UNIVERSIDAD CARLOS III DE MADRID (Dpto. de Tecnología Electrónica)
Sist. Dig. Basados en Microprocesador **(Gr. Ing. Telemática)**
29 de mayo de 2013 EXAMEN FINAL (3 horas)

EJERCICIO 1 (2.0 puntos):

En una arquitectura Harvard, con una memoria de programa de 32Kx16 y una memoria de datos de 128KB siendo los datos de 8 bits, y considerando que la CPU tiene 8 registros internos para datos, y que se implementa siguiendo una filosofía Load & Store:

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:

- 4 instrucciones de transferencia de datos entre memoria y registros internos, con direccionamiento absoluto.
- 7 instrucciones aritmético/lógicas de operar entre registros, indicando en la instrucción tanto los registros operandos, como el registro que almacena el resultado.
- 8 instrucciones aritmético/lógicas con un operando dado con direccionamiento inmediato, y donde el otro operando y el registro de resultado son el mismo.
- 3 instrucciones de control con direccionamiento inherente
- 8 saltos condicionales con direccionamiento relativo a contador de programa, siendo el desplazamiento relativo de más/menos 1KB

Al ser una arquitectura Harvard el bus de direcciones y de datos de la memoria de programa es distinto de la de la memoria de datos. Los datos se direccionan con 17 bits (128KB) y lo que se direccionan son palabras de 8 bits. En cuanto a las instrucciones, se direccionan con 15 bits, y cada dirección indica la ubicación de una palabra de 16 bits.
Por lo tanto las instrucciones tienen que ser múltiplos de palabras de 16 bits. Sin embargo para el cálculo inicial se va a considerar direccionamiento a nivel de byte.
Para los opcodes, se obtiene el mínimo número de bits necesario para codificar tantos opcodes de un tipo como se indican, que respectivamente son 2, 3, 3, 2 y 3.
La dirección absoluta del dato son 17 bits.
El dato inmediato será del tamaño de los demás datos, es decir, 8 bits.
La dirección relativa de programa, que es más/menos 1K, son 11 bits, porque se necesitan 10 bits para codificar 1K y al indicar el signo se necesita un bit más.
En cuanto a los bits necesarios para codificar el registro, como hay 8 registros se utilizarán 3 bits.
La codificación inicial sería:

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Tipo1	0	0	opcode		Rd				Dirección datos															
Tipo2									1	1	1	X	opcode			Rn			Ri			Rd		
Tipo3									0	1	opcode			Rd			Dato							
Tipo4																	1	1	0	X	X	X	opcode	
Tipo5									1	0	opcode			Dirección relativa										

Sin embargo, como las instrucciones tienen que ser múltipos de palabras de instrucción (es decir, múltiplos de 16 bits, la instrucción de tipo 4, se empezará a codificar desde el bit 15, de la única palabra existente.
En cuanto a la instrucción de tipo 1, se necesitarán dos palabras, y la primera de esas dos palabras empezará por 00 (bits 23 y 23 de la tabla anterior, para así saber que se trata de una instrucción de 2 palabras. De esta forma, una codificación de las instrucciones quedaría así:

	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Tipo1	0	0	opcode			Rd		X	X	X	X	X	X	X	X	DD
(2 pal.)	Dirección datos (DD)															
Tipo2	1	1	1	X	opcode			Rn			Ri			Rd		
Tipo3	0	1	opcode			Rd		Dato								
Tipo4	1	1	0	X	X	X	X	X	X	X	X	X	X	X	opcode	
Tipo5	1	0	opcode			Dirección relativa										

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

a) Número mínimo de palabras que usa una instrucción

Tal y como se ha comentado en el apartado anterior, la instrucción más pequeña se codifica en una única palabra de 16 bits.

b) Número máximo de palabras que usa una instrucción

Igualmente, según se expone en el apartado anterior, la instrucción más larga ocupa 2 palabras de 16 bits.

c) Tamaño del Registro de Instrucción

El registro de instrucción tendrá que ocupar lo que la instrucción más grande, y por tanto deberían ser 32 bits (2 palabras), aunque se podría optimizar a que ocupase sólo 24 bits.

d) Tamaño del Contador de Programa

El contador de programa utilizará el mismo número de bits que el bus de direcciones de la memoria de programa, es decir, 15 bits.

e) Tamaño de los Registros internos

Los registros internos sólo utilizan datos, por lo que el tamaño de dichos registros es de 8 bits.

UNIVERSIDAD CARLOS III DE MADRID (Dpto. de Tecnología Electrónica)
Sist. Dig. Basados en Microprocesador **(Gr. Ing. Telemática)**
29 de mayo de 2013 EXAMEN FINAL (3 horas)

EJERCICIO 2 (4.0 puntos):

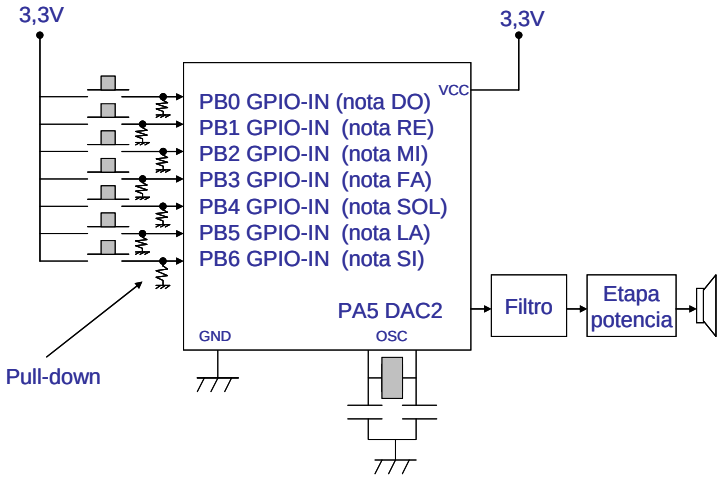
Dado el código fuente del Anexo I (que puede tener errores o carecer de algunas funciones) y sabiendo que el dispositivo es un piano electrónico simplificado (fosc=32 MHz), responda a las siguientes preguntas:

1) Indique los periféricos utilizados y describa su configuración.

- Pines PB0 al PB6 como GPIO de entrada sin PULL-UP, ni PULL-DOWN.
- DAC2 configurado con 12 bits de precisión.
- Timer4 configurado con un preescalado en microsegundos, con una interrupción por TOC asociada al canal 1 y configurada inicialmente a 1 ms (posteriormente este tiempo se modifica en función de la nota a reproducir).
- NVIC configurado para gestionar la interrupción del Timer 4, Canal 1, como TOC.

2) Dibuje el diagrama de bloques del sistema.

Dado que el programa interpreta que la pulsación se produce cuando lee un nivel alto y que los pines de entrada no están configurados como PULL-DOWN, será necesario hacerlo con hardware externo.

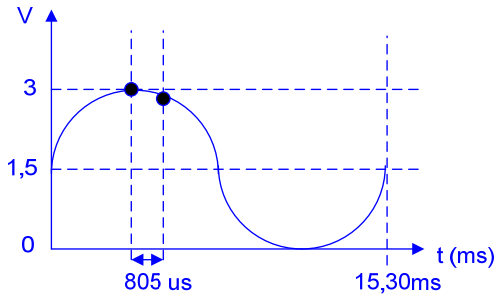


3) Indique el error de cuantificación del DAC

Dado que está configurado con 12 bits de precisión, el error será la mitad de un escalón, es decir:

Error = (3 voltios / 2^{12}) / 2 = 3 / 4096*2 = 366 uV

4) Dibuje de forma aproximada la onda de salida para la nota DO. Acote la amplitud, el periodo, el valor máximo, el valor mínimo, el valor medio y el tiempo entre dos muestras consecutivas cualquiera.



5) ¿Qué pasaría si pulsase y soltase una tecla?, ¿Qué pasaría si pulsase dos teclas a la vez y después soltase sólo una?

- Al pulsar activo una señal acústica que depende del pin asociado a la tecla (PB0 -> DO, PB1 -> RE, etc.), cuando suelto la tecla se inicia una cuenta atrás de 5 segundos, transcurridos los cuales la señal deja de reproducirse. Los 5 segundos, no son una cantidad exacta, dado que el tiempo restante se actualiza cada vez que se activa la interrupción del timer y por tanto depende del tiempo entre muestras, es decir, que siempre será superior a 5 segundos, e inferior a 5 segundos más el tiempo entre muestas.

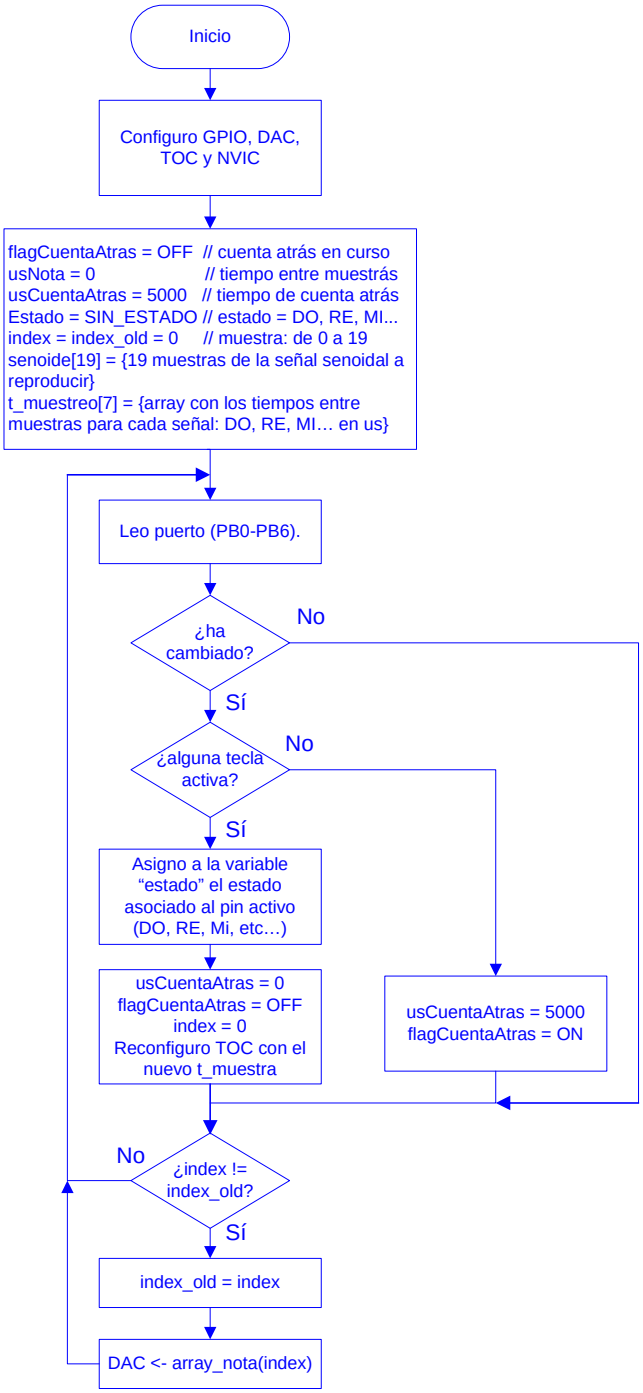
- Si pulso dos teclas, activará la señal acústica asociada a la tecla asociado al pin más bajo del puerto, dado que el filtrado lo hace la función "get_tecla" en un "if" secuencial.
  - Si suelto la tecla asociada al bit de menos peso el sistema dejará de reproducir la antigua nota y comenzará a reproducir la nota asociada a la tecla que he mantenido pulsada.
  - Sin embargo, si suelto la tecla más alta el sistema no notará cambio alguno y continuará reproduciendo la nota original.

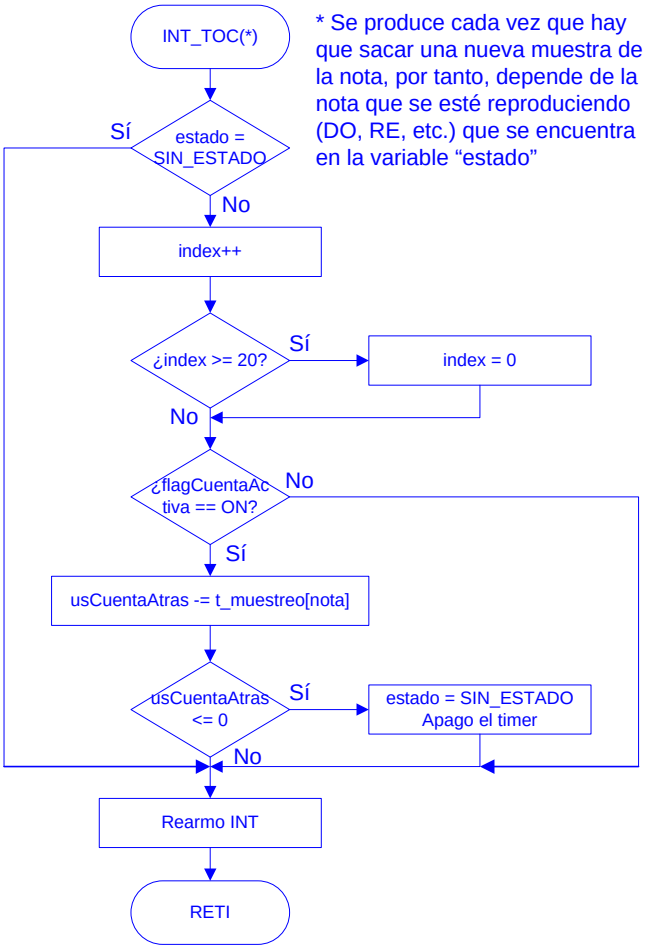
6) Dibuje el diagrama de flujo de la aplicación

UNIVERSIDAD CARLOS III DE MADRID (Dpto. de Tecnología Electrónica)

Sist. Dig. Basados en Microprocesador **(Gr. Ing. Telemática)**

29 de mayo de 2013 EXAMEN FINAL (3 horas)





* Se produce cada vez que hay que sacar una nueva muestra de la nota, por tanto, depende de la nota que se esté reproduciendo (DO, RE, etc.) que se encuentra en la variable "estado"

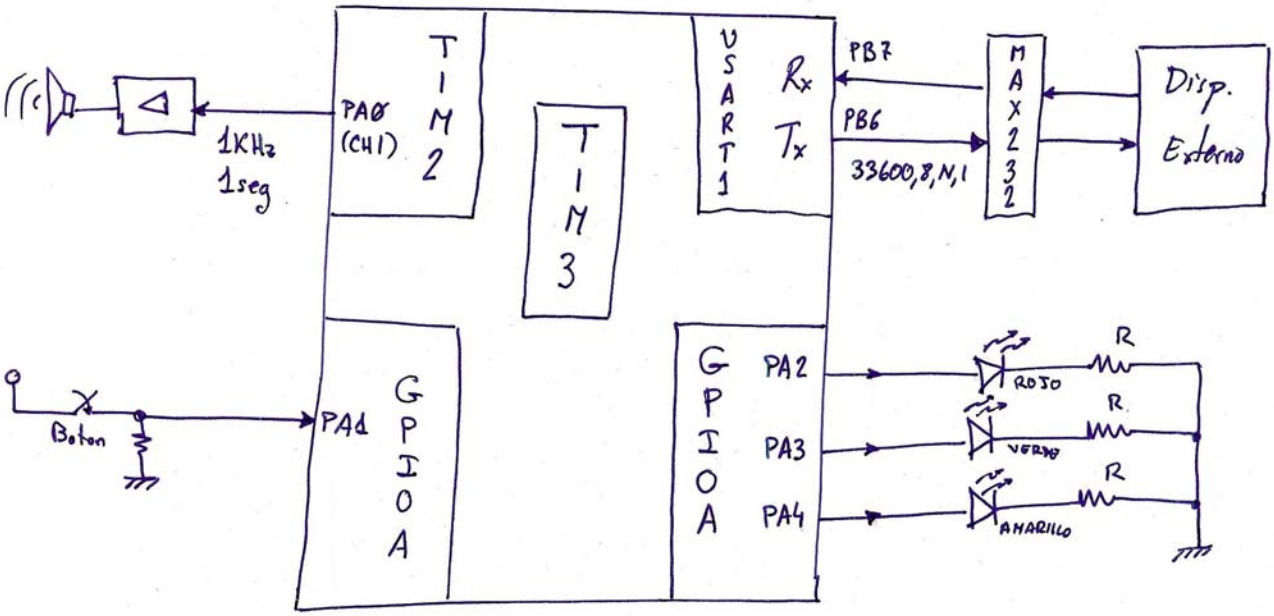
EJERCICIO 3 (4.0 puntos):

Se necesita diseñar una estación de control que se comunica con un dispositivo externo a través de un puerto serie asíncrono con características 33600,8,N,1. La comunicación se plantea como un sistema maestro esclavo, donde la estación de control es el maestro, y el dispositivo es el esclavo. Por tanto, toda comunicación con el dispositivo se iniciará por parte del maestro, mediante el envío de una trama de bytes de, al menos, 3 bytes, donde el 3º indica la longitud en número de bytes del resto del comando. Antes de enviarlo, el comando se encontrará previamente formado en una variable llamada buffer_salida. Los envíos se realizarán por espera activa.
En cuanto se ha terminado de enviar un comando, se esperará a qué el dispositivo envíe una respuesta, la cual debería empezar a llegar antes de que pasen 300ms. La respuesta está compuesta de un número de caracteres superior o igual a 1, donde el primer carácter indica el número de bytes que contendrá la respuesta. Se debe cumplir que, una vez iniciada la respuesta, el tiempo máximo entre un carácter y otro sea de 50ms.
El sistema adicionalmente dispondrá de 3 LEDs, uno rojo, uno verde y uno amarillo. El LED amarillo se encenderá mientras se esté enviando el comando. El LED verde se encenderá cuando se esté recibiendo la respuesta. El LED rojo se encenderá cuando haya habido un error de timeout, hasta que el usuario haya pulsado un botón. Tanto los LEDs como el botón son activos a nivel alto.
Además, cuando haya ocurrido un error de timeout, se generará una señal de 1KHz durante 1 segundo, para que, sacada por un pin, active un altavoz y haga sonar un pitido de 1KHz.

Con estas especificaciones, se pide diseñar la estación de control utilizando un microcontrolador ST32L152RB, teniendo en cuenta que la estación realizará muchas más operaciones, y que, por lo tanto, no puede quedarse en bucles de espera activa, salvo en el momento de la transmisión. Para ello conteste a las siguientes preguntas (si necesita hacer alguna suposición o decisión adicional sobre el diseño, indíquelo claramente):

a) Diagrama de bloques de la estación (10%)

El enunciado nos deja claro que hay que utilizar la USART tanto para recibir como para transmitir, por lo que se decide utilizar la USART1 y los pines PB7 y PB6 para dichas funciones.
En cuanto a las medidas de tiempo, realmente sólo sería necesario un temporizador, ya que todas las medidas de tiempo son secuenciales, es decir, cuando se miden los 300ms no se miden los 50ms, y cuando se genera el tono, no se realizan ninguna de las anteriores medidas. Aun así, y aprovechando que la generación del tono necesita una medida de tiempos más corta y que puede automáticamente conmutar la salida, se va a utilizar un temporizador distinto para esta función. Por lo tanto, la generación de tono se realizará por el canal 1 del TIM2, sacando la señal por PA0, mientras que el resto de tiempos se medirán con el TIM3.
En cuanto a los leds y el botón, se utilizará el GPIOA, utilizando los pines PA1 como entrada para el botón, y los PA2, PA3 y PA4 como salida para los LEDs.
Se supone, ya que no dice nada el enunciado, que el microcontrolador funciona con un reloj interno (pclk) de 32MHz (el habitualmente utilizado durante el curso).



b) ¿Necesitaría definir rutinas de atención a la interrupción? Si es que sí, indique cuáles, por qué y que función tendría. No realice aquí el diagrama de flujo. (20%)

Teniendo en cuenta que en el enunciado nos indican que salvo para la transmisión no se puede utilizar espera activa, habrá que utilizar interrupciones. Se establecen las siguientes RAIs:

- *Recepción de la USART, que hará las siguientes funciones:*
  - *Cogerá los datos recibidos*
  - *Controlará la longitud de la respuesta y avisará cuando la respuesta esté completa.*
  - *Activarla los time-out correspondientes, o los apagará cuando se haya terminado de recibir la respuesta.*
- *Control del timeout, que saltará cuando se haya producido un error de timeout y realizará:*
  - *Encender el LED rojo*
  - *Activar la generación del tono de 1KHz*
- *Control del generador del tono de 1KHz, que saltará cada vez que haya que cambiar de semiciclo del tono y realizará:*
  - *Conmutar la salida (aunque esto se hará de forma automática por la salida hardware externa*
  - *Re-configurar el momento del próximo semiciclo (t = t+0,5ms)*
  - *Controlar si se ha llegado ya a 1 segundo, y parar*

c) Configuración inicial de los periféricos involucrados, con explicación de dicha configuración. (20%)

Con lo dicho hasta la fecha, los periféricos involucrados son: GPIOA, TIM2, TIM3, USART1 y NVIC.
La configuración inicial de los mismos será la siguiente (las celdas a 0, significa que el valor no se toca respecto al valor que tuviera antes):

GPIOA: Se asignará al PA0 la funcionalidad alternativa de TIM2, al PA1 la entrada digital y a los PA2-PA4 la salida digital. Todo con la configuración por defecto ya que en este sistema no se necesitan ni velocidades restrictivas, ni salidas en colector abierto. Como la pulsación del botón la hará una persona y sólo servirá para apagar el LED rojo, no es necesaria una interrupción al poder ser inspeccionado directamente por el programa.

GPIOA MODER:

31							24	23							16
15							8	7							0
						0	1	0	1	0	1	0	0	1	0
						PA4 Salida		PA3 Salida		PA2 Salida		PA1 Entrada		PA0 AF	

GPIOA AFR:

31							24	23							16
15							8	7							0
												0	0	0	1
												PA0 como TIM2			

GPIOA OTYPER:

15							8	7							0
											0	0	0		
											Las 3 salidas por Push-pull				

GPIOA OSPEEDR:

31							24	23							16
15							8	7							0
						0	0	0	0	0	0				
						Todas salidas a velocidad por defecto									

GPIOA PUPDR:

31							24	23							16
15							8	7							0
												0	0		
												Flotante			

GPIOA BSRR:

31							24	23							16
0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
											LEDs apagados				
15							8	7							0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TIM2: Como se utiliza para sacar el tono de 1KHz (y medir 1s), tiene que contar intervalos de 0,5ms, conmutar la salida, que se actualice la funcionalidad TOC para que mida los siguientes 0,5ms, y a la vez detectar cuando se han obtenido 1s. Si se configura la cuenta en 0,01ms por paso del contador, y se inicializa el contador a 0, entonces en cada paso el CCR será incrementado en 50, y detectará el segundo cuando llegue a 100000. Como esta última cantidad es superior a lo que puede entrar en un registro de 16bits, se puede llevar esa cuenta manualmente a través de una variable, incrementándola cada vez que salte un cambio de 0,5ms, y comprobando si esa variable ha llegado a 2000. Se configurará con salida externa activa y en funcionalidad de toogle. Se dejará configurada la interrupción por TOC, pero no se arrancará el contador hasta que no sea necesario, es decir, se dejará parado.
Por simplificación del curso, los siguientes registros se dejan a 0: CR2, SMCR

TIM2 CR1:

15						8	7							0
						0	0	0	0	0	0	0	0	0
							Sin PWM							OFF

TIM2 CCMR1:

15						8	7							0
							0	0	1	1	0	0	0	0
								Salida en toogle					CH1 como TOC	

TIM2 CCER:

15						8	7							0
											0		0	1
														Salida Hw CH1

TIM2->CNT = 0; // Se pone a 0 el contador
TIM2->PSC = 319; // Con este pre-escalado, la cuenta es cada 0,01ms con pclk de 32MHz
TIM2->ARR = 0xFFFF; // Al no usar PWM no se necesita el auto-reload
TIM2->CCR1 = 50; // Se deja preparado el CCR para el primer salto a 0,5ms

TIM2 EGR:

15							8	7							0
									0		0	0	0	0	1
															Update

	15							8	7							0
TIM2 SR:				0	0	0	0					0	0	0	0	
				Limpia flags para que no salte al inicio IRQ								Limpia flags para que no salte al inicio IRQ				

	15							8	7							0
TIM2 DIER:		0		0	0	0	0	0		0		0	0	0	1	0
															IRQ CH1 ON	

TIM3: Se configurará para medir tiempos de 50ms y de 300ms, por lo que se configurará para que la cuenta sea en 1ms. No tendrá salida externa y se activará la interrupción por TOC. Dependiendo del tiempo a medir, se configurará el CCR a CCR+=50 o a CCR+=300. Se dejará parado el contador hasta que sea necesario medir estos tiempos.
Por simplificación del curso, los siguientes registros se dejan a 0: CR2, SMCR

	15							8	7							0
TIM3 CR1:							0	0	0	0	0	0	0	0	0	0
									Sin PWM							OFF

	15							8	7							0
TIM3 CCMR1:									0	0	0	0	0	0	0	0
										Sin salida					CH1 como TOC	

	15							8	7							0
TIM3 CCER:													0		0	0
																Sin Salida

TIM3->CNT = 0; // Se pone a 0 el contador
TIM3->PSC = 32000; // Con este pre-escalado, la cuenta es cada 1ms con pclk de 32MHz
TIM3->ARR = 0xFFFF; // Al no usar PWM no se necesita el auto-reload
TIM3->CCR1 = 300; // Se deja preparado el CCR en cualquier valor que no sea 0, para que no se active el flag de comparación

	15							8	7							0
TIM3 EGR:										0		0	0	0	0	1
																Update

	15							8	7							0
TIM3 SR:				0	0	0	0					0	0	0	0	
				Limpia flags para que no salte al inicio IRQ								Limpia flags para que no salte al inicio IRQ				

	15							8	7							0
TIM3 DIER:		0		0	0	0	0	0		0		0	0	0	1	0
															IRQ CH1 ON	

USART1: Se configurará, tal y como dice el enunciado, para que funcione a 33600,8,N,1, con la transmisión por espera activa, pero la recepción por interrupciones. La interrupción por recepción se podría dejar desactivada hasta que no sea necesaria (cuando haya terminado la transmisión), o dejarla activada siempre. En esta solución se dejará activada desde el principio.

	15							8	7							0
USART1 CR1:			1													
			ON													

	15							8	7							0
USART1 CR1:	0		1	0		0			0	0	1		1	1		
	Over Samp 16		ON	8bits		Sin paridad			Sin IRQ Tx	Sin IRQ Tx	IRQ Rx		Tx ON	Rx ON		

	15							8	7							0
USART1 CR2:			0	0												
			1 bit de parada													

```
USART1->CR3 = 0; // No se usa en el curso
```

Para calcular el baudrate con oversampling de 16 se hace el siguiente cálculo:

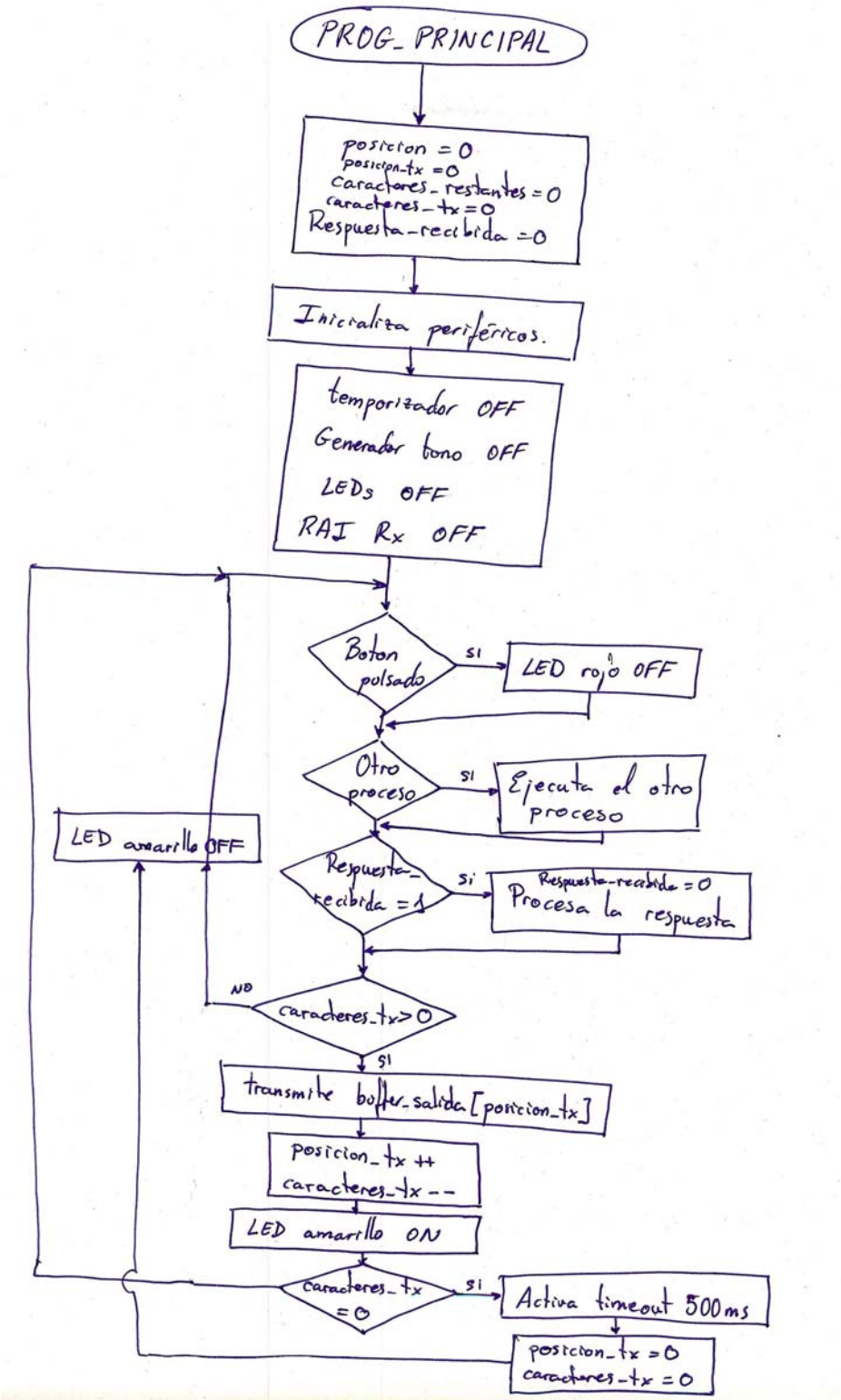
33600 = 32.000.000 / (8x2xUSARTDIV)
USARTDIV = 59,524
0,524 x 16 = 8,384
DIV_Mantisa = 59
DIV_Fraction = 8

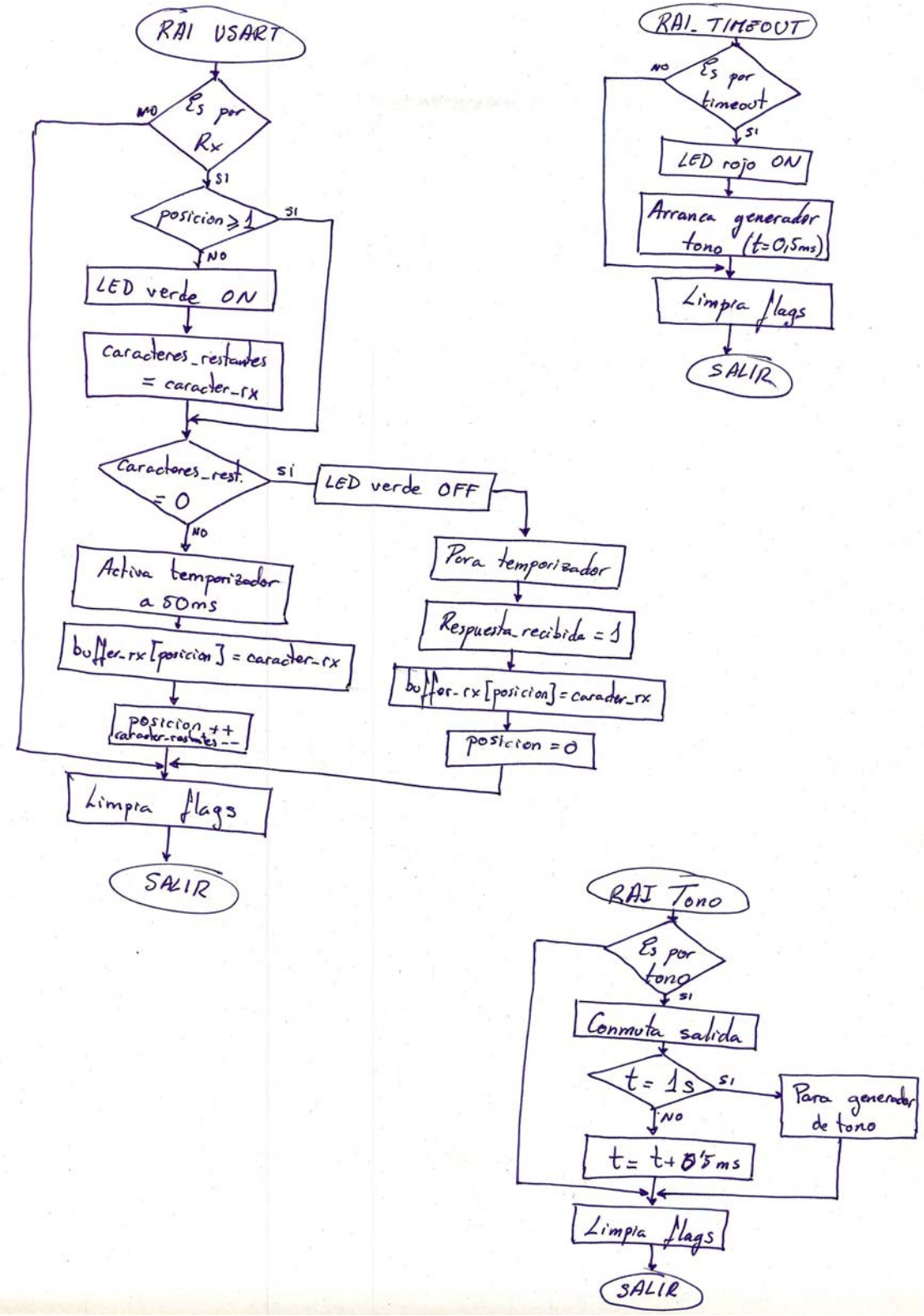
```
USART1->BRR = ((59<<4)&(~0x0F))|8;
```

NVIC: Se dejará inicialmente con todas las interrupciones activadas (TIM2, TIM3 y USART1).

```
NVIC->ISER[0] |= (1 << 28); // Se activa la IRQ por TIM2
NVIC->ISER[0] |= (1 << 29); // Se activa la IRQ por TIM3
NVIC->ISER[1] |= (1 << (37-32)); // Se activa la IRQ por USART1
```

d) Diagrama de flujo del programa principal y de todas aquellas rutinas necesarias. (50%)


PROG_PRINCIPAL
posicion = 0
posicion_tx = 0
caracteres_restantes = 0
caracteres_tx = 0
Respuesta_recibida = 0
Inicializa periféricos.
temporizador OFF
Generador tono OFF
LEDs OFF
RAI Rx OFF
Boton pulsado
SI
LED rojo OFF
Otro proceso
SI
Ejecuta el otro proceso
LED amarillo OFF
Respuesta_recibida = 1
SI
Respuesta_recibida = 0
Procesa la respuesta
NO
caracteres_tx > 0
SI
transmite buffer_salida [posicion_tx]
posicion_tx ++
caracteres_tx --
LED amarillo ON
caracteres_tx = 0
SI
Activa timeout 500ms
posicion_tx = 0
caracteres_tx = 0


RAI USART
Es por Rx
NO
SI
posicion ≥ 1
SI
NO
LED verde ON
caracteres_restantes = caracter_rx
caracteres_rest. = 0
SI
LED verde OFF
NO
Activa temporizador a 50ms
buffer_rx[posicion] = caracter_rx
posicion ++
caracter_restantes --
Limpia flags
SALIR
Para temporizador
Respuesta_recibida = 1
buffer_rx[posicion] = caracter_rx
posicion = 0
RAI_TIMEOUT
Es por timeout
NO
SI
LED rojo ON
Arranca generador tono (t=0,5ms)
Limpia flags
SALIR
RAI Tono
Es por tono
SI
Conmuta salida
t = 1s
SI
NO
Para generador de tono
t = t + 0'5 ms
Limpia flags
SALIR

Anexo I

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

/*****************************************************************************
Do 1: 65,406 Hz  -> 805 us entre muestras
                 (20 muestras -> 19 intervalos -> 1/(65,406 * 19) = 805 us)
Re 1: 73,416 Hz  -> 717 us
Mi 1: 82,407 Hz  -> 638 us
Fa 1: 87,307 Hz  -> 603 us
Sol 1: 97,999Hz  -> 537 us
La 1: 110 Hz     -> 478 us
Si 1: 123,471 Hz -> 426 us
*****************************************************************************/

#define DO              805   // us entre muestras
#define RE              717
#define MI              638
#define FA              603
#define SOL             537
#define LA              478
#define SI              426

#define ESTADO_DO       0     // estados
#define ESTADO_RE       1
#define ESTADO_MI       2
#define ESTADO_FA       3
#define ESTADO_SOL      4
#define ESTADO_LA       5
#define ESTADO_SI       6
#define SIN_ESTADO      7

#define ON              1
#define OFF             0

// Variables globales
int estado = SIN_ESTADO;
int usCuentaAtras = 5000;
int flagCuentaAtras = OFF;
int index=0;


// Valores de la señal de salida
int senoide[19] ={2048, 2713, 3306, 3762, 4033, 4089, 3924, 3555, 3023, 2385, 1711,
1073, 541, 173, 7, 63, 333, 790, 1383};

// Tiempos entre muestras para cada nota
int sample[7] = {DO, RE, MI, FA, SOL, LA, SI};
```

```
void TIM4_IRQHandler(void) {

   if ((TIM4->SR & 0x0002)!=0) {

            if (estado != SIN_ESTADO){

                  index++;
                  if(index >= 20) index = 0;

                  if(flagCuentaAtras == ON){
                        usCuentaAtras -= sample[estado];


                        if(usCuentaAtras <= 0){
                              estado = SIN_ESTADO;
                              TIM4->CR1 &= ~0x0001;
                        }
                  }

            }

            // reset timer
            TIM4->CNT = 0;
            TIM4->SR = 0x0000;   // Limpia todos los flags
      }
}


//
void config_toc4ch1(int tiempo){

   //
   TIM4->CR1 = 0x0000;
   TIM4->CR2 = 0x0000;
   TIM4->SMCR = 0x0000;
   TIM4->PSC = 31;
   //
   TIM4->CNT = 0;
   TIM4->ARR = 0xFFFF;
   TIM4->CCR1 = tiempo;
   //
   TIM4->DCR = 0;
   TIM4->DIER = 0x0002;
   // Modo de salida
   TIM4->CCMR1 = 0x0000;
   TIM4->CCMR2 = 0x0000;
   TIM4->CCER = 0x0000;
   //
   TIM4->CR1 |= 0x0001;

   TIM4->EGR |= 0x0001;
   TIM4->SR = 0;
   NVIC->ISER[0] |= (1 << 30);

}
```

```
void config_dac(void){
   //
   GPIOA->MODER |= 0x00000C00;
   DAC->CR = 0x00010000;
}

//
void config_gpio(void){
    GPIOB->MODER &= 0xC000;
}

//
int get_tecla(void){

      int tecla = -1;

    if ( (GPIOB->IDR&0x00000001) == (1<<0) ) {
            tecla = 0;
      }else if( (GPIOB->IDR&0x00000002) == (1<<1) ){
            tecla = 1;
      }else if( (GPIOB->IDR&0x00000004) == (1<<2) ){
            tecla = 2;
      }else if( (GPIOB->IDR&0x00000008) == (1<<3) ){
            tecla = 3;
      }else if( (GPIOB->IDR&0x00000010) == (1<<4) ){
            tecla = 4;
      }else if( (GPIOB->IDR&0x00000020) == (1<<5) ){
            tecla = 5;
      }else if( (GPIOB->IDR&0x00000040) == (1<<6) ){
            tecla = 6;
      }

      return(tecla);

}

int main(void){

   int index_old = index;
   int tecla = -1;
   int tecla_old = -1;

   Init_SDM();
   config_toc4ch1(1000);
   config_dac();
   config_gpio();

   while (1) {

        // coge la tecla pulsada
        tecla = get_tecla();

        // si ha cambiado
        if(tecla != tecla_old){

                  tecla_old = tecla;
```

```
                //
                if(tecla >= 0){
                      estado = tecla;
                      config_toc4ch1(sample[estado]);
                      index = 0;
                      flagCuentaAtras = OFF;

                }else{
                      //
                      usCuentaAtras = 5000;
                      flagCuentaAtras = ON;
                }

        }

      //
      if (index_old != index) {
            index_old = index;
            DAC->DHR12R2 = senoide[index];
      }
   }
}
```