SOLUCIÓN

EJERCICIO 1 (2.0 puntos):

En una arquitectura Harvard, con una memoria de programa de 16Kx16 y una memoria de datos de 64KB siendo los datos de 8 bits, y considerando que la CPU tiene 6 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:

- 3 instrucciones de transferencia de datos entre memoria y registros internos, con direccionamiento absoluto.
- 8 instrucciones aritmético/lógicas de operar entre registros, indicando en la instrucción tanto los registros operandos, como el registro que almacena el resultado.
- 4 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
- 5 saltos condicionales con direccionamiento relativo a contador de programa, siendo el desplazamiento relativo de más/menos 512B

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 16 bits (64KB) y lo que se direccionan son palabras de 8 bits. En cuanto a las instrucciones, se direccionan con 14 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, 2, 2 y 3.
La dirección absoluta del dato son 16 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 10 bits, porque se necesitan 9 bits para codificar 512 y al indicar el signo se necesita un bit más.
En cuanto a los bits necesarios para codificar el registro, como hay 6 registros se utilizarán 3 bits.
La codificación inicial sería:

	31	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	X	X	opcode																				
Tipo2									1	1	1	0	Opcode			Rn			Ri			Rd		
Tipo3									1	1	0	opcode		Rd					Dato					

Tipo4									*1*	*1*	*1*	*1*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*opcode*
Tipo5									*1*	*0*	*opcode*			*Dirección relativa*										

Sin embargo, como las instrucciones tienen que ser múltiplos 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 (bit 31 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:

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, 14 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.

EJERCICIO 2 (4.0 puntos):

Se ha desarrollado un programa para un dispositivo con un microcontrolador SMT32L5152RB (el programa figura en el Anexo I). Se sabe que se han conectado los siguientes periféricos al dispositivo:

PB0 Led blanco	PA0 Pulsador blanco
PB1 Led amarillo	PA1 Pulsador amarillo
PB2 Led azul	PA2 Pulsador azul
PB3 Led naranja	PA3 Pulsador naranja
PB4 Led rojo	PA4 Pulsador negro
PB5 Led verde	

Además se ha creado una función (invocada desde el programa) que se define como:

```
unsigned char Aleatorio(void);
```

Cuando esta función es invocada devuelve un número pseudoaleatorio comprendido entre 0 y 3.

Analiza el programa y responde a las siguientes cuestiones, indicando qué hace cada fragmento de código de una forma funcional, no describiendo cada instrucción (decir que var1 ++ incrementa var1 es algo trivial, hay que describir qué funcionalidad tiene var1 y para qué se incrementa), si una variable sólo puede tomar unos valores indica que valores son. Si algunas acciones tienen una duración definida, indícalo:

a) ¿Qué hace TIM4_IRQHandler? ¿Cuándo es llamada? (1/10).
b) Describe como están inicializados los periféricos (funcionalidad) (1/10).
c) ¿Qué se hace en el fragmento de código con el comentario `//(1)` hasta el `//(2)` no incluido? (1/10).
d) ¿Qué se hace en el fragmento de código con el comentario `//(2)` hasta el `//(3)` no incluido? (1/10).
e) ¿Qué se hace en el fragmento de código con el comentario `//(3)` hasta el final? (1/10).
f) ¿Qué hacen Func1 y Func2? (1/10).
g) Describe de una forma general qué hacen el programa y el dispositivo (3/10).

a) TIM4_IRQHandler es una RAI que se llama desde el Timer4 cada 100 ms. Comprueba que ha sido llamada por el evento del Timer y pone valores adecuados para reinicializar sus registros para otra interrupción en 100 ms. Además si la variable tiempo es distinta de 0 la decrementa. De esta manera en el programa puede inicializar dicha variable y medir intervalos de tiempo monitorizándola.

b) Los 16 bits del puerto A son inicializados como entradas estándar. En el puerto B todas las líneas se inicializan como entradas salvo las líneas PB0 .. PB5 que se inicializan como salidas.
El Timer4 se inicializa como TOC que tiene una entrada de reloj de 32 MHz, un Prescaler inicializado a 32.000 y por tanto con un reloj real de 1ms (1KHz). No activa ningún pin como salida. Pone CCR1 a 100, por lo que generará un evento cada 100 ms. Activa la salida de interrupción y la habilita en NVIC, por lo que producirá la interrupción periódica en el microcontrolador cada 100 ms.

c) Entra en un bucle mientras PA4 (pulsador negro) valga 0, y cuando pasa a 1, entra en otro bucle del que no sale hasta que la misma entrada valga 0. Por tanto se está esperando a que se pulse y se libere el pulsador negro para comenzar la actividad del programa. Finalmente se inicializa la variable mostrados a 0.

d) Tenemos un bucle principal, controlado por mostrados, que se repite 20 veces, porque entramos con mostrados = 0, y esta variable se incrementa en una unidad al final de este bucle.
Dentro de este bucle principal se llama a Func2 y se inicializa pulsaciones a 0.
Luego tenemos un bucle más interno que se repite mostrados+1 veces. Se espera a que se pulse uno de las cuatro pulsadores de colores, y si el pulsador pulsado es igual al valor que tenemos en Lista[pulsaciones], se incrementa pulsaciones. Si no coincide mostrados se pone a 19 para salir del bucle principal y pulsaciones a 30.
Finalmente se espera a que se libere la tecla pulsada y se incrementa mostrados.

e) En este fragmento de programa hay un bucle que hace que se encienda y apague un LED 5 veces a una frecuencia de 0,5 Hz, pues el LED se enciende durante un segundo y permanece apagado otro segundo.
El LED que parpadea es el rojo si pulsaciones vale 30, es decir que en el bucle descrito en d) se haya pulsado un pulsador que no coincidiera con el contenido

de Lista[pulsaciones] o el LED verde si todas las comparaciones han sido correctas.

f) Func1 inicializa las 24 posiciones del array Lista[] con valores aleatorios. Estos valores sólo pueden ser 1, 2, 4 u 8. Estos valores coinciden con los obtenidos en el puerto B al pulsar uno de los 4 pulsadores de colores.
Func2 recorre Lista[] en tantas posiciones como nos indique la variable mostrados+1. Por cada valor de Lista[] se activará una salida del puerto B. Es cada elemento de Lista[] quien contiene el n.º de LED a activar (por eso los valores 1, 2, 4 y 8) que se corresponde con cada uno de los LEDS de PB0..PB3 es decir LED de colores menos el rojo o el verde. Cada LED permanece encendido 2 segundos, situación controlada por la variable tiempo que se inicializa a 20 y se decrementa cada 0,1 s. Es decir Func2 enciende una secuencia de colores almacenada en Lista[] mostrando los primeros 'mostrados' colores.

g) Se trata del juego Simon. Se inicia cuando se pulsa y se libera el pulsador negro. Es un juego en el que se encienden una serie de 4 LEDS de colores de una forma aleatoria. Los jugadores deben pulsar los 4 pulsadores con los mismos colores, repitiendo la misma secuencia. La primera vez se muestra un color. Si el jugador lo pulsa correctamente, pasan a mostrase dos colores. Si el jugador pulsa la secuencia de 2, se muestran 3 colores y así sucesivamente. Si el jugador se equivoca en un color de la secuencia, el LED rojo parpadea 5 veces indicando que no ha llegado al final del juego. Si acierta continuamente el juego se detiene cuando se ha memorizado y pulsado correctamente una secuencia de 20 colores. Entonces parpadea 5 veces el LED verde.
Se debe volver a pulsar y liberar el botón negro para jugar otra partida.

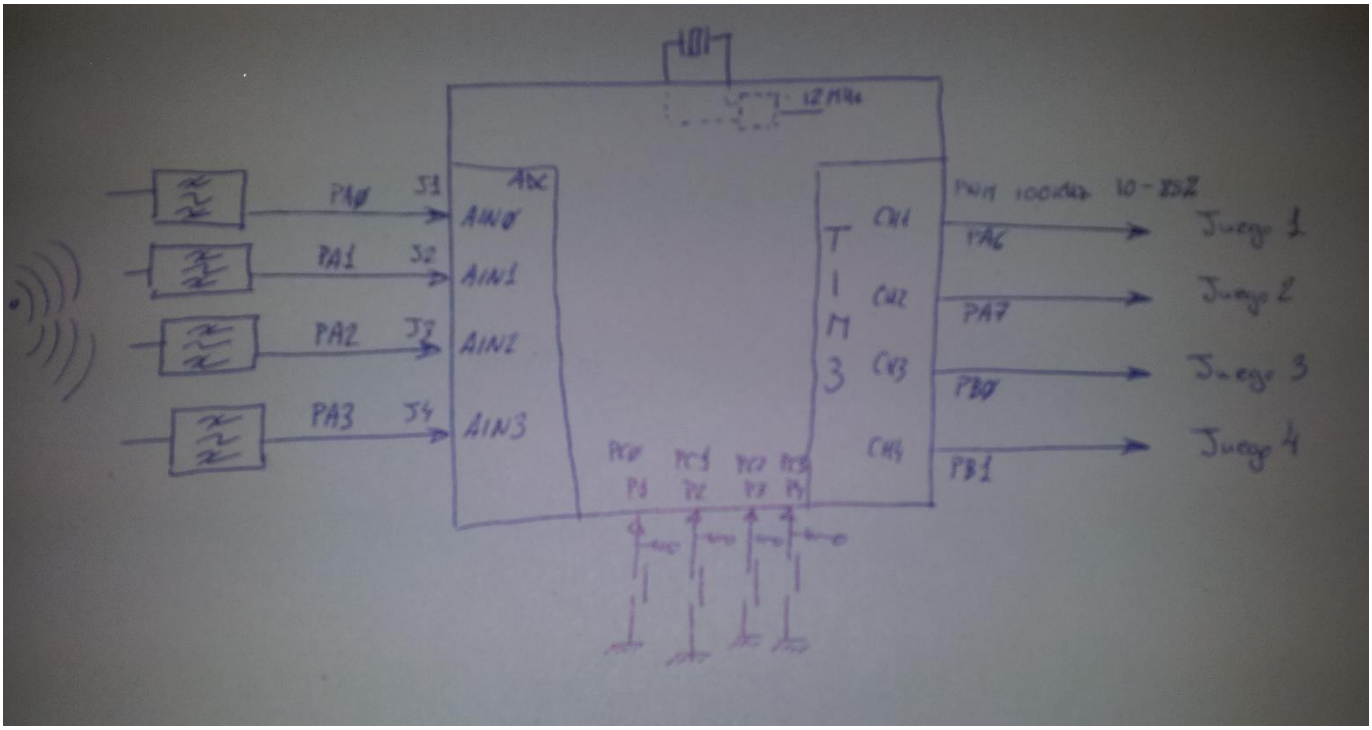
EJERCICIO 3 (4.0 puntos):

Con un microcontrolador STM32L152RB con pclk a 12MHz, sin tener por qué utilizar la placa STM32L_Discovery, se quiere diseñar un sistema de control de iluminación para una sala de fiestas. Los requisitos que nos pone el cliente son los siguientes:

- El control de iluminación estará compuesto por 4 juegos de luces.
- Cada juego de luces se controla con una misma señal PWM de 100KHz, cuyo duty cycle debe estar siempre entre el 10% y el 85%.
- La potencia luminosa de cada juego de luces responderá linealmente la intensidad sonora procedente de una banda de frecuencias:
  - Juego J1: 50 – 800 Hz
  - Juego J2: 800 – 4000 Hz
  - Juego J3: 4000 – 10000 Hz
  - Juego J4: 10000 – 20000 Hz
- Se dispone de unos filtros paso-banda externos que proporcionan a su salida un valor de continua de la intensidad sonora cada banda, siendo dicho valor de continua entre 0 y 3V.
- Adicionalmente se necesita tener un pulsador por cada juego de luces que, durante el tiempo que esté pulsado, ponga el juego de luces pulsado a máxima potencia, y retorne a su valor anterior una vez soltado el pulsador.

Con esta información, y aquellas aproximaciones o decisiones que Vd. considere necesario (siempre justificándolas) conteste a las siguientes preguntas:

a) Realice el diagrama de bloques de la solución



b) Indique si utilizaría interrupciones, y en caso afirmativo, para qué y con qué funcionalidad.

Las salidas por PWM no necesitan interrupciones, por lo que el programa simplemente se reduciría a examinar lo que ocurre en el ADC y en los pulsadores. No hay ninguna actividad adicional prioritaria y que haya que interrumpir, por lo que no son necesarias las interrupciones. Además hay que tener en cuenta que los pulsadores los acciona un ser humano y sólo tienen efecto en una parte visual, por lo que tampoco es realmente necesario considerar transitorios ni fuentes de interrupción externa (siempre se puede hacer una espera activa que atienda las pulsaciones de un usuario).

c) Configure los periféricos utilizados

TIM3: Se utilizará en configuración de TOC, con funcionalidad PWM, sacando las señales por sus cuatro canales, a través de PA6, PA7, PB0 y PB1. Como tiene que ser una señal de 100KHz (10us), y se tiene que tener resolución del 5%, habrá que ser capaz de calcular intervalos menores de 0.5us. Como el pclk es de 12MHz, no se puede obtener una medida exacta del 5%, por lo que se puede apurar lo más posible, y limitar a que el duty-cycle no sobrepase el 85%. Si no se pone pre-escalado, las cuentas serán de 83ns. De esta manera:

- *Ciclo de la señal (valor máximo del CNT) = 121 (redondeamos hacia arriba por seguridad)*
- *10% de duty-cycle = 13 (redondeamos hacia arriba por seguridad)*
- *85% de duty-cycle = 102*

Por lo tanto:

- *TIM3->CR1: ARPE=1, CEN=0, los demás a 0*
- *TIM3->CR2: todo a 0*
- *TIM3->SMCR: todo a 0*
- *TIM3->DIER: todo a 0 (sin interrupciones)*
- *TIM3->CCMR1 y TIM3->CCMR2: CCyS = 00, OCyCE=0, OCyM=110, OCyPE=1, OCyFE=0 (para y desde 1 hasta 4)*
- *TIM3->CCER: CCyNP=0, CCyP=0, CCyE=1 (para y desde 1 hasta 4)*

- *TIM3->PSC = 0 (sin pre-escalado)*
- *TIM3->ARR = 121 (el ciclo de la señal)*
- *TIM3->CCRy = 13 (para y desde 1 hasta 4)*
- *TIM3->SR = 0*
- *TIM3->EGR: UG=1 cuando haya que actualizar los registros (tras la inicialización y tras actualización del PWM)*
- *TIM3->CR1: CEN=1 (activa el contador)*

Además habría que configurar los distintos pines para que funcionaran como AF, y dentro del AF, como TIM3/4:

- *GPIOA->MODER|=0x00000001 << (2*6 +1);*
- *GPIOA->MODER&=~(0x00000001 << (2*6));*
- *GPIOA->AFR[0]=(0x02 << (6*4));*
- *GPIOA->MODER|=0x00000001 << (2*7 +1);*
- *GPIOA->MODER&=~(0x00000001 << (2*7));*
- *GPIOA->AFR[0]=(0x02 << (7*4));*
- *GPIOB->MODER|=0x00000001 << (2*0 +1);*
- *GPIOB->MODER&=~(0x00000001 << (2*0));*
- *GPIOB->AFR[0]=(0x02 << (0*4));*
- *GPIOB->MODER|=0x00000001 << (2*1 +1);*
- *GPIOB->MODER&=~(0x00000001 << (2*1));*
- *GPIOB->AFR[0]=(0x02 << (1*4));*

ADC: Hay que ponerlo en muestreo continuo, con múltiples canales (en concreto cuatro de ellos, siendo AIN0, AIN1, AIN2, AIN3), que estarán conectados al PA0, PA1, PA2, PA3, y que avise con cada conversión (no con cada secuencia). Las variaciones de frecuencia a medir serán, como mucho de 20KHz (máxima variación que percibe el oído), por lo que no es preciso un control tan exhaustivo de la medición. Habría que ver si se interrumpe en algo el cálculo interno, pero éste será solo una regla de tres para obtener el nuevo valor del PWM. En cuanto a la resolución, la medida va a imponer cambios en el DC del PWM, a nivel de tanto por uno, por lo tanto una serie de 75 valores. Esto implica que con una resolución de 8 bits es suficiente. Por lo tanto:

- *ADC1->CR2: ALIGN=0, EOCS=1 (aviso con cada conversión), DELS=000 (sin retardo), CONT=1 (conversión continua), ADON=0 (apagado)*
- *ADC1->CR1: OVRIE=0, RES=10, SCAN=0, EOCIE=0 (sin interrupciones y a 8bits)*
- *ADC1->SMPRx=0*
- *ADC1->SQR1: L=00011 (4 canales)*
- *ADC1->SQR5: SQ1=00000, SQ2=00001, SQ3=00010, SQ3=00011*
- *ADC1->CR2: ADON=1*
- *ADC1->CR2: SWSTART=1 (cuando se quieran iniciar las conversiones)*

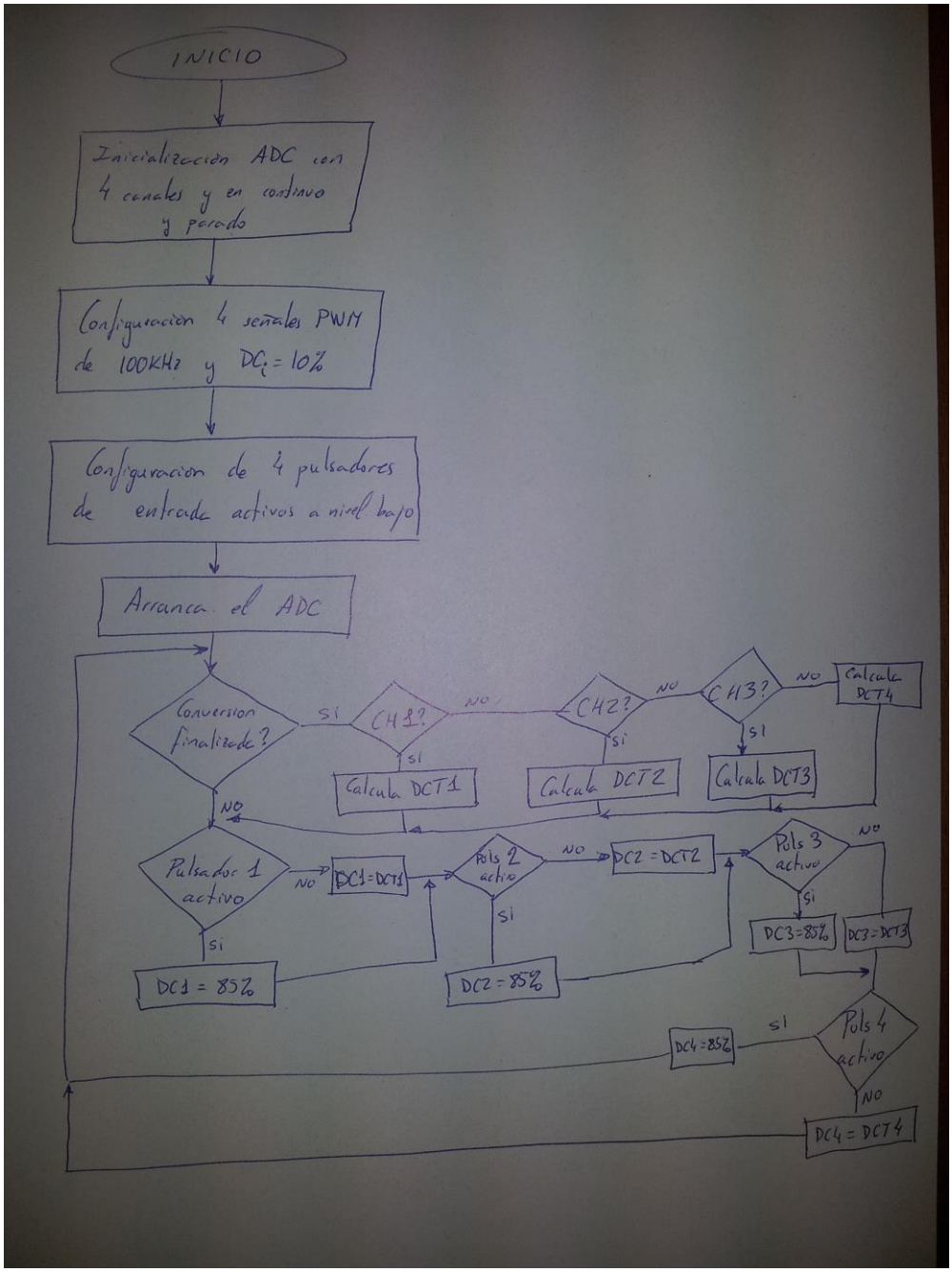
Previamente se han tenido que configurar todos los pines (PA0 – PA3) como analógicos:

- *GPIOA->MODER |= 0x0F;*

GPIO: Se configurarán los pines PC0 – PC3 como pines de entrada digital. Su información será activa a nivel bajo, tal y como se ha hecho el diagrama de bloques, por lo que si está pulsado será un 0, y si no está pulsado será un 1.

- *GPIOC->MODER &= ~(0x0F);*

d) Realice el diagrama de flujo de la solución



e) Si además se quisiera que las luces lucieran siempre de forma intermitente, con intervalos de tiempos modificables, ¿Cómo modificaría el diseño realizado?

Si no se pudiese jugar con el periodo de las señales de PWM, lo mejor sería utilizar un temporizador adicional, en modo TOC, sin salida hardware, que en ciclos alternativos, activase y desactivase la salida del PWM, asegurándose que siempre que está desactivada, se deja a 0. La frecuencia de parpadeo sería la que habría que configurar como ese otro TOC adicional.

Anexo I

```
#include "../stm32l1xx.h"
#include "../Biblioteca_SDM.h"
#include "../Utiles_SDM.h"
#include<math.h>

unsigned char tiempo = 0;

void Func1(unsigned char* array) {
    int i;
    for (i = 0; i < 20; i++)
        array[i] = pow(2,Aleatorio()); // La función pow(x,y) calcula la potencia x^y
}
void Func2(unsigned char* array, unsigned char mostrados) {
    int i;
    for (i = 0; i <= mostrados; i++) {
        tiempo = 20;
        GPIOB->ODR &= array[i];
        while (tiempo);
        tiempo = 2;
        GPIOB->ODR &= 0;
        while (tiempo);
    }
}
void TIM4_IRQHandler(void) {
    if ((TIM4->SR & 0x0002)!=0) {
        TIM4->SR &= ~0x0002;
        TIM4->CNT = 0;
        TIM4->CCR1 = 100;
        if (tiempo)
            tiempo --;
    }
}
int main(void){
    unsigned char Lista[20];
    unsigned char mostrados, pulsaciones;
    int valor, i;

    GPIOA->MODER = (0x00000000);
    GPIOB->MODER = (0x00000555);
    // El reloj que llega a TIM4 es de 32 MHz
    TIM4->CR1 = 0x0000;
    TIM4->CR2 = 0x0000;
    TIM4->SMCR = 0x0000;
    TIM4->PSC = 32000;
    TIM4->CNT = 0;
    TIM4->ARR = 0xFFFF;
    TIM4->CCR1 = 100;
    TIM4->DCR = 0;
    TIM4->DIER = 0x0002;
    TIM4->CCMR1 = 0x0000;
    TIM4->CCER = 0x0000;
    TIM4->CR1 |= 0x0001;
    TIM4->EGR |= 0x0001;
    TIM4->SR = 0;
    NVIC->ISER[0] |= (1 << 30);
```

```
/*****************************************************************************
    Comienzo del bucle principal
*****************************************************************************/
    while (1) {
        Func1(&Lista[0]);
        while (GPIOA->IDR & 0x0010);                                    // (1)
        while ((GPIOA->IDR & 0x0010) == 0);
        mostrados = 0;
        while (mostrados < 20) {                                        // (2)
            Func2(&Lista[0], mostrados);
            pulsaciones = 0;
            while (pulsaciones <= mostrados) {
                while ((GPIOA->IDR & 0x000F) == 0);
                if ((GPIOA->IDR & 0x000F) == Lista[mostrados])
                    pulsaciones ++;
                else {
                    mostrados = 19;
                    pulsaciones = 30;
                }
                while (GPIOA->IDR & 0x000F);
            }
            mostrados ++;
        }
        if (pulsaciones == 30)                                          // (3)
            valor = 0x0010;
        else
            valor = 0x0020;
        for (i=0; i<=4; i++) {
            tiempo = 10;
            GPIOB->ODR &= valor;
            while (tiempo);
            tiempo = 10;
            GPIOB->ODR = 0;
            while(tiempo);
        }
    }
}
```