Sistemas Basados en Microprocesador

Integración y desarrollo de una aplicación:
Sintonizador de Frecuencia Modulada (FM)

Alumnos:
MARCE GONZALEZ GARCIA
MARCOS MONTOYA ALARCON

Puesto Nº: x

2022-23

Departamento de Ingeniería Telemática y Electrónica
Universidad Politécnica de Madrid

Índice del documento

1 OBJETIVOS DE LA PRÁCTICA

1.1 Resumen de los objetivos de la práctica realizada

- Desarrollar un proyecto a partir del uso del sistema operativo CMSIS RTOSv2 (RTX).
- Construir la aplicación en torno a varios hilos/timers que trabajar simultáneamente y darán soporte a las funcionalidades del sistema.
- Gestionar e interactuar con un circuito sintonizador integrado de radio (RDA5807M), el cual trabaja en FM, conectado a un bus I2C.
- Ser capaz de configurar, inicializar y gestionar otros elementos disponibles de la tarjeta de aplicaciones (mbed App Board):

    - Joystick de 5 gestos para el control de la aplicación, conectado al GPIO.
    - Potenciómetro para el control del volumen, conectado a un canal del ADC.
    - Zumbador conectado a una salida PWM de un Timer
    - Sensor de temperatura LM75B conectado a otro bus I2C.
    - LCD para la visualización de datos conectado mediante SPI.

- Incorporar un canal de comunicaciones serie conectado con el PC con el USART para visualizar tramas mediante la aplicación TeraTerm.

1.2 Acrónimos utilizados

Identifique los acrónimos usados en su documento.

UART	Universal Asynchronous Receiver Transmitter
I2C	Inter-Integrated Circuit
LCD	Liquid Cristal Display
SPI	Serial peripheral interface
GPIO	General Purpose Input/Output
CMD	Command
HAL	Hardware Abstraction Layer
PMW	Pulse width modulation
FM	Frequency Modulation / Frecuencia modulada
ADC	Analog-to-Digital Converter
PC	Personal Computer
LED	Light-Emitting Diode
USB	Universal Serial Bus

1.3 Tiempo empleado en la realización de la práctica.

- **MÓDULO HORA:** 1H
- **MÓDULO LCD:** 3H
- **MÓDULO TEMPERATURA:** 1.5H
- **MÓDULO RDA5807M:** 15H

- **MÓDULO JOYSTICK:** 1.5H
- **MÓDULO ZUMBADOR:** 1H
- **MÓDULO VOLUMEN:** 2H
- **MÓDULO COM-PC:** 12H
- **MÓDULO PRINCIPAL:** 16H

[Tiempo empleado para realizar la práctica]: El tiempo total empleado ha sido de **53** horas.

1.4 Bibliografía utilizada

[RD1] *Diapositivas del Bloque 3*
https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/10486790/mod_resource/content/25/B3_2022.pdf

[RD2] *Enunciado práctica 3*
https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/10486791/mod_resource/content/26/B3_Dise%C3%B1o.pdf

[RD3] NUCLEO-F429ZI-USER MANUAL

[RD4] Esquemático mbed-Application-Board

[RD5] RDA5807M_datasheet_v1.8(2014)

[RD6] LM75B-Digital temperature sensor

[RD7] Ejemplo RDA5807M mbed

[RD8] Ejemplo LM75B mbed

[RD9] https://www.Keil.com

1.5 Autoevaluación.

Consultando la guía de la asignatura, podemos destacar que hemos adquirido entre otros, los siguientes resultados de aprendizaje:

RA735 - Analizar la arquitectura software y hardware de sistemas basados en microcontrolador de mediana complejidad.

RA263 - Integrar la solución de una aplicación en un microprocesador dotado de un sistema operativo empotrado.

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

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.

2 RECURSOS UTILIZADOS DEL MICROCONTROLADOR

2.1 Diagrama de bloques hardware del sistema.

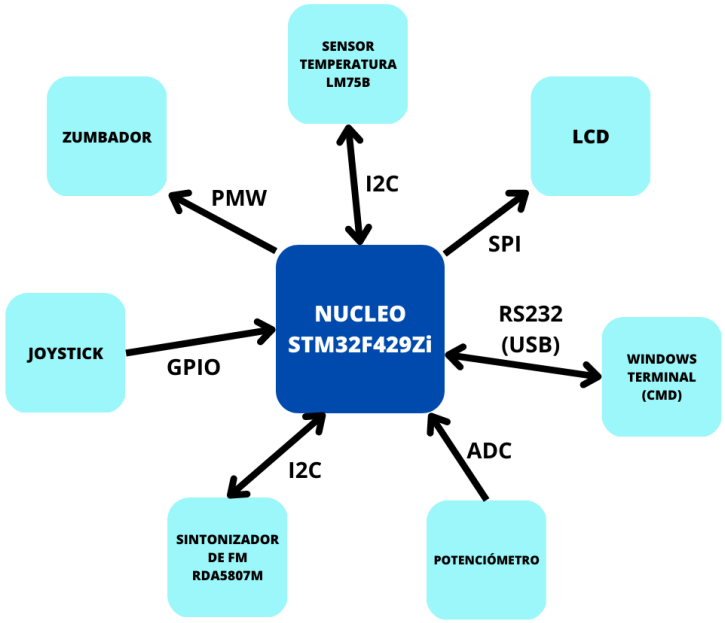


Figura 1. *Diagrama de bloques de los componentes hardware del sistema*

2.2 Cálculos realizados y justificación de la solución adoptada.

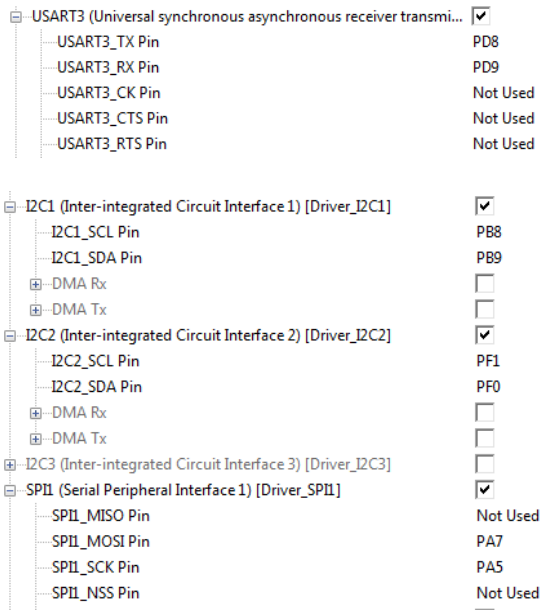
La configuración de los recursos del microcontrolador comienza por el siguiente conexionado de los pines y su configuración en Keil mediante el fichero *RTE_Device.h* :

Tabla 1. Conexiones entre la *mbed app board* y la tarjeta NUCLEO STM32F429Zi

		Pin mbed app board	STM32F429	NUCLEO STM32F429Zi
VCC	+3.3V	DIP40	+3.3V	CN8.7 / +3.3V
GND	GND	DIP1	GND	CN8.11 / GND
LCD (SPI)	MOSI	DIP5	PA7	CN7.14 / D11
	SCK	DIP7	PA5	CN7.10 / D13
	CS	DIP11	PD14	CN7.16 / D10
	A0	DIP8	PF13	CN10.2 / D7
	RESET	DIP6	PA6	CN7.12 / D12
Joystick	Down	DIP12	PE12	CN10.26 / D39
	Left	DIP13	PE14	CN10.28 / D38
	Center	DIP14	PE15	CN10.30 / D37
	Up	DIP15	PB10	CN10.32 / D36
	Right	DIP16	PB11	CN10.34 / D35
Zumbador	PWM	DIP26	PA3	CN9.1 / A0
Temperature (I2C)	SCL	DIP27	PB8	CN7.2 / D15
	SDA	DIP28	PB9	CN7.4 / D14
LED RGB	R	DIP23	PD13	CN10.19 / D28
	G	DIP24	PD12	CN10.21 / D29
	B	DIP25	PD11	CN10.23 / D30
Potenciómetro		DIP20	PC0	CN9.3 / A1

Tabla 2. Conexiones entre el circuito RDA5807M y la tarjeta NUCLEO STM32F429Zi[1]

RDA5807M			STM32F429	NUCLEO STM32F429Zi
Rojo	VCC		+3.3V	CN8.7 / +3.3V
Negro	GND		GND	CN8.11 / GND
Azul	SCL		PF1	CN9.19 / D69
Verde	SDA		PF0	CN9.21 / D68



Posteriormente, en código, programamos los siguientes valores para la configuración de las diferentes entradas GPIO:

```
void GPIO_Init(void){

  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_11|GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;

     //LED RGB
    HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, GPIO_PIN_SET);
     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);

  //Inicializar el pulsador Derecha

    GPIO_InitStruct.Pin = GPIO_PIN_11; //En que pin se encuentra
    GPIO_InitStruct.Mode= GPIO_MODE_IT_RISING; //Se detecta en los flancos de subida
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  //Inicializar el pulsador Izquierda

    GPIO_InitStruct.Pin = GPIO_PIN_14; //En que pin se encuentra
    GPIO_InitStruct.Mode= GPIO_MODE_IT_RISING; //Se detecta en los flancos de subida
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

     //Inicializar el pulsador ARRIBA

    GPIO_InitStruct.Pin = GPIO_PIN_10; //En que pin se encuentra
    GPIO_InitStruct.Mode= GPIO_MODE_IT_RISING; //Se detecta en los flancos de subida
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
    //Inicializar el pulsador ABAJO

  GPIO_InitStruct.Pin = GPIO_PIN_12; //En que pin se encuentra
  GPIO_InitStruct.Mode= GPIO_MODE_IT_RISING; //Se detecta en los flancos de subida
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

    //Inicializar el pulsador CENTRO

  GPIO_InitStruct.Pin = GPIO_PIN_15; //En que pin se encuentra
  GPIO_InitStruct.Mode= GPIO_MODE_IT_RISING; //Se detecta en los flancos de subida
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);//Habilitacion de interrupcion

    //Zumbador y poner un pin en modo pwm
    __HAL_RCC_GPIOA_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_3;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP ;

    //MODO ALTERNATE Y METEMOS EL TIMER PARA CONFIGURAR EL PIN COMO PWM
    GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    __HAL_RCC_TIM2_CLK_ENABLE(); // Habilitamos el reloj del timer
      tim2.Instance = TIM2;
      tim2.Init.Prescaler = 999;  //Un prescaler de 1000
      tim2.Init.Period = 19; //cambiar tempo

      // El channel tenemos que hacelo en toggle para los flancos de subida y bajada
      octim4.OCMode = TIM_OCMODE_TOGGLE;
      HAL_TIM_PWM_ConfigChannel(&tim2,&octim4,TIM_CHANNEL_4);
      HAL_TIM_PWM_Init(&tim2);
```

Módulo HORA (CLOCK)

- Configuramos un Timer periódico de 1 segundo ya que su funcionalidad tiene que ser la de un reloj.

Módulo LCD

- Configuración ***Timer 7*** para la función *Delay (n_microsegundos)*:
  Como el ***Timer 7*** está conectado al APB1 según el manual del STM32F429, entonces nos entran *84MHz*. Ponemos un *Preescaler* de 20 (que en realidad son 21) para *-> 84MHz/21 = 4MHz*. El *Period* lo pondremos tal que *(n_microsegundos/4)-1* para configurar el periodo del timer a partir del parámetro que le pasemos a la función.

- En la función *EscribeFrase()* calculamos la posición necesaria para elegir si escribimos en la Linea 1 *(posición 0)* o en la Linea 2 *(posición 256)*.

Módulo JOYSTICK

- En la gestión del ***Callback** del **Timer 2*** (usado para la gestión de rebotes) lo configuramos a *50ms* solo una vez para que en ese periodo de tiempo compruebe la gestión de rebotes y visualicemos que se estabilice la pulsación.
- El ***Timer 1*** lo establecemos periódico de *50ms* y lo creamos una vez acabe el *Timer2*.
  El cálculo que realizamos fue de comparar si era el timer se realizaba más de *19 veces* (*más los 50ms del* ***Timer 2***) para decretar que fuera una ***Pulsación Larga -> (19+1)veces * 50ms) = 1 seg.***

Módulo ZUMBADOR(PWM)

- Utilizamos el ***Timer2*** (declarado en el main.c) con un *Prescaler* de 999 y un *Period* de 19 que ajustan el sonido del Zumbador. Estos valores se podrían cambiar a otros si quisiéramos cambiar el tipo de sonido a otra frecuencia.

Módulo VOLUMEN

- Para estructurar los niveles de volumen establecimos la siguiente fórmula:

  Nivel de Volumen = (value*4.3)+1;

  En el que el value recogido en tensión, el cual puede ser un máximo de 3.3V lo multiplicamos por 4.3 para que el rango de volumen sea de 0 a 15 y el "*+1*" para que nuestro volumen nunca sea 0.

- En cuanto a la implementación de los LEDs para visulizar los rangos de volumen elegimos de 0 a 5, de 6 a 10 y de 11 a 15, con el propósito de dividirlo en 3 rangos.

Módulo TEMPERATURA

- Establecemos un *OsDelay* de 1 seg dentro del *while(1)* para que mida cada segundo.
- Para calcular la temperatura en grados centígrados (ºC), necesitábamos saber si es negativa o positiva.

  Para esto, utilizamos el bit 10. Si este es 1, la temperatura se calcularía de la siguiente manera:

  temperatura = -(temperatura_bytes+1)*0.125

  (teniendo en cuenta que la temperatura estaría en Complemento a 2)

  Si el bit 10 fuese 0:

  temperatura= temperatura_bytes*0.125

  Y para recoger el valor de la temperatura en los 2 bytes que nos interesan, lo hacemos con el siguiente comando:

  temperatura_bytes = ((temp[0]<<8|temp[1])>>5)

Módulo COM-PC

- Configuramos la velocidad del USART a 9600 Bits/sec, para que se adecue a la del TeraTerm.

3 SOFTWARE

3.1 Descripción de cada uno de los módulos del sistema

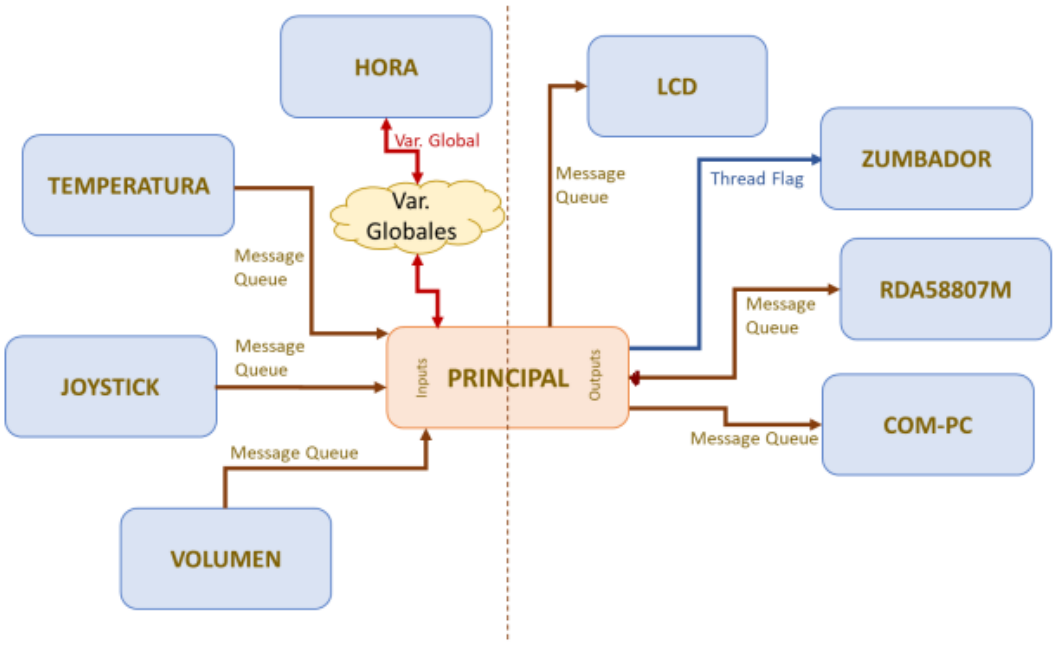


Figura 2. *Estructura de los módulos software que componen la aplicación*

Módulo HORA (CLOCK)
Funcionalidad: Módulo encargado de la gestión del reloj de Sistema.
Entrada: Ninguna.
Salida: Método de sincronización: Variables Globales
Actualiza las variables globales “horas”, “minutos” y “segundos”.
Ficheros: clock.c y clock.h

Módulo TEMPERATURA
Funcionalidad: Módulo encargado de la lectura de la temperatura proporcionada por el sensor LM75B conectado al bus I2C.
Entrada: Ninguna.
Salida: Método de sincronización: Message Queue
Envía un mensaje con la temperatura medida cada Segundo.
Ficheros: clock.c y clock.h

Módulo JOYSTICK
Funcionalidad: Módulo encargado de detectar e identificar una pulsación en el joystick.
Entrada: Ninguna.
Salida: Método de sincronización: Message Queue
Cada vez que se detecta una pulsación se envía un mensaje a una cola de mensajes con información de la tecla pulsada y si la pulsación ha sido normal o larga.
Ficheros: joystick.c y joystick.h

Módulo VOLUMEN

Funcionalidad: Módulo encargado de detectar cambios en el nivel de volumen.

Entrada: Ninguna.

Salida: Método de sincronización: Message Queue

Cada vez que se detecta un cambio de volumen se envía un mensaje a una cola de mensajes con información del nuevo nivel de volumen seleccionado.

Ficheros: vol.c y vol.h

Módulo LCD

Funcionalidad: Módulo encargado de representación de información por el LCD conectado al bus SPI.

Entrada: Método de sincronización: Message Queue

Thread leyendo de cola de mensajes con la información que debe representarse en el LCD y la línea en que debe representarse.

Salida: Ninguna.

Ficheros: lcd.c y lcd.h

Módulo ZUMBADOR

Funcionalidad: Módulo encargado de generar un pitido en el zumbador mediante la generación de una señal PWM.

Entrada: Método de sincronización: Thread Flags

Thread a la espera de un flag que indique que hay que generar un pitido con unos valores de frecuencia, ciclo de trabajo y duración predeterminados.

Salida: Ninguna.

Ficheros: pwm.c y pwm.h

Módulo RDA5807M

Funcionalidad: Módulo encargado de gestionar el funcionamiento del sintonizador conectado al bus I2C.

Entrada: Método de sincronización: Message Queue

Thread leyendo de una cola de mensajes la información del comando que se debe enviar mediante i2c al sintonizador.

Salida: Método de sincronización: Message Queue. Devuelve la frecuencia y la trama de los registros.

Ficheros: rda5807m.c y rda5807m.h

Módulo COM-PC

Funcionalidad: Módulo encargado de la comunicación con el PC a través de la línea serie integrada en el USB.

Entrada: Método de sincronización: Message Queue

Thread leyendo de cola de mensajes con la información que se debe enviar al PC (Teraterm).

Salida: Ninguna.

Ficheros: com.c y com.h

Módulo PRINCIPAL

Funcionalidad: Módulo principal del sistema que se encarga de coordinar todos los demás. Es el módulo que decide las acciones a tomar en función del estado del sistema y de la información que reciba del resto de los módulos.

Entrada: Método de sincronización: Message Queue y Variables Globales

Todas las salidas del resto de los módulos.

Salida: Métodos de sincronización: Message Queue y Thread Flags

Todas las entradas al resto de los módulos.

Ficheros: principal.c y principal.h

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

A continuación, explicaremos como gestiona el módulo principal, el resto de los módulos que componen el sistema:

MODO SELECTOR:
Se usará un selector de modo que se gestiona desde un menú. El usuario interactúa con él a través de pulsaciones izquierdas y derechas del joystick. El display LCD, mostrará la hora actualizada en todo momento, que se inicializa en ***00:00:00*** y el modo al que queramos acceder. La radio se encontrará apagada inicialmente.
Una vez hagamos una pulsación central larga en el lugar que nos encontremos del menú, accederemos al modo que viene mostrado por pantalla (*Figura 3*).

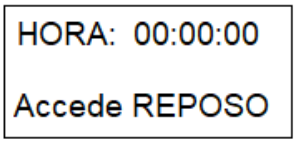


Figura 3.

Por lo tanto, este modo podemos decir que se gestiona gracias a un autómata con la siguiente configuración:

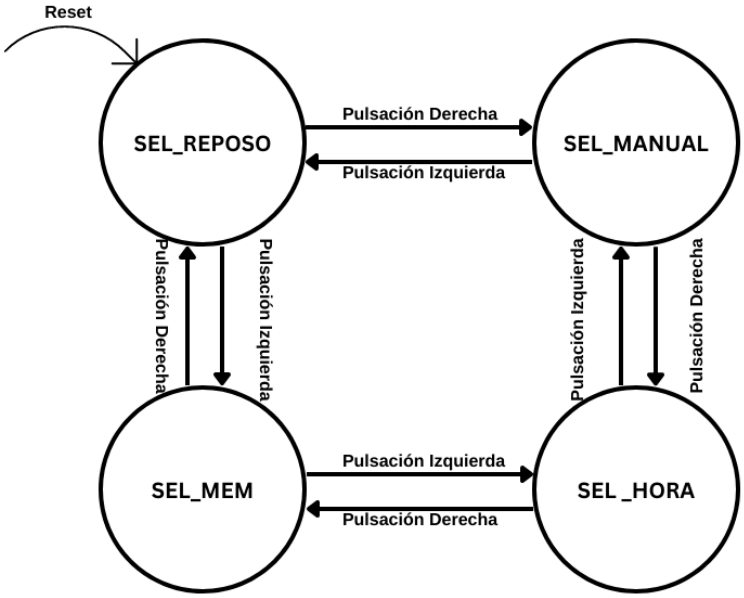


Figura 4.

MODO REPOSO:
En este modo se mostrará el mensaje *"SBM 2022",* la temperatura que será recogida cada segundo y la hora del sistema. En caso de que la radio esté encendida, este modo la apagará. Para salir de este modo necesitaremos una pulsación central larga para así volver al menú de nuevo(*MODO SELECTOR*).

MODO MANUAL:

La radio se activará y estará lista para usarse.

Se gestionará la radio a través de los pulsadores del Joystick de la siguiente manera:

- *Pulsación arriba* : Búsqueda de frecuencias superior con la función *SEEKUP*
- *Pulsación abajo*: Búsqueda de frecuencias inferior con la función *SEEKDOWN*
- *Pulsación derecha*: Sumaremos a nuestra frecuencia actual *+ 0.1*
- *Pulsación izquierda*: Restaremos a nuestra frecuencia actual *– 0.1*
- *Pulsación central larga*: Volveremos al menú de selección de modo.

En todo momento, dentro de este modo, se mostrará la hora, la temperatura, la frecuencia actual y el volumen (gestionado por el potenciómetro).

Cuando salgamos del modo manual, la radio se quedará encendida y podremos navegar por los diferentes modos escuchando la última frecuencia registrada.

MODO MEMORIA:

Podremos almacenar hasta 16 frecuencias diferentes

Utilizaremos para navegar en este modo el joystick de la siguiente manera.

- *Pulsación arriba* : Búsqueda de frecuencias superior con la función SEEKUP
- *Pulsación abajo*: Almacenamiento de la frecuencia actual en la posición que marque la pantalla.
- *Pulsación derecha*: Se selecciona y sintoniza la siguiente frecuencia memorizada
- *Pulsación izquierda*: Se selecciona y sintoniza la anterior frecuencia memorizada
- *Pulsación central larga*: Volveremos al menú

Podremos ver en el LCD la información de la hora, temperatura, frecuencia actual, volumen y posición de almacenamiento.

MODO PROGRAMACIÓN DE HORA:

Se podrá gestionar el reloj del programa modificando las horas, minutos y segundos a través del joystick.

Se ha creado un pequeño autómata para que el usuario tenga una interfaz con la que poder moverse por el cambio de hora sin que sea confuso.

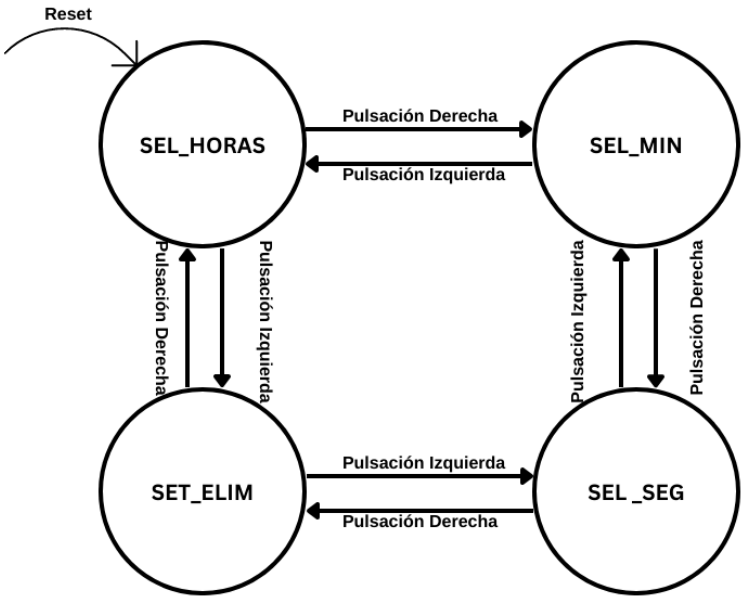


Figura 5.

En cada selección de horas, minutos y segundos, aparecerá en la parte superior de la pantalla la hora actual que vas a establecer. En su parte inferior, el parámetro del reloj que será modificado. Con una pulsación superior o inferior, aumentará o decrementará la variable que se esté actualizando.
Una vez conforme con la hora que se establecerá, con una pulsación larga central, se mostrará un mensaje de información con la hora nueva, una espera de 3 segundos y vuelta al menú inicial.
En la selección de eliminación, mostrará por pantalla un mensaje con la opción de borrar la memoria de las frecuencias almacenadas con una pulsación central larga. Si realizamos la pulsación, se eliminarán todas las memorias que estuvieran almacenadas. Además, actualizará la hora que se estableció en la selección de horas, minutos y segundos. A continuación, se visualizará, un mensaje informando que la memoria de frecuencias ha sido borrada y la hora establecida. Posteriormente, retornará al menú inicial con la radio apagada.

Todos estos modos comparten las siguientes funcionalidades:

En todo momento, el volumen, cuando detecta un cambio en el potenciómetro, se actualizará. Se ha implementado una comunicación visual con el usuario a través de un *LED RGB*. El volumen se encuentra en un rango del 0-15, por lo tanto, se ha establecido el siguiente código de colores: *Verde (0-5), Ámbar (5-10), Rojo (10-15)*. Este LED está constantemente activo ya que, si en algún momento se quiere inicializar la radio, se conozca el volumen inicial.

Todo comando que se le envié a la radio será registrado por la aplicación TeraTerm y mostrado en el cmd del PC. La radio mandará la trama que se encuentra en los registros de escritura y la hora que se han modificado con el siguiente formato:

HH : MM : SS --> XX XX XX XX XX XX XX XX XX XX XX XX.

3.3 Descripción de las rutinas más significativas que ha implementado.

Módulo HORA

static void Timer2_Callback (void const *arg)

Timer periódico que gestiona el funcionamiento del reloj del sistema. Actualizando las variables cada segundo.

Módulo LCD

void Thread_MsgQueue_LCD (void *argument)

Thread que recibe por cola de mensajes la información que debe representarse y la línea en la que debe representarse. Con la información recibida, escribiremos en la pantalla del LCD mediante nuestra función *EscribeFrase(char *frase, uint8_t line).*

Módulo JOYSTICK

static void Timer1_Callback (void const *arg)

Callback del Timer1. Este se encarga de identificar si la pulsación central fue LARGA o CORTA. A partir del conteo de un timer periódico de 50ms, envía mediante la cola del pulsador el resultado obtenido. Rearrancando el propio Timer si el botón siguiera pulsado y deteniéndolo si lo dejasemos de pulsar.

static void Timer2_Callback (void const *arg)

Callback del Timer2. Encargado de la gestión de rebote. Mediante un timer individual de 50 ms, evitando los posibles flancos de subida o bajada en caso de la existencia de rebotes. Envía a la cola del pulsador la identificación de cada uno de los gestos que realicemos con el Joystick.

Módulo TEMPERATURA

void Thread_MsgQueue_Temp (void *argument)

Recoge la temperatura medida cada segundo en una variable, comprobando si es positiva o negativa, y la envía por una cola de mensajes.

Módulo RDA5807M

void Thread_MsgQueue_RDA (void *argument)

Hilo responsabilizado en leer de una cola de mensajes la identificación del comando que queremos enviar al sintonizador, con su función correspondiente. También a partir de este Thread, leemos la trama de cada comando, que luego enviaremos al módulo COM-PC.

Módulo VOLUMEN

void Thread_Vol (void *argument)

Su primera función es recoger en una variable el voltaje del canal ADC. Mediante la fórmula explicada en el apartado 2.2 de este documento la adecuamos a nuestro rango y lo enviamos por la cola. Después, se irá recogiendo y transformando constantemente. Si el valor recogido es diferente al anterior, lo enviaremos por la cola.

Con la implementación extra que incorporamos, dividimos el rango del volumen en 3 y asignamos a cada tercio el encendido de un LED diferente.

Módulo ZUMBADOR

void Thread_Zumbador (void *argument)

Espera al flag para que comience su funcionamiento (pitido) y posteriormente, arrancha el timer2 en el channel 4. Para la generación de una señal PWM, se establece un nivel alto durante 250ms(tiempo que dormimos el hilo) y a continuación, un nivel bajo en la señal.

Módulo COM-PC

void Thread_COM (void *argument)

Hilo que gestiona la comunicación con el TeraTerm. Lee de la cola de mensajes, la información con la hora del sistema y la trama generada por el sintonizador. La convierte en un array de caracteres, y la envía al TeraTerm a través del driver del USART, previamente inicializado.

Módulo PRINCIPAL

void Thread_Principal (void *argument)

Thread que asume la coordinación del resto de módulos. Decide las acciones a realizar en función del estado y la información que vaya recibiendo desde las variables globales y las colas de mensajes. Con las identificaciones de los gestos del Joystick, va cambiando el modo del sistema.

4 DEPURACION Y TEST

4.1 **Pruebas realizadas.**

Módulo HORA

- ✓ La primera prueba que realizamos después de codificar el fichero *clock.c* fue comprobar mediante las ventanas Watch, visualizar que las variables “horas”, “minutos” y “segundos” incrementaran de manera correcta cada segundo. El resultado obtenido fue casi el buscado a excepción de que se nos había olvidado establecer la variable "horas" a 0 después de que se pasara de 23.
- ✓ Corregimos este pequeño error y volvimos a realizar la prueba anterior con un resultado exitoso.

Módulo LCD

- ✓ Al tener ya todas las funciones del *lcd.c* codificadas y testeadas de prácticas anteriores, simplemente probamos que el Thread leyera unas frases y la elección de la línea, que le pasábamos por la cola y los mostrase por pantalla como correspondía. El resultado fue el esperado.
  Luego comprobamos que se representara correctamente el módulo HORA, con esto tuvimos un problema el cual era que cuando alguna de las variables tenía el valor de una sola cifra se mostraba esa sola cifra en vez de un 0 delante. EJEMPLO: ***2:15:4*** en vez de ***02:15:04.***
  Esto lo solucionamos corregiendo los *"%"* que introducíamos en la función *sprintf(),* consiguiendo un éxito en esta última prueba.

Módulo JOYSTICK

- ✓ En este caso usamos tanto el módulo LCD como la visualización en ventana Watch. Creamos unos contadores para cada pulsación que fue lo que fuimos visualizando en el Watch para ver que no hubiera rebote, mientras que el LCD mostraba la pulsación actual. El resultado fue óptimo.

Módulo TEMPERATURA

- ✓ Al igual que el módulo anterior, nos hemos apoyado tanto en el módulo LCD como en la ventana Watch. Con esta última visualizábamos la variable "temperatura" y simultáneamente lo enviábamos al LCD. Comprobamos también los aumentos y disminuciones de la temperatura acercando el dedo al sensor. Los resultados fueron los esperados.

Módulo RDA5807M

- ✓ Tras cometer varios errores en la implementación del módulo por cuestiones de no entendimiento total del funcionamiento del chip y obtener resultados nada satisfactorios, gracias a la ayuda del profesor, conseguimos solucionarlos. A partir de aquí, utilizamos el módulo JOYSTICK para, con las distintas pulsaciones, probar las funcionalidades del sintonizador. Los resultados de estas pruebas fueron mayoritariamente bien. El único fallo que teníamos es que recibíamos un ruido blanco constante de desconocida procedencia. Varias pruebas después, identificamos que era un problema de rendimiento y lo solucionamos. Consiguiendo un perfecto funcionamiento final.

Módulo VOLUMEN

- ✓ Utilizando tanto la visualización de variables como el módulo del RDA, tras la codificación, realizamos las siguientes pruebas:
  Mientras observábamos la variable en el Watch, con los auriculares puestos y la radio en funcionamiento, fuimos variando el potenciómetro en todo el rango de valores para ver si se adecuaba a lo que esperamos. Obtuvimos unos buenos resulltados en la prueba.

Módulo ZUMBADOR

- ✓ A diferencia de otros, en este módulo, no vimos necesario el soporte de otro módulo. Establecimos una primera frecuencia a la señal PWM, con un acierto en el ensayo.
- ✓ A partir de aquí, realizamos un estudio con diversas frecuencias, hasta obtener un sonido que nos gustase.

Módulo COM-PC

- ✓ Después de codificar *com.c* , primero probamos a mandar una variable privada de array de caracteres con un mensaje que establecimos nosotros, en el propio fichero y comprobar que la configuración estaba correcta y que mostraba en el cmd.
- ✓ A continuación, intentamos pasar las variables del módulo HORA por colas de mensaje y mostrarlas en el cmd, también con un resultado esperado.
- ✓ Por último, con lo que más inconvenientes nos surgieron fue a la hora de recoger la trama y enviarla. Tras varias pruebas e intentos con resultados nada positivos, ya que primero la pasábamos en decimal, luego datos que no eran, etc. Conseguimos recoger correctamente la trama junto con las órdenes que realizábamos en el sintonizador. El resultado que buscábamos era que inicialmente nos saliera la trama que los profesores nos habían facilitado como ejemplo en el enunciado, el cual correspondía a la inicialización del sistema. El test acabo siendo exitoso. A partir de aquí, observábamos que registros y bytes iban cambiando dependiendo de la orden.

Módulo PRINCIPAL

La implementación de este módulo fue la más compleja ya que son casi 1000 líneas de código. Por lo que, tuvimos que realizar gran cantidad de pruebas para comprobar el funcionamiento y correcta interacción de cada pequeña parte del modulo con el resto de módulos. A la que íbamos obteniendo tanto resultados positivos como negativos, fuimos cambiando y perfeccionando el código hasta la versión definitiva. Las pruebas más significativas fueron las siguientes:

- ✓ Comprobación del sincronizado recibimiento de variables globales procedentes del módulo HORA.
- ✓ Verificación de la correcta entrada de colas de mensajes de mediciones provenientes de los diferentes módulos como por ejemplo el volumen o la temperatura.
- ✓ Identificación de la pulsaciones recogidas en el módulo JOYSTICK y posteriormente, control del cambio de modos.
- ✓ Envío de colas de mensajes a los diferentes módulos con la información necesaria para la gestión de estos.