**Cuestión (20 minutos – 2 puntos)**

Se tiene una arquitectura Harvard, con capacidad de 32Mpalabras de programa y 16MB de datos en palabras de 16 bits, con una capacidad de direccionar únicamente palabras. Dicha arquitectura contiene 16 registros internos propósito general y filosofía Load & Store. Con esta información, conteste a las siguientes preguntas:

1) (70%) Diseñe una codificación de los distintos tipos de instrucciones microprocesador, minimizando en lo posible el tamaño de la instrucción ajustando a números enteros de palabras. Los tipos de instrucciones que debe tener el microprocesador, son:

- 7 instrucciones de transferencia de datos entre memoria y registros internos, con direccionamiento directo.
- 5 instrucciones de transferencia de datos entre memoria y registros internos, con direccionamiento indexado.
- 14 instrucciones aritmético/lógicas de operar entre registros, siendo el resultado el mismo registro que uno de los operandos.
- 14 instrucciones aritmético/lógicas de operar entre registros, con uno de los operandos dado por direccionamiento inmediato, y dando el resultado en el otro operando.
- 6 instrucciones de control con direccionamiento inherente.
- 7 saltos condicionales con direccionamiento relativo a contador de programa, siendo el desplazamiento relativo de más/menos 1Mpalabra.

2) (30%) Indique una respuesta justificada a cada una de las siguientes preguntas:

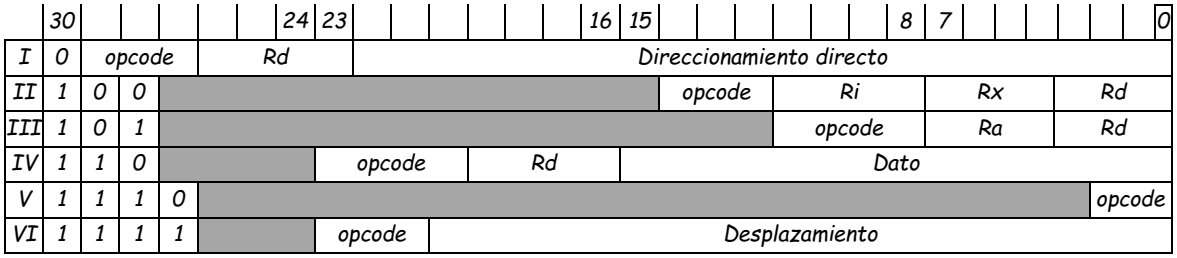
a) Tipos de buses internos, y tamaño de cada uno de ellos
b) Tamaño del Registro de Instrucción
c) Tamaño del Contador de Programa
d) Tamaño de los Registros internos

**SOLUCIÓN:**

*Al ser una arquitectura Harvard el sistema tiene buses para datos y buses para programa. En concreto, para los datos se tiene un bus de datos de 16 bits, y un bus de direcciones de 23 bits, ya que las palabras son de 16 bits. Para el programa, el bus de direcciones será de 25 bits, y el bus de datos dependerá del tamaño de la instrucción, que se verá después de la codificación de las mismas.*

*Además tenemos 16 registros de propósito general, por lo que los registros se referencian con 4 bits. Por último, en el caso del desplazamiento relativo de 1MB, el desplazamiento se dará con 20 bits, pero con la necesidad de incluir 1 bit más para dar el bit de signo (en realidad, complemento a 2).*

*Con todo esto, la codificación de instrucciones podría ser:*




2) (30%) Indique una respuesta justificada a cada una de las siguientes preguntas:

a) Tipos de buses internos, y tamaño de cada uno de ellos

*Tal y como se puede ver del apartado anterior, hay 4 buses.*

*Para datos: bus de direcciones de 23 bits y bus de datos de 16 bits*

*Para programa: bus de direcciones de 25 bits y de datos de 31 bits*

b) Tamaño del Registro de Instrucción

*El registro de instrucción tendrá que ocupar lo que la instrucción más grande, y por tanto deberían ser 31 bits.*

c) Tamaño del Contador de Programa

*El contador de programa utilizará el mismo número de bits que el bus de direcciones, es decir, 25 bits.*

d) Tamaño de los Registros internos

*Los registros internos son de propósito general, por lo que se utilizan para datos (16 bits) y para direcciones de los datos (23 bits). Por tanto los registros internos tendrán el máximo de los dos, es decir, 23 bits.*

**Problema 1 (60 minutos – 4 puntos)**

Para evitar la aglomeración de vehículos en un aparcamiento público, se va a desarrollar un sistema básico de control de acceso basado en un microcontrolador que ha utilizado durante el curso. A continuación se muestra el esquema del sistema y se describen las funcionalidades requeridas.

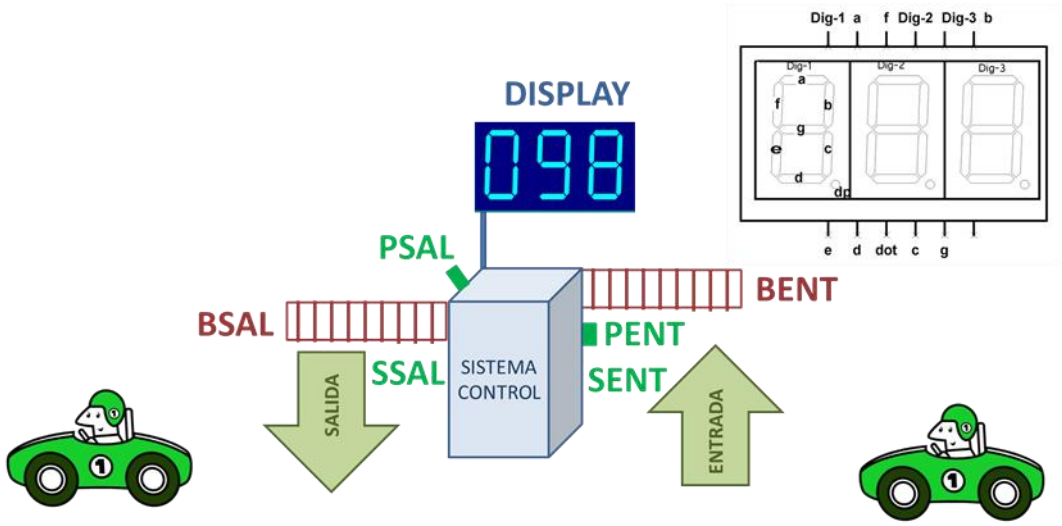



Figura 1. Esquema del sistema de control de acceso al aparcamiento y detalle de los pines del display

Cuando el sistema empieza a funcionar el aparcamiento está vacío y supondremos que cada vehículo que entra ocupa una única plaza de aparcamiento.

En la entrada, el sistema dispone de un display de siete segmentos de cátodo común de 3 dígitos (DISPLAY), en el que se representa el número de plazas libres en el aparcamiento en cada instante.

En la entrada, también hay un pulsador (PENT) para activar la elevación de la barrera de entrada (BENT). Cuando se pulsa PENT se debe levantar la barrera de entrada durante un tiempo, TBENT (supondremos que en ese tiempo entra un solo vehículo), en el caso de que haya plazas libres en el aparcamiento. Si algún conductor pulsa el botón de entrada con el aparcamiento completo la barrera permanecerá bajada y el conductor tendrá que volver a pulsar PENT cuando haya alguna plaza libre para que la barrera se eleve.

En la salida hay un pulsador de salida (PSAL). Cuando se pulsa, se eleva la barrera de salida (BSAL) durante un tiempo, TBSAL, en el cuál supondremos que sale un solo vehículo.

Ambas barreras se elevan cuando se pone un 1 lógico en el punto de conexión de la barrera al sistema de control (cuando en este punto de conexión hay un 0 lógico la barrera permanece bajada).

El sistema dispone también de dos sensores (uno en la entrada, SENT, y otro a la salida, SSAL) que permiten detectar si hay un obstáculo para mantener la barrera correspondiente elevada si el coche no la ha atravesado en el tiempo dado. La tensión de salida de estos sensores puede variar entre 0 y 5V. Una tensión de salida del sensor mayor que 1,3V indica que hay un obstáculo y la barrera correspondiente no debe cerrarse hasta que no desaparezca dicho obstáculo.

El código desarrollado para la aplicación es el que se muestra al final de este enunciado.

**DATOS: Se configura el oscilador del microcontrolador con $F_{osc}$ = 8MHz. El micro está alimentado entre 0 y 5V**

**Se pide:**

1. Represente el esquema del hardware para esta aplicación. Dibuje el microcontrolador como un bloque, indicando claramente qué pines del microcontrolador se conectan a cada una de las entradas o salidas de todos los elementos que componen el sistema (cada elemento debe estar claramente identificado en el diagrama con el mismo nombre que se les ha asignado en la figura 1). Además, rellene la siguiente tabla. (20%)

| PIN MICRO | HW CONECTADO | TIPO (E/S, DIGITAL/ ANALOGICO) | JUSTIFICACIÓN |
|---|---|---|---|
| | | | |

2. Represente el diagrama de flujo del programa principal y de la/s rutina/s de atención a la interrupción, incluyendo una descripción de lo que hace cada función que aparece en el código. (30%)

3. ¿Con qué recurso del microcontrolador se controla el tiempo de elevación de la barrera de salida y durante cuánto tiempo se mantiene elevada la barrera de salida (TBSAL) en el caso de que el sensor de salida no detecte ningún obstáculo? Justifique su respuesta incluyendo todos los cálculos necesarios para obtenerla. (15%)

4. ¿Cuál es número total de plazas que tiene disponible el aparcamiento según el código y cuál sería el máximo número de plazas que se podrían gestionar con el código suministrado? Razone su respuesta (10%)

5. Indique la configuración del ADC utilizada y cuál es el tiempo de conversión y el tiempo entre conversiones. Indique la modificación necesaria para que las conversiones se hagan más frecuentemente. (15%)

6. En el código falta una línea de código en el lugar señalado en el mismo. Escriba a continuación la línea de código que falta. Justifique su respuesta. (10%)

## CÓDIGO PROGRAMA CONTROL APARCAMIENTO

```
#include "Biblioteca_SDM.h"
#include "stm32l1xx.h"

unsigned int tiempo_BENT_up = 0;
unsigned int tiempo_BSAL_up = 0;
unsigned char barrera_entrada_subida = 0;
unsigned char barrera_salida_subida = 0;
unsigned char obstaculo_ENT = 0;
unsigned char obstaculo_SAL = 0;
unsigned char num_vehiculos=0;

void setup(void);
void representar(void); // ESTA FUNCIÓN REPRESENTA EN EL DISPLAY EL Nº DE VEHÍCULOS EN
                        // EL APARCAMIENTO (NO SE INCLUYE SU CÓDIGO)

void main(void){
   setup();
   while(1){
      representar();
      if ((EXTI->PR & (1<<0)) != 0) {
         EXTI->PR |= 1<<0;              // Limpia flag PENT
         if ((obstaculo_ENT == 0) && (barrera_entrada_subida == 0) && (num_vehiculos < 152)){
            barrera_entrada_subida = 1;
            GPIOB->BSRR |= 1<<3;        // Abre la BENT
            tiempo_BENT_up = 0;
            num_vehiculos++;
            }
         }
      if ((EXTI->PR & (1<<1)) != 0) {    // PSAL pulsado?
         EXTI->PR |= 1<<1;              // Limpia flag PSAL
         if ((obstaculo_SAL == 0) && (barrera_salida_subida == 0)){
            barrera_salida_subida = 1;
            GPIOB->BSRR |= 1<<2;        // Abre la BENT
            tiempo_BSAL_up = 0;
            num_vehiculos--;
            }
         }
      if ((barrera_entrada_subida == 1)  && (obstaculo_ENT ==0) && (tiempo_BENT_up > 20)){
         barrera_entrada_subida = 0;
         GPIOB->BSRR |= (1<<3)<<16;        // Cierra la BENT
         }
      if ((barrera_salida_subida == 1) && (obstaculo_SAL==0) && (tiempo_BSAL_up > 20)){
         barrera_salida_subida = 0;
         GPIOB->BSRR |= (1<<2)<<16;        // Cierra la BSAL
         }
      }
   }

void setup(void){
   // GPIOA: PA1 y PA5 como analógicos
   GPIOA->MODER |= 0x03 << (1*2);
   GPIOA->MODER |= 0x03 << (5*2);
   // GPIOA: PA6 - PA15 como salida digital
   GPIOA->MODER &= ~(0xFFFFF000);
   GPIOA->MODER |= 0x55555000;
   // GPIOB: PB0 y PB1 como entrada digital
   GPIOB->MODER &= 0xFFFFFF00;
   // GPIOB: PB2 y PB3 como salida digital
   GPIOB->MODER |= 0x05 << (2*2);
   // EXTI0 y EXTI1 a GPIOB
   SYSCFG->EXTICR[0] = 0x0011;
   // EXTI0 y EXTI1 habilitadas y por flanco de subida
   EXTI->IMR |= 0x03;
   EXTI->RTSR |= 0x03;
   EXTI->FTSR &= ~(0x03);
   // ADC con conversión simple de canales AIN1 y AIN5 alternativamente
   ADC->CR1 = 0x02000020; // 8 bits, sin SCAN y habilitación de IRQ por fin de conversion
   ADC->CR2 = 0x00000401; // No iniciado, alineado a derecha, IRQ con cada fin de conversión
individual, conversión simple, ADC encendido.
   ADC->SQR1 = 0x00100000;  // 2 elementos en la secuencia
   ADC->SQR5 = 0x01;    // canal AIN1
```

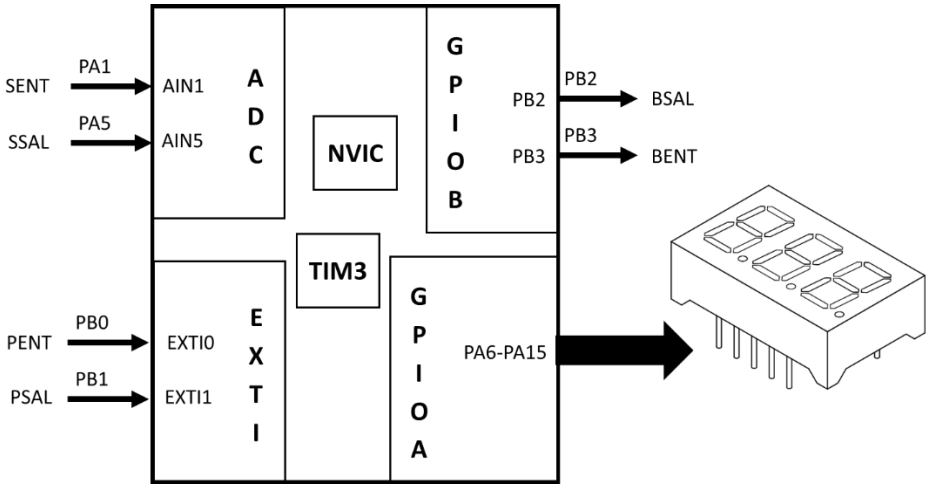
```
   // TIM3 CH1 por TOC para que cuente 1 segundo
   TIM3->CR2 = 0;
   TIM3->SMCR = 0;
   TIM3->CCMR1 = 0x0000;
   TIM3->CCER = 0;
   TIM3->CNT = 0;
   TIM3->PSC = 7999;
   TIM3->ARR = 0xFFFF;
   TIM3->CCR1 = 1000;
   // TIM3 por interrupciones
   TIM3->DIER = 0x0002;
   // Arranca TIM3
   TIM3->CR1 = 0x0001;
   // NVIC activar ADC y TIM3
   NVIC->ISER[0] |= 1 << 18;
   NVIC->ISER[0] |= 1 << 29;
   }

void TIM3_IRQHandler(void) {
   TIM3->SR &= ~(0x1 << 1);   // Limpia el flag
   tiempo_BENT_up++;
   tiempo_BSAL_up++;
   TIM3->CCR1 = 1000;         // Vuelve a contar 1 seg
   ADC->CR2 |= 1<<30;         // Arranca conversión
   }

void ADC1_IRQHandler(void) {
   ADC->SR &= ~(0x1 << 1);    // Limpia el flag
   if ((ADC->SQR5 & 0x001F)==1) {
      ADC->SQR5 = 0x05;       // canal AIN5
      if (ADC->DR > 67) obstaculo_ENT = 1;
      else obstaculo_ENT = 0;
      }
   else if ((ADC->SQR5 & 0x001F)==5) {
      ADC->SQR5 = 0x01;       // canal AIN1
      if (ADC->DR > 67) obstaculo_SAL = 1;
      else obstaculo_SAL = 0;
      }
   }
```

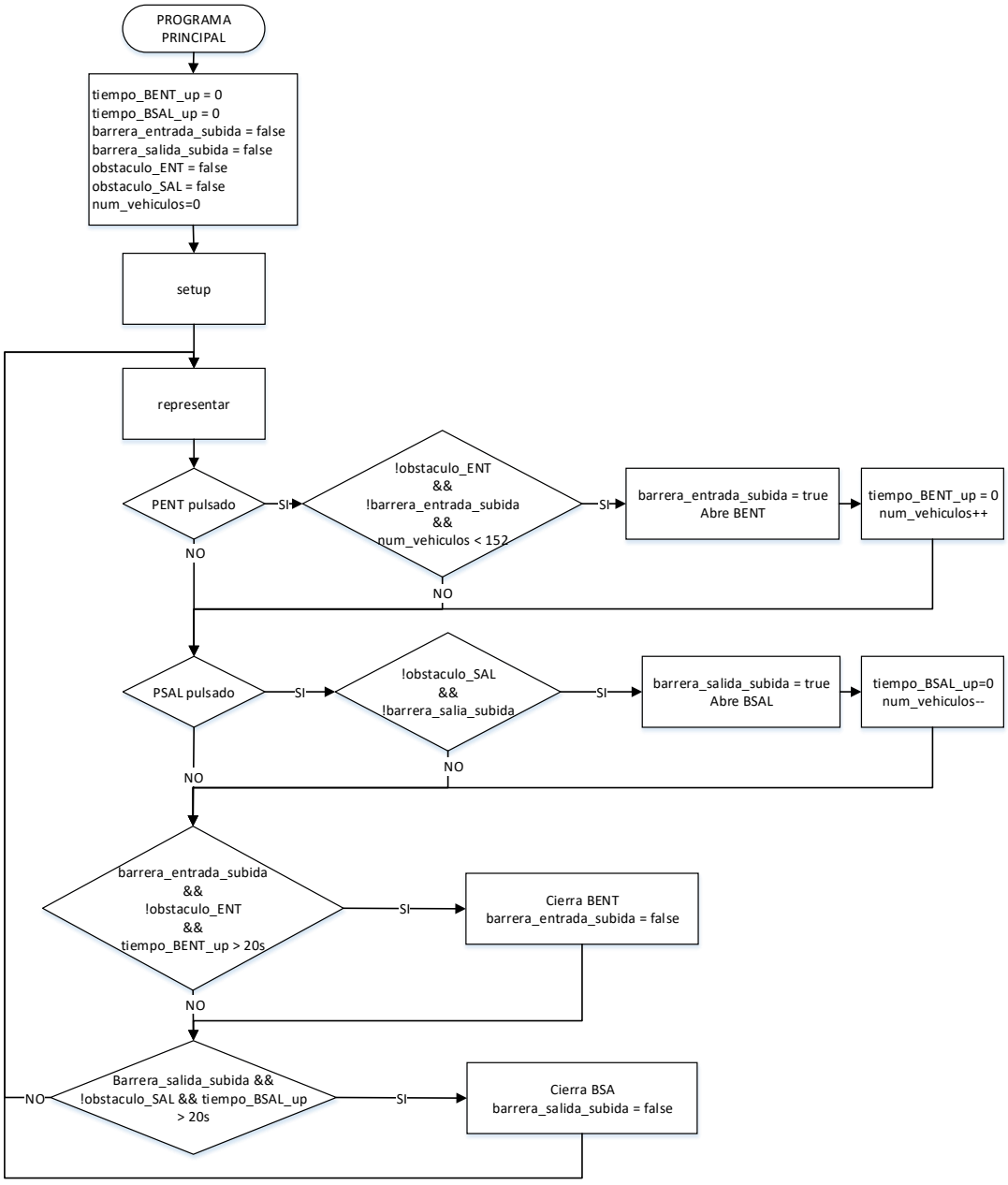
**SOLUCIÓN:**

*1. Atendiendo a los datos del ejercicio, y al código analizado, el diagrama de bloques es el siguiente:*

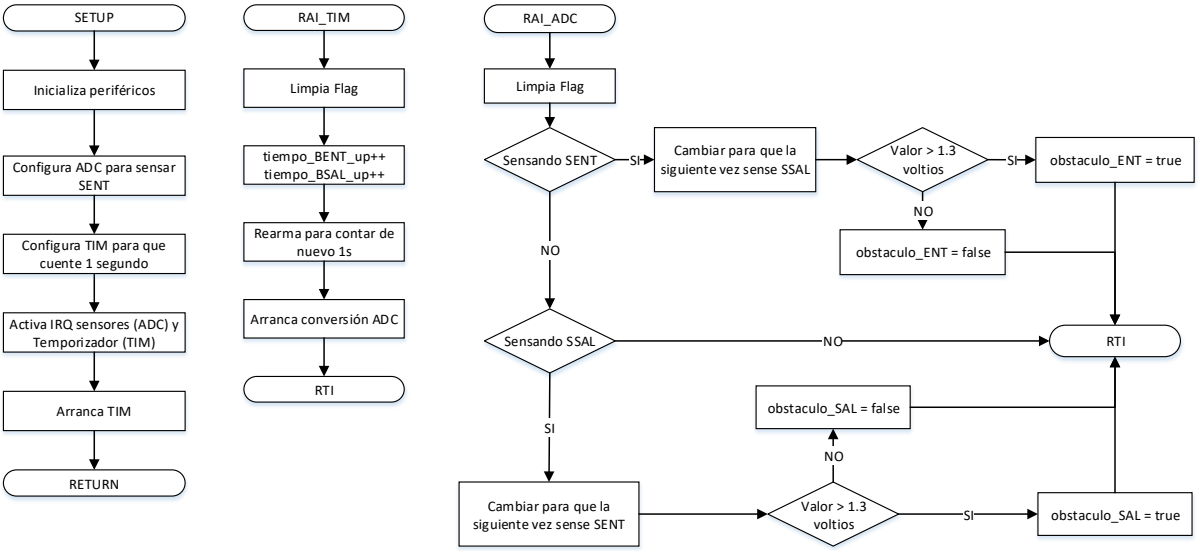



| PIN MICRO | HW CONECTADO | TIPO (E/S, DIGITAL/ ANALOGICO) | JUSTIFICACIÓN |
|---|---|---|---|
| *PA1* | *SENT* | *Entrada Analógica* | *Interrupción del ADC y configuración de canal que cambia en la RAI* |
| *PA5* | *SSAL* | *Entrada Analógica* | *Interrupción del ADC y configuración de canal que cambia en la RAI* |
| *PA6 – PA15* | *Display* | *Salida Digital* | *Son los únicos pines configurados que no tienen otra funcionalidad, por lo que deben ser utilizados en la función representar()* |
| *PB0* | *PENT* | *Entrada Digital* | *Entrada por EXTINT. Consulta su valor en el programa principal.* |
| *PB1* | *PSAL* | *Entrada Digital* | *Entrada por EXTINT. Consulta su valor en el programa principal.* |
| *PB2* | *BSAL* | *Salida Digital* | *Apertura y cierre de la barrera en el programa principal* |
| *PB3* | *BENT* | *Salida Digital* | *Apertura y cierre de la barrera en el programa principal* |

*2. El programa principal quedaría así:*




*Y las rutinas de la siguiente manera:*




*3. Se controla con el temporizador TIM3, el cual salta en la rutina de interrupción cada 1000 pasos. Teniendo en cuenta que PSC = 7999, y que la frecuencia de reloj es de 8MHz, el tiempo de cada paso es 1 milisegundo, por lo que la interrupción del TIM3 salta cada segundo. La barrera se mantiene abierta al menos 20 cuentas, por lo que al menos se mantiene abierta 20 segundos.*

*4. En el programa principal, cuando se detecta la pulsación del botón de entrada (PSEL), se hace la comprobación de que el num_vehiculos sea menor de 152 antes de abrir la barrera. Por lo tanto el número máximo de coches es 151.*

*Además, como la variable es un unsigned char, el número máximo de coches que se podrían gestionar, sería 255.*

*5. El ADC está configurado para hacer conversiones simples de un canal (cambiando el canal alternativamente entre el AIN1 y el AIN5). No se introduce retardo adicional entre conversiones en la configuración del ADC.*

*El reloj es de 8MHz, por lo que cada ciclo de conversión tarda 0,125 microsegundos. Se necesita un ciclo más que el número de bits con los que se convierte, por lo que el tiempo en una conversión es de 1,125 microsegundos. Este tiempo es mucho menor que el tiempo entre conversiones, ya que sólo se lanza una conversión cada vez que el TIM3 cuenta 1 segundo. Por lo tanto el tiempo entre conversiones es aproximadamente 1 segundo.*

*Para reducir este tiempo al mínimo, habría que cambiar la sentencia de arranque del ADC (ADC->CR2 |= 1<<30;), quitándola de la RAI del TIM3, y poniéndola al final de la RAI del ADC.*

*6. Falta limpiar el flag de conversión finalizada, por lo que habría que añadir:*

```
ADC->SR &= ~(1<<1);        // Limpia el flag
```

**Problema 2 (100 minutos – 4 puntos)**

Basándose en el microcontrolador que ha utilizado durante el curso con una frecuencia de reloj de 8MHz, se necesita que implemente un subsistema de control del suministro eléctrico de un sistema mayor, de tal forma que genere alarmas cuando dicho suministro no se ajuste a los requisitos del sistema. Para cumplir con esta función, al microcontrolador se le conectará un dispositivo externo que tiene como entrada la red eléctrica de suministro y como salida una señal cuadrada, $V_{cuadrada}$, del mismo periodo y ciclo de trabajo, *duty-cycle* (DC), que la señal de la red eléctrica, pero entre Vcc y masa (en condiciones normales, la frecuencia de la señal de la red eléctrica es de 50Hz con DC del 50%). El sistema admite frecuencias entre 40Hz y 60Hz y DC entre el 45% y el 55%

Partiendo de esta señal de entrada, el subsistema de control tiene que suministrar al mundo exterior la información a través de 2 LEDs (uno verde LV y uno rojo LR), un altavoz (al que le atacará una señal cuadrada de uno tono determinado) y una comunicación serie con baud rate de 19200, sin paridad, con 8 bits de datos y 1 bit de parada (19200,8,N,1). La información a ofrecer debe ser (por orden prioritario):

- Si no hay alarma: LV encendido, LR apagado, sin tono, sin comunicación serie
- Si frecuencia fuera de rango: LV apagado, LR encendido, tono de 1kHz, mensaje "HHMMSS_FR"
- Si DC fuera de rango: LV apagado, LR encendido, tono de 500Hz, mensaje "HHMMSS_DC"

Los mensajes sólo se envían una vez al empezar a ocurrir cada alarma. En dichos mensajes HH es la hora, MM es el minuto y SS es el segundo, cada uno de ellos tiene dos dígitos y por tanto se envía con dos caracteres ASCII. Para obtener dicha información, el subsistema cuenta con un Real Time Clock (RTC) al que se puede consultar la información a través de la siguiente función:

```
void ObtenerHora (unsigned char *hora, unsigned char *minuto, unsigned char *segundo);
```

Con estas especificaciones, y teniendo en cuenta que se valorará el nivel de optimización de la solución (es decir, menor uso de periféricos y menor tiempo de computación), conteste a las siguientes preguntas:

1. Enumere las tareas que realiza el microprocesador y los periféricos que utilizará para realizar cada una de ellas. Realice también el diagrama de bloques detallado del sistema, incluyendo todos los elementos del sistema mencionados en el enunciado y los pines de conexión del microcontrolador con cada uno de ellos (25%)

2. Describa como configuraría los periféricos utilizados para realizar la comunicación serie y generar la señal cuadrada que atacará al altavoz. Escriba el valor de los registros de configuración de estos periféricos. Justifique los valores utilizados (25%)

3. Indique los periféricos que se atenderán por interrupciones. Justifique su elección. Escriba las líneas de configuración necesarias para habilitar las fuentes de interrupción seleccionadas. (20%)

4. Diagrama de flujo del programa principal (utilice funciones para realizar las acciones necesarias en caso de que la frecuencia medida esté fuera de rango o el DC medido esté fuera de rango), diagramas de flujo de las funciones utilizadas en el principal y diagramas de flujo de la/s rutina/s de atención a la interrupción (30%)

**SOLUCIÓN:**

*Antes de empezar a decidir los módulos a utilizar, es necesario analizar las necesidades de temporización, para ver si se pueden compartir temporizadores. Estas necesidades son:*

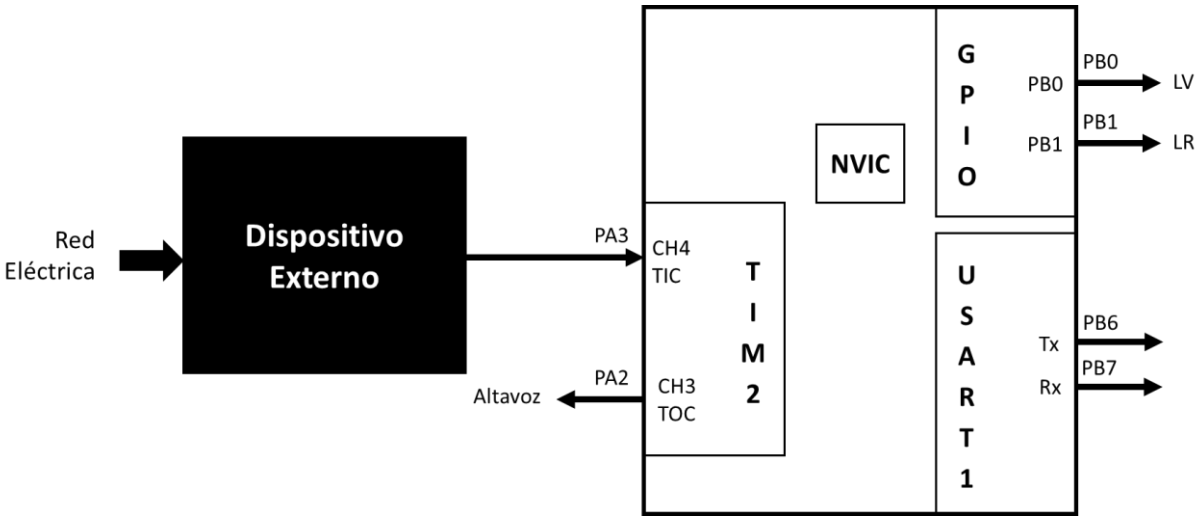
- *Para medir la frecuencia de la señal de entrada y el DC con un mínimo de resolución, debería considerarse una frecuencia de mínimo un orden de magnitud mayor, es decir, 500Hz. Redondeando, se puede considera una medida a 1KHz, es decir, un tiempo de paso del contador de 1ms.*
  - *Con un contador a ese ritmo se pueden llegar a medir fácilmente tiempos de incluso 60 segundos.*
  - *Modalidad de TIC, un único canal, entrada hardware*
- *Al altavoz le tiene que atacar una señal de 1KHz o de 500Hz. Lo más condicionante es 1KHz, que exige un funcionamiento en toogle cada 0,5ms. Por lo tanto el temporizador debería contar a ritmo de 50us o simplemente 10us.*
  - *Modalidad TOC, un único canal, con salida hardware en modo toogle*

*En resumen, se necesita un TIC con una entrada hardware y 1 TOC con salida hardware. Todo podría estar contenido en los cuatro canales de un TIM. Si atendemos a tiempos, con el requisito del altavoz de 10us, los resultados de cuenta para cada uno de los casos serían:*

- *Un periodo de señal habitual sería 2.000 pasos*
- *El toogle de 1KHz serían 50 pasos (COMP1K)*
- *El toogle de 500Hz serían 100 pasos (COMP500)*

*Por lo tanto se puede utilizar un único temporizador, con 2 canales y 2 pines asociados. De esta manera se deja el resto de los temporizadores libres para aquellas funcionalidades adicionales que requiera el sistema.*

*a) Atendiendo a los requisitos del enunciado, y a los cálculos anteriores, el diagrama de bloques queda de la siguiente manera:*



*c) Para liberar recursos es preciso que se utilice el mínimo tiempo posible del programa principal para las funcionalidades adicionales que se necesiten. Por tanto se utilizarán interrupciones todo lo posible. En concreto:*

- *La comunicación serie no tiene definida recepción, pero sí transmisión. La transmisión es por tramas además de longitud fija. Se puede evitar las rutinas de espera activa utilizando interrupciones en la transmisión. Se activarán una vez preparada la trama a enviar, y se desactivará en el envío del último carácter.*
- *Se configurará la interrupción del TIM2 de forma que interrumpa por cada uno de los canales usados de forma que:*
    - *Cuando sea por TOC del canal 3, se actualizará el nuevo valor de comparación (del CCR3).*
    - *Cuando sea por TIC del canal 4, se comprobará cuál es el flanco que ha ocurrido, y a partir de ahí se actualizará el valor de la frecuencia (si es flanco de subida) o del DC (si es flanco de bajada, respecto al flanco de subida anterior). Atendiendo al valor medido, se modificará el comportamiento de LV, LR, tono y trama a enviar.*

*b) Los periféricos a utilizar son, por lo tanto:*

*GPIOA: con función alternativa de TIM2 para los pines PA2 y PA3.*

```
GPIOA->MODER&=~(0x03 << (2x2)); // PA2 como Función Alternativa
GPIOA->MODER|=0x02 << (2x2);
GPIOA->MODER&=~(0x03 << (3x2)); // PB3 como Función Alternativa
GPIOA->MODER|=0x02 << (3x2);
GPIOA->AFR[0]&=~(0x0F << (2*4)); // TIM2 para PA2
GPIOA->AFR[0]|=0x01 << (2*4);
GPIOA->AFR[0]&=~(0x0F << (3*4)); // TIM2 para PA3
GPIOA->AFR[0]|=0x01 << (3*4);
```

*GPIOB: como salida digital en PB0 y PB1, y de función alternativa de USART1 para los pines PB6 y PB7.*

```
GPIOB->MODER&=~(0x03 << (0x2)); // PB0 como Digital Output
GPIOB->MODER|=0x01 << (0x2);
GPIOB->MODER&=~(0x03 << (1x2)); // PB1 como Digital Output
GPIOB->MODER|=0x01 << (1x2);
GPIOB->MODER&=~(0x03 << (6x2)); // PB6 como Función Alternativa
```

*GPIOB->MODER|=0x02 << (6x2);*

*GPIOB->MODER&=~(0x03 << (7x2)); // PB7 como Función Alternativa*

*GPIOB->MODER|=0x02 << (7x2);*

*GPIOA->AFR[0]&=~(0x0F << (6*4)); // USART1 para PB6*

*GPIOA->AFR[0]|=0x07 << (6*4);*

*GPIOA->AFR[0]&=~(0x0F << (7*4)); // USART1 para PB7*

*GPIOA->AFR[0]|=0x07 << (7*4);*

*TIM2: Como el pclk es de 8MHz, se toma un pre-escalado de 80 el contador llevará pasos de 10us. Se habilitarán las interrupciones por TIM2 a los 2 canales, aunque cada una de ellas se activará y desactivará en el momento que corresponda. Además la configuración será:*

- *Canal 3, con TOC, con salida hardware en modo toogle*
- *Canal 4, con TIC, con entrada hardware por ambos flancos*

*TIM2->CR1 = 0; // Todo por defecto, sin auto-reload y contador parado*

*TIM2->CR2 = 0; // Por defecto*

*TIM2->SMCR2 = 0; // Por defecto*

*TIM2->PSC = 79; // Pre-escalado de 80*

*TIM2->CCMR2 = 0; // Limpio todo el registro antes de configurar los dos canales (3 y 4)*

*Canal 3:*

*Ya ha quedado como TOC en CC3S, sin preload, y resto de valores por defecto*

*TIM2->CCMR2 |= 3<<4; Salida hardware en modo toogle*

*(otra opción, la que se usa en los diagramas de flujo, es dejar la IRQ activa desde el principio, pero indicarle a la salida que en lugar de ser toogle, sea forzada a 0 en la comparación, poniendo 2<<4 en lugar de 3<<4)*

*TIM2->CCER |= 1 << 8; Salida hardware activada*

*Dentro del programa ya se actualizará el TIM2->CCR3 al valor dado por CCR3+(COMP1K o COMP500)*

*Canal 4:*

*TIM2->CCMR2 |= 0x01 << (0+8); // como TIC con su propia entrada*

*Resto de valores del CCMR2 a cero, por defecto.*

*TIM2->CCER |= 0x01 << 15; Flanco de bajada activo*

*TIM2->CCER |= 0x01 << 13; Flanco de subida activo*

*TIM2->CCER |= 0x01 << 12; Entrada hardware activada*

*TIM2-> SR = 0; // Se deja limpio el registro de estado*

*Cuando se quiera activar la IRQ por CH3: TIM2->DIER |= 0x01 <<3;*

*Cuando se quiera desactivar la IRQ por CH3: TIM2->DIER &= ~(0x01 << 3);*

*Cuando se quiera activar la IRQ por CH4: TIM2->DIER |= 0x01 <<4;*

*Cuando se quiera desactivar la IRQ por CH4: TIM2->DIER &= ~(0x01 << 4);*

*USART1: Se configurará la comunicación a 19200, 8, N, 1, con la recepción inhabilitada y la transmisión habilitada. Se dejará deshabilitada la interrupción por buffer de transmisión vacío, hasta que se tenga que transmitir una trama.*

*USART1->CR1 = 0; // Limpio toda la configuración y la dejo con 8 bits, inhabilitada, sin paridad, sin IRQs activadas, y sin habilitar ni recepción ni transmisión*

*USART1->CR1 |= 1 << 3; // Habilito Tx*

*USART1->CR1|= 1 << 13; // Habilito la USART1*

*USART1->CR2 = 0; // Limpio toda la configuración y dejo 1 bit de parada*

*Para calcular el baudrate se toma el valor de la tabla para 16MHz con Oversampling = 0, y se divide por dos: value = 26,03125*

*De ahí se obtiene que la mantisa es 26 y la parte fraccionaria es 0,03125.*

*Se multiplica por 16 (= 0,5) y se coge el entero más cercano (se puede coger 0 o 1). Por tanto:*

*USART1->BRR = 26 << 4;*

*USART1->BRR |= 1;*

*USART1->SR = 0; //Se limpia el registro de estado*

*NO SE ACTIVAN LAS INTERRUPCIONES, HASTA NO EMPEZAR A ENVIAR EL PRIMER CARÁCTER*

*Cuando se quiera habilitar la IRQ por transmisión: USART1->CR1 |= 1<<7;*

*Cuando se quiera deshabilitar la IRQ por transmisión: USART1->CR1 &= ~(1<<7);*

*NVIC: Se activarán las interrupciones por TIM2 y por USART*

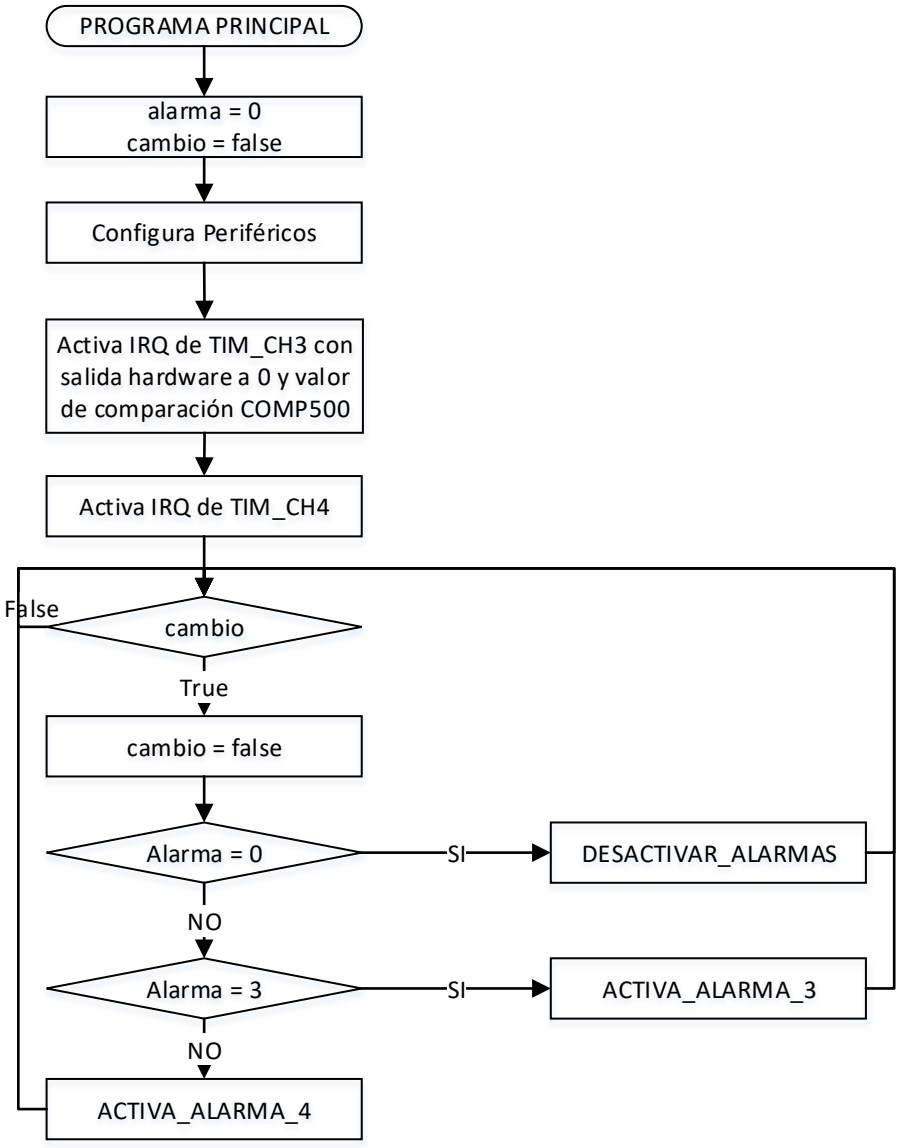
*Para activar la interrupciones de esos dos periféricos, se puede hacer desde el principio y luego controlarlo con el bit propio de cada periférico. En ese caso, la activación de esas dos fuentes de interrupción, sería:*

```
NVIC->ISER[0] = 1 << 28; // Activación por TIM2
NVIC->ISER[1] = 1 << (37-32); // Activación por USART1
```

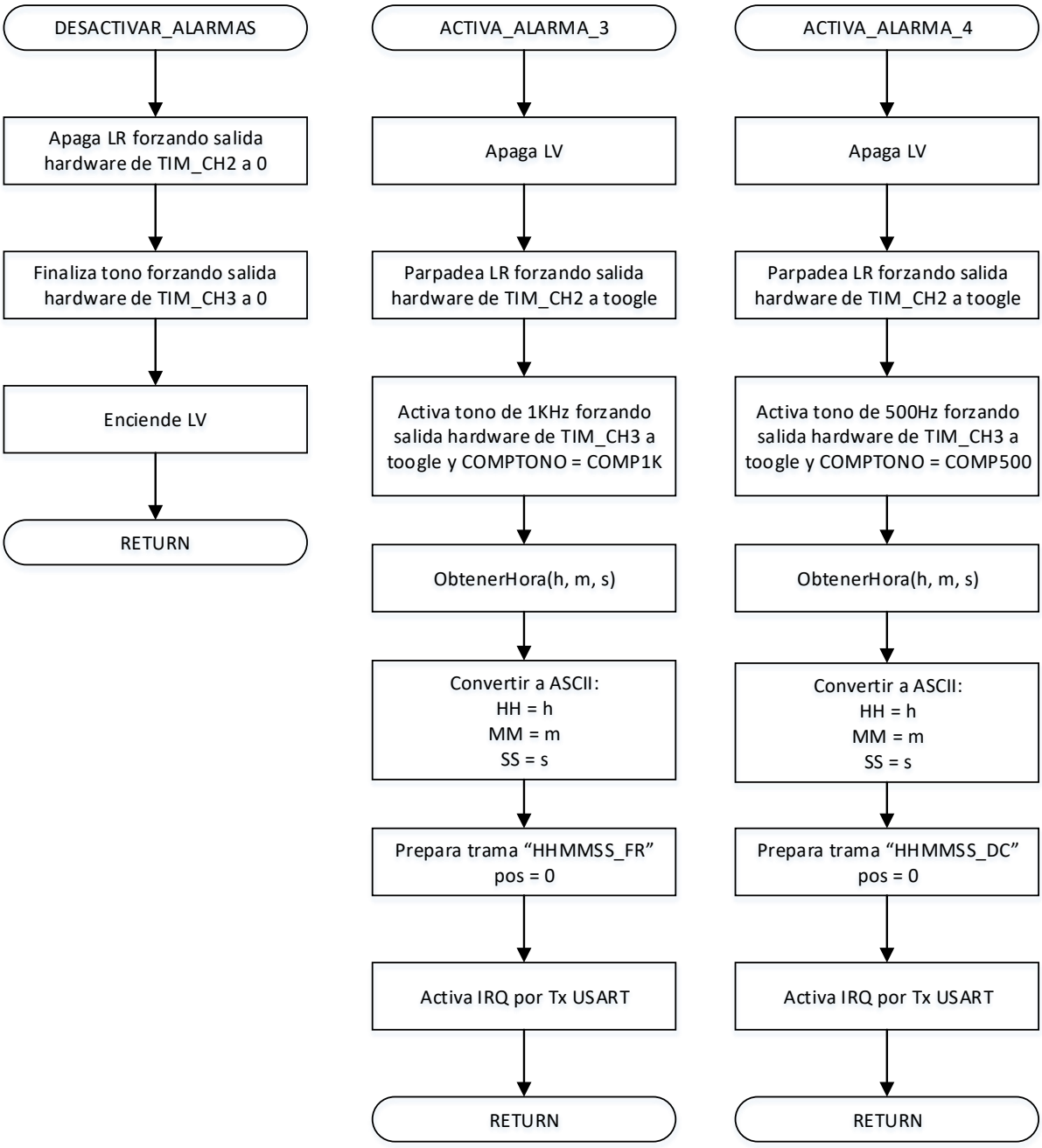
*d) con todas estas consideraciones los diagramas de flujo son como figuran a continuación teniendo en cuenta la siguiente terminología:*

- *alarma:*
  - *= 0: Sin alarma*
  - *= 3: Frecuencia fuera de rango*
  - *= 4: DC fuera de rango*
- *BUFLEN = longitud del buffer de transmisión. Aquí es fija y es 9*
- *cambio = flag que determina si hay un cambio en el estado de alarma*
- *CCR = registro que contiene el valor del temporizador (ya sea para comparar, o capturado)*
- *COMP1K = Número de pasos de comparación para obtener señal de 1KHz*
- *COMP500 = Número de pasos de comparación para obtener señal de 500Hz*
- *IRQ = Interrupción*
- *PERMAX = periodo máximo admitido (el equivalente a 40Hz)*
- *PERMIN = periodo mínimo admitido (el equivalente a 60Hz)*
- *pos = posición dentro buffer de transmisión del carácter a transmitir*
- *TIM_CH3 = Temporizador para tono de altavoz*
- *TIM_CH4 = Temporizador para captura de tiempos de flancos*

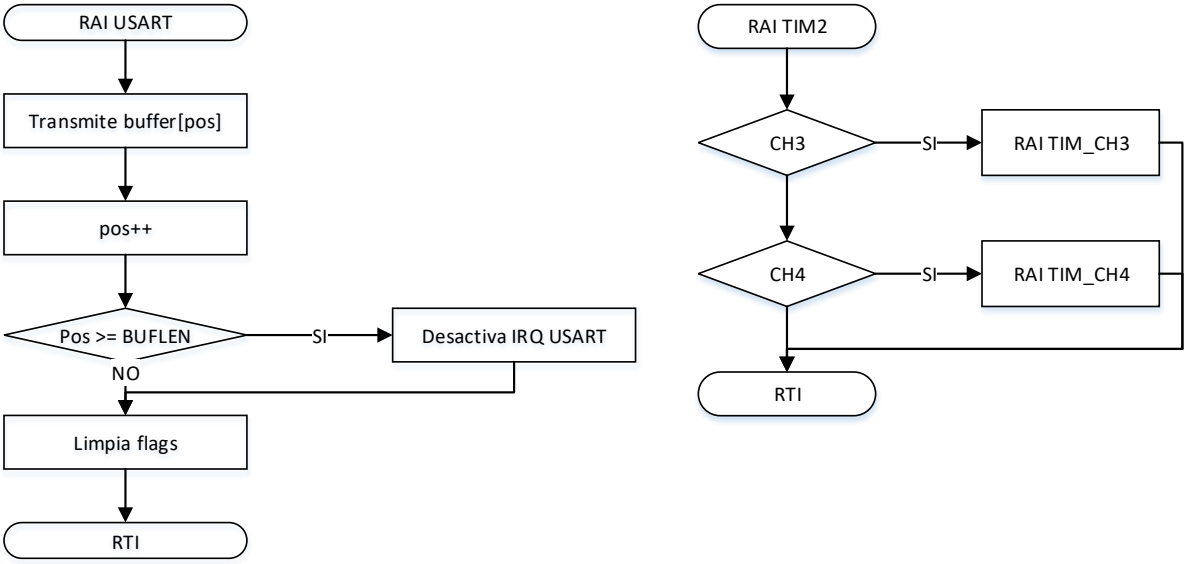
*El programa principal quedaría así (pendiente de incluir nuevas funcionalidades tanto en la inicialización como en el bucle principal):*

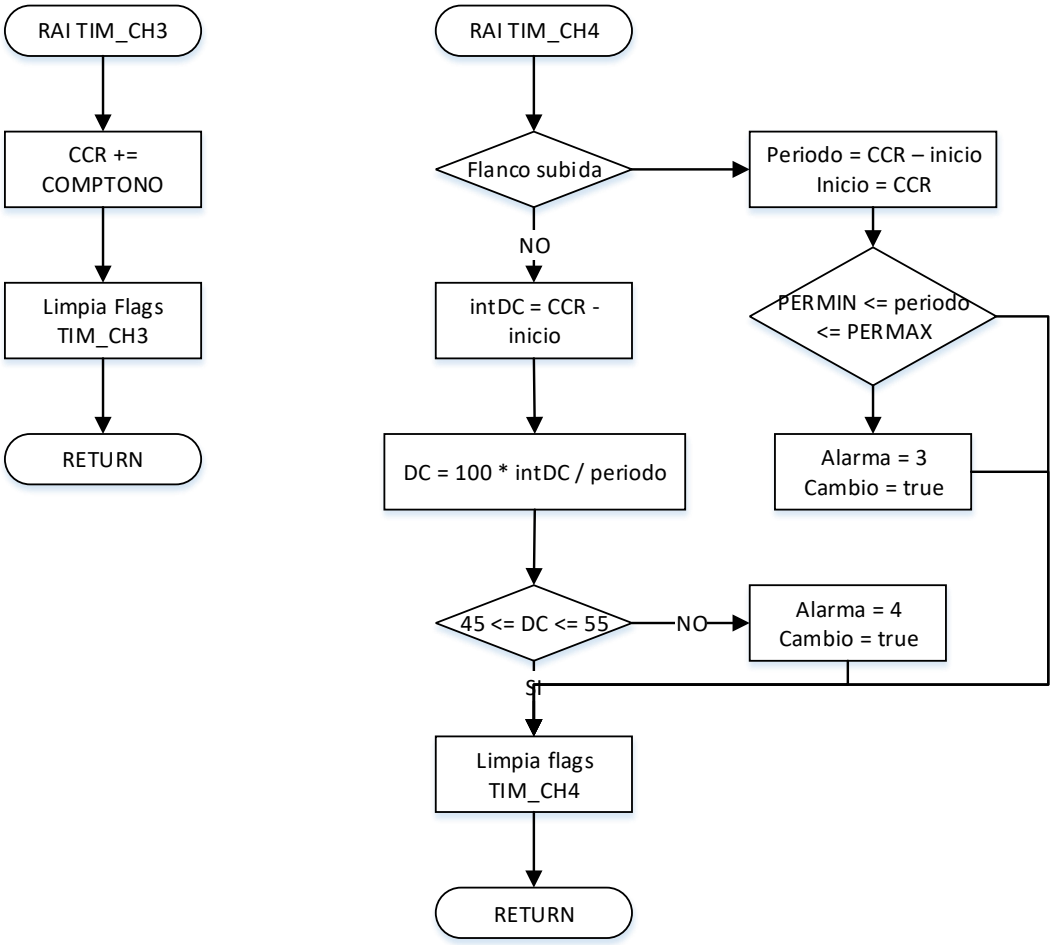



*Las alarmas a generar quedarían así:*




*Y las Rutinas de Atención a la Interrupción quedarían así:*


RAI USART
Transmite buffer[pos]
pos++
Pos >= BUFLEN
SI
Desactiva IRQ USART
NO
Limpia flags
RTI
RAI TIM2
CH3
SI
RAI TIM_CH3
CH4
SI
RAI TIM_CH4
RTI


RAI TIM_CH3
CCR += COMPTONO
Limpia Flags TIM_CH3
RETURN
RAI TIM_CH4
Flanco subida
Periodo = CCR – inicio
Inicio = CCR
NO
intDC = CCR - inicio
PERMIN <= periodo <= PERMAX
DC = 100 * intDC / periodo
Alarma = 3
Cambio = true
45 <= DC <= 55
NO
Alarma = 4
Cambio = true
SI
Limpia flags TIM_CH4
RETURN