursión en el manejo de ficheros desde un microcontrolador en una SD Card. En todos estos campos se han obtenido unos conocimientos básicos que me pueden servir en un futuro para seguir investigando en cada uno de ellos o, me pueden ayudar en mi carrera profesional si en algún momento surge la necesidad de trabajar con alguna de estas características. Por lo que este proyecto ha sido un impulso personal en algunos ámbitos que no se habían contemplado en el transcurso del aprendizaje en el grado de ingeniería electrónica y automática industrial.

Ha sido un trabajo de desarrollo personal y, además, de mucha utilidad para los usuarios que quieran trabajar con los periféricos con este tipo de microcontroladores.

# Anexos

## Anexo 1. Código aplicación servidor en SO Windows

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winsock.h>
char SendBuff [512], RecvBuff [512];
int main (int argc, char *argv []) {
    WSADATA wsaData;
    SOCKET conexion_servidor,conexion_cliente;
    struct sockaddr_in server, client;
    int resp, stsize;
    resp=WSAStartup(MAKEWORD(1,0), &wsaData);
    if(resp){
        printf("Error al inicializar socket\n");
        getchar();return resp;
    }
    conexion_servidor=socket (AF_INET, SOCK_STREAM, 0);
    if(conexion_servidor==INVALID_SOCKET) {
        printf("Error al crear socket\n");
        getchar(); WSACleanup();return WSAGetLastError();
    }
    memset(&server, 0, sizeof(server));
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_family = AF_INET;
    server.sin_port = htons(6000);
    resp=bind(conexion_servidor, (struct sockaddr *)&server, sizeof(server));
    if(resp == SOCKET_ERROR){
        printf("Error al asociar puerto e ip al socket\n");
        closesocket(conexion_servidor); WSACleanup();
        getchar();return WSAGetLastError();}
    if(listen(conexion_servidor, 2)==SOCKET_ERROR){
        printf("Error al habilitar conexiones entrantes\n");
        closesocket(conexion_servidor); WSACleanup();
        getchar();return WSAGetLastError();}
    printf("Esperando conexiones entrantes... \n");
    stsize=sizeof(struct sockaddr);
    conexion_cliente=accept (conexion_servidor, (struct sockaddr
*)&client,&stsize);
    if(conexion_cliente==INVALID_SOCKET) {
        printf("Error al aceptar conexión entrante\n");
        closesocket(conexion_servidor); WSACleanup();
        getchar();return WSAGetLastError();
    }
    printf("Conexión entrante desde: %s - Dirección Mini-DK2\n",
inet_ntoa(client.sin_addr));
    closesocket(conexion_servidor);
    strcpy(SendBuff, "Hola Cliente, hablando el servidor");
    printf("Enviando Mensaje... \n");
    send (conexion_cliente, SendBuff, sizeof(SendBuff), 0);
    printf("Datos enviados: %s \n", SendBuff);
    printf("Recibiendo Mensaje... \n");
    recv (conexion_cliente, RecvBuff, sizeof(RecvBuff), 0);
    printf("Datos recibidos: %s \n", RecvBuff);
    getchar();
    closesocket(conexion_cliente);
    WSACleanup();
    return (EXIT_SUCCESS);
}
```

## Anexo 2. Código aplicación cliente en SO Windows

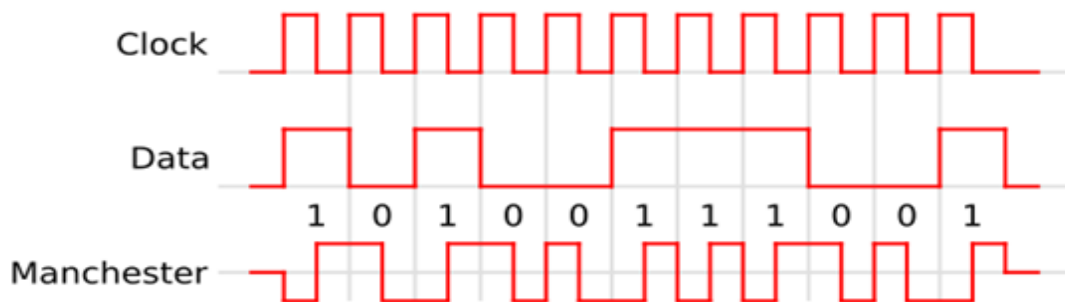
```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winsock.h>
char SendBuff[512], RecvBuff[512];
SOCKET cliente;
int main(int argc, char *argv[])
{
    WSADATA wsaData;
    printf("Quiero establecer una conexion con LPC1768 con IP -
192.168.1.170\n");
    int resp=WSAStartup(MAKEWORD(1,0), &wsaData);
    if(resp){
        printf("Error al inicializar socket\n");
        getchar();return -1;}
    SOCKADDR_IN target;
    target.sin_family = AF_INET;
    target.sin_port = htons (90);
    target.sin_addr.s_addr = inet_addr ("192.168.1.170");
    printf("Se ha asignado el puerto y la IP correctamente\n");
    cliente = socket (AF_INET, SOCK_STREAM, 0);
    if (cliente == INVALID_SOCKET)
    {
        printf("Error al crear socket\n");
        getchar();WSACleanup();return WSAGetLastError();}
    printf("Se ha creado el socket\n");
    if (connect(cliente, (SOCKADDR *)&target, sizeof(target)) ==
SOCKET_ERROR)
    {
        printf("Fallo al conectarse con el servidor\n");
        closesocket(cliente);
        WSACleanup();getchar();return WSAGetLastError();}
    printf("Se ha conectado con el servidor\n");

    printf("Conectado con %s \n",inet_ntoa(target.sin_addr));
    strcpy(SendBuff,"Hola LPC1768 - Hablando el cliente");
    printf("Enviando Mensaje... \n");
    send(cliente,SendBuff,sizeof(SendBuff),0);
    printf("Datos enviados: %s \n", SendBuff);
    printf("Recibiendo Mensaje... \n");
    recv(cliente,RecvBuff, sizeof(RecvBuff), 0);
    printf("Datos recibidos: %s \n", RecvBuff);
    getchar();
    closesocket(cliente);
    WSACleanup();
    return EXIT_SUCCESS;
}
```

### Anexo 3. Codificación Manchester

La **codificación Manchester** es un método de codificación eléctrica de una señal binaria en el que en cada tiempo de bit hay una transición entre dos niveles de señal. En cada bit se puede obtener la señal de reloj, lo que provoca la posibilidad de una sincronización del flujo de datos, por lo que a esta codificación se la llama autosincronizada.

En la codificación Manchester las señales de datos y de reloj, se juntan en una sola que autosincroniza el flujo de datos, si en algún punto de la transmisión se producen retardos. Cada bit que se codifica posee una transición en la mitad del periodo de ese bit, una transición de negativo a positivo representa un “1” y una transición de positivo a negativo representa un “0”. En la Figura 38 podemos observar un ejemplo de la codificación Manchester.



*Figura 38. Ejemplo de codificación Manchester*

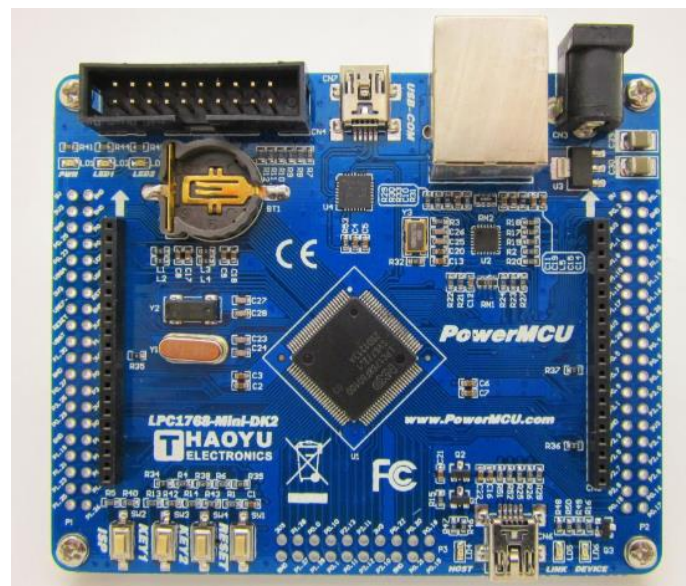
Una de las ventajas de esta codificación es que asegura que la componente continua de las señales es cero si se emplean valores positivos y negativos para representar los niveles de la señal, haciendo más sencilla la regeneración de la señal.

Una de las principales desventajas de esta codificación es que consume el doble de ancho de banda que las comunicaciones asíncronas y el espectro de la señal es más ancho. Los sistemas modernos de comunicación utilizan protocolos de codificación que poseen los mismos objetivos, pero con consumo menor de ancho de banda.

## Anexo 4. LPC1768

Microcontrolador basado en Cortex – M3, que se encuentra en la placa de desarrollo Mini – DK2. Las características del LPC1768 son las siguientes:

- Frecuencia de la CPU hasta 100MHz, Hasta 512 kB de memoria flash, Hasta 64 kB de memoria de datos.
- El LPC17xx posee los siguientes periféricos:
  - Ethernet MAC.
  - Interfaz USB.
  - Un controlador de DMA de 8 canales.
  - 4 UARTs, 2 canales CAN, 2 controladores SSP y un interfaz SPI, 3 Interfaces I2C, interfaz I2S
  - Un ADC de 12 bits y 8 canales de entrada y un DAC de 10 bit
  - 4 Timers de propósito general
  - 6 salidas PWM
  - Un reloj de tiempo real.
  - Hasta 70 pines de entrada/salida de propósito general organizados en 5 puertos
- Entorno de programación basado en C/C++
- Se utiliza el compilador ARM RealView



*Figura 39. Placa de desarrollo Mini – DK2 con microcontrolador LPC1768 basado en Cortex – M3*



## Anexo 5. Funciones utilizadas en el desarrollo del proyecto

Durante el desarrollo del proyecto, se han utilizado una gran variedad de funciones de distintos proyectos base, que en su conjunto han sido el motor del desarrollo del sistema operativo. Aunque durante la memoria se han ido explicando estas funciones, vamos a dejarlas explicadas todas ellas en este apartado del anexo.

- ***Void \_\_svc(1) open\_uart(uint8\_t UARTn, uint32\_t baudrate, osThreadId ID):*** función que configura el periférico de la comunicación serie, poniendo los valores adecuados en los registros pertinentes. También realiza la función de reserva de memoria para la estructura de control de acceso. Argumentos:
  - ***uint8\_t UARTn:*** se indica la UART que se quiere configurar, a través de las macros UART0, UART1, UART2, UART3.
  - ***uint32\_t baudrate:*** se indica la velocidad de transmisión de la UART.
  - ***osThreadId ID:*** se le debe pasar la ID del hilo en el que se llama la función.
- ***static int uartn\_set\_baudrate(uint8\_t UARTn, unsigned int baudrate):*** función que calcula los valores adecuados que se deben indicar en los registros de la UART para establecer la velocidad de transmisión indicada con el parámetro *baudrate*. Esta función se llama dentro de la función *open\_uart*. Argumentos:
  - ***uint8\_t UARTn:*** se indica la UART que se quiere configurar, a través del argumento *uint8\_t UARTn* de la función *open\_uart*.
  - ***unsigned int baudrate:*** la velocidad de transmisión desde la cual se calculan los valores adecuados para los registros de la UART. Se indica a través del argumento *uint32\_t baudrate* de la función *open\_uart*.
- ***void \_\_svc(2) write\_uart(uint8\_t UARTn, char \*datos, osThreadId ID):*** función que inicia la transmisión a través de la UART indicada. Introduce el primer dato para activar el flag de interrupción. Argumentos:
  - ***uint8\_t UARTn:*** se indica la UART por la que se quiere iniciar la transmisión.
  - ***char \*datos:*** indica el mensaje que se quiere transmitir.
  - ***osThreadId ID:*** se debe pasar como argumento la ID del hilo del que se llama la función.
- ***void \_\_svc(3) read\_uart(uint8\_t UARTn, char \*datos\_rx, osThreadId ID):*** función cuyo objetivo es la lectura de la información que llega al microcontrolador a través de la UART. Argumentos:
  - ***uint8\_t UARTn:*** se indica la UART por la que se quiere leer la información entrante.
  - ***char \*datos\_rx:*** argumento en el que se guarda la información recibida por la UART.

- ***osThreadId ID***: argumento al que se le debe pasar la ID del hilo desde el cual se llama a la función *read\_uart*.
- ***void UART0\_IRQHandler(void)***: función handler cuya función es recibir o transmitir la información a través de la UART0.
- ***void UART1\_IRQHandler(void)***: función handler cuya función es recibir o transmitir la información a través de la UART1.
- ***void UART2\_IRQHandler(void)***: función handler cuya función es recibir o transmitir la información a través de la UART2.
- ***void UART3\_IRQHandler(void)***: función handler cuya función es recibir o transmitir la información a través de la UART3.
- ***void \_\_svc(19) close\_uart(osThreadId ID)***: función cuyo objetivo es la liberación de memoria de la estructura de control de acceso que se reservó con la función *open\_uart*. Argumentos:
  - ***osThreadId ID***: se debe pasar como argumento la ID del hilo desde el cual se llama la función.
- ***P\_TETH \_\_svc(4) open\_ethernet(osThreadId ID)***: función encargada de la configuración de los registros pertinentes para poner en marcha el periférico Ethernet del microcontrolador. Además, se reserva memoria para las estructuras de configuración de la conexión y la estructura de control de acceso. La función devuelve la dirección de memoria en la que se ha reservado para la estructura de configuración. Argumentos:
  - ***osThreadId ID***: se debe pasar como argumento la ID del hilo desde el cual se llama la función *open\_ethernet*.
- ***void TCPLowLevelInit(void)***: función que inicializa el controlador LAN, resetea los flags e inicia los timer. Se utiliza dentro de la función *open\_ethernet*.
- ***void \_\_svc(5) write\_ethernet(char \*data\_tx, osThreadId ID)***: función encargada de transmitir información a través de la comunicación Ethernet, una vez establecida la comunicación. Argumentos:
  - ***char \*data\_tx***: argumento por el cual se indica la información que se quiere transmitir.
  - ***osThreadId ID***: argumento en el que se debe indicar la ID del hilo desde el cual se ha llamado a la función *write\_ethernet*.
- ***void TCPReleaseRxBuffer (void)***: función que indica que los datos en el búfer de recepción han sido leídos. El stack utiliza el búfer de recepción para los siguientes datos una vez indicado este hecho. Función utilizada dentro de la función *write\_ethernet* y *read\_ethernet*.
- ***void TCPTransmitTxBuffer (void)***: función que transmite la información almacenada en el búfer de transmisión. Función utilizada dentro de la función *write\_ethernet*.

- ***void TCPClose (void)***: función que cierra la conexión TCP establecida. Función utilizada dentro de la función *write\_ethernet*.
- ***void \_\_svc(6) read\_ethernet(unsigned char \*data\_rx, osThreadId ID)***: función cuyo objetivo es leer la información que llega por la comunicación Ethernet. Argumentos:
  - ***unsigned char \*data\_rx***: argumento a través del cual se llega a guardar la información recibida.
  - ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función *read\_ethernet*.
- ***void \_\_svc(8) close\_ethernet(osThreadId ID, struct OS\_TETH \*cls)***: función encargada de liberar la memoria reservada dentro de la función *open\_ethernet* y finalizar la comunicación a través de Ethernet. Argumentos:
  - ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
  - ***struct OS\_TETH \*cls***: se debe pasar la estructura a la que se le reservó memoria con la función *open\_ethernet*, para liberar esa memoria.
- ***void \_\_svc(7) connect\_ethernet(struct OS\_TETH \*add, osThreadId ID, uint8\_t num\_con)***: función encargada de establecer la conexión con los parámetros indicados en la estructura de configuración de conexión. Argumentos:
  - ***struct OS\_TETH \*add***: argumento por el que se pasa la estructura de configuración de la conexión.
  - ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
  - ***uint8\_t num\_con***: argumento que el usuario debe rellenar, si en el caso de que existan más de una estructura de configuración de la conexión, este parámetro hará referencia a la estructura en concreto.
- ***void add\_element (void)***: función que añade una nueva estructura a la lista de estructuras de configuración de la conexión. Esta función se llama dentro de la función *connect\_ethernet*.
- ***void check\_struct (osThreadId ID, uint8\_t num\_con)***: función que realiza la supervisión de la estructura de configuración de la conexión que se debe ejecutar en un momento determinado. Esta función se llama dentro de la función *connect\_ethernet*. Argumentos:
  - ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
  - ***uint8\_t num\_con***: parámetro que indica a la estructura en concreto a la que se refiere el usuario en el caso de que haya más de una estructura de conexión.

- ***void liberar(struct OS\_TETH \*erase)***: función encargada de liberar la memoria reservada de las estructuras de conexión. Esta función se llama dentro de la función *close\_ethernet*. Argumentos:
  - ***struct OS\_TETH \*erase***: estructura en cuestión a la que se quiere liberar la memoria.
- ***P\_LCD \_\_svc(9) open\_lcd(uint16\_t color,uint8\_t rotation, osThreadId ID)***: función cuyo objetivo es inicializar los registros adecuados para empezar a trabajar con la pantalla LCD del microcontrolador. Además, reserva la memoria para la estructura de control de acceso y también para la estructura en la que se indicarán los parámetros de lo que se quiera representar en la pantalla. Argumentos:
  - ***uint16\_t color***: parámetro que indica el color de la pantalla que se configurará al iniciar la pantalla. Para indicar el color se pueden utilizar las macros definidas: **BLACK, BLUE, RED, GREEN, CYAN, MAGENTA, YELLOW, WHITE**.
  - ***uint8\_t rotation***: parámetro que indica la orientación que se configurará al iniciar la pantalla LCD. Hay cuatro opciones, la opción que se debe indicar por defecto es el valor 3.
  - ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
- ***void lcdInitDisplay(void)***: Inicializa el controlador LCD ILI9325. Función utilizada dentro de la función *open\_lcd*.
- ***void setRotation(uint8\_t dir)***: esta función establece la dirección de escritura. La variable *dir* puede valer 0,1,2 ó 3. Por defecto se establece *dir* = 3 para una escritura de izquierda a derecha y de arriba hasta abajo. Función utilizada dentro de la función *open\_lcd*.
- ***void \_\_svc(10) write\_lcd(P\_LCD lcd, osThreadId ID)***: función que se encarga de representar lo que el usuario indique por medio de la estructura *P\_LCD lcd*. Argumentos:
  - ***P\_LCD lcd***: el argumento es la estructura en la cual el usuario debe indicar lo que se requiere representar en la pantalla.
  - ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
- ***void drawChar (uint16\_t x, uint16\_t y, char c, uint16\_t fColor, uint16\_t bColor, uint8\_t s)***: función que representa un carácter en la pantalla LCD. Esta función se utiliza en la función *write\_lcd*. Argumentos:
  - ***uint16\_t x***: coordenada x de la pantalla LCD.
  - ***uint16\_t y***: coordenada y de la pantalla LCD.
  - ***char c***: carácter en cuestión que se quiere representar.

- **uint16\_t fColor**: color con el que se quiere representar el carácter.
- **uint16\_t bColor**: color de fondo del carácter.
- **uint8\_t s**: tamaño con el que se quiere representar el carácter. Las opciones disponibles son **SMALL, MEDIUM, LARGE**.
- **void drawString(uint16\_t x, uint16\_t y, char \*c, uint16\_t fColor, uint16\_t bColor, uint8\_t s)**: función que representa una cadena en la pantalla LCD. Esta función se utiliza en la función *write\_lcd*. Argumentos:
  - **uint16\_t x**: coordenada x de la pantalla LCD.
  - **uint16\_t y**: coordenada y de la pantalla LCD.
  - **char \*c**: cadena en cuestión que se quiere representar.
  - **uint16\_t fColor**: color con el que se quiere representar la cadena.
  - **uint16\_t bColor**: color de fondo de la cadena.
  - **uint8\_t s**: tamaño con el que se quiere representar el carácter. Las opciones disponibles son **SMALL, MEDIUM, LARGE**.
- **void drawCircle(uint16\_t x0, uint16\_t y0, uint16\_t r, uint16\_t color)**: función que representa un círculo en la pantalla LCD. Esta función se utiliza en la función *write\_lcd*. Argumentos:
  - **uint16\_t x0**: coordenada x de la pantalla LCD.
  - **uint16\_t y0**: coordenada y de la pantalla LCD.
  - **uint16\_t r**: el valor del radio del círculo.
  - **uint16\_t color**: el color con el que se quiere representar el círculo en la pantalla.
- **void fillCircle(uint16\_t x0, uint16\_t y0, uint16\_t r, uint16\_t color)**: función que rellena el círculo de un determinado color. Esta función se utiliza en la función *write\_lcd*. Argumentos:
  - **uint16\_t x0**: coordenada x de la pantalla LCD.
  - **uint16\_t y0**: coordenada y de la pantalla LCD.
  - **uint16\_t r**: el valor del radio del círculo.
  - **uint16\_t color**: el color con el que se quiere rellenar el círculo en la pantalla.
- **void fillScreen(uint16\_t color)**: función que rellena la pantalla de un color determinado. Esta función se utiliza en la función *write\_lcd* y *open\_lcd*. Argumentos:
  - **uint16\_t color**: el color con el que se quiere rellenar la pantalla.
- **void drawRect(uint16\_t x, uint16\_t y, uint16\_t w, uint16\_t h, uint16\_t color)**: función que representa en pantalla un rectángulo. Esta función es utilizada dentro de la función *write\_lcd*. Argumentos:
  - **uint16\_t x**: coordenada x de la pantalla LCD.
  - **uint16\_t y**: coordenada y de la pantalla LCD.

- **uint16\_t w**: valor del ancho del rectángulo.
- **uint16\_t h**: valor del alto del rectángulo.
- **uint16\_t color**: color del rectángulo en pantalla.
- **void fillRect(uint16\_t x0, uint16\_t y0, uint16\_t w, uint16\_t h, uint16\_t color)**: función que rellena un rectángulo de un determinado color. Esta función es utilizada dentro de la función *write\_lcd*. Argumentos:
  - **uint16\_t x0**: coordenada x de la pantalla LCD.
  - **uint16\_t y0**: coordenada y de la pantalla LCD.
  - **uint16\_t w**: valor del ancho del rectángulo.
  - **uint16\_t h**: valor del alto del rectángulo.
  - **uint16\_t color**: color con el que se quiere rellenar el rectángulo en pantalla.
- **void drawFastLine(uint16\_t x0, uint16\_t y0, uint16\_t length, uint16\_t color, uint8\_t rotflag)**: función que representa una línea horizontal o vertical en pantalla. Esta función es utilizada dentro de la función *write\_lcd*. Argumentos:
  - **uint16\_t x0**: coordenada x de la pantalla LCD.
  - **uint16\_t y0**: coordenada y de la pantalla LCD.
  - **uint16\_t length**: longitud de la línea en cuestión.
  - **uint16\_t color**: color de la línea que se va a representar.
  - **uint8\_t rotflag**: parámetro que indica la orientación de la línea, horizontal o vertical. Un 0 indica horizontal y un 1 indica vertical.
- **void drawLine(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, uint16\_t color)**: función que representa una recta en pantalla. Esta función es utilizada dentro de la función *write\_lcd*. Argumentos:
  - **int16\_t x0**: coordenada x inicial de la pantalla LCD.
  - **int16\_t y0**: coordenada y inicial de la pantalla LCD.
  - **int16\_t x1**: coordenada x final de la pantalla LCD.
  - **int16\_t y1**: coordenada y final de la pantalla LCD.
  - **uint16\_t color**: color de la recta que se va a representar en pantalla.
- **void drawPixel(uint16\_t x, uint16\_t y, uint16\_t color)**: función que representa un píxel en pantalla LCD. Esta función es utilizada dentro de la función *write\_lcd*. Argumentos:
  - **uint16\_t x**: coordenada x de la pantalla LCD.
  - **uint16\_t y**: coordenada y de la pantalla LCD.
  - **uint16\_t color**: color con el que se va a representar el píxel.
- **void \_\_svc(11) close\_lcd(osThreadId ID)**: función encargada de liberar la memoria reservada en la función *open\_lcd* para la estructura de control de acceso y para la estructura de configuración de la representación en pantalla. Argumentos:

- ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
- ***void \_\_svc(12) open\_touch(osThreadId ID)***: función que inicializa los registros necesarios para el correcto funcionamiento de la pantalla LCD y además se encarga de la calibración de la pantalla táctil. También es la encargada de reservar memoria para la estructura de control de acceso. Argumentos:
  - ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
- ***void TP\_Init(void)***: inicialización del puerto SPI del ADS7843
- ***void LCD\_Initializtion(void)***: función que inicializa el controlador TFT de la pantalla. Esta función se utiliza dentro de la función *open\_touch*.
- ***void TouchPanel\_Calibrate(void)***: función encargada de calibrar la pantalla LCD para su uso como pantalla táctil. Esta función se utiliza dentro de la función *open\_touch*.
- ***void \_\_svc(13) write\_touch(osThreadId ID)***: función que representa en pantalla la entrada táctil sobre ésta. Argumentos:
  - ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
- ***uint8\_t getDisplayPoint(Coordinate \* displayPtr, Coordinate \* screenPtr, Matrix \* matrixPtr)***: función que convierte la entrada táctil sobre la pantalla en un valor de las coordenadas de la pantalla LCD. Esta función es utilizada dentro de la función *write\_touch*.
- ***void TP\_DrawPoint(uint16\_t Xpos, uint16\_t Ypos)***: función que dibuja los puntos que se han tocado de la pantalla LCD. Esta función se utiliza dentro de la función *write\_touch*. Argumentos:
  - ***uint16\_t Xpos***: coordenada X de la pantalla LCD.
  - ***uint16\_t Ypos***: coordenada Y de la pantalla LCD.
- ***void \_\_svc(18) close\_touch(osThreadId ID)***: función que libera la memoria reservada para la estructura de control de acceso que se reservó con la función *open\_touch*.
- ***void \_\_svc (14) open\_file(osThreadId ID)***: función que pone en funcionamiento el manejo de ficheros con el microcontrolador, al inicializar los registros pertinentes y reservar memoria para la estructura de control de acceso. Argumentos:
  - ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
- ***void MSD\_SPI\_Configuration(void)***: función que pone en marcha la configuración del puerto SPI de la tarjeta SD. Función utilizada dentro de la función *open\_file*.

- ***void debug\_frmwrk\_init(void)***: función que inicializa el trabajo de marco de depuración mediante la inicialización del puerto UART.
- ***FRESULT f\_mount (BYTE vol, FATFS \*fs)***: función que monta o desmonta una tarjeta SD. Esta función se utiliza dentro de la función *open\_file* y *close\_file*.

Argumentos:

- ***BYTE vol***: argumento que indica el número de la unidad lógica que va a ser montada o desmontada.
- ***FATFS \*fs***: puntero al nuevo objeto del sistema de archivos (NULL para la opción de desmontar una unidad lógica).
- ***void \_\_svc(15) write\_file(uint8\_t option, char \*text, char \*file, osThreadId ID)***: función que tiene como objetivos la creación de un nuevo archivo y escribir dentro de él, la creación de un directorio, apertura de un archivo ya existente y la escritura posterior en este. Argumentos:

- ***uint8\_t option***: argumento que indica una de las tres opciones posibles para la elección del usuario. Se pueden indicar por las macros creadas como: **NEW\_FILE**, **MKDIR**, **OPEN\_FILE**.
- ***char \*text***: puntero al texto que se quiere escribir.
- ***char \*file***: puntero a la dirección del archivo.
- ***osThreadId ID***: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.

- ***FRESULT f\_open (FIL \*fp, const TCHAR \*path, BYTE mode)***: función que abre o crea un archivo. Esta función es utilizada en las funciones *write\_file* y *read\_file*.

Argumentos:

- ***FIL \*fp***: puntero al objeto archivo en blanco.
- ***const TCHAR \*path***: puntero al nombre del archivo.
- ***BYTE mode***: argumento que indica el modo de acceso al archivo y flags del modo de apertura del archivo.

- ***FRESULT f\_write (FIL \*fp, const void \*buff, UINT btw, UINT \*bw)***: función que escribe en un archivo. Esta función se utiliza en la función *write\_file*.

Argumentos:

- ***FIL \*fp***: puntero al objeto archivo.
- ***const void \*buff***: Puntero a la información que va a ser escrita.
- ***UINT btw***: número de bytes que van a ser escritos.
- ***UINT \*bw***: Puntero al número de Bytes escritos.

- ***FRESULT f\_close (FIL \*fp)***: función que cierra un archivo previamente abierto. Esta función es utilizada en las funciones *write\_file* y *read\_file*. Argumentos:

- ***FIL \*fp***: puntero al archivo que se va a cerrar.

- ***FRESULT f\_mkdir (const TCHAR \*path)***: función cuyo objetivo es la de crear un directorio nuevo. Esta función se utiliza en la función *write\_file*. Argumentos:



- **const TCHAR \*path**: puntero a la ruta del directorio.
- **void \_\_svc(16) read\_file(uint8\_t option, char \*pth, char \*file, osThreadId ID)**: función que tiene como objetivo la lectura de un archivo, la muestra de la memoria de la unidad lógica y la muestra de los ficheros que contiene la unidad lógica. Argumentos:
  - **uint8\_t option**: argumento que tiene el objetivo la elección de las tres opciones antes mencionadas. Para ello se han creado tres macros que se deben utilizar por el usuario para indicar la opción que corresponda: **READ\_FILE, SHOW\_SIZE, SHOW\_FILES**.
  - **char \*pth**: puntero al directorio para escanear los ficheros que están presentes en él. Es utilizada en la opción **SHOW\_FILES**.
  - **char \*file**: puntero al nombre del archivo. Utilizada en la opción **READ\_FILE**.
  - **osThreadId ID**: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
- **FRESULT f\_read(FIL \*fp, void \*buff, UINT btr, UINT \*br)**: función que lee el contenido de un archivo. Función utilizada dentro de la función **read\_file**. Argumentos:
  - **FIL \*fp**: puntero al archivo.
  - **void \*buff**: puntero al búfer de datos.
  - **UINT btr**: número de bytes a leer.
  - **UINT \*br**: puntero al número de bytes leídos.
- **void \_\_svc(17)close\_file(uint8\_t option, const TCHAR \*f\_dir, osThreadId ID)**: función que tiene como objetivo la eliminación de un archivo o directorio, desmontar una unidad lógica o la liberación de la memoria reservada para la estructura de control de acceso. Argumentos:
  - **uint8\_t option**: argumento que indica la opción que ha elegido el usuario de las tres antes mencionadas. Para esto se han creado tres macros: **REMOVE, UNMOUNT y FREE\_MEM**.
  - **const TCHAR \*f\_dir**: puntero a la ruta del archivo o directorio.
  - **osThreadId ID**: argumento al que se debe indicar la ID del hilo desde el cual se ha llamado a la función.
- **FRESULT f\_unlink(const TCHAR \*path)**: función cuyo objetivo es la eliminación de un archivo o de un directorio. Función utilizada en la función **close\_file**. Argumentos:
  - **const TCHAR \*path**: puntero a la ruta del archivo o directorio.
- **FRESULT scan\_files(char\* path)**: función que escanea el directorio en busca de los archivos que en él se encuentran. Esta función es utilizada en la función **read\_file**. Argumentos:

- ***char\* path***: puntero a la ruta del directorio.
- ***int SD\_TotalSize(void)***: función que indica al usuario la cantidad de memoria que posee la tarjeta SD y la cantidad de memoria disponible. Esta función es utilizada en la función *read\_file*.
- ***int \_init\_box (void \*box\_mem, U32 box\_size, U32 blk\_size)***: función que inicializa un grupo de memoria de tamaño de bloque fijo. Cuando se inicializa el grupo de memoria, el kernel de RTX, maneja las solicitudes asignando un bloque de memoria del grupo de memoria. Argumentos:
  - ***void \*box\_mem***: puntero a la dirección de inicio del grupo de memoria.
  - ***U32 box\_size***: número de bytes en el grupo de memoria.
  - ***U32 blk\_size***: número de bytes en cada bloque del grupo.
- ***void \*rt\_alloc\_box (void \*box\_mem)***: función que asigna un bloque de memoria del grupo de memoria que comienza en la dirección *\*box\_mem*. Argumentos:
  - ***void \*box\_mem***: dirección de inicio del grupo de memoria.
- ***int rt\_free\_box (void \*box\_mem, void \*box)***: función que devuelve un bloque de memoria, que se asignó usando *rt\_alloc\_box*, de vuelta al grupo de memoria desde donde se obtuvo. Argumentos:
  - ***void \*box\_mem***: dirección de inicio del grupo de memoria.
  - ***void \*box***: puntero al bloque para liberar.

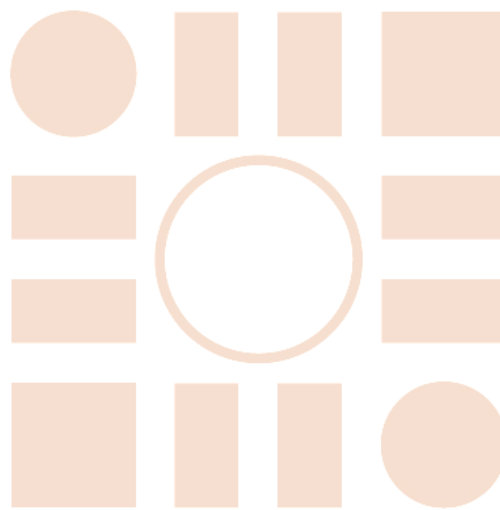
## Bibliografía

- [1] NXP Semiconductors (2016-12-19), “UM10360 – LPC178x/5x User manual”  
[Online]. Available: <https://www.nxp.com/docs/en/user-guide/UM10360.pdf>
- [2] S. S. Prieto, *Sistemas Operativos*. Universidad de Alcalá. Alcalá de Henares, 2004.
- [3] Fco. J. Ceballos Sierra, *C/C++ Curso de programación*. 3ª Edición. Alcalá de Henares: Ra-Ma, 2007.
- [4] ARM Keil. “SVC Functions”. [Online]. Available:  
[http://www.keil.com/support/man/docs/rlarm/rlarm\\_ar\\_svc\\_func.htm](http://www.keil.com/support/man/docs/rlarm/rlarm_ar_svc_func.htm)
- [5] ARM Keil. “RTOS Advantages”. [Online]. Available: [http://www.keil.com/rl-arm/rtx\\_rtosadv.asp](http://www.keil.com/rl-arm/rtx_rtosadv.asp)
- [6] “Cortex – M3 Processor: Input /Output”, class notes for Informática Industrial, Departamento de Automática, Universidad de Alcalá.
- [7] “Operating Systems in Industrial Applications”, class notes for Informática Industrial, Departamento de Automática , Universidad de Alcalá.
- [8] “Protocolo de control de Transmisión” (Sin fecha). En Wikipedia. Recuperado el 10 de febrero de 2019.  
[https://es.wikipedia.org/wiki/Protocolo\\_de\\_control\\_de\\_transmisi%C3%B3n](https://es.wikipedia.org/wiki/Protocolo_de_control_de_transmisi%C3%B3n)
- [9] “Sockets en Windows”, (2011, marzo 20) [Online]. Available:  
[http://www.programacionenc.net/index.php?option=com\\_content&view=article&id=73:sockets-en-windows&catid=37:programacion-cc&Itemid=55&showall=1](http://www.programacionenc.net/index.php?option=com_content&view=article&id=73:sockets-en-windows&catid=37:programacion-cc&Itemid=55&showall=1)
- [10] Samsung Electronics Co., “Samsung SD & MicroSD Card product family”, 2013.  
[Online] Available: <http://www.farnell.com/datasheets/1836582.pdf>
- [11] “Secure Digital” (Sin fecha). En Wikipedia. Recuperado el 24 de abril de 2019.  
[https://es.wikipedia.org/wiki/Secure\\_Digital](https://es.wikipedia.org/wiki/Secure_Digital)
- [12] “Codificación Manchester” (Sin fecha). En Wikipedia. Recuperado el 3 de mayo de 2019. [https://es.wikipedia.org/wiki/Codificaci%C3%B3n\\_Manchester](https://es.wikipedia.org/wiki/Codificaci%C3%B3n_Manchester)



Universidad de Alcalá

Escuela Politécnica Superior



ESCUELA POLITECNICA  
SUPERIOR



Universidad  
de Alcalá