

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](#_Toc133246528)

[1.1 Resumen de los objetivos de la práctica realizada 2](#_Toc133246529)

[1.2 Acrónimos utilizados 2](#_Toc133246530)

[1.3 Tiempo empleado en la realización de la práctica. 2](#_Toc133246531)

[1.4 Bibliografía utilizada 2](#_Toc133246532)

[1.5 Autoevaluación. 3](#_Toc133246533)

[2 RECURSOS UTILIZADOS DEL MICROCONTROLADOR 4](#_Toc133246534)

[2.1 Diagrama de bloques hardware del sistema. 4](#_Toc133246535)

[2.2 Cálculos realizados y justificación de la solución adoptada. 5](#_Toc133246536)

[3 SOFTWARE 6](#_Toc133246537)

[3.1 Descripción de cada uno de los módulos del sistema 6](#_Toc133246538)

[3.2 Descripción global del funcionamiento de la aplicación. Descripción del autómata y del diagrama con el comportamiento del software 7](#_Toc133246539)

[3.3 Descripción de las rutinas más significativas que ha implementado. 8](#_Toc133246540)

[4 DEPURACION Y TEST 17](#_Toc133246541)

[4.1 Pruebas realizadas. 17](#_Toc133246542)

[5 Otras informaciones sobre la aplicación 19](#_Toc133246543)

# Objetivos de la PRÁCTICA

T

## Resumen de los objetivos de la práctica realizada

## Acrónimos utilizados

|  |  |
| --- | --- |
| PIR | Passive InfraRed |
| GPIO | General Purpose Input/Output |
| 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
2. Datasheet, SHT30
3. Datasheet, HC-SR501

## Autoevaluación.

* Identificar en un documento de especificaciones técnicas de un sistema electrónico los requisitos técnicos necesarios para plantear diferentes alternativas tecnológicas para la implementación práctica del mismo.
* Desarrollar un sistema electrónico de mediana complejidad combinando diferentes tecnologías
* Construir sistemas electrónicos utilizando PCBs aplicando las técnicas de diseño adecuadas a la tipología del diseño
* Emplear la instrumentación de laboratorio, las herramientas de desarrollo y depuración comerciales para la integración y puesta a punto de circuitos y sistemas electrónicos.
* Saber generar documentación precisa y transferible sobre el sistema desarrollado.

EXPLICAR ALGUNA DIFICULTAD DEL PROYECTO

# RECURSOS UTILIZADOS DEL MICROCONTROLADOR

## Diagrama de bloques hardware del sistema.

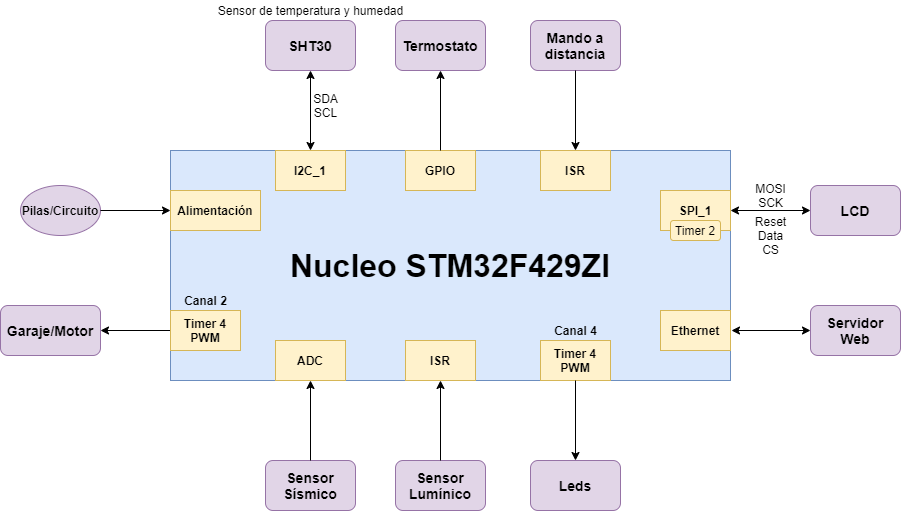


Figura 1: Conexión hardware

Diagrama

Descripción generada automáticamente

Figura 2: Conexiones del CMSIS

|  |  |
| --- | --- |
| Pines STM32F429ZI | Funcionalidad |
| PE11 | Interrupción PIR |
| PD12 | Interrupción mando a distancia |
| PC13 | Interrupción del botón de usuario |
| PC3 | ADC del LDR |
| PD15 | Salida PWM para la iluminación |
| PB8 | Pin SCL para el sensor de temperatura y humedad |
| PB9 | Pin SDA para el sensor de temperatura y humedad |
| PC6 | Salida para encender el ventilador |
| PB4 | Servomotor del ventilador |
| PD13 | Servomotor de las puertas del garaje |
|  |  |

Tabla 1: Conexión de los pines

|  |  |  |
| --- | --- | --- |
| Pines STM32F429ZI | Pines mbed | Funcionalidad |
| + 3,3 V | **DIP40** | Alimentación |
| GND | **DIP1** | Masa |
| PB5 | **DIP5** | MOSI |
| PA5 | **DIP7** | SCK |
| PD14 | **DIP11** | CS |
| PF13 | **DIP8** | A0 |
| PA6 | **DIP6** | RESET |

Tabla 2: Conexión entre la tarjeta NUCLEO STM32F429Zi y la mbed app board

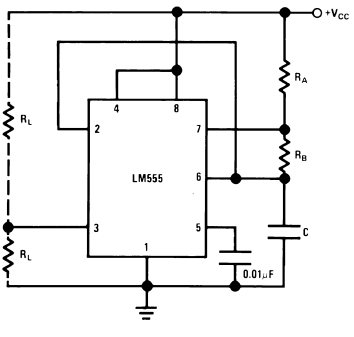
AUMENTAR LA TABLA DE PINES

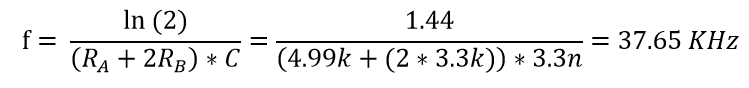
## Cálculos realizados y justificación de la solución adoptada.

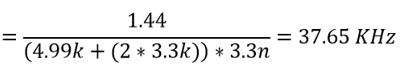
**Módulo del sensor de temperatura y humedad:**

**Módulo del LCD:**

**Módulo del sensor PIR:**

**Módulo del mando a distancia:** Frecuencia de funcionamiento del receptor





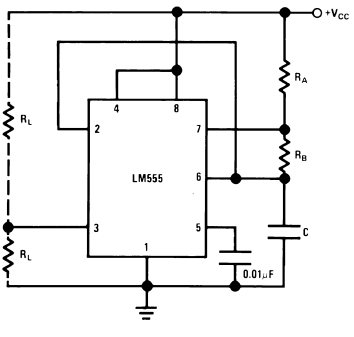
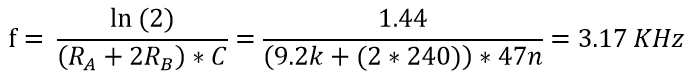
**Módulo del RTC:**

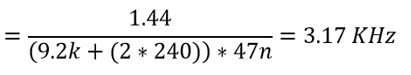
**Módulo del SNTP:**

**Módulo del LDR:**

**Módulo del Termostato:**

**Módulo del Sensor Sísmico:**





# SOFTWARE

## Descripción de cada uno de los módulos del sistema

**Módulo del sensor de temperatura y humedad:**

Este módulo configura el sensor de temperatura y humedad relativa que funciona con el protocolo de comunicación I2C. Se configura en su modo de adquisición periodica para luego leer los datos de los registros de lectura. Tras eso, se realiza una conversión de los datos obtenidos, se envía a una cola de mensajes y espera 1 segundo para volver a leer los registros, esto se realiza sucesivamente.

**Módulo del LCD:**

Este módulo configura inicialmente el protocolo de comunicación SPI para poder usar el LCD, tras esa

configuración inicial se inicializa una cola donde a través de una estructura guardará la información de la temperatura, la humedad relativa, las horas y la fecha. Asimismo, con la pulsación del pulsador azul se muestra en el LCD el consumo de la aplicación.

Tras la captura de los datos en el display mostrará la información obtenida de la cola de mensajes y refrescará cada 1 segundo.

**Módulo del sensor PIR:**

Este módulo gestiona las interrupciones generadas por el sensor PIR. Cuando hay una interrupción se envía por una cola de mensajes si se ha detectado algún movimiento.

**Módulo del mando a distancia:**

Este módulo gestiona la activación de una señal de habilitación para el módulo del garaje. Está señal tendrá dos posibles valores, nivel alto y nivel bajo, dependiendo de si el receptor LED, recibe información del emisor a través de una frecuencia determinada, en nuestro caso, alrededor de 38 KHz.

**Módulo del RTC:**

Este módulo configura el RTC. Cada que pasa 1 segundo aumenta los segundos en una unidad, simulando de esta forma un reloj.

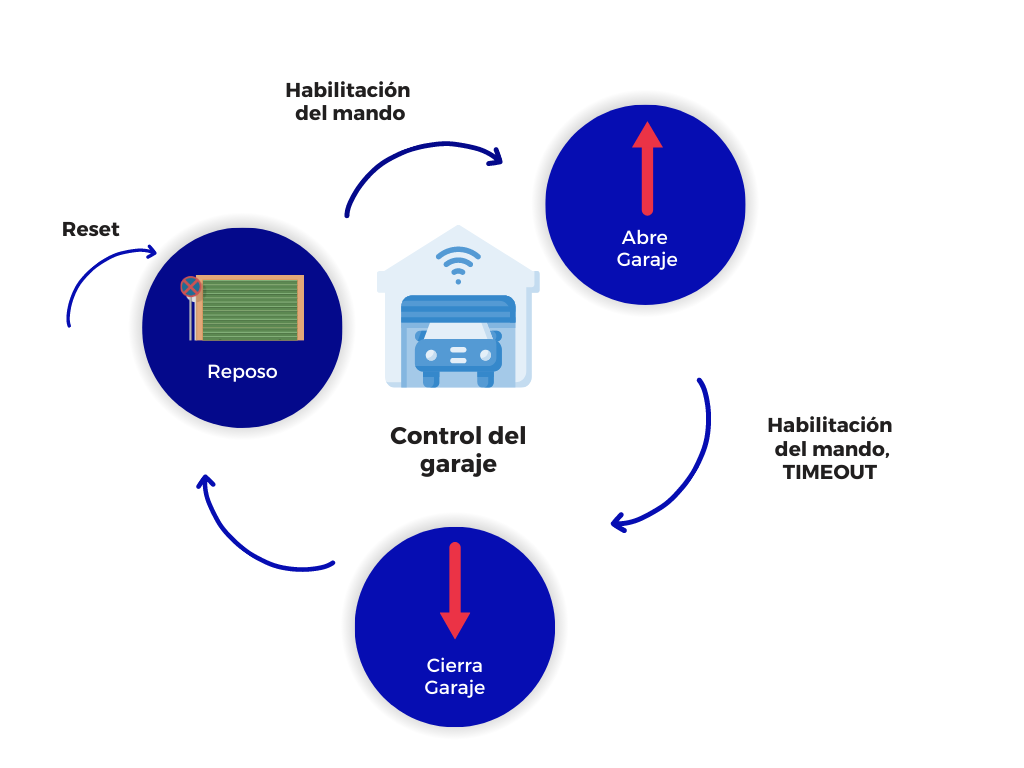
**Módulo del SNTP:**

Este módulo es el encargado de sincronizar el RTC con el tiempo real.

**Módulo del LDR:**

**Módulo del Termostato:**

**Módulo del Garaje:**

Una vez se ha recibido la señal de habilitación del módulo del mando a distancia, este bloque se encargar de manejar con un pwm la activación de un motor que abra o cierra la puerta de un garaje. Por defecto la puerta estará cerrada, ya que en caso de solo detectar una activación por parte del módulo mando a distancia (abrir), se cerrará sola con un cierto TIMEOUT. Después de la primera habilitación del módulo mando a distancia, sucesivas activaciones harán variar el estado de la puerta del garaje entre subir o bajar.

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

Escala de tiempo

Descripción generada automáticamente

ESTE DIAGRAMA ESTA POR MODIFICARSE

EXPLICAR EL PROCESO

## 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.  Más información: Tabla 2: Entradas y Salidas del Principal |
| Init\_All\_Pins | Init\_Pin\_Pulsador()  Init\_PIR\_Pin()  Init\_PWM\_Pin()  Init\_Mando\_Pin()  Falta añadir mas | Inicia todos los pines de la aplicación |
| Init\_All\_Threads | Init\_ThThermostato()  Init\_ThLCD()  Init\_ThPulsador()  Init\_ThRTC()  Init\_ThSNTP()  Init\_ThTemp\_Hum()  Init\_ThPIR()  Init\_ThLDR()  Init\_ThIluminacion()  Init\_ThMando() | Inicia todos los hilos de la aplicación |
| **Lcd.c** | Init\_ThLCD | LCD\_reset()  LCD\_Init() | Inicia el hilo del lcd.c y configura la comunicación SPI |
| ThLCD | Init\_MsgLCD()  LCD\_clear()  LCD\_update()  osMessageQueueGet()  Sprintf()  LCD\_symbolToLocalBuffer\_L1()  LCD\_symbolToLocalBuffer\_L2()  osThreadYield() | Muestra información sobre la temperatura, la humedad relativa, las horas y la fecha en el LCD en función de los datos que recibe por la cola de mensajes. |
| Init\_MsgLCD | 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 |
| **Temp\_Hum.c** | Init\_ThTemp | osThreadNew()  I2C -> Initialize ()  I2C -> PowerControl ()  I2C -> Control (); | Inicia el hilo de la temperatura y humedad, y configura la comunicación I2C |
| ThTemp | Init\_MsgTemp()  Init\_timer\_TempHum()  Config\_Communication()  ReadSensorData()  osMessageQueuePut() | Realiza las transacciones de escritura y lectura para obtener la temperatura y la humedad relativa.  Se hace una conversión y se envía a la cola de mensajes.  Inicia un timer virtual y espera a que se active un flag para volver a obtener valores de tempertura y humedad relativa. |
| I2C\_SignalEvent\_TEMP\_HUM | osThreadFlagsSet() | Activa un flag del hilo cuando se ha acabado una transferencia I2C. |
| Init\_MsgTemp | osMessageQueueNew() | Inicia la cola de mensajes del Temperatura.c |
| Init\_timer\_TempHum | osTimerNew() | Inicia un timer para leer los registros cada 1 segundo |
| Timer\_TempHum\_Callback | osThreadFlagsSet() | Callback del timer virtual, activa el flag 0x2 para indicar que puede volver a leer los registros del sensor. |
| Config\_Communication | Set\_PeriodicAdq\_Mode()  SetSingleShotMode()  I2C -> MasterTransmit ()  osThreadFlagsWait() | Elige el modo de adquisición de datos: 1 es para el modo periodico y 0 para obtener datos 1 vez.  Realiza una transacción de escritura para configurar los regitros del sensor. |
| Set\_PeriodicAdq\_Mode | ------ | Configura los comandos de escritura para que se modifiquen los registros de lectura periodicamente. |
| SetSingleShotMode | ------ | Configura los comandos de escritura para que se modifiquen los registros de lectura una vez. |
| ReadSensorData | I2C -> MasterReceive ()  osThreadFlagsWait() | Lee los registros del sensor, realiza la conversión de los datos y la información que se va a intrudir en la cola de mensajes. |
| **PIR.c** | Init\_PIR\_Pin | \_\_HAL\_RCC\_GPIOE\_CLK\_ENABLE()  HAL\_GPIO\_Init()  HAL\_NVIC\_EnableIRQ() | Inicializa y habilita las interrupciones del PIN E11 tanto la bajada como la subida. |
| Init\_MsgPIR | osMessageQueueNew() | Inicializa la cola de mensajes del PIR |
| EXTI15\_10\_IRQHandler | HAL\_GPIO\_EXTI\_IRQHandler() | Maneja las solicitud de las interrupciones externas del pin 11 |
| HAL\_GPIO\_EXTI\_Callback | HAL\_GPIO\_ReadPin()  osMessageQueuePut() | Vigila si el pin está a nivel alto o bajo después de la interrupción, en función de eso indica si ha habido movimiento o no y lo envía a la cola de mensajes. |
| **SNTP.c** | Init\_ThSNTP | osThreadNew() | Inicia el hilo del SNTP |
| SNTP\_Thread | Init\_Timer\_SNTP()  osTimerStart()  osThreadFlagWait() | Se encarga de sincronizar el RTC cada 5 minutos e inicializa el timer virtual. |
| Init\_Timer\_SNTP | osTimerNew() | Función que inicializa el timer virtual. |
| TimerSNTP\_Callback | Get\_SNTP\_Time()  osThreadFlagsSet() | Callback del timer virtual donde se obtiene el tiempo local que se usa para actualizar el RTC |
| get\_SNTP\_Time | netSNTPc\_GetTime() | Función encargada de enviar una petición SNTP |
| time\_callback | getLocalTime | Callback usado en la función netsSNTPc\_GetTime, si la función ha devuelto un valor no nulo realiza una conversión de segundos al tiempo local. |
| getLocalTime | strftime()  \*localtime()  Set\_RTC\_Time()  Set\_RTC\_Date() | Función encarga de actualizar el tiempo y la fecha del RTC al tiempo actual. |
| **RTC.c** | Init\_ThRTC | osThreadNew()  RTC\_Init() | Inicia el hilo del RTC |
| RTC\_Thread | Init\_timer\_RTC()  Init\_MsgRTC()  osTimerStart()  osThreadFlagsWait() | Inicializa el timer virtual y la cola de mensajes del RTC. También se encarga de actualizar el RTC cada 1 segundo |
| Init\_timer\_RTC | osTimerNew() | Función que inicializa el timer virtual. |
| TimerRTC\_Callback | Get\_Date\_Time()  osMessageQueuePut()  osThreadFlagsSet() | Callback del timer virtual donde se obtiene el tiempo y la fecha del RTC para enviarlo a una cola de mensajes. |
| Init\_MsgRTC | osMessageQueueNew() | Inicia la cola de mensajes del RTC.c |
| Get\_Date\_Time | HAL\_RTC\_GetTime()  HAL\_RTC\_GetDate() | Función que obtiene el tiempo y la fecha (En ese orden). |
| RTC\_Init | \_\_HAL\_RCC\_RTC\_ENABLE()  HAL\_PWR\_EnableBkUpAccess()  \_\_HAL\_RCC\_PWR\_CLK\_ENABLE()  HAL\_NVIC\_EnableIRQ()  HAL\_RTC\_Init() | Inicializa y habilita el RTC |
| Get\_RTC\_Time | HAL\_RTC\_GetTime() | Función que obtiene el tiempo del RTC |
| Get\_Date\_RTC | HAL\_RTC\_GetDate() | Función que obtiene la fecha del RTC |
| Set\_RTC\_Time | HAL\_RTC\_SetTime() | Función que configura el tiempo del RTC |
| Set\_RTC\_Date | HAL\_RTC\_SetDate() | Función que configura la fecha del RTC |
| **LDR.c** | Init\_ThLDR | osThreadNew()  ADC1\_pins\_F429ZI\_config() | Inicia el hilo del LDR y configura los pines del ADC |
| ThLDR | Init\_MsgLDR()  ADC\_getVoltage()  osMessageQueuePut() | Se encarga de inicializar la cola de mensajes y obtiene la tensión a la salida del LDR. Cuando el porcentaje de la tensión actual es diferente al porcentaje anterior se envía el porcentaje del pulso a la cola de mensajes. |
| Init\_MsgLDR | osMessageQueueNew() | Inicia la cola de mensajes del LDR.c |
| **Iluminacion.c** | Init\_ThIluminacion | osThreadNew()  Init\_PWM\_Iluminacion\_Pin() | Inicia el hilo de la Iluminacion y configura los pines |
| ThIluminacion | Init\_MsgIluminacion()  osMessageQueueGet()  Config\_PWM\_Pulse() | Se encarga de inicializar la colade mensajes y en función del dato recibido configura el PWM |
| Init\_PWM\_Iluminacion\_Pin | \_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE()  \_\_HAL\_RCC\_TIM4\_CLK\_ENABLE()  HAL\_GPIO\_Init()  HAL\_TIM\_PWM\_ConfigChannel() | Función que inicializa y habilita el canal 4 del Timer 4 como PWM. |
| 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() | Función que para el pulso del PWM, configura a un nuevo pulso y lo vuelve a iniciar. |
| Init\_MsgIluminacion | osMessageQueueNew() | Inicia la cola de mensajes del Iluminacion.c |
| **ThTermostato.c** | Init\_ThThermostato | osThreadNew() | Inicia el hilo de la ThTermostato.c |
| ThTermostato | Init\_Ventilador\_Msg()  osMessageQueueGet()  encender\_ventilador()  apagar\_ventilador() | Se encarga de inicializar la colade mensajes y en función del dato recibido enciendo o apaga el ventilador. |
| init\_ventilador | \_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE()  HAL\_GPIO\_Init()  HAL\_GPIO\_WritePin() | Función que inicializa el pin del Ventilador. |
| Init\_Ventilador\_Msg | osMessageQueueNew() | Inicia la cola de mensajes del Termostato |
| encender\_ventilador | HAL\_GPIO\_WritePin() |  |
| apagar\_ventilador | HAL\_GPIO\_WritePin() |  |
| **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 3: 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. |  |
| **Mensajes de entrada** | | |
| **Entradas** | **Acciones** | **Salidas** |
| mid\_MsgPIR | Lleva el dato de encender las luces de la casa cuando hay movimiento | mid\_MsgIluminacion |
| mid\_MsgRTC | Lleva los valores de las horas y fechas que se van a mostrar en el LCD y la web | mid\_MsgLCD |
| mid\_MsgLDR | Lleva los valores de la intensisdad de la luz | mid\_MsgIluminacion |
| mid\_MsgTemp\_Hum | Lleva los valores de la temperatura y humedad que se van a mostrar en el LCD y la web | mid\_MsgLCD |
| mid\_MsgPulsador | Cambia el modo del LCD para que muestre el consumo | mid\_MsgLCD |
| mid\_MsgMando | Lleva el dato que abre y cierra la puerta del garaje |  |
|  |  |  |

Tabla 4: 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 5: Entradas y Salidas del RDA580

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

**Nota: Esta tabla se tiene que borrar, se deja por el momento porque es más fácil hacer CopyPaste si se repite algo**

# DEPURACION Y TEST

## Pruebas realizadas.

**Prueba del sensor de temperatura y humedad:**

La prueba del sensor de temperatura y humedad no se necesitan condiciones de entrada.

Esta prueba consiste en verificar el funcionamiento del módulo Temp\_Hum.c, tales como los valores correspondientes de las medidas y el correcto funcionamiento de la cola de mensajes. Asimismo, sólo se necesita un módulo de test para comprobar el funcionamiento de la cola de mensajes y con las herramientas, Debugguer y Debug printf, ofrecidas por el entorno de desarrollo Keil permite observar los printf de forma que se puede comprobar los valores devueltos por el sensor.

Se espera que el módulo Temp\_Hum cuando realice una medida cada 1 segundo para enviar los resultados a la cola de mensajes y con la ayuda de otro módulo, Test, extrae los resultados y los muestra los resultados obtenidos con un printf.

Los resultados obtenidos son los esperados, cuando se obtiene información del sensor esta se envía a la cola de mensajes y el módulo de test extrae la información y la representa en el terminal.

**Prueba del sensor PIR:**

La prueba del sensor PIR necesita condiciones de entrada, esta es la salida del sensor que puede ser un nivel alto o un nivel bajo.

Esta prueba consiste en verificar el funcionamiento del módulo PIR, se requiere de un módulo de leds porque el sensor PIR es la encargada de encender o apagar la iluminación de la casa.

Se espera que cuando haya algún movimiento se debe encender un led, mientras que cuando no haya movimiento el led debe estar apagado.

Los resultados obtenidos son los esperados porque cuando hay movimiento en frente del sensor PIR detecta el movimiento el led se enciende cuando ha dejado de haber moviemiento el led se apaga.

**Prueba del sensor lumínico:**

**Prueba del sensor sísmico:**

**Prueba del LCD:**

La prueba del LCD requiere de los siguientes módulos externos: Temp\_Humc y RTC.c

La prueba consiste en comprobar la representación de la temperatura, la humedad relativa, las horas y la fecha en el LCD y el funcionamiento correcto de la cola de mensajes porque así es como este módulo va a obtener la informacion, por ello, se incluye un módulo Test que es la encargada de enviar la información a la cola de mensajes del módulo LCD. Con la pulsación del pulsador azul pasa a mostrar el valor del consumo.

Se espera que los valores de la temperatura, la humedad relativa, el consumo, las horas y la fecha varien cada segundo (Por lo menos debe variar las horas) en el LCD y que haya ningún problema con las colas de mensajes.

Los resultados son los esperados porque hay cambios en los valores de la información que representa el LCD cada 1 segundo.Además, cuando se pulsa el botón azul muestra el valor del consumo.

**Prueba del mando a distancia:**

**Prueba del termostato:**

**Prueba del Garaje:**

# Otras informaciones sobre la aplicación

2025

UN TEXTO IMPORTANTE

Lorem ipsum dolor sit amet consectetur adipiscing elit.

Lorem ipsum dolor sit amet consectetur adipiscing elit.