Sistemas Basados en Microprocesador

Departamento de Ingeniería Telemática y Electrónica
Universidad Politécnica de Madrid

2023-24

Integración y desarrollo de una aplicación: Detector de aceleración

Alumno:
A: Lukas Gdanietz de Diego..
B:……..

Puesto Nº: x

1 OBJETIVOS DE LA PRÁCTICA

1.1 Resumen de los objetivos de la práctica realizada

Se deben enumerar los objetivos que ha alcanzado al realizar la práctica. Enumérelos de forma precisa y sencilla.

1.2 Acrónimos utilizados

Identifique los acrónimos usados en su documento.

UART	Universal Asynchronous Receiver Transmitter
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
O.S. / OS	Operating System
HAL	Hardware Abstraction Layer
SWO	Serial Wire Output
CS	Chip Select
CMSIS	Cortex Microcontroller Software Interface Standard

1.3 Tiempo empleado en la realización de la práctica.

40h

1.4 Bibliografía utilizada

[RD1] Hoja de catalogo xxxx, libro, manual de usuario, etc

[RD2] Serial Wire Viewer (SWV) for ARM Cortex-M3 Processors. ARM Real-Time Trace aids debugging. [LINK]

[RD3] Using Serial Wire Viewer (SWV) to Confirm CPU Speed [LINK]

[RD4] How to display printf output over SWO in Keil µVision [LINK]

[RD5] Serial Wire Viewer (SWD + SWO) - fast & native Debugging [LINK]

[RD6] Reference Manual STM32F429

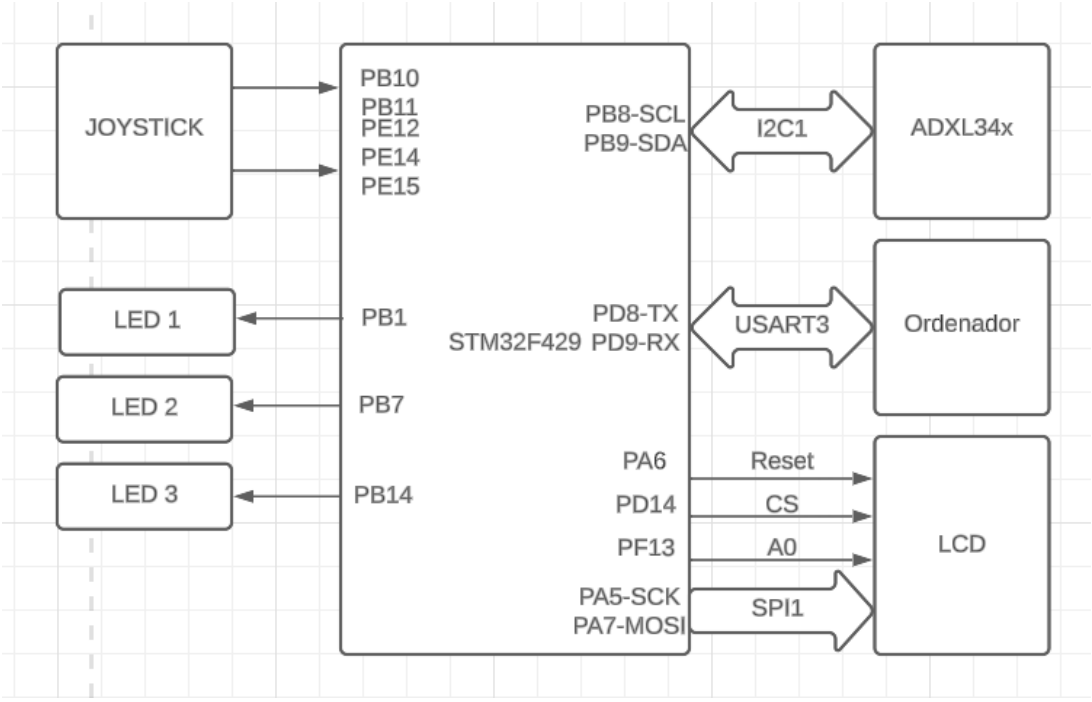
1.5Autoevaluación.

CODIGO	Objetivo	Adquisicion	Anotaciones

RA971	Manejar temporizadores hardware para gestionar la temporización y sincronización de una aplicación	✔️	
RA970	Establecer y gestionar las comunicaciones entre dos sistemas utilizando diferentes interfaces	✔️	
RA907	Desarrollo de aplicaciones en grupos de trabajo	✔️	
RA736	Interpretar las especificaciones de funcionamiento de un sistema basado en microcontrolador de mediana complejidad	✔️	
RA737	Escribir el código necesario para desarrollar una aplicación basada en microcontrolador de mediana complejidad	✔️	
RA730	Conectar un periférico a un microcontrolador utilizando interfaces basadas en protocolos estándar	✔️	
RA735	Analizar la arquitectura software y hardware de sistemas basados en microcontrolador de mediana complejidad	✔️	
RA733	Aprender a manejar cualquier periférico de mediana complejidad de un microcontrolador a partir de la documentación proporcionada por el fabricante	✔️	
RA968	Manejar instrumentación electrónica específica para el desarrollo de sistemas basados en microprocesador	✔️	
RA738	Elaborar el informe que justifica y describe la toma de decisiones adoptadas en el desarrollo de un proyecto y defenderlo oralmente con precisión y detalle	✔️	
RA734	Manejar entornos de CAD para la codificación, la compilación y la depuración de aplicaciones basadas en microcontrolado	✔️	

2 RECURSOS UTILIZADOS DEL MICROCONTROLADOR

2.1Diagrama de bloques hardware del sistema.



2.2Cálculos realizados y justificación de la solución adoptada.

- USART: El STM32 se configura para que utilice el periférico USART3 Se configura el periférico para que funcione de forma asíncrona, con 8 bits de datos, 0 bits de paridad, 1 bit de stop, y una velocidad de transmisión de 115200 Baudios. Además, se activan tanto la recepción como la transmisión, con salida en los pines PD8 (TX) & PD9 (RX).
- I2C: El STM32 se configura para que utilice el periférico I2C1. El periférico se inicializa para que opere a 400Khz.
- SPI: El STM32 se configura para utilizar el periférico SPI1 Se configura para que funcione en modo master, en modo 3 (Lógica en 1, y polaridad en 1). El tamaño de palabra es 8, y el sentido es del bit más significativo al bit menos significativo. Además, se configura para que opere a 2 MHz.

3 SOFTWARE

3.1 Descripción de cada uno de los módulos del sistema

En primer lugar quiero expresar claramente mis ideas y opiniones sobre la estructuracion de un proyecto de esta indole. En un programa embedded hay (en mi opinion) 3 capas.

1. La primera capa es la capa de configuracion y capa base. Aqui es donde se configuran los perifericos del microcontrolador y se inicializa el sistema. En esta capa se crean ademas las funciones de acceso para la siguiente capa

2. La segunda capa es la capa de software de abstraccion. Esta capa provee de metodos y funciones que no dependen del microcontrolador ni de su hardware. Tan solo usan funciones y metodos que les ha provisto la anterior capa.

3. La ultima capa es la capa de usuario y de programa. Aqui es donde la logica del proyecto toma lugar

Tener esta distincion permite organizar el proyecto y da lugar a un software limpio y mantenible

3.1.1 Ficheros centrales

Los ficheros centrales son ficheros comunes, usualmente headers (.h) a los cuales todos los modulos del sistema tienen acceso, con el objetivo de establecer reglas y servicios comunes. Ademas de todas las librerias que STM y ARM proveen, he incluido yo mismo algunas. A continuacion se nombran estos ficheros y los servicios que proveen:

- **stm_err.h**: Define una serie de codigos de error que pueden devolver las funciones de los modulos y el programa. Ademas define unos macros con los que se puede assertar y notificar a traves de stderr fallos que pueda estar sucediendo en el sistema
- **stm_log.h:** Define una serie de funciones que permiten el login via SWO trace. El codigo detecta si al iniciar el sistema el debugger esta conectado al sistema, en caso de no estarlo, desactiva las funciones de debuggeo, de tal forma que el sistema no espera para poder enviar por SWO.
- **circ_buf.h**: Provee de una serie de funciones que permiten alocar espacio para un buffer circular. Las funciones son agnosticas del tipo de datos que se introducen. El programa que hace uso de estas funciones, mantiene el "handle" de cada buffer que se implementa, las funciones permiten manipular el buffer circular sin que el codigo principal tenga que saber como funciona por detras.

3.1.2 Biblioteca ADXL34x

Se ha creado una biblioteca que provee funciones de control y lectura del sensor ADXL34x Esta biblioteca ha sido creado bajo la premisa "**Hardware Agnostic**". Esto nos permite utilizar la biblioteca en otros sistemas y/o programas que utilizan otro microcontrolador, un HAL distinto o ni siquieran utilizan una capa de abstraccion (Control directo del periferico I2C). Para conseguir este objetivo se utilizan funciones de acceso al periferico, que se pasan a un "manejador" (handle) que la biblioteca utilizara en cada uno de sus operaciones.

3.1.3 Modulo accel&temp

El modulo accel&temp hace uso de la biblioteca ADXL34x. Realiza la inicializacion del periferico I2C, provee las funciones de acceso al periferico I2C a la biblioteca, e inicializa un hilo que hara las medidas oportunas según el estado en el que se encuentra el sistema. Cada vez que se realiza una medida, se introduce en una cola de mensajes para que el hilo principal pueda recibir la medida, y realizar las operaciones requeridas por el sistema.

3.1.4 Modulo COM

El modulo COM es un modulo especifico del sistema cuya funcion es ocuparse de las comunicaciones con el Host. Para ello emplea el periferico USART a una velocidad de 115200 Baudios, 8 bits de datos, 1bit de stop y sin paridad. El modulo COM puede recibir y transmitir datagramas con una estructura predefinida y longuitud variable. Si el datagrama cumple los controles de validacion y no es descartada, se introduce en una cola de mensajes, donde otro hilo ejecuta la funcion que esta asociado con el ID del datagrama.

3.1.5 Modulo CLOCK

Mantiene la hora gracias a un timer del sistema operativo que llama un callback que incrementa la hora por 1 segundo. Además, incluye una función para cambiar la hora. La hora está disponible globalmente en todo el proyecto.

3.1.6 Modulo Joystick

El estado de los pines del joystick se comprueba y los eventos se envían al hilo principal mediante señales. El hilo del joystick (joystick_thread) espera las señales de interrupción y filtra los eventos del joystick, como pulsaciones largas y cortas. Estos eventos se colocan en una cola de mensajes (queueId) para su procesamiento posterior. La función init_joystick_proc inicializa el hilo del joystick y configura los pines GPIO necesarios para detectar las interrupciones.

El modulo implementa un sistema antirebote que ademas detecta entre pulsaciones cortas y largas. Cada vez que se detecta una pulsacion, se indica en un bit en una ventana corredera. Cada 50ms todos los bits almacenados se desplazan 1 bit hacia la izquierda, introduciendo en el bit menos significativo el estado del pin que produjo la interrupcion en primer lugar. Tras desplazarse 5 bits hacia la izquierda, el bit indica que se produjo un pulso en cierto pin. Si toda la ventana corredera esta activa al final, quiere decir que se produjo un pulso largo, sin embargo si solo el primer bit o un numero inferior a 5 bits esta activo, quiere decir que se produjo tan solo un pulso corto.

3.1.7 Modulo leds_nucleo

El modulo leds_nucleo permite encender o apagar los leds de la placa nucleo según como lo requiera el programa principal. El cambio de estados se realiza a traves de flags de sistema. Cada flag indica si uno de los leds debe estar activo o apagado.

3.1.8 Modulo Principal

El módulo principal gestiona la máquina de estados del dispositivo y coordina las tareas entre varios hilos, incluyendo "lcd", "com", "accel&temp", "Joystick" y "clock". Se encarga de interpretar comandos recibidos por "joystick" y "COM" para actualizar el estado de la máquina de estados según la lógica interna. Además, controla el funcionamiento de otros hilos utilizando Flags, colas de mensajes y funciones específicas según el estado y los comandos recibidos. Internamente, gestiona un buffer circular para almacenar las medidas más recientes.

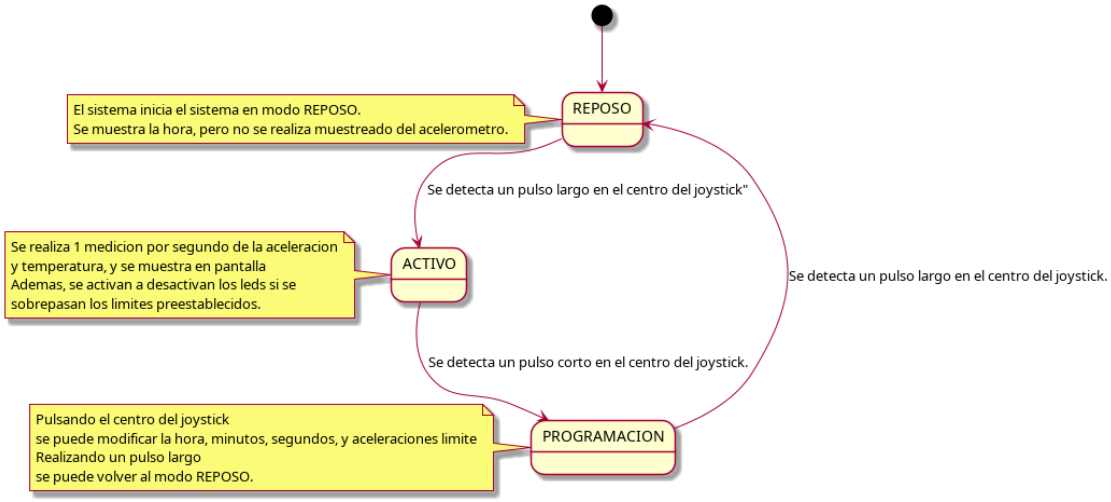
3.2Descripción global del funcionamiento de la aplicación.

En el programa hay principalmente 3 automatas. Estos automatas estan representados (malamente...) en los siguientes diagramas.

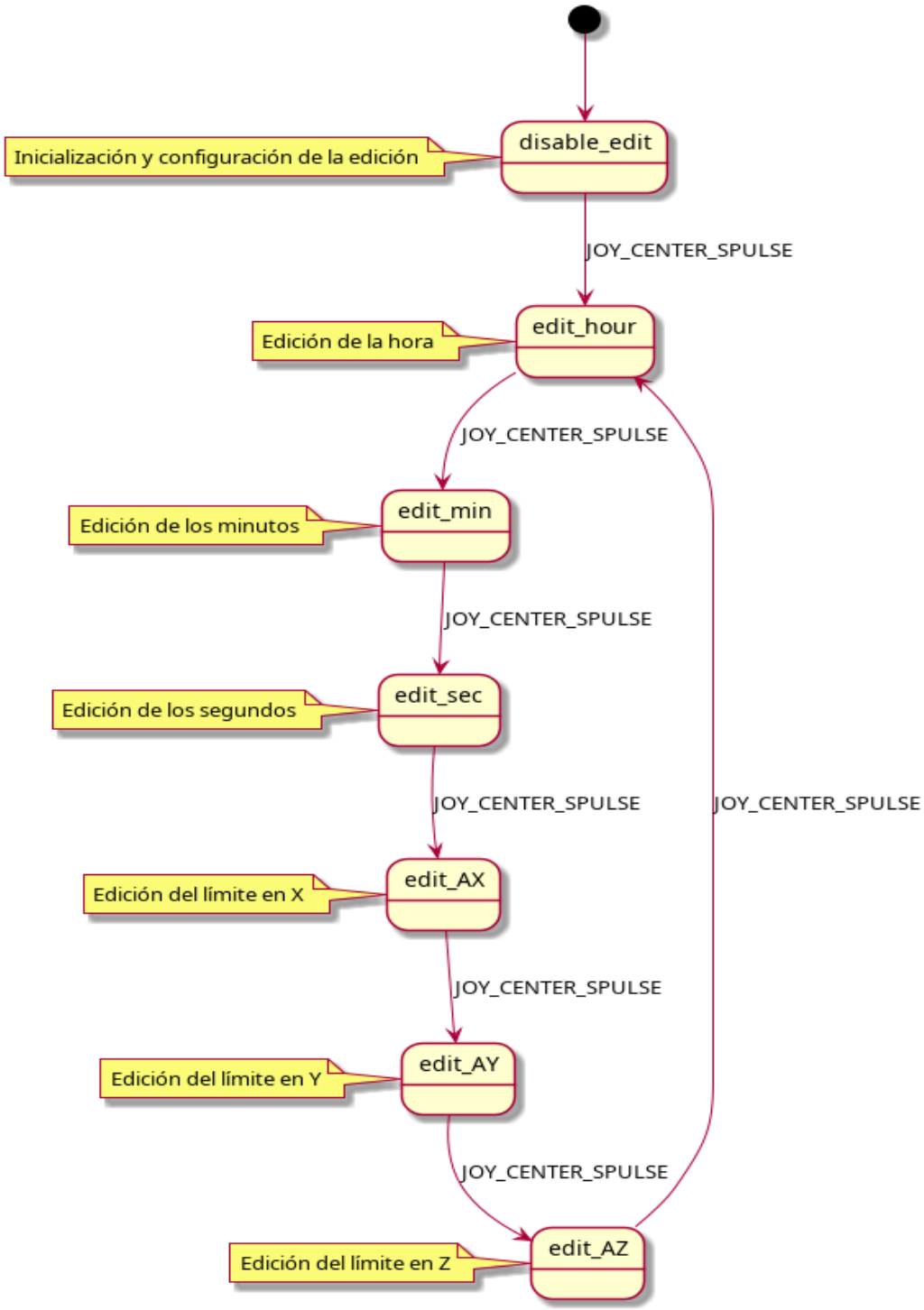
El primer y principal automata es el del programa principal. Tiene 4 estados.

- Estado REPOSO
- Estado ACTIVO
- Estado PROGRAMACION

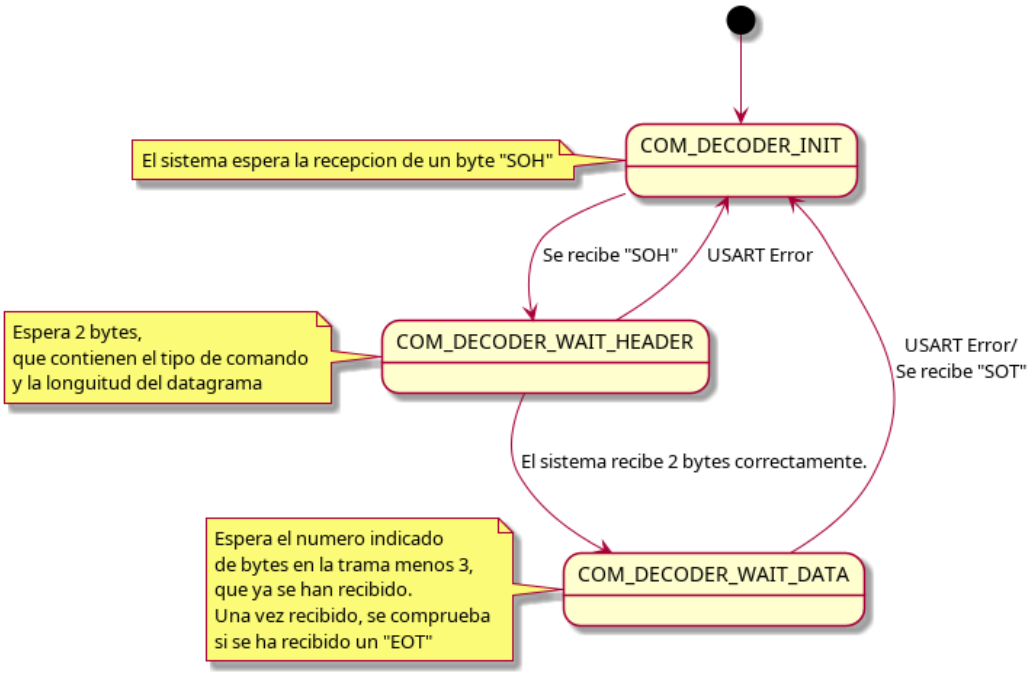
A cada uno de los estados se puede acceder al siguiente estado presionando el boton central del joystick. La siguiente maquina de estados esta presente en el proyecto.



El siguiente Automata es la maquina de estados del modo programacion. Segun los eventos que reciba del modulo JOYSTICK, el Modo programacion actuara de una manera u otra. Presionando el Centro del Joystick permite cambiar la selección del parametro a configurar. Mover el joystick arriba o abajo, permite incrementar o decrementar el valor. Estos cambios se producen con cambios cortos en el joystick. Para salir del modo programacion y fijar los valores, se debe realizar un pulso largo en ele centro del joystick.



Por ultimo esta la maquina de estados del decodificador COM. Para evitar usar los recursos del sistema se de forma continuada se ha buscado una solucion, que trata de estar solamente activo cuando se detecta el Start Of Header (SOH).
Inicia en COM_DECODER_INIT, y solamente espera al indicador de inicio de trama SOH
Cuando lo detecta pasa a COM_DECODER_WAIT_HEADER e incrementa el buffer de recepcion a 2 bytes. En este estado:
Cuando se vuelve a llenar el buffer, se incrementa el buffer por el tamaño que dice la cabezera que tendra el datagrama. Y se pasa al estado COM_DECODER_WAIT_DATA
Una vez se llena el buffer por ultima vez, se comprueba que no se ha cometido ningun error y que el datagrama cierra con End of File (EOF).
Si en cualquiera de estos estados se produce un error, se vuelve al estado COM_DECODER_INIT



3.3Descripción de las rutinas más significativas que ha implementado.

3.3.1 JOYSTICK

El módulo **joystick** se encarga de configurar los pines GPIO asociados para generar interrupciones que notifican eventos al sistema. Estas interrupciones activan un **hilo dedicado al joystick**, utilizando **flags** de hilo como mecanismo de señalización.
Una vez que el hilo recibe el flag correspondiente, se inicia un **temporizador de sistema** encargado de realizar múltiples lecturas de los pines conectados al joystick. Este temporizador ejecuta un ciclo de lectura de los pines un total de **6 veces**, con un intervalo de **50 ms** entre cada lectura. Los valores obtenidos son almacenados en un **array temporal**, que posteriormente pasa por un proceso de filtrado mediante una **máscara lógica**.

- Si **todas las lecturas** resultan positivas tras el filtrado, el sistema interpreta que se ha producido una **pulsación larga** en un pin específico, enviando esta notificación al **hilo principal**.
- Si el filtrado identifica entre **1 y 5 lecturas positivas** (en sentido ascendente), se considera una **pulsación corta**, notificando también este evento al hilo principal.
- En cualquier otro caso, el sistema descarta las lecturas y no se genera ninguna notificación.

Este diseño garantiza la robustez del sistema frente a lecturas espurias o ruido en los pines del joystick, asegurando una gestión precisa de las pulsaciones detectadas.

3.3.2 CLOCK

El módulo **Clock** se encarga de mantener actualizada la **hora del sistema**, utilizando una estructura que almacena la información en términos de **Horas, Minutos y Segundos**. La funcionalidad principal está respaldada por un **temporizador de sistema** que actúa como contador de segundos. Cada vez que este temporizador alcanza el límite establecido, incrementa el valor de los segundos y, cuando es necesario, ajusta los minutos y las horas.

3.3.3 LCD

El módulo **LCD** gestiona la representación gráfica del sistema mediante el manejo de un **buffer gráfico**. Este buffer contiene los caracteres y diagramas que deben mostrarse en función del estado del sistema. La rutina incluye el procesamiento de los datos que deben ser enviados al display y la gestión de la **comunicación con el LCD**, asegurando que la información se actualice en tiempo real y sea presentada de forma clara y consistente.

3.3.4 Accel&temp

El módulo **Accel&Temp** se dedica a realizar la lectura periódica del acelerómetro y el sensor de temperatura, los cuales son consultados una vez por segundo. Los datos obtenidos son procesados y almacenados, quedando disponibles para su uso por otros módulos del sistema.

3.3.5 COM

El módulo **COM** es responsable de la configuración y gestión del periférico **USART**, encargado de la comunicación serial. Su principal función es **enviar y recibir datagramas** con una estructura predefinida, utilizando una **máquina de estados** para garantizar un flujo de datos ordenado y eficiente.

Dada la naturaleza crítica de este módulo, se asigna al hilo encargado de su ejecución la **máxima prioridad** del sistema, asegurando que la comunicación no experimente retrasos ni interrupciones. Este diseño es especialmente importante para aplicaciones donde la sincronización precisa y la integridad de los datos son fundamentales.

3.3.6 Buffer Circular

Se implementaron rutinas clave para la gestión eficiente de un buffer circular:

1. **Inicialización (`init_Buffer`)**: Configura la memoria del buffer, define su capacidad y el tamaño de los datos a almacenar.
2. **Inserción de datos (`introducirDatoBuffer`)**: Añade elementos al buffer de forma cíclica, sobrescribiendo los más antiguos si está lleno.
3. **Eliminación (`borrarDatosBuffer`)**: Borra todos los elementos del buffer, reiniciando sus índices.
4. **Lectura (`recogerDatoBuffer`)**: Permite acceder a un elemento específico dentro del buffer.
5. **Conteo de elementos (`num_elementos`)**: Devuelve el número actual de elementos almacenados.
6.

4 DEPURACION Y TEST

4.1Pruebas realizadas.

Para todos los modulos se ha realizado un testbench donde se podia testear el funcionamiento del modulo mientras se desarrollaba. Para conseguir datos de los testbench se crearon MACROS y un sistema de Logs por SWO (Serial Wire Output) donde se podia depurar el codigo y ver cuando se producia un error. Sin embargo no he cumplido dos objetivos que deberian ser imprescimdibles para estas MACROS:

- Thread Safe:
  Cuando hay multiples hilos operando a la vez, y usan la depuracion serie, se pueden entremezclar los Logs, por tanto no es thread safe, e inutiliza el sentido de esta funcion.
- Deshabilitables por modulo
  La idea es que con los macros se pudiera discernir entre los diferentes tipos de salida y su proveniencia. Ademas debian ser deshabilitables por el fichero donde estuvieran situados

Cada uno de los modulos tiene un testbench en la carperta “test_apps”. Al ejecutar el susodicho testbench, se comproara la inicializacion correcta del modulo a testear y se podran observar en la consola o en el periferico su salida, tal como si estuviera en el programa final.

Idealmente se deberian hacer unity-tests, que establecen los requisitos iniciales y finales, comprueban que en la ejecucion en un sandbox del modulo, se siguen cumpliendo ls requisitos y permite la evolucion del modulo por features, y asegurando al desarrollador que cada vez que realiza un cambio en el modulo, este puede ser testeado antes de ser empujado a “produccion”, y que si hay un cambio que rompa con los requisitos, el programador sera notificado.