Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Desarrollo de una librería para manejo de la pantalla VM800 con el microcontrolador ESP32

Autor: David García Blasco
Tutor: Manuel Ángel Perales Esteve

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Desarrollo de una librería para manejo de la pantalla VM800 con el microcontrolador ESP32

Autor:
David García Blasco

Tutor:
Manuel Ángel Perales Esteve
Profesor titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo de Fin de Grado: Desarrollo de una librería para manejo de la pantalla VM800 con el microcontrolador ESP32

Autor: David García Blasco

Tutor: Manuel Ángel Perales Esteve

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

Agradecimientos

Querría agradecer a todas esas personas que han estado junto a mi durante todos los años de mi vida, tanto a mi familia como a mis amigos que me han dado su apoyo y ayuda en algunos de los momentos mas difíciles y sin los cuales todo esto no habría sido posible.

En especial, querría agradecer a los profesores que me han ayudado a formarme correctamente como Ingeniero, y previamente me han hecho querer dedicarme a esta profesión, haciendo hincapié en mi tutor de proyecto, Manuel Ángel Perales Esteve, cuya dedicación y paciencia ha hecho que la realización de este proyecto sea para mi tan enriquecedora como gratificante.

Por último, mencionar a todas las grandes personas que he conocido en este último año, realizando mi estancia en el extranjero, cuyo apoyo y compañía han sido imprescindible para mi evolución personal.

David García Blasco

Sevilla-Bologna, 2020

Resumen

Hoy en día, uno de los lenguajes de programación más potente y novedoso es Python, el cual es imprescindible conocer para la correcta formación de un ingeniero. Python es un lenguaje de alto nivel, lo que permite su aplicación como herramienta en una gran variedad de proyectos que serían de una mayor complejidad si se usaran otros lenguajes de menor nivel, como C. Además, posee una amplia gama de funciones preestablecidas, lo que facilita enormemente la programación al ser ésta mas simplificada y estructurada. Otra de las princiales ventajas de este lenguaje es que facilita el manejo de datos y tipos de variables, cosa que simplifica las operaciones matemáticas a realizar y elimina los errores presentes a la hora de trabajar con distintos tipos de variables. Por último, cabe destacar que, al ser un software libre, es posible encontrar numerosos ejemplos y funciones realizadas por otros usuarios, aumentando así el valor de este lenguaje gracias a una numerosa red de usuarios.

En cuanto al ámbito de la electrónica, este lenguaje también puede ser usado para programar microcontroladores. A través de su variante, Micropython, la programación de microcontroladores se vuelve mas sencilla y potente con respecto a anteriores lenguajes de programación. Gracias a la gran cantidad de librerías que dispone, sobretodo éstas que se encargan de la comunicación con perifericos, resuelve los principales problemas de bajo nivel y deja al programador solo el diseño de especificaciones. Como consecuencia, la programación resulta ser mas estructurada e interpretable que con otros lenguajes, mediante los que el programador debería realizar todas las configuraciones de bajo nivel, con el aumento de riesgo de errores que ello conlleva.

Teniendo esto presente, el objetivo de este proyecto será crear una librería en Micropython a partir de otra ya disponible en C, facilitada por el profesor, que permite el manejo y la comunicación de un microcontrolador con una pantalla táctil. Gracias a esta librería, el manejo de la pantalla se volverá mas sencillo e intuitivo, permitiendo su uso a personas con menor conocimiento de electrónica y programación que el necesario para la programación mediante la librería de C.

Según estos requerimientos, se ha procedido al diseño de la librería de forma estructurada: comenzando por las funciones mas básicas que permiten el traspase de datos entre el micro y la pantalla, siguiendo por las funciones necesarias para configurar y encender la pantalla, y terminando por una serie de funciones que facilitan la edición y el diseño de objetos gráficos que se deseen dibujar.

Durante la realización de la librería, se han ido realizando una serie de pruebas y controles para garantizar el correcto funcionamiento y cumplimiento de los requerimientos anteriores.

Por último, para poder mostrar el funcionamiento y la utilidad de la librería realizada, se ha diseñado un pequeño ejemplo donde se mostrarán algunas principales funciones que se pueden utilizar.

Abstract

Nowadays, one of the most powerful and novel programming languages is Python, which is essential to know for the correct training of an engineer. Python is a high-level language, which allows its application as a tool in a variety of projects that would be more complex if other lower-level languages were used, such as C. In addition, it has a wide range of pre-established functions, which greatly facilitates programming as it is more simplified and structured. Another of the main advantages of this language is that it facilitates the data management and types of variables, which simplifies the mathematical operations to be carried out and eliminates the errors present when working with different types of variables. Finally, it should be noted that, being free software, it is possible to find numerous examples and functions performed by other users, thus increasing the value of this language thanks to a large network of users.

As for the field of electronics, this language can also be used to program microcontrollers. Through its variant, MicroPython, the programming of microcontrollers becomes simpler and more powerful compared to previous programming languages. Thanks to the large number of libraries available, especially those that are in charge of communication with peripherals, it solves the main low-level problems and leaves the programmer with only the specification design. As a consequence, programming turns out to be more structured and interpretable than with other languages, through which the programmer should carry out all low-level configurations, with the increased risk of errors that this entails.

With this in mind, the objective of this project will be to create a library in MicroPython from another one already available in C, provided by the professor, which allows the management and communication of a microcontroller with a touch screen. Thanks to this library, the touchscreen operation will become easier and more intuitive, allowing its use to people with less knowledge of electronics and programming than necessary for programming through the C library.

According to these requirements, the library has been designed in a structured way: starting with the most basic functions that allow the transfer of data between the micro and the screen, followed by the functions needed to configure and turn on the screen, and ending with a series of functions that facilitate the editing and the graphical designs of objects that you want to draw.

During the realization of the library, a series of tests and controls have been carried out to guarantee the correct operation and compliance with the previous requirements.

Lastly, in order to show how the library works and how useful it is, a small example has been designed to show some of the main functions that can be used.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
I2C	Inter-Integrated Circuit
CS	Chip Select
MOSI	Master Output Slave Input
MISO	Master Intput Slave Output
SCK	Signa lof Clock
PD	Power Doppler
RAM	Random Access Memory
SRAM	Static Random Access Memory
FIFO	First-in First-out
USB	Universal Serial Bus
LCD	Liquid Cristal Display
PCB	Printed Circuit Board
RGB	Red Green Blue
LSB	Least Significant Bit
MSB	Most Significant Bit
Id	Identity
V	Voltio
MHz	Mega-herzio
kB	Kilo Byte
MB	Mega Byte
MIPS	Millions of instructions per second
DMIPS	Decens of millions of instructions per second

1 INTRODUCCIÓN

Este proyecto ha sido realizado para salvar una serie de carencias presentes el lenguaje de programación C, empleado en la librería primaria facilitada por el profesor, algunas de ellas ya mencionadas anteriormente. La principal de ellas es la complejidad de los proyectos a realizar mediante la librería y el gran riesgo de cometer errores que presenta. Esto se debe a que el entorno de lenguaje de programación de Micropython provee de funciones de mayor nivel para tratar temas que en el entorno de lenguaje C deben ser tratadas a bajo nivel, tales como la configuración del reloj, el protocolo de envío de datos, etc. Estas configuraciones en C son poco intuitivas y complejas, por lo que el riesgo de cometer errores difíciles de detectar es elevado.

Para el uso de la librería primaria, se utilizaba un microcontrolador que permitía su programación en C. En este proyecto, se propone el uso de otro microcontrolador que presenta grandes ventajas respecto al anterior. La principal es que contiene mayor memoria, pudiendo así realizar proyectos de mayor complejidad. También permite la conexión a internet a través de wifi y posibilita conexiones bluetooth con dispositivos cercanos, aumentando así la diversidad de proyectos que permite realizar.

Sin embargo, a pesar de las mejoras propuestas, se va a mantener la forma de comunicación entre la pantalla y el micro, mediante un canal de comunicación síncrona en serie (SPI), ya que cambiarla no generaría apenas beneficio y conllevaría un aumento de trabajo innecesario.

Cabe destacar la gran ventaja que supone el uso de librerías a la hora de afrontar este tipo de proyectos. Una librería no es mas que una serie de funciones que, almacenadas en la memoria del microcontrolador, facilitan el uso de una serie de herramientas. Si trataramos de afrontarlos de manera directa, en cada proyecto a realizar deberíamos implementar cada uno de los diversos protocolos de inicialización de la pantalla y de configuración, cosa que resulta altamente tedioso y supone mucho riesgo. Además, mediante el uso de librerías no es necesario grandes conocimientos de programación para el usuario. Al final de esta memoria se redactará el correcto uso de las funciones presentes en la librería, así como todas las herramientas de las que podemos hacer uso a través de ellas.

En esta memoria comenzaremos analizando las características técnicas del hardware utilizados, es decir, microcontrolador y pantalla. Se hará un análisis comparativo con el resto de productos dentro de su gama de productos, que podrían haber sido utilizados para este proyecto. También se hablará del harware utilizado para implementar la comunicación entre microcontrolador y pantalla. Continuaremos con los protocolos de comunicación, explicando las diferentes formas posibles de comunicación, las características generales de la comunicación serie síncrona o SPI, y los protocolos concretos que utiliza esta pantalla. A continuación, detallaremos cada una de las funciones diseñadas en la memoria, divididas según la función que realizan o su nivel. Por último, relataremos las diferentes pruebas de control realizadas durante el proyecto, añadiendo los principales errores que detectaron y su posterior solución.

2 DESCRIPCIÓN HARDWARE

En este capítulo se va desarrollar todo las características técnicas de hardware utilizado en el proyecto, así como la conexión entre le microcontrolador y la pantalla. Además, trataremos de localizar los diferentes productos dentro de la amplia gama de productos existentes, explicando también porque han sido escogidos.

El sistema está compesto por un microcontrolador que se comunica con una pantalla a través de cuatro buses de comunicación, implementado en cuatro líneas separadas, como muestra el siguiente esquema:

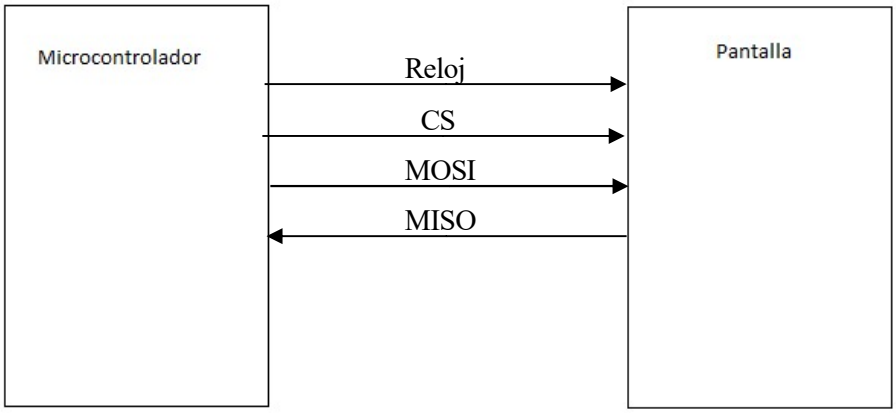

Figura 2.1: Esquema hardware

La descripción de los diferentes buses se incluirá en el siguiente cápitulo.

Incialmente se describirá una breve clasificación de los distintos tipos existentes de cada elemento y las características hardware principales de cada uno de ellos, tanto lo utilizado como lo que no.

2.1. Pantalla

Las pantallas táctiles se clasifican principalmente en tres categorías: resistivas, capacitivas y de onda acústica de superficie.

Las pantallas resistivas están formadas por varias capas de material. Al presionar sobre ella, dos de esas capas entran en contacto, produciendo así un cambio de la corriente eléctrica y detectándose así la pulsación. Son mas baratas, precisas y pueden ser utilizadas con el dedo o con un puntero. Sin embargo, precisan de un mayor grosor y disponen de un 15% menos de brillo que los otros tipos utilizados.

Las pantallas capacitivas están basadas en un sensor capacitivo. Están compuestas por una capa de aislamiento eléctrico, como el cristal, recubierto de un conductor transparente. Como el cuerpo humano es también un conductor eléctrico, al pulsar sobre ella se genera una distorsión del campo electromagnetico de la pantalla, que es medido a través del cambio en la capacitancia. Estas poseen una mejor calidad de la imagen y de la respuesta, e incluso algunas permiten poder usar varios dedos a la vez. Sin embargo, son mas caras y precisan un puntero especial.

Por último, las pantallas de onda acústica de superficie utilizan ondas ultrasónicas que pasan sobre el panel de la pantalla táctil. Al tocar el panel se absorbe parte de la onda, produciendo un cambio con el que se registra la posición. Esta tecnología es la mas avanzada, pero resulta ser la mas frágil de todas.

2.1.1 Características de la pantalla utilizada

En este proyecto hemos propuesto el uso de una pantalla resistiva, debido a que no necesitamos gran calidad de imagen ni gran luminosidad. Además, la ventaja de ser mas económico hace que resulte de mayor interés al no necesitar grandes prestaciones.

Concretamente, hemos utilizado la pantalla VM800B35A de 3.5 pulgadas (320x240 pixeles). Presenta un panel LCD con iluminación led.

Figura 2.2: Pantalla VM800B35A

Fuente: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.tme.eu%2Fhtml%2FES%2Fpantallas-tft-con-controlador-ft800-de-la-marca-bridgetek-ftdi%2Framka_13682_ES_pelny.html&psig=AOvVaw1QH1SUEU-KaPWQhSZE6xw8&ust=1582965355765000&source=images&cd=vfe&ved=0CAMQjB1qFwoTCIj5o9Pr8-cCFQAAAAAdAAAAABAE

Para su manejo, la pantalla contiene el controlador FT800 que se utiliza a través de un microcontrolador, con una comunicación SPI entre ellos. Además, permite el uso de un amplificador de audio y un micro altavoz, incluidos en la pantalla. La pantalla puede ser alimentada mediante un cable de alimentación, por el conector

SPI o mediante un puerto USB; en este proyecto será alimentada mediante el conector SPI. Se debe alimentar a 3.3V, pero posee un convertidor por si fuera necesario alimentarlo a 5V.

2.2 Microcontrolador

En este capítulo vamos a comparar las características del microcontrolador propuesto en este proyecto, ESP32 del fabricante Espressif Systems, con las del microcontrolador que se utilizaba anteriormente con la librería en lenguaje de programación C, MSP430 G2 del fabricante Texas Instruments.

El microcontrolador previamente utilizado contiene un microprocesador de 16 bits con un solo núcleo que rinde a 16 MIPS (Millions of Instructions Per Second), trabajando a una velocidad de 16Mhz. En cambio, el nuevo microprocontrolador presenta un microprocesador de 32 bits, que opera entre 160 y 240 Mhz rindiendo a 600 DMIPS (Decens of Millions of Instructions Per Second). Además, prensenta un co-procesador de ultra baja energía. Podemos comprobar como con el producto propuesto obtenemos una ventaja de prestaciones, ya que el nuevo microprocesador es mucho mas rápido y potente que el antiguo.

En cuanto a la memoria, el microcontrolador de Texas Instruments contiene 512 bytes de memoria SRAM (Static Random Access Memory) y 16kB de memoria FLASH, mientras que el microcontrolador propuesto presenta una memoria SRAM de 512KB y 4MB de memoria FLASH. Podemos observar como, mediante la incorporación de este nuevo microcontrolador, la capacidad de memoria aumenta significativamente tanto en la memoria RAM como en la FLASH.

Con respecto a las conexiones con otros perifericos, el microcontrolador inicial dispone de interfaces de conexión mediante UART (Universal Asynchronous Receiver-Transmitter), I2C (Inter-Integrated Circuit) y SPI (Serial Peripheral Interface); mientras que el nuevo microcontrolador, a parte de posibilitar dichas conexiones, también permite comunicación mediante Wifi y BLUETOOTH.

Tras estas comparaciones, podemos concluir con que el nuevo microcontrolador mejora notablemente las prestaciones obtenidas con respecto al antiguo; tanto en memoria, como en velocidad del microprocesador y posibilidades de comunicación con los perifericos.

A continuación, aparece una imagen de ambos microcontroladores, el nuevo a la derecha y el antiguo a la izquierda:

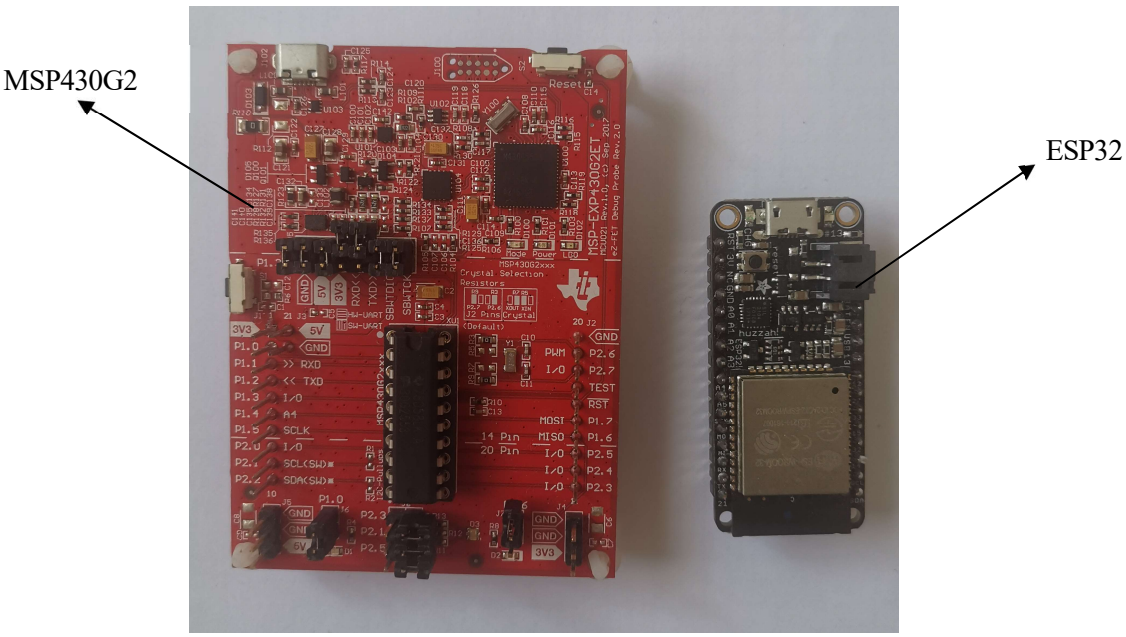

Figura 2.3 : Comparación entre microcontroladores

(antiguo a la izquierda y nuevo a la derecha)

En ella podemos apreciar también la diferencia de tamaño, por lo que al cambiar al microcontrolador nuevo hemos también obtenido ventajas al ser mucho mas reducido que el antiguo.

2.3 Conexión pantalla-microcontrolador

Para este proyecto, hay que diseñar un circuito impreso para fijar las conexiones y mejorar la comunicación. Como primer paso, se han empleado cables de protoboar junto con una protoboard para poder fijar las conexiones de la mejor manera posible. Tras esto, se debe diseñar realizar la pcb. Utilizando una protoboard, y como se verá más adelante, las conexiones no son tan buenas como deberían, lo que repercute en la selección de baudios para la comunicación SPI.

3 PROTOCOLOS DE COMUNICACIÓN

Como se ha mencionado anteriormente, en nuestra propuesta de proyecto no se va a modificar el canal de comunicación empleado en la librería primaria, ya que la pantalla exige comunicación SPI y cambiarla sería mucho mas laborioso y no generaría beneficio alguno.

Además, este protocolo de comunicación es muy adecuado cuando se tiene un dispositivo que debe realizar claramente el papel de maestro, el microcontrolador, y otro que debe solo obedecer, realizando el papel de esclavo; la pantalla.

Debido a que la parte más problemática y mas laboriosa de este proyecto resulta ser la comunicación SPI, se le ha dedicado un capítulo completo para poder explicarla con profundidad suficiente. Esta dificultad deriva de que la pantalla, según viene explicado en la guía de programación del FT800, necesita unos protocolos de iniciación y de comunición sin los cuales no se permite interactuar con ella.

Comenzaremos hablando de la comunicación SPI propiamente dicha y posteriormente hablaremos de los protocolos anteriormente mencionados.

3.1 Comunicación SPI

La comunicación SPI (Serial Peripheral Interface) se basa principalmente en una interacción entre uno o varios maestros, que dirige la interactuación, y uno o varios esclavos. En nuestro proyecto el micro hace las funciones del único maestro y la pantalla hace las del único esclavo, como se muestra en la figura:

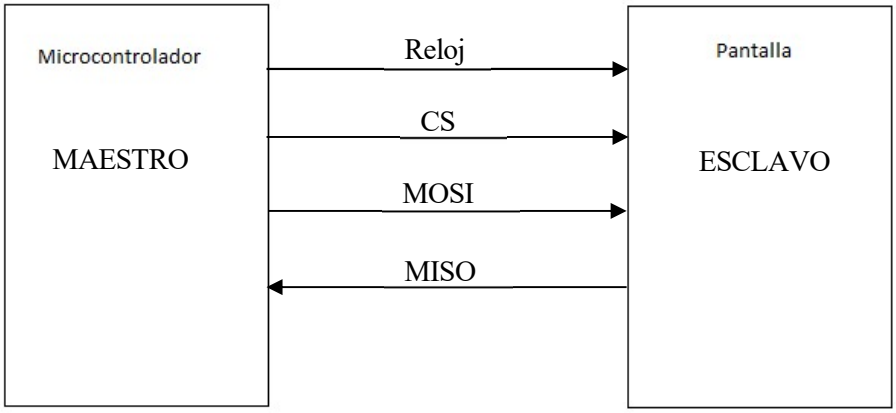

Figura 3.1: Esquema de los componentes de la comunicación

Se trata de una comunicación síncrona, ya que contiene un canal dedicado a una función del reloj, controlada por el maestro, que marca los tiempos de envio y de espera para ordenar la comunicación.

El maestro también controla el bus CS (Chip Select), a través del cual selecciona a nivel bajo al esclavo con el que quiere comunicar. De esta forma, el maestro tiene el poder de iniciar la comunicación con la pantalla.

Por último, esta forma de comunicación presenta dos buses de datos: MOSI (Master Output Slave Input), a través del cual el maestro envía datos al esclavo, y MISO (Master Input Slave Output), mediante el que el esclavo envía datos al maestro.

Todos los buses explicados son unidireccionales, siendo su dirección reflejada en la dirección de las flechas de la imagen anterior.

Además, se debe resaltar que la comunicación SPI es bidireccional, es decir, siempre que el maestro requiera un envio de datos desde el esclavo, paralelamente debe enviarle datos. Dicho de otro modo, tanto maestro como esclavo leen y escriben simultamenamente.

3.1.1 Características principales

A continuación, vamos a enumerar las principales características de la comunicación SPI, concretando en el caso

particular de este proyecto.

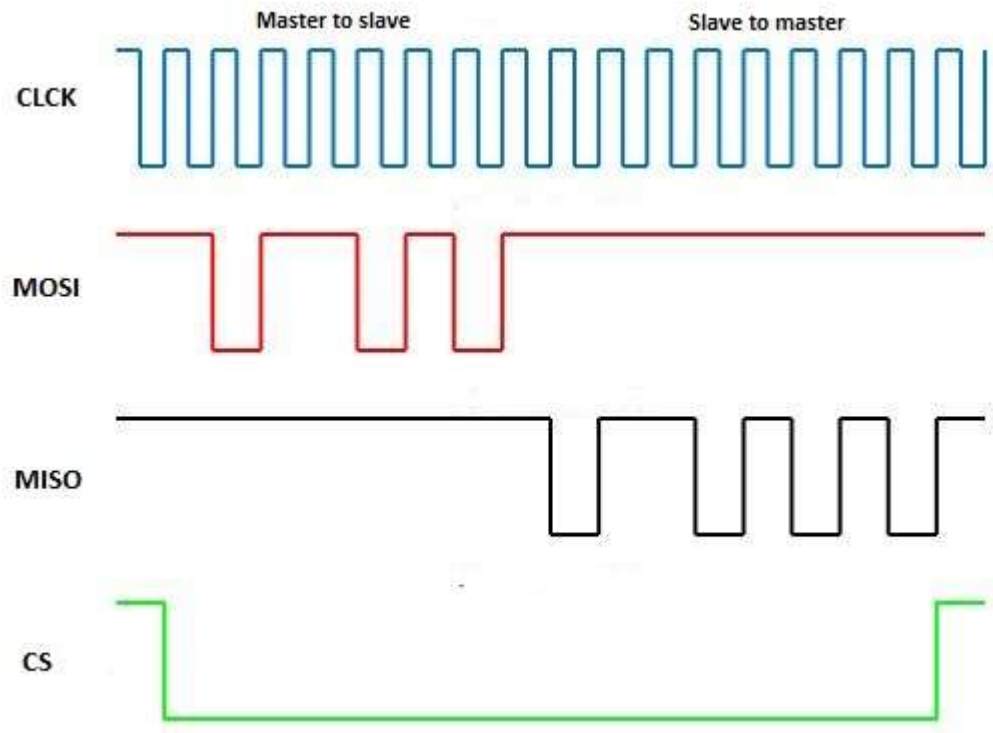

Figura 3.2: Ejemplo pulsos comunicación SPI

La imagen anterior muestra un ejemplo de los distintos pulsos obtenidos en cada uno de los buses empleados. Nos apoyaremos en ella para explicar bien el procedimiento de la comunicación SPI.

La primera línea de pulsos es el reloj, que es controlado por el maestro y se activa solo cuando es necesaria la comunicación.

Al iniciar la comunicación, como se puede observar, el maestro pone el bus Chip Select CS a nivel bajo, y restaura su nivel alto al acabarla.

Una vez modificado el CS, si se desea enviar datos al esclavo, el maestro debe enviar los diferentes bits a través del bus Master Output Slave Input MOSI, tomando como referencia los pulsos de reloj. En cambio, si se quisiera recibir datos desde el esclavo, éste debe mandar los pulsos a través del bus Master Input Slave Output MISO.

La longitud de cada dato puede elegirse en 7 o 8, aunque en este proyecto se ha optado por 8. Además, se puede elegir si se manda del bit más significativo al menos significativo (Most Significant Bit MSB) o al revés (Least Significant Bit LSB). En este proyecto se ha optado por MSB.

Debe haber un canal MISO por cada esclavo, por lo que en este proyecto solo habrá uno.

Por último, se debe elegir una velocidad de transmisión adecuada para que la pantalla lea los datos adecuadamente. Debido al hardware empleado para implementar los buses, mediante protoboard, no podemos elegir una velocidad demasiado grande, ya que, al aumentar la velocidad, las interferencias generadas afectarían en mayor medida a la comunicación.

3.1.2 Protocolo del SPI

Como hemos dicho anteriormente, el protocolo de comunicación es muy riguroso y sin él no es posible la comunicación. Vamos a explicar dicho protocolo desde el maestro y del esclavo.

- Maestro

El maestro es quien inicia la comunicación y quien decide con que esclavo establecerla:

1. Activa el CS del esclavo para seleccionarlo, en nuestro caso poniendo el pin a 0.
2. Envía el dato y lee el que recibe si interesara.
3. Espera a que termine de enviarse.
4. Desactiva el CS del esclavo para cortar la comunicación.

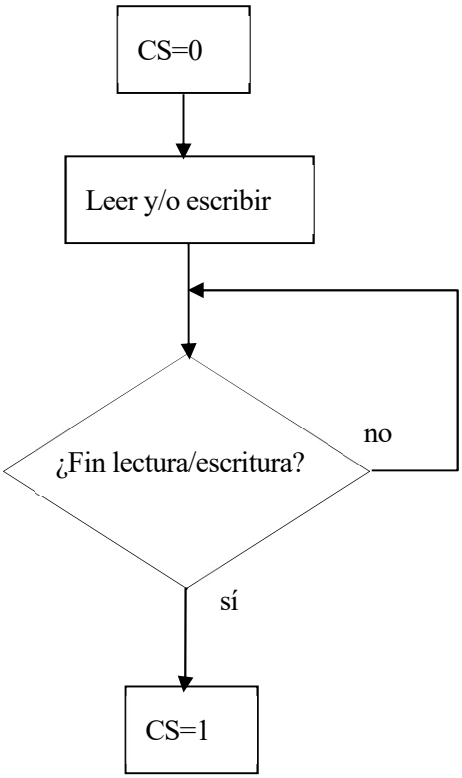

Figura 3.3: Esquema protocolo de comunicación desde el maestro

Este protocolo se repetirá cada vez que debamos escribir o leer algo en la pantalla.

- Esclavo

Como el esclavo no empieza la comunicación, hacemos que se active mediante interrupción. El esclavo permanece inactivo a la espera de recibir la interrupción, que será la llegada de la señal de reloj. Una vez que recibe la señal de reloj, empieza a recibir y a transmitir datos. Al acabar, señala la interrupción.

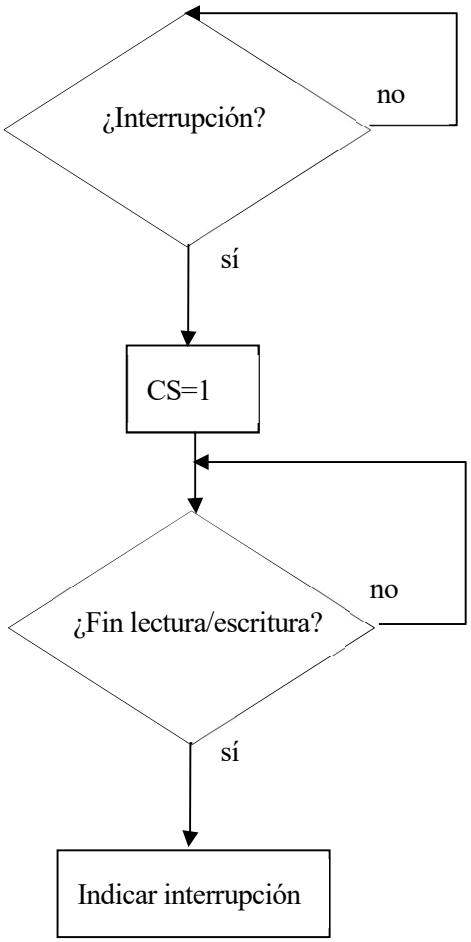

Figura 3.4: Esquema protocolo de comunicación desde esclavo

3.2 Inicialización del SPI

Lo que se va a explicar en esta sección no forma parte de la librería creada, sino que es lo que debemos hacer al inicio de cada programa para configurar el SPI. Antes de todo, debemos importar la librería que Micropython tiene incluida de SPI, de la forma: from machine include SPI.

3.2.1 Construcción objeto SPI

Dado que el lenguaje Python esta orientado a objetos, lo primero que debemos hacer es crear un objeto de clase SPI. Para ello, se usa la función SPI(id, sck=Pin(a), mosi=Pin(b), miso=Pin(c)), donde:

- Id puede ser 1 o 0, cuando queremos que la comunicación tome unos valores preestablecidos, o -1, cuando queremos nosotros modificar manualmente la comunicación, como es nuestro caso.
- Sck, donde se le indica el pin al que está conectado el reloj.
- Mosi, donde se indica el pin al que está conectado el MOSI.
- Miso, donde se indica el pin al que está conectado el MISO.

Las últimas tres configuraciones solo están habilitadas si Id=-1. También, debemos darle un nombre al objeto, en nuestro caso spi, ya que Python reconoce mayúsculas y minúsculas.

3.2.2 Inicialización

Una vez ya hemos creado el objeto SPI, debemos inicializarlo y darle una configuración específica. Ello se consigue con una función ya instalada en Micropython: spi.init(baudrate=b, polarity=p, phase=f), donde:

- Baudrate es la velocidad de comunicación, en baudios. Cuanto mayor sea más rapidez tendrá el dispositivo, pero los errores por ruido también serán más graves.
- Polarity y phase nos viene dado por el esclavo con el que vayamos a comunicarnos, en nuestro caso, ambas son a 0.

3.3 Protocolo para la escritura/lectura en la pantalla

En esta sección vamos a tratar como realizar la correcta escritura o lectura con un registro de la pantalla. La memoria consta de una variedad de registros, que vienen desarrollados en la guía de programación de FT800. Sus direcciones de memoria vienen definidas en otro archivo creado, FT800, donde aparecen como constantes sus direcciones de memoría, comandos preestableidos, configuraciones, etc. Por lo tanto, primero debemos indicar al esclavo con que registro queremos interactuar y luego hacer la operación deseada. Cabe destacar la diferencia que hay entre escribir en un registro o leerlo. Si se quiere escribir, se debe sumar 0x8000000 a la dirección mandada; mientras que si se desea leer no se debe hacer esta operación. Además, los datos a enviar pueden ser de 8, 16 o 32 bits, que deben ser enviados en paquetes de 8 bits.

El orden de actuación es el siguiente:

- Activamos el CS del esclavo para habilitar la comunicación (CS=0).
- Mandamos la dirección de memoria por con una función u otra, según se quiera leer o escribir.
- Realizamos la operación de lectura o escritura que deseemos.
- Desacctivamos el CS del esclavo para finalizar la comunicación.

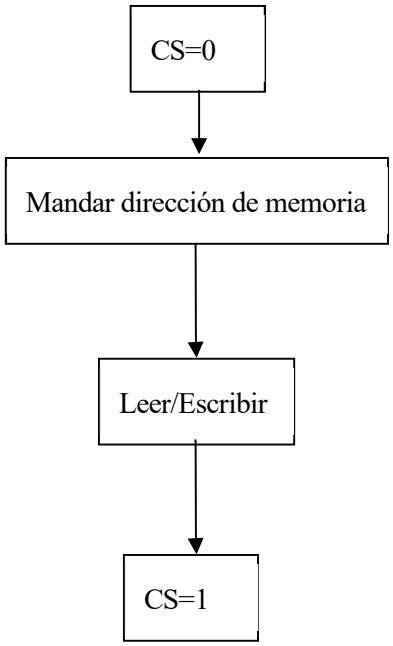

Figura 3.5: Esquema escribir/leer en dirección de memoria

Dichas funciones de enviar la dirección de memoria se desarrollarán adecuadamente en el capitulo posterior a este.

4 LIBRERÍA FT800ESP32

En este capítulo, vamos a desarrollar la librería realizada. Como hemos explicado anteriormente, el uso de librerías es esencial a la hora de afrontar problemas de programación. A través de ellas, se programan partes de código o funciones que se repiten en los proyectos a realizar y, de esta forma, los codigos aparecen en una forma mas organizada y estructurada.

Python y demás lenguajes de programación ya nos facilitan muchas librerías, eficaces para realizar operaciones matemáticas, manejar datos, etc. Sin ellas, todas estas operaciones básicas se deberían realizar en cada proyecto, haciendolos tediosos y probablemente menos eficaces.

Además, muchos perifericos y demás dispositivos como la pantalla utilizada en dicho proyecto requiere de ciertos protocolos de comunicación e iniciación que se deben incluir en cada proyecto a realizar con ellos, por lo que pueden ser incluidos en una librería que los realice directamente.

Así mismo, al ser la librería primaria una librería de prueba, no contenia funciones aptas para todos los objetos gráficos que nos permite la pantalla y que se verán mas adelante; por lo que también se ha aumentado el numero de funciones con respecto a la librería primara al crear las necesarias para los objetos restantes.

Por último, cabe volver a destacar que el empleo de librerías facilita mucho el empleo de estos perifericos, haciendo posible gracias a ellas su uso para un mayor rango de usuarios, no los sobradamente preparados en programación y otros conocimientos.

Nuestra librería consta de una clase, llamada FT_800, compuesta por muchos objetos función, que se pueden a su vez agrupar según su función: comunicación SPI, configuración de pantalla, diseño y táctil y comandos especiales de diseño. Por último, se explicará la función de calibración. Explicaremos cada uno de estos objetos funciones, desarrollando que función cumple, como la realiza y su correcto funcionamiento a alto nivel.

Para poder usar la librería, debemos crear un objeto de la librería, añadiéndole los objetos que va a necesitar modificar o utilizar. Además, toda librería debe contener una primera función _init_ donde se declaren todas las variables a utilizar, todos los pines a modificar y se les adjudique una variable a las entradas del sistema.

Junto a este archivo, la librería requiere de otro archivo también creado, llamado FT800. Este archivo no es mas que una lista de constantes de las que se hace uso en la librería. Estas constantes están sacadas directamente de un manual de programación de la pantalla, también facilitado por el profesor, donde se incluyen configuraciones, direcciones de memoria y demás constantes necesarias con respecto a la pantalla. No es necesario hacer este archivo, ya que podía incluirse en la misma librería, pero se ha realizado para mejorar la limpieza y la organización de ésta.

A continuación, se muestra un esquema de los grupos de funciones según su jerarquía y su utilización:

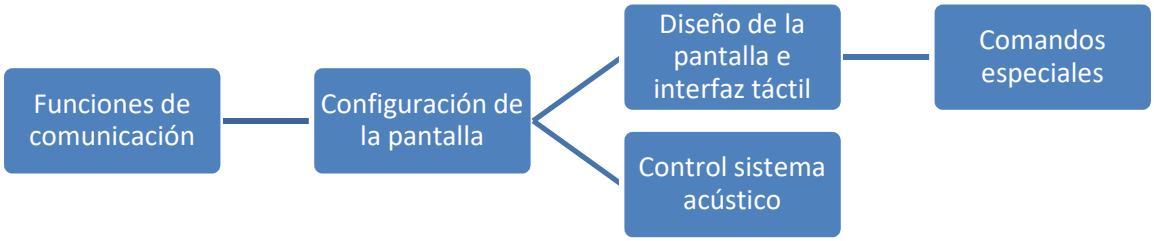

Figura 4.1: Esquema jerarquía de los tipos de funciones

4.1 Funciones de Comunicación SPI

Estas funciones son las más básicas creadas en la librería. De ellas se sirven el resto de funciones para utilizar la comunicación SPI de forma mas sencilla y visual.

4.1.1 LecEsc(data)

Esta función sirve para leer y escribir datos de 8 bits en la pantalla. Es la función mas básica de todas y de mas bajo nivel, de ella harán uso la mayoría de las funciones posteriores. Hace uso de la función spi.read de la clase SPI, ya incluida an Micropython, que recibe por entradas lo que se quiere escribir mediante SPI y el número de bytes que lee, mientras que por salida devuelve el byte que lee.

La función LecEsc tiene como entrada el dato que queremos escribir y devuelve lo que lee.

4.1.2 Write16(escritura)

Write16 es similar a la función anterior, salvo que escribe mediante SPI un dato de 16 Bits (2 Bytes), en vez de uno de 8. Hace uso de la función LecEsc anterior, ya que escribe el byte menos significativo, desplaza a la derecha el restante y lo escribe; de forma que lo que hace es divider en datos en bytes e ir escribiendolo uno a uno. Recibe como entrada únicamente el dato a escribir.

4.1.3 Write32(escritura)

Es una función equivalente a la anterior, salvo que escribe tres bytes en lugar de dos. Hace uso nuevamente de la función LecEsc ya que divide el dato en bytes y lo escribe uno a uno. Como único dato de entrada recibe el dato a escribir.

4.1.4 **Read32()**

Esta función utiliza la función LecEsc para leer un dato de 3 bytes, de forma que lee 3 bytes separados y los une siguiendo el orden MSB. No contiene ninguna entrada y devuelve el valor del dato obtenido.

Nota 1: Podría crearse del mismo modo la función Read16 para datos de dos bytes, pero no ha sido necesario ya que no se utiliza para el desarrollo de la librería.

4.1.5 **SendAdressRD(address)**

Sirve para mandar a la pantalla la dirección de memoria de la que queremos leer, como se explicó en el protocolo. La dirección es un dato de tres bytes que se introduce a la función (address) y deben enviarse uno a uno como ya se explicó. Además, para que la pantalla entienda que le estas mandando una dirección de memoria de la que vas a leer, hay que hacerle un cierto tratamiento al dato:

Primero, tenemos que tener en cuenta que la guía de programación del FT800 nos dice que al inicio y al final debemos enviar un byte de ceros. El primer byte de ceros se debe enviar porque las direcciones de memoria son de 24 bits, en vez de 32, mientras que el byte final se envía para que al dato le dé tiempo a ser leído por el SPI. Este último no pertenece a la dirección de memoria. Además, debemos forzar a que los dos bits mas significativos de la dirección deben estar forzados a ceros, lo que permite diferenciarlo de una dirección sobre la que se va a escribir.

De esta forma, el dato a enviar quedaría de la siguiente manera:

00000000 00xxxxxx xxxxxxxx xxxxxxxx

Donde las x muestran bits cualesquiera y no se ha mostrado el último byte de ceros al no pertenecer a la dirección.

4.1.6 **SendAdressWR(address)**

Esta función sirve para mandar a la pantalla una dirección de memoria sobre la que vamos a escribir. Es prácticamente igual que la función anterior, salvo dos pequeñas diferencias: los dos bits mas significativos de la dirección deben estar forzados a 1 y 0 respectivamente y no se debe mandar el byte de ceros del final.

De esta forma, el dato a enviar quedaría de la siguiente manera:

00000000 10xxxxxx xxxxxxxx xxxxxxxx

Donde las x muestran bits cualesquiera y tampoco se ha mostrado el último byte de ceros. Si no se siguen minuciosamente estos pasos no será posible la comunicación.

4.1.7 **HostCommand(hostcom)**

Al encendido de la pantalla, es necesario enviar unos comandos especiales y de una manera especial para poder iniciar la pantalla. Estos comandos, de un byte de longitud, se llaman hostcommand y nos vienen definidos en el archivo FT800.

Esta función trata el hostcommand introducido en la función para poder mandarlo de la forma adecuada, que es la siguiente:

- Activamos el CS.
- Forzamos a que los dos bits mas significativos sean 0 y 1 respectivamente, y mandamos el dato con la función LecEsc.
- Enviamos dos bytes de ceros separados.
- Desactivamos el CS.

4.1.8 HostCommandDummy()

Como hemos dicho anteriormente, la pantalla hace las funciones de esclavo y esta se debe despertar por impulso para iniciar la comunicación. Esto lo hacemos mandándole un "dummy" de ceros, es decir, tres bytes de ceros, precedidos por la activación del CS y con su desactivación posterior. Realmente esta función, al hacer uso de ella una sola vez, no haría falta crearla; pero se ha hecho para facilitar la lectura y depuración del código.

El modo de funcionamiento es el siguiente:

- Activamos el CS.
- Enviamos tres bytes de ceros separados.
- Desactivamos el CS.

4.2 Configuración de la pantalla

En grupo se encuentran las funciones destinadas a la configuración de la pantalla, así como la escritura en memoria y su inicialización.

4.2.1 inc_cmdoffset(current_off, size)

Esta función se encarga de calcular la dirección de la memoria FIFO donde el siguiente comando debe ser escrito. Una vez se ha escrito un comando, la función recibe la dirección de memoria anterior a que se escribiera (current_off) y lo incrementa según el tamaño del comando, introducido como entrada (size). También tiene en cuenta la saturación, que ocurre cuando supera el valor 4096, que hace que vuelva al inicio al ser una memoria circular.

4.2.2 EscRam32/16/8(dato)

Esta subsección describe tres funciones que, por su similitud, se pueden explicar a la vez. Ellas permiten escribir en la memoria RAM el dato introducido como entrada. Según el dato sea de tres, dos o un byte se utilizará una función u otra (EscRam32, EscRam16 o EscRam8 respectivamente). Primero se envía la dirección de memoria correspondiente, RAM_CMD mas el offset calculado, y tras esto se escribe el dato usando Write32, Write16 o LecEsc según el tamaño del dato. Por ultimo, se calcula el nuevo offset tras el comando, mediante la función inc_cmdoffset(current_off, size), prestando atención en poner size según el tamaño del dato introducido.

4.2.3 comando(commando)

Esta función se emplea para escribir comandos, de 3 bytes, introducidos como entradas en la memoria de la pantalla. Su dirección de memoria viene marcada por la dirección de comandos de la memoria RAM (RAM_CMD) mas el offset calculado con la función anterior. Podemos comprobar como dicha función es igual que EscRam32, pero se ha diferenciado porque los comandos introducidos son datos específicos almacenados en el archivo FT800.

4.2.4 comesperaFin()

Como se ha explicado anteriormente, para continuar con la comunicación en SPI se debe esperar a que se haya terminado de escribir el dato actual. Por lo tanto, esta función se asegura que la comunicación SPI no está ocupada y se puede utilizar. Para ello, lee los registros REG_CMD_READ y REG_CMD_WRITE y hasta que no coinciden no permiten avanzar en el programa. Esto solo ocurre cuando no se está trasmitiendo datos

mediante la comunicación SPI.

4.2.5 **PadFIFO()**

Como se ha podido comprobar, tanto los registros de memoria como los comandos de la memoria FIFO se agrupan en datos de 4 bytes. Sin embargo, a veces se deben escribir datos cuya longitud no podemos garantizar que sea de este tamaño, como por ejemplo cadenas de caracteres o números. Ante esta problemática, esta función comprueba si la dirección de la memoria FIFO es multiplo de cuatro, siendo así si todos los datos introducidos son datos de 4 bytes. En caso de no serlo, rellena de ceros la memoria resulante para que la dirección siguiente sea multiplo de cuatro.

4.2.6 **Inicia_pantalla()**

Esta función se utiliza para el encendido de la pantalla. En ella se desarrolla todo el protocolo de encendido, presente en la guía de programación del FT800, siendo éste muy estricto e imprescindible de realizar meticulosamente, ya que sino sería imposible el funcionamiento. A continuación, vamos a enumerar paso a paso los distintos registros y comandos que se deben escribir para realizar el encendido:

- Activación del pin PD, necesario para activar la alimentación de la pantalla.
- Espera de 16ms.
- Envío de 3 bytes de ceros, mediante la función HostCommandDummy, necesario para despertar la comunicación.
- Envío del Host Command FT_GPU_EXTERNAL_OSC presente en el archivo FT800, necesario para cambiar a reloj externo.
- Envío del Host Command FT_GPU_PLL_48M presente en el archivo FT800, necesario para configurar el reloj a 48MHz.
- Espera de 16ms.
- Envío del Host Command FT_GPU_CORE_RESET presente en el archivo FT800, necesario para resetear el núcleo.
- Espera de 16ms.
- Envío del Host Command FT_GPU_ACTIVE_M presente en el archivo FT800, necesario para asegurar que el FT800 está activo.

Nota 2: cada una de las esperas se realizarán con la función propia de Micropython sleep_us, incluida en la librería machine, donde debemos introducir el número de microsegundos que debe esperar.

A continuación, viene el momento crítico de la inicialización, donde se comprueba que todo lo realizado anteriormente funciona. Debemos comprobar que el dispositivo al que está conectado el micro es realmente el tipo de pantalla que deseamos. Si lo realizado anteriormente no se hubiera hecho rigurosamente, el micro no detectaría el tipo de pantalla que estamos utilizando. La variable donde vamos a escribir la identificación que lee el micro se llamará chipid. Solo si esta variable coincide con el número de identificación de la pantalla podremos seguir adelante. En nuestro caso, el número de identificación será 124 (7C si lo ponemos en hexadecimal). El protocolo para leer el número de identificación será el siguiente:

- Activamos el CS.
- Enviamos la dirección de memoria REG_ID en modo lectura.
- Leemos el contenido de dicho registro con la función LecEsc, introduciéndole un 0 al no querer escribir.
- Desactivamos el CS.
- Espera de 32ms.

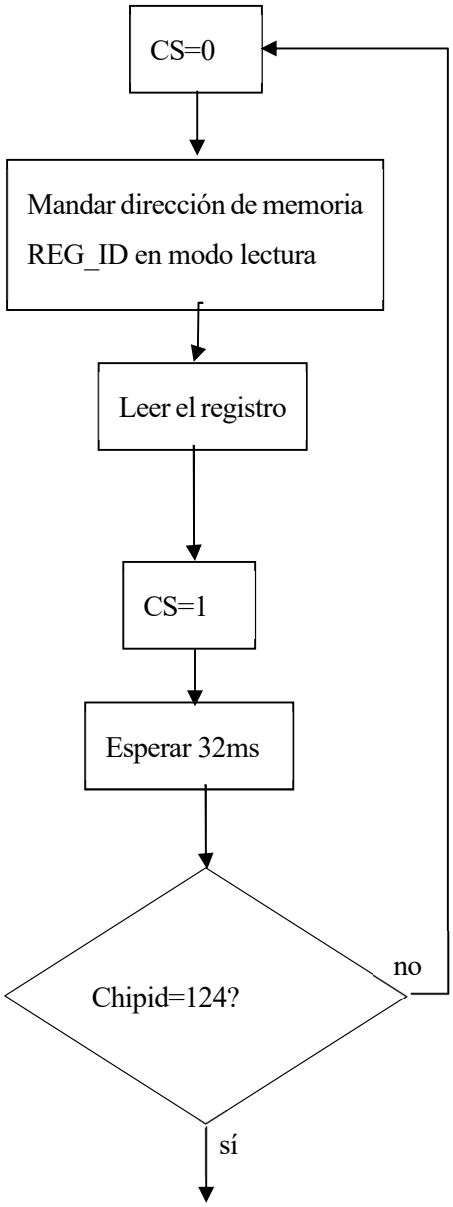

Figura 4.2: Esquema comprobación chipid

Todo el código anterior se ha introducido dentro de un bucle while para que el micro no deje de comprobar el dispositivo conectado hasta que sea el deseado. Así mismo, se ha pasado el chipid a entero para poderlo mostrar por pantalla comodamente en caso de necesidad.

Una vez superado este eslabón, continuamos con la inicialización. En los pasos siguientes, se van a ir escribiendo en distintas direcciones de memoria, siempre con el protocolo de activación/desactivación del CS, por lo que se va a ahorrar comentarlo en cada paso, para mejorar la legibilidad del documento:

- Escribir en la dirección de memoria REG_HCYCLE el dato HCYCLE de 2 bytes mediante la función Write16, necesario para ajustar el número de ciclos de reloj por cada línea horizontal.
- Escribir en la dirección de memoria REG_HOFFSET el dato HOFFSET de 2 bytes mediante la función Write16, necesario para ajustar el número de ciclos de reloj antes de que los pixeles sean escaneados.
- Escribir en la dirección de memoria REG_HSYNC0 el dato HSYNC0 de 2 bytes mediante la función Write16.
- Escribir en la dirección de memoria REG_HSYNC1 el dato HSYNC1 de 2 bytes mediante la función

Write16.

- Escribir en la dirección de memoria REG_VCYCLE el dato VCYCLE de 2 bytes mediante la función Write16, necesario para indicar el número de líneas del marco.
- Escribir en la dirección de memoria REG_VOFFSET el dato VOFFSET de 2 bytes mediante la función Write16, necesario para indicar el número de líneas tomadas tras empezar un nuevo marco.
- Escribir en la dirección de memoria REG_VSYNC0 el dato VSYNC0 de 2 bytes mediante la función Write16.
- Escribir en la dirección de memoria REG_VSYNC1 el dato VSYNC1 de 2 bytes mediante la función Write16.
- Escribir en la dirección de memoria REG_SWIZZLE el dato 2 mediante la función Write16, necesario para ajustar el control de los pines RGB de la pantalla LCD. Para mas información acerca de los valores que puede tomar, véase la guía de programación del FT800.
- Escribir en la dirección de memoria REG_PCLK_POL el dato PCLK_POL de 2 bytes mediante la función Write16, necesario para ajustar la polaridad del reloj, siendo 0 en el eje ascendente y 1 en el descendente (PCLK_POL=0).
- Escribir en la dirección de memoria REG_HSIZE el dato HSIZE de 2 bytes mediante la función Write16, necesario para indicar el número de ciclos de reloj por cada línea horizontal.
- Escribir en la dirección de memoria REG_VSIZE el dato VSIZE de 2 bytes mediante la función Write16, necesario para indicar el número de líneas de pixeles en un marco.
- Escribir en la dirección de memoria REG_PCLK el dato PCLK de 2 bytes mediante la función Write16, necesario para indicar el número por el que dividiremos el reloj principal (usualmente 5).
- Escribir en la dirección de memoria REG_TOUCH_RZTHRESH el dato 1200 de 2 bytes mediante la función Write16, donde se indica la sensibilidad de la pantalla táctil. El valor por defecto es 0xFFFF, es decir, sensibilidad máxima; pero usualmente se suele utilizar 1200 para añadir más resistencia.

Una vez realizada la configuración básica, procedemos a limpiar la pantalla a un color negro:

- Escribir en la dirección de memoria RAM_DL+0, correspondiente a la primera posición del registro de memoria, el dato 0x02000000 de 4 bytes mediante la función Write32, necesaria para poner a 0 los pines RGB de la pantalla. Dicho registro esta formado por 4 bytes siendo el menos significativo el correspondiente al color rojo, el siguiente al verde, el tercero al azul y el mas significativo a 00000010 (valor por defecto).
- Escribir en la dirección de memoria RAM_DL+4, correspondiente a la segunda posición del registro de memoria, el dato 0x26000007 de 4 bytes mediante la función Write32, necesaria para poner a 0 los pines CST (colour, stencil, tap) de la pantalla. Dicho registro esta formado por 4 bytes siendo el mas significativo 00100110 (valor por defecto), y los tres bits menos significativos CST, respectivamente.
- Escribir en la dirección de memoria RAM_DL+8, correspondiente a la tercera posición del registro de memoria, el dato 0 de 4 bytes mediante la función Write32.
- Escribir en la dirección de memoria REG_DLSWAP el dato 2 de 4 bytes mediante la función Write32, necesario para ajustar cuando representar en pantalla. Usualmente se añade 2, representar inmediatamente después de que el marco actual sea mostrado, aunque se podría poner otra configuración (véase la guía de programación del FT800).

Por último, debemos ajustar los pines GPIO de la pantalla, necesarios par que los módulos de desarrollo sean usados:

- Escribir en la dirección de memoria REG_GPIO_DIR el dato 0x83 de 1 bytes mediante la función LecEsc, necesario para indicar la dirección de los pines GPIO.
- Escribir en la dirección de memoria REG_GPIO el dato 0x81 de 1 bytes mediante la función LecEsc, necesario para deshabilitar el amplificador de audio, siendo 0x81 deshabilitado y 0x83 habilitado.

4.2.7 **Calibra()**

Mediante esta función, podemos calibrar la pantalla, mediante un pequeño juego que se explica a continuación. Tras ejecutarla, lo primero que aparece es una pantalla donde nos indica que hemos seleccionado la calibración, y permanece hasta que pulsemos sobre la pantalla. Tras pulsar, nos aparece una donde se nos indica que tenemos que pulsar sobre los diferentes puntos que van apareciendo. Al ir pusando, el microcontrolador va registrando los puntos y creando un vector de parámetros. Una vez pulsado sobre todos los puntos, nos aparece el vector de parámetros indicándonos que debemos apuntarlos. Estos datos deben ser introducidos en el archivo FT800, actualizando el registro REG_CAL. De esta forma, habremos calibrado la pantalla correctamente.

4.3 Funciones para el diseño de la pantalla y control del interfaz táctil

En esta sección vamos a tratar tanto las funciones necesarias para utilizar la función táctil como las funciones de diseño y dibujo, ya todas de alto nivel.

4.3.1 **LeePant()**

Mediante esta función se registra la posición presionada sobre la pantalla táctil. Se realiza leyendo en la dirección de memoria REG_TOUCH_SCREEN_XY. El dato obtenido contiene en la parte más significativa las coordenadas de la posición x y en la menos las de la posición y. Por ultimo, se ha añadido unas funciones para interpolar las variables de forma que varíen de 0 a 480 y de 0 a 272 respectivamente. Éstas, son almacenadas en dos variables internas de la librería, POSX y POSY, que pueden ser usadas desde el exterior.

4.3.2 **EspPant()**

A veces resultará útil mantener en espera la pantalla hasta que se pulse sobre ella, cosa que permite hacer esta función. Es de utilidad cuando se debe esperar a que se pulse una tecla, para no tener el micro en espera mediante un delay.

4.3.3 **Ejecutalista**

Una vez han sido introducidos los comandos en la memoria FIFO, estos deben ser ejecutados. Esta función se encarga de ejecutar ello. El procedimiento a seguir es el siguiente:

- Activar el CS.
- Enviar la dirección de memoria REG_CMD_WRITE en modo escritura.
- Escribir sobre ella la posición de memoria FIFO (cmdoffset).
- Desactivar CS.

4.3.4 **Dibuja**

Esta función se encarga de, una vez se le ha indicado todo lo que queremos dibujar en la pantalla, mostrarlo todo. Cada vez que cambiemos algo de lo que hemos representado en la pantalla debemos emplearla. Para ello, realiza lo siguiente: primero envía el comando CMD_DISPLAY, luego el comando CMD_SWAP y por último utiliza la función Ejecutalista explicada anteriormente.

4.3.5 **Colores**

La pantalla permite, para cada objeto, emplear tres niveles diferentes de coloreado: el color principal, el color background y el color foreground. No todos son utilizables en todos los objetos a dibujar, dependiendo de cada uno se permitirá modificar solo el color principal o también alguno de ellos. Posteriormente se detallará objeto por objeto de que colores dispone. Toda modificación del color se mantiene efectivo desde que aparece la función hasta que volvamos a emplearla. Las siguientes funciones se encargan de modificar los distintos colores:

- ComColor(R,G,B): permite seleccionar el colo principal que se usará. En ella introducimos la intensidad de cada uno de los colores primarios (rojo, verde y azul) que se agrupan en un solo dato de forma que el byte mas significativo es R, el siguiente G y el menos B. Por último, debemos mandar este dato sumado a 0x04000000 como comando.
- ComFgColor(R,G,B): sirve para seleccionar el colo de foreground que se usará. El funcionamiento es idéntico al de la función anterior, salvo que, a la hora de mandar el comando, no debemos sumarle la cifra de la función anterior.
- ComBgColor(R,G,B): permite seleccionar el color de background que se usará. El funcionamiento es idéntico al de la función ComFgColor(R,G,B), salvo que, a la hora de mandar el comando, no debemos sumarle la cifra de la función anterior.

4.3.6 **ComVertex2ff(x,y)**

Siempre que queramos dibujar líneas o figuras geométricas necesitamos indicarle a la pantalla algunos puntos de la pantalla. Para ello, utilizaremos esta función donde la entradas son las coordenadas de un punto (x,y). el funcionamiento es el siguiente:

- Primero debemos convertir las dos componentes en un solo dato, desplazando la coordenada x 19 posiciones y sumándole la coordenada y desplazada 4 posiciones.
- Activamos CS.
- Escribimos en la dirección de memoria de la FIFO (RAM_CMD+cmdoffset) el dato obtenido sumándole 0x40000000 para indicar que es una coordenada.
- Desactivamos CS.
- Usamos la función inc_cmdoffset para modificar la dirección actual de la memoria FIFO, tras haberle añadido el dato anterior.

A continuación se describirán las diferentes funciones necesarias para dibujar líneas o figuras geométricas:

- Linea(x1,y1,x2,y2,r): Esta función nos permite dibujar una línea en la pantalla, desde una posición inicial (x1,y1) hasta una posición final (x2,y2) con un grosor r, todo introducido como entrada. Para ello primero debemos enviar el comando CMD_BEGIN_LINES, luego el comando CMD_LINEWIDTH sumando el grosor de la línea multiplicado por 16 e introducir las cordenadas de inicio y fin con la función anterior. Para terminar, se debe enviar el comando CMD_END.
- Rectangulo(xr1,yr1,xr2,yr2,r,relleno): Mediante esta función podemos dibujar rectángulos rellenos, si relleno igual a 1, o huecos si relleno es igual a 0, desde la posicion inicial (xr1, yr1) hasta la posición final (xr2, yr2). En caso de que sea hueco, también debemos introducir el grosor del rectángulo, si fuese hueco podemos poner cualquier valor. Si fuese hueco, el procedimiento de la función es igual a la función anterior, de forma que se deben escribir las cuatro líneas que forman el rectángulo. En caso de que fuese relleno, el procedimiento es el siguiente: primero debemos enviar el comando CMD_BEGIN_RECTS, luego el comando CMD_LINEWIDTH sumando 80 e introducir las cordenadas de inicio y fin con la función anterior. Para terminar, ya sea hueco o relleno, se debe enviar el comando CMD_END.

- Circulo(x,y,r): Esta función permite dibujar un circulo con centro en la posición xy y de radio r, todo introducido por pantalla. Para su implementación, primero se envía el comando CMD_POINTSIZE sumándole el radio multiplicado por 16, se envía el comando CMD_BEGIN_POINTS, se añade las cordenadas xy con la función ComVertex2ff, y por ultimo, se envía el comando CMD_END.

4.3.7 **Brillo(int)**

Para variar el brillo, debemos hacer uso de esta función, introduciendo éste como entrada. Dicha opción tiene un rango de 0 a 128, siendo 0 totalmente apagado y 128 al brillo máximo. Para ello, la función escribe en la dirección REG_PWM_DUTY el valor del brillo introducido como entrada, con la previa activación del CS y su posterior desactivación.

4.3.8 **LeeBrillo()**

Para modificar el brillo de manera relativa al brillo actual, necesitamos hacer uso de esta función que nos indica el valor actual. Este valor se encuentra, como se ha explicado en la función anterior, entre 0 y 128; siendo 0 totalmente apagado y 128 al brillo máximo. Para ello, la función lee de la dirección REG_PWM_DUTY, con la previa activación del CS y su posterior desactivación. Dado que el valor obtenido se encuentra en hexadecimal, se le ha añadido una conversión para poder obtener directamente el valor en entero.

4.4 Control del Sistema acústico de la pantalla

En esta seccion vamos a tartar una serie de funciones que nos permitirán emitir notas musicales con la pantalla.

Para tocar notas musicales debemos emplear una serie de funciones. Para ello hay que seguir un protocolo especial:

1. La nota musical comienza a sonar desde que aparece la función Nota(instrum, nota).
2. Para apagar las notas que están sonando, debemos emplear la función FinNota().
3. El volumen de las notas se establece en todas las notas posteriores a la implementación de la función VolNota(vol) hasta que la función vuelve a ser empleada.

A continuación, se va a desarrollar cada una de las funciones mencionadas.

4.4.1 **Nota(instrum, nota):**

Esta función se utiliza para introducir notas musicales para su posterior interpretación. Para ello, debemos introducirle como entrada el instrumento (instrum) y la nota deseada (nota). Estos dos datos se modifican para formar uno solo de 1 byte siendo la nota el byte mas significativo y el instrumento el menos significativo. La función se desarrolla de la siguiente forma: primero escribe en la dirección de memoria REG_SOUND el dato anteriormente modificado. Por último, escribe en el registro REG_PLAY un 1 para proceder a tocar la nota.

4.4.2 **FinNota()**

Mediante esta función deja de sonar las notas anteriormente representada. Para ello, la función primero escribe en la dirección de memoria REG_SOUND 0x00 para eliminar el instrumento y la nota del registro. Por último, escribe en el registro REG_PLAY un 1 para, al haber introducido 0x00 en el registro anterior, no tocar ninguna nota.

4.4.3 VolNota(vol)

A través de esta función, vamos a modificar el volumen de las notas que tocaremos posteriormente, introduciendo este valor como entrada. Para ello, la función escribe en la dirección de memoria REG_VOL_SOUND el valor introducido como entrada.

4.5 Comandos especiales

En esta sección hablaremos de una serie de comandos graficos, ya introducidos por el fabricante, que nos permite hacer una serie de diseños especiales. Cada uno de ellos nos permite dibujar objetos tales como relojes, números, teclas… En la imagen siguiente, podemos ver un ejemplo de cada uno de estos objetos, junto con el comando que se debe enviar a la pantalla. Las otras dos imágenes nos muestran diferentes opciones de diseño a usar por las funciones y que colores podemos modificar de cada objeto creado, especificando si continen los colores de foreground y background a parte del color principal.

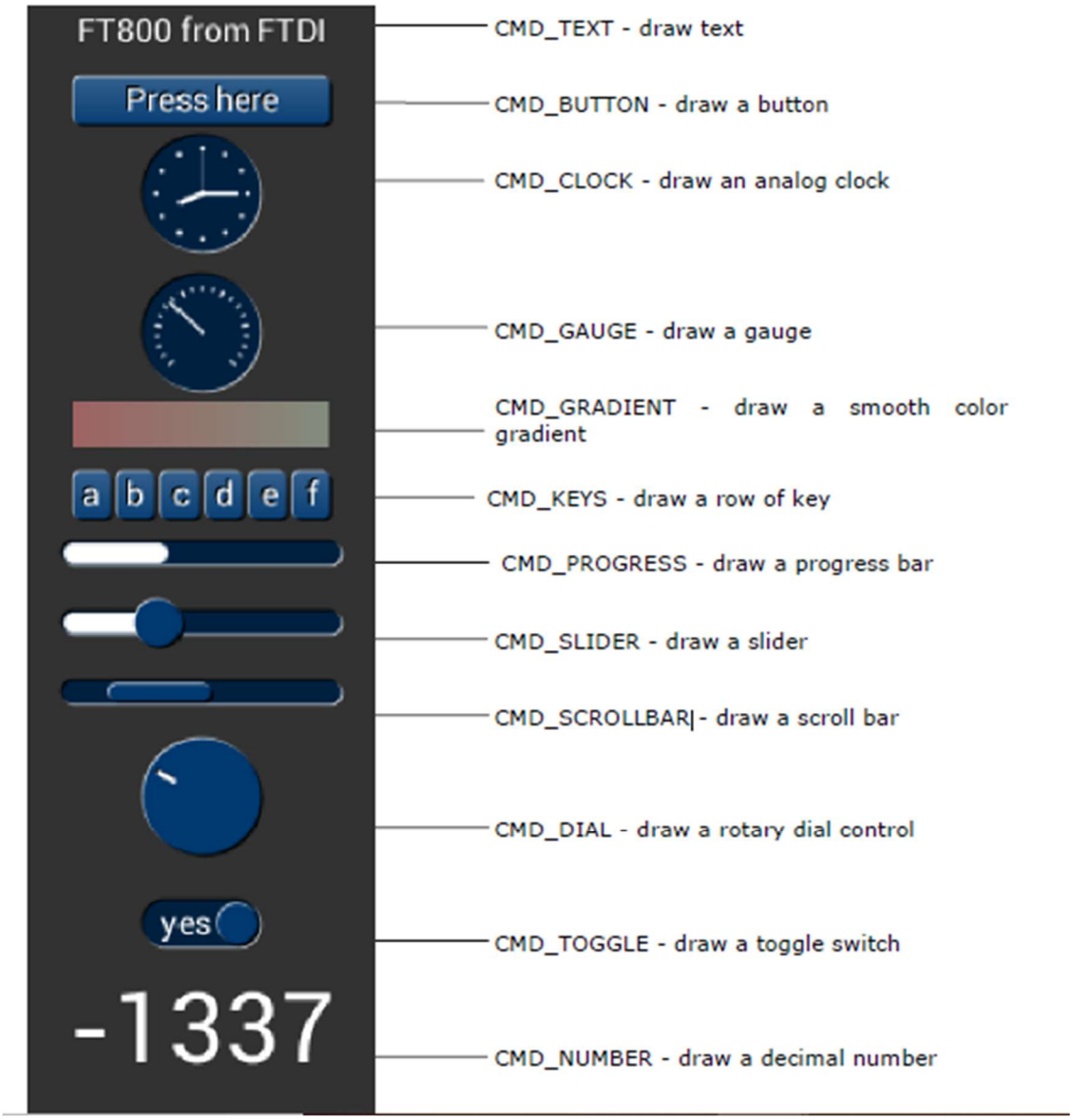

Figura 4.3: Comandos especiales de diseño

Fuente: Archivo FT800 Programmers Guide

Name	Value	Description	Commands
OPT_3D	0	Co-processor widget is drawn in 3D effect. The default option.	CMD_BUTTON,CMD_CLOCK,CMD_KEYS, CMD_GAUGE,CMD_SLIDER, CMD_DIAL, CMD_TOGGLE,CMD_PROGRESS, CMD_SCROLLBAR
OPT_RGB565	0	Co-processor option to decode the JPEG image to RGB565 format	CMD_IMAGE
OPT_MONO	1	Co-processor option to decode the JPEG image to L8 format, i.e., monochrome	CMD_IMAGE
OPT_NODL	2	No display list commands generated for bitmap decoded from JPEG image	CMD_IMAGE
OPT_FLAT	256	Co-processor widget is drawn without 3D effect	CMD_BUTTON,CMD_CLOCK,CMD_KEYS, CMD_GAUGE,CMD_SLIDER, CMD_DIAL, CMD_TOGGLE,CMD_PROGRESS, CMD_SCROLLBAR
OPT_SIGNED	256	The number is treated as 32 bit signed integer	CMD_NUMBER
OPT_CENTERX	512	Co-processor widget centers horizontally	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_CENTERY	1024	Co-processor widget centers vertically	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_CENTER	1536	Co-processor widget centers horizontally and vertically.	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_RIGHTX	2048	The label on the Co-processor widget is	CMD_KEYS,CMD_TEXT, CMD_NUMBER

Figura 4.4: Opciones de diseño

Fuente: Archivo FT800 Programmers Guide

Widget	CMD_FGCOLOR	CMD_BGCOLOR	COLOR_RGB
CMD_TEXT	NO	NO	YES
CMD_BUTTON	YES	NO	YES(label)
CMD_GAUGE	NO	YES	YES(needle and mark)
CMD_KEYS	YES	NO	YES(text)
CMD_PROGRESS	NO	YES	YES
CMD_SCROLLBAR	YES(Inner bar)	YES(Outer bar)	NO
CMD_SLIDER	YES(Knob)	YES(Right bar of knob)	YES(Left bar of knob)
CMD_DIAL	YES(Knob)	NO	YES(Marker)
CMD_TOGGLE	YES(Knob)	YES(Bar)	YES(Text)
CMD_NUMBER	NO	NO	YES
CMD_CALIBRATE	YES(Animating dot)	YES(Outer dot)	NO
CMD_SPINNER	NO	NO	YES

Figura 4.5: Colores utilizados por función

Fuente: Archivo FT800 Programmers Guide

4.5.1 Texto(x, y, fuente, ops, cadena)

Siempre que queramos mostrar una cadena de caracteres debemos emplear esta función. Para ello tenemos que introducir como entrada la posición de comienzo del taxto (x,y), la fuente, las opciones (ops) y la cadena que queremos introducir (cadena). El procedimiento a seguir por esta función es el siguiente: primero se escribe en Ram el comando CMD_TEXT, luego se escriben las entradas x, y, fuente y ops. A continuación, se debe escribir cada letra de la cadena introducida, siendo cada una de ellas un dato de un byte. Por último, introducimos un byte de ceros para evitar interferencias y llamamos a la función PadFIFO.

Dentro de las opciones que podemos introducir están las siguientes:

- OPT_CENTERX: que centra el texto horizontalmente.
- OPT_CENTERY: que centra el texto verticalmente.
- OPT_CENTER: que centra el texto en ambas direcciones.
- OPT_RIGHTX: que sitúa el texto lo mas a la derecha posible.

4.5.2 Boton(x, y, w, h, fuente, ops, cadena)

Esta función sirve para la creación de botonos donde se incluye dentro una cadena de caracteres. Como entrada recibe la posición del botón (x, y), el ancho (w), la altura (h), la fuente de la cadena de caracteres, las opciones (ops) y la cadena a introducir. Para la creación del botón se actúa de la siguiente forma: primero se escribe en RAM el comando CMD_BUTTON, de 4 bytes. Tras esto, se escribe en RAM x, y, w, h, ops y fuente; todas ellas de 2 bytes. A continuación, se debe escribir cada letra de la cadena introducida, siendo cada una de ellas un dato de un byte. Por último, introducimos un byte de ceros para evitar interferencias y llamamos a la función PadFIFO.

Dentro de las opciones de diseño que se pueden emplear se encuentra lo siguiente:

- 0: es la opción por defecto. Mantiene el formato 3D (botón no pulsado).
- OPT_FLAT: desactiva el formato 3D (botón pulsado).

4.5.3 **TocaBoton(x, y, w, h, fuente, ops, cadena)**

Como la implementación del pulsado de botones es siempre idéntica, se ha creado esta función de ayuda. No es necesaria para la implementación de los botones, pero gracias a ella podemos realizar el pulsado de botones de manera sencilla. Utiliza la función LeePant para detectar donde se está pulsando, ya que devuelve un 1 si se ha pulsado o un 0 si no. Si detecta que se ha pulsado sobre el botón, mediante un if, cambia la opción introducida al botón, para que el botón aparezca como pulsado y devuelve un 1 para señalizarlo. En caso de no haber pulsado, vuelve a la configuración anterior y devuelve un 0.

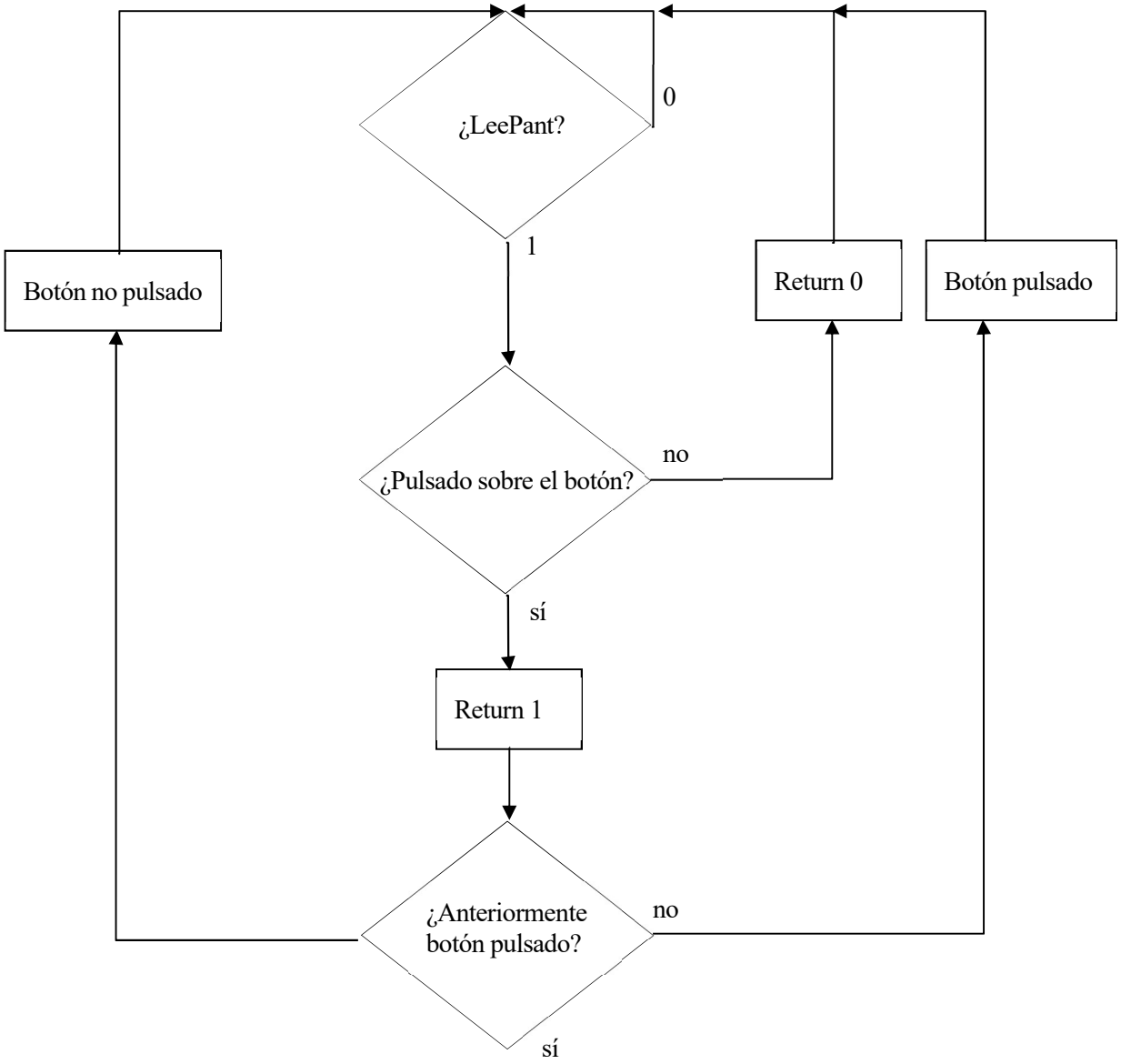

Figura 4.6: Diagrama de flujo función TocaBotón

4.5.4 **Reloj(x, y, r, ops, h, m, s, ms)**

Mediante esta función se permite introducir relojes analógicos dentro de la pantalla. Para ello, debemos introducirle la posición del reloj (x, y), el radio (r), las opciones (ops), y la hora que marca (h, m, s, ms); como podemos ver en la imagen posterior. Para implementarlo, la función primero escribe en RAM el comando CMD_CLOCK de 4 bytes y luego todas las entradas, que son todas de 2 bytes.

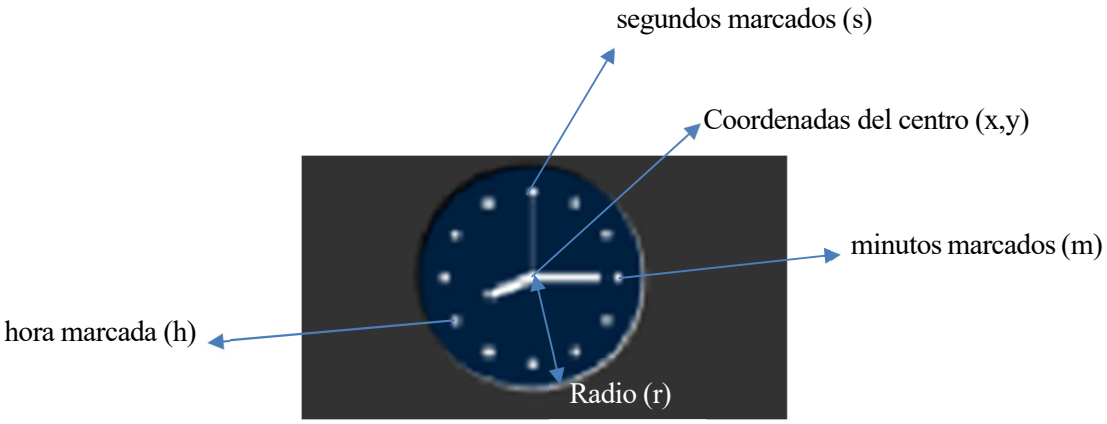

Figura 4.7: Ejemplo reloj

Fuente: Archivo FT800 Programmers Guide

Dentro de las opciones de diseño que se pueden utilizar están las siguientes:

- 0: es la opción por defecto y activa el efecto 3D.
- OPT_FLAT: desactiva el efecto 3D.
- OPT_NOBACK: con esta opción el fondo del reloj se muestra no dibujado.
- OPT_NOTICKS: con esta opción no se muestran las divisiones que determinan las distintas horas.
- OPT_NOSECS: con esta opción no se muestra la manecilla de los segundos.
- OPT_NOHANDS: con esta opción no se muestra ninguna de las manecillas.
- OPT_NOHM: con esta opción no se muestran las manecillas de los minutos y segundos.

4.5.5 **Medidor(x, y, r, ops, max, min, val, range)**

Esta función sirve para añadir un medidor de aguja a la pantalla. Para ello debemos introducirle la posición del medidor (x, y), el radio del medidor (r), las opciones de diseño (ops), número de subdivisiones mayores (max), número de subdivisiones menores (min), el valor que indique el medidor (val) y el valor máximo que alcanza (rango); como se ilustra en la imagen posterior. La función primero escribe en RAM el comando CMD_GAUGE de 4 bytes y luego todas las entradas, que son todas de 2 bytes.

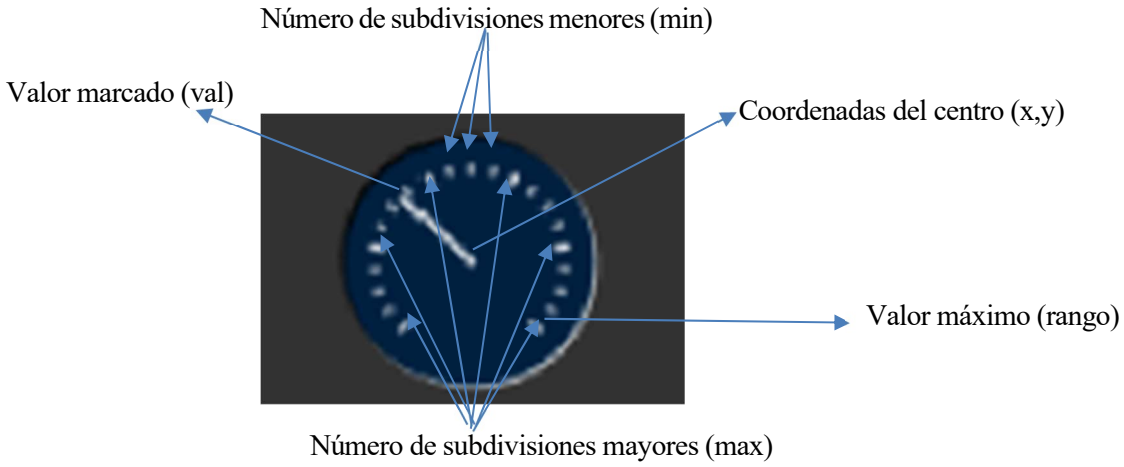

Figura 4.8: Ejemplo medidor

Fuente: Archivo FT800 Programmers Guide

Dentro de las opciones de diseño que se pueden utilizar están las siguientes:

- 0: es la opción por defecto y activa el efecto 3D.
- OPT_FLAT: desactiva el efecto 3D.
- OPT_NOBACK: con esta opción el fondo del reloj se muestra no dibujado.
- OPT_NOTICKS: con esta opción no se muestran las divisiones que determinan las distintas horas.
- OPT_NOPOINTER: con esta opción no se dibujan los puntos correspondientes a las disiviones del medidor.

4.5.6 **ComGradiente(x1, y1, col1, x2, y2, col2)**

La pantalla nos permite introducir una forma de coloreo de objetos especial, un gradiente de colores. Esta función crea una pantalla de gradiente de colores formando un rectángulo entre los puntos x1 e y1, correspondiente a la posición de inicio y los puntos x2 e y2 que establecen la posición final. Además, el gradiente va desde un color inicial, col1, hasta uno final, col2. Todos estos datos se introducen como entrada. Para ello, la función primero escribe en RAM el comando CMD_GRADIENT de 4 bytes y luego todas las entradas, que son todas de 2 bytes.

4.5.7 **Teclas(x, y, w, h, fuente, ops, cadena)**

Esta función se utiliza para introducir teclas en la pantalla, con una cadena de caracteres si quisiéramos. Como entrada recibe la posición del botón (x, y), el ancho (w), la altura (h), la fuente de la cadena de caracteres, las opciones (ops) y la cadena a introducir. Para la creación del botón se actúa de la siguiente forma: primero se escribe en RAM el comando CMD_KEYS, de 4 bytes. Tras esto, se escribe en RAM x, y, w, h, ops y fuente; todas ellas de 2 bytes. A continuación, se debe escribir cada letra de la cadena introducida, siendo cada una de ellas un dato de un byte. Por último, introducimos un byte de ceros para evitar interferencias y llamamos a la

función PadFIFO.

Dentro de las opciones de diseño que se pueden emplear se encuentra lo siguiente:

- 0: es la opción por defecto. Mantiene el formato 3D (botón no pulsado).
- OPT_FLAT: desactiva el formato 3D (botón pulsado).
- OPT_CENTER: con esta función las teclas se reducen al mínimo tamaño posible.

4.5.8 **Avance(x, y, w, h, ops, val, range)**

Esta función se utiliza para introducir un indicador de avance o progreso en forma de barra hoeizontal. Para ello, debemos introducir la posición de la parte superior izquierda (x, y), el ancho de la barra (w), la altura (h), las opciones de diseño (ops), el valor que toma el avance (val) y el máximo valor que puede tomar (range); como se observa en la siguiente figura. Para ello, la función escribe en RAM el comando CMD_PROGRESS de 4 bytes, y el resto de los valores de entrada; todos ellos de 2 bytes.

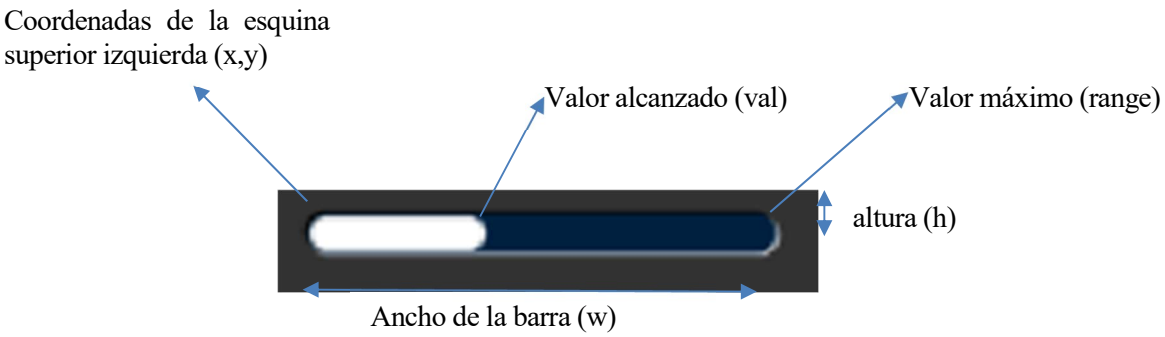

Figura 4.9: Ejemplo avance

Fuente: Archivo FT800 Programmers Guide

Dentro de las opciones de diseño que se pueden emplear se encuentra lo siguiente:

- 0: es la opción por defecto. Mantiene el formato 3D (botón no pulsado).
- OPT_FLAT: desactiva el formato 3D (botón pulsado).

4.5.9 **Deslizador(x, y, w, h, ops, val, range)**

Mediante esta función, vamos a introducir un deslizador horizontal como el de la figura. Para ello, debemos introducir la posición de la parte superior izquierda (x, y), el ancho de la barra (w), la altura (h), las opciones de diseño (ops), el valor que toma el avance (val) y el máximo valor que puede tomar (range), como se puede observar en la siguiente figura. Su implementación es la siguiente: la función escribe en RAM el comando CMD_SLIDER de 4 bytes, y el resto de los valores de entrada; todos ellos de 2 bytes.

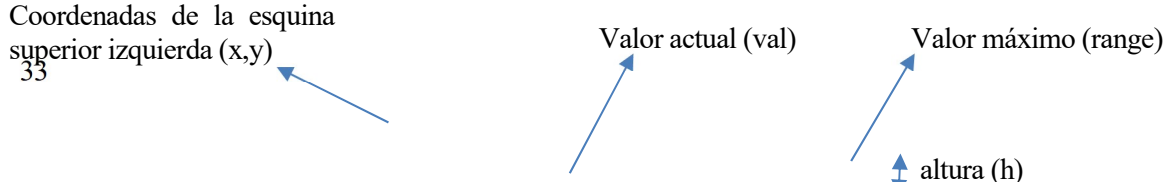

Figura 4.10: Ejemplo deslizador

Fuente: Archivo FT800 Programmers Guide

Podemos observar como esta función es idéntica a la anterior, ya que lo único que cambia es el diseño gráfico, al haber un circulo junto al valor actual. Esto es porque la función anterior se emplea para medir el avance de una variable, mientras que esta suele utilizarse como regulador lineal, pulsando sobre el circulo y moviéndolo para cambiar el valor. Para ello, aunque no se ha incluido dentro de la librería, podría crearse otra función que, mediante la función LeePant(), detecte cuando se esta pulsando sobre el circulo del regulador y reconozca cuanto se ha cambiado.

Dentro de las opciones de diseño que se pueden emplear se encuentra lo siguiente:

- 0: es la opción por defecto. Mantiene el formato 3D (botón no pulsado).
- OPT_FLAT: desactiva el formato 3D (botón pulsado).

4.5.10 Scrollbar(x, y, w, h, ops, val, size, range)

La pantalla también nos permite generar barras deslizantes, implementado mediante esta función, vamos a introducir un deslizador horizontal como el de la figura. Para ello, debemos introducir la posición de la parte superior izquierda (x, y), el ancho del regulador (w), la altura (h), las opciones de diseño (ops), el valor que toma el avance (val), el tamaño de la barra (size) y el máximo valor que puede tomar (range). Para ello, la función escribe en RAM el comando CMD_SCROLLBAR de 4 bytes, y el resto de los valores de entrada; todos ellos de 2 bytes. Esta opción de diseño es muy parecida a la anterior salvo que, en esta, ambos lados del deslizador contienen el mismo color.

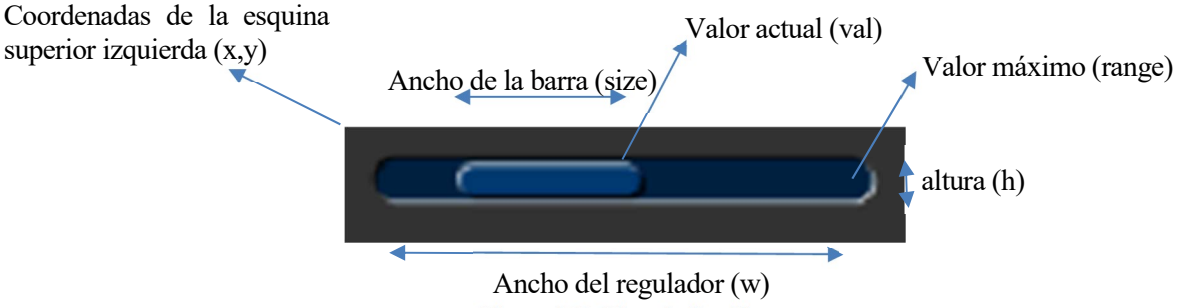

Figura 4.11: Ejemplo Scrollbar

Fuente: Archivo FT800 Programmers Guide

Dentro de las opciones de diseño que se pueden emplear se encuentra lo siguiente:

- 0: es la opción por defecto. Mantiene el formato 3D (botón no pulsado).
- OPT_FLAT: desactiva el formato 3D (botón pulsado).

4.5.11 **Dial(x, y, r, ops, val)**

Mediante esta función vamos a introducir un seleccionador giratorio. Para ello, debemos introducir la posición del centro del circulo (x, y), el radio de éste (r), las opciones de diseño (ops) y el valor que marca (val). Para ello, la función escribe en RAM el comando CMD_DIAL de 4 bytes, y el resto de los valores de entrada; todos ellos de 2 bytes.

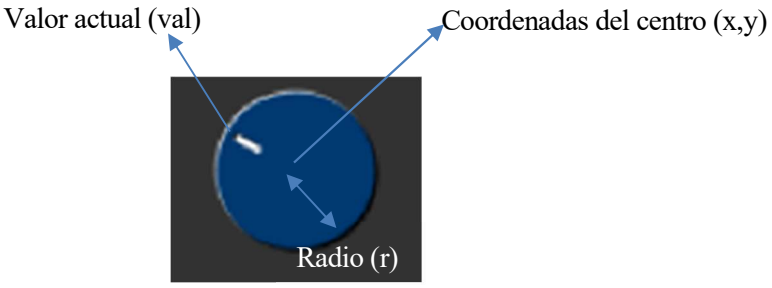

Figura 4.12: Ejemplo dial

Fuente: Archivo FT800 Programmers Guide

Al igual que como se ha explicado para la función scrillbar, el objeto dial está orientada más como regulador de una variable analógica que como indicador del valor. Por ello, también se podría crear una función que detectara la pulsación sobre el objeto y variara la variable a la que se encuentra asociada según donde se pulse.

Dentro de las opciones de diseño que se pueden emplear se encuentra lo siguiente:

- 0: es la opción por defecto. Mantiene el formato 3D (botón no pulsado).
- OPT_FLAT: desactiva el formato 3D (botón pulsado).

4.5.12 **Interruptor(x, y, w, fuente, ops, estado, cadena)**

Esta función se utiliza para introducir un interruptor con una cadena de caracteres en su interior. Para ello, debemos introducir la posición de la esquina izquierda superior del interruptor (x, y), la anchura de éste (w), la fuente de la cadena de caracteres (fuente), las opciones de diseño (ops), el estado del interruptor (estado); ya sea ON/OFF; y la cadena a introducir. Para la creación del botón se actúa de la siguiente forma: primero se escribe en RAM el comando CMD_TOGGLE, de 4 bytes. Tras esto, se escribe en RAM x, y, w, h, ops y fuente; todas ellas de 2 bytes. A continuación, se debe escribir cada letra de la cadena introducida, siendo cada una de ellas un dato de un byte. Por último, introducimos un byte de ceros para evitar interferencias y llamamos a la función PadFIFO. Al añadir la cadena, si queremos diferenciar qué aparezca cuando esté ON y qué cuando esté OFF debemos ponerlo de la forma: cadOn \xff cadOff.

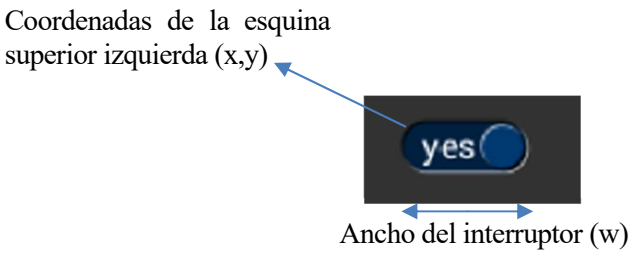

Figura 4.13: Ejemplo interruptor

Fuente: Archivo FT800 Programmers Guide

Dentro de las opciones de diseño que se pueden emplear se encuentra lo siguiente:

- 0: es la opción por defecto. Mantiene el formato 3D (botón no pulsado).
- OPT_FLAT: desactiva el formato 3D (botón pulsado).

4.5.13 **Numero(x,y,fuente,ops,num)**

Esta función se emplea para añadir números a la pantalla. Tiene un procedimiento muy parecido a la función texto, ya que ésta anterior también se podría usar para introducir números. Debemos escribir en Ram el comando CMD_NUMBER y las entradas correspondientes a la posición de inicio (x, y), fuente, la opción (ops) y el numero deseado (num).

Dentro de las opciones que podemos introducir están las siguientes:

- OPT_CENTERX: que centra el texto horizontalmente.
- OPT_CENTERY: que centra el texto verticalmente.
- OPT_CENTER: que centra el texto en ambas direcciones.
- OPT_RIGHTX: que sitúa el texto lo mas a la derecha posible.
- OPT_SIGNED: que añade signo al número introducido.

5 PRUEBAS DE FUNCIONAMIENTO

En este capítulo vamos a explicar las diferentes pruebas y comprobaciones que hemos debido hacer a lo largo del proyecto para asegurar el correcto funcionamiento de éste. Algunas resultan triviales, pero otras requerirán de un desarrollo mayor. Como prueba final, en el capítulo siguiente, se expondrá el programa de ejemplo realizado para la comprobación del correcto funcionamiento de la librería.

La estructura de la explicación de cada prueba de control es la siguiente: primero se ajuntará una tabla a modo de esquema de la prueba a realizar, incluyendo objetivos, comentarios, etc. Tras esto, se explicará más en profundidad como se ha desarrollado la prueba y que resultado se ha obtenido.

La parte mas problemática de este proyecto resulta ser la comunicación SPI, ya que cualquier pequeño error de código resulta fatal para la comunicación, por lo que la mayoría de las pruebas se centrarán en este aspecto.

5.1 Confirmación del microcontrolador

<table>
<tr><td colspan="5">Objetivos
Confirmar que el microcontrolador contiene la cantidad de memoria adecuada para la librería</td></tr>
<tr><td colspan="2">Equipo adicional
Ninguno</td><td colspan="3">Entrada externa
Ninguna</td></tr>
<tr><td colspan="5">Descripción de las actividades</td></tr>
<tr><td>Nº</td><td>Descripción de la actividad</td><td>Criterios</td><td>Comentarios de los resultados</td><td>check</td></tr>
<tr><td>5.1.1</td><td>Redacción del archivo de constantes FT800</td><td>Comprende un tercio de la memoria total</td><td>Solo ocupó una pequeña parte de la memoria del microcontrolador</td><td>Sí</td></tr>
<tr><td>5.1.2</td><td>Repetición con el modelo inmediatamente inferior de microcontrolador</td><td></td><td>Memoria casi al completo</td><td>No</td></tr>
<tr><td colspan="5">Éxito de la prueba
Sí</td></tr>
</table>

Tabla 5.1: Resumen prueba1

La primera prueba que se realizó, aunque trivial, fue comprobar que el micro seleccionado cumplía los requisites de memoria necesarios en el proyecto. Tras realizar una de las partes que mas memoria requería, la redacción del archivo constantes FT800 (ocupa aproximadamente un tercio de la memoria total de la librería). Se comprobó como un el microcontrolador finalmente seleccionado cumplía de sobra los requisitos de memoria; ya que esta parte apenas ocupó una pequeña parte de la memoria disponible del proyecto. Así mismo, se pudo comprobar

como otros modelos parecidos, pero con menor memoria, resultaban inadecuados al no poder albergar un proyecto tan pesado como éste.

5.2 Inicio de comunicación

Objetivos Identificación de la pantalla por el microcontrolador				
Equipo adicional Osciloscopio		**Entrada externa** Ninguna		
Descripción de las actividades				
Nº	**Descripción de la actividad**	**Criterios**	**Comentarios de los resultados**	**check**
5.2.1	Escritura de funciones LecEsc, Hostcommad y SendAddressRD.			Sí
5.2.2	Inclusión tiempos de espera según librería primaria	Uso de sleep_us en microsegundos		Sí
5.2.3	Declaración pines SPI (MOSI, MISO, SCLK)	Mirar Pin Layout del microcontrolador para no escojer pines críticos		Sí
5.2.4	Elegir correcta velocidad de transmisión			Sí
5.2.5	Ejecución y visualización del chipid obtenido, usando printf	Su valor debe ser 124 (7C en hexadecimal)	Valor obtenido: bytes de error	No
5.2.6	Visualización en osciloscopio de las señales SCLK y MOSI	Es conocida la forma que deben tomar dichas señales	Señales repetidas 8 veces: error en función LecEsc	Sí
5.2.7	Repetición 5.2.5 tras solventar error	Su valor debe ser 124 (7C en hexadecimal)	Valor obtenido: 1 solo byte de error	No
5.2.8	Visualización en osciloscopio de las señales SCLK y MOSI	Es conocida la forma que deben tomar dichas señales	Valor obtenido: 1 bit erroneo en los bits de ceros de despertar a la pantalla	No
5.2.9	Creación de la función HostCommadDummy para enviar los bits de ceros necesarios para despertar la pantalla			Sí

5.2.10	Repeticion 5.2.5 tras solventar error	Su valor debe ser 124 (7C en hexadecimal)	Valor obtenido igual al esperado: pantalla detectada	Sí
Éxito de la prueba Sí				

Tabla 5.2: Resumen prueba 2

Lo primero que debemos configurar es la comunicación SPI, ya que sin ella no podremos realizar la mayor parte del proyecto. Según se ha explicado anteriormente, la prueba de fuego de la comunicación es el llamado chipid, la correcta lectura de la identificación de la pantalla por el micro, por lo que el primer objetivo será ese. Para ello, se deben crear unas funciones básicas para inicializar la comunicación.

La comprobación del chipid se encuentra en la función de encendido de la pantalla (Inicia_pantalla), aquella que inicia la comunicación y realiza las configuraciones iniciales.

Para escribir la primera parte de la función, es necesario escribir las funciones LecEsc, Hostcommand y SendAdressRD, que ya se han explicado anteriormente. Así mismo, para añadir los tiempos de espera, se ha incluido en la librería la función sleep_us de utime, ya incluida en Micropython, donde podremos añadir tiempos de espera en microsegundos.

Además, hay que configurar el SPI declarando los pines SCLK, MOSI y MISO. Los dos primeros fueron colocados en los pines recomendados por el datasheet, pero para comprobar que se pueden colocar en cualquier otro siempre que no sea un pin crítico, se ha colocado el MISO en el 4. Por último, se añadió la velocidad de transmisión que, gracias a las indicaciones del profesor, fue ajustada a un nivel más bajo de lo que había sido ajustada anteriormente.

Tras esto, comenzamos a escribir la función hasta donde se encuentra la comprobación chipid. En este punto, se ha añadido un printf para poder observar los distintos chipid que se obtienen y si estos coinciden con el valor que debe ser, valor que se es conocido a priori.

Tras escribirlo todo se observó como se obtuvo una solución ilógica: se obtenían 8 bytes de error.

Ante este problema, se acudió laboratorio a intentar depurar el código, con la ayuda de un osciloscopio para ver los distintos pulsos de cada una de las señales de SCLK, MOSI y MISO, ya que se sabía a priori qué debía transmitirse en cada una teóricamente. Comenzamos por SCLK y MOSI:

Lo primero que llamó la atención de los resultados obtenidos fue que los diferentes datos que se debían enviar estaban repetidos ocho veces, por lo que pudo encontrarse el primero de los problemas: en la función de LecEsc, mediante el uso de la función spi.read, se debía añadir el número de bytes a leer; mientras que se había interpretado como el número de bits, y por eso salían multiplicado por ocho.

Una vez solventado el primer error, se volvió a ejecutar el programa y se comprobó como el chipid obtenido era el mensaje de error, esta vez solo una vez. Como no se lograba encontrar el problema, se optó por escribir en un folio los diferentes bits que se enviaban y así compararlo con lo que se debería enviar teóricamente. De esta manera se encontró el error: uno de los bytes de ceros que debían enviarse contenía un número diferente a 0. Esto sucede por haber usado la función Hostcommand para enviar estos bytes ya que, como se ha explicado en el desarrollo de la función, ésta modifica el dato para enviarlo siempre comenzando por 1, por lo que un byte de 0 se convierte en 10000000.

Para solucionar este problema, se tuvo que crear una función que enviara 3 bytes de 0 (HostcommandDummy), que es lo que se nos exige. Una vez hecho esto, y tras volver a ejecutar el programa, el chipid si dió lo que debía dar y la pantalla se encendió, por lo que se dió por solventado el primer eslabón y se continuó escribiendo el programa.

5.3 Primera pantalla

Objetivos Creación una pantalla de un color introducido como entrada				
Equipo adicional Osciloscopio		**Entrada externa** Ninguna		
Descripción de las actividades				
Nº	**Descripción de la actividad**	**Criterios**	**Comentarios de los resultados**	**check**
5.3.1	Escritura de funciones Write16, Write32, comando, comesperaFin y SendAddressWR.			Sí
5.3.2	Escribir la función New_Screen			Sí
5.3.3	Ejecutar la función con varios colores diferentes		La pantalla no reacciona	No
5.3.4	Visualización en osciloscopio de las señales SCLK y MOSI	Es conocida la forma que deben tomar dichas señales	Valor obtenido igual al esperado	Sí
5.3.5	Visualización en osciloscopio de las señales SCLK y MISO	Es conocida la forma que deben tomar dichas señales	Comunicación inexsistente	No
5.3.6	Escribir y leer un en una dirección de memoria con valor actual conocido	Al leer, se debe obtener el valor que ha sido escrito	Valor obtenido: valor anterior a lectura Detección de error en escritura: en función SendAddressWR	Sí
5.3.7	Repetir 5.3.3 tras solventar el error	La pantalla debe adoptar los diferentes colores introducidos	Valor obtenido: la pantalla realiza lo esperado	Sí
Éxito de la prueba Sí				

Tabla 5.3: Resumen prueba 3

Tras haber superado la prueba del chipid, se pudo continuar escribiendo la función, escribiendo antes las funciones necesarias para el resto: SendAdressWR, Write16 y Write32. Además, se escribió la función New_screen, con la que podrá intentarse el nuevo eslabón a superar: conseguir crear una pantalla de un color

introducido como entrada.

Para ello, primero fue necesario escribir las funciones comesperaFin y comando. Una vez escrito todo, se ejecutó el programa probando con distintos colores, pero la pantalla no reaccionaba. Tras varios intentos, se optó por volver al laboratorio a examinar lo enviado a través de las comunicaciones MOSI y MISO. La observación de la señal MOSI no presentó nada extraño, por lo que se pasó a comprobar el MISO. En ella, a través del osciloscopio, se descibrió que la comunicación no funcionaba, ya que no enviaba pulso alguno. Al no percibir error alguno en los protocolos de inicialización de la pantalla, se pasó a comprobar función por función detenidamente para depurar posibles errores.

Uno de los procedimientos de búsqueda que utilizados fue leer y escribir en una dirección de memoria de la que se conocía el contenido. De esta forma se observó cómo el micro leía correctamente el valor existente, pero no lograba escribir ninguno, por lo que el fallo debía estar en la escritura; y mas concretamente en la función SendAdressWR al ser la única función nueva que no se utilizó para la lectura del chipid, que ya se demostró superada.

Resultó acertada esta tesis ya que no se habían realizado correctamente las operaciones presentes en dicha función, por lo que la dirección de memoria que enviaba para ser escrita era errónea. Tras solventar el problema, se volvió a probar el programa con varias pantallas y todo fue a la perfección, ya estaba lista la comunicación SPI.

5.4 Comprobación táctil

Objetivos Comprobación de la correcta lectura táctil				
Equipo adicional Ninguno		**Entrada externa** Ninguna		
Descripción de las actividades				
Nº	**Descripción de la actividad**	**Criterios**	**Comentarios de los resultados**	**check**
5.4.1	Escritura de función LeePant			Sí
5.4.2	Mostrar por pantalla los diferentes valores que se obtienen al avanzar hacia la derecha (eje x positivo) y hacia la abajo (eje y positivo)	480x272: Al avanzar de izquierda a derecha, la coordenada x debe ir de 0 a 480; y al hacerlo de arriba abajo la coordenada y debe hacerlo de 0 a 272.	Las coordenadas variaban entre 0 y 1000; y la coordenada y disminuye al ir de arriba hacia abajo	No
5.4.3	Interpolación de ambas coordenadas para ajustarlos a los valores adecuados		Solución adecuada, pero con errores de precisión por redondeo	Sí
5.4.4	Comprobación de la relevancia de los	Las funciones usan valores con histéresis	Errores no	Sí

	errores de precisión	para evitar errores	relevantes	
Éxito de la prueba Sí				

Tabla 5.4: Resumen prueba 4

En cuanto a las funciones de diseño, no hubo especial dificultad. Se comenzó con lo mas fácil, dibujar una linea, prosiguiendo con rectangulos, circulos…; y así sucesivamente. Sin embargo, al configurar la lectura táctil, se descubrió cómo los valores proporcionados en la lectura de los correspondientes registros no eran admisibles, ya que tanto la coordenada x como la y variaban de 0 a 1000, cuando deberían avanzar de 0 a 480 y de 0 a 272 respectivamente, al ser la pantalla de 480x272 pixeles. Ademas, la coordenada y disminuia al ir de arriba hacia abajo, cosa que contradice el normal mapeo de pantallas. Como primera solución, se optó por una solución sencilla: interpolar los datos obtenidos para que las coordenadas que se obtenían de la función variaran según las normas deseadas. Dicha solución en cambio presenta un error ya que, al interpolar y luego redondear a valores enteros, se cometen errores. Sin embargo, dado que el resto de las funciones utilizan rangos de valores en vez de valores absolutos de lectura táctil, dicho error no tiene consecuencias demasiado relevantes.

5.5 Diseños avanzados

Objetivos Comprobación de funcionamiento de funciones de diseño de alto nivel				
Equipo adicional Ninguno		**Entrada externa** Wifi		
Descripción de las actividades				
Nº	**Descripción de la actividad**	**Criterios**	**Comentarios de los resultados**	**check**
5.5.1	Diseño de funciones de diseño de alto nivel			Sí
5.5.2	Elaboración de ejemplo para mostrar su funcionamiento, con botones y un reloj analogico que muestre la hora	Conectar a internet el micro para obtener la hora y la fecha actual para así aumentar la complejidad	La manecilla de los segunderos se detiene a los 14 segundos	No
5.5.3	Comprobación de la comunicación wifi		El micro se bloquea incluso sin conectarlo a internet: no es problema de la comunicación Wifi	Sí

5.5.4	Posible error debido a frecuencia de respuesta del micro demasiado alta	Hay error si el registro REG_CMD_READ es igual a 0xFFF	Valor del registro adecuado: no es problema de frecuencia alta	No
5.5.5	Disminución de frecuencia de escritura en pantalla	Frecuencia anterior demasiado alta: posible caso de error	Error aun existente: no es problema de la frecuencia de escritura	Sí
5.5.6	Lectura de los buffers de lectura y escritura	Cuando no están siendo utilizados, ambos deben ser multiplos de 4 e iguales	Cuando error, buffers no iguales: micro en función comesperaFin indefinidamente	No
5.5.7	Comprobación de saturación de buffers en función FIFO	El error sucedía tras saturar la memoria FIFO	Error en función	Sí
5.5.8	Repetición de 5.5.2		Funcionamiento adecuado	Sí
Éxito de la prueba Sí				

Tabla 5.5: Resumen prueba 5

A continuación, se realizó la elaboración de diversas funciones de diseño, ya de mas alto nivel como botones, teclas etc. Para poder probarlas, se diseñó un pequeño juego donde se movía una bola utilizando varias teclas de la pantalla táctil.

Se propuso conectar el microcontrolador a internet para obtener así la hora y el día actual. Para ello, fue facilitado una serie de programas a modo de ejemplo con los que podría aprender, sin entrar mucho en detalle, como funciona la comunicación. Con ello, se pudo modificar el programa anteriormente realizado añadiéndole un botón que, al pulsarlo, haga aparecer una pantalla nueva donde se muestre la hora (mediante un reloj analógico) y fecha actual. Se podrá volver a la pantalla anterior con otro botón situado en la esquina superior izquierda.

Al comenzar a diseñar la nueva pantalla y se conseguió colocar un reloj analógico. Podía comprobarse que la hora actual era la marcada porque se podía ver avanzar a la manecilla de los segundos correctamente. Sin embargo, ocurría un problema inesperado: cuando se entraba en dicha pantalla y la manecilla comenzaba a andar, a los 14 segundos se paraba. Se comprobó que no era un error de la comunicación via internet, ya que se paraba porque la comunicación con el Micro se quedaba bloqueada.

Según decía el manual, podría ser que no le diera tiempo al micro de la pantalla a responder, ya que se está utilizando un bucle infinito que se repite cada cierto tiempo, cuya frecuencia tienes que marcar tú y podría ser demasiado alta. Si fuera eso, al leer el registro REG_CMD_READ en vez de dar un valor de lectura adecuado daría 0xFFF. Se comprobó y resultó que esta no era esta la causa del error.

A continuación, se intentó realizar la escritura a una frecuencia inferior (cada 50ms), ya que estaba realizandose a una frecuencia demasiado alta (cada 0.5ms); pero tampoco solucionó el problema.

También se intentó leer los buffers de lectura y escritura, ya que si funcionan correctamente ambos debían dar un numero multiplo de 4. Sí eran multiplos de cuatro, pero resultó que, al producirse el error, ambos buffers eran

diferentes (88 y 84), por lo que la pantalla se quedaba infinitamente en la función comesperaFin, de la que solo se sale cuando ambos registros sean iguales. Además, antes de producirse el error, ambos buffers llegaban a valores próximos a la saturación; por lo que el error podría encontrarse en la función de saturación de la memoria FIFO.

En efecto así era, un error en dicha función que modificaba una variable errónea, por lo que los buffers no llegaban a saturar correctamente. Una vez solventado el error, el programa volvió a funcionar correctamente.

6 MANUAL DE USUARIO

Este capítulo sirve para crear una guía de uso de la librería, detallando cada una de las partes que debe tener el programa deseado y las funciones imprescindibles. Se añadirá un diagrama de flujo para resumir a modo de esquema lo aquí explicado. Por último, se encuentra una pequeña demo a modo de ejemplo donde se comenta el código empleado y los resultados que se van obteniendo en la pantalla.

6.1 Librerías e inicialización del SPI

Lo primero que debemos hacer al realizar un programa, es introducir las librerías a realizar. Para el correcto uso de la librería desarrollada, debemos introducir, usando el comando import, la FT800ESP32 (la librería desarrollada) y la FT800 (listado de defines y registros). Además, debemos añadir de la librería machine las clases Pin y SPI.

A continuación, debemos inicializar la comunicación SPI, creando una variable objeto de la siguiente forma:

```
"nombre_variable"=SPI(-1, sck=Pin(n1), mosi=Pin(n2), miso=Pin(n3))
```

Poniendo a gusto el nombre de la variable a utilizar, n1 el pin del micro donde ira conectado la señal de reloj, n2 el pin de comunicación MOSI y n3 el pin de comunicación MISO. Notesé que en muchos micros te muestran pines recomendados para estas funciones. La primera entrada tiene que ser -1 para así poner los pines a gusto.

A continuación, debemos crear variables objeto para los pines CS y PD, que serán pasados como parámetros para el resto de funciones.

Por ultimo, debemos inicializar la comunicación con la función:

```
"nombre_variable".init(baudrate=b, polarity=p, phase=f)
```

Siendo b la velocidad de transmisión, p la polaridad y f la fase, que dependerán del micro utilizado y el hardware de comunicación.

6.2 Inicio de pantalla

Lo primero que temenos que hacer para empezar a utilizar la librería es inicializarla, creando una variable objeto de esta de la siguiente forma:

```
"nombre_variable2"=FT800ESP32.FT_800(spi, cs, PD)
```

Donde se le tiene que pasar como parametros de entrada las variables objeto correspondiente a la función SPI y a los pines CS y PD.

Tras esto, debemos llamar a la función IniciaPantalla() para iniciar la comunicación y encender la pantalla. Esto solo se debe hacer una vez al inicio de programa.

6.3 Dibujo en pantalla

Siempre que queramos dibujar en la pantalla debemos de seguir el mismo protocolo: crear una nueva pantalla, seleccionar los colores, definir todos los objetos gráficos a representar y llamar a la función dibujar.

La creación de una nueva pantalla se hace mediante la función New_screen(R,G,B), donde se le debe introducir el contenido de cada color primario que tendrá el color de fondo de la pantalla.

A continuación, debemos seleccionar el color de los objetos, debiendo ser definido el color principal (ComColor(R,G,B)), el color Foreground (ComFgColor(R,G,B)) o el color Background (ComBgColor(R,G,B)). También, se le puede asignar un gradiente de colores mediante la función ComGradiente. Estos colores mantendrán su valor desde que son seleccionados hasta que se le cambia su valor, englobando a todos los objetos definidos en medio.

Para dibujar objetos en la pantalla, se pueden utilizar las funciones de alto nivel desarrolladas en el capítulo 4. Siempre que se cambie algo de la pantalla se deben definir todos objetos presentes, aunque no hayan cambiado.

Por último, para que todo sea dibujado en la pantalla, se debe llamar a la función Dibuja(). Ésto debe hacerse siempre que se realice algún cambio sobre lo dibujado.

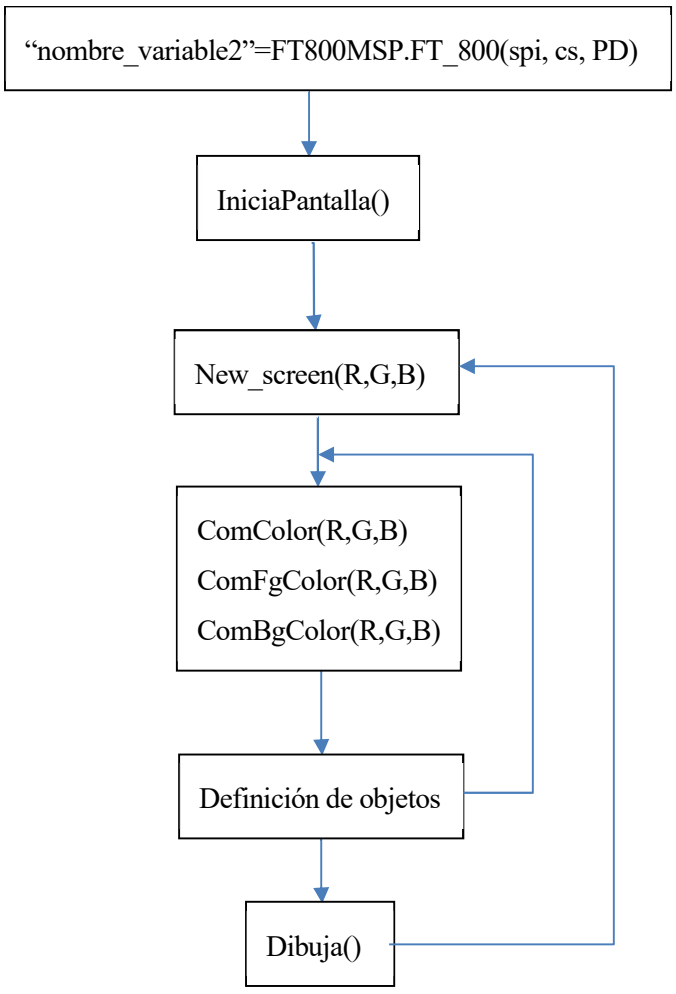

Figura 6.1: Esquema creación de proyecto

6.4 Demostración

Para la demostración del manual de usuario anteriormente explicado, se ha diseñado esta pequeña demo. El objetivo de esta va a ser crear una primera pantalla con un botón que se mantenga esperando a que sea pulsado. Una vez se pulse, aparecerá otra pantalla con un texto recuadrado. El procedimiento a seguir es el siguiente:

Lo primero que debemos hacer es insertar las librerías necesarias. Como hemos explicado, siempre que queramos hacer un proyecto empleando la librería FT800ESP32, además de esta, debemos incluir la librería FT800; donde se encuentran las constantes utilizadas para el diseño de la pantalla. También debemos incluir las clases SPI y PIN de la librería machine que se encuentra instalada por defecto en nuestro microcontrolador. Además, se ha incluido la clase sleep_us de la librería utime, que será necesaria para implementar el tiempo de espera.

```
import FT800ESP32
import FT800
from machine import Pin, SPI
from utime import sleep_us
```

A continuación, debemos inicializar la comunicación SPI. Para ello, haremos uso de la clase SPI, donde le añadiremos las entradas -1, para poder seleccionar que pines se usarán para los diferentes canales de comunicación, y los pines escogidos para las señales SCK, MOSI y MISO, mediante la clase PIN.

```
spi=SPI(-1, sck=Pin(5), mosi=Pin(18), miso=Pin(19))
```

Tras esto, debemos iniciar la comunicación añadiendo la velocidad de transmisión, la polaridad y la fase. Además, seleccionaremos los pines escogidos para las señales PD y CS

```
spi.init(baudrate=100000, polarity=0, phase=0)
cs=Pin(15)
PD=Pin(14)
```

Una vez hecho esto, podemos iniciar la pantalla. Para ello, debemos crear una variable objeto, añadiéndole como entrada las variables de las funciones SPI , CS y PD. Además, debemos usar la función Inicia_pantalla() para iniciar la comunicación y encenderla.

```
pant=FT800ESP32.FT_800(spi, cs, PD)
pant.Inicia_pantalla()
```

Tras esto, ya podemos empezar a dibujar la primera pantalla. Según como dice el manual, debemos crear la pantalla añadiéndole los colores de fondo, según la codificación RGB.

```
pant.New_screen(0x00, 0x00, 0xFF)
```

Ahora podemos añadirle los objetos a representar, en nuestro caso un texto y un botón. Antes de todo, debemos configurar el color del texto y de las letras del botón. En este proyecto solo se modificará el color principal, dejando el de foreground y background por defecto. Una vez añadido todos los objetos, usamos la función Dibuja() para cargarlo todo en la pantalla y mostrarlo.

```
pant.ComColor(0xFF, 0xFF, 0xFF)
```

```
  pant.Texto(160,150, 20, FT800.OPT_CENTERX,"Toca la pantalla para ver
equipo ganador")
  pant.Boton(150, 100, 50, 40, 20, 0, "mostrar")
  pant.Dibuja()
```

El resultado obtenido es el siguiente:

Figura 6.2: Primera Pantalla

A continuación, añadimos un bucle de espera hasta que se pulse el botón, gracias a la función TocaBoton que devuelve un 1 si esto sucede. Una vez pulsado, pasamos a la segunda pantalla.

```
  while pant.TocaBoton(150, 100, 50, 40, 20, 0, "mostrar")!=True:
      sleep_us(100)
```

Repetimos el procedimiento anterior, cambiando el color y añadiendo un texto y un rectángulo en vez de un botón. Por último, no debemos olvidarnos de usar la función Dibuja() para mostrarlo todo por pantalla.

```
  pant.New_screen(0x00, 0xFF, 0x00)
  pant.ComColor(0x00, 0x00, 0x00)
  pant.Texto(160,150, 30, FT800.OPT_CENTERX,"Real Betis Balompie")
  pant.Rectangulo(20, 150, 300, 190, 1, 0)
  pant.Dibuja()
```

El resultado obtenido es el siguiente:

Figura 6.3: Segunda Pantalla

Aquí acabaría el programa diseñado. Podemos comprobar como, una vez realizados los protocolos de inicio, la programación de la pantalla se vuelve repetitiva; basándose en crear una pantalla nueva por cada cambio que queramos hacer y añadiendo todos los objetos a mostrar. Aunque no se ha podido apreciar en esta demo, es muy importante añadir todos los objetos que se deban mostrar, a pesar de que la nueva pantalla presente objetos de la anterior sin haber variado nada de ellos.