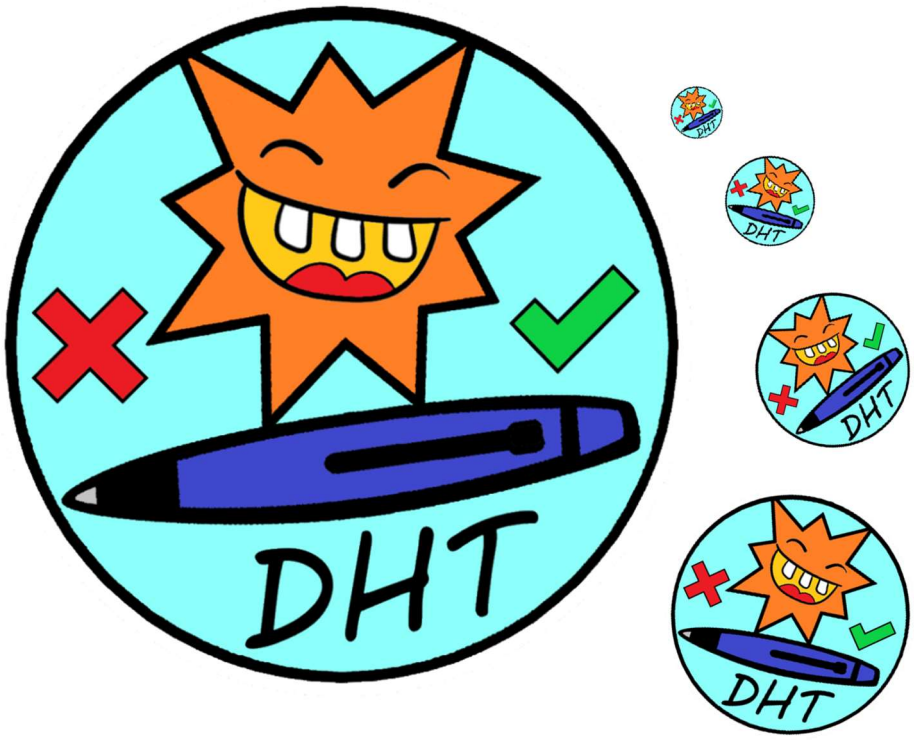
Digital HandWriting Trainer



Proyecto Final

SISTEMAS ELECTRÓNICOS PARA AUTOMATIZACIÓN (SEPA)

4º GIERM

SERGIO LEÓN DONCEL | ÁLVARO GARCÍA LORA

Índice

Introducción. Presentación del proyecto

Descripción de idea:

Si miramos a un tiempo atrás no muy lejano, se podía ver como los estudios eran algo sólo al alcance de los agraciados. Gran parte de la población era analfabeta y esto lastraba el progreso de la sociedad.

Hoy en día, se tiene la suerte en la mayoría de lugares del planeta de contar con una educación. Y no tan solo eso, sino que, de la mano del gran avance tecnológico vivido, el aprendizaje es cada vez más rápido.

Dentro de esta situación, nos surgió la idea de ayudar a los niños/niñas que están aprendiendo a escribir, haciéndoles el camino más interactivo y amigable, además de acelerar su aprendizaje y motivación para la superación de sus capacidades. Todo esto a la vez que se familiarizan con la tecnología.

Tradicionalmente el aprendizaje de la escritura se ha hecho en las escuelas mediante cuadernillos físicos. Esta forma de trabajar ha obligado a las familias y centros a tener que adquirir numerosas cantidades de libros de actividades, fichas y recursos en papel en las primeras etapas de la educación. Este método puede ser sustituido por sistemas como el propuesto, ahorrando materiales, pudiéndose reutilizar tantas veces como el alumno necesite y pudiéndose pasar a diferentes cursos a lo largo de los años. Todo ello supone una inversión más que rentable para los centros educativos y familias.

Por ello, presentamos nuestro producto de Digital Handwriting Trainer. Consiste en un dispositivo el cual permite dibujar en una pantalla táctil siguiendo las figuras que se muestran, proporcionando calificaciones y un registro del progreso que van alcanzando los más pequeños.

Extensión a otras aplicaciones:

- Tableta gráfica para dibujos: Hojas blancas o cuadriculadas para dar rienda suelta a la imaginación, con selección de colores y tipos de lápiz. Inclusión de imágenes para rellenar coloreando.
- Cuadernillo digital completo: Adición del abecedario, figuras geométricas …

Fases y coordinación de tareas

Fases del proyecto afrontadas en orden temporal:

- Creación de librería para configuración de periféricos
- Reproducción de sonidos con sintetizador de la pantalla
- Diseño, codificación y almacenamiento de imágenes propias en RAM de pantalla
- Lectura de RAM de pantalla para uso de imágenes propias en el sistema
- Planteamiento, desarrollo y experimentos de distintos algoritmos para la resolución del problema de escritura
- Crítica para elección sobre mejor algoritmo y decisión del método a seguir representación de escritura y calificaciones
    - Lectura digito actual de zona de RAM de pantalla
    - Transformación a matriz de dígito actual en tamaño original de imagen
    - Transformación a matriz en color dinámica
    - Codificación de zonas acertadas, fallos, superficie sin rellenar y fondo en matriz dinámica
    - Representación de matriz dinámica
    - Almacenamiento de información de ejercicios en matriz codificada para cálculo de calificaciones
    - Detección de pruebas no válidas, posibles trampas
- Diseño de interfaces gráficas y máquina de estados
- Uso de memoria ROM no volátil, salvar progreso
- Optimización de código y medidas experimentales de tiempos para cumplir tiempo de refresco objetivo venciendo limitación de lista instrucciones pantalla
- Animaciones a partir de imágenes propias
- Introducción de clave al sistema con teclado táctil
- Planteamiento de algoritmo codificación SHA256 hash para claves

En cuanto a división y coordinación del trabajo, en líneas generales hemos trabajado de forma conjunta en las distintas fases con reparto de tareas dentro de cada una para lograr alcanzar la resolución de cada una de ellas de forma más rápida, efectiva y con vistas a conseguir mejores soluciones gracias a críticas y realimentación por ambas partes.

Hemos trabajado de forma más independiente (con revisión el uno del otro) en ciertas tareas particulares:

- Sergio:
    - Librería de configuración de periféricos
    - Funciones y subrutinas de codificación y decodificación de matrices para lectura y escritura en RAM
    - Optimización y testeo de tiempos de ejecución
    - Teclado táctil y encriptación hash de claves

- Álvaro:
  - Máquina de estados
  - Desarrollo de funciones relacionadas con cálculo de calificaciones, progresos y lectura/escritura de RAM de pantalla
  - Diseño e implementación de interfaces gráficas con funciones y subrutinas asociadas
  - Uso de memoria no volátil
  - Diseño de material audiovisual

Solución hardware adoptada

Elección de microcontrolador

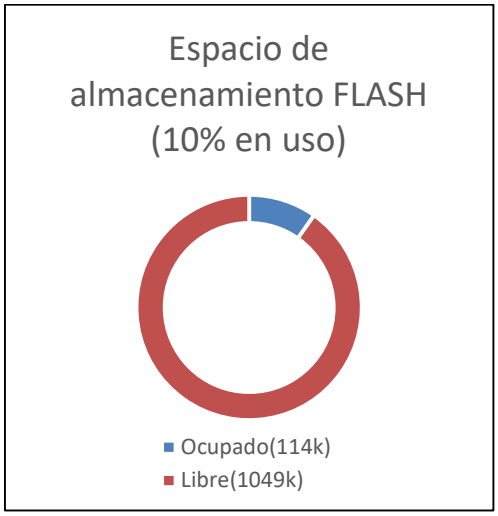
Para nuestro sistema particular, el uso del microcontrolador y placa proporcionados en la asignatura supone una solución simple y barata a la vez que potente, ajustada a la medida de nuestra aplicación embebida.

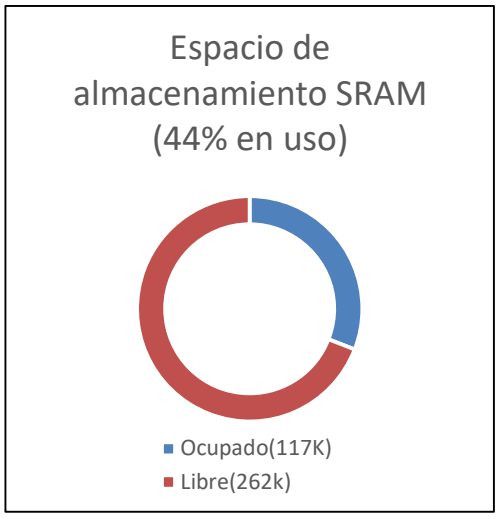
Podemos conseguir un tiempo de refresco táctil aceptable para el uso requerido y velocidades de ejecución y procesamiento más que convenientes.

En cuanto a memoria, se dispone de un amplio espacio disponible para el desarrollo de las tareas necesarias que debe realizar el programa, pudiendo adoptar soluciones de alto nivel por medio de librerías sin estar preocupados por agotar los recursos.

Además, permite que el programa sea ampliamente escalable, ya que el espacio sobrante podría utilizarse para ampliar el banco de imágenes, añadir nuevas funcionalidades más avanzadas como la lectura SD por medio del puerto USB, uso de ethernet para protocolo IP de comunicación en Internet…

El uso actual del procesador es:





Uso de periféricos

- Timer: Temporización para modo bajo consumo y medida de tiempos transcurridos para distintas utilidades en el desarrollo del programa.

- EEPROM: Memoria de solo lectura programable y borrable eléctricamente. Se trata de una memoria no volátil de 6kB para almacenar el progreso del alumno y configuración del sistema.

- CCM: Módulo de aceleración por hardware criptográfico para descargar al micro y reducir tiempo en el cálculo de AES (estándar de encriptación avanzada), DES (estándar de encriptación de datos) y SHA/MD5 (algoritmos de hash seguros). Se utiliza para codificar la clave en un hash de 256 bits según el algoritmo estándar SHA256, y almacenarlo en ROM para el acceso al sistema.

- QSSI (modo SPI): Comunicación microcontrolador – pantalla.

Pantalla FT800

Utilizada para la HMI (interfaz humano-máquina). Se hace uso de su memoria RAM gráfica de 256 kB, tanto en lectura como escritura, para la carga de imágenes, transformaciones de escala, y translaciones de movimiento para crear animaciones, a través de órdenes ejecutadas por el módulo VM800B.

También se ha hecho uso de su motor de sonidos, el cual permite generar efectos sonoros a través de un sintetizador, controlado por el valor de los registros de volumen y ajuste de tono.

Además, para poder cargar las imágenes en el microcontrolador (que no cuenta con sistema de archivos) y transferirlas a memoria RAM de pantalla, se ha utilizado un programa soporte del fabricante FTDI, *img_cvt,* que genera imágenes en *rawh* (formato que almacena información de imagen en un vector de bytes) para incluirlas en el código.

Por último, pero quizá lo más importante, su utilidad como pantalla táctil resistiva para poder detectar pulsaciones de forma que se registren los trazos realizados y permitir dibujar.

Explicación del programa

Librería propia de configuración

Para una mejor estructuración del código, he desarrollado mi propia librería para la configuración de periféricos, de forma que se añada un nivel más de abstracción para simplificar todo y compactar el setup de un periférico en una única función.

Para alcanzar dicho objetivo, se ha hecho uso de las funciones variádicas de C, las cuales pueden recibir un número indefinido de parámetros, ya que utilizan la pila (stack) de tipo LIFO para leer argumentos en orden.

Para su uso es necesario una librería de la biblioteca estándar de C, llamada *stdarg*, que incluye funciones como va_start (permite el acceso a los argumentos), *va_arg* (accede al siguiente argumento), *va_end* (finaliza el recorrido de argumentos)…

Hay que tener en consideración que se ha presupuesto un uso correcto de dichas funciones, lo que quiere decir que debe coincidir el número de argumentos que se indica que son pasados, así como el tipo de variable que se espera. Un uso incorrecto podría generar una falla que bloquee al microcontrolador.

Las funciones disponibles en esta librería son:

-*ConfiguraReloj120MHz():* Dado que en la gran mayoría de ocasiones, se desea aprovechar la máxima potencia del micro y esto siempre se realiza con la misma llamada a *SysCtlClockFreqSet*, me ha parecido oportuno utilizar esta función sin argumentos para abstraerse de los diferentes parámetros de configuración del reloj. Utiliza el oscilador externo de 25MHz junto al PLL a 480 MHz para conseguir los 120 MHz de reloj interno. Devuelve la velocidad conseguida realmente.

-*HabilitaPerifericos():* Se trata de una función variádica, por lo que su primer argumento será el número de argumentos útiles que recibirá en la llamada. Seguidamente, los siguiente serán las macro correspondiente a cada uno de los periféricos que se pretenden habilitar usando la función *SysCtlPeripheralEnable.*

-*defineTipoPines():* Nuevamente una función variádica cuyo primer argumento es el número de argumentos útiles a recibir en la llamada. Después se pasarán como argumentos (en bloque de 3 parámetros para cada pin), consecutivamente:

+Tipo de pin: 0 si de desea definir pin de salida o 1 para entrada.

+Puerto: macro de puerto asociado.

+Pin: macro de pin deseado.

-*ConfiguraTimer()*: Utilizada para activar timer con el periodo deseado (indicado en milisegundos) y con rutina de interrupción asociada a su desbordamiento. Las características invariantes de los timers inicializados con esta función son:

+ La fuente del timer será en cualquier caso el reloj interno de 16 MHz.

+Se seleccionará el modo periódico y Split (16 bits).

+Se aplicará un preescalado a la fuente de 256 (16M/256=62.5 kHz).

-*ModoBajoConsumo():* Permite utilizar el modo de bajo consumo del procesador, en el cual el procesador utiliza la técnica de optimización de energía denominada clock gating (que elimina o ignora la señal de reloj). Se trata de una función variádica que necesita el número de argumentos útiles y los periféricos que se mantienen activos (y que por tanto pueden despertar mediante interrupción al micro de este modo).

- *ConfiguraPantalla():* Recibe el número del boosterpack al que la pantalla está conectado y la frecuencia de reloj del micro. No debe llamarse si no se conecta la pantalla a dicho boosterpack ya que sino bloquea la ejecución del programa.

- *ConfiguraSensorsBoosterpacck():* Recibe el número de boosterpack al que se conecta el Sensors Boosterpack y la frecuencia de reloj del micro. No debe llamarse si no se conecta dicha placa a dicho boosterpack ya que sino bloquea la ejecución del programa.

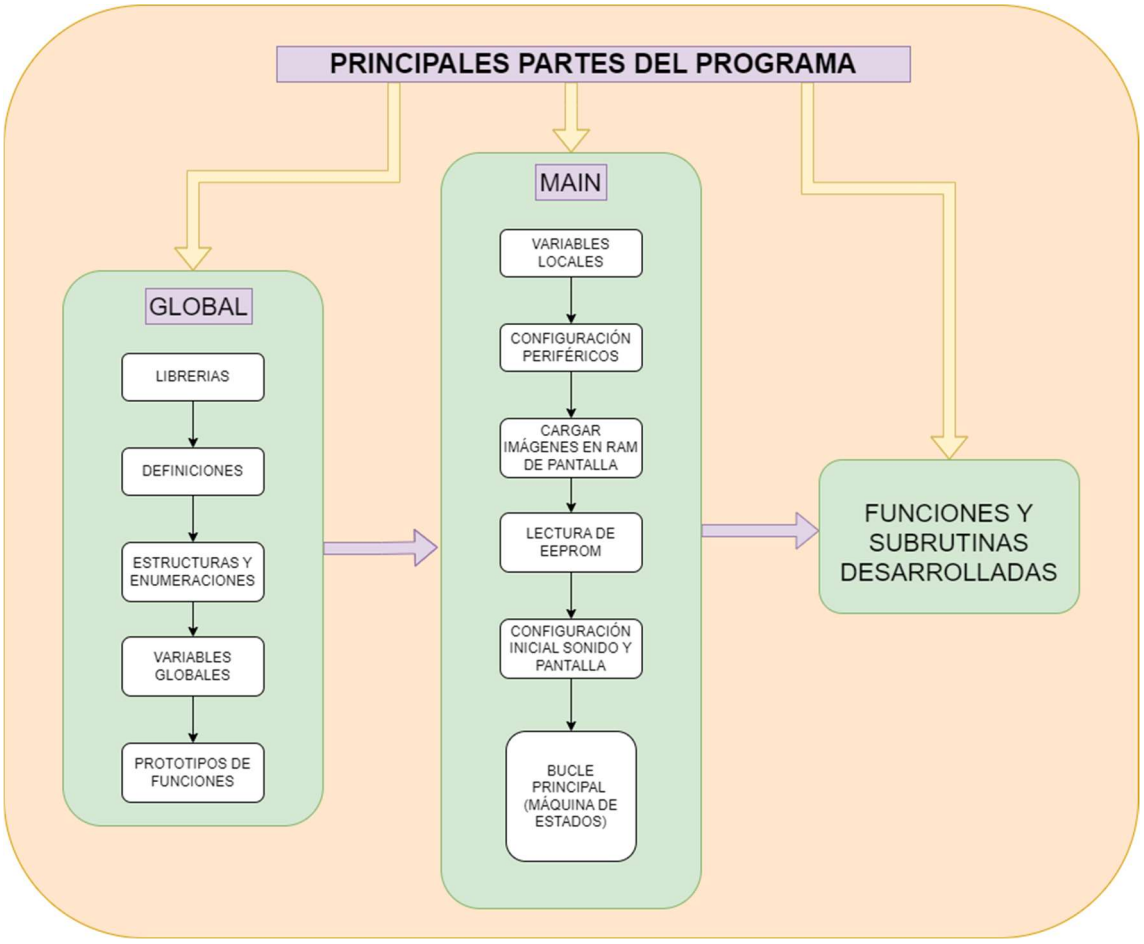
- *ConfiguraPWM():* Permite configurar salida PWM a la frecuencia indicada del puerto y pin indicados.

- *ConfiguraEEPROM():* Su única función es la de renombrar la función EEPROMInit, que se encarga de recuperarse si el último apagado de alimentación fue durante escritura.

- *ConfiguraCCM_SHA256():* Prepara el uso del módulo CCM para utilizar el algoritmo SHA256 para encriptación.

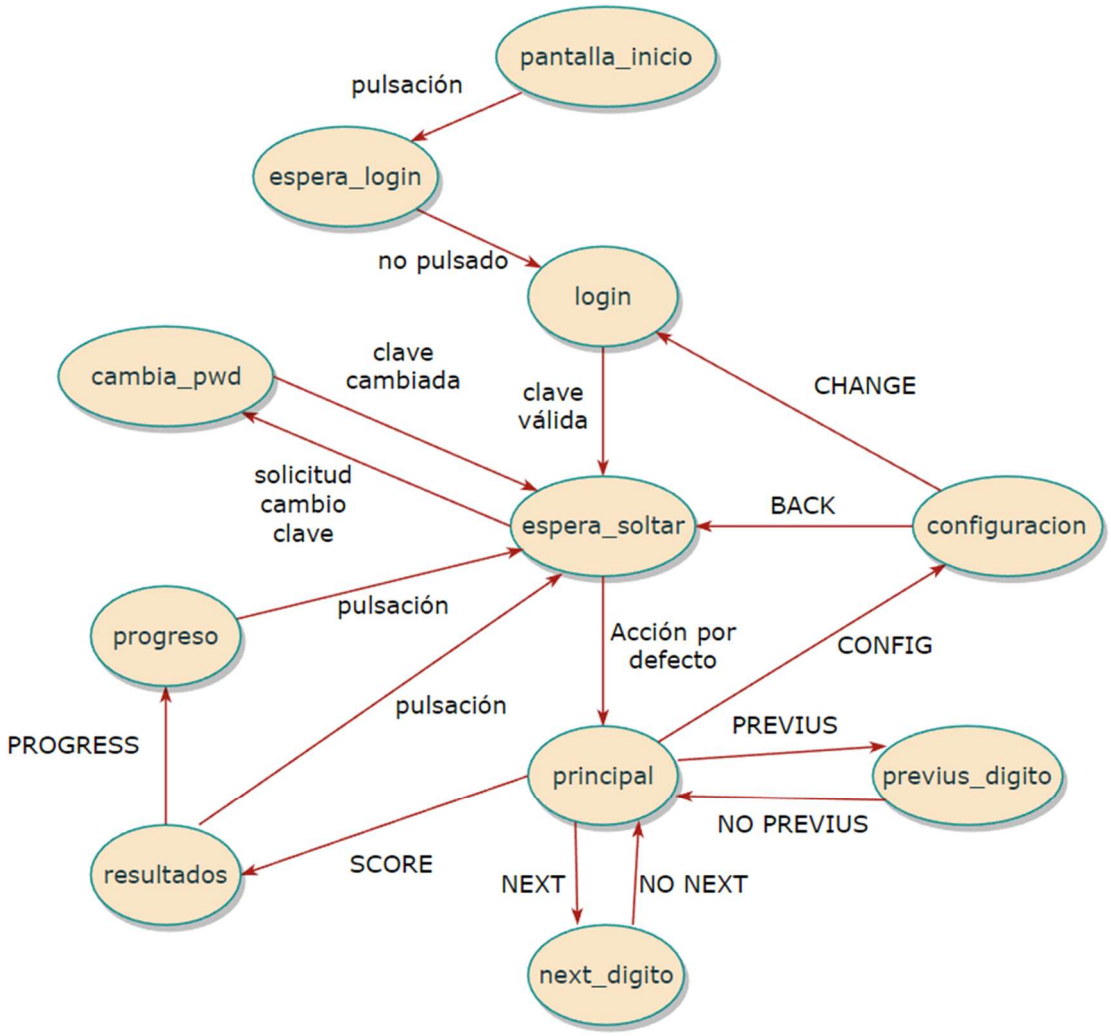
Estructura general

La estructura del programa desarrollado y su distribución en código queda resumida en el esquema que se muestra:



Máquina de estados

Centrándonos en la máquina de estados implementada en la función principal, la cual supone la base del proyecto, presentamos los estados con los que cuenta y recogemos las transiciones entre los mismos en el siguiente diagrama:



De forma resumida, realizamos una breve explicación de cada uno de los estados:

- pantalla_inicio: Mostramos animación inicial realizada con imagen propia cargada en RAM de pantalla junto al nombre de nuestro prototipo de producto. Cuenta con una melodía de sonido continua realizada con el sintetizador de la pantalla. Se permanece en este estado hasta que se pulse cualquier parte de la pantalla.

- espera_login: Espera hasta que se levanta el lápiz produciéndose el fin de la pulsación, pudiendo pasar al estado de inicio de sesión.

- login: Estado en el cual debe introducirse una clave correcta almacenada de forma codificada en memoria no volátil para proceder al desbloqueo del dispositivo. Se permanece en este estado hasta que el usuario introduzca la clave válida.

- espera_soltar: Usado por varios estados para implementar la espera necesaria hasta que se levante el lápiz de la pantalla tras una pulsación que supone un cambio de estado. Es deseable esta espera para evitar pulsaciones indeseadas en las distintas interfaces asociadas a estados diferentes.

- principal: Supone el estado en el cual se representa la interfaz gráfica asociada a la realización del ejercicio de repaso de dígitos para el aprendizaje y práctica del alumno. Cuenta con la zona de dibujo sobre el dígito, multitud de botones para las funciones de borrado de pantalla, siguiente y anterior dígito, acceso al panel de configuración, petición de la calificación correspondiente al ejercicio actual realizado y selección de grosor de lápiz.

- next_digito: Implementa una espera hasta fin de pulsación del botón asociado en el estado principal. Cambia al siguiente dígito reiniciando todas las estructuras de datos necesarias para representación y tratamiento de datos para calificación.

- previus_digito: Idéntico al estado next_digito pero decrementando el dígito actual.

- resultados: Muestra por pantalla una calificación a la actuación realizada por el alumno en el ejercicio actual. Además, proporciona un mensaje distinto en función del rango en el que se encuentre la nota obtenida. En caso de no dibujar un cierto porcentaje en comparación a la superficie del dígito en cuestión se muestra un mensaje de error. Con esto se pretende realizar una aproximación a la evitación de trampas. Dicha identificación tiene en cuenta el grosor del trazo seleccionado. Tiene nuevamente en cuenta el fin de la pulsación del botón asociado. Cuenta con botones para regresar a la pantalla principal para continuar con los ejercicios y para acceder al registro de mejores marcas obtenidas.

- progreso: Se muestra la lista con las mejores calificaciones conseguidas en cada dígito y el progreso total y restante respecto al máximo de calificaciones posibles, proporcionado al alumno o al tutor al cargo del menor la información necesaria para conocer el progreso alcanzado en el proceso de aprendizaje. Esta información se encuentra almacenada en memoria no volátil y cambia de forma continua con el funcionamiento del producto.

- configuracion: Implementa un panel de configuración donde es posible cambiar el volumen de las notificaciones y melodías del sistema. Así mismo, cuenta con las opciones de reset del progreso actual alcanzado y el cambio de clave del dispositivo. Los cambios realizados se almacenan en la memoria no volátil.

- cambia_pwd: Cumple con la función del cambio de contraseña del sistema, se accede al mismo cuando desde el menú de configuración se pulsa el botón de cambiar la clave y se introduce la clave actual, pasando por tanto por el estado de login.

Clasificación de funciones/subrutinas

Las funciones y subrutinas desarrolladas pueden clasificarse atendiendo al papel que juegan para la consecución de cada una de las partes principales del proyecto:

Almacenamiento imágenes en RAM de pantalla

Como se mencionaba anteriormente, para poder disponer de las imágenes en el programa del microcontrolador sin sistema de ficheros, se inicializan las imágenes como vectores en bytes (tipo char con valor entre 0-255).

Gracias a la herramienta img_cvt, codificamos las imagénes de los distintos números en formato L1, en el cuál cada byte del vector contiene 8 píxeles (en orden de más a menos significativo).

Por tanto, en nuestro programa hemos inicializado una matriz donde cada fila se corresponde al vector asociado a la imagen del dígito correspondiente. Dada que las imágenes utilizadas son 64x64, la longitud de los vectores es de 512 elementos (4096 píxeles).

Previo a la escritura en memoria RAM de este vector, guardamos información sobre la imagen a cargar en una estructura *caracteristicalmagen*, cuyos campos son addr (dirección de memoria donde se guarda la imagen), format (formato L1,L8,RGB332…), dim (estructura que contiene las dimensiones horizontal y vertical de la imagen).

Por último, se crea un buffer de 4 bytes (mediante desplazamiento de bits) para cargarlo en memoria RAM de pantalla mediante la función *Esc_Reg()*, y en un bucle que recorre bloques de 4 en 4 bytes del vector, cargamos cada una de las imágenes de los dígitos a partir de la dirección de memoria RAM_G (0x0).

De una manera parecida, se carga también el logo de nuestra aplicación en la dirección de memoria consecutiva libre, en formato RGB332 (donde cada píxel se corresponde a un byte de tal manera que los 3 bits más significativos codifican el color rojo, los 3 siguientes el verde y los 2 últimos el azul), con un tamaño de 100x100. En total esto significa un vector de 10 kB, que nuevamente se escribe en RAM gracias a *Esc_Reg()* a partir de la dirección 0x1400.

Por tanto, tras ejecutar la función cargar_imagenes(), en la memoria gráfica de la pantalla tendremos el siguiente mapa de memoria:

0x0 (RAM_G) hasta 0x200	Imagen Dígito 0 (Formato L1)
0x200 hasta 0x400	Imagen Dígito 1 (Formato L1)
0x400 hasta 0x600	Imagen Dígito 2 (Formato L1)
0x600 hasta 0x800	Imagen Dígito 3 (Formato L1)
0x800 hasta 0xA00	Imagen Dígito 4 (Formato L1)
0xA00 hasta 0xC00	Imagen Dígito 5 (Formato L1)
0xC00 hasta 0xE00	Imagen Dígito 6 (Formato L1)
0xE00 hasta 0x1000	Imagen Dígito 7 (Formato L1)
0x1000 hasta 0x1200	Imagen Dígito 8 (Formato L1)
0x1200 hasta 0x1400	Imagen Dígito 9 (Formato L1)
0x1400 hasta 0x1600	Imagen Logo (Formato RGB332)

Representación de imágenes y animaciones

Explicación básica de cómo se lee de RAM y representa una imagen propia por pantalla. Aplicación de eso para pantalla de inicio y extensión a animaciones.

Una vez que tenemos las imágenes almacenadas en memoria RAM de la pantalla usada haremos un uso continuo de las mismas en la interfaz principal para realizar los ejercicios y en la pantalla inicial creando incluso animaciones con ellas.

Para hacer esto debemos acceder a la dirección de comienzo de cada imagen en la memoria, las cuales son conocidas y tenemos a nuestra disposición para escribir el contenido en la zona del mapa de memoria de la pantalla encargada de la representación de todos los gráficos (Graphics Engine Command Buffer).

La función dibuja_imagen() es la encargada de realizar esto mismo. Recibe una estructura cuyos campos son las características de la imagen (dirección de comienzo en RAM de pantalla, formato de codificación de sus datos (en nuestro caso L1 o REGB332), y dimensiones de la misma), factor de escala que aplicaremos, y coordenadas del vértice superior izquierdo. Hacemos uso de la función Comando() que disponemos en la librería ft800_TIVA.c y resuelve la comunicación por SPI con la pantalla. Aplicando las macros adecuadas somos capaces de seleccionar la dirección (CMD_BITMAP_SOURCE), el formato de codificación y disposición (CMD_BITMAP_LAYOUT), de aplicar el factor de escala (CMD_BITMAP_TRANSFORM_A, CMD_BITMAP_TRANSFORM_E y CMD_BITMAP_SIZE), posicionar la imagen (CMD_VERTEX2II). Para dibujar el bitmap en pantalla (CMD_BEGIN_BMP).

Cálculo de resultados y progreso

Disponemos de la matriz matrizDibujoRealizado de tamaño igual al de las imágenes de los dígitos originales (64 x 64 pix). Cada componente de la matriz representa por tanto un valor asociado a cada pixel de la imagen del dígito concreto.

La idea principal para el cálculo de calificaciones reside en el contenido de esta matriz. Cada vez que se pinta sobre el número y se actualiza la matriz asociada a la imagen dinámica tras la comparación con la imagen original para saber si se está pintando sobre superficie de dígito o sobre el fondo (acierto y fallo respectivamente) guardamos dicha información también en esta matriz. Esto se hace en la función ejercicio_dibujar().

La matriz se inicializa y resetea en el programa cuando es necesario con la información de matriz_imagen_original, la cual es la replica en matriz del contenido codificado en memoria RAM de cada imagen de número usada en cada momento del programa.

El código establecido es el siguiente:

- Fondo sin pintar (zona negra de imagen dinámica):
  - Corresponde a un 0 en matrizDibujoRealizado.
- Superficie de dígito sin pintar (zona blanca de imagen dinámica):
  - Corresponde a un 2 en matrizDibujoRealizado.

- Zona acertada sobre dígito (Verde):
  - Corresponde a un 1 en matrizDibujoRealizado.
- Zona fallada fuera de dígito (Roja):
  - Corresponde a un -1 en matrizDibujoRealizado.

De esta forma, disponemos de información en todo momento de la cantidad de pixeles en dimensiones originales (ya que la escala, como estamos viendo simplemente es una cuestión para mejorar visualización sin hacer un uso excesivo de memoria, pero trabajamos en dimensiones reales) correspondientes a espacio no pintado, acierto, fallos y superficie del dígito sin colorear. Gracias a esto, mediante la función calculaScore() hacemos el cálculo de la puntuación obtenida tras terminar y confirmar ejercicio de pintado de un número concreto.

En la función comentada se hace un conteo del número de píxeles correspondientes a aciertos (cont_verde), fallos (cont_rojos) y superficie blanca intacta (cont_blanco).

Se considera la relación porcentual de pintado frente a superficie de dígito como:

$$\frac{\text{cont\_verde} + \text{cont\_rojo}}{\text{cont\_verde} + \text{cont\_blanco}} = \frac{\text{sup\_pintada}}{\text{sup\_digito}}$$

Aplicando un umbral dependiente al grosor del lápiz usado podemos exigir que la relación mostrada supere un determinado valor para tomar la prueba como válida. En caso contrario se mostrará un mensaje de error.

Si la prueba resulta correcta, el cálculo de la calificación se ha implementado como la relación existente entre el número de aciertos y el número total de píxeles coloreados:

$$\frac{\text{cont\_verde}}{\text{cont\_verde} + \text{cont\_rojo}} = \frac{\text{aciertos}}{\text{total}}$$

En función del resultado obtenido se muestra un mensaje complementario a la calificación.

RANGO CALIFICACIÓN	MENSAJE
10	PERFECT !!
9.5 – 10	AMAZING
8.5 – 9.5	WELL DONE
7 – 8.5	NICE
5 – 7	MORE OR LESS
0 - 5	TRY AGAIN

Las mejores marcas son almacenadas para tener un registro del progreso del alumno. La lista con dichas calificaciones máximas obtenidas se implementa con la subrutina progress_list().

Además, mostramos el progreso total de forma gráfica mediante el uso de una barra progresiva. Esto se consigue con la llamada a la función ComProgressBar(), a la cual pasamos como argumentos la posición y dimensiones requeridas, el valor de la

variable asociada al progreso total y el rango de representación. Esto proporciona una idea intuitiva de la lejanía o cercanía a la que se encuentra el menor de completar completamente el ejercicio de escritura manual propuesto de forma perfecta.

Configuración

A través de la pantalla principal, tenemos un botón de CONFIG que tras ser pulsado cambia el estado de la máquina a *configuracion*. Dentro de este, se espera hasta que este botón haya dejado de ser pulsado, momento en el cual se muestra:

- Mensaje superior de "CONFIGURATION PANEL".

- Slider en pantalla para actualizar el valor de la variable *volumen.* Además, si este cambia se escribe el nuevo valor en la memoria EEPROM en la dirección 0x0 para que tras un reinicio de alimentación el valor se mantenga.

- Botón RESET para reseteo de calificaciones, en el cual se comprueba mediante flag que se haya pulsado y dejado de pulsar, para resetear los valores del vector *best_scores* a 0 y escribir dichos valores nulos en EEPROM a partir de la dirección 0x8. Además, se ha incorporado un variable contador de tiempo *t* (que se incrementa en la rutina de interrupción del timer, por lo que cuenta de 50 en 50 ms) que mantenga el mensaje DONE hasta alcanzar el valor de 60 (3 segundos de tiempo transcurrido).

- Botón CHANGE de cambio de contraseña (que nuevamente espera mediante bandera a ser dejado de pulsar) que cambiará el estado a login para que se vuelva a solicitar la contraseña antes de permitir cambiarla, por seguridad, y activará *flagChangePWD* para indicar al este último estado que debe avanzar hacia el estado *cambia_pwd*.

- Botón BACK que permite volver a la pantalla principal.

Interfaz gráfica

El sistema cuenta con imágenes propias, animaciones de las mismas, una serie de botones táctiles para la transición cómoda entra pantallas, barra de progreso, barra deslizante para selección de grosor del lápiz, así como multitud de mensajes informativos.

En cuento al grosor disponible, brindamos la oportunidad al usuario de modificarlo de acuerdo con su gusto o nivel de dificultad con el que desee practicar. Usamos la función ComSlider() indicándole la variable a modificar, las dimensiones de la barra deslizante y el rango.

Se ha escogido como idioma el inglés para fomentar el aprendizaje pasivo a través de lo que para los niños puede resultar un juego, de forma que se familiaricen con palabras básicas del lenguaje.

El diseño del logotipo se ha realizado para que resulte atractivo a los niños.

Los números se han almacenado en blanco y negro para reducir la cantidad de memoria ocupada (512 bytes de formato L1 frente a los 4096 bytes que ocuparía una imagen de tamaño 64x64 pix en formato RGB332).

Teclado en pantalla

Para la implementación del teclado en pantalla, utilizado para el login y cambio de contraseña en nuestra aplicación, se ha dividido el software en una función de alto nivel, *teclado()*, que realiza las llamadas necesarias a las funciones *ComKeys()*, que dibuja la fila de botones con las teclas que se indiquen en la cadena pasada como parámetro, y *Keys()*, utilizada para redibujar con efecto FLAT si la tecla ha sido pulsada y devuelve el valor en codificación ASCII pulsada, o *NO_PULSADO* (valor de una enumeración) en el caso correspondiente.

La función de alto nivel teclado incluye también 2 botones que hacen de teclas "especiales" ya que estas no devuelven un valor en ASCII ni se corresponden a ningún carácter, sino que se utilizan como ENTER (para avanzar) o DEL (para borrar último carácter).

A bajo nivel, podemos mencionar:

- *ComKeys():* Con la función EscribeRam32 utiliza el puerto SPI para comunicarse con la pantalla y enviarle el comando CMD_KEYS con los parámetros x,y (esquina superior izquierda donde comienza la fila de teclas), w,h (anchura y altura de toda la fila de botones), font (fuente utilizada), options (efecto plano, 3D, centrado en x o en y…), cadena de caracteres con cada uno de los caracteres a colocar en cada tecla por orden.

- *Keys():* En primera instancia comprueba la longitud de la cadena que recibe, para luego poder ir recorriendo tecla a tecla y comprobar si la pantalla ha sido pulsada en el lugar asociado. Devuelve como resultado 0 (NO_PULSADO) o el valor en codificación ASCII del carácter correspondiente a la tecla.

Codificación de clave

Dado que en nuestra aplicación contamos con una clave para el acceso, se necesita guardar en memoria no volátil (EEPROM en nuestro caso) un valor con el que contrastar la contraseña introducida para ver si es correcta o no.

Siguiendo los estándares de seguridad aconsejados en la actualidad, no es en absoluto buena idea almacenar en texto plano en memoria la contraseña de acceso, ya que podría ser leída por terceros fácilmente. Por tanto, se plantea la opción de utilizar una función criptográfica apropiada para el microcontrolador que no sea viable romper. Una opción interesante, ya que se dispone del módulo CCM que nos acelera el proceso y es capaz de utilizar funciones hash actuales recomendadas por la NSA (Agencia Nacional de Seguridad), es recurrir a SHA256.

Este algoritmo es capaz de dar una salida única (denominada hash) para cada entrada que recibe (independientemente de la longitud de los datos), con un tamaño fijo de 32 bytes. Además, este hash es muy seguro, ya que realizar la desencriptación es muy complicado, 2 entradas con un único carácter de diferencia generarán 2 hashes muy diferentes entre sí.

La única manera de averiguar la contraseña sería ir probando entradas hasta obtener una salida con el hash deseado, lo cual para contraseñas que sigan los criterios adecuados es muy costoso computacionalmente y requeriría mucho tiempo.

Hecha esta breve justificación del camino elegido, se procede a explicar el estado *cambia_pwd,* en el que se codifica y almacena en EEPROM el hash obtenido a partir de la contraseña facilitada por el usuario. En este estado, se hace uso del teclado en pantalla explicado anteriormente, y a través de una cadena *stringTeclado* y un índice que apunta al elemento a escribir en la cadena, se guarda temporalmente en esta la contraseña introducida. Cuando se reciba pulsación de la tecla ENTER, se utilizará la función SHAMD5DataProcess() para generar el hash correspondiente a la clave introducida. Luego, byte a byte se escribirá en EEPROM el resultado para ser utilizado como hash esperado para contrastar con la clave aportada durante el login en el programa.

El estado login, por tanto, funciona de manera muy similar, con la diferencia de que ahora el hash generado a partir de la clave introducida será comparado con el hash esperado (previamente leído de la EEPROM), y se compararán byte a byte para en caso de coincidir permitir avanzar el estado de la máquina.

Uso memoria EEPROM

Hacemos uso de memoria no volátil para almacenar la configuración del sistema (volumen y clave) y nivel de aprendizaje alcanzado (lista de calificaciones y progreso total).

Usamos para ello las funciones EEPROMRead() (lectura) y EEPROMProgram() (escritura). Las direcciones usadas se encuentran recogidas en la siguiente tabla:

DIRECCIONES	CONTENIDO
0x0	Volumen
0x4	Progreso Total
0x8 – 0x30	Best Scores Vector
0x30 – 0x50	Hash de clave

Destacamos que almacenamos toda la información como tipo de dato unsigned int, sin embargo, en el desarrollo del programa hacemos uso de esos datos con valores decimales en caso de las calificaciones.

Durante el funcionamiento habitual el usuario puede modificar los valores almacenados en EEPROM al variar el volumen, practicar la escritura, o cambiar la clave.

Sintetizador

El sonido en el sistema se ha implementado por medio del sintetizador que cuenta la pantalla.

La pantalla inicial de bienvenida cuenta con una melodía continua, la interfaz principal de botones cuenta con sonidos de notificaciones al avanzar, retroceder, borrar el dibujo actual y acceder a la obtención de resultados. La idea es que suponga un atractivo añadido para los niños.

Además, en caso de cometer fallos saliéndose del digito concreto con el que se esté practicando suena una alarma que avisa del fallo, es decir, además de la representación de los píxeles en rojo se incluye este aviso sonoro.

Las funciones que usamos para gestionar el sonido usadas son:

- VolNota(): Elección del volumen (0-255)
- TocaNota(): Damos el tipo de sonido que se reproducirá y la nota del mismo. La tabla para selección de los mismos puede encontrarse en la documentación que proporciona el fabricante de la pantalla.
- FinNota(): Parar el sonido actual.

Verificación y optimización de tiempos (Sergio)

Para nuestra aplicación, el tiempo de respuesta del sistema es sumamente importante ya que este marcará el funcionamiento de nuestra captación del trazo dibujado.

Típicamente, hoy en día, los paneles táctiles utilizados para dibujar funcionan a una tasa de muestreo táctil de 120 Hz (8.3 ms). Como es evidente, para esta aplicación a mayor tasa, mejor funcionamiento ya que se dispondrá de menor pérdida de información en el trazo registrado. Este tiempo también dependerá del grosor del trazo (ya que esto caracteriza la resolución de la cuadrícula de celdas de pulsaciones táctiles).

Teniendo lo anterior en cuenta, era necesario saber el coste en tiempo de ejecución de nuestro programa, para saber a qué frecuencia poder cerrar la tasa de actualización (lo cual dependerá del periodo de timer configurado).

Para realizar su medición, se ha añadido al código la habilitación del timer integrado del sistema, SysTick (24 bits), configurado a 120 MHz, en la que cada ciclo de reloj se incrementa la variable *ticks*.

De esta forma, si guardamos al principio de nuestro bucle while el valor de ticks, recién despertados del SLEEP, y posteriormente justo antes de terminar la iteración calculamos la diferencia entre el valor anterior y el actual, tendremos en décimas de segundo el tiempo de ejecución de nuestro código.

Realizando estas pruebas, podemos ver como el tiempo de ejecución para el peor de los casos es de aproximadamente 20 ms.

Este tiempo, en su mayor parte, es debido a la comunicación SPI micro-pantalla en la cual se envían numerosas órdenes para representación en alguno de los estados (una opción para optimizar el programa y reducir este tiempo sería utilizar los 2 registros Macro, tal y como recomienda el fabricante, aunque no fue solución válida en nuestro caso al necesitar variar más de 2 parámetros de una pantalla a otra en el mismo estado, o utilizar apéndices en una versión más avanzada).

Por tanto, elegimos nuestra tasa de refresco en 25 ms. Con esto conseguimos captar el trazo sin problema, a una velocidad moderada (normal para uso cotidiano) en el caso de mínimo grosor, y a gran velocidad si se elige un trazo mayor.

Además, con esto se asegura que la respuesta que el usuario pueda esperar del sistema sea siempre la misma, haciendo más estable nuestro sistema.

Método/algoritmo seguido para imagen dinámica

Para conseguir registrar el trazo dibujado por el usuario en pantalla e ir representándolo de forma dinámica manteniendo lo previo, se probó inicialmente guardando en un vector los píxeles pulsados con la representación de los píxeles pulsados como círculos de radio pequeño. Sin embargo, dicha idea no fue factible, debido al breve espacio de almacenamiento (4 kB). Esto hacía que conforme se iba dibujando más y más la lista de instrucciones a pantalla aumentara, llegando un punto donde se superaba su límite.

Por tanto, se planteo una forma que debía ser más eficiente en cuanto a memoria de instrucciones se refiere. La solución alcanzada consiste en leer la imagen original del dígito para crear una copia a color en otra dirección de memoria de la RAM gráfica. Con esta copia, ir escribiendo los bytes correspondientes en memoria al píxel pulsado, de forma que se quede almacenado mostrando color verde para acierto y color rojo para fallo, el trazo realizado por el usuario. De esta manera, no incrementamos la lista de instrucciones conforme dibujamos. Además, se consiguió reducir considerablemente el tiempo de ejecución de nuestro código, cerrando así la tasa de actualización táctil a mayor velocidad y mejorando la respuesta del programa.

Entrando más en detalle en el algoritmo implementado, los pasos seguidos son:

1)Leer imagen original del dígito en memoria, descodificando el formato L1 en una matriz *matriz_imagen_original* de píxeles a 0 (negro) y 1(blanco). Para conseguirlo se utiliza Lee_Reg() para obtener 4 bytes de las direcciones de memoria correspondientes a la imagen. Se lee fila a fila (64 bits), separando en 32 píxeles de izquierda y derecha, y aplicando los desplazamientos en bits necesarios y la máscara de bit adecuada, se obtiene bit a bit cada píxel. Para esto se utiliza la función *LeeImagenFormatoL1*().

2)Convertir matriz binaria de imagen original en matriz a color (con 3 bits de color rojo, 3 bits de color verde y 2 bits de color azul). Para esto se utiliza la función *traduceMatriz2RGB*().

3)Codificamos la matriz de color RGB a vector en formato RGB332, el cual pasará a contener los datos de la imagen a color que deseamos meter en memoria RAM gráfica. Para esto se utiliza la función *codificaMatrizFormatoRGB332*().

4)Almacenamos este nuevo vector en memoria con la función *guardaImagenDin()*, la cual recorre el vector entero de 4 en 4 bytes, para ir escribiendo en la memoria gráfica de la pantalla en la zona de memoria reservada para la imagen dinámica.

5) Cíclicamente, se llama a la función ejercicio_dibujar en el estado principal del programa. En ella se llama a su vez a la función dibuja_imagen(). Supone una de las funciones más importantes de nuestro código ya que es en ella donde se realiza la comparación del lugar donde se produce las pulsaciones dentro del espacio permitido para el ejercicio.
Estas comparaciones se hacen entre la posición de la pulsación en pantalla, transformada al espacio de la imagen original para comprobar si se está pulsando en un lugar correspondiente al dígito (blanco, 1) o fuera del mismo (negro, 0). A partir de esta información se almacena los colores verde y rojo según el caso en la matriz dinámica que tenemos. Además, se almacena la información necesaria en la matriz para el cálculo de las calificaciones.
Únicamente los nuevos valores pintados son almacenados en memoria RAM para la actualización de la imagen, evitando de esa forma trabajar con la imagen completa y agilizando el proceso. También es en esta función donde se encuentra la llamada a las funciones de representación de toda la interfaz principal (botones y slider).

Cuando se solicita limpiar pantalla, se llama a la función *resetImagenDin()*, la cuál recurre a llamar a las funciones anteriores: *traduceMatriz2RGB()* para volver a tener la matriz a color de la imagen original del dígito, *codificaMatrizFormatoRGB332()* para volver a pasarla a formato RGB332 y vector, *guardaImagenDin()* para sobreescribir la imagen dinámica almacenada en memoria gráfica *y reseteaMatrizTrazoPintado()* para resetear la puntuación.

Resultados obtenidos

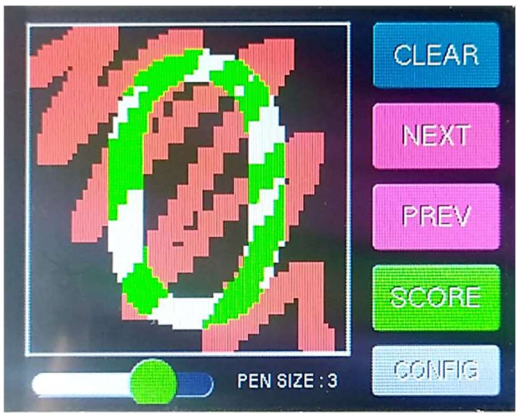
A continuación, mostraremos multitud de imágenes sacadas del sistema para mostrar los resultados conseguidos, interfaces y funcionalidades implementadas.

- Pantalla principal y animaciones:

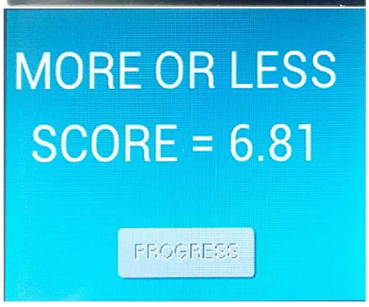
- Teclado e inicio de sesión:

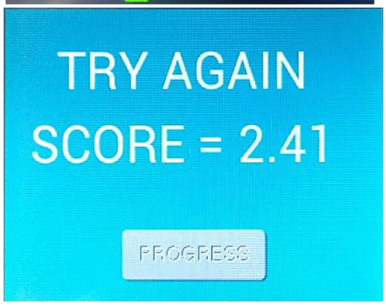
- Interfaz principal, espacio para practicar:

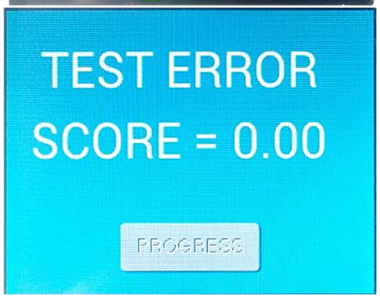
- Ejemplos de dibujado sobre dígitos

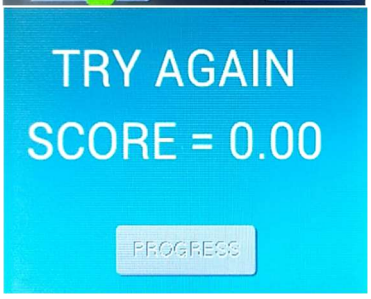


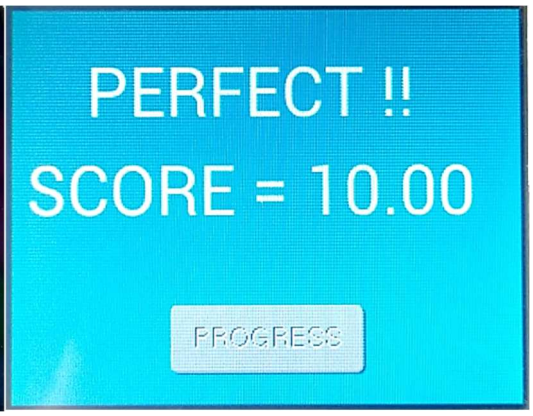
- Ejemplos de calificaciones





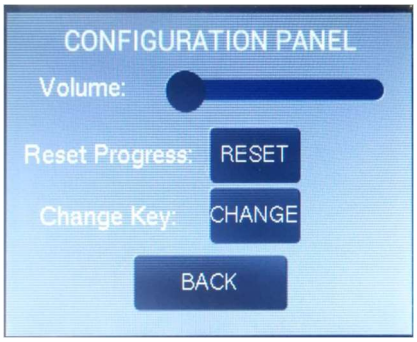


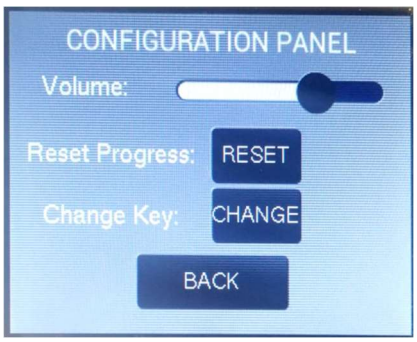


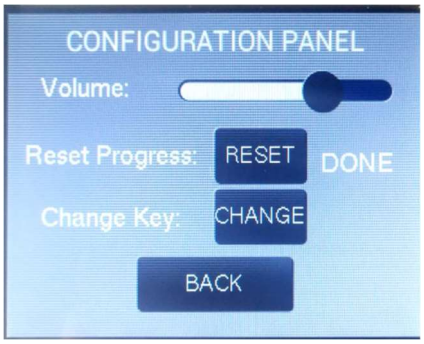


- Progreso alcanzado:

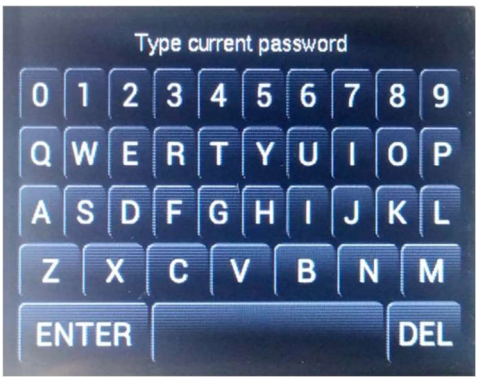
- Panel de configuración:

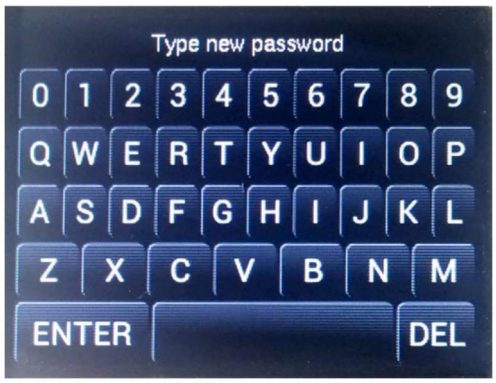






- Cambio de contraseña:





Manual de usuario

Aunque nuestro programa se trate de un prototipo, o versión inicial de producto, es capaz de ser de utilidad dentro de su ámbito. Por tanto, se procede a explicar el funcionamiento del sistema a un nivel de abstracción muy alto en el que sólo se especifica cómo usarlo, y no cómo se consigue (ya que esto ha sido anteriormente explicado para quién lo desee conocer).

-Al conectar el dispositivo a alimentación, la pantalla se encenderá con la **animación** de nuestro logo moviéndose y variando su tamaño a lo largo de la pantalla, a la vez que emitiendo una melodía.

-Si se pulsa la pantalla, se accederá a la interfaz de **login**. En esta se muestra un teclado por pantalla, en el cual se debe escribir la contraseña para iniciar sesión. Una vez introducida, se pulsará ENTER para comprobar si es correcta. En caso positivo, se accederá a la pantalla principal, sino no se mostrará ningún error, simplemente asegúrese de haber escrito bien la contraseña. Para cuando detecte el fallo, o si se equivocó previamente escribiendo, puede hacer uso de la tecla DEL para borrar el último carácter introducido y rectificar. Si es la primera vez que utiliza el dispositivo, la clave correcta será el valor por defecto de fábrica: *BETIS*.

-En la pantalla **principal**, tiene distintas utilidades. Podrá observar un recuadro a la izquierda en el cuál se muestra un dígito. Dentro de este, usted puede dibujar. El objetivo aquí será repasar la zona de imagen en blanco, de tal forma que se siga el dígito y de esta forma pueda aprender a escribirlo.

Comentar que dispone de una barra deslizante en la zona inferior para variar el grosor dentro de 4 niveles a su elección.

Por otro lado, mientras realiza el trazo podrá observar como de pasar por zona blanca se dibujará en verde (lo cual tendrá efecto positivo en la valoración de puntuación), pero si se equivoca y dibuja fuera en lo negro, aparecerá en rojo (lo que restará puntuación) además de emitir un pitido indicando que se está saliendo.

Si no le convence el intento realizado, puede hacer uso del botón en la esquina superior derecha *CLEAR* que le permitirá limpiar la imagen para realizar el intento desde cero nuevamente.

Si desea cambiar para practica con otro dígito, dispone de los botones *NEXT* para ir a la siguiente imagen o *PREV* para volver a la anterior.

Si ha terminado su intento, puede utilizar el botón *SCORE* para que le sea mostrada la puntuación adquirida. Debe ser consciente de que aplicamos un filtro para evitar trampas, de forma que no será considerado un intento válido si no ha terminado de escribir el dígito.

Por último, dispone de un apartado de configuración para hacerle cómodo el uso de nuestra solución, al que puede acceder pulsando el botón *CONFIG* de la esquina inferior derecha.

-En la interfaz de **puntuación**, puede ver un mensaje donde se le informa de su nota numérica obtenida (en una escala del 0 al 10). Para sacar la máxima nota debe haber completado el dígito (no es necesario hacerlo del mismo grosor que la imagen) sin salirse. En cambio, obtendrá una puntuación de 0 si ha trazado todo fuera del dígito. Si se le muestra el mensaje TEST ERROR, es porque ha saltado el filtro anti-cheats.

Pulsando en cualquier lugar de la pantalla menos el botón de *PROGRESS* accederá de nuevo a la interfaz principal. Si por el contrario desea que se le muestre una recopilación de resultados en todos los dígitos y su progreso, pulse el botón.

Una vez en la pantalla de **progreso**, puede volver a la principal pulsando en cualquier lugar.

-Si ha accedido al menú de **configuración**, dispone de distintas opciones:

+ Con una barra deslizante puede elegir el volumen del sonido, de forma que pueda anularlo por completo para jugar de forma silenciosa, o ponerlo al valor máximo para escuchar con claridad el sonido.

+ Con el botón de *RESET* puede borrar las puntuaciones registradas sobre su progreso hasta el momento. Se le mostrará por unos segundos un mensaje DONE confirmando la operación.

+ Con el botón CHANGE accederá a cambiar la clave del sistema. Debe saber que una vez lo pulse, le será requerida la clave actual y no podrá salir de este estado hasta que introduzca la contraseña correcta. Una vez introducida correctamente, se le pedirá introducir la nueva contraseña deseada. Finalmente, volverás a la pantalla inicial con la clave actualizada.

***SUMA IMPORTANCIA:** Las claves son guardadas en memoria permanente (no será borrado al reiniciar el sistema ni desconectarlo de la fuente de alimentación) cifradas. Sin embargo, no existe mecanismo para evitar fuerza-bruta, por lo que se recomienda encarecidamente utilizar una contraseña larga y complicada para evitar que personas sin permiso puedan acceder al dispositivo.

Anexo de los códigos fuente

DigitalHWTrainer.c

```
1.  #include <PheripheralConfigurationLibrary.h>
2.  #include "stdio.h"
3.  #include <string.h>
4.
5.
6.  //--------------------------------------------------------------------------
    -------------
7.
8.  #define SLEEP SysCtlSleep()
9.  //#define SLEEP SysCtlSleepFake()
10.
11. #define N_IMAGENES 11
12.
13. #define TIME 25
14.
15. //BOTONES
16. #define CLEAR Boton(230, 5, 80, 40, 22, "CLEAR")
17. #define NEXT Boton(230, 55, 80, 40, 22, "NEXT")
18. #define PREVIUS Boton(230, 105, 80, 40, 22, "PREV")
19. #define SCORE Boton(230, 155, 80, 40, 22, "SCORE")
20. #define CONFIG Boton(230, 205, 80, 30, 21, "CONFIG")
21. #define PROGRESS Boton(100, 180, 120, 40, 22, "PROGRESS")
22. #define BACK Boton(100, 180, 120, 40, 22, "BACK")
23. #define RESET Boton(160, 85, 70, 40, 22, "RESET")
```

```
24. #define CHANGE Boton(160, 130, 70, 40, 22, "CHANGE")
25.
26. //--------------------------------------------------------------------------
    -------------
27. // Estructuras y enumeraciones del programa
28. struct dimensiones
29. {
30.     int x;
31.     int y;
32. };
33.
34. struct caracteristicasImagen
35. {
36.     unsigned long addr;
37.     int format;
38.     struct dimensiones dim;
39. };
40.
41. enum formatoImagen
42. {
43.     L1 = 1,
44.     RGB332 = 4
45. };
46.
47. enum estados
48. {
49.     pantalla_inicio,
50.     espera_login,
51.     login,
52.     espera_soltar,
53.     principal,
54.     resultados,
55.     next_digito,
56.     previus_digito,
57.     progreso,
58.     configuracion,
59.     cambia_pwd
60. };
61. enum estados estado = pantalla_inicio;
62.
63. enum teclaEspecial
64. {
65.     DEL=-1,SPACE=-10,NO_PULSADO=0,ENTER=-100
66. };
67.
68. // Variables globales concretas del programa
69. bool Flag_ints = false;
70. int t = 0;
71. struct caracteristicasImagen atrImagenes[N_IMAGENES];
72. int matrizDibujoRealizado[64][64];
73. volatile long int ticks = 0;
74.
75. unsigned char vector_codificado[4096];
76. int matriz_imagen_original[64][64]; // Matriz 64*64 que contiene
    informacion del digito concreto extraido de la RAM de la pantalla
77. int matriz_imagen_din_RGB[64][64][3];          // Matriz 64*64 que contiene
    informacion del digito concreto extraido de la RAM de la pantalla
78.
79. // SONIDO
80. int xilofono = 0x08;
81. int nota_xilofono = 50;
82. int alarm = 0x010;
83. int nota_alarm = 100;
84. int scoreSound = 0x045;
85. int nota_score = 50;
```

```
int notify = 0x057;
int nota_notify = 50;
unsigned int volumen;
unsigned int volumenAnt;

char string[20];
unsigned int radioTrazo=0;
unsigned int best_scores[10];
unsigned int progreso_total;
//*****************************************************************************
//
//*****************************************************************************
// Prototipo de funciones
void IntTick(void)
  {
      ticks++;
  }
  void IntTimer1(void);
  void SysCtlSleepFake(void);
  extern void SysCtlSleep(void);
  void cargar_imagenes(void);
  void ejercicio_dibujar(void);
  void dibuja_imagen(struct caracteristicasImagen parametrosImagen, float factor_escala, int xUpperCorner, int yUpperCorner);
  void LeeImagenFormatoL1(struct caracteristicasImagen atrib_imagen);
  void traduceMatriz2RGB(void);
  void reseteaMatrizTrazoPintado(void);
  void codificaMatrizFormatoRGB332(void);
  void guardaImagenDin(void);
  void resetImagenDin(void);
  float calculaScore(void);
  void ComSlider(int x, int y, int w, int h, int options, unsigned int *val, int range);
  void progress_list(void);
  void ComProgressBar(int x, int y, int w, int h, int options, int val, int range);
  int teclado(void);
  void ComKeys(int16_t x, int16_t y, int16_t w, int16_t h, int16_t font, uint16_t options, const char *s);
  int Keys(int16_t x, int16_t y, int16_t w, int16_t h, int16_t font, const char *s);
  //*****************************************************************************

  int main(void)
  {
      //Variables locales main
      int i;
      int digitoActual = 0;
      float scoreRes = 0;
      char stringScore[20];
      bool flagClear = false;
      bool flagReset = false;
      bool flagChange = false;
      bool flagChangePWD = false;
      bool flagTime = false;
      bool flagDerecha = true;
      char desplaz = 0;
      int teclaPuls, teclaPulsAnt=NO_PULSADO,indTecla=0;
      char stringTeclado[50] = "Write password";
      unsigned int hashGen[8];
      unsigned int expectedHash[8];
      bool flagValida;
```

```
143.
144.        /*CONFIGURACION DE PERIFERICOS*/
145.        RELOJ = ConfiguraReloj120MHz();
                                                                    // Fijar la
    velocidad del reloj
146.        HabilitaPerifericos(7, SYSCTL_PERIPH_TIMER1, SYSCTL_PERIPH_GPIOF, SY
    SCTL_PERIPH_GPION, SYSCTL_PERIPH_PWM0, SYSCTL_PERIPH_GPIOG, SYSCTL_PERIPH_E
    EPROM0, SYSCTL_PERIPH_CCM0); //LISTA DE TODOS LOS PERIFERICOS UTILIZADOS
147.        defineTipoPines(3 * 2, 1, GPIO_PORTF_BASE, GPIO_PIN_0 | GPIO_PIN_4,
    1, GPIO_PORTN_BASE, GPIO_PIN_0 | GPIO_PIN_1);
148.        ConfiguraTimer(1, TIME, IntTimer1);
149.        ModoBajoConsumo(1, SYSCTL_PERIPH_TIMER1);
150.        ConfiguraPantalla(2, RELOJ);
151.        ConfiguraSensorsBoosterpack(1, RELOJ);
152.        ConfiguraPWM(PWM0_BASE, GPIO_PG0_M0PWM4, GPIO_PORTG_BASE, GPIO_PIN_0
    , 50);
153.        ConfiguraEEPROM();
154.        ConfiguraCCM_SHA256();      //Configuracion del modulo criptografico
    de aceleracion por hardware
155.
156.
157.        // Depuracion de tiempo en ejecucionn
158.        SysTickIntRegister(IntTick);
159.        SysTickPeriodSet(12000);
160.        SysTickIntEnable();
161.        SysTickEnable();
162.
163.        IntMasterEnable(); // Habilitacion global de interrupciones
164.
165.        //-------------------------------------------------------------------
    ------------------------------------
166.        // CARGAR IMAGENES EN RAM DE PANTALLA
167.        cargar_imagenes();
168.
169.        /*EJECUTAR PREVIAMENTE PARA INICIALIZAR MEMORIA ROM LA PRIMERA VEZ
    CON LOS VALORES PREDETERMINADOS DE FABRICA */
170.    //    unsigned int borrar=0;
171.    //    EEPROMProgram(&borrar, 0x0, sizeof(borrar)); //Escribir EEPROM
172.    //    EEPROMProgram(&borrar, 0x4, sizeof(borrar));
173.    //    int i;
174.    //    for(i=0;i<10;i++){
175.    //        EEPROMProgram(&borrar, 0x8+i*4, sizeof(borrar));
176.    //    }
177.    //    char stringAux[50] = "BETIS";
178.    //    SHAMD5DataProcess(SHAMD5_BASE, (uint32_t *) stringAux, 5,
    hashGen); //calcular hash a partir de algoritmo SHA256
179.    //    for(i=0;i<8;i++)
180.    //    {
181.    //        EEPROMProgram(&(hashGen[i]), 0x30+i*4, sizeof(hashGen[i]));
182.    //    }
183.
184.        //Leer EEPROM al comienzo del programa
185.        EEPROMRead(&volumen, 0x0, sizeof(volumen));
186.        volumenAnt=volumen;
187.        EEPROMRead(&progreso_total, 0x4, sizeof(progreso_total));
188.        for(i=0;i<10;i++){
189.            EEPROMRead(&best_scores[i], 0x8+i*4, sizeof(best_scores[i]));
190.        }
191.
192.        VolNota(volumen);
193.        TocaNota(xilofono, nota_xilofono);
194.
195.        dibuja_imagen(atrImagenes[10], desplaz, desplaz, 30);
196.
197.        // BUCLE PRINCIPAL
```

```
198.        while (1)
199.        {
200.            SLEEP; // dormir y despertar a los 50 ms para hacer sincrono el
    funcionamiento
201.            Lee_pantalla();
202.            switch (estado)
203.            {
204.
205.            case pantalla_inicio:
206.                Nueva_pantalla(0, 0, 0);
207.                //ANIMACI?N
208.                ComColor(255, 255, 255);
209.                ComTXT(160, 15, 22, OPT_CENTER, "DIGITAL HANDWRITING
    TRAINER");
210.                if(flagDerecha){
211.                    desplaz++;
212.                    if(desplaz>=240)
213.                        flagDerecha = false;
214.                }
215.                else{
216.                    desplaz --;
217.                    if(desplaz<=0)
218.                        flagDerecha = true;
219.                }
220.                dibuja_imagen(atrImagenes[10], (2-
    ((float)desplaz*0.005)), desplaz, 30);
221.
222.                Dibuja();
223.
224.
225.                if (POSY != 0x8000)
226.                {
227.                    LeeImagenFormatoL1(atrImagenes[digitoActual]);
228.                    resetImagenDin();
229.                    estado = espera_login;
230.                }
231.
232.
233.                break;
234.
235.            case espera_login:
236.                if (POSY == 0x8000){
237.                    FinNota();
238.                    estado = login;
239.                }
240.                break;
241.
242.            case login:
243.                Nueva_pantalla(0, 0, 0);
244.                teclaPuls=teclado();
245.                if(teclaPuls != NO_PULSADO && teclaPuls != teclaPulsAnt){
246.                    if(teclaPuls == ENTER)
247.                    {
248.                        if(indTecla>0)
249.                        {
250.                            SHAMD5DataProcess(SHAMD5_BASE, (uint32_t *) stri
    ngTeclado, indTecla-1, hashGen);
251.                            flagValida=true;
252.                            for(i=0;i<8;i++)
253.                            {
254.                                EEPROMRead(&(expectedHash[i]), 0x30+i*4, siz
    eof(expectedHash[i]));
255.
256.                                if(hashGen[i]!=expectedHash[i])
257.                                    flagValida=false;
```

```
                            }
                            if(flagValida)
                            {
                                estado = espera_soltar;
                                indTecla=0;
                            }

                        }

                    }
                    else if(teclaPuls == DEL){
                        stringTeclado[indTecla-1]='\0';
                        indTecla --;
                        if(indTecla<=0) indTecla=0;
                    }
                    else{
                        stringTeclado[indTecla]=(char)teclaPuls;
                        stringTeclado[indTecla+1]='\0';
                        indTecla ++;
                    }
                }
                teclaPulsAnt = teclaPuls;
                ComColor(255, 255, 255);
                ComTXT(160, 20, 21, OPT_CENTER, stringTeclado);
                Dibuja();


                break;

            case espera_soltar:
                if (POSY == 0x8000){
                    FinNota();
                    if(flagChangePWD)
                    {
                        strcpy(stringTeclado,"Write new password");
                        estado = cambia_pwd;
                    }
                    else
                    {
                        strcpy(stringTeclado,"Write current password");
                        estado =principal;
                    }
                }
                break;

            case principal:
                Nueva_pantalla(0, 0, 0);
                ejercicio_dibujar();
                Dibuja();

                //Borrar dibujado
                if(CLEAR){
                    flagClear=true;
                }
                if(!CLEAR && flagClear){
                    flagClear=false;
                    resetImagenDin();
                    TocaNota(notify, nota_notify);
                }
                //Siguiente digito
                if(NEXT){
                   digitoActual++;
                   if(digitoActual>9)
                       digitoActual=0;
                   TocaNota(notify, nota_notify);
```

```
323.                    estado =next_digito;
324.                }
325.                //Anterior digito
326.                if(PREVIUS){
327.                    digitoActual--;
328.                    if(digitoActual<0)
329.                        digitoActual=9;
330.                    TocaNota(notify, nota_notify);
331.                    estado = previus_digito;
332.                }
333.                //Resultados
334.                if(SCORE){
335.                    scoreRes = calculaScore();
336.                    // Actualiza progreso
337.                    if(best_scores[digitoActual] < (int)(scoreRes*100)){
338.                        best_scores[digitoActual] = scoreRes*100;
339.                        EEPROMProgram(&best_scores[digitoActual], 0x8+digitoA
    ctual*4, sizeof(best_scores[digitoActual]));
340.                    }
341.                    progreso_total = 0;
342.                    for(i=0;i<10;i++) progreso_total += (int)(best_scores[i]/
    100);
343.                    EEPROMProgram(&progreso_total, 0x4, sizeof(progreso_total
    ));
344.                    resetImagenDin();
345.                    TocaNota(scoreSound, nota_score);
346.                    estado = resultados;
347.                }
348.                //Configuracion
349.                if(CONFIG){
350.                    estado = configuracion;
351.                }
352.
353.                break;
354.
355.            case next_digito:
356.                if(!NEXT ){
357.                    LeeImagenFormatoL1(atrImagenes[digitoActual]);
358.                    resetImagenDin();
359.                    estado = principal;
360.                }
361.                break;
362.
363.            case previus_digito:
364.                if(!PREVIUS){
365.                    LeeImagenFormatoL1(atrImagenes[digitoActual]);
366.                    resetImagenDin();
367.                    estado = principal;
368.                }
369.                break;
370.
371.
372.            case resultados:
373.                if(!SCORE){
374.                    Nueva_pantalla(0, 153, 153);
375.                    //Score
376.                    sprintf(stringScore, "SCORE = %.2f", scoreRes);
377.                    ComColor(255, 255, 255);
378.                    ComTXT(150, 110, 31, OPT_CENTER, stringScore);
379.                    //Comentario
380.                    ComColor(255, 255, 255);
381.                    ComTXT(150, 50, 31, OPT_CENTER, string);
382.                    //Boton Acceso Panel Progreso
383.                    PROGRESS;
384.                    Dibuja();
```

```
385.                    if(POSY != 0x8000){
386.                        estado = espera_soltar;
387.                    }
388.                    if(PROGRESS){
389.                        resetImagenDin();
390.                        estado = progreso;
391.                    }
392.                }
393.                break;
394.
395.            case progreso:
396.                if(!PROGRESS){
397.                    Nueva_pantalla(200, 0, 100);
398.                    progress_list();
399.                    ComFgcolor(50,50,255);
400.                    ComTXT(272, 80, 18, OPT_CENTER, "TOTAL");
401.                    ComProgressBar(260, 105, 25, 100, 0, 100-
    progreso_total, 100);
402.                    Dibuja();
403.
404.                    if(POSY != 0x8000){
405.                        estado = espera_soltar;
406.                    }
407.                }
408.                break;
409.
410.            case configuracion:
411.                if(!CONFIG){
412.                    Nueva_pantalla(160, 160, 160);
413.                    ComTXT(160, 20, 28, OPT_CENTER, "CONFIGURATION PANEL");
414.                    // Volumen
415.                    ComSlider(140, 50, 150, 15, 0, &volumen, 255);
416.                    if(volumen != volumenAnt){
417.                        volumenAnt = volumen;
418.                        EEPROMProgram(&volumen, 0x0, sizeof(volumen)); //Wri
    te EEPROM
419.                        VolNota(volumen);
420.                    }
421.                    ComFgcolor(255,255,255);
422.                    ComTXT(60, 55, 23, OPT_CENTER, "Volume:");
423.                    // Reseteo calificaciones
424.                    ComTXT(80, 105, 23, OPT_CENTER, "Reset Progress:");
425.                    ComTXT(80, 140, 23, OPT_CENTERX, "Change Key:");
426.                    ComFgcolor(0,0,0);
427.                    RESET;
428.                    if(RESET && !flagReset)
429.                        flagReset = true;
430.                    if(!RESET && flagReset){
431.                        flagReset = false;
432.                        flagTime = true;
433.                        for(i=0;i<10;i++){
434.                            best_scores[i]=0;
435.                            EEPROMProgram(&best_scores[i], 0x8+i*4, sizeof(b
    est_scores[i]));
436.                        }
437.                        progreso_total = 0;
438.                        t=0;
439.                    }
440.                    if(t<((1000/TIME)*3) && flagTime){
441.                        ComTXT(270, 100, 23, OPT_CENTERX, "DONE");
442.                    }
443.                    else if(t>=(200*2) && flagTime)
444.                        flagTime = false;
445.
446.                    //Boton cambio de password
```

```
447.                    ComFgcolor(0,0,0);
448.                    if(CHANGE && !flagChange)
449.                    {
450.                        flagChange=true;
451.                    }
452.                    if(!CHANGE && flagChange)
453.                    {
454.                        flagChange=false;
455.                        flagChangePWD = true;
456.                        estado = login;
457.                    }
458.                    // Salida a principal
459.                    ComFgcolor(0,0,0);
460.                    BACK;
461.                    Dibuja();
462.
463.                    if(BACK)
464.                        estado = espera_soltar;
465.                }
466.                break;
467.
468.            case cambia_pwd:
469.                Nueva_pantalla(0, 0, 0);
470.                teclaPuls=teclado();
471.                if(teclaPuls != NO_PULSADO && teclaPuls != teclaPulsAnt){
472.                    if(teclaPuls == ENTER)
473.                    {
474.                        if(indTecla>0)
475.                        {
476.                            SHAMD5DataProcess(SHAMD5_BASE, (uint32_t *) stri
    ngTeclado, indTecla-1, hashGen);
477.                            flagValida=true;
478.                            for(i=0;i<8;i++)
479.                            {
480.                                EEPROMProgram(&(hashGen[i]), 0x30+i*4, sizeo
    f(hashGen[i]));
481.                            }
482.                            flagChangePWD=false;
483.                            indTecla=0;
484.                            estado =espera_soltar;
485.                        }
486.
487.                    }
488.                    else if(teclaPuls == DEL){
489.                        stringTeclado[indTecla-1]='\0';
490.                        indTecla --;
491.                        if(indTecla<=0) indTecla=0;
492.                    }
493.                    else{
494.                        stringTeclado[indTecla]=(char)teclaPuls;
495.                        stringTeclado[indTecla+1]='\0';
496.                        indTecla ++;
497.                    }
498.                }
499.                teclaPulsAnt = teclaPuls;
500.                ComColor(255, 255, 255);
501.                ComTXT(160, 20, 21, OPT_CENTER, stringTeclado);
502.                Dibuja();
503.
504.
505.                break;
506.
507.            default:
508.                break;
509.            }
```

```
510.        }
511.    }
512.
513.
514.    //-------------------------------------------------------------------------
    ---------------------------------------
515.
516.    void SysCtlSleepFake(void) // rutina de modo dormir falsa
517.    {
518.        while (!Flag_ints);
519.        Flag_ints = false;
520.    }
521.
522.    void IntTimer1(void) // rutina de interrupcion del timer 1
523.    {
524.        TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
525.        Flag_ints = true;
526.        t++;
527.        SysCtlDelay(10); // espera tiempo para dejar limpiar el registro
528.    }
529.
530.    void ejercicio_dibujar(void)
531.    {
532.        float factor_escala = 3.2;
533.        int xUpperCorner = 10;
534.        int yUpperCorner = 5;
535.        int dimX = 64;
536.        int dimY = 64;
537.        char stringWidth[20];
538.        long int tiempo = ticks;
539.        int x_ind, y_ind;
540.        int dirAsociada;
541.        int radio,fila,col;
542.        bool flagSonido = false;
543.
544.        radio = 0.3*radioTrazo; //Radio del trazo para pintar
545.        // Comprobar coordenada de localizacion, marcar fallo o acierto,
    actualizar imagen dinamica y reproducir nota
546.        if ((POSY != 0x8000) && (POSX > xUpperCorner && POSX < (dimX * facto
    r_escala + xUpperCorner) && POSY > yUpperCorner && POSY < (dimY * factor_es
    cala + yUpperCorner)))
547.        {
548.
549.            x_ind = (int)(POSX - xUpperCorner) / factor_escala;
550.            y_ind = (int)(POSY - yUpperCorner) / factor_escala;
551.            for(fila=y_ind-radio;fila<=y_ind+radio;fila++){
552.                for(col=x_ind-radio;col<=x_ind+radio;col++){
553.                    if(fila>=0 && fila<dimY && col>=0 && col<dimX){
554.                        if (matriz_imagen_original[fila][col]==1)
555.                        {
556.                            matrizDibujoRealizado[fila][col] = 1; // Zona
    blanca = Verde (1)
557.                            matriz_imagen_din_RGB[fila][col][0] = 0;
558.                            matriz_imagen_din_RGB[fila][col][1] = 7;
559.                            matriz_imagen_din_RGB[fila][col][2] = 0;
560.                        }
561.                        else
562.                        {
563.                            matrizDibujoRealizado[fila][col] = -1; // Zona
    negra = Roja (-1)
564.                            matriz_imagen_din_RGB[fila][col][0] = 7;
565.                            matriz_imagen_din_RGB[fila][col][1] = 0;
566.                            matriz_imagen_din_RGB[fila][col][2] = 0;
567.                            flagSonido = true;
568.                        }
```

```
                    }
                }
            }
            // Actualizo imagen dinamica
            for(fila=y_ind-radio;fila<=y_ind+radio;fila++){
                for(col=x_ind-radio;col<=x_ind+radio;col++){
                    if(fila>=0 && fila<dimY && col>=0 && col<dimX){
                        dirAsociada = (col + fila * 64);
                        vector_codificado[dirAsociada] = (matriz_imagen_din_RGB[fila][col][0] << 5) + (matriz_imagen_din_RGB[fila][col][1] << 2) + (matriz_imagen_din_RGB[fila][col][2]);
                        int imagen = (vector_codificado[dirAsociada + 3] << 24) | (vector_codificado[dirAsociada + 2] << 16) | (vector_codificado[dirAsociada + 1] << 8) | vector_codificado[dirAsociada];
                        Esc_Reg(RAM_G + 10 * 512 + 10000 + dirAsociada, imagen);
                    }
                }
            }
            if(flagSonido)
                TocaNota(alarm, nota_alarm);
        }
        else
            FinNota();

    dibuja_imagen((struct caracteristicasImagen){(RAM_G) + (10 * 512)+10000, RGB332, {dimX, dimY}}, factor_escala, xUpperCorner, yUpperCorner);

    // Dibujo rectangulo en blanco
    ComColor(255, 255, 255);
    ComRect(xUpperCorner, yUpperCorner, (dimX * factor_escala + xUpperCorner), (dimY * factor_escala + yUpperCorner), 0);

    // Boton de borrado
    ComFgcolor(0,150,150);
    CLEAR;

    // Boton siguiente digito
    ComFgcolor(255,0,127);
    NEXT;

    // Boton previo digito
    ComFgcolor(255,0,127);
    PREVIUS;

    // Boton resultados
    ComFgcolor(100,200,30);
    SCORE;

    // Slider grosor trazos
    ComSlider(20, 220, 100, 15, 0, &radioTrazo, 10);
    ComFgcolor(255,255,255);
    sprintf(stringWidth, "PEN SIZE : %d", radio+1);
    ComTXT(175, 225, 20, OPT_CENTER, stringWidth);

    // Boton configuracion
    ComFgcolor(160,160,160);
    CONFIG;

    }

    void dibuja_imagen(struct caracteristicasImagen parametrosImagen, float factor_escala, int xUpperCorner, int yUpperCorner)
```

```
626.    {
627.        ComColor(255, 255, 255);
628.
629.        int linestride;
630.        switch (parametrosImagen.format)
631.        {
632.        case L1:
633.            linestride = parametrosImagen.dim.x / 8;
634.            break;
635.        case RGB332:
636.            linestride = parametrosImagen.dim.x / 1;
637.            break;
638.        }
639.
640.        Comando(CMD_BITMAP_SOURCE + RAM_G + parametrosImagen.addr);
641.        switch (parametrosImagen.format)
642.        {
643.        case L1:
644.            Comando(CMD_BITMAP_LAYOUT + FORMAT_L1 + (linestride << 9) + para
    metrosImagen.dim.y);
645.            break;
646.        case RGB332:
647.            Comando(CMD_BITMAP_LAYOUT + FORMAT_RGB332 + (linestride << 9) +
    parametrosImagen.dim.y);
648.            break;
649.        }
650.        Comando(CMD_BITMAP_TRANSFORM_A + (int)(1 / factor_escala * 256));
651.        Comando(CMD_BITMAP_TRANSFORM_E + (int)(1 / factor_escala * 256));
652.        Comando(CMD_BITMAP_SIZE + (0 << 20) + (0 << 19) + (0 << 18) + ((int)
    (parametrosImagen.dim.x * factor_escala) << 9) + (int)(parametrosImagen.dim
    .y * factor_escala));
653.        Comando(CMD_BEGIN_BMP);
654.        Comando(CMD_VERTEX2II + (xUpperCorner << 21) + (yUpperCorner << 12)
    + (0 << 7) + 0);
655.        // SUMAMENTE IMPORTANTE DESHACER TRANSFORMACION ANTES DE REPRESENTAR
    FUENTES DE PANTALLA PARA TEXTOS
656.        Comando(CMD_BITMAP_TRANSFORM_A + 256);
657.        Comando(CMD_BITMAP_TRANSFORM_E + 256);
658.    }
659.
660.    void cargar_imagenes(void)
661.    {
662.
663.        unsigned long imagen;
664.        int i, j;
665.
666.        // DIGITOS// LOGO// Lectura de nueva fila (64 bits)
667.            pix_0_31_des = Lee_Reg(atrib_imagen.addr + dir);
668.            pix_32_63_des = Lee_Reg(atrib_imagen.addr + (dir + 4));
669.            // Reordeno
670.            for (orden = 3; orden >= 0; orden--)
671.            {
672.                pix_0_31 += (pix_0_31_des & mask_ult_byte) << (orden * 8);
673.                pix_0_31_des = pix_0_31_des >> 8;
674.                pix_32_63 += (pix_32_63_des & mask_ult_byte) << (orden * 8);
675.                pix_32_63_des = pix_32_63_des >> 8;
676.            }
677.            // Almaceno fila en matriz de comparacion
678.            for (col = 63; col >= 32; col--)
679.            {
680.                matriz_imagen_original[fila][col] = (pix_32_63 & mask_ult_bi
    t);
681.                pix_32_63 = pix_32_63 >> 1;
682.            }
683.            for (col = 31; col >= 0; col--)
```

```
684.            {
685.                matriz_imagen_original[fila][col] = (pix_0_31 & mask_ult_bit
    );
686.                pix_0_31 = pix_0_31 >> 1;
687.            }
688.        }
689.    }
690.
691.    void decodificaFormatoL1()
692.    {
693.        int i, j;
694.        for (i = 0; i < 64; i++)
695.        {
696.            for (j = 0; j < 64; j++)
697.            {
698.                vector_codificado[(j + i * 64) / 8] = matriz_imagen_original
    [i][j] + vector_codificado[(j + i * 64) / 8];
699.                if ((j + i * 64) % 8 != 7)
700.                    vector_codificado[(j + i * 64) / 8] = vector_codificado[
    (j + i * 64) / 8] << 1;
701.            }
702.        }
703.    }
704.
705.    void codificaMatrizFormatoRGB332()
706.    {
707.        int i, j;
708.        for (i = 0; i < 64; i++)
709.        {
710.            for (j = 0; j < 64; j++)
711.            {
712.                vector_codificado[j + i * 64] = (matriz_imagen_din_RGB[i][j]
    [0] << 5) + (matriz_imagen_din_RGB[i][j][1] << 2) + (matriz_imagen_din_RGB[
    i][j][2]);
713.            }
714.        }
715.    }
716.
717.    void traduceMatriz2RGB()
718.    {
719.        int i,j;
720.        for (i = 0; i < 64; i++)
721.        {
722.            for (j = 0; j < 64; j++)
723.            {
724.                if (matriz_imagen_original[i][j] == 1)
725.                {
726.                    matriz_imagen_din_RGB[i][j][0] = 7;
727.                    matriz_imagen_din_RGB[i][j][1] = 7;
728.                    matriz_imagen_din_RGB[i][j][2] = 3;
729.                }
730.                else
731.                {
732.                    matriz_imagen_din_RGB[i][j][0] = 0;
733.                    matriz_imagen_din_RGB[i][j][1] = 0;
734.                    matriz_imagen_din_RGB[i][j][2] = 0;
735.                }
736.            }
737.        }
738.    }
739.
740.    void guardaImagenDin()
741.    {
742.        unsigned long buffer;
743.        int i;
```

```
744.        for (i = 0; i < 4096; i += 4)
745.        {
746.            buffer = (vector_codificado[i + 3] << 24) + (vector_codificado[i
    + 2] << 16) + (vector_codificado[i + 1] << 8) + vector_codificado[i];
747.            Esc_Reg(RAM_G + 10 * 512 + 10000 + i, buffer);
748.        }
749.    }
750.
751.    void reseteaMatrizTrazoPintado()
752.    {
753.        int i, j;
754.        for (i = 0; i < 64; i++)
755.        {
756.            for (j = 0; j < 64; j++)
757.            {
758.                matrizDibujoRealizado[i][j] = 2*matriz_imagen_original[i][j]
    ;
759.            }
760.        }
761.    }
762.
763.    void resetImagenDin(){
764.        traduceMatriz2RGB();
765.        codificaMatrizFormatoRGB332();
766.        guardaImagenDin();
767.        reseteaMatrizTrazoPintado();
768.    }
769.
770.    float calculaScore(){
771.        float scoreRes=0;
772.        int cont_rojo = 0;
773.        int cont_verde = 0;
774.        int cont_blanco = 0;
775.        int i,j;
776.        float relacionPintadoDigito;
777.        int umbralValido;
778.
779.        for (i = 0; i < 64; i++)
780.        {
781.            for (j = 0; j < 64; j++)
782.            {
783.                if(matrizDibujoRealizado[i][j] == 1)
784.                    cont_verde ++;
785.                if(matrizDibujoRealizado[i][j] == -1)
786.                    cont_rojo ++;
787.                if(matrizDibujoRealizado[i][j] == 2)
788.                    cont_blanco ++;
789.            }
790.        }
791.
792.        // MENSAJES
793.        relacionPintadoDigito = (float)(cont_verde+cont_rojo)/(float)(cont_v
    erde+cont_blanco)*100;
794.        umbralValido = 22*(0.3*radioTrazo)+10;
795.        if(relacionPintadoDigito<umbralValido){
796.            sprintf(string, "TEST ERROR");
797.        }
798.
799.        else{
800.        if ((cont_verde+cont_rojo) != 0)
801.            scoreRes=((float)cont_verde/(float)(cont_verde+abs(cont_rojo)))*
    10;
802.
803.        if(scoreRes>=10){
804.            sprintf(string, "PERFECT !!");
```

```
805.        }
806.        else if(scoreRes>=9.5){
807.            sprintf(string, "AMAZING");
808.        }
809.        else if(scoreRes>=8.5){
810.            sprintf(string, "WELL DONE");
811.        }
812.        else if(scoreRes>=7){
813.            sprintf(string, "NICE");
814.        }
815.        else if(scoreRes>=5){
816.            sprintf(string, "MORE OR LESS");
817.        }
818.        else
819.            sprintf(string, "TRY AGAIN");
820.        }
821.
822.        return scoreRes;
823.    }
824.
825.    void ComSlider(int x, int y, int w, int h, int options, unsigned int *va
    l, int range){
826.        //x,y: posicion de esquina lateral izquierda del widget
827.        //w: anchura de la barra en pixeles
828.        //h: altura de la barra (grosor) en pixeles
829.        //options: efecto 3D o plano
830.        //*val: valor a rellenar en la barra (se pasa como puntero para
    pasar parametro por referencia y poder modificar su valor
831.        //range: rango que puede tomar el valor a representar
832.
833.    #ifdef VM800B35
834.    #ifdef ESCALADO
835.        x=(x*2)/3;
836.        y=(y*15)/17;
837.        w=(w*2)/3;
838.        h=(h*15)/17;
839.    #endif
840.    #endif
841.        int markerx_c = (w*(*val)/(float)range)+x; //posicion estimada del
    indicador del slider
842.        if( POSX > markerx_c-h && POSX < markerx_c+h  && POSY > y-
    h && POSY < y+h ){ //Comprobar si se esta pulsando el indicador
843.            if(POSX<x) //saturar por la izquierda
844.                POSX=x;
845.            else if(POSX>x+w) //saturar por la derecha
846.                POSX=x+w;
847.            *val = (int)((float)range/(float)w*(POSX-x)); //actualizar valor
    del valor asociado al slider
848.        }
849.        EscribeRam32(CMD_SLIDER);
850.        EscribeRam16(x);
851.        EscribeRam16(y);
852.        EscribeRam16(w);
853.        EscribeRam16(h);
854.        EscribeRam16(options);
855.        EscribeRam16(*val);
856.        EscribeRam16(range);
857.        EscribeRam16(0);
858.    }
859.
860.    void progress_list(){
861.        int dig;
862.        char string[30];
863.
864.        ComColor(255, 255, 255);
```

```
865.        ComTXT(160, 20, 30, OPT_CENTER, "BEST SCORES");
866.        ComColor(0, 0, 0);
867.        ComTXT(160, 50, 24, OPT_CENTER, "--------- Current Progress --------
    -");
868.
869.        for(dig=0;dig<10;dig++){
870.            sprintf(string, "DIGITO %d - - - - >
    %.2f", dig, ((float)best_scores[dig]/(float)100));
871.            if(dig%2) ComColor(0, 0, 0);
872.            else ComColor(255, 255, 255);
873.            ComTXT(120, 75+dig*16+1, 18, OPT_CENTER, string);
874.        }
875.
876.        ComTXT(160, 230, 24, OPT_CENTER, "----------------------------------
    ----------------------");
877.    }
878.
879.    void ComProgressBar(int x, int y, int w, int h, int options, int val, in
    t range){
880.        //x,y: posicion de esquina lateral izquierda del widget
881.        //w: anchura de la barra en p□xeles
882.        //h: altura de la barra (grosor) en p□xeles
883.        //options: efecto 3D o plano
884.        //val: valor a rellenar en la barra
885.        //range: rango que puede tomar el valor a representar
886.
887.    #ifdef VM800B35
888.    #ifdef ESCALADO
889.        x=(x*2)/3;
890.        y=(y*15)/17;
891.        w=(w*2)/3;
892.        h=(h*15)/17;
893.    #endif
894.    #endif
895.        EscribeRam32(CMD_PROGRESS);
896.        EscribeRam16(x);
897.        EscribeRam16(y);
898.        EscribeRam16(w);
899.        EscribeRam16(h);
900.        EscribeRam16(options);
901.        EscribeRam16(val);
902.        EscribeRam16(range);
903.        EscribeRam16(0);
904.    }
905.
906.    int teclado(void)
907.    {
908.        int teclaPulsada=NO_PULSADO;
909.        ComKeys(10,38,300,35,28,0,"0123456789");
910.        ComKeys(10,77,300,35,28,0,"QWERTYUIOP");
911.        ComKeys(10,116,300,35,28,0,"ASDFGHIJKL");
912.        ComKeys(10, 155, 300, 35, 28, 0, "ZXCVBNM");
913.        teclaPulsada=Keys(10,38,300,35,28,"0123456789");
914.        if(teclaPulsada==NO_PULSADO)
915.            teclaPulsada=Keys(10,77,300,35,28,"QWERTYUIOP");
916.        if(teclaPulsada==NO_PULSADO)
917.            teclaPulsada=Keys(10,116,300,35,28,"ASDFGHIJKL");
918.        if(teclaPulsada==NO_PULSADO)
919.            teclaPulsada=Keys(10, 155, 300, 35, 28, "ZXCVBNM");
920.        if(Boton(100, 194, 157, 40, 28, ""))
921.            teclaPulsada=' ';
922.        if(Boton(260, 194, 50, 40, 28, "DEL"))
923.            teclaPulsada=DEL;
924.        if(Boton(10,194,87,40,28,"ENTER"))
925.            teclaPulsada=ENTER;
```

```
926.        return teclaPulsada;
927.    }
928.
929.    void ComKeys(int16_t x, int16_t y, int16_t w, int16_t h, int16_t font, u
    int16_t options, const char *s){
930.
931.    #ifdef VM800B35
932.    #ifdef ESCALADO
933.        x=(x*2)/3;
934.        y=(y*15)/17;
935.        w=(w*2)/3;
936.        h=(h*15)/17;
937.    #endif
938.    #endif
939.
940.        //Detectar tecla pulsada
941.        EscribeRam32(CMD_KEYS);
942.        EscribeRam16(x);
943.        EscribeRam16(y);
944.        EscribeRam16(w);
945.        EscribeRam16(h);
946.        EscribeRam16(font);
947.        EscribeRam16(options);
948.        while(*s) EscribeRam8(*s++);
949.        EscribeRam8(0);
950.        PadFIFO();
951.
952.    }
953.
954.    int Keys(int16_t x, int16_t y, int16_t w, int16_t h, int16_t font, const
     char *s)
955.    {
956.    #ifdef VM800B35
957.    #ifdef ESCALADO
958.        x=(x*2)/3;
959.        y=(y*15)/17;
960.        w=(w*2)/3;
961.        h=(h*15)/17;
962.    #endif
963.    #endif
964.        int i, longitud=0, resul = 0;
965.        char cadena[50];
966.        while(*(s+longitud))
967.        {
968.            cadena[longitud]=*(s+longitud);
969.            longitud++;
970.        }
971.        cadena[longitud]='\0';
972.
973.
974.        Lee_pantalla();
975.        for(i=0;i<longitud;i++)
976.        {
977.            if(POSX>x+i*(w/longitud) && POSX<x+i*(w/longitud)+w/longitud &&
    POSY>y && POSY<(y+h)){
978.                ComKeys(x,y,w,h,font,cadena[i],cadena);
979.                resul=*(s+i);
980.                break;
981.            }
982.        }
983.        return resul;
984.    }
```

PheripheralConfigurationLibrary.h

```
1. /*Libreria para la configuración de periféricos del microcontrolador
   TM4C1294NCPDT. Creada por Sergio Leon Doncel (03 Noviembre 2022)*/
2. //Consideraciones: se hace uso de funciones variádicas sin comprobar ni el
   número de argumentos recibidos ni el tamaño de los argumentos obtenidos, lo
   que puede
3. //conllevar al bloqueo del micro si se hace uso incorrecto de dichas
   funciones, tener especial cuidado
4.
5. #ifndef _PHERIPHERALCONFIGURATIONLIBRARY_H_
6. //------------------------------------------------------------------------
   ---------------------------------
7. #define _PHERIPHERALCONFIGRATIONLIBRARY_H_
8.
9. #include <stdint.h>
10. #include <stdbool.h>
11. #include <stdarg.h> //permite el uso de funciones variadicas (numero de
   parametros a una funcion variable)
12.
13. #include "driverlib2.h"
14. #include "HAL_I2C.h"
15. #include "sensorlib2.h"
16. #include "FT800_TIVA.h"
17.
18. /*-------------Definicion de variables--------------*/
19. //-----------Necesario para velocidad del procesador y pantalla
20. uint32_t RELOJ;
21.
22. //------- Necesario para uso de pantalla
23. extern char chipid;                         //Almacena el ID leido de la
   pantalla FT800
24. extern unsigned long cmdBufferRd;          // Almacena el valor leido por el
   registro de lectura
25. extern unsigned long cmdBufferWr;          // Almacena el valor leido por el
   registro de escritura
26. extern unsigned long POSX, POSY, BufferXY; //Variables para localizacion de
   la posicion del pulso en la pantalla
27. extern unsigned int CMD_Offset;
28.
29. //-------- Necesario para la calibracion de pantalla
30. //#define AGL_pantalla
31. #define SLD_pantalla
32. #define VM800B35
33.
34. extern const int32_t REG_CAL[6]; //Pantalla 3.5 pulgadas
35. extern const int32_t REG_CAL5[6]; //Pantalla 5 pulgadas
36.
37. //------Necesario para uso de SensorsBoosterPack
38. // BME280
39. extern int g_s32ActualTemp;
40. extern unsigned int g_u32ActualPress;
41. extern unsigned int g_u32ActualHumity;
42.
43. // BMI160/BMM150
44. extern uint8_t returnValue;
45. extern struct bmi160_gyro_t        s_gyroXYZ;
46. extern struct bmi160_accel_t       s_accelXYZ;
47. extern struct bmi160_mag_xyz_s32_t s_magcompXYZ;
48.
49. //Calibration off-sets
50. extern int8_t accel_off_x;
51. extern int8_t accel_off_y;
52. extern int8_t accel_off_z;
53. extern int16_t gyro_off_x;
54. extern int16_t gyro_off_y;
```

```
55. extern int16_t gyro_off_z;
56.
57. //---------------Necesario para uso del servomotor
58. extern unsigned int Max_pos;
59. extern unsigned int Min_pos;
60. extern unsigned int posicion;
61.
62. /*-----------Prototipo de funciones----------------------*/
63. uint32_t ConfiguraReloj120MHz(void);
64. //Si se desea otra frecuencia de trabajo para el micro, se puede modificar
    manualmente el comportamiento de esta funcion
65. void HabilitaPerifericos(unsigned int parameters_number, ...);
66. //1) Parametro de SYSCTL_PERIPH_?
67. void defineTipoPines(unsigned int parameters_number, ...);
68. //Pasar numero de argumentos y en bloques de 3 parametros lo siguiente:
69. //1) Tipo: 1-Salida | 0-Entrada
70. //2) Puerto: GPIO_PORT?_BASE
71. //3) Pines de dicho puerto: GPIO_PIN_?
72. void ConfiguraTimer(int numTimer, unsigned int periodo, void (*ptr)(void));
73. //1) Numero de Timer
74. //2) Periodo del timer en ms
75. //3) Puntero a rutina de interrupcion del timer
76. void ModoBajoConsumo(unsigned int parameters_number, ...);
77. //Pasar numero de argumentos y perifericos que mantener activos que puedan
    despertar el micro con sus interrupciones
78. void ConfiguraPantalla(unsigned char boosterpack_number, uint32_t RELOJ);
79. //Pasar numero del boosterpack en el que esta conectado y frecuencia de
    Reloj del micro
80. void ConfiguraSensorsBoosterpack(unsigned char boosterpack_number, uint32_t
     RELOJ);
81. //Pasar numero del boosterpack en el que esta conectado y frecuencia de
    Reloj del micro
82. void ConfiguraPWM(uint32_t base,uint32_t PuertoPin, uint32_t puerto, uint8_
    t pin, unsigned int frecuenciaPWM);
83. //1) base -> Salida PWM a usar: PWM?_BASE
84. //2) PuertoPin -> Conexion pin con señal PWM: GPIO_PG?_M0PWM? (mirar en
    tabla de diapositiva 47 tema 4)
85. //3) puerto: GPIO_PORT?_BASE
86. //4) pin: GPIO_PIN_?
87. //5) frecuenciaPWM
88. void ConfiguraEEPROM();
89. void ConfiguraCCM_SHA256();
90.
91. //-------------------------------------------------------------------------
    ---------------------------------
92. #endif
```

PheripheralConfigurationLibrary.c

```
1.  #include "PheripheralConfigurationLibrary.h"
2.
3.
4.  /*-------------Definicion de variables--------------*/
5.  //----------- Necesario para uso de pantalla
6.  char chipid = 0;                          //Almacena el ID leido de la
    pantalla FT800
7.  unsigned long cmdBufferRd = 0x00000000;          // Almacena el valor leido
    por el registro de lectura
8.  unsigned long cmdBufferWr = 0x00000000;          // Almacena el valor leido
    por el registro de escritura
9.  unsigned long POSX, POSY, BufferXY; //Variables para localizaciÃ³n de la
    posicion del pulso en la pantalla
10. unsigned int CMD_Offset = 0;
11.
12. //----------- Necesario para la calibracion de pantalla
13.
```

```
#ifdef AGL_pantalla
//Pantalla Alvaro
const int32_t REG_CAL[6]={21696,-78,-614558,498,-17021,15755638}; //Pantalla 3.5 pulgadas
const int32_t REG_CAL5[6]={32146, -1428, -331110, -40, -18930, 18321010}; //Pantalla 5 pulgadas
#endif

#ifdef SLD_pantalla
//Pantalla Sergio
const int32_t REG_CAL[6]={24816,397,-1325076,99,-16775,16138575}; //Pantalla 3.5 pulgadas
const int32_t REG_CAL5[6]={32146, -1428, -331110, -40, -18930, 18321010}; //Pantalla 5 pulgadas
#endif

//---------Necesario para SensorsBoostPack
// BME280
int g_s32ActualTemp   = 0;
unsigned int g_u32ActualPress  = 0;
unsigned int g_u32ActualHumity = 0;

// BMI160/BMM150
uint8_t returnValue;
struct bmi160_gyro_t        s_gyroXYZ;
struct bmi160_accel_t       s_accelXYZ;
struct bmi160_mag_xyz_s32_t s_magcompXYZ;

//Calibration off-sets
int8_t accel_off_x;
int8_t accel_off_y;
int8_t accel_off_z;
int16_t gyro_off_x;
int16_t gyro_off_y;
int16_t gyro_off_z;

//---------Necesario para Servomotor
unsigned int Max_pos = 4200; //3750 (2 ms de 20 ms == 10% del periodo --> 0.1 * 37500 = 3750)
unsigned int Min_pos = 1300; //1875 (1 ms de 20 ms == 5% del periodo --> 0.05 * 37500 = 1875)
unsigned int posicion;


/*------------Declaracion comportamiento funciones-------------*/

uint32_t ConfiguraReloj120MHz(void)
{
    uint32_t RELOJ = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN | SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);
    return RELOJ;
}

void HabilitaPerifericos(unsigned int parameters_number, ...)
{
    va_list ap;
    int i;
    va_start(ap,parameters_number);
    for(i=0;i<parameters_number;i++)
    {
        SysCtlPeripheralEnable(va_arg(ap, uint32_t));
    }
    va_end(ap);
}

void defineTipoPines(unsigned int parameters_number, ...)
{
```

```
74.     va_list ap;
75.     int i;
76.     va_start(ap,parameters_number);
77.     for(i=0;i<parameters_number/3;i++)
78.     {
79.         bool tipo = va_arg(ap, uint32_t);
80.         uint32_t puerto = va_arg(ap, uint32_t);
81.         uint8_t pines = va_arg(ap, uint32_t);
82.         if(tipo)
83.         {
84.             GPIOPinTypeGPIOOutput(puerto, pines);
85.         }
86.         else GPIOPinTypeGPIOInput(puerto, pines);
87.     }
88.     va_end(ap);
89. }
90.
91. void ConfiguraTimer(int numTimer, unsigned int periodo, void (*ptr)(void))
92. {
93.     uint32_t base;
94.     uint32_t interruptTimer;
95.     switch (numTimer) {
96.         case 0:
97.             base = TIMER0_BASE;
98.             interruptTimer = INT_TIMER0A;
99.             break;
100.            case 1:
101.                base = TIMER1_BASE;
102.                interruptTimer = INT_TIMER1A;
103.                break;
104.            case 2:
105.                base = TIMER2_BASE;
106.                interruptTimer = INT_TIMER2A;
107.                break;
108.            case 3:
109.                base = TIMER3_BASE;
110.                interruptTimer = INT_TIMER3A;
111.                break;
112.            case 4:
113.                base = TIMER4_BASE;
114.                interruptTimer = INT_TIMER4A;
115.                break;
116.            case 5:
117.                base = TIMER5_BASE;
118.                interruptTimer = INT_TIMER5A;
119.                break;
120.        }
121.
122.        TimerClockSourceSet(base, TIMER_CLOCK_PIOSC);   //Fijar reloj que
    usara T1: 16MHz
123.        TimerConfigure(base, TIMER_CFG_SPLIT_PAIR| TIMER_CFG_A_PERIODIC | TI
    MER_CFG_B_PERIODIC);    //T1 periodico y conjunto (32b)
124.        TimerPrescaleSet(base, TIMER_A, 255);    //T1 preescalado a 256:
    16M/256=62.5 KHz
125.        uint32_t PeriodoTimer = 62.5*periodo;
126.        TimerLoadSet(base, TIMER_A, PeriodoTimer -1); //Fijar periodo
127.        TimerIntRegister(base,TIMER_A,ptr); //Definicion de interrupcion
    asociada al Timer
128.        IntEnable(interruptTimer); //Habilitar interrupcion global del Timer
129.        TimerIntEnable(base, TIMER_TIMA_TIMEOUT);
130.        TimerEnable(base, TIMER_A);  //Habilitar Timer1
131.    }
132.
133.    void ModoBajoConsumo(unsigned int parameters_number, ...)
134.    {
135.        va_list ap;
136.        int i;
```

```
137.        va_start(ap,parameters_number);
138.        for(i=0;i<parameters_number;i++)
139.        {
140.            SysCtlPeripheralSleepEnable(va_arg(ap, uint32_t));
141.        }
142.        va_end(ap);
143.        SysCtlPeripheralClockGating(true);
144.    }
145.
146.    void ConfiguraPantalla(unsigned char boosterpack_number, uint32_t RELOJ)
147.    {
148.        HAL_Init_SPI(boosterpack_number, RELOJ);  //Boosterpack a usar,
    Velocidad del MC
149.            Inicia_pantalla();       //Arranque de la pantalla
150.            int i;
151.
152.            //Pantalla 3.5 pulgadas
153.        #ifdef VM800B35
154.            for(i=0;i<6;i++)    Esc_Reg(REG_TOUCH_TRANSFORM_A+4*i, REG_CAL[i
    ]);
155.        #endif
156.            //Pantalla 5 pulgadas
157.        #ifdef VM800B50
158.            for(i=0;i<6;i++)    Esc_Reg(REG_TOUCH_TRANSFORM_A+4*i, REG_CAL5[
    i]);
159.        #endif
160.    }
161.
162.    void ConfiguraEEPROM()
163.    {
164.        EEPROMInit(); //Se recupera si el ultimo apagado fue durante
    escritura
165.    }
166.
167.    void ConfiguraSensorsBoosterpack(unsigned char boosterpack_number, uint3
    2_t RELOJ)
168.    {
169.    Conf_Boosterpack(boosterpack_number, RELOJ); //Indicar que el sensors
    boosterpack esta conectado a los pines del boosterpack 2
170.    //Configurar sensor de temperatura, presion y humedad relativa
171.    bme280_data_readout_template();
172.    bme280_set_power_mode(BME280_NORMAL_MODE);
173.    //Configurar acelerometro y giroscopio
174.    bmi160_initialize_sensor();
175.    bmi160_config_running_mode(APPLICATION_NAVIGATION);
176.    //Configurar sensor de luz
177.    OPT3001_init();
178.    }
179.
180.    void ConfiguraCCM_SHA256()
181.    {
182.
183.        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_CCM0))
184.        {
185.        }
186.        SHAMD5Reset(SHAMD5_BASE);
187.        SHAMD5ConfigSet(SHAMD5_BASE, SHAMD5_ALGO_SHA256);
188.    }
189.
190.
191.    void ConfiguraPWM(uint32_t base,uint32_t PuertoPin, uint32_t puerto, uin
    t8_t pin, unsigned int frecuenciaPWM)
192.    {
193.        PWMGenConfigure(base, PWM_GEN_2, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO
    _SYNC); //Configurar el pwm0, generador 2, contador descendente y sin
    sincronizacion (actualizacion automatica)
```

```
194.        PWMClockSet(base,PWM_SYSCLK_DIV_64);    // Configuracion del reloj:
    1.875MHz
195.        GPIOPinConfigure(PuertoPin);            // Configurar el pin a PWM
196.        GPIOPinTypePWM(puerto, pin); // Configurar el pin a PWM
197.
198.        unsigned int PeriodoPWM=1875000/frecuenciaPWM;
199.        PWMGenPeriodSet(base, PWM_GEN_2, PeriodoPWM-1); //Definicion del
    periodo de la PWM
200.        PWMOutputState(base, PWM_OUT_4_BIT , true);     //Habilitar la salida
    PWM
201.        PWMGenEnable(base, PWM_GEN_2);      //Habilita el generador 2
202.        posicion=(Max_pos+Min_pos)/2;
203.        PWMPulseWidthSet(base, PWM_OUT_4, posicion);    //Inicialmente, 1ms -
    -> Posicion media
204.    }
```

*Se han añadido macros necesarias en la librería ft800_TIVA dentro del archivo de cabecera, por tanto se adjunta también dicho código en EV.