Proyecto Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Desarrollo de una Librería en MicroPython para el Manejo de una Pantalla Táctil

Autor: Danny Martín Jiménez
Tutor: Manuel Ángel Perales Esteve

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

Sevilla, 2019

Proyecto Fin de Grado

Ingeniería de Electrónica, Robótica y Mecatrónica

Desarrollo de una Librería en MicroPython para el Manejo de una Pantalla Táctil

Autor:
Danny Martín Jiménez

Tutor:
Manuel Ángel Perales Esteve
Profesor Contratado Doctor

Dpto. de Ingeniería Eléctrica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Autor: Danny Martín Jiménez

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:

El Secretario del Tribunal

Fecha:

A mi familia

A mis maestros

Agradecimientos

Antes de iniciar el desarrollo del proyecto me gustaría dedicar unas palabras a mi familia, a mi padre, mi madre y mis dos hermanos, por todo el apoyo que me han dado durante los años de carrera, al igual que el apoyo económico que han realizado.

También me gustaría valorar a todos los compañeros que he tenido durante el grado, en especial a mis amigos, gracias a ellos he podido afrontar y superar cada obstáculo que se ha presentado en el largo camino hasta aquí.

Finalmente deseo agradecer de corazón a todo el magnífico profesorado que he podido disfrutar, y que siempre han tenido tiempo para atender mis necesidades y dudas, y los que tantas cosas me han enseñado, no solo en el ámbito de la carrera sino también en personalidad, ayudándome así a ser mejor persona.

Danny Martín Jiménez

Sevilla, 2019

Resumen

El proyecto sobre el que vamos a tratar consiste en la elaboración de una librería, un documento donde se reúnen diversas funciones con el fin de ayudar a futuros programadores para que no tengan que realizar de nuevo las funciones ya declaradas, en el entorno de programación MicroPython.

Se verá además las ventajas e inconvenientes que nos presenta el lenguaje mencionado, además de la pantalla táctil empleada sobre la que se ha trabajado y elaborado la librería.

El objetivo de este documento es por un lado servir como complemento al código desarrollado, como por otro parte explicar y desarrollar los posibles usos al igual que los motivos por los que fueron elegidos el modelo de microcontrolador y pantalla en torno a los cuales se ha elaborado el proyecto.

Abstract

The main aim of the project we are going to talk about is the creation of a library, a file where multiple functions are found with the goal of helping new programmers in the future, in MicroPython programming environment.

Also it will be see the pros and cons about the use of the programming language we have just mention, also about the tactic screen that was used.

The goal of this document is on one hand the option of being use as a supplement of the developed code. On the other hand to explain and develop the possible uses as well as the reasons why the microcontrolator and the screen that are being used in the project were chosen.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

$<$	Menor o igual
$>$	Mayor o igual

1 INTRODUCCIÓN

Hoy en día estamos muy acostumbrados a ver por todas partes pantallas táctiles, y cada vez lo analógico se va viendo superado por lo digital. Pero la mayoría de las personas desconocen cuándo y cómo surgió el uso de las pantallas de esta forma, y mucho menos a quién hemos de agradecer por tan útil invento.

Necesitamos irnos hasta la época de los sesenta, más concretamente en torno a los años 1965 y 1967, para encontrarnos con lo que sería la primera piedra para la creación de las pantallas táctiles de hoy en día. En aquellos años un inventor británico, E.A. Johnson, comienza a escribir una serie de artículos en los que describía una pantalla táctil capacitiva. Originalmente su idea consistía en aplicar esta tecnología al control aéreo, pero no logró resultados y acabo abandonando su estudio.

El siguiente paso no fue realizado hasta los setenta, cuando por fin se patenta el concepto de pantalla táctil a manos del doctor George Samuel Hurst al lograr crear la primera interfaz electrónica táctil en 1971, gracias al sensor "Touch" bajo el nombre de "Elograph"[1]. No fue hasta 1975 cuando Estados Unidos, su lugar de nacimiento, le concede la patente y empieza su desarrollo a mayor escala con la ayuda de Elographics, una empresa que el mismo había creado. A pesar de ello aún había un gran camino que recorrer pue la pantalla en aquel momento no era transparente, pero eso no la privo de ser mencionada como uno de los cien productos tecnológicos más importantes de la época en 1973. Demostrando que desde los inicios se vio el potencial que podría llegar a ser esta tecnología.

Al contrario que E.A.Johnson, el doctor Samuel Hurst siempre había tenido la intención de convertir aquella tecnología en un método para leer la información de una forma más fácil, y eso es lo que le hace sacar al mercado la primera pantalla resistiva táctil por 1982, comenzando así la carrera en el mercado de que empresa daría con la tecla de incorporar a sus productos esta tecnología.

Un año después, en 1983, sale al mercado el primer ordenador comercial con tecnología táctil en su pantalla de la mano de Hewlett-Packard bajo el nombre de HP-150, en la siguiente imagen podemos observar el anuncio que se realizó en aquel año para promocionarlo.

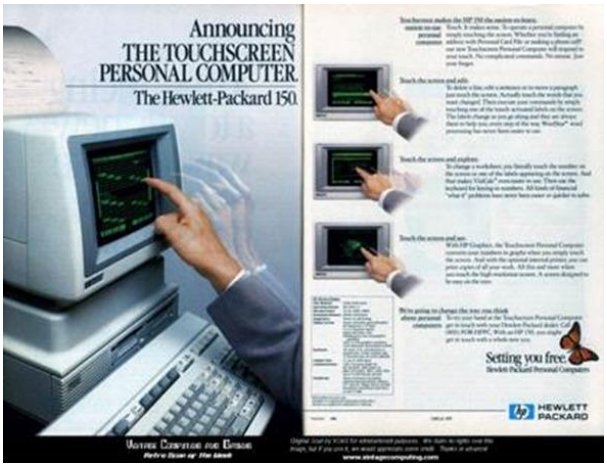

Figura 1-1: Anuncio primer ordenador táctil [1]

Este ordenador contaba con un ingenioso sistema que detectaba los objetos no transparentes en la pantalla a través de un sistema de transmisores y receptores infrarrojos. Ese sistema no podría considerarse una tecnología táctil como tal sino más bien óptica pero aun así los historiadores le dan el honor de ser la primera vez que la tecnología táctil llegaba de forma comercial, y como es lógico al ser la primera aquella versión tenía muchos

[1] Empresa actualmente conocida bajo el nombre de Elo Touch Solutions o ELO para abreviar.

inconvenientes, como la necesidad de una limpieza minuciosa, y pronto se vio desplazada y mejorada.

En los noventa, más concretamente en 1993, la empresa Apple lanzo al mercado la primera PDA táctil llamado Newton y pese a su poco éxito comercial fue la precursora de la incorporación de esta tecnología a dispositivos móviles y otros dispositivos de mano, surgiendo por ello los primeros smartphones. Además contaba con reconocimiento de escritura en la que además se incorporaba un lápiz táctil.

Figura 1-2: PDA Newton [2]

Figura 1-3: IBM teléfono inteligente [3]

En el mismo año IBM lanzó el primer teléfono inteligente, sin ningún botón y con más funciones que simplemente llamar, contaba con funciones de calendario, bloc de notas, beeper[2], y hasta una PDA y máquina de fax, todo ello mostrado en su pantalla táctil. Tras salir al mercado bajo el nombre IBM Simon fue rápidamente quitado de la venta por varios meses por problemas de software, pero tras volver estuvo en circulación por dos años, cuando dejó de venderse por culpa de su elevado precio y su batería, la cual tenía una escasa duración y no permitía al dispositivo gozar de mucha autonomía.

Tras el fracaso de su PDA Newton, Apple dejó el sector y apareció Palm, en 1996, con su serie Pilot, incorporando nuevas tecnologías táctiles que dejaban obsoletas las anteriores versiones que habían salido hasta la fecha, teniendo así un nuevo salto en esta tecnología y comenzando una carrera entre las empresas para dar al consumidor novedades que dejaran a la competencia como productos antiguos.

Figura 1-5: Windows XP Tablet Edition [5]

Figura 1-4: Nintendo DS [4]

[2] Más conocido como buscapersonas o mensáfono.

Ya entrados en el siglo XXI la empresa Microsoft lanzo en 2002 Windows XP Tablet Edition, que sería el inicio de la gran influencia de las tecnologías táctiles en pequeños dispositivos electrónicos, hasta lo que tenemos hoy en día, donde ya el foco dejo de ser tanto la pantalla para centrarse en los procesadores y el tamaño de los componentes.

Otro hecho notable que aumentó la popularidad de esta tecnología fue la salida al mercado, en 2004, por parte de la compañía de Nintendo de la Nintendo DS, a partir de la cual todas las consolas portátiles de esta compañía hasta la actualidad han presentado pantallas táctiles.

Aunque el mayor boom sin duda alguna se produjo en 2007 gracias a Apple, que al contrario que la anterior vez tuvo éxito con la comercialización del primer IPhone que rompía con el canon de los móviles hasta el momento al consistir exclusivamente de la tecnología táctil y la pantalla, sin ningún botón.

Hoy en día aún se puede apreciar lo importante que fue este acontecimiento pues todos los smartphones que se ven consisten en un modelo similar, incluso se nos hace raro ya encontrar algún dispositivo móvil con botones analógicos.

1.1 Técnologías de las pantallas táctiles

En este apartado se desarrollaran los diferentes tipos de pantallas táctiles que existen en la actualidad, así como se explicaran las diferencias y los diferentes usos de cada una de ellas.

1.1.1 Resistivas

Este tipo de tecnología táctil consiste en un diseño de varias capas, permitiendo que cuando el usuario toque una zona de la pantalla alguna capa se ponga en contacto con otra identificando así la pulsación correspondiente, gracias a que internamente se produce un cambio en la resistencia, y por tanto en la corriente eléctrica. El problema como se puede observar a simple vista es el tiempo de reacción, es decir, la respuesta es más lenta y además en un inicio incapaz de detectar múltiples pulsaciones al mismo tiempo o gestos.

Este problema se ha ido solucionando poco a poco, siendo presentadas mejoras que permiten el multitouch, y en la sensibilidad del toque, normalmente todas ellas introducidas desde el mundo de la telefonía.

Otro inconveniente llega desde el brillo, aunque no es tanto un problema sí que es necesario mencionar que pierde entorno a un 25% del brillo por la necesidad de pasar por las diferentes capas de presión.

Pero no todo son desventajas, las pantallas de este tipo cuentan con una gran resistencia al polvo y al agua, además de un coste de producción muy pequeño, permitiendo así que gocen de una gran popularidad y que estén presentes en multitud de aparatos electrónicos.

1.1.2 Capacitivas

Esta tecnología se basa en un campo electrostático que se ve distorsionado cuando un elemento conductor eléctrico, como el dedo de una persona, entra en contacto con la superficie de la pantalla, que es un cristal funcionando como una capa de aislamiento eléctrico recubierta a su vez por un conductor transparente.

A primera vista podemos encontrar uno de las desventajas de este modelo, y es que no se detecta pulsación si la pantalla es tocada por un elemento no conductor como suelen ser la mayoría de lápices táctiles del mercado, pensando más en el tipo anteriormente mencionado.

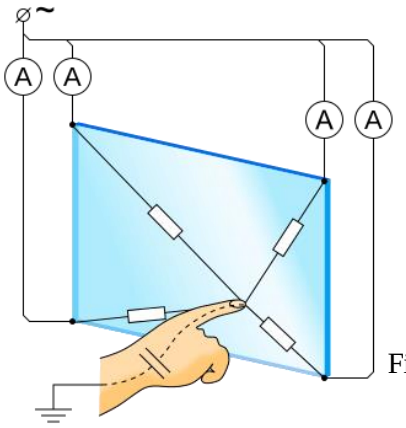

Figura 1-6: Funcionamiento pantalla capacitiva [6]

La posición es calculada de diferentes formas según el diseñador de la pantalla que estemos utilizando, pero en general tras identificar el lugar del toque se envía a un controlador para que lo procese. Esto permite al usuario gozar de una mayor precisión pero no cuenta con un detector de presión por lo que todos los toques serán interpretados iguales.

Al contrario que en las pantallas resistivas, estas pantallas permiten una mayor calidad y brillo, además del multitouch, y cuentan con la ventaja de una detección y respuesta mucha más rápida y precisa.

1.2 Repercusión de las pantallas táctiles

Como ya hemos visto lo que empezó como una alocada idea se convirtió con el paso del tiempo en uno de los inventos más famosos de la última época, y no es para menos, ahora mismo las pantallas táctiles están presentes en cada día de nuestra vida, desde tabletas, móviles hasta en supermercados y bancos.

El efecto que han tenido es bastante grande, siendo su uso bastante común en países del primer mundo, como reconoce el famoso psicólogo Andrew Przybylski en un estudio realizado para la universidad de Oxford sobre la psicología de la ciencia. "Las pantallas digitales son ahora una parte intrínseca de la infancia moderna, de la generación de los llamados nativos digitales" [7] con esta afirmación el psicólogo refleja como las nuevas generaciones tienen tan interiorizadas esta tecnología que les costaría pensar en un mundo sin su uso.

Además de los efectos positivos y su uso tan diverso, también encontramos una influencia negativa debido al uso excesivo que se realiza, es decir, no produce efectos nocivos y perjudiciales en los usuarios salvo que se realice un uso inadecuado.

Las lesiones más comunes conocidas suelen estar centradas en los dedos, aunque también puede llegar a afectar a la vista por tiempos de exposición altos así como mala iluminación del entorno que obliga a forzar más de lo que se debería la visión.

1.3 Justificación del trabajo

Las librerías en programación son esenciales para agilizar los códigos y hacerlos mucho más interpretables para otras personas distintas al programador, también reducen la complejidad al encargarse de un nivel más bajo del que luego tiene el usuario.

Las pantallas táctiles ya se han mencionado las múltiples utilidades y usos que tienen, pero para poder utilizarlas es necesario contar con ese pilar sobre el que aplicar las ideas y deseos de uso que se tengan, por ello se hace

necesario disponer de una librería, más aún en Python, al ser un lenguaje abierto que cualquiera puede usar y ayudarse de todas las librerías que aportan sus usuarios.

1.3.1 Solución propuesta

Para ayudar a futuros estudiantes que deseen aprender Python y sus derivados, así como a la comunidad de este lenguaje se decidió elaborar una librería con la que poder manejar alguna pantalla táctil.

De esta forma los que deseen trabajar con esta no necesite perder tiempo mirando internamente como realizarlo, y solo deba usar funciones básicas y fáciles de emplear, dejando más tiempo para que puedan elaborar mayores aplicaciones.

Además de esta forma se podrá acercar a personas ajenas a todo este amplio campo al encontrarse más facilidades y no desesperarse al no tener ayuda ni un punto de partida en el cual apoyarse.

2 HARDWARE

Empezaremos hablando de los elementos externos que se han seleccionado para la realización de este proyecto, en cada punto no solo se procederá a comentar el elemento en cuestión y en disponer de una imagen para ayudar a su reconocimiento, sino también los motivos de su selección, debido a que cualquier elección que se realice afectará enormemente en el desarrollo.

Esto hará que el proyecto no sea de uso inmediato para quienes deseen tomar otros componentes a los aquí seleccionados, pero no por eso este documento será ineficiente para ellos, pues encontraran este trabajo como guía de cómo proceder.

2.1 Microcontrolador Adafruit ESP32 feather board

El microcontrolador es el principal componente del trabajo, siendo el encargo de permitir el uso del sistema operativo de MicroPython y además guardar en memoria la librería que se desarrollará.

Por este motivo su elección es un asunto muy importante, teniendo un gran abanico de posibilidades en el mercado, aunque nos vamos a decantar hacia el microcontrolador de la empresa Adafruit.

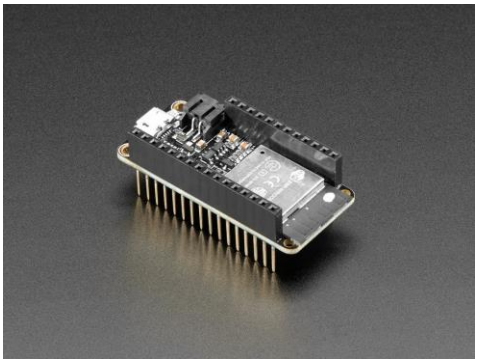

Figura 2-1: ESP32 Adafruit [8]

2.1.1 Motivos del empleo

Los motivos de esta elección son variados, para comenzar el dispositivo necesitado debía de tener una gran memoria y capacidad de procesado para ser apto para el software requerido, el siguiente punto es el deseo de gracias a la librería que se desarrollara dar pie a muchos proyectos diferentes, para esto necesitábamos unas prestaciones altas, como de comunicaciones bluetooth, internet…

Todo esto nos lo otorga el producto ESP32, mejora del tan usado ESP8266, incorporando una mayor velocidad y el ya mencionado módulo WiFi.

Otras características que incorpora esta placa son su gran rango de conexionado, al portar 3 UARTS, 12 entradas ADC, PWM en todos los pines, 3 SPI, entre otros; poseer doble núcleo y un amplificador analógico de ruido ultra-low.

2.2 Pantalla FT800

La pantalla táctil es un producto de la empresa FTDI Chip, la cual tiene una amplia gama de diferentes pantallas, además de otros muchos servicios.

En este ocasión se ha decidido seleccionar la versión FT800, la cual se subdivide en dos diferentes pantallas, una más reducida de 3.5 pulgadas y otra mayor de 5 pulgadas.

Para los ejemplos y demostraciones se utilizara la pantalla más pequeña, pero todo se realizara pensando en aplicaciones para ambas pantallas, a pesar de las pequeñas diferencias que existen entre ellas.

Figura 2-2: Pantallas táctiles serie FT800 [9]

2.2.1 Diferencias entre las versiones

Aparte del tamaño entre ambas pantallas existen unas leves diferencias que forzaran que sean programadas de formas ligeramente distintas.

La principal diferencia es la posición de los bits de los colores, al estar en la pantalla de 3.5 pulgadas intercambiados intentando dar una mayor comodidad al usuario, mientras que ya en la de 5 pulgadas ese intercambio se eliminó al no mostrar mejoras significativas.

2.2.2 Motivos de su empleo

Se ha optado por elegir esta pantalla entre todas las existentes en el mercado, tanto de la propia empresa como de otras muchas, por sus prestaciones a un reducido coste. Entre todas las prestaciones que se nos presentan tenemos unas guías elaboradas por la compañía de ayuda para el manejo y programación de la pantalla, así como funciones internas como poseer audio, pantalla de buena resolución, 320x240 pixel, y una gran variedad de colores en el rango de 262k diferentes.

2.3 Presupuesto

Gracias a que el software, del cual se informará en el siguiente punto, es de código abierto el coste del proyecto será exclusivo de la parte del hardware, donde solo hay que destacar las ya mencionadas pantalla y microcontrolador. Puesto que otros elementos como cableado es despreciable, y los embellecedores son completamente ajenos al proyecto, con la única función de hacer más vistosa la pantalla táctil.

Elemento	Coste
Pantalla	51.05€
Microcontrolador	19.97€
Total	71.02€

Tabla 2-I: Presupuesto del proyecto [10]

Como se ve en la tabla, este proyecto es bastante económico para todas las funciones que estos dispositivos nos aportan, más aún tras la realización del proyecto ya que se puede usar de base para multitud de ideas empleando una pantalla táctil y MicroPython, como ya se mencionaran más adelante, en el apartado de conclusiones.

3 SOFTWARE

Además del hardware, es necesario especificar los programas que se han empleado para la realización de este proyecto, aunque al igual que ocurría con el apartado físico debemos recordar que se pueden usar muchos otros programas diferentes, al existir un amplio repertorio en internet.

Los programas que se enumeraran y describirán a continuación han sido seleccionados en base a su comodidad, interfaz y su accesibilidad, además de que claramente cumplan el objetivo deseado.

3.1 Python

Python es el lenguaje de programación que vamos a emplear por lo que es obligatorio la necesidad de instalarlo, siendo así uno de los pocos programas que no cuentan con otras posibilidades pero tiene la ventaja de que funciona en todos los sistemas operativos del mercado.

Figura 3-1: Logotipo de Python [11]

Este lenguaje de programación es multiparadigma e interpretado, siendo su principal característica y diferencia con otros lenguajes conocidos, como puede ser el caso de C o Java, el ser dinámico, es decir, su compilación línea a línea del código en cuestión, además del uso de palabras antes que el uso de símbolos.

Esto último es consecuencia de su pensamiento y animo de que los códigos sean legibles, fáciles de entender y para nada encriptados, como podría ocurrir en caso de usar símbolos que impedirían a los usuarios menos avanzados interpretar partes ya realizadas por otros usuarios.

Como ejemplo de esa mentalidad también tenemos la ausencia de llaves, ya que podría acabar siendo un proceso realmente complicado discernir donde empiezan y acaban estas, para solucionarlo se emplea la tabulación como muestra de pertenencia a una condición anterior (como el caso de condicionales o bucles).

Por último hemos de hacer mención a la gran importancia que tiene este lenguaje en el campo de la informática, ya que es el más usado últimamente para las aplicaciones de inteligencia artificial, en especial gracias a su comunidad, la facilidad de aprendizaje y lo potente que es, es decir, que permite realizar gran multitud de tareas complejas.

3.1.1 Instalación

El proceso de instalación es muy sencillo y guiado, existiendo además numerosos tutoriales en Internet de ayuda

en caso de existir alguna complicación. Basta con ir a la página oficial de Python[3], y como se ve en la imagen desde ahí podremos descargarnos la versión más reciente sin problema alguno.

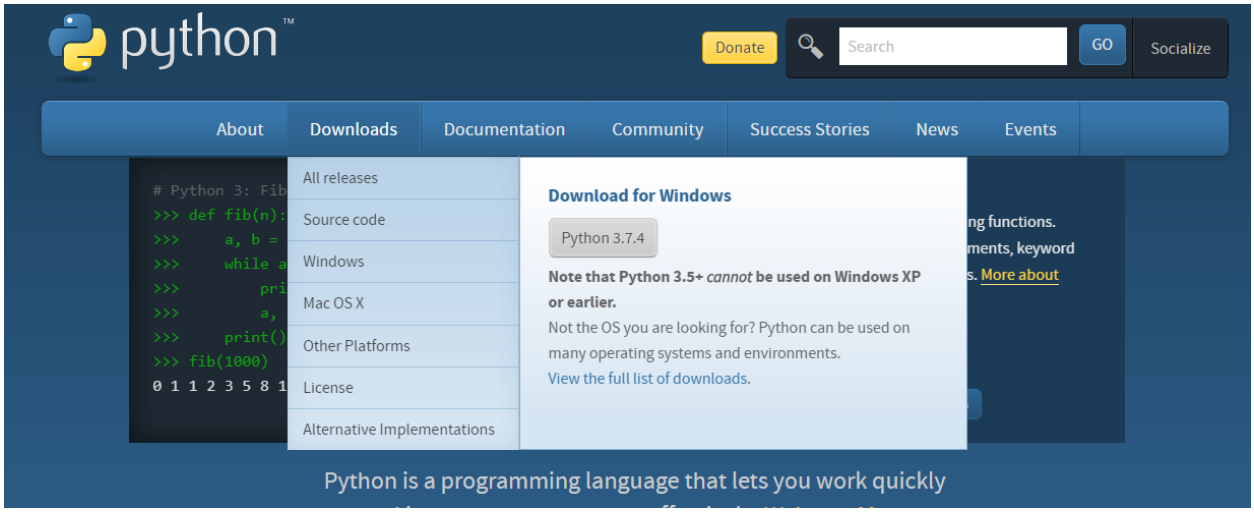

Figura 3-2:Página principal Python [12]

3.1.2 ESPTool

Además, dentro del programa tendremos que instalar ESPTool, un archivo realizado en Python que viene integrado junto a otros muchos archivos en una carpeta interna del programa para permitir que el usuario instale los complementos que desee.

Este programa interno será el encargado de incorporar el firmware de MicroPython al microcontrolador seleccionado, según un proceso que será explicado más adelante en el apartado de Montaje.

Su proceso de instalación es bastante simple, solo necesitaremos entrar en la pantalla de comandos del ordenador e introducir la siguiente instrucción dentro del terminal de Python, o del centro de comandos en caso de haber establecido su interacción durante la descarga:

```
pip install esptool
```

En caso de que no viniera el archivo en Python se puede descargar desde la siguiente página: https://github.com/espressif/esptool

3.2 MicroPython

Para ajustarse a las nuevas necesidades un físico australiano, Damien George, implemento el software de Python para hacer una versión optimizada que pudiera ser utilizada en dispositivos electrónicos de pequeño tamaño, como lo son los microcontroladores, manteniendo sus señas de identidad como lo son el código fácil de

[3] https://www.python.org/

interpretar, su programación línea a línea…

Su principal característica radica en su reducido tamaño[4], siendo este la capacidad mínima del hardware que hemos de adquirir, al perder prestaciones con respecto a su versión para ordenadores, que en un microcontrolador no se emplearían de forma habitual.

Por este motivo algunas funciones que venían definidas en librerías en Python cambiaran ligeramente o no estarán disponibles aquí, salvo que se incorporen nuevas librerías, en caso de que terminen siendo necesitadas por el usuario.

3.2.1 Instalación

Al contrario que antes, no tendremos que realizar ninguna instalación en nuestro ordenador, pues solo necesitaremos descargar un paquete de instalación con el firmware para posteriormente poder instalarlo dentro del microcontrolador seleccionado.

Para obtener el archivo necesario tendremos que irnos a la página oficial, y de ahí acceder a la pestaña de descargas. En la página principal veremos una función muy buena, como es la programación online para practicar y comprobar sin necesidad de meternos en la placa que el código es correcto.

Aquí hemos de hacer un pequeño inciso para recalcar un aspecto importante, como podemos observar el firmware solo está disponible para ciertas placas, lo que nos reduce bastante la elección de un microcontrolador, aunque se espera que un futuro cada vez esté disponible para un mayor espectro.

3.3 uPyCraft

Este programa será el encargado de establecer la comunicación con la placa, además del lugar donde iremos programando las funciones de la librería.

Su elección radica en una interfaz simple y cómoda, también incorpora revisión de sintaxis para identificar posibles errores de ese tipo y poder solucionarlos sin necesidad de tener que buscar por todo el código elaborado.

Otra ventaja respecto a otros programas que podemos encontrar por internet es que no es necesario instalación, lo que facilita su portabilidad al contrario de los que requieren instalación previa.

3.4 Drivers de Silabs

La función de estos drivers es permitir que el ordenador pueda conectarse e identificar el puerto por el que se conecta la placa seleccionada, en este caso recordemos que es ESP32.

Para poder obtenerlos hemos de ir a la página principal[5] y allí explorar dentro del menú de productos los drivers USB to UART[6], donde deberemos descargar la versión que deseemos, teniendo cuidado con no elegir alguna versión en fase beta, ya que podría tener algunos problemas.

[4] 1108 KB
[5] https://www.silabs.com
[6] https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

En nuestro caso hemos tomado en específico la versión 6.7.6 para Windows.

4 DESARROLLO

En este apartado se procederá a desarrollar los pasos que se han ido siguiendo para la realización del trabajo, desde la instalación del firmware hasta las pruebas realizadas de funcionamiento de la librería programada, pasando por las conexiones y las funciones creadas.

4.1 Montaje

Para saber cómo se han de conectar el microcontrolador a la pantalla táctil primero hemos de informarnos de ambos a través de sus respectivos datasheets. Este paso es muy importante ya que nos revelará información crucial para evitar quemar componentes internos de los dispositivos y con eso ayudar a que no terminen estropeados y que respondan de la manera esperada.

En este caso, apreciamos que los voltajes de salida del microcontrolador llegan hasta una tensión máxima de 3V lo que nos facilita las cosas ya que la pantalla seleccionada trabaja hasta un rango máximo de 5V.

Además contamos con la ayuda de que para facilitar el proceso, la pantalla táctil menciona físicamente los nombres de los diferentes puertos permitiendo que sean identificados más rápidamente y no se den confusiones con ellos.

4.1.1 Configuración de pines E/S

4.1.1.1 PD

El pin PD hace referencia a "power down" y es empleado para determinar si la comunicación con la pantalla esta activada o no, es decir, es el encargado de despertar a la pantalla táctil para que reciba y actualice información de sus otros puertos.

Este pin será configurado de solo salida, y no tendremos ninguna limitación de cuál seleccionar en el microcontrolador. Será decisión del usuario elegir un puerto para su comunicación, que posteriormente habrá que informar su número a la función que se encarga de establecerlos correctamente.

4.1.1.2 CS

El pin CS hace referencia a "Chip Select" y es empleado de habilitar o no los buses hacia los que se envían los datos dependiendo si está a un valor alto o no. Cada vez que se quiera leer o mandar datos hacia la pantalla táctil habrá de dejarlo a nivel bajo, y al finalizar volver a bloquear los buses en cuestión por seguridad

Al igual que el caso anterior, será configurado como un puerto de solo salida, sin ninguna limitación extra a la hora de seleccionar el pin, aunque por comodidad se recomienda que sea contiguo al anterior.

4.1.1.3 SCK

El pin SCK hace referencia a "Soft Clock", es el encargado de regular la velocidad con la que se transmite cada bit en ambos sentidos, siendo de vital importancia para el correcto funcionamiento, pues de estar mal configurado algunos paquetes de datos podrían perderse o identificarse erróneamente.

El microcontrolador tiene un puerto específico para esta función de comunicación, por lo que el usuario no tiene opción como en los anteriores casos, aunque sí que tiene libertad para indicar la velocidad de comunicación

además de otros parámetros.

4.1.1.4 MOSI

Es uno de los pines por los que se comparte información entre los dispositivos, en específico, este puerto transmite datos del microcontrolador a la pantalla táctil, como su propio nombre indica, "Master Output Slave Input".

Por tanto es un puerto de salida de nuestra placa, y como pasaba anteriormente con el SCK tiene un pin en específico destinado a tal tarea.

4.1.1.5 MISO

Junto a MOSI son los únicos pines por los que circula información, y al contrario que el anterior este se encarga de llevar los datos de la pantalla táctil al microcontrolador, siendo por tanto el primer y único pin de entrada que vamos a tener configurado en nuestro dispositivo. Su nombre significa: "Master Input Slave Output".

Como pasaba con el MOSI, el MISO también dispone de un pin específico para la tarea, siendo imposible destinar otro, aunque eso no significa que no dábamos configurarlo como solo entrada por seguridad.

4.1.2 Conexionado

Ya hemos mencionado anteriormente que tan solo 2 de todos los pines que hemos de conectar a la pantalla son elegibles por el usuario, el resto están completamente definidos y resulta imposible su conexión a otros puertos.

Quedando por tanto una conexión similar a la que se muestra a continuación a modo de ejemplo:

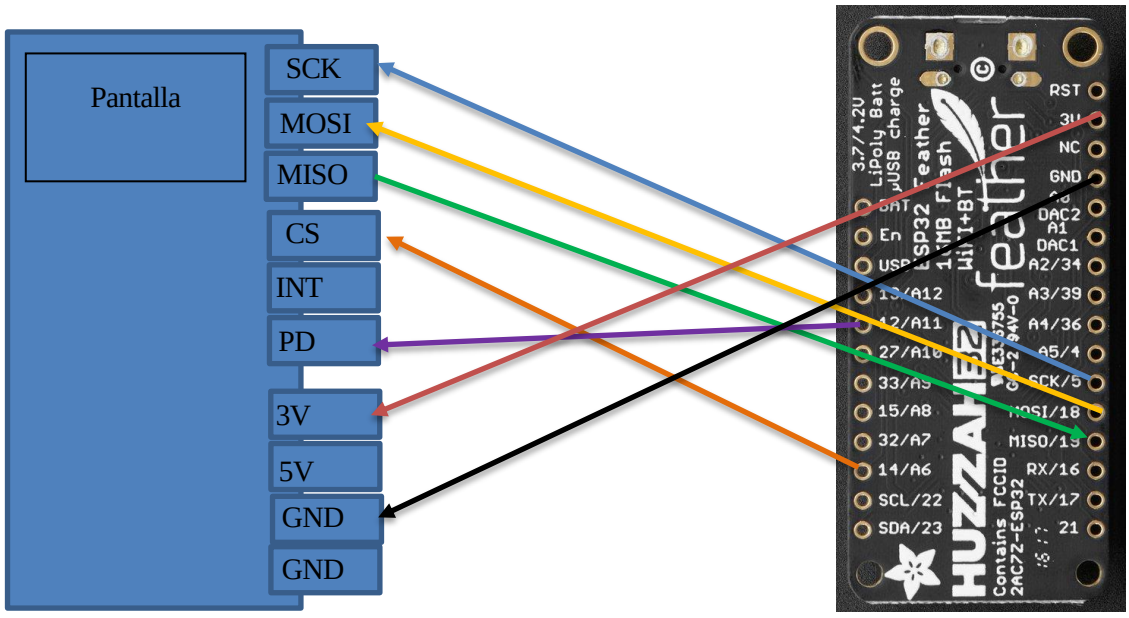

Figura 4-1: Placa ESP32 [13]

En este caso los pines PD y CS que pueden ser seleccionados han sido conectados a los puertos 12 y 14 respectivamente. El caso del pin INT que presenta la pantalla podemos omitirlo, puesto que su función no la

vamos a emplear en este proyecto y por tanto no tenemos obligación de conectarlo a ninguno de los pines del microcontrolador.

4.1.3 Instalación del firmware

Antes de comenzar a programar hemos de proceder a introducir MicroPython en el microcontrolador seleccionado, para ello seguiremos los siguientes pasos, teniendo en cuenta que pueden verse afectados por posibles cambios de nombre en el archivo que contenga el firmware como por el puerto por el que ha sido conectado al ordenador.

Primero abriremos una pantalla de comandos, la forma de abrirlo dependerá del sistema operativo con el que se esté trabajando. A continuación, si se han realizado correctamente la instalación de Python y ESPtool tan solo tendremos que introducir la siguiente línea de comandos, respetando la dirección del puerto a la que se ha introducido el microcontrolador.

```
esptool.py –port /<insertar dirección del puerto> erase_flash
```

Con esto eliminaremos toda la memoria flash que tuviera anteriormente, es recomendable hacerlo incluso cuando el dispositivo es nuevo para asegurarnos una posterior instalación sin problemas.

Vamos a comentar brevemente el formato de dirección del puerto existente a los sistemas operativos más comunes, Windows y Linux. En el primero bastaría con mirar en la pestaña de dispositivos que puerto está ocupado e introducir /COMx siendo x el número del puerto en cuestión; el segundo corresponde a la estructura /dev/ttyUSBx, necesitando mirar previamente usando el cmd el número.

Tras este paso, procedemos a escribir la memoria flash con el entorno de programación deseado, teniendo su respectivo archivo (ver sección de Software) descargado y guardado en una localización conocida.

```
esptool.py –chip esp32 –port /<dir. puerto> write_flash –z 0x1000 <dir.archivo y nombre de este>.bin
```

Tras esperar un determinado tiempo se habrá instalado el firmware en el dispositivo y ya podremos comenzar a trabajar con ello, aunque es recomendable realizar alguna prueba de que todo funciona correctamente, para ello abriremos uPyCraft y tras conectar con el dispositivo probaremos algunos comandos de Python que se encuentren disponibles en MicroPython.

4.2 Estructura

Para conseguir tener una librería más ordenada y fácil de manejar se ha decidido diferenciar entre dos tipos de funciones, las funciones creadas con el fin de ser usadas por el usuario y aquellas que se han realizado para evitar repetir numerosas veces los mismo pasos y sirven al creador para reducir el código además de aportar enormemente en funciones de debugging.

Además de lo mencionado previamente a las funciones se enlistarán una serie de constantes que tendrán valores

como las direcciones de registro de la pantalla, sacados de la guía de programador[7], teniendo en cuenta las diferencias existentes entre las pantallas de 3.5 pulgadas y la de 5 pulgadas.

En Python como no existen constantes a las que no se pueda modificar su valor hemos de advertir al usuario que no emplee nombres iguales, ya que sobrescribiría el valor y pondría en riesgo el sistema.

4.2.1 Tabla de operadores

En todos los lenguajes de programación existen una serie de operadores destinados a agilizar las funciones más habituales que emplean los usuarios, Python aunque busque tener un código fácil de interpretar y nada enmascarado también los usa.

Debido a esto, se enunciaran a continuación los operadores empleados en la elaboración de la librería así como su significado, desde los más comunes y obvios hasta los que puedan ser complicados de entender.

Operador	Ejemplo	Función que realiza
+	X+Y	Realiza la función aritmética de suma entre los términos X e Y
-	X-Y	Realiza la función aritmética de resta entre los términos X e Y
*	X*Y	Realiza la función de multiplicación entre los dos términos empleados
/	X/Y	Realiza la función de división entre X e Y, dando como resultado el cociente de la operación.
&	X&Y	Operación que realiza la función lógica AND, no tiene sentido en números en base decimal, solo en base binaria.
|	X|Y	Operación que realiza la función lógica OR, no tiene sentido en números en base decimal, solo en base binaria.
>>	X>>Y	Operación que realiza en X un desplazamiento de Y bits hacia la derecha. Solo tiene sentido en base binaria, los bits que salen del rango se pierden.
<<	X<<Y	Realiza la operación de desplazamiento en X de Y bits hacia la izquierda. Solo tiene sentido en base binaria, se incorporan tantos 0

[7] https://www.ftdichip.com/Support/Documents/ProgramGuides/FT800%20Programmers%20Guide.pdf

		como bits desplazados.
<	X<Y	Operación lógica que devuelve TRUE en caso de que X sea menor que Y, y FALSE en caso contrario
>	X>Y	Operación lógica que devuelve TRUE en caso de que X sea mayor que Y, y FALSE en caso contrario.
==	X==Y	Operación lógica que compara el valor de X e Y, dando como resultado TRUE si tienen el mismo valor, y FALSE en caso contrario.
! =	X!=Y	Operación lógica que compara el valor de X e Y, dando como resultado TRUE si tienen valores distintos, mientras que da FALSE si tienen el mismo valor
| =	X|=Y	Forma reducida de realizar la operación OR: X=X|Y

Tabla 4-I: Operadores empleados

4.3 Códigos

Como ya se ha mencionado anteriormente, la estructura del código se dividirá en dos grandes bloques, las internas serán funciones más destinadas a configurar variables internas de la pantalla y al envío y recepción de datos; mientras que las funciones externas se centraran usualmente en aquellas funciones de alto nivel que tengan consecuencias visuales en la pantalla táctil, de tal manera que si el usuario decide realizar algún cambio en estas no se encuentre con un lenguaje de bajo nivel que pueda darle problemas.

4.3.1 Declaración de constantes

Previo al inicio de crear las funciones hemos de declarar todas las constantes que se vayan a necesitar, pudiendo identificar varios grupos de variables en esta sección.

4.3.1.1 Registros

En este grupo se encuentran definidos la posición en memoria de la pantalla de los diferentes parámetros importantes que componen su memoria fija, y que durante el uso puedan ser modificados o sean necesarios en algún momento.

4.3.1.2 Comandos

La pantalla táctil está programada para que al recibir ciertos valores realice determinadas acciones concretas, estas son llamadas los comandos que regulan las acciones que posteriormente se ejecutan. Su valor viene definido en la guía de programador que nos ofrece el fabricante para las pantallas de una misma serie.

4.3.1.3 Opciones

En algunas funciones de las que se realizaran existirán diversas maneras de querer ejecutar una misma instrucción, por ejemplo en lugar de definir el vértice superior izquierdo definir el punto central de un elemento. Para ello se han creado las siguientes instrucciones, sacadas de un ejemplo para Arduino[8], estas al contrario que las anteriores sí están pensadas para ser empleadas por el usuario.

4.3.1.4 Apoyo musical

Estas constantes son completamente opcionales y su única función es servir de apoyo al usuario para introducir ciertos valores, al contrario que antes no son comunes para la mayoría de funciones, al estar destinadas solo para aquellas que empleen el altavoz.

Al altavoz hay que determinarla la nota y el tono que se quiere por medio de valores numéricos, como eso es contra intuitivo estas definiciones se encargan de solucionar el problema.

4.3.2 Funciones internas

4.3.2.1 Envió y recepción de 1 byte

```
HAL_SPI_ReadWrite(data)
```

Se encarga de enviar un valor entero menor de un byte de longitud y además recibir un valor entero menor de un byte de longitud, tras recibirlo lo transforma de binario a entero y devuelve ese valor.

En caso de que solo se quisiera mandar se recibiría un 0 simbólico.

- Data: valor entero que se quiere enviar, su rango posible es de 0 a 255.

scFTo.HAL_SPI_ReadWrite(18)

4.3.2.2 Funciones de activación y desactivación de los pines de salida

```
HAL_SPI_CSHigh()
HAL_SPI_CSLow()
HAL_SPI_PDhigh()
HAL_SPI_PDlow()
```

Básicamente se encargan de poner la salida del correspondiente pin, ya sea el configurado como chip select o power down a un valor alto o bajo según nos interese.

Estas funciones podrían ser quitadas y simplemente emplear en su defecto la función definida como value de la librería, declarada anteriormente, machine, pero hacerlo de esta manera permite identificar mejor cuando esta a valor alto y cuando a bajo, además de que el usuario no tendrá que cargar dicha librería al ya estar incluida en la nuestra.

[8] https://www.ftdichip.com/Support/Documents/AppNotes/AN_275_FT800_Example_with_Arduino.pdf

scFTo.HAL_SPI_CSHigh()
scFTo.HAL_SPI_CSLow()

scFTo.HAL_SPI_PDHigh()

scFTo.HAL_SPI_PDLow()

4.3.2.3 Delay

```
DELAY ()
```

Se encarga de tomar de la librería utime la función de pausa, la cual nos permite dormir el microcontrolador por los milisegundos que le pasemos como referencia a la función, en este caso deseamos que sean 800ms, y todas las veces que se llame a esta función será el mismo tiempo de espera.

scFTo.DELAY()

4.3.2.4 Envio de dirección para escritura

```
SENDADDRESSWR( Memory_Address)
```

Esta función se encarga de mandar una dirección sobre la que posteriormente se escribirá un valor, para ello según se especifica en la guía de programador de la pantalla táctil los bits que hemos de enviar se corresponden con la siguiente forma: 0000 0000 10 seguido de la dirección de memoria deseada, según se explica en la página 17 del citado documento.

Se logra a través de un "and" y un "or" contiguos, forzando el primero a conseguir un valor nulo en el segundo bit y con el segundo obligando a tomar un valor alto al primer bit sin afectar a los demás, como se puede ver mejor a continuación.

$$Bit: xxxx\ xxxx\ and\ 1011\ 1111 (0xBF) = x0xx\ xxxx$$
$$Bit: x0xx\ xxxx\ or\ 1000\ 0000\ (0x80) = 10xx\ xxxx$$

- Memory_Address: dirección de memoria en la que se quiere escribir posteriormente un determinado valor. Su valor está limitado a 8 bytes y puede ser pasado tanto en hexadecimal como un decimal.

scFTo.SENDADDRESSWR(1081344)

4.3.2.5 Envio de dirección para lectura

```
SENDADDRESSRD( Memory_Address)
```

Igual que con la anterior función, en esta ocasión queremos enviar una dirección para poder leer el valor que hay presente en ella, para ello hemos de seguir la guía que nos proporciona la empresa de la pantalla, donde nos indican que hemos de indicar que queremos leer a través de forzar unos bits a 0, antes de la dirección deseada.

Esto lo logramos realizando una función "and" con el valor hexadecimal de 3F, el cuál posee los primeros 2 bits nulos y el resto a 1 lo que permite eliminar el valor de los dos primeros sin afectar al resto de componentes, que indican la dirección deseada en la memoria.

- Memory_Address: dirección de memoria en la que se quiere leer posteriormente su valor. Su valor está limitado a 8 bytes y puede ser pasado tanto en hexadecimal como un decimal.

---

scFTo.SENDADDRESSRD(1081344)

4.3.2.6 Escritura

Una vez hemos avisado a la pantalla si deseamos escribir hemos de mandar los valores deseados, para ello se usa la función ya explicada de HAL_SPI_ReadWrite, pero esta solo puede mandar como mucho un byte, por lo que hemos de configurar hasta tres funciones nuevas que se encarguen de recibir unos bits determinados y dividirlos en grupos de bytes para pasárselos a dicha función, recordando que primero habremos de enviar los menos significativos.

```
WRITE_32( SPIValue32)
WRITE_16( SPIValue16)
WRITE_8( SPIValue8)
```

Estas funciones se encargan de ir enmascarando el valor recibido para que siempre se envíen el límite de 8 bits, 1 byte, impidiendo así que por error busquemos enviar más bits de los posibles al mismo tiempo y se acabe produciendo una pérdida de información.

- SPIValue32: valor numérico en hexadecimal o decimal de un rango de hasta 32 bits, es decir, 4 bytes.
- SPIValue16: valor numérico en hexadecimal o decimal de un rango de hasta 16 bits, es decir, 2 bytes.
- SPIValue8: valor numérico en hexadecimal o decimal de un rango de 1 byte.

---

scFTo.WRITE_32(1081344)

scFTo.WRITE_16(500)

scFTo.WRITE(46)

4.3.2.7 Lectura

Al igual que en la escritura, cada vez que leamos debemos saber que es lo que esperamos recibir, e ir incorporando en una variable todos los grupos de bytes según su orden de importancia, en el caso que recibamos más de 8 bits.

Como la función es de enviar y recibir, hemos siempre de mandar a la hora de recibir un valor, por ello se envía un valor residual de 0 que no afectará en nada.

```
READ_32()
READ_8()
```

Como podemos apreciar, al saber que nunca recibiremos valores de dimensiones diferentes a 4 bytes o 1 byte

solo tendremos que realizar dos programas, teniendo en cuenta en el primero la consecuencia de que primero se reciben los bits menos significativos, y que luego tendremos que ir desplazando los más significativos sin perder el valor de los anteriores, de ahí que se realice un "or" con valor de 0 en aquellos bits ya recibidos para así mantener su valor.

---

scFTo.READ_32()

scFTo.READ_8()

4.3.2.8 HostCommand

```
HostCommand( Host_Command)
```

Para enviar comandos hemos de respetar el formato establecido por la pantalla, en este caso necesitamos mandar un total de 3 bytes, el primero de ellos ha de tener su bit más significativo a 0 y el contiguo a este a 1. Esto se logra con la máscara establecida de un "and" con 0x3F y un "or" con 0x40, el primero nos asegura un 0 para los dos bits más significativos sin influir en el resto, mientras que con el segundo conseguimos que el sexto bit acabe obteniendo el valor deseado de 1.

Los dos bytes restantes que hemos de mandar se corresponden a los determinados bytes basura, es decir, bytes sin valor alguno.

- Host_Command: valor numérico del comando que se desea introducir, ya sea en hexadecimal o decimal. Se recomienda mirar los defines de la librería para encontrar el valor de los comandos más habituales.

---

scFTo.HostCommand(0x62)

4.3.2.9 DummyRead

```
DummyRead()
```

Un método para despertar la pantalla es enviar los llamados bytes basura, aquellos bytes sin valor, esta función por tanto se encarga de mandar hasta 3 de estos con el fin de despertar la pantalla.

---

scFTo.DummyRead()

4.3.2.10 Incremento de Offset

```
FT800_IncCMDOffset(Current_Offset, Command_Size)
```

Esta es una de las funciones más importantes de la librería pues se encarga de regular el offset cuando se añaden comandos a la lista de comandos FIFO, es decir, se encarga de que ninguno sobrescriba un comando ya enviado pendiente por ejecutarse.

Además una vez llega al límite, 4095, el offset ha de resetearse ya que es circular según se explica en la página

68 de la guía del programador de la pantalla táctil FT800.

- Current_offset: actual valor numérico del puntero del offset interno de la pantalla.
- Command_Size: número de bytes que ocupan los comandos ejecutados.

scFTo.FT800_IncCMDOffset(1081344,4)

4.3.2.11 Espera a que finalice un comando

```
ComEsperaFin()
```

En esta ocasión la función se encarga de esperar a que ejecute el pintado de la pantalla, para ello compara los buffers de escritura y lectura, cuando sean iguales se saldrá del bucle y a la variable global CMD_Offset la modifica con el valor del registro que posean los de escritura y lectura al terminar la espera.

scFTo.ComEsperaFin()

4.3.2.12 Escritura en RAM

```
EscribeRam32( dato32)
EscribeRam16( dato16)
EscribeRam8( dato8):
```

Para escribir en RAM hemos de tener cuidado con el offset, por ello es importante modificar la variable global al finalizar, con el número de bytes que hemos incorporado con el único fin de evitar que se pisen instrucciones.

Al igual que en la función de escribir se crearon 3, a la hora de mandar datos a la RAM hemos de tener el mismo número, ya que la necesitamos para los casos de 4,2 y 1 byte.

- Dato32: valor numérico de 32 bits como máximo que queramos escribir en una RAM previamente determinada.
- Dato16: valor numérico de 16 bits como máximo que queramos escribir en una RAM previamente determinada.
- Dato8: valor numérico de 8 bits como máximo que queramos escribir en una RAM previamente determinada.

scFTo.EscribeRam32(4294967050)

scFTo.EscribeRam16(800)

scFTo.EscribeRam8(0)

```
EscribeRamTxt(cadena)
```

También podemos mandar cadenas de caracteres, para ello debemos ir mandando carácter a carácter hasta finalizar, además de posteriormente mandar el terminador, representado con un 0.

En Python las cadenas de caracteres se tratan como vectores, por lo que crearemos una variable que inicie en 0 y vaya aumentando hasta que sea superior al tamaño de dicho vector, así lo recorreremos entero.

- Cadena: Conjunto de caracteres que se desean escribir en una RAM previamente definida.

---

scFTo.EscribeRamTxT("hola")

4.3.2.13 Punto de inicio de la primitiva geometría

```
ComVertex2ff( x, y):
```

Función base para la elaboración de las figuras geométricas más simples como lo son cuadrados, líneas, círculos, etc.

- X: valor numérico de la coordenada en el eje X.
- Y: valor numérico de la coordenada en el eje Y.

---

scFTo.ComVertex2ff(50,124)

4.3.2.14 Ejecución de la cola FIFO

```
Ejecuta_Lista()
```

Con esta función se comenzarían a desarrollar todas las funciones que se han ido almacenando en la cola esperando para ser realizadas con un offset concreto.

---

scFTo.Ejecuta_Lista()

4.3.2.15 Revisión de tamaño

```
PadFIFO()
```

El offset ha de ir en grupos de 4 bytes, por lo que es esencial comprobar cuando se ha insertado menos de dicha cantidad para rellenar con 0s y así la siguiente instrucción que se envié se pueda leer de forma correcta, además de no afectar al funcionamiento de la anterior.

scFTo.PadFIFO()

4.3.2.16 Leer pantalla táctil

```
Lee_pantalla()
```

Las pantallas táctiles permiten un flujo de datos bidireccional, es decir, no solo se encargan de recibir y pintar todo aquello que le hayamos programado, como cualquier pantalla, sino que además nos permite saber en que punto exacto se ha tocado, aplicación realmente útil para la creación de elementos como botones.

scFTo.Lee_pantalla()

4.3.3 Funciones externas

4.3.3.1 Configuración de la pantalla

```
conf_pant(var1)
```

Esta función se encarga de configurar los registros internos de la pantalla que varían según el tamaño de la misma.

Es importante que el usuario ejecute esta función antes que las demás pues podría causar grandes problemas pese a que esta configurado en la previa declaración que se empleen por defecto los valores de la pantalla de 3.5 pulgadas.

Esta función se subdivide en dos secciones, la primera es una declaración para que las variables configuradas sean de carácter global para todo el código y otra en la que se les da el valor correspondiente según una condición establecida.

- Var1: valor entero que será el que determine el tamaño de la pantalla, siendo 3 y 5 la pantalla de 3.5 y la 5 pulgadas respectivamente.

En caso de recibir otro valor, al ser posible enviar cualquier valor que se desee, el programa no hará nada y devolverá un 0 indicando al usuario que ha salido algo mal.

scFto.conf_pant(3)

4.3.3.2 Inicialización de los pines

```
HAL_Init_SPI(pin1, pin2)
```

Esta función se encarga de configurar los pines, ya explicados anteriormente, pudiendo el usuario elegir los dos de libre configuración a través de las variables que recibe, siendo la primera de ellas la encargada del pin de chip select mientras la segunda será la que determine el puerto del power down.

En esta función se emplean funciones de una librería que hay que incorporar al inicio de la nuestra, Pin y SPI de la librería machine, con esto el usuario no tendrá que incorporarlas manualmente él, reduciendo así la dificultad de ejecución de la nuestra.

- Pin1: valor numérico del pin que se vaya a configurar como PD, es decir, como power down. A pesar de que la función pueda recibir cualquier valor, la función fallará si esta por encima de los pines del microcontrolador, o coincide el valor con un pin no configurable para esta determinada función.
- Pin2: valor numérico del pin que se vaya a configurar como CS, es decir, como chip select. A pesar de que la función pueda recibir cualquier valor, la función fallará si esta por encima de los pines del microcontrolador, o coincide el valor con un pin no configurable para esta determinada función.

---

scFTo.HAL_Init_SPI(14,12)

4.3.3.3 Escritura en RAM

```
Comando( COMM)
```

Básicamente esta función es EscribeRam32, solo que pensada para su aplicación para tanto el usuario como otras funciones de la librería ejecuten rápidamente diferentes comandos.

- COMM: valor numérico en hexadecimal o decimal asociado a algún comando preestablecido por la pantalla.

---

scFTo.Comando(0x0400000)

4.3.3.4 Color de la siguiente instrucción que se va a realizar

```
ComColor( R, G, B)
```

Esta función permite al usuario seleccionar el color con el que se representará la siguiente instrucción que realice, teniendo una amplia libertad de selección al disponer de más de 16 millones de posibles colores diferentes.

Es la primera función que necesita la selección de pantalla previamente pues como ya se menciono, la posición de los colores difiere entre la pantalla de 3.5 pulgadas y la de 5 pulgadas.

- R: valor numérico asociado a la tonalidad roja que se desea. Su rango va de 0 a 255, es decir, 1 byte.
- G: valor numérico asociado a la tonalidad verde que se desea. Su rango va de 0 a 255, es decir, 1 byte.
- B: valor numérico asociado a la tonalidad azul que se desea. Su rango va de 0 a 255, es decir, 1 byte.

Quedando el color final determinado por la combinación de los valores aportados por los tres argumentos.

scFTo.ComColor(45,62,148)

```
ComFgcolor( R, G, B)
ComBgcolor( R, G, B)
```

Los argumentos de estas dos funciones son iguales que los de la función anterior, por lo que se omitirá su explicación.

scFTo.ComFgcolor(76,84,2)

scFTo.ComBg(200,160,0)

En ciertas aplicaciones no solo necesitaremos configurar el color, sino también el color de fondo (BG) o de primer plano (FG), para ello se pensó las dos funciones que se acaban de presentar.

Para diferenciar cuando se emplea una y cuando otra podemos observar la siguiente tabla según la información que nos proporciona la guía de la pantalla.

Instrucción	ComFgcolor	ComBgcolor	ComColor
ComTXT	No afecta	No afecta	Sí
Boton	Sí	No afecta	Sí(rótulo)
ComTeclas	Sí	No afecta	Sí(texto)
ComScrollbar	Sí(barra interior)	Sí(barra exterior)	No afecta
ComNum	No afecta	No afecta	Sí

Tabla 4-II:Función de color

Hay más comandos a los que va a afectar estas funciones, pero al no haber sido incluidos se ha optado por dejarlos fuera de esta tabla, para ellos se recomienda acudir a la guía.

Aparte de las mencionadas funciones, existe otra a la que debería de afectar estas instrucciones para alterar el color, pero esta programada para no hacerlo, al venir ya definido un color dentro de la misma. Esto esta realizado ya que no tiene gran importancia para el desarrollo del proceso, en especial ya que no es un elemento fijo, sino uno temporal.

4.3.3.5 Escritura en la pantalla

```
ComTXT( x, y, fuente, ops, cadena)
```

Esta función nos permite escribir el texto que deseemos, con una serie de propiedades definidas en los argumentos de la función.

- X: Valor numérico de la coordenada en el eje X.
- Y: valor numérico de la coordenada en el eje Y.
- Fuente: valor numérico, en el rango de 0 a 31 que nos permite elegir el tamaño de la fuente de escritura.
- Ops: valor numérico al que va asociado diferentes opciones a la hora de como tomar los valores que se mandan en el argumento, estas opciones van recogidas en las definiciones previas a las funciones en nuestra librería. De no darle ningún valor se tomara como defecto la opción 0, que se corresponde a considerar las coordenadas recibidas como el vértice superior izquierdo desde el que se comenzará la escritura.
- Cadena: Conjunto de caracteres que se deseen representar en la pantalla en una posición dada.

---

scFto.ComTxT(80,90,25,OPT_CENTERX,"hola")

```
ComNum( x, y, fuente, ops, Num)
```

Al igual que lo podemos realizar con letras también disponemos la posibilidad de escribir números, contando con el mismo número de opciones, se pueden representar números por la anterior opción en caso de ser necesario, aunque gastaría un poco más de recursos que la función aquí presente.

- X: Valor numérico de la coordenada en el eje X.
- Y: valor numérico de la coordenada en el eje Y.
- Fuente: valor numérico, en el rango de 0 a 31 que nos permite elegir el tamaño de la fuente de escritura.
- Ops: valor numérico al que va asociado diferentes opciones a la hora de como tomar los valores que se mandan en el argumento, estas opciones van recogidas en las definiciones previas a las funciones en nuestra librería. De no darle ningún valor se tomara como defecto la opción 0, que se corresponde a considerar las coordenadas recibidas como el vértice superior izquierdo desde el que se comenzará la escritura. Resaltar que para números negativos será necesario emplear la opción "OPT_SIGNED".
- Num: Valor del número que se desea representar en la pantalla táctil

---

Para el número 31: scFTo.ComNum(50,50,20,0,31)

Para el número -31: scFTo.ComNum(50,50,20,OPT_SIGNED,31)

4.3.3.6 Teclas

```
ComTeclas( x, y, w, h, fuente, ops, Keys)
```

Figura 4-2: Visualización de las Teclas [14]

Con esta función podemos crear teclas, que no botones, con el texto que deseemos dentro además de poder cambiar tanto su tamaño como lugar, teniendo incluso la posibilidad de dar otros efectos, como que se centren de otra manera o quitarles el efecto 3D, como el que tienen en la imagen.

- X: Valor numérico de la coordenada en el eje X.
- Y: valor numérico de la coordenada en el eje Y.
- W: valor numérico correspondiente a la anchura de la tecla.
- H: valor numérico correspondiente a la altura de la tecla.
- Fuente: valor numérico, en el rango de 0 a 31 que nos permite elegir el tamaño de la fuente de escritura.
- Ops: valor numérico al que va asociado diferentes opciones a la hora de como tomar los valores que se mandan en el argumento, estas opciones van recogidas en las definiciones previas a las funciones en nuestra librería. De no darle ningún valor se tomara como defecto la opción 0, que se corresponde al efecto 3D de las teclas, ver imagen asociada anteriormente, para eliminar este efecto usar el valor definido en la declaración OPT_FLAT.
- Keys: Conjunto de caracteres que se deseen representar en la pantalla en una dentro de la tecla.

---

scFTo.ComTeclas(50,60,20,20,15,0,"1")

4.3.3.7 Representar en pantalla

```
Dibuja()
```

Una vez ejecutamos esta función, se representará en pantalla todas aquellas instrucciones que se le haya ido dando a la pantalla con las otras funciones. Hay que mencionar que solo se ejecutará durante un instante, por ello debe estar siempre dentro de un bucle eterno de tal manera que parezca que esta pintado de forma continuada.

scFto.Dibuja()

4.3.3.8 Iniciador de la pantalla táctil

```
Inicia_pantalla()
```

Esta función ha de ser empleada tras haber configurado el tipo de pantalla que se esta empleando y con los pines ya definidos, y antes de todo aquello que queramos realizar pues se encarga de varios apartados importantes.

Por un lado inicializa la pantalla, despertándola inicialmente, y comprueba que no haya ningún problema en la comunicación a través de una comprobación de los valores de ciertos registros.

Por otro lado se encarga de dar valor a diferentes registros internos de la pantalla y tras esto configura la pantalla para que este completamente lista para lo que se desee realizar con ella, pudiendo habilitar o no el amplificador de audio a través de las últimas escrituras, teniendo que introducir 0x81 para apagar el audio si no se quiere utilizar.

Una posible mejora sería pasar por referencia un 0 o un 1 con el que poder determinar si se requiere el audio, pero al ser algo secundario y fácilmente cambiable, incluso pudiendo trabajar perfectamente con esta configuración cuando no se trabaja con el audio se ha dejado como esta, activado siempre.

scFTo.Inicia_pantalla()

4.3.3.9 Color de fondo de la pantalla

```
Nueva_pantalla( R, G, B)
```

El color de fondo de la pantalla puede ser cambiado a través de esta función, a la que hay que pasarle por referencia los valores de las diferentes tonalidades que se quiera, en el formato conocido RGB.

Como es conocido no se espera que el usuario tenga problemas usando esta función, pero existe el caso de que inserte un color fuera de rango lo que podría o bien desconfigurar el comando o bien modificar el color, según cual haya sido el color implicado que se haya excedido y la pantalla usada.

- R: valor numérico asociado a la tonalidad roja que se desea. Su rango va de 0 a 255, es decir, 1 byte.
- G: valor numérico asociado a la tonalidad verde que se desea. Su rango va de 0 a 255, es decir, 1 byte.
- B: valor numérico asociado a la tonalidad azul que se desea. Su rango va de 0 a 255, es decir, 1 byte.

Quedando el color final determinado por la combinación de los valores aportados por los tres argumentos.

scFTo.Nueva_pantalla(40,180,60)

4.3.3.10 Debugging de los registros

Las siguientes funciones están más pensadas en una aplicación de debbugging para usuarios que dominen el campo de la programación y estén realizando proyectos con esta pantalla en el entorno de MicroPython, que para usuarios promedios que solo busquen realizar cierta aplicación con los elementos que se le da en la librería.

Esto se debe a que la utilidad de esta función es informar al usuario que valor hay en el registro que el quiera conocer, de esta forma realizar una comprobación manual de los puntos donde no se realice lo que el usuario busque.

```
Lee_Reg(dir)
```

- Dir: Dirección del registro del que se desea conocer su valor.

Lectura=scFto.Lee_Reg(REG_SOUND)

Mientras que la siguiente función es su contraparte, encargándose de la escritura en la dirección aportada por el usuario, siempre empleando valores de 32 bits.

```
Esc_Reg( dir, valor)
```

- Dir: Dirección del registro en el que se desea sobrescribir su valor.
- Valor: valor numérico que se desea escribir en la dirección asociada.

scFto.Esc_Reg(REG_VOL_SOUND, 24)

4.3.3.11 Espera

```
Espera_pant()
```

Esta función se encarga de bloquear la ejecución del código elaborado por el usuario hasta que se realice un toque en cualquier lugar de la pantalla táctil. Para ello se ejecuta un bucle while, en el que mientras el valor de POSY sea el de reset, significa que no se ha pulsado, se mantendrá leyendo la pantalla esperando que POSY varíe su valor. Una vez se pulse, se realiza otro bucle igual que el anterior, pero en esta ocasión con el fin de notificar que se ha dejado de pulsar, es decir, que la variable ha vuelto a su valor de reset, esto se realiza ya que consideramos un toque como pulsar y retirar no solo realizar una pulsación, además de que esto último podría provocar ligeros problemas en la aplicación, como sería que al pulsar se realizaran varias operaciones rápidamente sin que el usuario pudiera reaccionar.

scFTo.Espera_pant()

4.3.3.12 Líneas

Para realizar un conjunto de líneas rectas debemos emplear los siguientes comandos, previa definición del color y del grosor de estas. El color ya se ha mencionado anteriormente como realizarlo, pero no el caso de la configuración del grosor, para poder determinarlo hemos de invocar la siguiente función que recibe como parámetro el grosor deseado

```
ComLineWidth( width)
ComLineas(xo,yo, xf, yf)
```

Esta última función es la encargada de realizar una línea entre la posición inicial, dada por el primer grupo de parámetros, y la posición final, indicada en el segundo grupo. El usuario podría realizar a mano esta configuración, aunque se ha realizado esta función para simplificar en todo lo posible los comandos que debe aprenderse para poder manejar la pantalla en todo su apogeo.

- Width: valor numérico del grosor de la línea deseado. Su rango va entre 16 y 4095, con una precisión 1/16 Píxeles.
- Xo: valor numérico de la coordenada en el eje X del punto inicial de la recta.
- Yo: valor numérico de la coordenada en el eje Y del punto inicial de la recta.
- Xf: valor numérico de la coordenada en el eje X del punto final de la recta.
- Yf: valor numérico de la coordenada en el eje Y del punto final de la recta.

scFTo.ComLineWidth(40)

scFTo.ComLineas(10,20,50,50)

4.3.3.13 Scrollbar

```
ComScrollbar( x, y, w, h, ops, val, size, range):
```

Se podrá realizar un scroll junto a su barra en cualquier lugar de la pantalla, tanto en vertical como en horizontal sin problema alguno, tan solo es necesario a lo sumo un poco de prueba y error para configurar correctamente todos los elementos necesarios, que se enumeran a continuación.

- X: Valor numérico de la coordenada en el eje X.
- Y: valor numérico de la coordenada en el eje Y.
- W: valor numérico correspondiente a la anchura de la barra.
- H: valor numérico correspondiente a la altura de la barra.

- Ops: valor numérico al que va asociado diferentes opciones a la hora de como tomar los valores que se mandan en el argumento, estas opciones van recogidas en las definiciones previas a las funciones en nuestra librería. De no darle ningún valor se tomara como defecto la opción 0, que se corresponde al efecto 3D del scrollbar, para eliminar este efecto usar el valor 255.
- val: valor numérico de la posición inicial del scroll dentro de la barra, con un rango determinado entre 0 y el valor de range que se pasara posteriormente.
- Size: valor numérico del tamaño del scroll.
- Range: máximo valor numérico que puede tomar el scroll al ser desplazado por la barra.

Según los valores de w y h la barra se pintara en horizontal o vertical, siendo para el primer caso necesario que el valor de w sea superior al de h. En caso contrario estaremos en el segundo caso.

scFto.Scrollbar(20,50,120,8,0,10,40,100)

En el ejemplo se representara un scrollbar con el vértice superior izquierdo en la posición (20,50) de la pantalla, con una anchura de 120 y una altura de 8. Tendrá efecto 3D, su posición inicial estará en el valor 10 de un rango de medidas de 0 a 100, con una barra que mide 40.

Figura 4-3:Scrollbar asociado al ejemplo [15]

4.3.3.14 Boton

El usuario cuando quiera realizar un botón podrá elegir entre dos funciones, la primera de ellas se encargará de solo pintar el botón en el estado deseado y configurado por el usuario, mientras que la segunda función pintará el botón dependiendo si esta pulsado o no, devolviendo al usuario dicha información.

```
ComButton( x, y, w, h, font, ops, cadena)
```

- x: valor numérico de la coordenada X del vértice superior izquierdo del botón.
- y: valor numérico de la coordenada Y del vértice superior izquierdo del botón.
- W: valor numérico correspondiente a la anchura del botón.
- H: valor numérico correspondiente a la altura del botón.
- font: valor numérico de la fuente deseada para la cadena de caracteres que se escribirá dentro.
- ops: Según su valor el botón será representado como pulsado o no pulsado, 1 será el valor en el primer caso, 0 para el restante.

- cadena: conjunto de caracteres que se desea representar dentro del botón.

Estado=scFTo.ComButton(80,40,30,30,10,1,"pulsa")

```
Boton(x, y, w, h,  font, cadena)
```

- x: valor numérico de la coordenada X del vértice superior izquierdo del botón.
- y: valor numérico de la coordenada Y del vértice superior izquierdo del botón.
- W: valor numérico correspondiente a la anchura del botón.
- H: valor numérico correspondiente a la altura del botón.
- font: valor numérico de la fuente deseada para la cadena de caracteres que se escribirá dentro.
- cadena: conjunto de caracteres que se desea representar dentro del botón.

---

scFTo.Boton(80,40,30,30,10,"pulsa")

En ambos ejemplos se pintara el mismo botón con la diferencia que en el segundo su estado podrá variar dependiendo si se pulsa dentro del botón la pantalla táctil, además la función nos proporcionara ese información para que podamos emplearla en el código de manera sencilla.

4.3.3.15 Puntos

```
ComPointSize(size)
```

Al igual que ocurría con las líneas, los puntos pueden configurar su tamaño a través de esta función, aunque no se encargará de ser pintados, esta pensada para aquellos que deseen emplear el comando manual para pintar los puntos que él desee.

- Size: valor numérico del tamaño del punto deseado.

---

scFTo.ComPointSize(16)

Aunque lo normal será que se ejecute la siguiente función, que engloba tanto la configuración del tamaño como su posterior pintado, siendo como la mejora de la anterior.

```
Com_Punto( x, y, R)
```

- x: valor numérico de la coordenada X del centro del punto.
- y: valor numérico de la coordenada Y del centro del punto.
- R: valor numérico correspondiente al radio del punto, también denominado anchura de este.

scFto.Com_Punto(50,50,16)

4.3.3.16 Calibración

La pantalla táctil tiene cierta sensibilidad, aunque en ocasiones puede llegar a ser demasiada o muy poca, reduciendo la efectividad del control que ofrece, para evitar estos casos la siguiente función se encarga de una forma sencilla y cómoda de regular los parámetros internos para ofrecer la mejor experiencia posible al usuario.

```
Calibra_touch()
```

Tras ejecutar esta función y pulsar los puntos que se muestran en la pantalla, se acabara regulando adecuadamente para las siguientes aplicaciones. Si bien no es necesario, se recomienda siempre calibrar cuando se activa la pantalla, y cada vez que se muestren dificultades a la hora de la detección del toque.

scFto.Calibra_touch()

4.3.3.17 Degradado

```
ComGradient(x0, y0, color0, x1, y1, color1)
```

Además de poder colocar el fondo de un tono determinado, podemos realizar un ligero degradado entre dos colores, configurando los colores iniciales y finales además de los puntos de inicio y final, pudiendo extenderlo por todo el fondo en caso que se desee, con tan solo poner como punto inicial el punto origen, y como final el vértice inferior derecho de la pantalla.

- X0: valor numérico de la coordenada X inicial a partir de la cual se realizara el degradado.
- Y0: valor numérico de la coordenada Y inicial a partir de la cual se realizara el degradado.
- Color0: valor numérico de 32 bits correspondiente al color inicial del degradado.
- X1: valor numérico de la coordenada X final del degradado.
- Y1: valor numérico de la coordenada Y final del degradado.
- Color1: valor numérico de 32 bits correspondiente al color final del degradado.

scFTo.ComGradient(15,20,255,60,80,0)

Para saber los valores de color0 y color1 correctos se recomienda emplear la siguiente función

```
Colr(R, G, B)
```

Dicha función nos devuelve el valor numérico asociado al color que hemos configurado, para posteriormente pasarlo como argumento a la función anterior.

- R: valor numérico asociado a la tonalidad roja que se desea. Su rango va de 0 a 255, es decir, 1 byte.
- G: valor numérico asociado a la tonalidad verde que se desea. Su rango va de 0 a 255, es decir, 1 byte.
- B: valor numérico asociado a la tonalidad azul que se desea. Su rango va de 0 a 255, es decir, 1 byte.

Colore=scFto.Colr(200,20 20)

4.3.3.18 Altavoz

Además de la pantalla táctil, esta nos ofrece la posibilidad de utilizar el altavoz que lleva incorporado para aumentar los diversos usos posibles que se puedan pensar, para ello se han diseñado unas 4 funciones que facilitaran su uso.

Debemos recordar que estas funciones no funcionaran en caso de que previamente en la función de inicialización de la pantalla, se haya cambiado los últimos valores, de tal manera que no este encendido el altavoz.

```
VolNota(volumen)
```

Esta función nos permite configurar el volumen con el que se emitirá las notas a continuación, funcionando de igual manera que la función de color a la representación de figuras en la pantalla táctil.

```
TocaNota(instr, nota)
```

Además de esta función que permite la emisión de una nota por tiempo indefinido, se apoya en una serie de definiciones que ayudan al usuario ha conocer el valor numérico tanto de las notas musicales normales como de los posibles instrumentos que nos ofrece el fabricante.

```
FinNota()
```

Esta función se acompaña de la anterior, ya que se encarga de silenciar el altavoz, es decir, de finalizar la reproducción de la emisión de una nota, siendo completamente inútil en caso de ser empleada sin estar reproduciendo ningún sonido por el altavoz.

```
Fadeout()
Fadein()
```

Finalmente igual que existen funciones de degradado de color, tenemos degradado de tono, existiendo tanto el degradado ascendente como descendente, ambas funciones tienen un formato similar, ya que se basan en un bucle donde se va incremento o decreciendo según la función correspondiente el valor de una variable, i, dejando entre cada interacción un pequeño tiempo de pausa. Lo único que varía es que en la función creciente, antes de finalizar se realiza una última interacción donde se emite al tono máximo.

- Volumen: valor numérico que representa la intensidad con la que sonará la nota que se configuren a continuación.
- Instr: valor numérico que según viene definido por el fabricante se le asociara un instrumento u otro, se añaden varios en las declaraciones iniciales de la librería.
- nota: valor numérico que según viene definido por el fabricante se le asociara una nota u otra, se añaden varias en las declaraciones iniciales de la librería.

```
scFTo.volNota(20)
scFTo.TocaNota(S_PIANO, N_DO)
scto.FinNota()
```

```
scFTo.volNota(20)
scFTo.TocaNota(S_PIANO, N_DO)
scFTo.Fadeout()
scto.FinNota()
```

```
scFTo.volNota(20)
scFTo.TocaNota(S_PIANO, N_DO)
scFTo.Fadeint()
scto.FinNota()
```

4.4 Pruebas realizadas

Con el fin de poder certificar el correcto funcionamiento de la librería realizada así como ofrecer un ejemplo que sirva como una pequeña guía a los futuros usuarios, se decidió crear un código que emplee varias operaciones de las que ofrece la pantalla. Aunque cualquier código pudiera cumplir el objetivo se realizo simple y bastante comentado ya que tiene que ser fácilmente interpretable.

5 CONCLUSIONES

Tanto la pantalla como el microcontrolador, así como el entorno de programación, tienen sus límites definidos por lo que este trabajo no tiene mucho margen de maniobra, pese a todo ofrece al programador un gran número posibilidades de manera cómoda.

Hay que puntualizar que esta librería no funcionará con pantallas diferentes a las empleadas, por lo que tiene fecha de caducidad, en especial en un campo que esta en continuo desarrollo y en la que las empresas cambian constantemente sus productos buscando siempre conseguir innovar en un competido mercado. Pese a ello la librería actual siembra las bases de como son las librerías para pantallas táctiles de manera que cualquier otra persona puede modificarla y ajustarla a la pantalla que desee, al tener un funcionamiento similar, respetando la configuración de bits establecida.

Otro aspecto que queremos resaltar es la limitación actual de MicroPython, el lenguaje de programación esta configurado para ser empleado por pocos microcontroladores del mercado reduciendo considerablemente su uso. Se espera que esto vaya cambiando en los próximos años, pues Python, el lenguaje del que procede, es uno de los más empleados actualmente y el deseo de reducir el tamaño de los elementos electrónicos hará que cada vez tome más importancia su versión reducida y ajustada para procesadores de reducido tamaño.

La librería es pues una recopilación de algunas de las funciones que nos ofrece la pantalla elegida, aunque existen más que se pueden incorporar, siguiendo una estructura muy similar a la que ha sido elaborada en otras funciones. Aquí se han recogido las que se han considerado más comunes, por tanto debemos tener en cuenta que esta librería estará en continuo desarrollo por aquellos que la empleen usando de esta manera la mayor de las ventajas de un código abierto, su comunidad.

Durante la realización del trabajo se ha apreciado que la pantalla presenta grandes errores si es empleado con cables en lugar de un circuito impreso, pudiendo resultar en algunos pixeles de la pantalla sin alterar como en no realizar las funciones pertinentes que se le han solicitado realizar.

6 ANEXO

6.1 Librería completa

```
from machine import Pin
from machine import SPI
CMDBUF_SIZE=4096
CMD_APPEND=4294967070
CMD_BGCOLOR=4294967049
CMD_BITMAP_TRANSFORM=4294967073
CMD_BUTTON=4294967053
CMD_CALIBRATE=4294967061
CMD_CLOCK=4294967060
CMD_COLDSTART=4294967090
CMD_CRC=4294967043
CMD_DIAL=4294967085
CMD_DLSTART=4294967040
CMD_EXECUTE=4294967047
CMD_FGCOLOR=4294967050
CMD_GAUGE=4294967059
CMD_GETMATRIX=4294967091
CMD_GETPOINT=4294967048
CMD_GETPROPS=4294967077
CMD_GETPTR=4294967075
CMD_GRADCOLOR=4294967092
CMD_GRADIENT=4294967051
CMD_HAMMERAUX=4294967044
CMD_IDCT=4294967046
CMD_INFLATE=4294967074
CMD_INTERRUPT=4294967042
CMD_KEYS=4294967054
CMD_LOADIDENTITY=4294967078
CMD_LOADIMAGE=4294967076
CMD_LOGO=4294967089
CMD_MARCH=4294967045
CMD_MEMCPY=4294967069
CMD_MEMCRC=4294967064
CMD_MEMSET=4294967067
CMD_MEMWRITE=4294967066
CMD_MEMZERO=4294967068
CMD_NUMBER=4294967086
CMD_PROGRESS=4294967055
CMD_REGREAD=4294967065
CMD_ROTATE=4294967081
CMD_SCALE=4294967080
CMD_SCREENSAVER=4294967087
CMD_SCROLLBAR=4294967057
CMD_SETFONT=4294967083
CMD_SETMATRIX=4294967082
CMD_SKETCH=4294967088
CMD_SLIDER=4294967056
CMD_SNAPSHOT=4294967071
CMD_SPINNER=4294967062
CMD_STOP=4294967063
CMD_SWAP=4294967041
CMD_TEXT=4294967052
CMD_TOGGLE=4294967058
CMD_TOUCH_TRANSFORM=4294967072
CMD_TRACK=4294967084
CMD_TRANSLATE=4294967079
CMD_POINTSIZE=218103808
```

```
#Defines sacados del ejemplo de Arduino

OPT_CENTER=1536
OPT_CENTERX=512
OPT_CENTERY=1024
OPT_FLAT=256
OPT_MONO=1
OPT_NOBACK=4096
OPT_NODL=2
OPT_NOHANDS=49152
OPT_NOHM=16384
OPT_NOPOINTER=16384
OPT_NOSECS=32768
OPT_NOTICKS=8192
OPT_RIGHTX=2048
OPT_SIGNED=256

#MIS PROPIOS DEFINES, en hexadecimal. Quizas tenga que cambiarlos

CMD_BEGIN_BMP=0x1f000001
CMD_BEGIN_POINTS=0x1f000002
CMD_BEGIN_LINES=0x1f000003
CMD_BEGIN_LINESTRIP=0x1f000004
CMD_BEGIN_EDGESTRIP_R=0x1f000005
CMD_BEGIN_EDGESTRIP_L=0x1f000006
CMD_BEGIN_EDGESTRIP_A=0x1f000007
CMD_BEGIN_EDGESTRIP_B=0x1f000008
CMD_BEGIN_RECTS=0x1f000009
CMD_END=0x21000000
CMD_DISPLAY=0
CMD_LINEWIDTH=0x0E000000   #ERROR EN LA DOCUMENTACION: PONE 0x06000000

RAM_CMD=1081344
RAM_DL=1048576
RAM_G=0
RAM_PAL=1056768
RAM_REG=1057792


REG_ANALOG=1058104
REG_ANA_COMP=1058160
REG_BIST_CMD=1058124
REG_BIST_EN=1058132
REG_BIST_RESULT=1058128
REG_BUSYBITS=1058008
REG_CLOCK=1057800
REG_CMD_DL=1058028
REG_CMD_READ=1058020
REG_CMD_WRITE=1058024
REG_CPURESET=1057820
REG_CRC=1058152
REG_CSPREAD=1057892
REG_CYA0=1058000
REG_CYA1=1058004
REG_CYA_TOUCH=1058100
REG_DATESTAMP=1058108
REG_DITHER=1057884
REG_DLSWAP=1057872
REG_FRAMES=1057796
REG_FREQUENCY=1057804
REG_GPIO=1057936
REG_GPIO_DIR=1057932
REG_HCYCLE=1057832
REG_HOFFSET=1057836
REG_HSIZE=1057840
REG_HSYNC0=1057844
REG_HSYNC1=1057848
REG_ID=1057792
```

```
REG_INT_EN=1057948
REG_INT_FLAGS=1057944
REG_INT_MASK=1057952
REG_MACRO_0=1057992
REG_MACRO_1=1057996
REG_MARCH_ACC=1058144
REG_MARCH_DIR=1058136
REG_MARCH_OP=1058140
REG_MARCH_WIDTH=1058148
REG_OUTBITS=1057880
REG_PCLK=1057900
REG_PCLK_POL=1057896
REG_PLAY=1057928
REG_PLAYBACK_FORMAT=1057972
REG_PLAYBACK_FREQ=1057968
REG_PLAYBACK_LENGTH=1057960
REG_PLAYBACK_LOOP=1057976
REG_PLAYBACK_PLAY=1057980
REG_PLAYBACK_READPTR=1057964
REG_PLAYBACK_START=1057956
REG_PWM_DUTY=1057988
REG_PWM_HZ=1057984
REG_RENDERMODE=1057808
REG_ROMSUB_SEL=1058016
REG_ROTATE=1057876
REG_SNAPSHOT=1057816
REG_SNAPY=1057812
REG_SOUND=1057924
REG_SWIZZLE=1057888
REG_TAG=1057912
REG_TAG_X=1057904
REG_TAG_Y=1057908
REG_TAP_CRC=1057824
REG_TAP_MASK=1057828
REG_TOUCH_ADC_MODE=1058036
REG_TOUCH_CHARGE=1058040
REG_TOUCH_DIRECT_XY=1058164
REG_TOUCH_DIRECT_Z1Z2=1058168
REG_TOUCH_MODE=1058032
REG_TOUCH_OVERSAMPLE=1058048
REG_TOUCH_RAW_XY=1058056
REG_TOUCH_RZ=1058060
REG_TOUCH_RZTHRESH=1058052
REG_TOUCH_SCREEN_XY=1058064
REG_TOUCH_SETTLE=1058044
REG_TOUCH_TAG=1058072
REG_TOUCH_TAG_XY=1058068
REG_TOUCH_TRANSFORM_A=1058076
REG_TOUCH_TRANSFORM_B=1058080
REG_TOUCH_TRANSFORM_C=1058084
REG_TOUCH_TRANSFORM_D=1058088
REG_TOUCH_TRANSFORM_E=1058092
REG_TOUCH_TRANSFORM_F=1058096
REG_TRACKER=1085440
REG_TRIM=1058156
REG_VCYCLE=1057852
REG_VOFFSET=1057856
REG_VOL_PB=1057916
REG_VOL_SOUND=1057920
REG_VSIZE=1057860
REG_VSYNC0=1057864
REG_VSYNC1=1057868

GRIS_CLARO=0xE0E0E0
GRIS_OSCURO=0x606060

S_TRIANG=0x04
S_BEEP=0x05
S_ALARM=0x06
```

```
  S_WARBLE=0x07
  S_PIPS=0x10
  S_XILO=0x41
  S_PIANO=0x46

  N_DO=60
  N_REB=61
  N_RE=62
  N_MIB=63
  N_MI=64
  N_FA=65
  N_SOLB=66
  N_SOL=67
  N_LAB=68
  N_LA=69
  N_SIB=70
  N_SI=71
  FT_GPU_INTERNAL_OSC=0x48
  FT_GPU_EXTERNAL_OSC=0x44
  FT_GPU_PLL_48M=0x62
  FT_GPU_PLL_36M=0x61
  FT_GPU_PLL_24M=0x64
  FT_GPU_ACTIVE_M=0x00
  FT_GPU_STANDBY_M=0x41
  FT_GPU_SLEEP_M=0x42
  FT_GPU_POWERDOWN_M=0x50
  FT_GPU_CORE_RESET=0x68
  RELOJ=3000 #revisar este valor
  #definiciones que van a variar su valor
  POSX=0
  POSY=0
  PANTALLA=3
  HCYCLE=408
  HOFFSET=70
  HSYNC0=0
  HSYNC1=10
  VCYCLE=263
  VOFFSET=13
  VSYNC0=0
  VSYNC1=2
  PCLK_POL=0
  HSIZE=320
  VSIZE=240
  PCLK=5
  CS=1
  PD=1
  spi=1
  CMD_Offset=0
  #Funcion para determinar el tipo de pantalla que se tiene. Es necesario que se utilice
antes que el resto de las demas funciones o daria problemas
  def conf_pant(var1):
    if (var1==3) or (var1==5):
      global PANTALLA
      global HCYCLE
      global HOFFSET
      global HSYNC0
      global HSYNC1
      global VOFFSET
      global VCYCLE
      global VSYNC0
      global VSYNC1
      global PCLK_POL
      global HSIZE
      global VSIZE
      global PCLK
      PANTALLA=var1
      if (PANTALLA==3):
        #lista de defines que van con la pantalla ft800 de 3.5 pulgadas
        HCYCLE =408
```

```
        HOFFSET=70
        HSYNC0=0
        HSYNC1=10
        VCYCLE=263
        VOFFSET=13
        VSYNC0=0
        VSYNC1=2
        PCLK_POL=0
        HSIZE=320
        VSIZE=240
        PCLK=5
      else:
        #lista de defines que van con la pantalla de ft800 de 5 pulgadas
        HCYCLE =548
        HOFFSET=43
        HSYNC0=0
        HSYNC1=41
        VCYCLE=292
        VOFFSET=12
        VSYNC0=0
        VSYNC1=10
        PCLK_POL=1
        HSIZE=480
        VSIZE=272
        PCLK=5
      return 1
    else:
      return 0

  def HAL_Init_SPI(pin1, pin2):
    global CS
    global PD
    global spi
    #CS=chip select
    #recomendamos pin14
    CS=Pin(pin1,Pin.OUT)
    #PD=power down
    #recomendamos pin12
    PD=Pin(pin2,Pin.OUT)
    spi=SPI(baudrate=600000,polarity=0,phase=0, bits=8, firstbit=SPI.MSB, sck=Pin(5),
mosi=Pin(18), miso=Pin(19))

  def HAL_SPI_ReadWrite(data):
    dat=bytearray(1)
    recibido=bytearray(1)
    dat[0]=data
    spi.write_readinto(dat,recibido)
    recibido=int.from_bytes(recibido,"big") #int


    return recibido

  def HAL_SPI_CSHigh():
    CS.value(1)


  def HAL_SPI_CSLow():
    CS.value(0)


  def HAL_SPI_PDhigh():
    PD.value(1)

  def HAL_SPI_PDlow():
    PD.value(0)
```

```
  def DELAY ():
    from utime import sleep_ms
    sleep_ms(800) #mirar tiempo luego despues
  def SENDADDRESSWR( Memory_Address):
    # Write out the address. Only the lower 3 bytes are sent, with the most significant
byte sent first
    # Mask off the first byte to send
    SPI_Writebyte = ((Memory_Address & 0x00FF0000) >> 16)
    # Since this is a write, the MSBs are forced to 10
    SPI_Writebyte = (SPI_Writebyte & 0xBF | 0x80)
    # Call the low-level SPI routine for this MCU to send this byte
    HAL_SPI_ReadWrite(SPI_Writebyte)
    SPI_Writebyte = ((Memory_Address & 0x0000FF00) >> 8)
    HAL_SPI_ReadWrite(SPI_Writebyte)
    SPI_Writebyte = (Memory_Address & 0x000000FF)
    HAL_SPI_ReadWrite(SPI_Writebyte)

  def SENDADDRESSRD( Memory_Address):
    # Write out the address. Only the lower 3 bytes are sent, with the most significant
byte sent first
    # Mask off the first byte to send
    SPI_Writebyte = ((Memory_Address & 0x00FF0000) >> 16)
    # Since this is a read, the upper two bits are forced to 00
    SPI_Writebyte = (SPI_Writebyte & 0x3F)
    # Call the low-level SPI routine for this MCU to send this byte
    HAL_SPI_ReadWrite(SPI_Writebyte)
    SPI_Writebyte = ((Memory_Address & 0x0000FF00) >> 8)
    HAL_SPI_ReadWrite(SPI_Writebyte)
    SPI_Writebyte = (Memory_Address & 0x000000FF)
    HAL_SPI_ReadWrite(SPI_Writebyte)
    # Send dummy 00 as required in the FT800 datasheet when doing a read
    HAL_SPI_ReadWrite(0x00)



  def WRITE_32( SPIValue32):
    SPI_Writebyte = (SPIValue32 & 0x000000FF)
    # Write the first (least significant) byte
    HAL_SPI_ReadWrite(SPI_Writebyte)
    SPI_Writebyte = ((SPIValue32 & 0x0000FF00) >> 8)
    HAL_SPI_ReadWrite(SPI_Writebyte)
    SPI_Writebyte = ((SPIValue32 & 0x00FF0000)>> 16)
    HAL_SPI_ReadWrite(SPI_Writebyte)
    SPI_Writebyte = ((SPIValue32 & 0xFF000000) >> 24)
    # Write the last (most significant) byte
    HAL_SPI_ReadWrite(SPI_Writebyte)

  def WRITE_16( SPIValue16):
    SPI_Writebyte = (SPIValue16 & 0x00FF)
    # Write the first (least significant) byte
    HAL_SPI_ReadWrite(SPI_Writebyte)
    SPI_Writebyte = ((SPIValue16 & 0xFF00) >> 8)
    # Write the last (most significant) byte
    HAL_SPI_ReadWrite(SPI_Writebyte)

  def WRITE_8( SPIValue8):
    # Write the data byte
    HAL_SPI_ReadWrite(SPIValue8)
  def READ_32():
    # Read the first data byte (this is the least significant byte of the register). SPI
writes out dummy byte of 0x00
    # Get the byte which was read by the low-level MCU routine
    SPI_DWordRead = HAL_SPI_ReadWrite(0x00)
    # Read the actual data byte. We pass a 0x00 into the SPI routine since it always
writes when it reads
    # Get the byte which was read by the low-level MCU routine
    DWordTemp = HAL_SPI_ReadWrite(0x00)
    # Put the byte into a 32-bit variable (shifted 8 bits up)
    SPI_DWordRead |= (DWordTemp << 8)
```

```
    # Read the actual data byte. We pass a 0x00 into the SPI routine since it always
writes when it reads
    # Get the byte which was read by the low-level MCU routine
    DWordTemp = HAL_SPI_ReadWrite(0x00)
    # Put the byte into a 32-bit variable (shifted 16 bits up)
    SPI_DWordRead |= (DWordTemp << 16)
    # Read the actual data byte. We pass a 0x00 into the SPI routine since it always
writes when it reads
    # Get the byte which was read by the low-level MCU routine
    DWordTemp = HAL_SPI_ReadWrite(0x00)
    # Put the byte into a 32-bit variable (shifted 24 bits up)
    SPI_DWordRead |= (DWordTemp << 24)
    # Return the byte which we read
    return(SPI_DWordRead)


  def READ_8():
    # Read the data byte. We pass a 0x00 into the SPI routine since it always writes
when it reads
    # SPI_Writebyte = 0x00;
    # Get the byte which was read by the low-level MCU routine
    SPI_Readbyte = HAL_SPI_ReadWrite(0x00)
    # Return the byte which we read
    return(SPI_Readbyte)
  def HostCommand( Host_Command):
    # Chip Select Low
    HAL_SPI_CSLow()
    # This is the command being sent
    SPI_Writebyte = (Host_Command & 0x3F | 0x40)
    HAL_SPI_ReadWrite(SPI_Writebyte)
    # Sending dummy byte
    # SPI_Writebyte = 0x00
    HAL_SPI_ReadWrite(0x00)
    HAL_SPI_ReadWrite(0x00)
    # Chip Select High
    HAL_SPI_CSHigh()
  def DummyRead():
    #   byte PortRead = 0x00;
    #  byte SPI_byte_Read = 0x00;
    # SPI_Writebyte = 0x00
    # Chip Select Low
    HAL_SPI_CSLow()
    # Read/Write
    # Sending dummy 00 byte
    HAL_SPI_ReadWrite(0x00)
    # Sending dummy 00 byte
    HAL_SPI_ReadWrite(0x00)
    # Sending dummy 00 byte
    HAL_SPI_ReadWrite(0x00)
    # Chip Select High
    HAL_SPI_CSHigh()
  def  FT800_IncCMDOffset(Current_Offset, Command_Size):
    New_Offset = Current_Offset + Command_Size
    if(New_Offset > 4095):
      New_Offset = (New_Offset - 4096)
    return New_Offset
  def ComEsperaFin():
    global CMD_Offset
    cmdBufferWr=0
    cmdBufferRd=1
    while(cmdBufferWr != cmdBufferRd):
      HAL_SPI_CSLow()
      SENDADDRESSRD(REG_CMD_WRITE)
      # Read the value of the REG_CMD_WRITE register
      cmdBufferWr = READ_32()
      HAL_SPI_CSHigh()
      HAL_SPI_CSLow()
      SENDADDRESSRD(REG_CMD_READ)
      # Read the value of the REG_CMD_READ register
```

```
      cmdBufferRd = READ_32()
      HAL_SPI_CSHigh()
      # Comparo buffer de escritura con lectura. Cuando sean iguales, habrá dejado la
pantalla de pintar
    # Check the actual value of the current WRITE register (pointer)
    CMD_Offset = cmdBufferWr
  def EscribeRam32( dato):
    global CMD_Offset
    HAL_SPI_CSLow()
    # Writing to next available location in FIFO (FIFO base address + offset)
    SENDADDRESSWR(RAM_CMD + CMD_Offset)
    # Vertex 2F    01XXXXXX XXXXXXXX XYYYYYYY YYYYYYYY
    WRITE_32(dato)
    HAL_SPI_CSHigh()
    # Move the CMD Offset since we have just added 4 bytes to the FIFO
    CMD_Offset = FT800_IncCMDOffset(CMD_Offset, 4)
  def EscribeRam16( dato):
    global CMD_Offset
    HAL_SPI_CSLow()
    # Writing to next available location in FIFO (FIFO base address + offset)
    SENDADDRESSWR(RAM_CMD + CMD_Offset)
    # Vertex 2F    01XXXXXX XXXXXXXX XYYYYYYY YYYYYYYY
    WRITE_16(dato)
    HAL_SPI_CSHigh()
    # Move the CMD Offset since we have just added 4 bytes to the FIFO
    CMD_Offset = FT800_IncCMDOffset(CMD_Offset, 2)
  def EscribeRam8( dato):
    global CMD_Offset
    HAL_SPI_CSLow()
    # Writing to next available location in FIFO (FIFO base address + offset)
    SENDADDRESSWR(RAM_CMD + CMD_Offset)
    # Vertex 2F    01XXXXXX XXXXXXXX XYYYYYYY YYYYYYYY
    WRITE_8(dato)
    HAL_SPI_CSHigh()
    # Move the CMD Offset since we have just added 4 bytes to the FIFO
    CMD_Offset = FT800_IncCMDOffset(CMD_Offset, 1)
  def EscribeRamTxt(cadena):
    i=0
    while(i<len(cadena)) :
      EscribeRam8(cadena[i])
      i=i+1
    EscribeRam8(0)
  def Comando( COMM):
    global CMD_Offset
    HAL_SPI_CSLow()
    # Writing to next available location in FIFO (FIFO base address + offset)
    SENDADDRESSWR(RAM_CMD + CMD_Offset)
    # Write the DL_START command
    WRITE_32(COMM)
    HAL_SPI_CSHigh()
    # Move the CMD Offset since we have just added 4 bytes to the FIFO
    CMD_Offset = FT800_IncCMDOffset(CMD_Offset, 4)
  def ComVertex2ff( x, y):
    """ Args:
        x(int): coordenada en x.
        y(int): coordenada en y.
       Returns: void
    """
    global CMD_Offset
    coord=x
    coord=coord<<19
    coord=coord+ (y<<4)
    HAL_SPI_CSLow()
    # Writing to next available location in FIFO (FIFO base address + offset)
    SENDADDRESSWR(RAM_CMD + CMD_Offset)
    # Vertex 2F    01XXXXXX XXXXXXXX XYYYYYYY YYYYYYYY
    WRITE_32(0x40000000+coord)
    HAL_SPI_CSHigh()
    # Move the CMD Offset since we have just added 4 bytes to the FIFO
```

```
    CMD_Offset = FT800_IncCMDOffset(CMD_Offset, 4)
  def ComColor( R, G, B):
    """ Args:
      R(int): 0-255 de tonalidad roja.
      G(int): 0-255 de tonalidad verde.
      B(int): 0-255 de tonalidad azul.
      Returns: void
    """
    if PANTALLA==3:
      color=(R<<8)+G
      color=(color<<8)+B
    else:
      color=(B<<8)+G
      color=(color<<8)+R
                                                                                #
Color RGB   00000100 BBBBBBBB GGGGGGGG RRRRRRRR  (B/G/R = Colour values)
    Comando(0x04000000+color)
  def ComTXT( x, y, fuente, ops, cadena):
    """ Args:
        x(int): coordenada en el eje X.
        y(int): coordenada en el eje Y.
        fuente(int):  rango permitido de 0 a 31, nos permite elegir la fuente.
        ops(int): valor 0 indica que las coordenadas son del pixel superior izquierdo,
tambi茅n puede tener el valor de: OPT_CENTERX, OPT_CENTERY.
        Cadena(char):texto a representar.
        Returns: void
    """
    EscribeRam32(CMD_TEXT)
    EscribeRam16(x)
    EscribeRam16(y)
    EscribeRam16(fuente)
    EscribeRam16(ops)
    i=0
    while(i<len(cadena)) :
      EscribeRam8(int.from_bytes(cadena[i].encode(),"big"))
      i=i+1
    EscribeRam8(0)
    PadFIFO()
  def ComNum( x, y, fuente, ops, Num):
    """ Args:
      x(int): coordenada en el eje X.
      y(int): coordenada en el eje Y.
      fuente(int):  rango permitido de 0 a 31, nos permite elegir la fuente.
      ops(int): valor 0 indica que las coordenadas son del pixel superior izquierdo,
tambi茅n puede tener el valor de: OPT_CENTERX, OPT_CENTERY. Para n煤meros con signo usar
OPT_SIGNED.
      Num(long): n煤mero a representar.
      Returns: void
    """
    EscribeRam32(CMD_NUMBER)
    EscribeRam16(x)
    EscribeRam16(y)
    EscribeRam16(fuente)
    EscribeRam16(ops)
    EscribeRam32(Num)
    PadFIFO()

  def ComTeclas( x, y, w, h, fuente, ops, Keys):
    """ Args:
      x(int): posición X del v茅rtice superior izquierdo del Scrollbar.
      y(int): posición Y del v茅rtice superior izquierdo del Scrollbar.
      w(int): longitud en X del Scrollbar.
      h(int): longitud en Y del Scrollbar.
      fuente(int): rango permitido de 0 a 31, nos permite elegir la fuente
      ops(int):  0 si queremos 3D, OPT_FLAT para quitarlo
      Keys(char): texto que queremos que aparezca en la tecla a crear
      Returns: void
```

```
  """
  EscribeRam32(CMD_KEYS)
  EscribeRam16(x)
  EscribeRam16(y)
  EscribeRam16(w)
  EscribeRam16(h)
  EscribeRam16(fuente)
  EscribeRam16(ops)
  i=0
  while(i<len(Keys)) :
    EscribeRam8(int.from_bytes(Keys[i].encode(),"big"))
    i=i+1
  EscribeRam8(0)
  PadFIFO()

def Ejecuta_Lista():
  """ Args: void
      Returns:void
  """
  HAL_SPI_CSLow()
  SENDADDRESSWR(REG_CMD_WRITE)
  WRITE_16(CMD_Offset)
  HAL_SPI_CSHigh()

def PadFIFO():
  """ Args: void
      Returns:void
  """
  pad=CMD_Offset & 0x0003
  if pad>0:
    i=0
    while i<4-pad :
      EscribeRam8(0)
      i=i+1
def Dibuja():
  """ Args: void
      Returns:void
  """
  Comando(CMD_DISPLAY)
  Comando(CMD_SWAP)
  Ejecuta_Lista()

def Inicia_pantalla():

  # Put the Power Down. pin high to wake FT800
  HAL_SPI_PDhigh()
  DELAY()

  # Read location 0 to wake up FT800
  DummyRead()
  # Change the PLL to external clock - optional
  #HostCommand(FT_GPU_EXTERNAL_OSC)
  # Ensure configured to 48 MHz
  HostCommand(FT_GPU_PLL_48M)
  DELAY()

  # Reset the core
  HostCommand(FT_GPU_CORE_RESET)
  DELAY()

  # Read address 0 to ensure FT800 is active
  HostCommand(FT_GPU_ACTIVE_M)
  # Read the Chip ID to check comms with the FT800 - should be 0x7C
  chipid = 0x00
  while(chipid != 0x7C):
    # CS low
    HAL_SPI_CSLow()
    # Send the address
    SENDADDRESSRD(REG_ID)
```

```
      # Read the actual value
      chipid = READ_8()
      # Si el programa esta pillado aqui, es que no esta bien conectada la placa. CS
high
      HAL_SPI_CSHigh()
      DELAY()
  # ============================================================
  # Write the display registers on the FT800 for your particular display
  # ============================================================
    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_HCYCLE)
    WRITE_16(HCYCLE)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_HOFFSET)
    WRITE_16(HOFFSET)
    HAL_SPI_CSHigh()
    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_HSYNC0)
    WRITE_16(HSYNC0)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_HSYNC1)
    WRITE_16(HSYNC1)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_VCYCLE)
    WRITE_16(VCYCLE)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_VOFFSET)
    WRITE_16(VOFFSET)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_VSYNC0)
    WRITE_16(VSYNC0)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_VSYNC1)
    WRITE_16(VSYNC1)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_SWIZZLE)
    WRITE_16(2)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_PCLK_POL)
    WRITE_16(PCLK_POL)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_HSIZE)
    WRITE_16(HSIZE)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_VSIZE)
    WRITE_16(VSIZE)
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
```

```
    SENDADDRESSWR(REG_PCLK)
    WRITE_16(PCLK)
    HAL_SPI_CSHigh()

    # ================================================================
                                                                        #
Configure the touch screen
                                                                        #
================================================================

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_TOUCH_RZTHRESH)
    WRITE_16(1200)
    HAL_SPI_CSHigh()

                                                                        #
================================================================
                                                                        #
Send an initial display list which will clear the screen to a black colour
                                                                        #
================================================================
                                                                        # Set
the colour which is used when the colour buffer is cleared

    HAL_SPI_CSLow()
    SENDADDRESSWR(RAM_DL+0)
    WRITE_32(0x02000000)
    HAL_SPI_CSHigh()
                                                                        #
Clear the Colour, Stencil and Tag buffers. This will set the screen to the 'clear'
colour set above.
    HAL_SPI_CSLow()
    SENDADDRESSWR(RAM_DL+4)
    WRITE_32(0x26000007)
    HAL_SPI_CSHigh()
                                                                        #
Display command ends the display list
    HAL_SPI_CSLow()
    SENDADDRESSWR(RAM_DL+8)
    WRITE_32(0x00000000)
    HAL_SPI_CSHigh()
                                                                        #
Writing to the DL_SWAP register tells the Display Engine to render the new screen
designed above.
    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_DLSWAP)
    WRITE_32(0x00000002)
    HAL_SPI_CSHigh()

                                                                        #
================================================================
    # Set the GPIO pins of the FT800 to enable the display now
                                                                        #
================================================================
                                                                        #
note: Refer to the GPIO section in the FT800 datasheet and also the connections to the
GPIO
                                                                        #
used on the particular development module being used
                                                                        # Bit
7 enables the LCD Display. It is set to output driving high
                                                                        # Bit
1 enables the Audio Amplifier. It is set to output driving low to shut down the amplifier
since audio not used here
                                                                        # Bit
0 is unused but set to output driving high.
    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_GPIO_DIR)
    WRITE_8(0x83)
```

```
    HAL_SPI_CSHigh()

    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_GPIO)
    WRITE_32(0x83)
    HAL_SPI_CSHigh()

  def Nueva_pantalla( R, G, B):
    """ Args:
      R(int): 0-255 de tonalidad roja.
      G(int): 0-255 de tonalidad verde.
      B(int): 0-255 de tonalidad azul.
      Returns: void
    """
    if PANTALLA==3:
      color=(R<<8)+G
      color=(color<<8)+B
    else:
      color=(B<<8)+G
      color=(color<<8) +R
    ComEsperaFin()
    Comando(CMD_DLSTART)
    # Clear Color RGB   00000010 BBBBBBBB GGGGGGGG RRRRRRRR  (B/G/R = Colour values)
    Comando(0x02000000+color)
    # Clear 00100110 -------- -------- -----CST   (C/S/T define which parameters to
clear))
    Comando(0x26000007)

  def Lee_pantalla():
    """ Args: void
      Returns:void
    """
    global POSX
    global POSY
    HAL_SPI_CSLow()
    SENDADDRESSRD(REG_TOUCH_SCREEN_XY)
                                                                                #Lee
valor pantalla tactil
    BufferXY = READ_32()
    HAL_SPI_CSHigh()
    POSX=(BufferXY & 0xffff0000)>>16
    POSY=BufferXY & 0x0000FFFF


  def Lee_Reg(dir):
    """ Args:
      dir(int): dirección de la que se desea leer el valor
      Returns: int
    """
    HAL_SPI_CSLow()
    SENDADDRESSRD(dir)
    Buff_temp = READ_32()          # Lee valor
    HAL_SPI_CSHigh()
    return(Buff_temp)


  def Esc_Reg( dir, valor):
   """ Args:
    dir(int):
    valor(int):
    Returns: void
   """
   HAL_SPI_CSLow()
   SENDADDRESSWR(dir)
   WRITE_32(valor)
   HAL_SPI_CSHigh()

  def ComScrollbar( x, y, w, h, ops, val, size, range):
```

```
    """ Args:
      x(int): posición X del v茅rtice superior izquierdo del Scrollbar.
      y(int): posición Y del v茅rtice superior izquierdo del Scrollbar.
      w(int): longitud en X del Scrollbar.
      h(int): longitud en Y del Scrollbar.
      ops(int): 0 si queremos efecto 3D, 256 si queremos anular efecto 3D.
      val(int):  posición inicial del scroll.
      size(int): tamaño del scroll.
      range(int): máximo valor permitido del scroll.
      Returns: void
    """
    EscribeRam32(CMD_SCROLLBAR)
    EscribeRam16(x)
    EscribeRam16(y)
    EscribeRam16(w)
    EscribeRam16(h)
    EscribeRam16(ops)
    EscribeRam16(val)
    EscribeRam16(size)
    EscribeRam16(range)

  def ComFgcolor( R, G, B):
    """ Args:
      R(int): 0-255 de tonalidad roja.
      G(int): 0-255 de tonalidad verde.
      B(int): 0-255 de tonalidad azul.
      Returns: void
    """
    #no hace falta poner color como long ya que python pasa automáticamente, en caso de
ser necesario de int a long.
    if PANTALLA==3:
      color=R
      color=(color<<8)+G
      color=(color<<8)+B
    else:
      color=B
      color=(color<<8)+G
      color=(color<<8)+R
    EscribeRam32 (CMD_FGCOLOR)
    EscribeRam32 (color)

  def ComBgcolor( R, G, B):
    """ Args:
      R(int): 0-255 de tonalidad roja.
      G(int): 0-255 de tonalidad verde.
      B(int): 0-255 de tonalidad azul.
      Returns: void
    """
    #no hace falta poner color como long ya que python pasa automáticamente, en caso de
ser necesario de int a long.
    if PANTALLA==3:
      color=R
      color=(color<<8)+G
      color=(color<<8)+B
    else:
      color=B
      color=(color<<8)+G
      color=(color<<8)+R
    EscribeRam32 (CMD_BGCOLOR)
    EscribeRam32 (color)

  def ComButton( x, y, w, h, font, ops, cadena):
    """ Args:
      x(int): posición X del v茅rtice superior izquierdo del botón.
      y(int): posición Y del v茅rtice superior izquierdo del botón.
```

```
    w(int): longitud en X del botón.
    h(int): longitud en Y del botón.
    font(int): fuente deseada.
    ops(int): pulsado(1), sin pulsar(0)
    cadena(char): palabras dentro del botón
    Returns: void
  """
  EscribeRam32(CMD_BUTTON)
  EscribeRam16(x)
  EscribeRam16(y)
  EscribeRam16(w)
  EscribeRam16(h)
  EscribeRam16(font)
  EscribeRam16(ops)
  i=0
  while(i<len(cadena)) :
    EscribeRam8(int.from_bytes(cadena[i].encode(),"big"))
    i=i+1
  EscribeRam8(0)
  PadFIFO()

def Boton(x, y, w, h,  font, cadena):
  """ Args:
    x(int): posición X del v茅rtice superior izquierdo del botón.
    y(int): posición Y del v茅rtice superior izquierdo del botón.
    w(int): longitud en X del botón.
    h(int): longitud en Y del botón.
    font(int): fuente deseada.
    cadena(char): palabras dentro del botón
    Returns:
    bool: The return value. True for press, False otherwise.
  """
  Lee_pantalla()
  if(POSX>x and POSX<(x+w) and POSY>y and POSY<(y+h)):
    ComButton(x,y,w,h,font,OPT_FLAT,cadena)
    return 1
  else:
    ComButton(x,y,w,h,font,0,cadena)
    return 0
def Espera_pant():
  """ Args: void
    Returns:void
  """
  while(POSY==0x8000):
    Lee_pantalla()
  DELAY()
  while(POSY!=0x8000):
    Lee_pantalla()

def ComLineWidth( width):
  """ Args:
    width(int): grosor de la línea.
    Returns:void
  """
  Comando(CMD_LINEWIDTH+width*16)


def ComPointSize(size):
  """ Args:
    size(int): tamaño del punto.
    Returns:void
  """
  Comando(CMD_POINTSIZE+size*16)
```

```
def Com_Punto( x, y, R):
  """ Args:
    x(int): posición X del punto.
    y(int): posición Y del punto.
    R(int): tamaño del punto.
    Returns:void
  """
  #import uctypes
  #x1=x | uctypes.UINT16
  #y1=y | uctypes.UINT16
  #R1=R | uctypes.UINT16
  #Point Size  00001101 00000000 SSSSSSSS SSSSSSSS  (S = Size value)
  Comando(CMD_POINTSIZE+R*16)
  Comando(CMD_BEGIN_POINTS)
  ComVertex2ff(x,y)
  Comando(CMD_END)



def Calibra_touch():
  """ Args: void
      Returns:void
  """
  from utime import sleep_ms
  ComEsperaFin()
  Comando(CMD_DLSTART)
  #Clear Color RGB   00000010 BBBBBBBB GGGGGGGG RRRRRRRR  (B/G/R = Colour values)
  Comando(0x02f7f7f7)
  #Clear 00100110 -------- -------- -----CST  (C/S/T define which parameters to clear))
  Comando(0x26000007)
  ComColor(0x00,0x00,0x00)
  ComTXT(60,30,27,0,"Pulsa en el punto")
  EscribeRam32(CMD_CALIBRATE)
  Dibuja()
  sleep_ms(RELOJ/6)
  #espera(500)
  ComEsperaFin()
  Comando(CMD_DLSTART)
  #Clear Color RGB   00000010 BBBBBBBB GGGGGGGG RRRRRRRR  (B/G/R = Colour values)
  Comando(0x02ffff00)
  #Clear 00100110 -------- -------- -----CST  (C/S/T define which parameters to clear))
  Comando(0x26000007)
  Dibuja()

def ComGradient(x0, y0, color0, x1, y1, color1):
  """ Args:
    x0(int): posición X inicial.
    y0(int): posición Y inicial.
    color0(int): color en el inicio.
    x1(int): posición en X final.
    y1(int): posición en Y final.
    color1(int): color en el final.
    Returns:void
  """
  EscribeRam32(CMD_GRADIENT)
  EscribeRam16(x0)
  EscribeRam16(y0)
  EscribeRam32(color0)
  EscribeRam16(x1)
  EscribeRam16(y1)
  EscribeRam32(color1)



def VolNota(volumen):
```

```
    """ Args:
      volumen(char): volumen al que queremos que suene el siguiente sonido que se
programe
      Returns:void
    """
    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_VOL_SOUND)
    WRITE_8(volumen)
    HAL_SPI_CSHigh()


  def TocaNota(instr, nota):
    """ Args:
     instr(int): instrumento que deseamos, valor num茅rico de este seg煤n queda definido
por el fabricante.
     nota(int): nota que deseamos tocar. Valor num茅rico definido por el fabricante
     Returns:void
    """
    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_SOUND)
    WRITE_16(nota+(instr<<8))
    HAL_SPI_CSHigh()
    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_PLAY)
    WRITE_8(1)
    HAL_SPI_CSHigh()


  def FinNota():
    """ Args: void
          Returns:void
    """
    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_SOUND)
    WRITE_16(0x0000)
    HAL_SPI_CSHigh()
    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_PLAY)
    WRITE_8(1)
    HAL_SPI_CSHigh()

  def Fadeout():
   from utime import sleep_ms
   i=100
   while i>=0:
     HAL_SPI_CSLow()
     SENDADDRESSWR(REG_PWM_DUTY)
     WRITE_8(i)
     HAL_SPI_CSHigh()
     sleep_ms(RELOJ/1500)
     i-=3
    #API to perform display fadein effect by changing the display PWM from 0 till 100
and finally 128
  def Fadein():
    from utime import sleep_ms
    i=0
    while i<=100:
      HAL_SPI_CSLow()
      SENDADDRESSWR(REG_PWM_DUTY)
      WRITE_8(i)
      HAL_SPI_CSHigh()
      sleep_ms(RELOJ/1500)
      i+=3
    HAL_SPI_CSLow()
    SENDADDRESSWR(REG_PWM_DUTY)
    WRITE_8(128)
    HAL_SPI_CSHigh()
  def ComLineas(xo,yo, xf, yf):
```

```
  Comando(CMD_BEGIN_LINES)
  ComVertex2ff(xo,yo)
  ComVertex2ff(xf,yf)
  Comando(CMD_END)


def color(R, G, B):
  """ Args:
    R(int): 0-255 de tonalidad roja.
    G(int): 0-255 de tonalidad verde.
    B(int): 0-255 de tonalidad azul.
    Returns: void
  """
  if PANTALLA==3:
    colr=R
    colr=(color<<8)+G
    colr=(color<<8)+B
  else:
    colr=B
    colr=(color<<8)+G
    colr=(color<<8)+R
  return colr
```

REFERENCIAS

[1] Inforoadictos(2013). La pantalla táctil: sus orígenes. Consultado en:
https://inforadictos.com/la-pantalla-tctil-sus-orgenes/

[2] Wikipedia(2019). Apple Newton. Consultado en:
https://es.wikipedia.org/wiki/Apple_Newton

[3] Xataca(2014). Feliz veinte cumpleaños, IBM Simon Personal: la imagen de la semana. Consultado en:
https://www.xataka.com/historia-tecnologica/feliz-veinte-cumpleanos-ibm-simon-personal-la-imagen-de-la-semana-1

[4]Circuitbank. Nintendo DS. Consultado en:
https://circuitbank.com/products/nintendo-ds

[5]Wikipedia(2019). Pilot 1000. Consultado en:
https://es.wikipedia.org/wiki/Pilot_1000

[6]Wikipedia(2019). Pantalla táctil. Consultado en:
https://es.wikipedia.org/wiki/Pantalla_t%C3%A1ctil#Tipos

[7] Andrew Przybylski, «Las pantallas digitales son ahora una parte intrínseca de la infancia moderna, de la generación de los llamados nativos digitales» revista Psychological Science, Universidad de Oxford, 2017.

[8] Adafruit(2019). Adafruit HUZZAH32-ESP32 Feather Board. Consultado en:
https://www.adafruit.com/product/3619

[9] FTDChip(2019). VM800B-FT800. Consultado en:
https://www.ftdichip.com/Products/Modules/VM800B.html

[10] Información obtenida de las páginas Adafruit(2019) y BRTChip(2019). Consultado en:
https://www.adafruit.com/product/3619

https://brtchip.com/product/vm800b35a-bk/

[11] Python(2019).Welcome to Python. Consultado en:
https://www.python.org/

[12] Python(2019). Download Python. Consultado en:
https://www.python.org/

[13] FábricaDigital(2019). Adafruit Huzzah32- ESP32 Feather. Consultado en:
https://fabricadigital.org/tutoriales/adafruit-huzzah32-esp32-feather/

[14] FTDiChip (2014). FT800 Series programmer guide (pág. 188). Consultado en:
https://www.ftdichip.com/Support/Documents/ProgramGuides/FT800%20Programmers%20Guide.pdf

[15] FTDiChip (2014). FT800 Series programmer guide (pág 195). Consultado en:
https://www.ftdichip.com/Support/Documents/ProgramGuides/FT800%20Programmers%20Guide.pdf

Bibliografía

Páginas consultadas a la hora de realizar el estudio sobre la historia de las pantallas táctiles, así como sus tipos y clasificaciones:

https://inforadictos.com/la-pantalla-tctil-sus-orgenes/

https://www.xatakamovil.com/desarrollo/pantallas-tactiles-capacitivas-vs-resistivas

https://www.lainformacion.com/salud/investigacion-medica/exposicion-pantallas-consecuencias-negativas-adolescentes_0_990802422.html

https://www.tucanaldesalud.es/es/voz-especialista/efectos-uso-excesivo-pantallas-tactiles

https://es.wikipedia.org/wiki/Pantalla_t%C3%A1ctil#Tipos

Documentación empleada para conocer y asentar cierta base en el campo de la programación en MicroPython, además de su importancia respecto a otros lenguajes:

https://teslabem.com/blog/que-es-micropython/

https://aprendiendoarduino.wordpress.com/tag/mosi/

http://ligdigonzalez.com/por-que-utilizar-python-para-machine-learning/

https://www.instructables.com/id/MicroPython-and-UPyCraft-on-ESP32/

https://brtchip.com/vm800b-ft800-development-platform/

Páginas empleadas como ayuda a la hora de programar la librería, ya sea por el uso de ciertas funciones externas como por conocer los requisitos de la pantalla usada:

http://docs.micropython.org/en/v1.9.3/wipy/library/machine.Pin.html

http://docs.micropython.org/en/v1.9.3/wipy/library/utime.html

http://docs.micropython.org/en/v1.9.3/wipy/library/machine.html

https://www.ftdichip.com/Support/Documents/ProgramGuides/FT800%20Programmers%20Guide.pdf

https://docs.micropython.org/en/latest/library/uctypes.html

https://www.ftdichip.com/Support/Documents/AppNotes/AN_275_FT800_Example_with_Arduino.pdf

https://cdn-learn.adafruit.com/downloads/pdf/adafruit-huzzah32-esp32-feather.pdf

https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT800.pdf

https://docs.micropython.org/en/latest/esp32/tutorial/intro.html

ÍNDICE DE CONCEPTOS

No se encuentran entradas de índice.

GLOSARIO