Reporte Trabajo Practico Final PDM

## **Alumno: Durante Matias Nahuel**

## **Link al Repositorio:**

<https://github.com/matuuted/Proyecto_Final>

## **Descripción del Practico Final**

El proyecto final consiste en el desarrollo de un firmware embebido para la placa **NUCLEO-STM32F446RE**, diseñado para medir y visualizar la inclinación de un objeto en tiempo real.

El sistema integra diferentes periféricos conectados mediante el bus I²C y una interfaz UART, permitiendo obtener las lecturas del sensor MPU6050, calcular el ángulo, mostrarlo en un LCD 16×2, y transmitirlo a una PC para su monitoreo por consola.

Además, el LED2 de la placa actúa como un indicador visual cuya frecuencia de parpadeo varía en función del ángulo medido.

## **Estructura del Software**

El firmware está estructurado en módulos para cada periférico, más un archivo principal (***sys\_sm.c***) donde se desarrolló la máquina de estados encargada del control y funcionamiento general del sistema:

### SYS\_SM.c

Es el **módulo principal del proyecto**, encargado del desarrollo de la **máquina de estados del sistema**.

Su función es coordinar todo el flujo de ejecución del firmware, gestionando la inicialización, lectura de sensores, actualización de la interfaz de usuario y la señalización visual mediante el LED propio de la placa.

#### Responsabilidades principales

* **Inicializar los periféricos:** LCD, RTC (DS3231) y sensor MPU6050.
* **Leer y procesar los datos del sensor** para obtener el ángulo de inclinación.
* Actualizar la información en el **LCD** y enviar los datos por **UART**.
* **Modular el LED** de la placa NUCLEO según el ángulo calculado.

#### SOURCE FILES

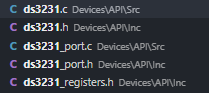
* **sys\_sm.c:** Contiene la implementación completa de la máquina de estados:
  + - SM\_InitOS(): inicializa las estructuras internas y crea la tarea principal del sistema
    - SM\_Iter(): ejecuta el loop periódico manejando los distintos estados del sistema.

#### HEADER FILES

* **sys\_sm.h:** Define los tipos públicos, enumeraciones de estados, prototipos de funciones principales (SM\_InitOS, SM\_Iter) y estructuras globales del sistema

### DRIVER DS3231

El **DS3231** es un **reloj en tiempo real (RTC)** que mantiene la hora y fecha del sistema. La idea de su utilización es **poder generar un reporte con fecha y hora sobre el ángulo medido por el MPU6050**.

El driver actual consta de 5 archivos:  


A continuación, se describe su funcionalidad:

#### SOURCE FILES

* **ds3231.c:** Se declaran las funciones de inicialización, lectura y escritura del reloj.
* **ds3231\_port.c:** Implementa las funciones de bajo nivel para leer y escribir registros a través del bus I²C.

#### HEADER FILES

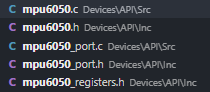
* **ds3231.h:** Se definen estructuras y prototipos de funciones públicas del DS3231.
* **ds3231\_port.h:** Define los prototipos y macros internos utilizados por **ds3231\_port.c**
* **ds3231\_registers.h:** Contiene las direcciones de los registros y las máscaras de bits utilizadas en el driver del DS3231.

### DRIVER MPU6050

El **MPU6050** es un sensor que combina un acelerómetro triaxial y un giroscopio, permitiendo obtener información sobre la inclinación y el movimiento del sistema.

Para este proyecto, se desarrolló un driver modular para manejar el chip, encargado de inicializar el sensor y realizar lecturas periódicas de los ejes X, Y y Z, para posteriormente calcular el ángulo de inclinación.

El módulo está compuesto por los siguientes archivos:



#### SOURCE FILES

* **mpu6050.c:** Contiene la inicialización del sensor, lectura de registros y conversión de valores a unidades físicas.
* **mpu6050\_port.c:** Al igual que ds3231\_port.c, implementa las funciones de acceso directo a los registros del dispositivo a través del bus I²C.

#### HEADER FILES

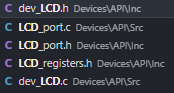
* **mpu6050.h:** Define las estructuras, enumeraciones y funciones públicas del módulo, así como el ***MPU6050\_Handler*** utilizado por el firmware para encapsular la información del dispositivo.
* **mpu6050\_port.h:** Declara los prototipos y constantes internas del puerto, utilizadas por mpu6050\_port.c.
* **mpu6050\_registers.h:** Define las direcciones de los registros internos del sensor.

### DRIVER LCD 16x2

El **LCD 16×2** permite mostrar la hora actual y el ángulo de inclinación medido por el sensor MPU6050.

Para su control, se desarrolló un driver completamente modular que utiliza el expansor **PCF8574**, conectado mediante el bus I²C, permitiendo reducir el número de pines utilizados del microcontrolador.

El módulo que maneja el LCD está compuesto por los siguientes archivos:



#### SOURCE FILES

* **dev\_LCD.c:** Contiene las funciones principales de control del display: inicialización, escritura de texto, limpieza y posicionamiento del cursor.
* **LCD\_port.c:** Implementa las rutinas de bajo nivel necesarias para enviar comandos y datos al LCD mediante el PCF8574 (que funciona mediante I²C).

#### HEADER FILES

* **dev\_LCD.h:** Define los prototipos públicos, constantes y funciones auxiliares del módulo.
* **LCD\_port.h:** Declara las macros internas de conexión entre el expansor PCF8574 y el LCD, a través del bus I²C.
* **LCD\_registers.h:** Contiene los comandos, flags y direcciones de memoria del controlador HD44780 utilizados internamente para el control del LCD.

## **Periféricos**

En el proyecto se utilizaron los siguientes periféricos del microcontrolador STM32F446RE:

* **LED GPIO** (LD2): Se utiliza el LED integrado en la placa como indicador visual del estado del sistema. Su frecuencia de parpadeo varía de acuerdo al ángulo de inclinación medido por el MPU6050
* **I²C1** (PB8 - SCL / PB9 - SDA): Es el bus principal de comunicación utilizado en este proyecto. A través de él se conectan los tres periféricos externos:
  + **MPU6050**: sensor acelerómetro/giroscopio para medición de inclinación.
  + **DS3231**: reloj en tiempo real (RTC) que medir la hora actual.
  + **LCD 16×2 con PCF8574**: display para visualizar hora y ángulo medidos.
* **UART4** (PA0 - TX / PA1 - RX): Se utiliza únicamente el canal **TX** para transmitir datos ***desde el microcontrolador hacia una PC host***, donde se pueden visualizar los reportes de la hora y el ángulo medido mediante una terminal serial (en mi caso utilice **PuTTY**).

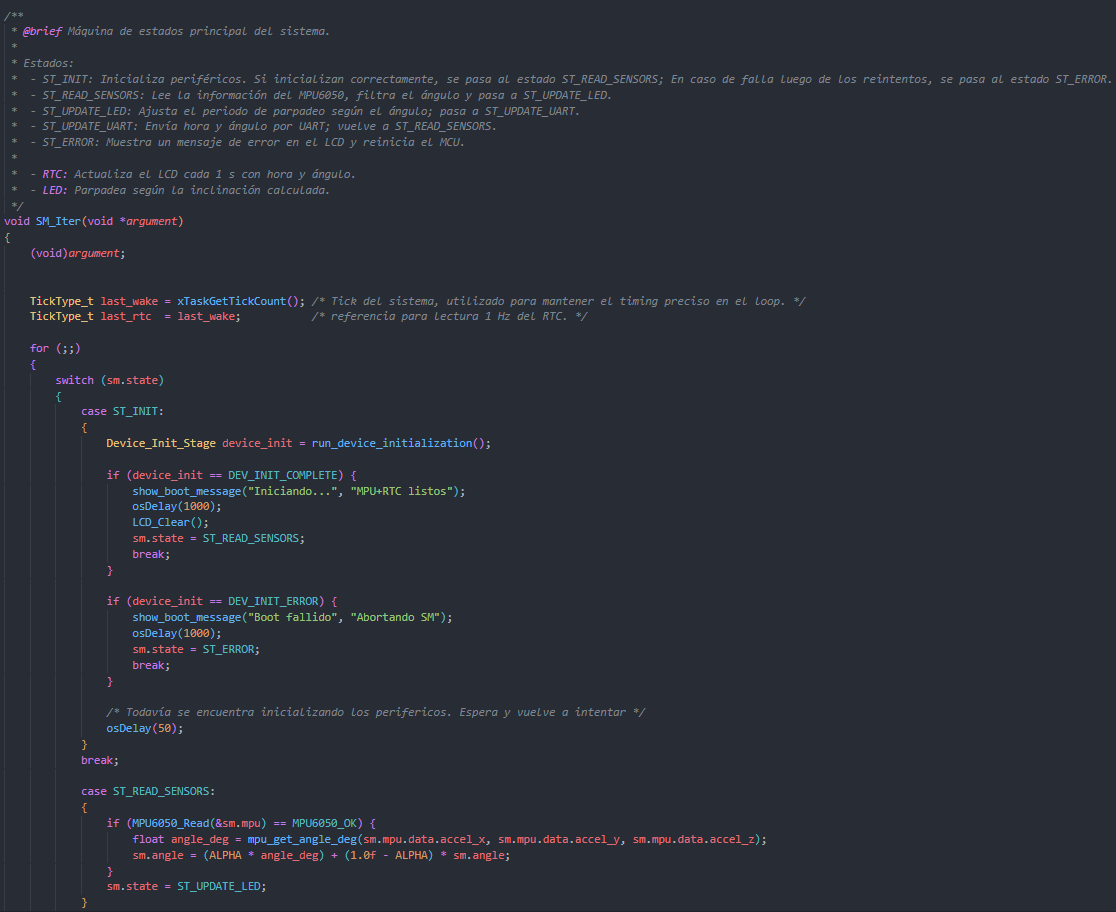
## **Variable Global Privada**

El módulo ***sys\_sm.c***, contiene las siguientes variables globales privadas.

* **boot\_status:** Estructura que encapsula la información sobre el progreso de inicialización de los periféricos, así como los intentos de inicialización y su resultado
* **sm:** Estructura principal de la máquina de estados (SM\_Handler), donde se almacenan las variables del sistema.



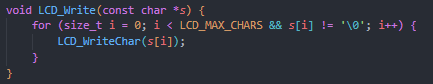
## **MEF**



## **Power of ten rules utilizadas**

Se detallan a continuación las “Power of Ten Rules” utilizadas, junto con una explicación breve para cada caso y ejemplos donde es posible:

1. **Evitar control de flujo completo como goto y recursiones**: No se utilizan ***goto*** ni funciones recursivas en ningún módulo.
2. **Todos los bucles deben tener límites fijos:** Se utilizaron bucles con limites fijos. Por ejemplo, para enviar datos al LCD, hay un for loop que itera según la cantidad de caracteres que quiero enviar.



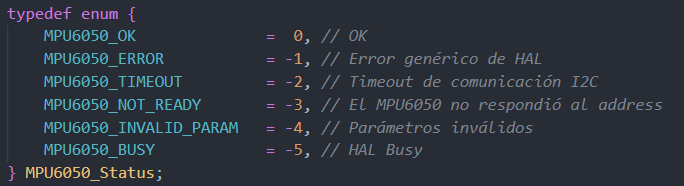
1. **Evitar el uso de asignación dinámica de memoria:** toda la memoria fue asignada en compilación.
2. **Restringir las funciones a una página impresa simple:** hay **una función que no cumple esto**, **SM\_Iter ().** Esta función extensa porque ***contiene el switch/case de estados de la SM***, así como las tareas que se ejecutan periódicamente. Se mantuvo así porque es más simple para leer la funcionalidad en un solo lugar.
3. **Usar un mínimo de dos chequeos en tiempo de ejecución por función (assertions).** En general, la gran mayoría de funciones realizan un pre chequeo de la información que ingresa, **aunque no todas cumplen el chequeo post**.
4. **Restringir la visibilidad de la información al mínimo posible:** Cada módulo expone las funciones y variables necesarias para cumplir su funcionamiento.
5. **Chequear el valor de retorno de todas las funciones non-void:** se intentó realizar la comprobación en cada una de las funciones que tienen un parámetro de retorno.
6. **Usar el preprocesador moderadamente:** Se usa el preprocesador de forma mínima. Solo se utiliza en #defines e #ifdef, pero no en gran cantidad.
7. **Limitar el uso de punteros a una sola desreferencia. No usar punteros a funciones:** No se utilizaron punteros a funciones en la maquina de estados del proyecto.
8. **Compilar con todos los warnings activos.** No hay warning ni errores en el compilado del proyecto.

## **Técnica de programación**

Además de las prácticas ya mencionadas (Power of Ten), se aplicaron las siguientes técnicas para mantener el código seguro, legible y portable.

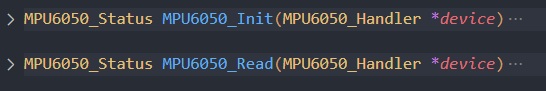
##### Control de Errores

Todas las funciones principales de los módulos retornan una variable de su tipo, mapeada con los valores de provenientes de la HAL de STM “HAL\_StatusTypeDef”.



### Estandarización de nombres:

Las funciones comienzan con el prefijo del módulo en mayúscula (por ejemplo: **LCD\_, DS3231\_, MPU6050\_, SM\_**), permitiendo identificar fácilmente de qué módulo proviene cada función cuando se utilizan desde archivos externos.



## **Comentario que explique por qué y no el como**

La función que tiene el SM que se encarga de la inicialización de los periféricos realiza el siguiente orden (LCD→RTC→MPU) para poder brindar feedback del estado de arranque y avisar en caso de fallas en la inicialización, utilizando el LCD como medio.

