UNIVERSIDAD CARLOS III DE MADRID (Dpto. de Tecnología Electrónica)
Sist. Dig. Basados en Microprocesador **(Gr. Ing. Telemática)**
21 de junio de 2013 EXAMEN CONV. EXTRAORD. (3 horas)

SOLUCIÓN

EJERCICIO 1 (2.0 puntos):

Diseñe un microprocesador con las siguientes características:

- 16KB de memoria de datos.
- 8KB de memoria de programa.
- Juego de 128 instrucciones de 16 bits.
- Bus de datos de 8 bits.
- Minimice la velocidad de carga de la instrucción (Fetch).

Se pide:

a) En caso de implementarse con arquitectura Von Neuman:

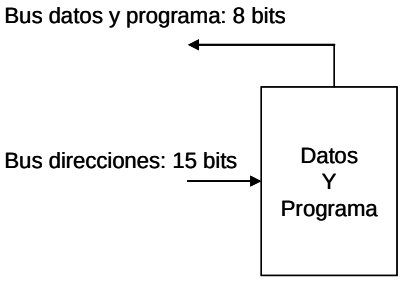
a. Diagrama de bloques indicando el tamaño de los buses principales.

En esta arquitectura la memoria de programa y de datos están juntas, como el bus de datos es de 8 bits, esto obliga a que el bus de programa también lo sea y por tanto el tamaño de palabra será de 1 byte.

Como necesitamos 16KB de datos y 8KB de programa, tenemos una necesidad de 24KB. No podemos direccionar 24KB, tendremos que irnos a la primera potencia de 2 que esté por encima de 24KB, es decir, 32KB.

Por tanto:

- Bus de direcciones (32KB): 15 líneas de dirección (bits).
- Bus de datos/programa: 8 bits



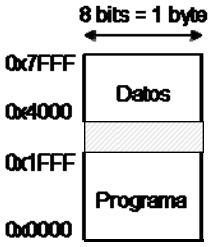
b. Mapa de memoria.

32KB de memoria en total -> 15 líneas -> 0x7FFF

16KB de datos -> 14 líneas -> 0x3FFF

8KB de programa -> 13 líneas -> 0x1FFF

Juego de 128 instrucciones de 16 bits -> No es un condicionante de diseño para la memoria, sino una característica interna. Estas instrucciones, sean como sean, se almacenarán en la memoria de programa.



c. Tamaño del contador de programa y del registro de instrucción.

PC: 15 bits (los necesarios para direccionar 32KB), si bien también podría ser de 13 bits limitando el acceso sólo a la memoria ROM (es decir, ni se podría ampliar la memoria de programa del dispositivo, ni se podría utilizar la memoria de datos para aplicaciones).

RI: 16 bits (tamaño de la instrucción).

b) En caso de implementarse con arquitectura Harvard:

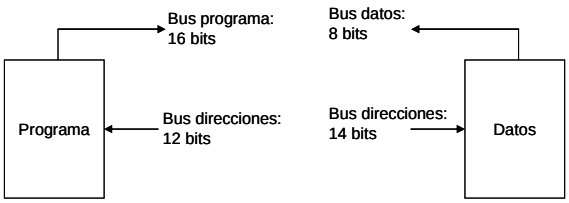
a. Diagrama de bloques indicando el tamaño de los buses principales.

En este caso la memoria de programa y de datos están separadas y además se nos pide que optimicemos la velocidad de carga de la instrucción, por lo que nos obliga a que toda la instrucción se cargue (pase de la memoria de programa al RI) en un solo paso, por lo que nos obliga a que el tamaño de palabra de la memoria de programa sea de 2 bytes (16 bits).

Es decir, por un lado tenemos la memoria de datos que tendrá un bus de datos de 8 bits y un tamaño de memoria de 16KB.

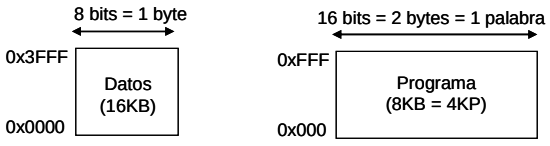
Y por otro lado tenemos una memoria de programa que tendrá un bus de programa de 16 bits (tamaño de palabra 2 bytes) y un tamaño de memoria de 8KB, dado que el tamaño de palabra son 2 bytes, en realidad la necesidad de direccionamiento es de 4KPalabras (8KB / 16 bits por palabra -> 4KPalabras).

Por tanto la memorias y los buses quedarían como a continuación se puede ver:



b. Mapas de memoria.

Atendiendo a los condicionantes anteriores, los mapas de memoria quedarían de la siguiente manera:



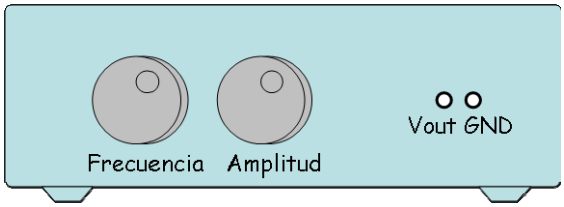
c. Tamaño del contador de programa y del registro de instrucción.

PC: 12 bits (los necesarios para direccionar 4KPalabras).

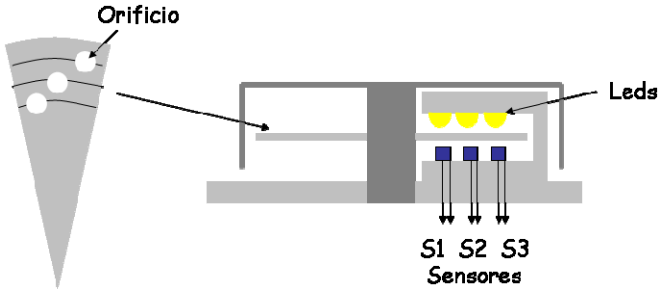
IR: 16 bits (tamaño de la instrucción).

EJERCICIO 2 (4.0 puntos):

El generador de onda de la siguiente imagen produce una señal de triangular de frecuencia y amplitud variable construida con 10 muestras por periodo.



La amplitud de la onda varía entre 0 y 3V, y es controlada mediante una rueda de giro "sin fin" cuyo mecanismo se puede ver en la siguiente imagen.



La rueda dispone de un orificio cada 10º, es decir, dispone de un total de 36 orificios. Dichos orificios están situados a tres radios distintos, de forma que a su paso por los leds se excita únicamente el sensor asociado al mismo. Esto quiere decir que cuando la rueda gira en sentido horario los sensores ópticos se excitan siguiendo la siguiente secuencia S3-> S2 ->S1, mientras que cuando gira en sentido antihorario la secuencia de excitación es la siguiente S1 -> S2 ->S3.
El sentido horario implicará un incremento de 10mV en la amplitud de la tensión de salida por cada 10º girados, mientras que el giro en sentido antihorario supondrá -10mV por cada 10º de giro.

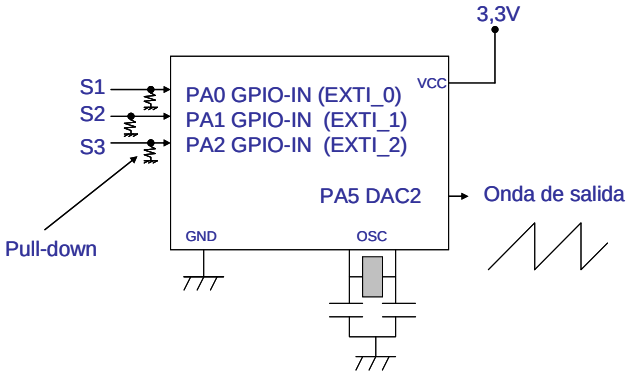
La rueda que controla la frecuencia, es una rueda asociada a un potenciómetro y regula la frecuencia entre 1KHz y 10KHz cuando la señal ofrecida por el potenciómetro vale 0 V y 3 V respectivamente.

Como parámetros de desarrollo se han marcado los siguientes:

2. La recogida de información de la rueda de control de frecuencia se gestionará con la interrupción del ADC. En esta interrupción, la variable global "tiempo_muestra" se actualizará con el tiempo que debe transcurrir entre dos muestras consecutivas emitidas por el dispositivo para una determinada frecuencia.
3. La recogida de información de la rueda de control de amplitud se realizará con interrupciones externas (EXT_INT). Utilizando la variable global "estado" que controlará el estado actual (S1, S2 y S3).
4. El muestreo de la señal de salida se realizará mediante la interrupción de un TOC que activará el flag "flagSacarDato".
5. El flujo principal del programa (main), usará las variables "flagSacarDato", "estado" y "amplitud" para generar la onda con el periférico correspondiente (gestionando también el control de la amplitud de la misma).

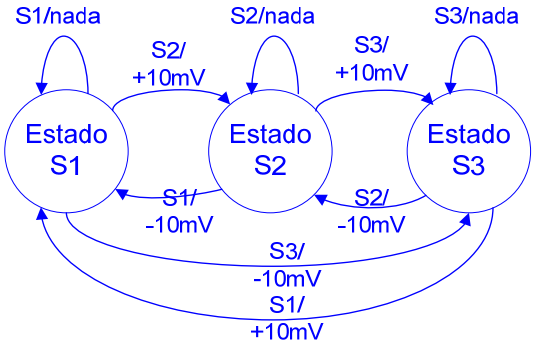
Se pide:

a) Diagrama de bloques.

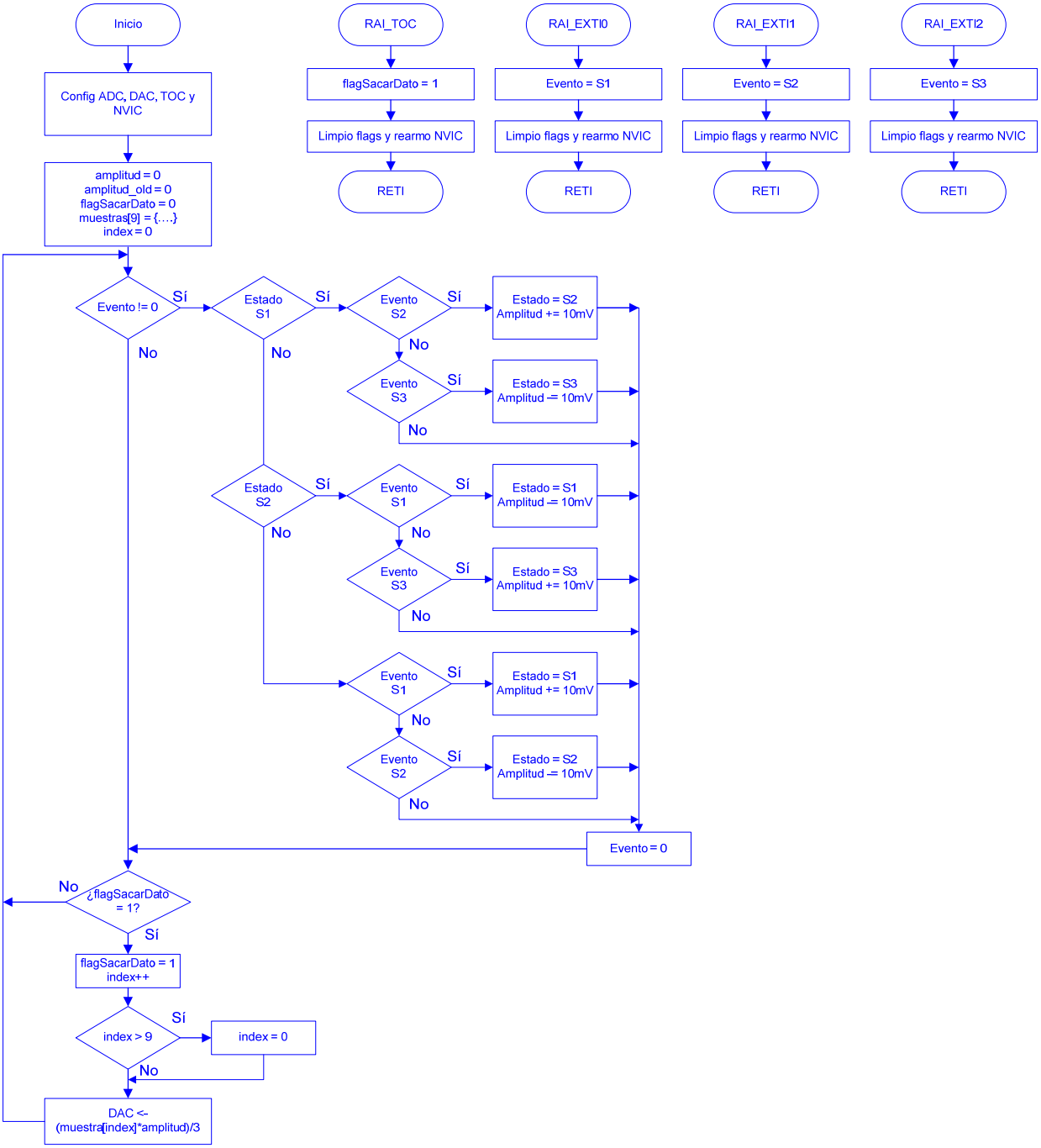


b) Diagrama de flujo.

Aunque no lo piden, vamos a dibujar un diagrama de estados antes de dibujar el diagrama de flujo, eso nos ayudará a entender la gestión de eventos en función del estado.



Por tanto, el diagrama de flujo será el siguiente:


Inicio
Config ADC, DAC, TOC y NVIC
amplitud = 0
amplitud_old = 0
flagSacarDato = 0
muestras[9] = {....}
index = 0
RAI_TOC
flagSacarDato = 1
Limpio flags y rearmo NVIC
RETI
RAI_EXTI0
Evento = S1
Limpio flags y rearmo NVIC
RETI
RAI_EXTI1
Evento = S2
Limpio flags y rearmo NVIC
RETI
RAI_EXTI2
Evento = S3
Limpio flags y rearmo NVIC
RETI
Evento != 0
Sí
No
Estado S1
Evento S2
Estado = S2 Amplitud += 10mV
Evento S3
Estado = S3 Amplitud -= 10mV
Estado S2
Evento S1
Estado = S1 Amplitud -= 10mV
Estado = S3 Amplitud += 10mV
Estado = S1 Amplitud += 10mV
Estado = S2 Amplitud -= 10mV
Evento = 0
¿flagSacarDato = 1?
flagSacarDato = 1
index++
index > 9
index = 0
DAC <- (muestra[index]*amplitud)/3

c) Código de configuración de los pines del micro (únicamente los GPIO, sin las EXTI, ni el NVIC).

```
// PA0, PA1 y PA2 como GPIO de entrada
GPIO->MODER &= 0xFFFFFFC0;

// PA5 como Salida analógica
GPIOA->MODER |= 0x00000C00;
```

d) Código de la RAI asociada a alguna de las interrupciones externas.

```
#define     Evento_S1  1
#define     Evento_S2  2
#define     Evento_S3  3

void EXTI0_IRQHandler(void) {
  if ( (EXTI->PR & 0x01) != 0) {
       Evento = Evento_S1;
       EXTI->PR |= 0x01;  // Limpia Flag
    }
 }
```

EJERCICIO 3 (4.0 puntos):

En las próximas hojas aparece el código de un programa a cargar en un sistema basado en un STM32L152RB, **al cual le puede faltar alguna función y/o alguna definición y/o contener algún error de programación**. El programa corresponde a la implementación de un velocímetro de bicicleta, capaz de mostrar por la pantalla la información sobre la distancia recorrida, la velocidad instantánea y la temperatura. De dicho sistema se conocen las siguientes características:

- Tiene dispositivos conectados cuya señal de entrada al STM32L152RB, es un valor cualquiera entre 0 y 3,3V, con una variación de velocidad menor de 1Hz.
- Puede tener algún otro dispositivo más conectado.

Se pretende que el alumno analice dicho código y, a partir de ahí conteste **razonadamente** a las siguientes preguntas (*algunas de las justificaciones se pueden realizar indicando las líneas de código donde se encuentra la evidencia*):

1. ¿Qué elementos (periféricos) del STM32L152RB se están utilizando? (10%)
2. Teniendo en cuenta que tiene una configuración de reloj que hace que el pclk de todos los periféricos vaya a 12MHz, indique la configuración de cada uno de los periféricos del microcontrolador cuando se encuentren en funcionamiento. Absténgase de simplemente decir el valor de cada registro de configuración; lo que se pide es la funcionalidad que se obtiene. Es imprescindible detallar la escala temporal (caso de que exista) que utilizan los periféricos. (10%)
3. De un significado breve a las siguientes variables del programa. No tienen por qué ser acrónimos de ningún tipo (20%):
    a. cero
    b. uno
    c. dos
    d. tres
    e. cuatro
    f. cinco
    g. seis
4. El código presenta al menos tres errores. Dos de esos errores está en los nombres de las funciones, mientras que el otro se encuentra en el funcionamiento del mismo. Encuéntrelos, justifíquelos y dé una solución a los mismos. (20%)
5. Analizando la pureza del código, un experto expone una crítica a la línea 35. ¿Podría decir qué crítica es, y cómo la solucionaría? (10%)
6. Realice el Diagrama de flujo de todas y cada una de las funciones utilizadas, así como del programa principal. No haga siempre una transposición directa del código en un diagrama de flujo, sino represente la funcionalidad obtenida, mediante dicho diagrama de flujo. (30%)

ANEXO I

```
1   #include "stm32l1xx.h"
2   #include "Biblioteca_SDM.h"
3   unsigned char radio=23;
4   unsigned uno, dos;
5   unsigned char cero;
6   unsigned tres;
7   unsigned cuatro;
8   unsigned cinco;
9   unsigned seis;
10
11  void RAI1 (void) {
12     EXTI->PR = 0x01;
13     NVIC->ICER[0] |= (1 << 6);
14     if ((EXTI->RTSR & 0x01) == 0) {
15        TIM4->CCR2 = TIM4->CNT + 2000;
16        TIM4->CR1 |= 0x0001;
17        EXTI->RTSR |= 0x01;
18        EXTI->FTSR &= ~(0x01);
19        }
20     else {
21        TIM4->CR1 &= ~(0x0001);
22        EXTI->FTSR |= 0x01;
23        EXTI->RTSR &= ~(0x01);
24        }
25     NVIC->ISER[0] |= (1 << 6);
26     }
27
28  void RAI2 (void) {
29     if (TIM4->SR & 0x0004 == 1) {
30        cero = 1;
31        TIM4->CR1 &= ~(0x0001);
32        TIM4->SR = 0x0004;
33        }
34     else {
35        seis = 2 * 3142 * radio;
36        tres += seis;
37        dos = TIM4->CCR1 - uno;
38        if (dos < 0) dos += 0xFFFFFFFF;
39        cinco = seis / dos;
40        uno = TIM4->CCR1;
41        TIM4->SR = 0x0002;
42        }
43     }
44
45  int main (void) {
46     Init_SDM();
47     Init_LCD();
48     cero = 0;
49     GPIOA->MODER |= 0x00000300;
50     GPIOA->MODER &= ~(1 << (0*2 +1));
51     GPIOA->MODER &= ~(1 << (0*2));
52     GPIOA->PUPDR &= ~(11 << (0*2));
53     GPIOB->MODER |= 0x00000001 << (2*6 +1);
```

```
54      GPIOB->MODER &= ~(0x00000001 << (2*6));
55      GPIOB->AFR[0] &= ~(0x0F << (4*6));
56      GPIOB->AFR[0] |= 0x02 << (4*6);
57      ADC1->CR2 &= ~(0x00000001);
58      ADC1->CR1 = 0x02000000;
59      ADC1->CR2 = 0x00000472;
60      ADC1->SMPR1 = 0;
61      ADC1->SMPR2 = 0;
62      ADC1->SMPR3 = 0;
63      ADC1->SQR1 = 0x00000000;
64      ADC1->SQR5 = 0x00000004;
65      ADC1->CR2 |= 0x00000001;
66      while ((ADC1->SR&0x0040)==0);
67      ADC1->CR2 |= 0x40000000;
68      EXTI->FTSR |= 0x01;
69      EXTI->RTSR &= ~(0x01);
70      SYSCFG->EXTICR[0] = 0;
71      EXTI->IMR |= 0x01;
72      NVIC->ISER[0] |= (1 << 6);
73      TIM4->CR1 = 0x0000;
74      TIM4->CR2 = 0x0000;
75      TIM4->SMCR = 0x0000;
76      TIM4->PSC = 12000;
77      TIM4->CNT = 0;
78      TIM4->ARR = 0xFFFF;
79      TIM4->CCR2 = 0;
80      TIM4->DCR = 0;
81      TIM4->DIER = 0x0006;
82      TIM4->CCMR1 = 0x0001;
83      TIM4->CCMR2 = 0x0000;
84      TIM4->CCER = 0x0003;
85      TIM4->CR1 |= 0x0001;
86      TIM4->EGR |= 0x0001;
87      TIM4->SR = 0;
88      NVIC->ISER[0] |= (1 << 30);
89      uno = 0;
90      while (1) {
91         if (cero!=0) {
92            uno = 0;
93            tres = 0;
94            }
95         while ((ADC1->SR & 0x0002)==0);
96         cuatro = (unsigned char)(ADC1->DR & 0x000000FF);
97         MuestraVelocimetro(cinco, tres, cuatro);
98         }
99      }
```

SOLUCIÓN

1) Los elementos que se están utilizando son:

- *Conversión Analógica/Digital (líneas 57-65)*
- *Temporizador (Timer 4) en funcionalidades de TIC y TOC (líneas 73-87)*
- *Interrupción externa 0 (EXTI0) (líneas 68-71)*
- *Controlador de Interrupciones Vectorizadas (líneas 72, 88)*

2) La configuración de los distintos periféricos es:

- *ADC: En la línea 58 se configura a 8 bits, y en la línea 59 se configura en modo burst, a la velocidad más lenta. En las líneas 63-64 se configura un único canal que sea el AIN4. Por lo que se hacen medidas continuas de un valor analógico externo (suponemos que es la temperatura).*
- *Timer 4: En la línea 76 se configura el pre-escalado, para que el temporizador mida unidades de milisegundos. Además se activa la funcionalidad TIC, mediante el flanco de bajada en el canal 1 y pin PB6. En un determinado momento se utiliza la funcionalidad TOC, para medir 2 segundos desde que se pulsa el reset (se activa la EINT1). Todo evento (tanto el TIC como el TOC) provoca interrupción, y en la RAI correspondiente (RAI2), se mira qué servicio ha dado la interrupción y se procede en consecuencia. El temporizador funciona siempre en modo continuo, no reseteándose nunca, por lo que todas las medidas se hacen de forma relativa (en relación al valor anterior o el actual del CNT).*
- *EXTI: Se utiliza el EXTI0 por PA0 y se configura para que salte por flanco de bajada inicialmente, y luego se cambia la polaridad, pasando a flanco de subida, o flanco de bajada, según interese (el cambio se hace en la RAI1). La configuración inicial se hace en las líneas 68-71.*
- *NVIC: Se activan interrupciones por dos canales de entrada: la EXTI0 (línea 72), y el TIM4 (línea 88). En la RAI1 se desactiva el NVIC del EXTI0 y luego se vuelve a activar.*

3) Los significados de cada variable son:

- *cero: es un flag que se activa cuando se recibe una pulsación por EXTI0, y se mantiene durante un tiempo de 2 segundos mínimo. Provoca el poner a cero la distancia y velocidad.*
- *uno: valor de tiempo capturado en la anterior medida*
- *dos: tiempo que ha transcurrido entre la anterior medida (paso por vuelta) y la actual. Tiempo de vuelta de rueda.*
- *tres: distancia acumulada*

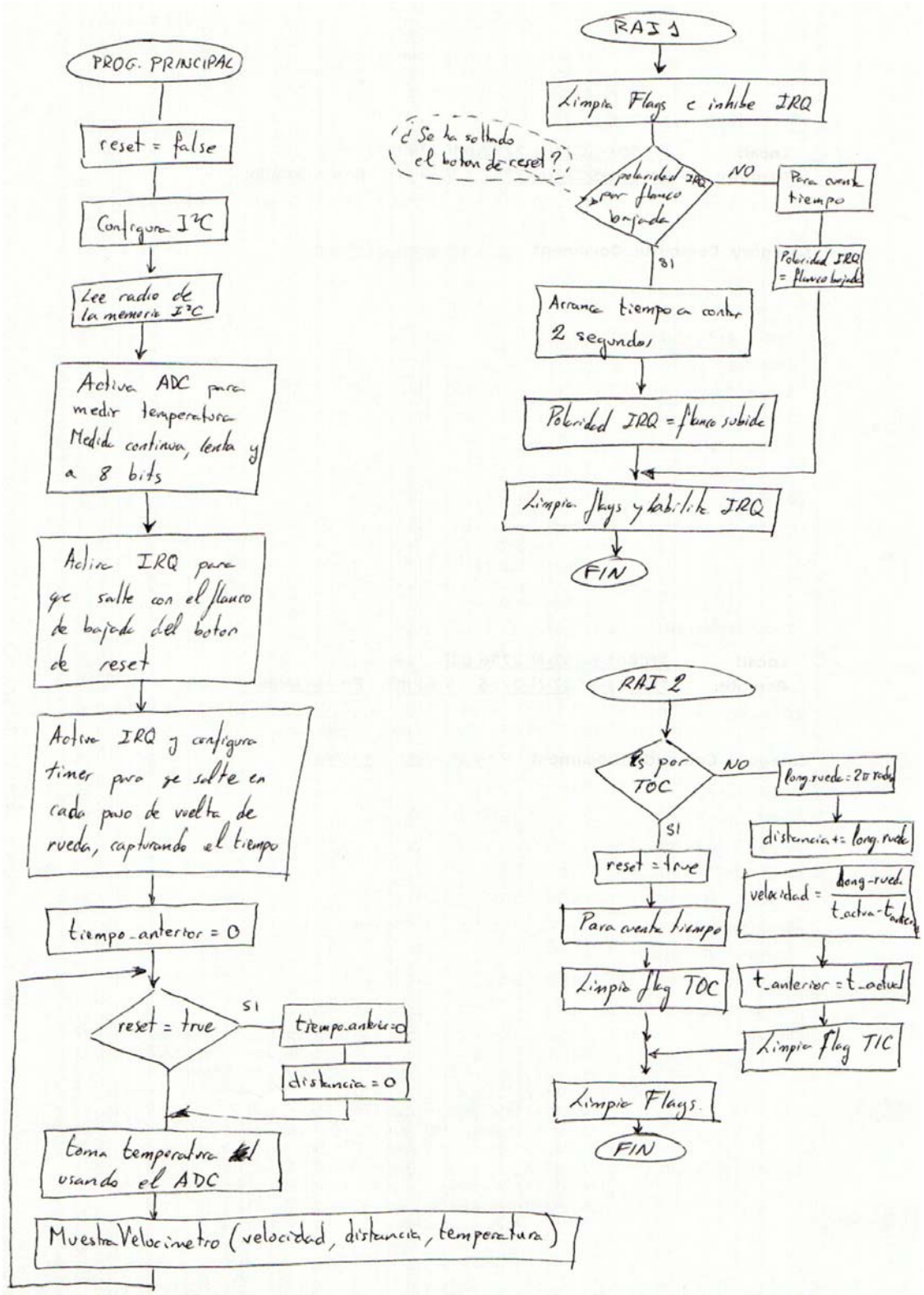
- *cuatro: temperatura obtenida del ADC*
- *cinco: velocidad instantánea*
- *seis: longitud de la rueda.*

4) Los tres principales errores son:

- *La RAI1 realmente es la RAI de la EXTI0, por lo que habría que sustituir RAI1 por EXTI0_IRQHandler*
- *La RAI2 es la RAI del TIM4, por lo que habría que sustituir RAI2 por TIM4_IRQHandler*
- *La variable cero (el reset), no se vuelve a poner nunca a cero, por lo que, una vez activado, siempre se quedará en esa rama del bucle y por tanto no mostrará ninguna medida real de velocidad y distancia.*

5) Quitando menciones a la precisión de la medida de la longitud de la rueda (la cual se hace así para no hacer cálculos en aritmética de punto fijo, y sólo usar aritmética entera), el problema es que ese cálculo sólo depende del valor de radio, el cual es una constante y por lo tanto no se vuelve a cambiar. Por tanto, no es lógico que esa línea se encuentre ejecutándose cada vez que salte la RAI, sino que debería haber sido calculada la variable "seis" al principio del programa para que sólo se ejecute una vez, o incluso haberlo configurado como una constante, para evitar cualquier tipo de ejecución.

6) Los diagramas de flujo, una vez traducidas las variables, son:


PROG. PRINCIPAL
reset = false
Configura I²C
Lee radio de la memoria I²C
Activa ADC para medir temperatura Medida continua, lenta y a 8 bits
Activa IRQ para que salte con el flanco de bajada del boton de reset
Activa IRQ y configura timer para que salte en cada paso de vuelta de rueda, capturando el tiempo
tiempo_anterior = 0
reset = true
SI
tiempo_anterior=0
distancia = 0
Toma temperatura usando el ADC
MuestraVelocimetro (velocidad, distancia, temperatura)
RAI 1
Limpia Flags e inhibe IRQ
¿Se ha soltado el boton de reset?
polaridad IRQ por flanco bajada
NO
Para cuenta tiempo
Polaridad IRQ = flanco bajada
SI
Arranca tiempo a contar 2 segundos
Polaridad IRQ = flanco subida
Limpia flags y habilita IRQ
FIN
RAI 2
Es por TOC
NO
long.rueda = 2π radio
distancia += long.rueda
SI
reset = true
velocidad = long.rued. / t_activa - t_anterior
Para cuenta tiempo
t_anterior = t_actual
Limpia flag TOC
Limpia flag TIC
Limpia Flags
FIN