

Ingeniería de Sistemas Electrónicos

Casa Domótica

*Curso 2022-2023*

*Bloque 2*

**Autores:**

Hao Feng Chen Fu

Álvaro Rodriguez Piñeiro

Jaime Ruiz López

Gonzalo Santa Cruz del Río

Valeriu Petre Stanca

Índice del documento

[1 Objetivos de la PRÁCTICA 2](#_Toc130478891)

[1.1 Resumen de los objetivos de la práctica realizada 2](#_Toc130478892)

[1.2 Acrónimos utilizados 2](#_Toc130478893)

[1.3 Tiempo empleado en la realización de la práctica. 2](#_Toc130478894)

[1.4 Bibliografía utilizada 2](#_Toc130478895)

[1.5 Autoevaluación. 2](#_Toc130478896)

[2 RECURSOS UTILIZADOS DEL MICROCONTROLADOR 3](#_Toc130478897)

[2.1 Diagrama de bloques hardware del sistema. 3](#_Toc130478898)

[2.2 Cálculos realizados y justificación de la solución adoptada. 3](#_Toc130478899)

[3 SOFTWARE 4](#_Toc130478900)

[3.1 Descripción de cada uno de los módulos del sistema 4](#_Toc130478901)

[3.2 Descripción global del funcionamiento de la aplicación. Descripción del autómata y del diagrama con el comportamiento del software 4](#_Toc130478902)

[3.3 Descripción de las rutinas más significativas que ha implementado. 5](#_Toc130478903)

[4 DEPURACION Y TEST 12](#_Toc130478904)

[4.1 Pruebas realizadas. 12](#_Toc130478905)

[5 Otras informaciones sobre la aplicación 13](#_Toc130478906)

[5.1 Capturas de la RTOS 13](#_Toc130478907)

# Objetivos de la PRÁCTICA

T

## Resumen de los objetivos de la práctica realizada

## Acrónimos utilizados

|  |  |
| --- | --- |
|  |  |
|  |  |
| I2C | Inter-Integrated Circuit |
| SPI | Serial Peripherical Interface |
| ISR | Interrupt Service Routine |
| PWM | Pulse Width Modulation |
| LCD | Liquid Cristal Display |
| LED | Light-Emitting Diode |
|  |  |
| ADC | Analog-to-Digital Converter |
|  |  |

## Tiempo empleado en la realización de la práctica.

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

## Bibliografía utilizada

1. Manual de usuario, Carmine Noviello – Mastering STM32

## 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 indícelo.

# RECURSOS UTILIZADOS DEL MICROCONTROLADOR

## Diagrama de bloques hardware del sistema.

## Cálculos realizados y justificación de la solución adoptada.

# SOFTWARE

## Descripción de cada uno de los módulos del sistema

## Descripción global del funcionamiento de la aplicación. Descripción del autómata y del diagrama con el comportamiento del software

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Módulo** | **Funciones** | **Otras funciones que se llama** | **Más detalles** |
| **Principal.c** | Init\_ThPrincipal | osThreadNew() | Inicia el hilo principal. |
| ThPrincipal | osMessageQueueGet()  osMessageQueuePut() | Gestiona las colas de mensajes de diferentes módulos, activa el flag del módulo pwm.c, etc.  Más información: Tabla 2: Entradas y Salidas del Principal |
| **Clock.c** | Init\_ThHora | osThreadNew() | Inicia el hilo del reloj con 00:00:00 |
| ThHora | ----- | Cuenta las horas, minutos y segundos. |
| **Joystick.c** | Init\_Thjoy | osThreadNew() | Inicia el hilo del joystick. |
| Thjoy | osMessageQueuePut()  osThreadFlagsWait()  osTimerStart() | Espera a que se produzca una interrupción del joystick y luego un timer virtual para eliminar los rebotes |
| EXTI15\_10\_IRQHandler | HAL\_GPIO\_EXTI\_IRQHandler() | Habilita las interrupciones de los pines correspondientes |
| HAL\_GPIO\_EXTI\_Callback | osThreadFlagsSet() |  |
| Init\_timer | osTimerNew() | Crea un timer virtual |
| Timer\_Bounce\_Callback | HAL\_GPIO\_ReadPin()  HAL\_GetTick()  osMessageQueuePut() | Distingue el tipo de gesto (Arriba, Derecha, Centro, etc)  Distingue el tipo de pulso (Corto o largo) y añade un mensaje a la cola de mensajes |
| Init\_MsgQueue | osMessageQueueNew() | Inicia una cola de mensajes |
| **Lcd.c** | Init\_ThLCD | LCD\_reset()  LCD\_Init() | Inicia el hilo del lcd.c y configura la comunicación SPI |
| ThLCD | Init\_MsgQueue\_Data()  LCD\_clear()  LCD\_update()  osMessageQueueGet()  Sprintf()  LCD\_symbolToLocalBuffer\_L1()  LCD\_symbolToLocalBuffer\_L2() | Muestra información en el LCD en función del modo de funcionamiento y de los datos que recibe por la cola de mensajes. |
| Init\_MsgQueue\_Data | osMessageQueueNew() | Inicia la cola de mensajes del LCD |
| LCD\_reset | SPI\_Init()  HAL\_GPIO\_Init()  HAL\_GPIO\_WritePin()  Reset\_Signal() | Inicia el SPI, configura los pines de datos, de reset y de CS, y genera una señal de reset |
| LCD\_Init | LCD\_wr\_cmd() | Envía comandos para configurar el LCD |
| LCD\_update | LCD\_wr\_cmd()  LCD\_wr\_data() | Actualiza los pixeles del LCD en función del contenido de un buffer de tamaño 512 |
| LCD\_clear | LCD\_update() | Pone a 0 todos los elementos del buffer de tamaño 512 y actualiza los pixeles del LCD |
| LCD\_symbolToLocalBuffer\_L1 | ------ | Escribe en las páginas 0 y 1 del LCD |
| LCD\_symbolToLocalBuffer\_L2 | ------ | Escribe en las páginas 2 y 3 del LCD |
| **Temperatura.c** | Init\_ThTemp | osThreadNew()  \_\_HAL\_RCC\_I2C1\_CLK\_ENABLE ()  I2C -> Initialize ()  I2C -> PowerControl ()  I2C -> Control (); | Inicia el hilo de la temperatura y configura la comunicación I2C |
| ThTemp | I2C -> MasterReceive ()  I2C->MasterTransmit()  Init\_MsgTemp();  osMessageQueuePut() | Realiza las transacciones de escritura y lectura para obtener el valor de la temperatura.  Se hace una conversión y se envía a la cola de mensajes. |
| I2C\_SignalEvent\_TEMP | osThreadFlagsSet() | Activa un flag del hilo cuando se ha acabado una transferencia I2C. |
| Init\_MsgTemp | osMessageQueueNew() | Inicia la cola de mensajes del Temperatura.c |
| **Rda5807m.c** | Init\_ThRDA | I2C->Initialize ()  I2C->PowerControl ()  I2C->Control ()  I2C->Control () | Inicia el hilo y configura la comunicación I2C |
| ThRDA | Init\_MsgQueue\_RDA()  Init\_MsgQueue\_RDA\_salida()  ***\*Todos los comandos inferiores a esta fila de este módulo\**** | Recibe mensajes del principal y en función de comando realiza una operación.  Tras eso obtiene le valor de la frecuencia y la configuración para enviarlo a la cola de mensajes.  **Más información: Tabla 3: Entradas y Salidas del Principal** |
| Init\_MsgQueue\_RDA\_salida | osMessageQueueNew() | Inicia la cola del RDA5807M |
| I2C\_SignalEvent\_RDA | Escribir\_Comando()  Leer\_Registros()  osThreadFlagsSet() | Cada vez que termina una transacción de lectura o escritura activa el flag del hilo.  Si ha habido una nack, vuelve a realizar la transacción. |
| Escribir\_Comando | Configurar\_Volumen()  I2Cdrv\_2->MasterTransmit()  osThreadFlagsWait() | Realiza una transacción de escritura para configurar el sintonizador |
| Leer\_Registros | I2Cdrv\_2->MasterReceive()  osThreadFlagsWait() | Realiza una transacción de lectura para obtener la información de los registros |
| Iniciar\_Radio | Escribir\_Canal()  Escribir\_Comando() | Enciende la radio y configura para sintonizar el canal 62 |
| Apagar\_Radio | Escribir\_Comando() | Apaga la radio |
| Configurar\_Volumen | ----- | Configura el nivel de volumen de la radio |
| SeekUp | Escribir\_Canal()  Escribir\_Comando()  GetFrecuencyAndChannel() | Realiza un seek up y se obtiene el valor de la frecuencia |
| SeekDown | Escribir\_Canal()  Escribir\_Comando()  GetFrecuencyAndChannel() | Realiza un seek down y se obtiene el valor de la frecuencia |
| Tune100kUp | Escribir\_Comando()  GetFrecuencyAndChannel() | Sintoniza una frecuencia 100 kHz superior al actual |
| Tune100kDown | Escribir\_Comando()  GetFrecuencyAndChannel() | Sintoniza una frecuencia 100 kHz inferior al actual |
| GetFrecuencyAndChannel | Leer\_Registros() | Obtiene el valor de frecuencia y el valor del canal sintonizado |
| AddToBuffer | ----- | Añade una frecuencia/canal al buffer de tamaño 16 |
| ClearMemoryBuffer | ----- | Elimina los elementos del buffer de tamaño 16 |
| MoveToRight | Escribir\_Comando()  GetFrecuencyAndChannel() | Mueve una posición a la derecha en el buffer de memoria |
| MoveToLeft | Escribir\_Comando()  GetFrecuencyAndChannel() | Mueve una posición a la izquierda en el buffer de memoria |
| BarridoFrecuencia | SeekUp()  GetFrecuencyAndChannel()  Escribir\_Comando() | Realiza un barrido de todo el rango de frecuencias y guarda en la memoria las 16 estaciones que captan mejores señales.  Para usarlo se hace con un pulso largo hacia abajo |
| **Pwm.c** | Init\_ThPWM | osThreadNew()  \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE ()  \_\_HAL\_RCC\_TIM2\_CLK\_ENABLE ()  HAL\_GPIO\_Init()  HAL\_TIM\_PWM\_Init()  HAL\_TIM\_PWM\_ConfigChannel | Inicia el hilo del pwm y configura el canal 4 del timer 2 al modo PWM. |
| ThPWM | osThreadFlagsWait()  HAL\_TIM\_PWM\_Start()  HAL\_TIM\_PWM\_Stop() | Espera a que llegue active un flag para generar una señal pwm durante 25 ms |
| **Vol.c** | Init\_PWM\_Pin | osThreadNew() | Inicia el hilo del volumen |
| ThVol | ADC1\_pins\_F429ZI\_config ()  Init\_PWM\_Pin()  ADC\_Init\_Single\_Conversion()  ADC\_getVoltage()  Config\_PWM\_Pulse()  osMessageQueuePut() | Inicia los pines del ADC y del PWM.  Se obtiene un valor analógico de tensión en función del potenciómetro.  Se hace una conversión para que el rango del volumen sea entre 0 y 15. Con eso configura los pulsos PWM.  Mete el valor de volumen en la cola. |
| Init\_PWM\_Pin | \_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE ()  \_\_HAL\_RCC\_TIM4\_CLK\_ENABLE ()  HAL\_GPIO\_Init()  HAL\_TIM\_PWM\_ConfigChannel() | Inicia los pines para el LED rojo del RGB de la mbed. |
| Init\_MsgQueue\_Volumen | osMessageQueueNew() | Inicia la cola del hilo del volumen |
| Config\_PWM\_Pulse | HAL\_TIM\_PWM\_DeInit()  HAL\_TIM\_PWM\_Stop()  HAL\_TIM\_OC\_Init()  HAL\_TIM\_PWM\_Init()  HAL\_TIM\_PWM\_ConfigChannel()  HAL\_TIM\_PWM\_Start() | Configura los pulsos PWM para variar la intensidad de luz emitida del LED RGB. |
| **Adc.c** | ADC1\_pins\_F429ZI\_config | HAL\_GPIO\_Init() | Configura los pines del ADC |
| ADC\_Init\_Single\_Conversion | HAL\_ADC\_Init() | Inicia el ADC para que haga una sola conversión de 12 bits de resolución. |
| ADC\_getVoltage | HAL\_ADC\_ConfigChannel()  HAL\_ADC\_Start()  HAL\_ADC\_GetValue() | Obtiene el valor de la tensión analógica que cae a la salida del potenciómetro. |
| **Com.c** | Init\_ThCOM | osThreadNew()  USART ->Initialize()  USART ->PowerControl()  USART ->Control () | Inicia el hilo para las comunicaciones con el PC y configura la comunicación USART/UART. |
| ThCom | Init\_MsgQueue\_COM()  Init\_MsgQueue\_COM\_Frame()  Sprintf()  USART->Send()  osThreadFlagsWait() | Inicia las 2 colas del módulo.  En función del modo de funcionamiento, envía diferentes Strings al PC.  Si la radio está funcionando muestra el comando que se envía al sintonizador. |
| Init\_MsgQueue\_COM | osMessageQueueNew() | Inicia la cola de los datos del LCD. |
| Init\_MsgQueue\_COM\_Frame | osMessageQueueNew() | Inicia la cola para las tramas de 12 bytes del sintonizador de radio. |
| myUSART\_callback | osThreadFlagsSet() | Cuando la transmisión se completa pone el flag de Com.c a 1 |

Tabla 1: Rutinas de los módulos

**Tabla Principal:**

|  |  |  |
| --- | --- | --- |
| **Eventos/señales** | | |
| **Entradas** | **Acciones** | **Salidas** |
| Horas  Minutos  Segundos | Indica el valor del reloj, esta información se manda al LCD y al terminal del PC. | mid\_MsgLCD\_Data / mid\_MsgCOM |
| joystick.tipo\_gesto | En función del modo de funcionamiento envía mensajes a la RDA5807M o modifica las horas.   * REPOSO * RADIO MANUAL * RADIO MEMORIA * PROGRAMACION DE HORA | mid\_MsgRDA  mid\_MsgLCD\_Data |
| **Mensajes** | | |
| **Entradas** | **Acciones** | **Salidas** |
| mid\_MsgJoy | Lleva los datos del modo de funcionamiento del sistema e indica el tipo de gesto pulsado en el joystick.  Si hay una pulsación el activa el flag del zumbador. | tid\_ThPWM |
| mid\_MsgTemp | Lleva los valores de la temperatura que se van a mostrar en el LCD y el terminal del PC. | mid\_MsgLCD\_Data / mid\_MsgCOM |
| mid\_MsgRDA\_salida | Lleva los valores de la frecuencia, configuración del sintonizador, la posición de memoria e indica si hay una nueva trama de bytes. | mid\_MsgLCD\_Data / mid\_MsgCOM / mid\_MsgCOM\_Frame |
| mid\_MsgVolumen | Lleva el valor del volumen | mid\_MsgLCD\_Data / mid\_MsgCOM / mid\_MsgRDA |

Tabla 2: Entradas y Salidas del Principal

**Tabla RDA5807M:**

|  |  |  |
| --- | --- | --- |
| **Eventos/señales** | | |
| **Entradas** | **Acciones** | **Salidas** |
| data\_RDA | Estructura que contiene el valor del volumen, el comando y el modo de funcionamiento | ----- |
| data\_RDA.volumen | Se usa para configurar el volumen del sintonizador | Variable local: Volumen\_actual |
| data\_RDA.comando | En función del valor de comando realiza una operación u otra:   1. Inicia la radio 2. Realiza un Seek Up 3. Realiza un Sek Down 4. Sintoniza una frecuencia 100 kHz superior 5. Sintoniza una frecuencia 100 kHz inferior 6. Añade una frecuencia al buffer de memoria 7. Mueve en 1 posición a la derecha en el buffer 8. Mueve en 1 posición a la izquierda en el buffer 9. Limpia el buffer 10. Apaga la radio 11. Memorización automática de las estaciones con mejor señal | salida\_RDA.configuracion  salida\_RDA.frecuencia  salida\_RDA.pos\_memoria |

Tabla 3: Entradas y Salidas del RDA580

# DEPURACION Y TEST

## Pruebas realizadas.

**Prueba del reloj:**

**Prueba del LCD:**

**Prueba del Joystick:**

**Prueba de radio:**

**Prueba del zumbador:**

**Prueba del volumen:**

**Prueba del RGB:**

**Prueba de comunicación:**

# Otras informaciones sobre la aplicación

## Capturas de la RTOS

2025

UN TEXTO IMPORTANTE

Lorem ipsum dolor sit amet consectetur adipiscing elit.

Lorem ipsum dolor sit amet consectetur adipiscing elit.