Manual de Usuario MarchOns

Version 1.0.0

Julián Sánchez, Maria Del Mar Arbelaez, Manuel Velasquez

December 12, 2024

Contents

1 Introducción.

Este documento contiene la información técnica sobre el proyecto realizado para la asignatura de Electrónica Digital 3, sus funcionalidades y si implementación.

El MarcOns es un reloj que muestra el conteo de pasos, y despliega en su pantalla una serie de bioestadísticas como el pulso cardíaco, el numero de pasos, las calorías quemadas y la distancia recorrida. Ademas de mostrar datos como la hora, la fecha y la batería del reloj. El sistema fue diseñado e implementado basados en el MCU RP2040 y en la tarjeta de desarrollo suministrada por la empresa waveshare, la cual integra un conector USB tipo C, un cargador de alta eficiencia de baterías de litio y polímeros ETA6096, una unidad de medición inercial QMI8658A y un controlador GC9A01A para la pantalla TFT de 1,28 pulgadas, ademas se integra una batería de litio y polímero de 3.7v de 400mAh, un sensor de pulso MAX30102 y un RTC externo DS1302, sobre estos sensores y los periféricos extra se detalla su configuración y funcionamiento en este archivo, así como la estructura del software y algunas funciones importantes para el entendimiento del funcionamiento del reloj.

2 Especificaciones Técnicas.

Como microcontrolador para la implementación del reloj se utiliza un RP2040, diseñado por la compañía Raspberry Pi, utiliza un procesador de doble núcleo Arm Cortex M0+ @ $135\mathrm{MHz}$. contiene una SRAM de 264KB y 2MB de memoria FLASH, el microcontrolador va sobre una plataforma de desarrolla fabricada por la empresa Waveshare la cual integra el MCU con un conector USB tipo C, el cual funciona para la programación del microcontrolador como para la carga de la batería de litio, carga que es administrada por el circuito integrado ETA6096, este CI es capaz de suministrar $2.5\mathrm{A}$ de corriente de carga a la batería. Esta plataforma también integra una unidad de medición inercial (IMU) QMI8658A, la cual utiliza tecnología MEMS 6D, proporciona una interfaz Host-procesador compatible con I2C, ideal para aplicaciones que requieran funcionalidad basada en movimiento y detección de eventos de movimiento, como la detección de pasos. y finalmente en esta plataforma de desarrollo se encuentra con el GC9A01, un controlador SoC de 262.144 colores para pantallas de cristal liquido TFT con resolución de

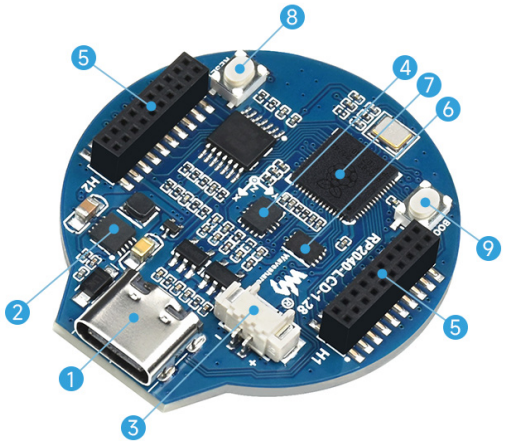

Figure 1: Componentes enumerados de la tarjeta de desarrollo.

240RGBx240.

1. USB tipo C
2. ETA6096
3. Conector de la bateria
4. QMI8658A
5. Pines de los gpios del RP2040
6. W25Q16JVUXIQ. 2MB NOR-Flash
7. RP2040
8. Botón de RESET
9. Botón de BOOT

Además de esta tarjeta de desarrollo, se tiene la implementación de un sensor de ritmo cardíaco, el MAX30102 es un sensor de método no invasivo que permite medir el porcentaje de saturación de oxigeno en la hemoglobina (SaO2) en sangre utilizando un circuito fotoeléctrico. El MAX30102 es un dispositivo que integra un pulsioxímetro y un monitor de frecuencia cardíaca. Posee dos LED: un led rojo (660nm) y un led infrarrojo (920nm), un foto detector, óptica especializada, filtro de luz ambiental entre 50 y 60Hz, y un conversor ADC delta sigma de 16 bits y de hasta 1000 muestras por segundo, proporciona una interfaz Host-procesador compatible con I2C.

Figure 2: MAX30102

El RTC, DS1302 es un circuito integrado alimentado por una bateria externa el cual, en todo momento, registra la fecha, hora y dia de la semana al igual que un reloj convencional. Estos datos son proporcionados mediante una interfaz SPI.

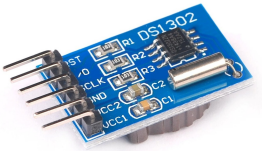

Figure 3: DS1302

La integración de estos sensores con la tarjeta de desarrollo se puede observar mediante el siguiente diagrama de bloques:

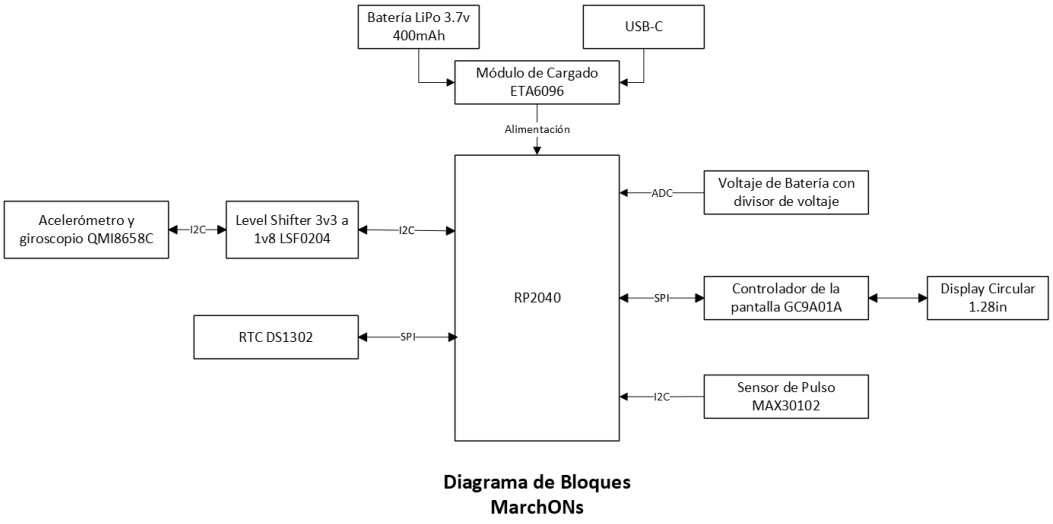

Figure 4: Diagrama de bloques del MarchOns

3 Estructura de Software.

La estructura del software para la implementación de las funcionalidades del reloj, se implementa una jerarquía de librerías que permite el orden y el entendimiento del código de manera clara y portable a otras plataformas de desarrollo.

```
Firmware/
|-- include/
|   +-- hardware/
|   |   +-- imu.h
|   |   +-- LCD.h
|   |   +-- ds1302.h
|   |   +-- max30102.h
|   |
|   +-- drivers/
|   |   +-- i2c_driver.h
|   |   +-- spi_driver.h
|   |
|   +-- utils/
|   |   +-- register_utils.h
|   |
|   +-- lib.h
|   +-- pulse_read.h
|   |
|   |
|-- lvgl/
|-- src/
|   +-- hardware/
|   |   +-- imu.c
|   |   +-- LCD.c
|   |   +-- ds1302.c
|   |   +-- max30102.c
|   |
|   +-- drivers/
|   |   +-- i2c_driver.c
|   |   +-- spi_driver.c
|   |
|   +-- lib.c
|   +-- pulse_read.c
|   +-- Firmware.c
|   |
+-- lv_config.h
+-- pico_sdk_import.cmake
+-- CMakeLists.txt
```

Esta jerarquía de clasificación por carpetas y archivos permite versatilidad al código, separando los controladores de los periféricos del rp2040, las librerías de hardware para cada uno de los sensores dispuestos en la implementación y la unión de estos en archivos generales donde se utilizan funciones propias de la libreria sin afectar la inicialización ni la configuración de los sensores.

A continuación descripciones mas detallas del contenido de los archivos de la estructura de código

3.1 Funciones de i2c_driver.h

El archivo `i2c_driver.h` contiene funciones para manejar la comunicación I2C, desde la configuración hasta la lectura y escritura de datos.

3.1.1 Funciones para la configuración del I2C

- `smartwatch_i2c_init`: Inicializa el puerto I2C, los pines SDA y SCL, los pines de interrupción de la IMU y el callback de interrupción.

3.1.2 Funciones de escritura

- **`i2c_write_byte`:** Escribe un solo byte en un registro de un dispositivo I2C.
- **`i2c_write_nbytes`**: Escribe varios bytes en un registro de un dispositivo I2C.

3.1.3 Funciones de lectura

- **`i2c_read_byte`:** Lee un byte de un registro de un dispositivo I2C.
- **`i2c_read_nbytes`:** Lee varios bytes de un registro de un dispositivo I2C.

3.2 Funciones de spi_driver.h

El archivo `spi_driver.h` contiene funciones para manejar la comunicación SPI, incluyendo la configuración, escritura de bytes, y el uso de DMA y PWM.

3.2.1 Funciones para la configuración del SPI

- **`SPI_init`:** Inicializa el driver SPI, configurando los registros necesarios.
- **`SPI0_init`:** Inicializa el driver SPI0.

- **`config_dma`:** Configura el DMA para el driver SPI.
- **`config_pwm`:** Configura el PWM para controlar el brillo de la pantalla.
- **`config_gpio`:** Configura los pines GPIO para el driver SPI.
- **`config_RTC_pins`:** Configura los pines GPIO para el driver SPI0.

3.2.2 Funciones de escritura

- **`SPI_WriteByte`:** Escribe un byte en el puerto SPI.
- **`SPI_Write_nByte`:** Escribe una cadena de bytes en el puerto SPI.
- **`SPI0_WriteByte`:** Escribe un byte en el puerto SPI0.
- **`SPI0_beginTransmission`:** Inicia la transmisión escribiendo el valor requerido del CS y el comando de comunicación.
- **`SPI0_endTransmission`:** Termina la transmisión escribiendo el valor requerido del CS.

3.2.3 Funciones de lectura

- **`SPI0_ReadByte`:** Lee un byte del puerto SPI0.

3.2.4 Funciones de control del brillo

- **`set_pwm`:** Configura el nivel de PWM para controlar el brillo de la pantalla.

3.3 Funciones de ds1302.h

El archivo `ds1302.h` contiene funciones para manejar el módulo DS1302, que incluye la configuración, lectura y escritura de registros, así como la gestión de la memoria RAM interna del módulo.

3.3.1 Configuración del RTC

- **`DS1302_init`:** Inicializa el módulo RTC con un valor de respaldo en caso de que la hora esté incorrecta. También maneja la configuración basada en la conexión USB.

- **`SetIsRunning`:** Inicia o detiene el RTC según el valor booleano.
- **`GetIsRunning`:** Verifica si el RTC está corriendo.
- **`SetIsWriteProtected`:** Permite o bloquea la escritura en el RTC.
- **`GetIsWriteProtected`:** Verifica si el RTC está protegido contra escritura.

3.3.2 Lectura y escritura de registros

- **`getReg`:** Lee un registro del módulo RTC.
- **`setReg`:** Escribe un valor en un registro del módulo RTC.
- **`SetDateTime`:** Escribe la fecha y hora en el RTC.
- **`GetDateTime`:** Lee la fecha y hora del RTC.

3.3.3 Memoria RAM del RTC

- **`SetMemory`:** Escribe un valor en la memoria RAM del RTC.
- **`GetMemory`:** Lee un valor de la memoria RAM del RTC.

3.3.4 Validación de fecha y hora

- **`IsDateTimeValid`:** Verifica si la fecha y hora del RTC son válidas.
- **`DateIsValid`:** Valida una estructura `datetime_t`.

3.3.5 Impresión y cálculo de fecha

- **`print_datetime`:** Imprime una estructura `datetime_t` en la consola serial.
- **`dayofweek`:** Calcula el día de la semana a partir de una fecha válida.

3.4 Funciones del Sensor IMU

read_imu_step_count: Función para leer los pasos detectados por el podómetro del sensor IMU. La función accede a los registros `STEP_CNT_LOW`, `STEP_CNT_MIDL`,

STEP_CNT_HIGH y devuelve la cantidad total de pasos detectados. *Retorno:* La cantidad de pasos detectados (uint32_t).

enable_pedometer: Esta función habilita el podómetro del sensor IMU, ajustando la tasa de datos de salida (ODR) a 50Hz, activando el acelerómetro y finalmente el podómetro. *Retorno:* Ninguno.

imu_config_pedometer_params: Configura los parámetros del podómetro, como la cantidad de muestras por lote, umbrales para la detección de picos y duración de los pasos. Configura todos los parámetros relacionados con el funcionamiento del podómetro. *Retorno:* 0 si la configuración fue exitosa, -1 en caso contrario.

config_interrupts: Habilita las interrupciones del sensor IMU configurando el modo push-pull y mapeando la interrupción al pin INT1, lo que permite la detección de eventos específicos. *Retorno:* Ninguno.

reset_imu: Resetea el sensor IMU realizando un soft reset y luego verifica que el sensor haya sido reseteado correctamente. *Retorno:* 0 si el sensor fue reseteado correctamente, -1 en caso contrario.

QMI8658_init: Inicializa el sensor IMU, reseteando el sensor, verificando la conexión y configurando el podómetro para su uso. *Retorno:* Ninguno.

3.5 Funciones del Driver LCD

LCD_init: Inicializa el driver LCD, configurando los registros necesarios para su funcionamiento correcto. *Parámetros:*

- scan_dir: Dirección de escaneo del display.

Retorno: Ninguno.

LCD_1IN28_SetWindows: Configura las dimensiones del display, estableciendo el área en la que se quiere dibujar en la pantalla. *Parámetros:*

- Xstart: Coordenada X de inicio.
- Ystart: Coordenada Y de inicio.
- Xend: Coordenada X de fin.
- Yend: Coordenada Y de fin.

Retorno: Ninguno.

LCD_Clear: Limpia la pantalla del display pintando toda la pantalla con un color específico. *Parámetros:*

- `color`: Color con el que se va a pintar la pantalla.

Retorno: Ninguno.

LCD_Display: Muestra una imagen en el display. La imagen debe ser un arreglo de 16 bits. *Parámetros:*

- `image`: Arreglo de 16 bits que contiene la imagen.

Retorno: Ninguno.

LCD_1IN28_DisplayWindows: Muestra una imagen en el display en un área especificada por las coordenadas dadas. La imagen debe ser un arreglo de 16 bits. *Parámetros:*

- `Xstart`: Coordenada X de inicio.
- `Ystart`: Coordenada Y de inicio.
- `Xend`: Coordenada X de fin.
- `Yend`: Coordenada Y de fin.
- `Image`: Arreglo de 16 bits que contiene la imagen.

Retorno: Ninguno.

3.6 Funciones del Driver MAX30102

pulse_getIR_flag: Obtiene el estado del indicador de bandera del sensor IR. *Retorno:* Valor booleano que indica el estado de la bandera.

pulse_setIR_flag: Establece el estado de la bandera del sensor IR. *Parámetros:*

- `set`: Estado de la bandera (valor booleano).

Retorno: Ninguno.

max_init: Inicializa el módulo MAX30102, configurando los modos de los LEDs, SPO2, los SLOTS, la corriente de los LEDs y la FIFO. *Retorno:* Ninguno.

pulse_getIR: Obtiene la última medida leída de la FIFO (canal IR). *Retorno:* Medida IR de 32 bits.

3.7 Archivo de configuración del smartwatch

Este archivo define las constantes, estructuras y funciones necesarias para configurar el sistema del smartwatch. A continuación se detallan las principales definiciones y funciones.

3.7.1 Definiciones clave

- **PLL_SYS_KHZ**: Frecuencia del PLL del sistema, definida como 270 MHz (270×10^3 Hz).
- **INT1_PIN**: Pin de interrupción 1, asignado al número 23.
- **WOM_INT**: Máscara para la interrupción por detección de movimiento (WoM).
- **STEP_INT**: Máscara para la interrupción por paso detectado.
- **DISP_HOR_RES**: Resolución horizontal del display, definida como 240 píxeles.
- **DISP_VER_RES**: Resolución vertical del display, definida como 240 píxeles.

3.7.2 Estructuras

flags_t: Estructura que contiene las banderas para el sistema:

- **step_flag**: Bandera para indicar la detección de un paso (1 bit).
- **wom_flag**: Bandera para indicar la detección de movimiento (WoM) (1 bit).

update_main_screen: Función que actualiza la pantalla principal del smartwatch. *Descripción:* Realiza el chequeo de las variables y banderas de software para actualizar el contenido de la pantalla principal. *Retorno:* Ninguno.

smartwatch_main: Función principal de configuración del sistema. *Descripción:* Configura los clocks del sistema, los pines de interrupción e inicializa las librerías de hardware. *Retorno:* Entero que indica el estado de la inicialización del sistema.

Dentro de este archivo se encuentran funciones de tipo static que son usadas para la creación y administración de la interfaz gráfica, mediando LVGL.

3.8 Funciones de pulse_read.h

El archivo `pulse_read.h` contiene funciones y estructuras relacionadas con la medición del pulso utilizando el sensor MAX30102. Las funciones principales incluyen el almacenamiento y suavizado de muestras de IR, así como el cálculo del pulso por minuto.

3.8.1 Estructuras

- **beat_detector**: Estructura que almacena los parámetros y datos necesarios para detectar el pulso, incluyendo:
  - **`sample_rate`:** Tasa de muestreo (ej. 100 Hz).
  - **`window_size`:** Tamaño de la ventana de datos para análisis.
  - **`smoothing_window`:** Tamaño de la ventana de suavizado.
  - **`sample`:** Array que almacena las muestras de IR.
  - **`timestamps`:** Array que almacena las marcas de tiempo de las muestras.
  - **`filtered_samples`:** Array para las muestras filtradas.
  - **`round`:** Número de rondas del análisis.
  - **`peak_len`:** Longitud del pico detectado.

3.8.2 Funciones

- **add_sample**: Guarda una muestra de IR y aplica un proceso de suavizado.
- **calculate_heart_rate**: Calcula el pulso por minuto (BPM) analizando las muestras de IR, detectando los picos y su distancia.

4 Configuraciones.

Para los diferentes periféricos que se utilizan en el proyecto, tanto propios del rp2040 como externos, todos precisan de configuraciones detalladas la cuales se proceden a señalar en este apartado del manual.

4.1 Parámetros de Configuración del Podómetro

El archivo contiene los parámetros para la configuración del podómetro del sensor IMU QMI8658A. Estos parámetros controlan diversos aspectos del cálculo y detección de pasos, tales como la cantidad de muestras por ventana, los umbrales de detección de picos, las duraciones máximas y mínimas para la detección de pasos, y otros parámetros relacionados con la precisión y el conteo de pasos.

4.1.1 Parámetros

- **PED_SAMPLE_CNT_L**: 0x32 (Cantidad de muestras por ventana para el cálculo).
- **PED_SAMPLE_CNT_H**: 0x00 (Cantidad de muestras por ventana para el cálculo).
- **PED_FIX_PEAK2PEAK_L**: 0xD0 (Umbral para la detección válida de pico a pico).
- **PED_FIX_PEAK2PEAK_H**: 0x00 (Umbral para la detección válida de pico a pico).
- **PED_FIX_PEAK_L**: 0x66 (Umbral para la detección de picos en comparación con el promedio).
- **PED_FIX_PEAK_H**: 0x00 (Umbral para la detección de picos en comparación con el promedio).
- **PED_TIME_UP_L**: 0xC8 (Duración máxima (ventana de tiempo) para un paso. Si no se detectan picos dentro de este límite, se restablece el cálculo).
- **PED_TIME_UP_H**: 0x00 (Duración máxima (ventana de tiempo) para un paso).
- **PED_TIME_LOW**: 0x14 (Duración mínima para un paso. Los picos detectados dentro de este tiempo se ignoran).
- **PED_CNT_ENTRY**: 0x05 (Cantidad mínima de pasos continuos para comenzar el conteo de pasos válidos).
- **PED_FIX_PRECISION**: 0x00 (Se recomienda un valor de 0 para una mayor precisión).

- **PED_SIG_COUNT**: 0x01 (Número de pasos tras los cuales se actualizan los registros del podómetro).
- **PEDOMETER_EN**: 0x10 (Máscara para habilitar el podómetro).

4.2 Configuración Del DMA.

La implementación del reloj implica el uso del DMA para brindar una especia de paralelismo al flujo del reloj, dado que el puerto `spi1` está constantemente actualizando la pantalla y enviando datos de los buffers donde se guardan los colores y las imágenes a mostrar en la pantalla, utilizar el DMA brinda una optimización que permite liberar de tareas concurrentes al nucleo, para que este se encargue de procesar los datos.

4.3 Código de Configuración del DMA

```
void config_dma(void) {
   dma_tx = dma_claim_unused_channel(true); // Reclama un canal DMA
    // Configuración por defecto
    c = dma_channel_get_default_config(dma_tx);

    // Configuración del tamaño de transferencia
    channel_config_set_transfer_data_size(&c, DMA_SIZE_8);

    // Asigna el canal DREQ del SPI para sincronización
   channel_config_set_dreq(&c, spi_get_dreq(SPI_PORT, true));
}
```

4.4 Descripción de la Configuración

1. **Reclamo del canal DMA**: Se utiliza `dma_claim_unused_channel` para obtener un canal libre.

2. **Configuración por defecto**: Se aplica `dma_channel_get_default_config` para inicializar los parámetros básicos.

3. **Tamaño de transferencia**: El tamaño de los datos transferidos se define como 8 bits (1 byte).

4. **Asignación de DREQ**: Se establece el periférico SPI como fuente de la solicitud de transferencia (`spi_get_dreq`).

4.5 Resumen del Proceso de Configuración

La configuración del DMA se complementa con los siguientes pasos en el código principal:

- Configuración del canal con `dma_channel_configure` para definir origen, destino y cantidad de datos.
- Habilitación de interrupciones del DMA con: `dma_channel_set_irq0_enabled`.
- Implementación de un manejado de interrupción para notificar la finalización de la transferencia.

5 LVGL

Para el uso de una interfaz gráfica en el reloj se utilizó la librería de LVGL (light and Versatile Graphics Library) la cual es una librería de código abierto para crear interfaces gráficas en sistemas embebidos de manera rápida y fácil, este librería cuenta con una optimización en el uso de memoria lo que la hace perfecta para su uso en este tipo de aplicaciones. Dentro del proyecto esta libreria se integra para crearlos widgets y elementos basicos de la pantalla, asi como la administración de recursos de la interfaz grafica y la actualización dinamica de los datos en la interfaz grafica.

5.1 Creación de una Pantalla con LVGL

La función `create_screen1()` configura una pantalla principal en la librería LVGL (*LittlevGL*), donde se muestran varios widgets que incluyen etiquetas (*labels*), barras de progreso (*bars*) y arcos (*arcs*). Estos elementos permiten mostrar información clave como la hora, fecha, estado de la batería, pasos, calorías y distancia recorrida. A continuación, se explica cómo se utilizan las etiquetas y otros widgets en esta función.

5.1.1 Etiquetas en LVGL

En LVGL, las etiquetas (*labels*) se crean utilizando la función `lv_label_create()`, la cual genera un objeto de texto que puede personalizarse con diferentes fuentes, tamaños y estilos. Las etiquetas se alinean en la pantalla o respecto a otros widgets mediante `lv_obj_align()` o `lv_obj_align_to()`.

- **Etiqueta de hora:** Se muestra en el centro superior de la pantalla utilizando la fuente `montserrat_48` y el texto inicial `"15:45"`.
- **Etiqueta de fecha:** Ubicada debajo de la hora, utiliza la fuente `montserrat_18` y muestra la fecha inicial `"10/12/2024 WED"`.
- **Etiqueta de batería:** Muestra el símbolo de batería lleno (`LV_SYMBOL_BATTERY_FULL`) y el porcentaje inicial de `"100%"`, alineada en la esquina superior derecha.
- **Etiqueta de pasos:** Asociada a una barra de progreso (*progress bar*) que indica el número de pasos realizados, inicializando el texto en `"Steps: 0"`.
- **Etiqueta de calorías:** Mostrada debajo de un arco (*arc*) que representa visualmente las calorías quemadas. El texto inicial es `"0"`.
- **Etiqueta de distancia:** Similar a la de calorías, pero asociada a un arco que muestra la distancia recorrida. El texto inicial es `"0m"`.

5.1.2 Widgets Adicionales

- **Barra de pasos:** Creada con `lv_bar_create()`, esta barra visualiza el progreso hacia una meta de 1000 pasos.
- **Arco de calorías y distancia:** Los arcos (`lv_arc_create()`) muestran gráficamente el progreso en calorías quemadas y distancia recorrida. Estos se configuran para rotar 270 grados y eliminar su estilo de perilla (`LV_PART_KNOB`).
- **Indicador de pulso cardíaco:** Llamado mediante la función: `create_heart_pulse_indicator()`, este widget se incluye como parte de la pantalla principal.

5.1.3 Resumen del Flujo

Cada widget se crea dentro de la pantalla `screen1`, que actúa como contenedor principal. Los estilos y alineaciones se aplican para garantizar una disposición coherente y estéticamente agradable de los elementos. Este diseño permite presentar múltiples tipos de datos relevantes de manera simultánea en un dispositivo compacto.

La interfaz gráfica resultante del uso de esta librería se muestra a continuación:

Figure 5: Interfaz Grafica

5.2 Actualización Dinámica de Variables y Widgets

El código en el archivo `lib.c` para actualizar dinámicamente los valores de diversas métricas mostradas en los widgets del reloj inteligente, utilizando la librería LVGL. A continuación, se describen las principales funciones y cómo actualizan las etiquetas, barras y arcos de la interfaz gráfica.

5.2.1 Actualización de Pasos (`update_steps`)

Esta función utiliza el conteo de pasos obtenido desde el sensor IMU, formateando el valor en una cadena con la función `snprintf()`. El progreso se muestra en una barra (*bar*) configurada mediante `lv_bar_set_value()`,

y el texto con el número de pasos se actualiza en una etiqueta (*label*) utilizando `lv_label_set_text()`.

5.2.2 Actualización de Tiempo y Fecha (`update_time`)

Se obtiene la fecha y hora actuales a través de una estructura `datetime_t`. Los datos se formatean como cadenas de texto para mostrar:

- **Hora:** Formato `HH:MM`, actualizada en una etiqueta centrada en la parte superior.
- **Fecha:** Formato `DD/MM/AAAA DOW` (día de la semana abreviado), mostrada debajo de la hora.

5.2.3 Actualización de Distancia (`update_distance`)

La distancia recorrida se calcula multiplicando el número de pasos por un factor constante (0.75 metros por paso). Un arco (*arc*) visualiza el progreso hacia una distancia máxima de 15 km, mientras que el valor en metros se muestra en una etiqueta debajo del arco.

5.2.4 Actualización del Nivel de Batería (`update_battery`)

Se calcula el nivel de batería en porcentaje a partir del voltaje leído por el ADC. Dependiendo del voltaje, se elige un símbolo de batería (`LV_SYMBOL_BATTERY_FULL`, `LV_SYMBOL_BATTERY_3`, etc.), y se concatena con el porcentaje. El resultado se muestra en una etiqueta alineada en la esquina superior derecha.

5.2.5 Actualización de Ritmo Cardíaco (`update_hr`)

El ritmo cardíaco en pulsaciones por minuto (bpm) se calcula y se muestra como texto en una etiqueta junto al indicador gráfico de pulso cardíaco.

5.2.6 Actualización de Calorías (`update_calories`)

Las calorías quemadas se calculan en función del número de pasos y la frecuencia cardíaca. Este valor se visualiza mediante un arco y una etiqueta, donde el máximo mostrado es 50 calorías.

5.2.7 Flujo Principal de Actualización

El `main` inicializa las variables y widgets del reloj inteligente, ejecutando las funciones de actualización dentro de un bucle principal. Esto permite que los datos del usuario se actualicen en tiempo real conforme se obtienen nuevos valores de los sensores.

5.2.8 Aspectos Destacados

- Uso de `snprintf()` para formatear cadenas antes de mostrarlas.
- Actualización visual mediante las funciones de LVGL: `lv_label_set_text()`, `lv_bar_set_value()`, `lv_arc_set_value()`, entre otras.
- Integración con sensores para datos dinámicos como pasos, batería y ritmo cardíaco.
- Disposición coherente de widgets en la pantalla con funciones como `lv_obj_align()`.

6 Flujo Del Programa.

Visto todas las configuraciones y el funcionamiento de los periféricos y las librerías, todos estos conceptos y funciones se unen en el siguiente diagrama de flujo para la integración completa del reloj.

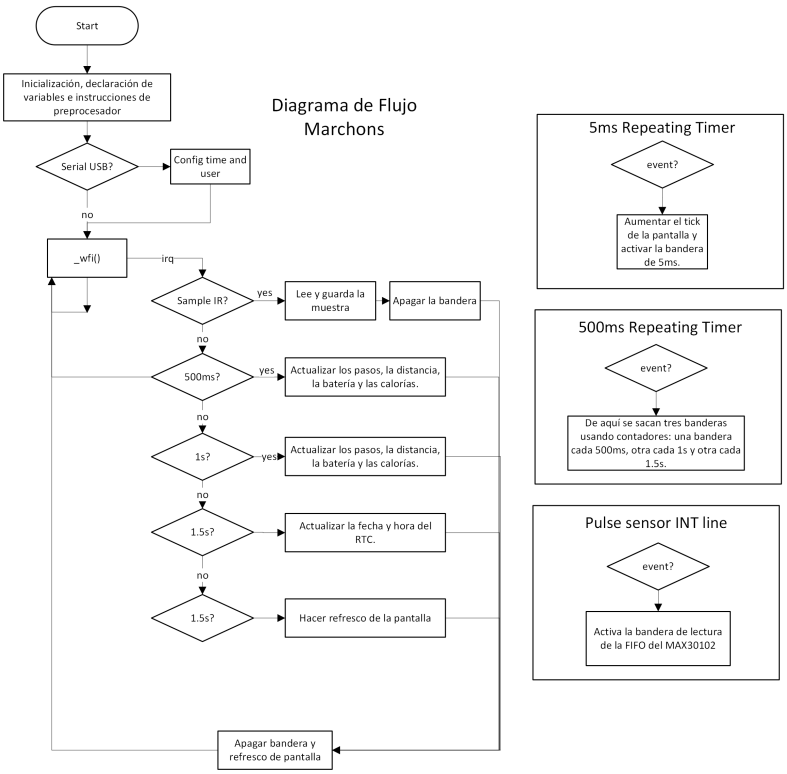

Figure 6: Diagrama de flujo del programa

1. **Inicio:**
    - El sistema comienza con la inicialización, donde se declaran las variables globales y se configuran las instrucciones del preprocesador necesarias para el funcionamiento.
2. **Verificación de conexión USB:**
    - Se verifica si existe una conexión USB activa.
    - En caso afirmativo, el sistema procede a configurar la hora y los parámetros del usuario.
    - Si no hay conexión USB, el sistema continúa con el flujo principal.
3. **Lazo principal:**
    - El sistema entra en un estado de bajo consumo energético utilizando la instrucción `_wfi()` (“Wait for Interrupt”), esperando eventos externos o interrupciones.

- En este lazo, se ejecutan las siguientes tareas periódicamente dependiendo de las banderas activadas:
  - Si se activa la bandera de 500 ms, se actualizan los valores de pasos, distancia recorrida, nivel de batería y calorías quemadas.
  - Si se activa la bandera de 1 s, se realiza nuevamente la actualización de pasos, distancia, batería y calorías.
  - Si se activa la bandera de 1.5 s, se actualizan la fecha y la hora del RTC (Real-Time Clock) y se refresca la pantalla.

4. **Timers repetitivos:**
   - El sistema cuenta con timers configurados para eventos específicos:
     - **Timer repetitivo de 5 ms:** Incrementa el contador de ticks y activa la bandera correspondiente.
     - **Timer repetitivo de 500 ms:** Utiliza contadores para activar banderas en intervalos de 500 ms, 1 s y 1.5 s.
5. **Línea de interrupción del sensor de pulso:**
   - Si se detecta un evento en la línea de interrupción del sensor de pulso, se activa la bandera para iniciar la lectura de la FIFO del sensor MAX30102.
6. **Fin del lazo principal:**
   - Finalmente, se apagan las banderas activas y se realiza un refresco final de la pantalla antes de volver al estado de bajo consumo.

7 Inicialización y Configuración del Smartwatch

El proceso de inicialización y configuración del smartwatch comienza con la activación de la interfaz SPI para la comunicación con el RTC (*Real-Time Clock*). A continuación, se verifican y configuran los estados iniciales del dispositivo:

1. **Protección contra escritura:** Se verifica si el RTC está protegido contra escritura. Si este es el caso, se habilita la escritura en el dispositivo y se inicializan en cero las posiciones de memoria utilizadas para almacenar los pasos acumulados.
2. **Estado de funcionamiento:** Si el RTC no está en funcionamiento, se establece la fecha y hora a un valor de respaldo (`backup_time`) y se habilita su funcionamiento. Además, se reinician en cero las posiciones de memoria correspondientes a los pasos.
3. **Validación de fecha y hora:** En caso de que el RTC indique una pérdida de confianza en la fecha y hora, se restaura el tiempo utilizando el valor de respaldo y se valida la información almacenada.
4. **Inicialización de parámetros del usuario:** Los valores iniciales del usuario se configuran por defecto en las posiciones de memoria dedicadas:
    - Peso: 60 kg.
    - Altura: 170 cm.
    - Edad: 50 años.
5. **Configuración mediante conexión USB:** Si se detecta una conexión USB y el sistema está en modo de configuración, se permite al usuario personalizar los parámetros del dispositivo a través de la consola serial:
    - Fecha y hora: El usuario ingresa año, mes, día, hora y minutos. La validez de la fecha se verifica automáticamente antes de continuar.
    - Peso, altura y edad: Estos parámetros son configurados según las entradas del usuario, verificando que estén dentro de límites razonables.
    - Reinicio de pasos: Se reinician a cero las posiciones de memoria relacionadas con los pasos acumulados.

    Finalmente, se actualiza la fecha y hora del RTC, y se habilita su funcionamiento.

Este flujo de inicialización asegura que el smartwatch comience con parámetros consistentes, permitiendo al usuario establecer configuraciones person-

alizadas cuando sea necesario.