Trabajo Practico N4

31/07/2017

Matías Pan - 783/9

Agustín Sánchez - 939/2

Jorge Stranieri - 917/5

Planteo general

Se realizarán un conjunto de ejercicios con el fin de entender la programación de periféricos externos con puertos de entrada/salida. Para esto utilizamos el kit dado por la cátedra.

Inciso N°1: Controladora de Led RGB

Consigna

Se desea controlar el encendido y el color de un LED RGB mediante un teclado matricial y un potenciómetro.

Para controlar la intensidad de iluminación cuando el LED esté encendido, se utilizará un potenciómetro resistivo conectado en el canal ADP2 (terminal PTA2) del ADC. Es decir, que moviendo el potenciómetro se podrá variar la tensión en la entrada del conversor y así generar el efecto deseado en la intensidad del LED.

Mediante el teclado matricial podrá encenderlo o apagarlo, controlar el color, el parpadeo o generar un barrido de colores. A continuación se muestran las funciones del teclado a implementar:

TECLA	FUNCION
*	ON
#	OFF
1	RED
2	GREEN
3	BLUE
4 y 7	RED+ y RED-
5 y 8	GREEN+ y GREEN-
6 y 9	BLUE+ y BLUE-
A y B	Velocidad+ y Velocidad-
C	Parpadeo ON-OFF
D	Barrido de colores R-G-B ON-OFF
0	WHITE

El LED RGB se conectará en configuración Ánodo común, con tres resistencias a los puertos PC1, PC2 y PC3.

El valor de las resistencias es de 470 ohm para limitar la corriente máxima en cada led a 5mA.

Interpretación

Mediante el kit provisto por la cátedra que conforman: el microcontrolador MCS9HS08, un led RGB, un teclado matricial 4x4 y un potenciómetro; se debe poder controlar el led RGB (encendido, apagado, regular la intensidad, etc) con el teclado matricial y el potenciómetro.

Los datos de los cuales disponemos son: el valor de la resistencia de cada led, que es de 470 ohm, para limitar la corriente máxima por ellos que es de 5mA. Máxima corriente de salida por pin = +- 25 mA. La corriente máxima por puerto no debe superar +- 100 mA.

Con el potenciómetro se debe poder regular la intensidad de iluminación del led. Además nosotros interpretamos que el potenciómetro define la intensidad máxima del led, debido a que con el teclado también se puede cambiar la misma. Las acciones que se solicitan son: Encender y apagar el led, encender y apagar individualmente cada color (RGB), subir y bajar la intensidad de cada color, una acción de parpadeo del led, subir y bajar la velocidad a la que parpadea, y efectuar una función de barrido de colores para el led.

Se debe tener cuidado con la configuración del microcontrolador ya que la corriente que pasa por los pines no debe exceder el máximo soportado. Dentro de la función *ledcontroller_init()* se setea PTCDS=0x0F; indicando que se setean los puertos en modo high drive por precaución que no se exceda la corriente máxima de salida.

Resolución

Conexionado de los periféricos

- El led RGB se conecta a los terminales 1,2 y 3 del Puerto C del microcontrolador.
- El teclado matricial 4x4 se conecta en los terminales del Puerto B del microcontrolador.
- El potenciómetro se conecta en el terminal 2 del puerto A del microcontrolador.

Conversor ADC (Analógico-Digital)

Permite convertir señales analogicas a valores digitales(binario) para que puedan ser procesados en el uC.

Funcionamiento: El canal seleccionado mediante ADCH es muestreado por el circuito de muestreo y retención, es decir el capacitor se carga con el valor de la tensión de entrada.

Luego el capacitor se conecta al conversor de aproximaciones sucesivas para realizar la conversión.

Este último comienza y tarda 16.5 ciclos en finalizar. El bit ADACT indica este proceso en curso.

El resultado puede seleccionarse con MODE en 8 o 10 bits y es almacenado en los registros ADCRH:ADCRL.

Luego el flag COC indica fin de conversión y puede generar un pedido de interrupción si está habilitada.

En el modo de **conversión discontinuo** luego de que finaliza la conversión el periférico se detiene.

El flag COCO se borra automáticamente cuando se lee el resultado en ADCR

Una nueva conversión puede comenzar, escribiendo el registro de estado ADCSC1 o mediante una señal de disparo en la entrada ADHWT usualmente usada con el RTC o la IRQ.

En el modo **conversión continua** la nueva conversión comienza tan pronto finaliza la anterior, una a continuación de la otra.

Configuración del ADC en nuestro trabajo:

- Modo de conversión: Continuo.
- Clock a usar: Busclock con Frecuencia F = 8 MHz.
- Formato de salida o resultado: 10 bits.
- Pin=PTA2/ADP2.
- Initialization Channel 2.

La señal recibida por el ADC es convertida a un valor digital entre 0 y 1023. Luego acotamos estos valores para ser usados en el algoritmo. Usamos una regla de tres simple para esto definimos que la intensidad máxima del led se de entre 0 y 10. Por lo tanto:

$$0 \leq ADCR \leq 1023$$

$$0 \leq Intensidad \leq 10$$

$$1023 \rightarrow 10$$

$$ADCR \rightarrow Intensidad$$

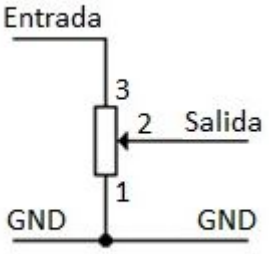
Con esto nos quedaría una función que acota el valor de conversión del ADC:

$$Intensidad = \frac{ADCR * 10}{1023}$$

De esta forma los valores usados en el código para cambiar la intensidad del led es entre 0 y 10.

Potenciómetro

Las conexiones del potenciómetro son las siguientes:



- En su entrada: V_{DD}= 5 V
- En el pin de tierra: GND
- Y a la salida: V_{out}= ADP2

El valor de la salida V_{out} vendrá dado por el divisor de tensión que se genera internamente:

$$V\,out = \frac{V\,dd\,.\,R2}{R1+R2}$$

Si consideramos al potenciómetro como lineal, a la resistencia total R y al desplazamiento relativo X, con valores entre 0 y 1, tenemos que:

$$R2 = R\,.\,X \qquad R1 = R\,.\,(1-X) \qquad V\,out = \frac{V\,dd\,.\,R\,.\,X}{R(X+1-X)} = V\,dd\,.\,X$$

Por lo que:

$$0 \leq V\,out \leq V\,dd$$

Resolución en V de la medición de tensión del potenciómetro

Como el valor de salida del conversor Analogico-Digital varia entre 0 - 1023 y el de la salida del potenciómetro lo hace entre 0 - 5V, la resolución en V vendrá dada por la división 5V/1024. Por lo tanto, la resolución de la medición de tensión es de 4.88 mV.

Generación de señales PWM

Para la generación de señales PWM individuales para cada led, las efectuamos por software. La función que se encarga de la generación de señales PWM es ***ledcontroller_pwm_handler(char led_index)***; que recibe por parámetro el led al que se debe efectuar alguna acción, debido por el cambio en alguno de sus parámetros. Para que esto funcione, modelamos a cada led individualmente con características como el estado si está prendido o no, su intensidad actual, etc.

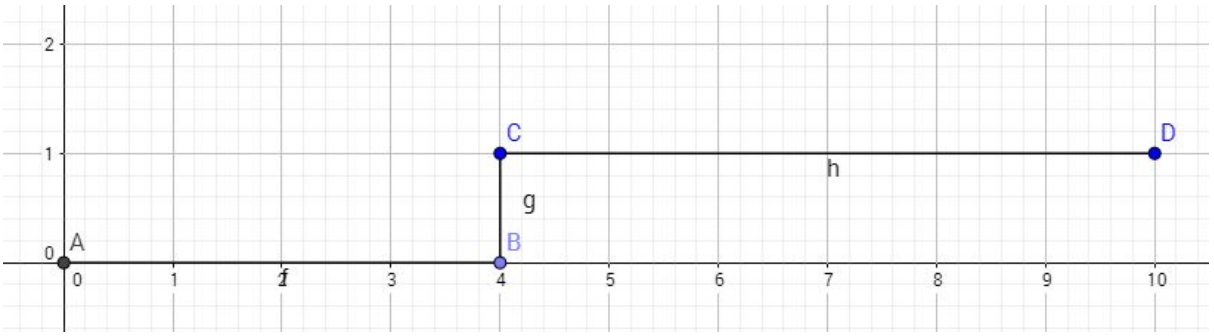
Para la generación de señales PWM definimos un período de 10 ms, teniendo así una frecuencia f = 100 Hz. Luego cada led está compuesto por su *duty_cycle*, que indica el ciclo de trabajo de cada led controlando así su intensidad, y por *cycle_iteration*, que es una variable para controlar la iteración actual de la señal por software.

Para generar el prendido y apagado del led con la intensidad deseada, por la señal PWM por software, primero calculamos la intensidad que queremos que tenga el led, que lo guardamos en una variable, para luego usarla en la generación de la señal PWM, llamada *true_intensity*= $(led.dutyCycle * ledIntensity)/INTENSITY\,SCALE$. Siendo INTENSITY_SCALE = 10.

Debido a las características del led RGB, el encendido se da cuando recibe un valor bajo en su terminal, por lo que definimos como ciclo de trabajo el porcentaje en que la señal PWM está en bajo.

Luego definimos el prendido y apagado del led dependiendo del *cycle_iteration* actual en el que se encuentre y calculando: $PWMPERIOD - trueIntensity$. De esta forma, si *cycle_iteration* es menor que este resultado, el led debe prenderse, si no debe apagarse.

Por ejemplo: teniendo *duty_cycle=8; led_intensity=5;* El valor de *true_intensity* nos quedaría igual a 4. Con el calculo anterior (PWMPERIOD - *true_intensity*) la señal nos quedaría:



Descripción de la resolución:

Para la resolución de este trabajo lo podemos dividir en tres partes importantes. La librería ***ledcontroller.c*** que es la más extensa y de la cual se ejecutan todas las acciones que deben poder verse reflejadas en el led RGB, así como también la generación de la señales PWM, y además las funciones que atienden las interrupciones están contenidas en él mayormente. Las otras atenciones a interrupciones están dadas en ***keyboard.c*** que es el mismo código que se usó para el TP2 para detectar qué tecla se presionó y en ***potenciometer.c*** que en esta librería se lee el valor del potenciómetro periódicamente, por si llega a haber algún cambio. En la librería ***main.c*** es importante destacar que se inicializa el valor del módulo del RTC con interrupción deshabilitada y se hacen las inicializaciones correspondientes de los periféricos.

Es importante aclarar que dentro de la librería ***ledcontroller.c*** existen dos grandes estados ON y OFF, y que dentro del estado ON hay 3 modos de ejecución. Definimos la arquitectura de esta manera para hacer más sencillo el razonamiento de cuándo y cómo realizar cierta acción y cuáles no deberían interferir con la actual.

No diseñamos el sistema completamente como una máquina de estados, sin embargo tiene un comportamiento similar. La principal razón por esa decisión es que quisimos que el sistema se lo mas flexible posible, por lo que casi todas las acciones o comandos pueden ejecutarse cualquiera sea el modo del estado ON, salvo pocas excepciones.

Los modos de ejecución que definimos son 4:

- OFF: Sistema completamente apagado.
- NORMAL: Control manual del led, acorde a lo que se ingrese por teclado.
- BLINKING: Titileo del led a una frecuencia determinada.
- SWEEPING: Barrido de colores prefijado.

Para una mejor abstracción a la hora de trabajar con el led RGB, definimos una estructura con los atributos necesarios para manejar cada led. La estructura de la misma es:

led_type*:* (estructura del led)

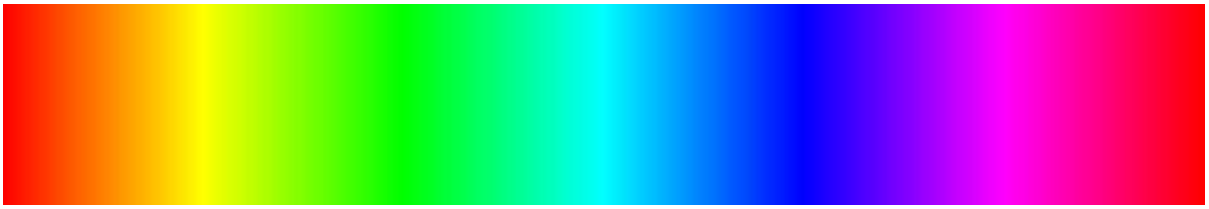
- ***duty_cycle*** (Indica el ciclo de trabajo actual y permite controlar la intensidad).
- ***cycle_iterarion*** (Iteración actual para el control de la señal PWM).
- ***state*** *(*Indica si el led está activado o no).
- ***mask*** (Es una máscara que utilizamos para abstraernos de la dirección física del puerto).

Interrupciones: La única fuente de interrupciones está controlado por el módulo RTC para que se active una vez cada 1 ms. Luego para atender las distintas atenciones a los periféricos o alguna acción a realizar utilizamos contadores dentro de la función de atención de interrupción del RTC. La única acción que se realiza cada 1 ms es la atención a la función ***ledcontroller_interrupt_handler()***. Los contadores para la atención de los periféricos son:

- Teclado: Se atiende cada 30 ms.
- Potenciómetro: Se atiende cada 25 ms.

Función Sweep

Lo que se pide en el inciso es un barrido de los colores R-G-B. Nosotros interpretamos que el barrido de los colores debería de ser una transición de intensidades entre los colores rojo, verde y azul, logrando así un mayor espectro de colores. Para ello las intensidades de cada led deberán ir disminuyendo y aumentando a medida que se da la transición de color. De esta forma pudimos lograr un barrido arcoiris, como ilustra la imagen:



El algoritmo de barrido divide el barrido en 6 partes, en las que en cada una mantiene uno de los colores con intensidad máxima, uno de los colores con intensidad mínima y el último color modificándose, por esto mismo se divide en 6 partes.

Leds

Gracias a la estructura que definimos para los leds con sus atributos, podemos manejarnos con ellos fácilmente con pasar un índice, que este indica el led en cuestión.

- RED = 0
- GREEN = 1
- BLUE = 2

Con el arreglo que creamos *rgb[ ]* manejamos los 3 colores del led. En la inicialización indicamos a los puertos correspondientes del puerto C como salida, forzamos a que comiencen en estado OFF o apagado, y además les asignamos un máscara propia a cada led. Esta máscara que le asignamos a cada led es para abstraernos posteriormente de el puerto al cual está conectado a cada led y además la utilizamos en el prendido o apagado de los leds.

Para prender/apagar los leds modificamos el valor del registro PTCD utilizando la la máscara del led el cual queremos prender/apagar. La modificación deberá aplicarse solamente en la posición del registro correspondiente al led y el resto del registro debería mantenerse igual para evitar incoherencias en el sistema. Para ello es lo que usamos la máscara correspondiente a cada led, la cual es un número de 8 bits con un 1 en la posición del led y 0 en los otros casos.

Para prender el led, usando el valor actual que tiene el PTCD en el momento y una función lógica AND con el valor de la máscara negada del led que se pasa como parámetro. De esta forma cambiamos el valor del puerto forzandolo a 0.

Por ejemplo, si suponemos la máscara del led conectado a PTC1, 00000010:

```
PTCD  xxxxxxxx
and
!mask 11111101
--------------
resul xxxxxx0x
```

Análogamente para apagar los leds usamos el mismo método, solamente que usamos una función lógica OR con el valor de la máscara del led que se pasa como parámetro. Así forzamos el valor del puerto a 1.

Por ejemplo, si suponemos la máscara del led conectado a PTC1, 00000010:

```
PTCD  xxxxxxxx
or
mask  00000010
--------------
resul xxxxxx1x
```

Función Blink

Para implementar el titileo del led simplemente se implementaron dos funciones: **blink_toggle**, la cual intercambiará entre el modo NORMAL y BLINKING según corresponda; y **ledcontroller_blink_handler** la cual se encarga de la acción de titileo en sí. Esta última se encarga de intercambiar los estados de titileo con el periodo actual, el cual variará según lo desee el usuario. Para lograr el titileo simplemente se intercambia entre dos estados apagado y prendido, los cuales no modifican el estado previo del led sino que, si se está en modo de tilteo apagado → no se atiende la señal PWM, manteniendo el led apagado. Esto genera que el usuario pueda activar o desactivar el led sin importar que se encuentre en este modo y que además al entrar o salir del modo BLINKING se conserven las configuraciones del led.

Función Vel Up & Down

Las funciones para aumentar la frecuencia del titileo o disminuirlo simplemente modifican el periodo de la función detallada previamente. Definimos un valor máximo y mínimo del periodo, además de un valor en el que se modifica el periodo, para que sea más simple trabajar en un ambiente controlado.

Dichos valores son (en ms):

- **BLINK_MAX:** 1000
- **BLINK_MIN:** 100
- **BLINK_STEP:** 300

Función Intensity UP & Down

Las funciones para subir y bajar la intensidad de los leds mediante el teclado, cambian el ciclo de trabajo o porcentaje de intensidad del led.

El potenciómetro indica el límite de intensidad máxima de todo el sistema, mientras que el ciclo de trabajo de cada led lo que indica es el porcentaje de intensidad entre 0 y ese valor máximo. Por lo que el porcentaje que el ciclo de trabajo de cada led representa se puede tomar como un porcentaje relativo y no absoluto.

Anexo: Archivos de C

Para una visión más detallada del código, consultar el repositorio: https://github.com/trorik23/CDyuC-TP4 .

MCUInit.c(solo el codigo agregado manualmente por nosotros):

```
/* User declarations and definitions */
#include "ledcontroller.h"
#include "keyboard.h"
#include "potentiometer.h"

// variables para controlar el llamado a funciones con su periodo correspondiente.
// Funcionan a manera de contador
char keyboard_iterations=0;
char potentiometer_iterations=0;
/*   Code, declarations and definitions here will be preserved during code generation
*/
/* End of user declarations and definitions */


// isrVrtc es la funcion que sera llamada gracias a la interrupcion de RTC
// cada 1 ms debido a las configuraciones iniciales. Se comporta de manera de
// llamar a las funciones que necesitan correr periodicamente y en particular
// cada una con su periodo.
// keyboard_check_key: cada KEYBOARD_CHECK_PERIOD ms
// ledcontroller_interrupt_handler: cada 1 ms
// potentiometer_interrupt_handler: cada POTENTIOMETER_PERIOD ms
// En particular, las ultimas dos solo son llamadas si el estado de
// ledcontroller se encuentra en ON
__interrupt void isrVrtc(void)
{
  keyboard_iterations++;
  if(keyboard_iterations==KEYBOARD_CHECK_PERIOD){
      keyboard_check_key();
      keyboard_iterations=0;
  }
  if(ledcontroller_is_on()){
      potentiometer_iterations++;
      if(potentiometer_iterations==POTENTIOMETER_PERIOD){
          potentiometer_interrupt_handler();
          potentiometer_iterations=0;
      }
      ledcontroller_interrupt_handler();//se llama cada 1 ms
  }

  // se indica que se atendio la interrupcion
  RTCSC_RTIF=1;
}
```

Main.c

```c
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "keyboard.h"
#include "ledcontroller.h"
#include "potentiometer.h"

#ifdef __cplusplus
extern "C"
#endif
void MCU_init(void); /* Device initialization function declaration */

void main(void) {
    MCU_init();
    // Inicializamos el Real Time Counter para que se realiza una interrupcion cada 1 ms
    // En particular, se inicializa con las interrupciones apagadas.
    RTCSC=0x88;

    // Inicializamos los modulos del sistema
    keyboard_init();
    potentiometer_init();
    ledcontroller_init();

    // Habilitamos la interrupcion del RTC, para dar comienzo al sistema
    for (;;) {
        // Iteramos indefinidamente llamando a ledcontroller_run para permitir
        // el funcionamiento constante del controlador del led.
        ledcontroller_run();
    }
}
```

Keyboard.h

```c
#ifndef KEYBOARD_H_
#define KEYBOARD_H_

// KEYBOARD_CHECK_PERIOD indica el periodo en milisegundos con la que se va a llamar
// a la rutina keyboard_check_key
#define KEYBOARD_CHECK_PERIOD 30

// keyboard_check_key se encarga de implementar el algoritmo de
// identificacion de la tecla presionada. Si se detecta el presionado
// y liberado de una tecla se pushea el valor de la tecla al buffer
// keyevent.
void keyboard_check_key(void);

// keyboard_init inicializa las configuraciones del puerto B
// en el que esta conectado el teclado matricial
void keyboard_init(void);

#endif
```

Keyboard.c

```c
#include <mc9s08sh8.h>
#include "keyevent.h"
#include "keyboard.h"

// NOT_PRESSED es un valor de control utilizado para las funciones internas de keyboard
// que indica que la columna o fila no esta presionada
#define NOT_PRESSED 42 //meaning of life

// EMPTY_KEY es lo correspondiente a un valor nulo
const char EMPTY_KEY = ' ';

// variables internas utilizadas para la identificacion de la tecla presionada
char last_pressed_key;
char pressed_row;
char pressed_column;

// mapeo de teclas correspondiente al teclado matricial
const char keymap[4][4] = { { '1', '2', '3', 'A' },
                            { '4', '5', '6', 'B' },
                            { '7', '8', '9', 'C' },
                            { '*', '0', '#', 'D' } };

void check_column(void);
void check_row(void);

// keyboard_init inicializa las configuraciones del puerto B
// en el que esta conectado el teclado matricial
void keyboard_init(void){
    PTBDD = 0x0F;
    PTBPE = 0xF0;
    last_pressed_key=EMPTY_KEY;
}

// keyboard_check_key se encarga de implementar el algoritmo de
// identificacion de la tecla presionada. Si se detecta el presionado
// y liberado de una tecla se pushea el valor de la tecla al buffer keyevent
void keyboard_check_key(void) {
    check_column();
    if (pressed_column == NOT_PRESSED) {
        if (last_pressed_key == EMPTY_KEY)
            return;
        keyevent_push(last_pressed_key);
        last_pressed_key = EMPTY_KEY;
        return;
    }
    check_row();
    if (pressed_row == NOT_PRESSED) {
        return;
    }
    last_pressed_key = keymap[pressed_row][pressed_column];
}
```

```
52.
53. // check_row de forma analoga, devuelve en pressed_row la fila correspondiente a la
54. // tecla presionada. Realiza la deteccion, realizando un "barrido" de filas.
55. void check_row(void) {
56.     char current_row = 0;
57.     while (current_row < 4) {
58.         PTBD=0x0F;
59.         switch(current_row) {
60.         case 0:
61.             PTBD_PTBD0=0;
62.             break;
63.         case 1:
64.             PTBD_PTBD1=0;
65.             break;
66.         case 2:
67.             PTBD_PTBD2=0;
68.             break;
69.         case 3:
70.             PTBD_PTBD3=0;
71.             break;
72.         }
73.         check_column();
74.         if(pressed_column!=NOT_PRESSED) {
75.             pressed_row=current_row;
76.             PTBD=0x00;
77.             return;
78.         }
79.         current_row++;
80.     }
81.     pressed_row = NOT_PRESSED;
82. }
83.
84. // check_column realiza el primer chequeo para la identificacion de la tecla
85. // presionada. Devuelve en pressed_column la columna correspondiente a la
86. // tecla presionada, si es que lo hubiese.
87. void check_column(void) {
88.     if (PTBD_PTBD7 == 0) {
89.         pressed_column = 3;
90.         return;
91.     }
92.     if (PTBD_PTBD6 == 0) {
93.         pressed_column = 2;
94.         return;
95.     }
96.     if (PTBD_PTBD5 == 0) {
97.         pressed_column = 1;
98.         return;
99.     }
100.        if (PTBD_PTBD4 == 0) {
101.            pressed_column = 0;
102.            return;
103.        }
104.        pressed_column = NOT_PRESSED;
105.    }
```

Keyevent.h

```h
#ifndef KEYEVENT_H_
#define KEYEVENT_H_
// KEYEVENT acutua como un buffer de eventos de teclas presionadas.
// En particular, como buffer de un solo caracter a la vez

// keyevent_push almacena la tecla c, pasa por parametro, en el buffer
void keyevent_push(char);

// keyevent_pop retira del bufffer y devuelve la ultima tecla presionada
char keyevent_pop(void);

// leyevent_is_empty devuelve 1 si no hay ningun evento para consumir.
// Devuelve 0, en caso contrario
char keyevent_is_empty(void);

#endif
```

Keyevent.c

```c
#include "keyevent.h"

// key almacena el evento y actua como un buffer de un solo caracter
static char key;

// empty indica si hay un evento(tecla presionada) en el buffer que deberia ser consumido.
// empty=1 indica que esta vacio, que no hay evento.
char empty = 1;

// keyevent_push almacena la tecla c pasa por parametro, en el buffer
void keyevent_push(char c) {
    key = c;
    empty = 0;
}

// keyevent_pop retira del bufffer y devuelve la ultima tecla presionada
char keyevent_pop() {
    empty = 1;
    return key;
}

// leyevent_is_empty devuelve 1 si no hay ningun evento para consumir.
// Devuelve 0, en caso contrario
char keyevent_is_empty() {
    return empty;
}
```

Potentiometer.h

```
#ifndef POTENTIOMETER_H_
#define POTENTIOMETER_H_

// POTENTIOMETER_PERIOD indica el periodo en milisegundos con la que se va a llamar
// a la potentiometer_interrupt_handler
#define POTENTIOMETER_PERIOD 25

// potentiometer_init encargado de inicializar con la configuracion correspondiente
// al ADC integrado
void potentiometer_init(void);

// potentiometer_interrupt_handler es el encargado de leer el valor del registro del
// conversor analogico-digital y actualizar la intensidad de ledcontroller en
// la escala correspondiente
void potentiometer_interrupt_handler(void);

#endif
```

Potentiometer.c

```
#include "potentiometer.h"
#include "mc9s08sh8.h"
#include "ledcontroller.h"

// MAX_VALUE indica el valor maximo que se puede leer del potenciometro,
// correspondiente a las configuraciones del conversor analogico-digital
#define MAX_VALUE 1023

// potentiometer_init encargado de inicializar con la configuracion correspondiente
// al ADC integrado
void potentiometer_init(){
    // bus clock 8Mhz
    // modo de conversion continuo
    // data format 10 bits
    // pin=PTA2/ADP2
    // inicializacion channel 2
    APCTL1=0x00U;
    ADCCFG=0x08U;
    ADCCV=0x00U;
    ADCSC2=0x00U;
    ADCSC1=0x22U;
}

// potentiometer_interrupt_handler es el encargado de leer el valor del registro del
// conversor analogico-digital y actualizar la intensidad de ledcontroller en
// la escala correspondiente
void potentiometer_interrupt_handler(){
    // calcular intensidad
    // 0 < ADCR (valor sensor) < MAX_VALUE
    // 0 <   intensidad led    < INTENSITY_SCALE
    // donde ITENSITY_SCALE es un valor de escala exportado por ledcontroller
    ledcontroller_set_intensity((ADCR*INTENSITY_SCALE)/MAX_VALUE);
}
```

Ledcontroller.h

```h
#ifndef LEDCONTROLLER_H_
#define LEDCONTROLLER_H_

// INTENSITY_SCALE exporta la escala en que se trabajo la intensidad maxima del sistema.
// 10: 100%
// 0 :  0%
#define INTENSITY_SCALE 10

// ledcontroller_init se encarga de las configuraciones iniciales de ledcontroller. Inicializa
// los puertos en que se conecta cada led (PTC pines 1,2 y 3) e inicializa todas las variables
// internas necesarias. Se empieza en el estado apagado.
void ledcontroller_init(void);

// ledcontroller_run es el encargado de leer los eventos de teclas presionadas.
// En base a la tecla presionada y si el ledcontroller esta apagado o no, deriva
// la accion a sus funcion interna correspondiente
void ledcontroller_run(void);

// ledcontroller_interrupt_handler funcion que es llamada cada INTERRUPT_PERIOD, osea 1 ms en el caso normal.
// Se encarga de los llamados a las funciones del control del PWM de cada led, del barrido y del blinkeo,
// segun corresponda.
void ledcontroller_interrupt_handler(void);

// ledcontroller_set_intensity establece la intensidad maxima recibida por parametro. El parametro
// debe ser coherente a la escala de la intensidad dada por INTENSITY_SCALE
void ledcontroller_set_intensity(char);

// ledcontroler_is_on devuelve si el estado del ledcontroller esta apagado o prendido. Permite la validacion
// de si se debe llamar ledcontroller_interrupt_handler en caso de que el sistema este prendido.
// 1: prendido(NORMAL,BLINKING o SWEEPING)
// 0: apagado(OFF)
char ledcontroller_is_on(void);

#endif
```

Ledcontroller.c

```c
#include <mc9s08sh8.h>
#include "ledcontroller.h"
#include "keyevent.h"
// todos los periodos definidos en ms

// INTERRUPT_PERIOD indica cada cuanto se llama ledcontroller_interrupt_handler
// Todos lo periodos indicados para cada funcionalidad del sistema, deberian ser
// multiplos es INTERRUPT_PERIOD
```

```
9.  #define INTERRUPT_PERIOD 1
10. 
11. #define PWM_PERIOD 10 // periodo de la señal PWM
12. 
13. // MAX_DUTY_CYCLE indica el valor correspondiente a ciclo de trabajo 100%
14. #define MAX_DUTY_CYCLE (PWM_PERIOD/INTERRUPT_PERIOD)
15. #define MIN_DUTY_CYCLE 0
16. 
17. #define SWEEP_COLOR_PERIOD 50 // tiempo activo de cada color en el barrido
18. 
19. #define BLINK_MAX 1000 // periodo maximo de blinkeo
20. #define BLINK_MIN 100 // periodo minimo de blinkeo
21. #define BLINK_STEP 300 // valor de salto al modificar el periodo del blinkeo
22. 
23. // valores correspondientes a cada color dentro del arreglo de leds rgb[]
24. #define RED 0
25. #define GREEN 1
26. #define BLUE 2
27. 
28. void state_on(void);
29. void state_off(void);
30. void set_white(void);
31. void blink_vel_down(void);
32. void blink_vel_up(void);
33. void blink_toggle(void);
34. void sweep_toggle(void);
35. void led_activate(char);
36. void led_desactivate(char);
37. void led_toggle_state(char);
38. void led_on(char);
39. void led_off(char);
40. void led_intensity_down(char);
41. void led_intensity_up(char);
42. void ledcontroller_pwm_handler(char);
43. void ledcontroller_blink_handler(void);
44. void ledcontroller_sweep_handler(void);
45. 
46. // current_state indica el modo en que se encuentra el ledcontroller
47. // OFF: sistema completamente apagado
48. // NORMAL: controlar manual del led, acorde a las teclas presionadas
49. // BLINKING: titileo del led a una frecuencia determinada
50. // SWEEPING: barrido de colores prefijado
51. typedef enum{OFF,NORMAL,BLINKING,SWEEPING} state;
52. state current_state;
53. 
54. // estructura para el almacenamiento de informacion necesaria de cada led
55. struct led_type{
56.     char duty_cycle; // indica el ciclo de trabajo actual. Permite controlar la
    intensidad individual del led
57.     char cycle_iteration; // iteracion actual para el control de PWM por software
58.     char state; // indica si el led esta apagado o prendido. state=1 >> prendido
59.     char mask; // mascara para permitir el control de los puertos correspondientes a
    cada led. Permite abstraer la direccion fisica del puerto
60. };
```

```
61.
62. // rgb es un arreglo para contener la informacion de los 3 leds Red, Green y Blue
63. struct led_type rgb[3];
64.
65. // led_intensity indica la intensidad total del sistema. Establece el % maximo de
    intensidad en
66. // la que el ledcontroller puede trabajar. Sera el que se vera modificada por el
    potenciometro
67. char led_intensity;
68.
69. // variables correspondientes para el control de blinkeo
70. int blink_period = 700;
71. unsigned int blink_interrupt_counter;
72. char unsigned blink_state;//1->ON
73.
74. // variables correspondientes para el control de barrido de colores
75. unsigned int sweep_interrupt_counter;
76. char sweep_color_counter;
77.
78. // variable auxiliar para la lectura de eventos de tecla
79. static char key;
80.
81.
82. // ledcontroller_run es el encargado de leer los eventos de teclas presionadas.
83. // En base a la tecla presionada y si el ledcontroller esta apagado o no, deriva
84. // la accion a sus funcion interna correspondiente
85. void ledcontroller_run(){
86.     if(keyevent_is_empty())return;
87.     key=keyevent_pop();
88.     /*
89.         KEY MAP
90.         =======
91.            *    --> state_on()
92.            #    --> state_off()
93.            A    --> blink_vel_up()
94.            B    --> blink_vel_down()
95.            C    --> blink_toggle()
96.            D    --> sweep_toggle()
97.            0    --> set_white()
98.          R/G/B
99.          1/2/3  --> led_toggle_state(led)
100.            4/5/6  --> led_intensity_up(led)
101.            7/8/9  --> led_intensity_down(led)
102.       */
103.       if(current_state==OFF){
104.           // en el modo OFF, se ignoran todas las teclas menos la
105.           // correspondiente a la tecla de encender el ledcontroller
106.           if(key!='*') return; //ignorar todo menos ON
107.           state_on();
108.       }else{
109.           if(key=='*') return; //ignorar los ON
110.           switch(key){
111.               case '#': state_off();      break;
112.               case 'A': blink_vel_up();   break;
```

```
113.                 case 'B': blink_vel_down(); break;
114.                 case 'C': blink_toggle();   break;
115.                 case 'D': sweep_toggle();   break;
116.                 case '0': set_white();      break;
117.                 case '1':
118.                 case '2':
119.                 case '3': led_toggle_state(((key-'0')-1) % 3); break;
120.                 case '4':
121.                 case '5':
122.                 case '6': led_intensity_up(((key-'0')-1) % 3); break;
123.                 case '7':
124.                 case '8':
125.                 case '9': led_intensity_down(((key-'0')-1) % 3);break;
126.                 default: /*error*/ break;
127.             }
128.         }
129.     }
130.
131.     // state_on cambia el estado actual al estado NORMAL. Al intercambiarse entre los
    modos prendidos
132.     // y apagado del sistema, la configuracion independiente de cada led se conserva,
    no se resetea.
133.     void state_on(void){
134.         current_state=NORMAL;
135.     }
136.
137.     // state_off apaga completamente el sistema, podiendo volver a prenderse solo con
    la tecla
138.     // correspondiente al encendido. Apaga forzadamente los leds por si estaban a la
    mitad del
139.     // control de PWM. No desactiva ni desconfigura nada, permitiendo que al
    reanudarse se mantengan
140.     // los estados en que estaba antes cada led.
141.     void state_off(void){
142.         current_state=OFF;
143.         led_off(RED);
144.         led_off(BLUE);
145.         led_off(GREEN);
146.     }
147.
148.     // set_white setea el led RGB al color blanco. Fuerza el estado NORMAL, lo que
    significa que desactiva
149.     // el barrido o el titileo si es que se encontraban activos. Para prender en
    blanco, se activan y se
150.     // setea la intensidad de cada led al maximo.
151.     void set_white(void){
152.         current_state=NORMAL;
153.         rgb[RED].duty_cycle=MAX_DUTY_CYCLE;
154.         rgb[GREEN].duty_cycle=MAX_DUTY_CYCLE;
155.         rgb[BLUE].duty_cycle=MAX_DUTY_CYCLE;
156.         led_activate(RED);
157.         led_activate(GREEN);
158.         led_activate(BLUE);
159.     }
```

```
160.
161.    // ledcontroller_init se encarga de las configuraciones iniciales de
    ledcontroller. Inicializa
162.    // los puertos en que se conecta cada led (PTC pines 1,2 y 3) e inicializa todas
    las variables
163.    // internas necesarias. Se empieza en el estado apagado.
164.    void ledcontroller_init(void){
165.        current_state=OFF;
166.        // configurar los puertos como salida
167.        PTCDD_PTCDD1=1;
168.        PTCDD_PTCDD2=1;
169.        PTCDD_PTCDD3=1;
170.        // modo high drive
171.        PTCDS=0x0F;
172.        // setear las mascaras de cada led al bit que corresponden en PTC.
173.        // Esto nos permite abstraer la direccion fisica de cada led y utilizar una
    sola
174.        // funcion para el prendido y apagado de los leds
175.        rgb[RED].mask=PTCD_PTCD1_MASK;
176.        rgb[GREEN].mask=PTCD_PTCD2_MASK;
177.        rgb[BLUE].mask=PTCD_PTCD3_MASK;
178.        // arrancan todos los leds en su maxima intensidad
179.        rgb[RED].duty_cycle=MAX_DUTY_CYCLE;
180.        rgb[GREEN].duty_cycle=MAX_DUTY_CYCLE;
181.        rgb[BLUE].duty_cycle=MAX_DUTY_CYCLE;
182.        // se activan los los leds
183.        rgb[RED].state=1;
184.        rgb[GREEN].state=1;
185.        rgb[BLUE].state=1;
186.        // se fuerzan las salidas de los leds a apagado, por si se encontraban activas
187.        led_off(RED);
188.        led_off(GREEN);
189.        led_off(BLUE);
190.    }
191.
192.    // led_desactivate desactiva y fuerza el apagado del led indicado. Recibe como
    parametro el
193.    // indice del led a apagar dentro del arreglo rgb.
194.    void led_desactivate(char led_index){
195.        rgb[led_index].state=0;
196.        led_off(led_index);
197.    }
198.
199.    // led_activate activa el led indicado. Recibe como parametro el indice del led
200.    // a prender dentro del arreglo rgb. No hace falta forzae a "prendido" porque el
    mismo
201.    // pwm lo va a prender
202.    void led_activate(char led_index){
203.        rgb[led_index].state=1;
204.    }
205.
206.    // led_toggle_state activa o desactiva el estado del led indicado. Recibe como
    parametro el
```

```
207.    // indice dentro del arreglo rgb. Realiza el cambio de estado con las funciones
    led_activate
208.    // y led_desactivate segun su estado actual
209.    void led_toggle_state(char led_index){
210.        if(rgb[led_index].state) led_desactivate(led_index);
211.        else led_activate(led_index);
212.    }
213.
214.    // led_on prende el led indicado modificando la salida correspondiente del puerto
    PTC.
215.    // Recibe como parametro el indice dentro del arreglo rgb.
216.    // Esto lo logra de forma general, utilizando la mascara de cada led. Dado las
    caracteristicas
217.    // del led RGB, el led se prender con el valor 0
218.    void led_on(char led_index){
219.        PTCD= PTCD & (~rgb[led_index].mask);
220.        /*
221.        PTCD  xxxxxxxx
222.        and
223.        !mask 11111101
224.        --------------
225.        resul xxxxxx0x
226.        */
227.    }
228.
229.    // led_off apaga el led indicado de forma analoga a led_on. Led apagado se da con
    el valor HIGH
230.    void led_off(char led_index){
231.        PTCD = PTCD | rgb[led_index].mask;
232.        /*
233.        PTCD  xxxxxxxx
234.        or
235.        mask  00000010
236.        --------------
237.        resul xxxxxx1x
238.        */
239.    }
240.
241.    // led_intensity_up aumenta la intensidad del led indicado por parametro. Aumentar
    la intensidad
242.    // significa aumentarle el ciclo de trabajo de la señal PWM. Se limita a
    MAX_DUTY_CYCLE.
243.    // En particular te deja cambiar la intensidad ya sea en modo NORMAL y BLINKING,
    pero no en
244.    // SWEEPING dado que generaria inconsistencias en el barrido de colores.
245.    void led_intensity_up(char led_index){
246.        if(current_state==SWEEPING) return;
247.        if(rgb[led_index].duty_cycle==MAX_DUTY_CYCLE) return;
248.        rgb[led_index].duty_cycle++;
249.    }
250.
251.    // led_intensity_down disminuye la intensidad del led indicado por parametro. Se
    resuelve de forma
252.    // analoga a led_intensity_down
```

```
253.    void led_intensity_down(char led_index){
254.        if(current_state==SWEEPING) return;
255.        if(rgb[led_index].duty_cycle==MIN_DUTY_CYCLE) return;
256.        rgb[led_index].duty_cycle--;
257.    }
258.
259.    // ledcontroller_interrupt_handler funcion que es llamada cada INTERRUPT_PERIOD,
    osea 1 ms en el caso normal.
260.    // Se encarga de los llamados a las funciones del control del PWM de cada led, del
    barrido y del blinkeo,
261.    // segun corresponda.
262.    void ledcontroller_interrupt_handler(void){
263.        if(current_state==SWEEPING)ledcontroller_sweep_handler();
264.        if(current_state==BLINKING)ledcontroller_blink_handler();
265.        // No atualiza, ni prende, los leds si es que se encuentra en modo titileo
266.        // y en el estado de titileo apagado
267.        if(current_state==BLINKING && blink_state==0) return;
268.        if(rgb[RED].state)ledcontroller_pwm_handler(RED);
269.        if(rgb[GREEN].state)ledcontroller_pwm_handler(GREEN);
270.        if(rgb[BLUE].state)ledcontroller_pwm_handler(BLUE);
271.
272.    }
273.
274.    // ledcontroller_pwm_handler se encarga de la generacion de la señal PWM por
    software, del led indicado por paramtro.
275.    // Permite controlar la intensidad del led de acuerdo a su ciclo de trabajo y a la
    maxima intensidad del
276.    // led RGB en general, dada por led_intensity.
277.    // Utiliza cycle_iteration para controlar, en cada iteracion dentro de un mismo
    perido de señal PWM,
278.    // el apagado o encendido del led segun corresponda.
279.    void ledcontroller_pwm_handler(char led_index){
280.        /* ejemplo de funcionamiento
281.            Asumiendo duty_cycle: 8(80%) y led_intensity: 5(50%)
282.            Por lo que true_intensity: 40%
283.            Iteracion: 0123456789012345678 9
284.            Señal:     ____------____------
285.            Notar que el ciclo de trabajo indica el % en que la señal esta activa,
286.            y en nuestro caso estar activa significa estar en 0
287.        */
288.        char true_intensity=(rgb[led_index].duty_cycle*led_intensity)/INTENSITY_SCALE;
289.
    if(rgb[led_index].cycle_iteration<(PWM_PERIOD/INTERRUPT_PERIOD-true_intensity))
290.            led_off(led_index);
291.        else
292.            led_on(led_index);
293.        rgb[led_index].cycle_iteration++;
294.        if(rgb[led_index].cycle_iteration==PWM_PERIOD/INTERRUPT_PERIOD)
    rgb[led_index].cycle_iteration=0;
295.    }
296.
297.    // ledcontroller_set_intensity establece la intensidad maxima recibida por
    parametro. El parametro
298.    // debe ser coherente a la escala de la intensidad dada por INTENSITY_SCALE
```

```
299.    void ledcontroller_set_intensity(char intensity){
300.        led_intensity=intensity;
301.    }
302.
303.    // ledcontroller_blink_handler se encarga de intercambiar los estados de titileo
    con el periodo
304.    // indicado por blink_period. Dicho periodo variara segun las indicaciones del
    usuario. Para lograr
305.    // el titieo se intercambia entre blink_state=0 (apagado) y
    blink_state=1(prendido). Al apagar los leds
306.    // no se desactivan sino que simplemente se fuerza la salida del led a apgado, y
    no se prenderan dado que
307.    // ledcontroller_interrupt_handler no atendera la generacion de la señal PWM si se
    encuentra en este
308.    // estado de titileo. Esto genera que el usuario pueda activar y desactivar cada
    led independientemente,
309.    // sin importar que se encuentre en este modo.
310.    void ledcontroller_blink_handler(void){
311.        if(blink_interrupt_counter++<(blink_period/INTERRUPT_PERIOD))return;
312.        blink_interrupt_counter=0;
313.        if(blink_state){
314.            blink_state=0;
315.            led_off(RED);
316.            led_off(GREEN);
317.            led_off(BLUE);
318.        }else{
319.            blink_state=1;
320.        }
321.    }
322.
323.    // blink_toggle cambia entre los modos NORMAL y BLINKING segun en que modo se
    encontraba.
324.    // Si se activa el modo BLINKING, hace las inicializaciones correspondientes.
325.    void blink_toggle(void){
326.        if(current_state!=BLINKING){
327.            current_state=BLINKING;
328.            //init blink
329.            blink_interrupt_counter=0;
330.            blink_state=1;
331.
332.        }else{
333.            current_state=NORMAL;
334.        }
335.    }
336.
337.    // blink_vel_down disminuye la frecuencia de titileo
338.    void blink_vel_down(void){
339.        if(blink_period==BLINK_MAX)return;
340.        blink_period+=BLINK_STEP;
341.    }
342.
343.    // blinkc_vel_up aumenta la frecuencia de titileo
344.    void blink_vel_up(void){
345.        if(blink_period==BLINK_MIN)return;
```

```
346.        blink_period-=BLINK_STEP;
347.    }
348.
349.    // ledcontroler_is_on devuelve si el estado del ledcontroller esta apagado o
    prendido. Permite la validacion
350.    // de si se debe llamar ledcontroller_interrupt_handler en caso de que el sistema
    este prendido.
351.    // 1: prendido(NORMAL,BLINKING o SWEEPING)
352.    // 0: apagado(OFF)
353.    char ledcontroller_is_on(void){
354.        if(current_state==OFF) return 0;
355.        else return 1;
356.    }
357.
358.    // sweep_toggle cambia entre los modos NORMAL y SWEEPING segun en que modo se
    encontraba.
359.    // Si se activa el modo SWEEPING, hace las inicializaciones correspondientes. A
    diferencia
360.    // del modo BLINKING, este estado si fuerza la activacion de todos los leds, para
    lograr el
361.    // efecto de arcoiris deseado. De todas maneras, luego de la inicializacion, se
    permitira al
362.    // usuario activar o desactivar cada led por separado. No se permitira modificar
    sus intensidades
363.    // individuales.
364.    void sweep_toggle(void){
365.        if(current_state!=SWEEPING){
366.            current_state=SWEEPING;
367.            //init sweep
368.            sweep_interrupt_counter=0;
369.            sweep_color_counter=0;
370.            led_activate(RED);
371.            led_activate(GREEN);
372.            led_activate(BLUE);
373.            rgb[RED].duty_cycle=MAX_DUTY_CYCLE;
374.            rgb[GREEN].duty_cycle=MIN_DUTY_CYCLE;
375.            rgb[BLUE].duty_cycle=MIN_DUTY_CYCLE;
376.        }else{
377.            current_state=NORMAL;
378.        }
379.    }
380.
381.    // ledcontroller_sweep_handler se encarga de barrido de colores en forma de
    arcoiris, intercalando
382.    // por todas las combinaciones de colores RGB (de forma simplificada). Cada uno de
    los colores se
383.    // va a mantener un tiempo definido por SWEEP_COLOR_PERIOD.
384.    void ledcontroller_sweep_handler(void){
385.        if(sweep_interrupt_counter++<(SWEEP_COLOR_PERIOD/INTERRUPT_PERIOD))return;
386.        sweep_interrupt_counter=0;
387.        // El algoritmo de barrido divide el barrido en 6 partes, en las que en
388.        // cada una mantiene uno de los colores con intensidad maxima, uno de los
    colores con
389.        // intensidad minima y el ultimo color modificandose.
```

```
390.
391.        //no importa si estan prendidos o no-->barrido
392.        switch(sweep_color_counter/MAX_DUTY_CYCLE){
393.            case 0:  rgb[GREEN].duty_cycle++; break;//0-9
394.            case 1:  rgb[RED].duty_cycle--;   break;//10-19
395.            case 2:  rgb[BLUE].duty_cycle++;  break;//20-29
396.            case 3:  rgb[GREEN].duty_cycle--; break;//30-39
397.            case 4:  rgb[RED].duty_cycle++;   break;//40-49
398.            case 5:  rgb[BLUE].duty_cycle--;  break;//50-59
399.        }
400.        sweep_color_counter=(sweep_color_counter+1) % (MAX_DUTY_CYCLE*6);
401.
402.        /* ejemplo
403.            Tomando como ejemplo que los duty_cycles solo pueden ser 0-3
404.            Iteracion: 0 1 2|3 4 5|6 7 8|9 10 11|12 13 14|15 16 17|0 1 2...
405.            ------------------------------------------------------------------
406.            R:         3 3 3|2 1 0|0 0 0|0  0  0| 1  2  3| 3  3  3|3 3 3...
407.            G:         1 2 3|3 3 3|3 3 3|2  1  0| 0  0  0| 0  0  0|1 2 3...
408.            B:         0 0 0|0 0 0|1 2 3|3  3  3| 3  3  3| 1  2  3|0 0 0...
409.        */
410.    }
```