### Alumno: Durante Matias

### Plataforma embebida: *NUCLEO-STM32F446re*

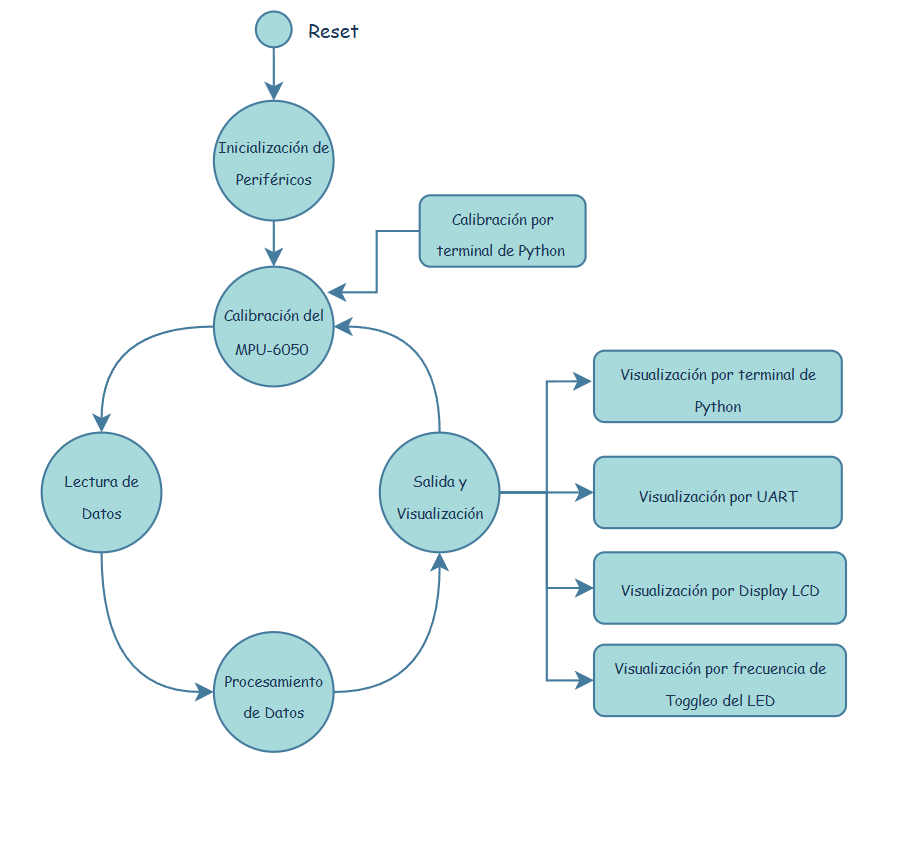
## Aplicación:

Cada vez que el equipo detecte un cambio de inclinación mediante el sensor **MPU-6050**, se actualizará un **patrón visual en el LED de la placa Nucleo**, cuya **frecuencia de titilación dependerá del ángulo** medido.  
Además, se enviará por la consola de la **UART** una línea completa con la hora obtenida del **RTC (DS3231)** y los valores de los sensores.  
La configuración del sistema se realizará desde una **consola en Python**, mientras que los **resultados también se visualizarán en un display LCD 16x2**.

### Periféricos:

**I2C,UART, LEDs**

## Diagrama de la MEF y descripción de los estados



A continuación, una breve descripción de cada uno de los estados:

* **Inicialización de periféricos:** es el estado inicial del sistema. Se configuran los periféricos principales (**I²C**, **UART**, **Timers** y **GPIO**) y se verifica la comunicación con el sensor **MPU-6050** y el reloj **DS3231**. Una vez completada la inicialización correctamente, el sistema pasa al estado de calibración.
* **Calibración del MPU-6050:** en este estado se realiza la calibración de los sensores, ya sea de forma automática o a través de la **terminal de Python**. Finalizada la calibración, se pasa al estado de lectura de datos.
* **Lectura de datos:** el sistema toma lecturas del **MPU-6050** mediante el bus I²C a la frecuencia configurada. Los datos obtenidos se almacenan temporalmente para ser procesados en el siguiente estado.
* **Procesamiento de datos:** en este estado se aplica el **filtro complementario del MPU6050**, donde se combina la información del acelerómetro y el giróscopo para obtener los ángulos de pitch y roll. Además, se calcula el ángulo absoluto, que determinará el patrón visual del LED y los datos a transmitir.
* **Salida y visualización:** este estado contiene todas las formas de salida y visualización del sistema. Los resultados procesados se muestran a través de:
  + La **terminal de Python (OPCIONAL) Se puede leer la información en caso de ser requerido**.
  + La **interfaz UART**, enviando una línea con los valores y la hora del RTC.
  + El **display LCD 16×2**, que actualiza la información visible en tiempo real.
  + La **frecuencia de parpadeo del LED de la Nucleo**, que varía según el ángulo de inclinación detectado.

Una vez finalizada la actualización, el sistema vuelve al estado de lectura de datos para continuar el ciclo.

## módulos de software a implementar.

* mpu6050.c / mpu6050.h — Driver del MPU-6050 (usa mpu6050\_port.c).
* mpu6050\_port.c / mpu6050\_port.h — Primitivas I²C (basadas en I2CM\_\*).
* ds3231.c / ds3231.h — Driver del DS3231 (usa ds3231\_port.c).
* ds3231\_port.c / ds3231\_port.h — Primitivas I²C (basadas en I2CM\_\*).
* dev\_UART.c / dev\_UART.h — Envío de tramas por UART, implementado en el curso.
* dev\_ LED.c / dev\_LED.h — Patrones del LED.
* app\_SM.c / app\_SM.h — Máquina de estados principal.

## Prototipos de las principales funciones públicas y privadas de cada módulo:

### mpu6050.c

* **MPU6050\_Status** MPU6050\_Init(MPU6050\_Handler \*device);
  + Inicializa el sensor.
* **MPU6050\_Status** MPU6050\_ReadSelfAddr(uint8\_t \*self\_addr);
  + Lee el registro WHO\_AM\_I. Recibe un puntero donde guardar la dirección y devuelve estado.
* **MPU6050\_Status** MPU6050\_SetDLPF(uint8\_t dlpf\_cfg);
  + Configura el filtro digital. recibe la cfg (0–6) y devuelve estado.
* **MPU6050\_Status** MPU6050\_SetSampleRate(uint8\_t sample\_rate);
  + Configura la tasa de muestreo; recibe divisor y devuelve estado.
* **MPU6050\_Status** MPU6050\_SetAccelRange(MPU6050\_Handler \*device, MPU6050\_AccelFS scale);
  + Configura el rango del acelerómetro. Recibe handler y escala, devuelve estado.
* **MPU6050\_Status** MPU6050\_SetGyroRange(MPU6050\_Handler \*device, MPU6050\_GyroFS scale);
  + Configura el rango del giroscopio. Recibe handler y escala, devuelve estado.
* **MPU6050\_Status** MPU6050\_ReadRaw(MPU6050\_DataRaw \*raw);
  + Lee 14 bytes crudos de datos; recibe la estructura y devuelve estado.
* **MPU6050\_Status** MPU6050\_Read(MPU6050\_Handler \*device, MPU6050\_Data \*data);
  + Lee datos convertidos (g, dps, °C).
* **MPU6050\_Status** MPU6050\_Reset(void);
  + Resetea el dispositivo.
* **MPU6050\_Status** MPU6050\_Wake(uint8\_t clock);
  + Despierta el dispositivo y selecciona fuente de reloj.

### **mpu6050\_port.c**

* **HAL\_StatusTypeDef** MPU6050\_is\_ready(void);
  + Verifica si el dispositivo responde en el bus I²C.
* **HAL\_StatusTypeDef** MPU6050\_register\_write(uint8\_t reg, uint8\_t data);
  + Escribe un byte en un registro. Recibe dirección y dato.
* **HAL\_StatusTypeDef** MPU6050\_register\_block\_write(uint8\_t \*data, uint16\_t len);
  + Escribe varios bytes secuenciales. Recibe buffer y longitud.
* **HAL\_StatusTypeDef** MPU6050\_register\_read(uint8\_t reg, uint8\_t \*data);
  + Lee un byte de un registro.
* **HAL\_StatusTypeDef** MPU6050\_register\_block\_read(uint8\_t reg, uint8\_t \*data, uint16\_t len);
  + Lee varios bytes secuenciales. Recibe dirección inicial, buffer y longitud, devuelve estado HAL.

### **ds3231.c**

* **DS3231\_Status** DS3231\_Init(void);
  + Inicializa el RTC y verifica presencia en el bus.
* **DS3231\_Status** DS3231\_ReadTime(DS3231\_Time \*time);
  + Lee la hora/fecha actual. Recibe puntero a estructura de tiempo y devuelve estado.
* **DS3231\_Status** DS3231\_SetTime(uint8\_t year, uint8\_t month, uint8\_t date, uint8\_t day, uint8\_t hour, uint8\_t min, uint8\_t sec);
  + Configura fecha y hora.
* **DS3231\_Status** DS3231\_GetTemperature(float \*temp);
  + Lee temperatura interna; recibe puntero a float y devuelve estado.
* **DS3231\_Status** DS3231\_Enable32KHz(bool enable);
  + Habilita/deshabilita salida 32kHz.
* **DS3231\_Status** DS3231\_SetSQWFreq(uint8\_t rs\_bits);
  + Configura frecuencia de onda cuadrada. Recibe bits RS y devuelve estado.
* **DS3231\_Status** DS3231\_SetAging(int8\_t offset);
  + Ajusta offset de envejecimiento; recibe valor y devuelve estado.
* **DS3231\_Status** DS3231\_GetAging(int8\_t \*offset);
  + Lee offset de envejecimiento. Recibe puntero a int8 y devuelve estado.

### **ds3231\_port.c**

* **HAL\_StatusTypeDef** DS3231\_is\_ready(void);
  + Verifica si el dispositivo responde.
* **HAL\_StatusTypeDef** DS3231\_register\_write(uint8\_t reg, uint8\_t data);
  + Escribe un byte en un registro.
* **HAL\_StatusTypeDef** DS3231\_register\_block\_write(uint8\_t \*data, uint16\_t len);
  + Escribe varios bytes. Recibe buffer y longitud, devuelve estado HAL.
* **HAL\_StatusTypeDef** DS3231\_register\_read(uint8\_t reg, uint8\_t \*data);
  + Lee un byte de un registro
* **HAL\_StatusTypeDef** DS3231\_register\_block\_read(uint8\_t reg, uint8\_t \*data, uint16\_t len);
  + Lee varios bytes secuenciales. Recibe dirección inicial, buffer y longitud, devuelve estado HAL.

### **DEV\_TIM\_LED.c / .h**

* **void** LED\_Init(void);
  + Configura el LED y el TIMs (blink/breathe). No recibe parámetros. No devuelve.
* **void** LED\_SetBlinkHz(float f\_hz);
  + Ajusta frecuencia de titilación
* **void** LED\_SetBreathe(float f\_hz);
  + Activa efecto de respirar mediante la modulación de PWM (TIM3).

### **DEV\_LCD.c / .h**

* **bool** LCD\_Init(void);
  + Inicializa el LCD. Devuelve true si OK.
* **void** LCD\_Clear(void);
  + Limpia la pantalla y vuelve a (0,0).
* **void** LCD\_Home(void);
  + Cursor a origen sin borrar contenido.
* **void** LCD\_SetCursor(uint8\_t col, uint8\_t row);
  + Posiciona el cursor en columna y fila.
* **void** LCD\_WriteChar(char c);
  + Escribe un carácter en la posición actual.
* **void** LCD\_Write(const char \*s);
  + Escribe una cadena sin salto de línea.
* **void** LCD\_Printf(const char \*fmt, ...);
  + Imprime con formato tipo printf.
* **void** LCD\_WriteAt(uint8\_t col, uint8\_t row, const char \*s);
  + Escribe cadena en (col,row).
* **void** LCD\_Backlight(bool on);
  + Enciende/apaga la luz de fondo.
* **void** LCD\_Cursor(bool on);
  + Muestra/oculta cursor.
* **void** LCD\_BlinkCursor(bool on);
  + Activa parpadeo del cursor.

### **DEV\_UART.c / .h**

* **bool** uartInit(void);
  + Inicializa el periférico UART con la configuración definida (baudrate, 8N1, etc). Devuelve True si la inicialización fue exitosa.
* **bool** uartDeInit(void);
  + Desinicializa el periférico UART.
* **void** uartSendString(uint8\_t \* pstring);
  + Envía una cadena terminada en '\0' por el puerto UART.
* **void** uartSendStringSize(uint8\_t \* pstring, uint16\_t size);
  + Envía un bloque de datos de tamaño fijo por UART. Recibe un puntero al buffer y la cantidad de bytes a transmitir.
* **void** uartReceiveStringSize(uint8\_t \* pstring, uint16\_t size);
  + Recibe un bloque de datos de tamaño fijo desde UART. Almacena los bytes en el buffer indicado.