Índice del documento

Departamento de Ingeniería Telemática y Electrónica

Universidad Politécnica de Madrid

2023-24

Sistemas Basados en Microprocesador

Integración y desarrollo de

una aplicación:

Controlador de una válvula termostática

Alumno:

A: Sara Jiménez Muñoz

B: Antonio Castellana Hajjar

Puesto Nº: x

[1 Objetivos de la PRÁCTICA 2](#_Toc88129291)

[1.1 Resumen de los objetivos de la práctica realizada 2](#_Toc88129292)

[1.2 Acrónimos utilizados 2](#_Toc88129293)

[1.3 Tiempo empleado en la realización de la práctica. 2](#_Toc88129294)

[1.4 Bibliografía utilizada 2](#_Toc88129295)

[1.5 Autoevaluación. 2](#_Toc88129296)

[2 RECURSOS UTILIZADOS DEL MICROCONTROLADOR 3](#_Toc88129297)

[2.1 Diagrama de bloques hardware del sistema. 3](#_Toc88129298)

[2.2 Cálculos realizados y justificación de la solución adoptada. 3](#_Toc88129299)

[3 SOFTWARE 4](#_Toc88129300)

[3.1 Descripción de cada uno de los módulos del sistema 4](#_Toc88129301)

[3.2 Descripción global del funcionamiento de la aplicación. Descripción del autómata con el comportamiento del software (si procede) 4](#_Toc88129302)

[3.3 Descripción de las rutinas más significativas que ha implementado. 4](#_Toc88129303)

[4 DEPURACION Y TEST 5](#_Toc88129304)

[4.1 Pruebas realizadas. 5](#_Toc88129305)

# Objetivos de la PRÁCTICA

T

## 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.

## Acrónimos utilizados

Identifique los acrónimos usados en su documento.

|  |  |
| --- | --- |
| UART | Universal Asynchronous Receiver Transmitter |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Tiempo empleado en la realización de la práctica.

Debe realizar una descripción sencilla del tiempo que ha dedicado a la realización de las actividades relacionadas con la práctica.

|  |  |
| --- | --- |
|  | **[Tiempo empleado para realizer la práctica]:** El tiempo total empleado ha sido de x horas. |

## Bibliografía utilizada

1. Hoja de catalogo xxxx, libro, manual de usuario, etc

## Autoevaluación.

Autoevalúese comprobando los objetivos de aprendizaje indicados en la guía de la asignatura. Compruebe si supera los objetivos de adquisición obligatoria. Si ha encontrado dificultades en la realización de la práctica indicelo.

# RECURSOS UTILIZADOS DEL MICROCONTROLADOR

## Diagrama de bloques hardware del sistema.

Este apartado debe contener un diagrama de bloques donde se identifiquen claramente los elementos utilizados en el diseño o ejercicio. Debe elaborar una figura que muestre todos los elementos utilizados de la tarjeta NUCLEO STM32F429ZI y su interconexión con los elementos externos (tarjeta de aplicaciones, sensores, etc.).

## Cálculos realizados y justificación de la solución adoptada.

En este punto debe describir como ha configurado cada uno de los recursos del microcontrolador, los cálculos que haya realizado y los valores programados en los registros más significativos.

# SOFTWARE

## Descripción de cada uno de los módulos del sistema

**3.1.1 Módulo joy.c y joy.h**

El módulo joy gestiona las pulsaciones del joystick, identificando la dirección (UP, DOWN, LEFT, RIGHT, MIDDLE) y la duración (corta o larga). La información se envía mediante una cola de mensajes al sistema.

**Funcionamiento**

1. **Inicialización**:
   * **init\_Th\_joy**: Crea el hilo del joystick y una cola de mensajes (id\_MsgQueue\_joy).
   * **init\_joy**: Configura los pines GPIO como interrupciones para las direcciones del joystick.
2. **Hilo del joystick (Th\_joy)**:
   * Procesa las interrupciones y clasifica la pulsación:
     + **Antirrebote (REB)**: Usa un temporizador de 50 ms (tim\_reb) para eliminar falsos positivos.
     + **Duración (CHECK)**: Usa un temporizador periódico (tim\_pul) para medir si la pulsación es corta (<1s) o larga (≥1s).
   * Envía un mensaje a la cola con la dirección y duración detectadas.
3. **Interrupciones**:
   * **EXTI15\_10\_IRQHandler**: Detecta la pulsación y llama a HAL\_GPIO\_EXTI\_Callback.
   * **HAL\_GPIO\_EXTI\_Callback**: Determina la dirección usando comprobarPines y activa el hilo con la bandera IRQ.
4. **Temporizadores**:
   * **tim\_reb**: Antirrebote (50 ms).
   * **tim\_pul**: Mide duración de la pulsación.
5. **Estructuras**

* **MSGQUEUE\_OBJ\_JOY**:
  + dir\_t dir: Dirección detectada.
  + uint8\_t dur: 0 (corta), 1 (larga).

**3.1.2 Módulo hora.c y hora.h**

Este módulo implementa un reloj digital que actualiza y muestra la hora en formato **HH:MM:SS**. Utiliza un hilo (Th\_hora) y un temporizador periódico de 1 segundo para gestionar el conteo de segundos, minutos y horas.

**Funcionamiento**

1. **Inicialización**:
   * **Init\_Th\_hora**: Crea el hilo Th\_hora, encargado de manejar la lógica del reloj.
   * **timer\_seg**: Temporizador periódico que genera una interrupción cada segundo y activa el hilo mediante la bandera SEG.
2. **Hilo del reloj (Th\_hora)**:
   * **Inicio del reloj**: Configura la pantalla LCD para mostrar el tiempo.
   * **Ciclo principal**: Espera la bandera SEG para ejecutar la lógica del reloj:
     + Incrementa los **segundos**, habilitando un incremento de minutos cuando llega a 59.
     + Incrementa los **minutos**, habilitando un incremento de horas cuando llega a 59.
     + Incrementa las **horas**, reiniciándolas a 0 al alcanzar 23.
     + Resetea el reloj si se activa la señal hab\_reset.
   * **Actualización del display**: Muestra la hora actual en la pantalla LCD mediante dataToBuffer.
3. **Estructuras y Variables**

* **Variables globales**:
  + seg, min, hor: Almacenan los valores actuales de segundos, minutos y horas.
  + hab\_min, hab\_hor, hab\_reset: Señales para controlar el fin de cada unidad de tiempo y reiniciar el reloj.
* **Temporizador (timer\_seg)**:
  + Genera una interrupción cada 1 segundo para avanzar el tiempo.
* **Pantalla LCD**:
  + linea1 y linea2: Buffers para el texto que se muestra en el display.

**3.1.3 Módulo ace.c y ace.h**

Este módulo implementa la funcionalidad de lectura de datos de un acelerómetro a través de la interfaz I2C. Proporciona las medidas de aceleración en los ejes **X, Y, Z** y la temperatura del dispositivo, las cuales son enviadas a una cola de mensajes para ser procesadas.

**Funcionamiento**

1. **Inicialización**:
   * **init\_Th\_ace**:
     + Configura la interfaz I2C para la comunicación con el acelerómetro.
     + Crea el hilo Th\_ace, encargado de manejar las lecturas periódicas del dispositivo.
     + Inicializa la cola de mensajes para transmitir los datos.
   * **init\_i2c**:
     + Configura el controlador I2C, asignando la dirección del esclavo (0x68) y limpiando el bus I2C.
2. **Hilo de adquisición (Th\_ace)**:
   * Configura los registros del acelerómetro:
     + Activa el dispositivo a través del registro PWR\_MGMT\_1.
     + Configura el rango del acelerómetro para ±2g mediante el registro ACCEL\_CONFIG.
   * Inicia un bucle infinito donde realiza las siguientes operaciones cada segundo:
     + Envía un comando al acelerómetro para leer los registros de datos.
     + Recibe y procesa los valores crudos de aceleración (X, Y, Z) y temperatura.
     + Convierte los datos crudos en unidades físicas:
       - Aceleración en **g** (división por 16384 para ±2g).
       - Temperatura en grados Celsius.
     + Envía los datos procesados a través de la cola de mensajes.
3. **Temporización**:
   * Utiliza un temporizador (tim\_1seg) para espaciar las lecturas a intervalos de 1 segundo.

**Estructuras y Variables**

* **Cola de mensajes (id\_MsgQueue\_ace)**:
  + Almacena las mediciones procesadas de aceleración y temperatura en una estructura MSGQUEUE\_OBJ\_ACE.
* **Variables del acelerómetro**:
  + accel\_x, accel\_y, accel\_z: Valores crudos de aceleración en los ejes X, Y y Z.
  + temp\_raw: Valor crudo de temperatura.
* **Mensaje (msg\_ace)**:
  + ox, oy, oz: Valores de aceleración convertidos a unidades físicas.
  + temp: Temperatura convertida a grados Celsius.

**Flujo General**

1. **Configuración inicial**:
   * Configuración de la comunicación I2C.
   * Configuración del acelerómetro.
2. **Lectura de datos**:
   * Se solicitan los datos de los registros correspondientes del acelerómetro.
   * Se reciben y convierten los datos crudos en unidades físicas.
3. **Envío de datos**:
   * Los valores de aceleración y temperatura se empaquetan y envían a la cola de mensajes.
4. **Intervalo de lectura**:
   * Se espera 1 segundo antes de realizar la siguiente medición.

**3.1.4 Módulo lcd.c y lcd.h**

Este módulo del LCD implementa una gestión avanzada de un display gráfico, incluyendo funciones para inicializar, resetear, actualizar, y mostrar caracteres en dos líneas de texto. A continuación, un desglose y resumen del código:

**Funciones Principales:**

**1. Inicialización y Reseteo del LCD**

* **LCD\_Reset:**
  + Configura los pines GPIO para la comunicación SPI.
  + Realiza un reset físico del LCD con el pin de reset.
  + Configura el temporizador para los retrasos requeridos en el proceso de inicialización.
* **LCD\_init:**
  + Envía una serie de comandos al LCD para configurarlo:
    - Apaga el display, establece polarizaciones, dirección de memoria, encendido del LCD, etc.
    - Ajusta el contraste del display con LCD\_wr\_cmd(0x81) seguido del valor deseado.

**2. Escritura en el LCD**

* **LCD\_wr\_cmd y LCD\_wr\_data:**
  + Envían comandos o datos al LCD mediante SPI.
  + Manejan los pines de control (A0, CS) para diferenciar entre comandos y datos.

**3. Actualización del LCD**

* **LCD\_update:**
  + Recorre el buffer local (buffer) y actualiza todas las páginas del LCD, que están divididas en cuatro partes (0 a 3).
  + Utiliza el comando LCD\_wr\_data para enviar los datos almacenados en buffer.

**4. Gestión del Buffer**

* **LCD\_symbolToLocalBuffer\_L1 y LCD\_symbolToLocalBuffer\_L2:**
  + Toman un símbolo (carácter ASCII) y lo convierten a su representación gráfica usando la fuente Arial12x12.h.
  + Colocan los datos correspondientes a cada línea en el buffer local, manejando las páginas adecuadas:
    - Línea 1: Páginas 0 y 1.
    - Línea 2: Páginas 2 y 3.
* **LCD\_symbolToLocalBuffer:**
  + Selecciona la línea donde debe escribirse el símbolo (1 o 2) y llama a la función correspondiente.
* **dataToBuffer:**
  + Toma dos cadenas de caracteres (data\_L1 y data\_L2) y las procesa para colocarlas en el buffer.
  + Rellena el resto del buffer con ceros si hay espacio libre después de los caracteres.

1. **5. Hilo del LCD**

* **Th\_lcd:**
  + Espera mensajes en la cola del LCD (id\_MsgQueue\_lcd) que contengan datos para las dos líneas.
  + Cuando recibe un mensaje, llama a dataToBuffer para actualizar el buffer y refrescar el display.

**Estructuras de Datos:**

* **MSGQUEUE\_OBJ\_LCD:**
  + Contiene dos líneas de texto (linea1 y linea2) para mostrar en el LCD.
* **buffer[512]:**
  + Buffer local de 512 bytes que almacena los datos gráficos para el LCD.

**SPI y GPIO:**

* Usa **SPI** para la comunicación con el LCD:
  + Configura SPI como maestro con polaridad CPOL1, fase CPHA1 y MSB primero.
  + Velocidad configurada a 20 MHz.
* Configuración de GPIO:
  + Pines:
    - F13: A0 (control de comandos/datos).
    - A6: Reset.
    - D14: CS (chip select).

**3.1.4 Módulo led.c y led.h**

Este módulo LED está diseñado para gestionar tres LEDs utilizando un hilo de RTOS y utiliza un hilo para procesar señales (flags) que controlan los estados de los LEDs:

**Funciones y Componentes Clave**

**1. Inicialización**

* **led\_init**
  + Configura los pines GPIO necesarios para controlar los LEDs:
    - Pinos utilizados: **PB0, PB7, y PB14**.
    - Configuración:
      * Salida digital (Output Push-Pull).
      * Pull-Up para evitar estados flotantes.
      * Alta velocidad.
    - Inicializa todos los LEDs en estado apagado (GPIO\_PIN\_RESET).
* **init\_Th\_led**
  + Llama a led\_init para inicializar los LEDs.
  + Crea un hilo (id\_Th\_led) que se encargará de gestionar los LEDs basado en los flags recibidos.

**2. Hilo del LED**

* **Th\_led:**
  + Es el hilo principal encargado de gestionar los LEDs.
  + Se mantiene en un bucle infinito esperando señales mediante osThreadFlagsWait.
  + Interpreta las señales para encender o apagar LEDs:
    - **Flags de encendido:**
      * LED1: Enciende PB0 (LED verde).
      * LED2: Enciende PB7 (LED azul).
      * LED3: Enciende PB14 (LED rojo).
    - **Flags de apagado:**
      * nLED1: Apaga PB0 (LED verde).
      * nLED2: Apaga PB7 (LED azul).
      * nLED3: Apaga PB14 (LED rojo)

**Estructuras de Datos**

* **Flags (uint32\_t flags):**

Representan el estado de la flag de los LEDs (encendido/apagado) mediante bits.

**GPIO y Control de LEDs**

**1. Pines Utilizados:**

* PB0: Controla LED1.
* PB7: Controla LED2.
* PB14: Controla LED3.

**2. Encendido/Apagado:**

* Encendido: Los bits de las señales (LED1, LED2, LED3) activan los LEDs correspondientes.
* Apagado: Los bits complementarios (nLED1, nLED2, nLED3) desactivan los LEDs.

**Interacción con el RTOS**

**1. Hilo (Th\_led):**

* Encapsula la lógica de control de los LEDs.
* Maneja múltiples señales en paralelo gracias a osThreadFlagsWait.

**2. Flags:**

* Mecanismo eficiente para encender o apagar LEDs sin necesidad de consultas constantes

**3.1.4 Módulo com.c y com.h**

Este módulo implementa la comunicación serial mediante USART, gestionando la recepción y transmisión de datos con hilos dedicados y colas de mensajes.

1. **Funcionamiento**
2. **Inicialización:**
   * init\_USART:
     + Configura e inicializa el controlador USART3 para la comunicación serial en modo asincrónico.
     + Ajusta los parámetros de transmisión (8 bits de datos, sin paridad, 1 bit de parada) y una velocidad de 115200 baudios.
     + Habilita las líneas de recepción (RX) y transmisión (TX).
   * Init\_MsgQueue\_com:
     + Crea las colas de mensajes id\_MsgQueue\_com\_rx y id\_MsgQueue\_com\_tx, utilizadas para el intercambio de datos entre hilos.
   * init\_Th\_com:
     + Inicia los hilos de recepción (Th\_com\_rx) y transmisión (Th\_com\_tx).
     + Llama a las funciones de inicialización de USART y colas.
3. **Hilo de Recepción (Th\_com\_rx):**
   * Escucha datos entrantes byte por byte a través de USART.
   * Procesa una trama completa según el siguiente flujo:
     + Detecta el inicio de trama (SOH).
     + Almacena el comando (CMD) y la longitud del mensaje (LEN).
     + Recibe el contenido del payload hasta encontrar el final de la trama (EOT).
   * Cuando se recibe una trama válida, la empaqueta en un objeto MSGQUEUE\_OBJ\_COM y la envía a la cola de recepción (id\_MsgQueue\_com\_rx).
4. **Hilo de Transmisión (Th\_com\_tx):**
   * Espera mensajes en la cola de transmisión (id\_MsgQueue\_com\_tx).
   * Envía la trama byte por byte mediante USART, siguiendo la estructura:
     + Inicio de trama (SOH).
     + Comando (CMD).
     + Longitud (LEN).
     + Payload.
     + Fin de trama (EOT).
5. **Estructuras y Variables**

* **Colas de mensajes:**
  + id\_MsgQueue\_com\_rx: Almacena las tramas recibidas.
  + id\_MsgQueue\_com\_tx: Almacena las tramas a enviar.
* **Estructura del mensaje (MSGQUEUE\_OBJ\_COM):**
  + SOH\_type: Byte que indica el inicio de la trama.
  + CMD: Comando recibido o enviado.
  + LEN: Longitud del mensaje.
  + payload[]: Datos de la trama.
  + EOT\_type: Byte que indica el final de la trama.
  + payOK: Indicador de que el payload es válido.
* **Variables:**
  + byte: Buffer para recibir datos byte por byte.
  + cnt: Contador para almacenar y procesar el payload.

1. **Flujo General**
2. **Configuración Inicial:**
   * Configuración del controlador USART y creación de colas de mensajes.
   * Inicio de los hilos dedicados a recepción y transmisión.
3. **Recepción de Datos:**
   * Procesa tramas completas y las envía a la cola de recepción.
4. **Transmisión de Datos:**
   * Envía tramas completas utilizando los datos de la cola de transmisión.
5. **Sincronización:**
   * Uso de flags de hilos (osThreadFlagsSet y osThreadFlagsWait) para gestionar los eventos de recepción y transmisión.
6. **Configuración de USART**

* **Parametros:**
  + Modo: Asincrónico.
  + Bits de datos: 8.
  + Paridad: Ninguna.
  + Bits de parada: 1.
  + Control de flujo: Ninguno.
  + Velocidad: 115200 baudios.
* **Control de líneas:**
  + Habilitación de transmisor y receptor.

1. **Sincronización y Eventos**

* **CallBack\_USART3:**
  + Gestiona los eventos de transmisión y recepción.
  + Activa flags para sincronizar los hilos.

## Descripción global del funcionamiento de la aplicación. Descripción del autómata con el comportamiento del software (si procede)

En este punto debe explicar el funcionamiento completo de su aplicación. Debe aportar detalles de todos los elementos que forman la aplicación y de cómo interaccionan entre sí. Puede utilizar autómatas para describir el funcionamiento del software.

## Descripción de las rutinas más significativas que ha implementado.

En este punto debe enumerar y describir la funcionalidad de las rutinas más importantes que ha implementado.

# DEPURACION Y TEST

## Pruebas realizadas.

**1. LED**

**Objetivo de la Prueba**

* Verificar que los LEDs (LED1, LED2 y LED3) cambian de estado correctamente en función del valor de la variable leds, controlada por un temporizador periódico y las flags enviadas mediante osThreadFlagsSet.

**Proyecto de Keil Utilizado**

* **Proyecto LED\_Test**: Incluye la inicialización del hilo Th\_Thread, el temporizador periódico y la lógica para actualizar el estado de los LEDs.

**Condiciones de Entrada**

1. Inicialización correcta del hilo Th\_Thread y del temporizador osTimer.
2. Ejecución del hilo Th\_Thread, que:

• Incrementa periódicamente la variable leds cada segundo mediante el callback del temporizador.

• Determina el estado de los LEDs según el valor de la variable leds:

* LED1 se activa cuando leds > 3.
* LED2 se activa cuando leds > 8.
* LED3 se activa cuando leds > 14.

• Envía flags a través de osThreadFlagsSet para actualizar el estado de los LEDs.

1. **Resultados Esperados**

• Los LEDs cambian de estado de acuerdo con la siguiente secuencia:

• Cuando leds <= 3: LED1, LED2 y LED3 apagados.

• Cuando 4 ≤ leds <= 8: Solo LED1 encendido.

• Cuando 9 ≤ leds <= 14: LED1 y LED2 encendidos.

• Cuando 15 ≤ leds <= 19: LED1, LED2 y LED3 encendidos.

• Cuando leds = 20: Todos los LEDs apagados, y la secuencia se reinicia.

1. **Resultados Obtenidos**

* **Correcto**: La secuencia de encendido y apagado de los LEDs se ejecuta conforme a lo esperado, y los LEDs responden adecuadamente según el valor de las flags enviadas por el hilo Th\_Thread.

**2. LCD**

**Objetivo de la Prueba**

* Verificar que las líneas de texto enviadas a la cola id\_MsgQueue\_lcd se actualizan y muestran correctamente.

**Proyecto de Keil Utilizado**

* **Proyecto LCD\_Test**: Incluye el módulo de LCD y el hilo de prueba para enviar mensajes con líneas de texto.

**Condiciones de Entrada**

1. Inicialización correcta de la cola id\_MsgQueue\_lcd.
2. Ejecución del hilo de prueba (Th\_Thread) que:
   * Actualiza las líneas de texto con un contador (cnt) cada segundo.
   * Envía las líneas actualizadas a la cola.

**Resultados Esperados**

* La pantalla LCD muestra:
  + Línea 1: " SBM 2024".
  + Línea 2: "prueba: numero X", donde X es un contador que se incrementa cada segundo.

**Resultados Obtenidos**

* **Correcto**: Las líneas de texto se actualizan y se muestran en la pantalla correctamente, con el contador incrementándose cada segundo.

**3. Acelerómetro**

**Objetivo de la Prueba**

* Verificar que los datos enviados a la cola id\_MsgQueue\_ace se reciben correctamente y se procesan sin errores.

**Proyecto de Keil Utilizado**

* **Proyecto ACC\_Test**: Incluye el módulo del acelerómetro y el hilo de prueba para obtener mensajes desde la cola.

**Condiciones de Entrada**

1. Inicialización correcta de la cola id\_MsgQueue\_ace.
2. Datos válidos enviados a la cola desde el módulo del acelerómetro.

**Resultados Esperados**

* Los datos del acelerómetro son leídos correctamente desde la cola y están disponibles para análisis adicional.

**Resultados Obtenidos**

* **Correcto**: Los datos enviados a la cola se reciben correctamente, y el hilo de prueba permite verificar los datos en un punto de interrupción (breakpoint).

**4. Joystick**

**Objetivo de la Prueba**

* Verificar que la dirección (dir) y la duración (dur) de las pulsaciones del joystick se reciben correctamente desde la cola id\_MsgQueue\_joy.

**Proyecto de Keil Utilizado**

* **Proyecto JOY\_Test**: Incluye el módulo del joystick y el hilo de prueba para leer los datos de dirección y duración.

**Condiciones de Entrada**

1. Inicialización correcta de la cola id\_MsgQueue\_joy.
2. Envío de datos válidos de dirección y duración desde el módulo del joystick.

**Resultados Esperados**

* Las variables globales direccion y duracion se actualizan con los valores recibidos desde la cola.

**Resultados Obtenidos**

* **Correcto**: Las variables direccion y duracion se actualizan correctamente con los valores enviados, y se puede verificar el funcionamiento mediante puntos de interrupción (breakpoints).

**5. Hora**

**Objetivo de la Prueba**

* Verificar que las variables globales relacionadas con la hora corresponden al valor correcto.

**Proyecto de Keil Utilizado**

* **Proyecto HORA\_Test**: Incluye el módulo de la hora y permite acceder a las variables globales para su validación.

**Condiciones de Entrada**

1. Las variables de hora son actualizadas correctamente por el módulo correspondiente.

**Resultados Esperados**

* Las variables globales muestran los valores correctos, coincidiendo con los datos introducidos o calculados por el sistema.

**Resultados Obtenidos**

* **Correcto**: Las variables de hora coinciden con los valores esperados.