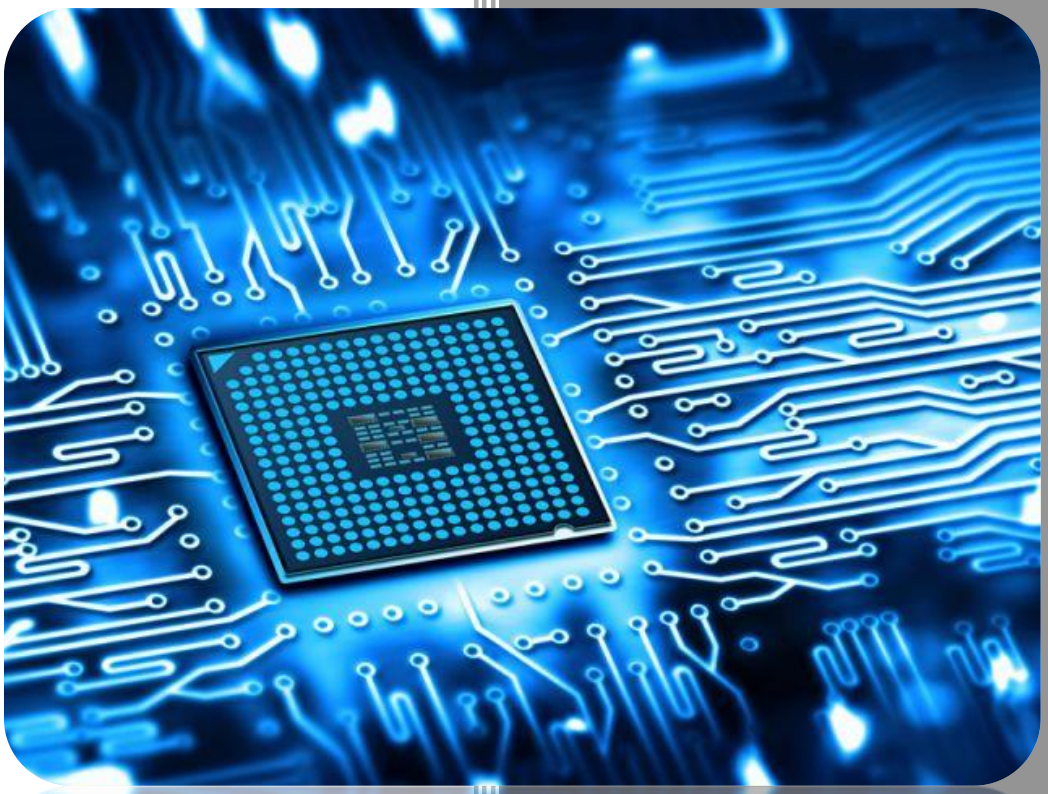
TP1

Circuitos digitales y microcontroladores



UNIVERSIDAD
NACIONAL
DE LA PLATA

Arreche, Cristian Carlos 01515/4
Blasco, Federico Matías 01678/4
15-4-2019

Control de periféricos externos con puertos de entrada/salida

Índice

1. Conexión de LEDs al microcontrolador

1.1. Problema

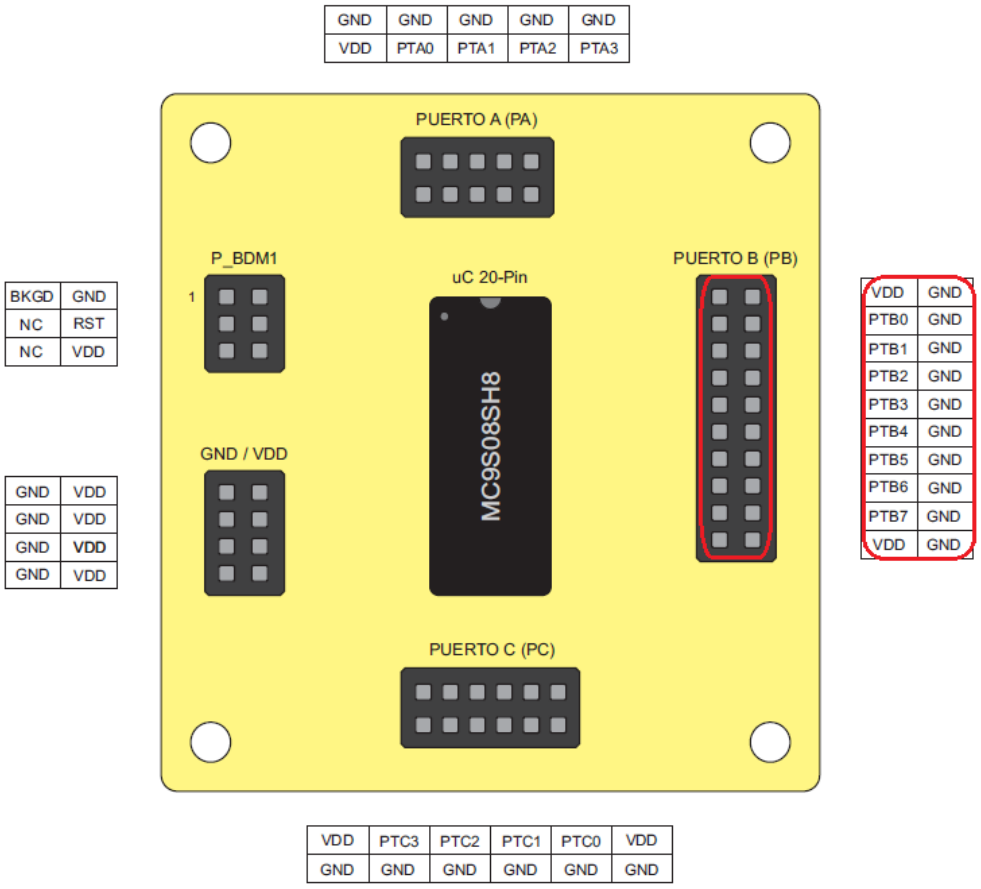
Se desea conectar 8 LEDs de diferentes colores al PORTB del microcontrolador. Realice el esquema eléctrico de la conexión.

Calcule la resistencia serie para que la corriente por cada LED no supere la capacidad de corriente de cada salida individual del PORTB y de todas las salidas en funcionamiento simultáneo.

1.2. Interpretación

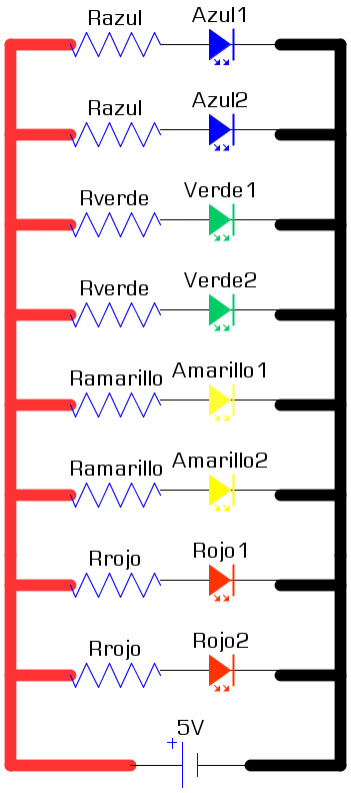
Dado el microcontrolador MC9S08SH8 CPJ (incluido en el kit) y un conjunto de 8 LEDs, se deberá resolver y analizar el ejercicio provisto por la cátedra.

El objetivo es realizar la conexión correspondiente de dichos LEDs al puerto B del microcontrolador, el cual se puede observar en el siguiente esquema de la placa:



1.3. **Resolución**

Para un mejor entendimiento del conexionado que se realizará, a continuación se puede observar el esquema eléctrico del mismo:



Por otro lado, el puerto B de la placa es controlado por los siguientes registros de datos:

	7	6	5	4	3	2	1	0
R W	PTBD7	PTBD6	PTBD5	PTBD4	PTBD3	PTBD2	PTBD1	PTBD0
Reset:	0	0	0	0	0	0	0	0

Port B Data Register (PTBD)

Donde, observando la hoja de datos, podemos ver que cada pin individual del puerto B soporta una corriente máxima de 5mA. Por lo tanto, hay que establecer una resistencia adecuada para que la corriente por cada LED no supere este máximo, ya que más allá de este límite se puede dañar permanentemente el dispositivo.

Entonces:

$$I_l = \frac{V_o - V_D}{R} \quad < \quad 5mA$$

Despejando:

$$R > \frac{V_o - V_D}{I_{Imáx}} = \frac{5V - V_D}{0.005A}$$

Siendo V_0 la tensión de alimentación de la placa (5V) y V_D la caída de tensión de cada LED, que varía según su color. Los valores de estos últimos (brindados por la cátedra) son los siguientes:

Led azul → V_D = 2.6V

Led verde → V_D = 1.9V

Led amarillo → V_D = 1.8V

Led rojo → V_D = 1.8V

De esta manera, claramente, podemos ver que la resistencia serie para que la corriente por cada LED no supere la capacidad de corriente de cada salida individual del puerto B, será distinta para cada LED. Reemplazando V_D en la última expresión para R, llegamos a los siguientes valores:

R_{azul} = 480 Ω

R_{verde} = 620 Ω

$R_{amarillo}$= 640 Ω

R_{rojo} = 640 Ω

Luego, observando nuevamente la hoja de datos, podemos ver que la corriente máxima total soportada por el dispositivo es de 120 mA. Siendo 8 pines, si todos se encuentran prendidos en simultáneo, la corriente máxima que circule será:

$$I_{total} = \sum_{i=1}^{8} I_{imáx} = 8 * 5mA = 40mA$$

De esta manera, podemos concluir que por más que estén todos los LEDs prendidos en simultáneo, no se excederá el límite de corriente máxima soportada por el dispositivo, y no habrá riesgo de dañarlo.

2. Encendido y apagado de LEDs del mismo color

2.1. Problema

Realice un programa que permita encender y apagar alternadamente los LEDs del mismo color. Realice una función que implemente un retardo adecuado para permitir la visualización.

Simule con el IDE cuántos ciclos de reloj consume el algoritmo de retardo y calcule el tiempo en ms, suponiendo un reloj de 8MHz. Saque conclusiones sobre las ventajas y desventajas de este tipo de retardo.

2.2. Interpretación

Dado el microcontrolador MC9S08SH8 CPJ (incluido en el kit) y un conjunto de 8 LEDs, se deberá resolver y analizar el ejercicio provisto por la cátedra.

Dicho ejercicio se basa en el encendido y apagado de los LEDs del mismo color de manera alternada, donde habrá que tener en cuenta el tiempo de retardo para cada par de LEDs con el fin de que pueda apreciarse y distinguirse claramente dicha secuencia.

Además, habrá que contemplar el momento en el que se llega al último par de LEDs del mismo color, donde habrá que "reiniciar" la secuencia.

2.3. Resolución

Para llevar a cabo la resolución del problema, se programó el código correspondiente en el IDE CodeWarrior. Donde las tareas se ejecutarán en el lazo infinito del main, con el fin de que la secuencia pueda ser apreciada de manera continua mientras el microcontrolador se encuentre conectado.

El pseudocódigo para la resolución del programa es el siguiente:

```
Apagar todos los LEDs
      Repetir infinitamente
            Para cada par de LEDs del mismo color
                  encender LEDs
                  esperar 200mS.
                  apagar LEDs
```

Como mencionamos anteriormente, el objetivo del retardo es, en este caso, la correcta visualización de los LEDs. En un primer momento, dicho retardo había sido implementado con un bucle for vacío (estructura de control iterativa) de la siguiente manera:

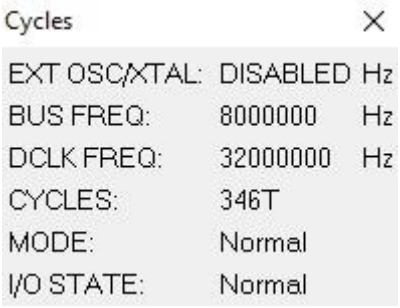
```
for(cant=n; cant>0; cant--);
```

Donde la variable *cant* es inicializada en la cantidad de veces que se quiera ejecutar dicho bucle. Por lo tanto, cada vez que quiera usarse un retardo de X unidades de tiempo, sería necesario calcular, aproximadamente, qué valor de n será el correcto para contar con ese tiempo de retardo suponiendo un reloj de 8MHz. Lo cual resulta muy tedioso y complejo.

Por este motivo, y al tratarse de una función que utilizaremos con bastante frecuencia para realizar los trabajos de las prácticas, decidimos implementar una función llamada `delay([tiempo])`, donde directamente se puede pasar por parámetro el retardo que se desee en milisegundos. De esta manera, es mucho más práctico la utilización de retardos, ya que, una vez implementada dicha función, no hay que realizar ningún tipo de cálculo para estimar X cantidad de tiempo.

Para lograr esto, realizamos un bucle for con n=15, de manera de poder determinar la cantidad de instrucciones de assembler que se ejecutan en dicho

bucle (el valor n=15 fue elegido de manera de hacer un promedio entre varias iteraciones).



Como podemos observar, dicha ejecución del for (n = 15) necesita 346 ciclos de reloj, o sea, que la ejecución de 1 bucle for necesita de, aproximadamente, 346/15 = 23 ciclos de reloj. Como se realizan 8000 ciclos de reloj por milisegundo, se realizarán, nuevamente de manera aproximada, 348 ejecuciones del for por milisegundo.

De esta manera, el código de dicha función será el siguiente:

```
void delay(unsigned short retardo){
      unsigned short temp;
      unsigned short cant;

      for(cant=retardo; cant>0; cant--){
            for(temp=348; temp>0; temp--);
      }
}
```

Donde el parámetro recibido es directamente el tiempo en ms de retardo a utilizar. Por lo tanto, por cada milisegundo de retardo que se desee, se ejecutará un bucle for de 348 iteraciones.

Este tipo de retardos presenta una gran desventaja, y es que los cálculos realizados anteriormente son válidos únicamente para el microcontrolador que se está utilizando (MC9S08SH8) o, en su defecto, para aquellos que cuenten con una misma frecuencia de reloj de 8MHz. Caso contrario, si la frecuencia de reloj del microcontrolador que se esté utilizando es distinta, deberán realizarse nuevamente estos cálculos.

A su vez, como mencionamos anteriormente, la manera para calcular el retardo en unidades de tiempo involucra determinar las sentencias de assembler para el bucle for, lo que no es algo muy preciso, ya que estas varían para, por ejemplo, la inicialización del bucle y las iteraciones próximas a esta. Por lo que hay que tener en cuenta un error de conversión en la función *delay()*.

3. Encendido y apagado secuencial de LEDs

3.1. Problema

Realice un programa que permita encender los LEDs conectados al puerto B en la siguiente secuencia de bits: 0-1-2-3-4-5-6-7 y vuelva a empezar.

3.2. **Interpretación**

Dado el microcontrolador MC9S08SH8 CPJ (incluido en el kit) y un conjunto de 8 LEDs, se deberá resolver y analizar el ejercicio provisto por la cátedra.

De la misma manera que en el punto anterior (2.), habrá que tener en cuenta un tiempo de retardo para la correcta visualización de cada LED, pero ahora el objetivo será ir realizando un corrimiento de bits de manera que se vaya respetando la secuencia que plantea el problema.

3.3. **Resolución**

Considerando que el esquema de conexión y la configuración del puerto B es la misma que el punto anterior, la modificación que se realizará consiste únicamente en cómo va a ser la secuencia de visualización de encendido y apagado de los LEDs. Esta se realizará de manera automática por software con X cantidad de tiempo por igual para cada LED

Partiendo del pseudocódigo, el objetivo del programa es:

```
Apagar todos los LEDs
Repetir infinitamente
      Para cada LED conectado
            encender LED
            esperar 200mS.
            apagar LED
```

Aprovechando el operador de corrimiento lógico de bits de C, basta con una sola instrucción para respetar la secuencia de encendido solicitada.

Nuevamente habrá que contemplar el caso en que el LED encendido sea el último y habrá que “reiniciar” dicha secuencia.

4. Conexión de un pulsador mecánico al microcontrolador

4.1. **Problema**

Se desea conectar un pulsador a una entrada digital del microcontrolador. Investigue los posibles esquemas de conectar un pulsador y determine el algoritmo más adecuado para detectar en cada caso cuándo el pulsador se presiona y cuándo se suelta. ¿Qué es una resistencia de pull up? ¿y de pull down?

Investigue sobre el efecto de rebote que produce el pulsador y los métodos para eliminar este efecto por software.

4.2. **Interpretación**

Dado el microcontrolador MC9S08SH8 CPJ (incluido en el kit), un pulsador mecánico y un conjunto de 8 LEDs, se deberá resolver y analizar el ejercicio provisto por la cátedra.

El objetivo del ejercicio es comprender las posibles maneras de conectar el pulsador mecánico al microcontrolador, así como también analizar los

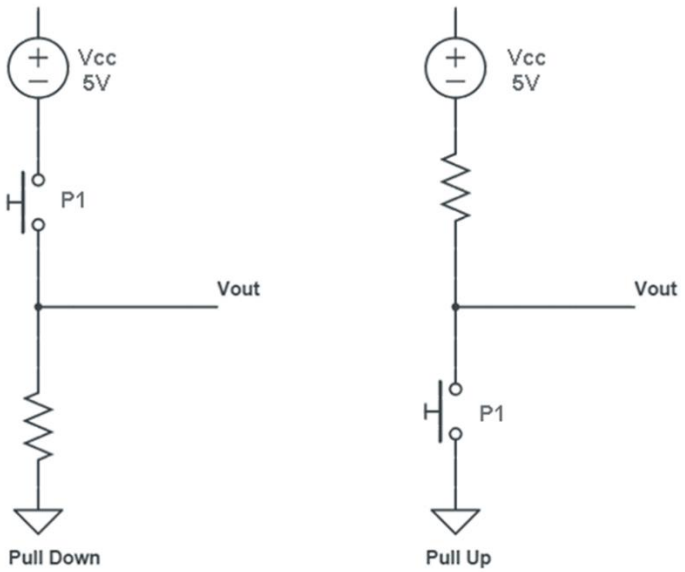
problemas, ventajas y desventajas que tiene cada una de estas conexiones y sus posibles soluciones.

Además, al no tratarse de un pulsador ideal, la transición entre los estados del pulsador (0 y 1) no es perfecta. Es decir, cuando el pulsador es presionado una única vez, puede simular que fue presionado varias veces seguidas de manera repetitiva, generando un comportamiento incorrecto. A este problema, se lo denomina efecto rebote.

4.3. Resolución

En cuanto a las resistencias mencionadas, lo primero a destacar es que no son un tipo de resistencias especiales, sino que se tratan de resistencias comunes pero que están dispuestas en un circuito de una determinada manera. Dependiendo de la configuración se determinará si la resistencia es pull up o pull down.

Como funcionalidad básica, estas resistencias establecen un estado lógico en un pin o entrada de un circuito cuando este se encuentra en estado de reposo. Como bien indica su nombre, la resistencia pull up establece un estado HIGH en dicha entrada y las resistencias pull down establecen un estado LOW. Esto evita los falsos estados que se producen por el ruido generado en los circuitos electrónicos. En la siguiente imagen vemos cómo deberíamos situar las resistencias para cada modo:



4.3.1. Resistencia pull up

Establece un estado HIGH en reposo, con el pulsador abierto, no hay paso de corriente y la entrada digital ve directamente la fuente de alimentación, es decir los 5V.

Cuando pulsamos el interruptor, cerramos el circuito y por tanto la entrada digital está directamente conectada a masa. La resistencia es necesaria en este montaje para no cortocircuitar la fuente.

El pseudocódigo para este tipo de resistencias sería:

```
Si la entrada del pulsador esta en LOW
      Delay(20ms)//Para el rebote
      Mientras la entrada del pulsador siga en LOW
            //No hacer nada
      Delay(20ms)//Para el rebote
      //Hacer el procesamiento correspondiente cuando se
      acciona el pulsador
```

4.3.2. **Resistencia pull down**

Por el contrario, establece un estado LOW en reposo, con el interruptor abierto no hay paso de corriente. Por lo tanto, no hay caída de potencial en la resistencia y la entrada digital "ve" masa, es decir está en LOW.

Cuando pulsamos y cerramos el circuito, se produce una caída de tensión en la resistencia por el paso de la corriente. Esta caída de tensión, que son los 5V directos de la fuente es lo que ve nuestra entrada digital, y por tanto estará en HIGH mientras esté pulsado.

En este tipo de montajes se suele usar resistencias de $10K\Omega$ para controlar el consumo de corriente del circuito.

El pseudocódigo para este tipo de resistencias sería:

```
Si la entrada del pulsador esta en HIGH
      Delay(20ms)//Para el rebote
      Mientras la entrada del pulsador siga en HIGH
            //No hacer nada
      Delay(20ms)//Para el rebote
      //Hacer el procesamiento correspondiente cuando se
acciona el pulsador
```

Una vez analizado esto, es posible conectar el pulsador de cualquiera de estas dos maneras, solo habrá que escribir un algoritmo que detecte el accionamiento del mismo de manera correcta. En este trabajo práctico, se utilizó una resistencia de pull up, ya que el microcontrolador MC9S08SH8 integrado en la placa de práctica tiene la posibilidad de habilitar una resistencia de pull up interna con un registro llamado PTCPE (Port C Pull Enable Register)

4.3.3. Efecto rebote ocasionado por el pulsador mecánico

El pulsador es una chapa metálica, que al pulsarla flexa y toca los contactos cerrando el circuito. Pero al liberarlo, normalmente, se producen unos rebotes, que en aplicaciones de precisión puede provocar “falsos positivos”. El rebote presente en los pulsadores dependerá de la calidad y desgaste de este.

Generalmente, el tiempo del efecto rebota se encuentra entre los 10 – 20 ms.

El efecto rebote puede ser solucionado tanto por hardware como por software, a continuación, veremos algunas de las posibles soluciones:

Con un latch SR, un interruptor de bidireccional y dos resistencias pull up

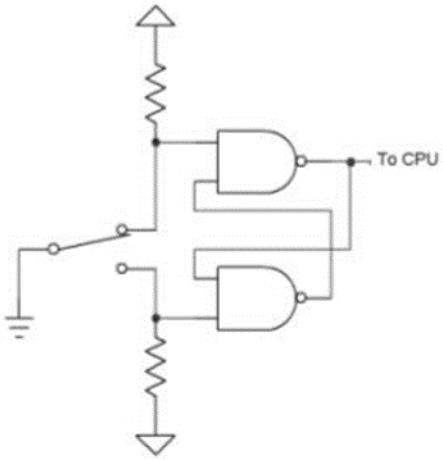


Figure 1: The SR debouncer

Con un circuito resistivo-capacitivo

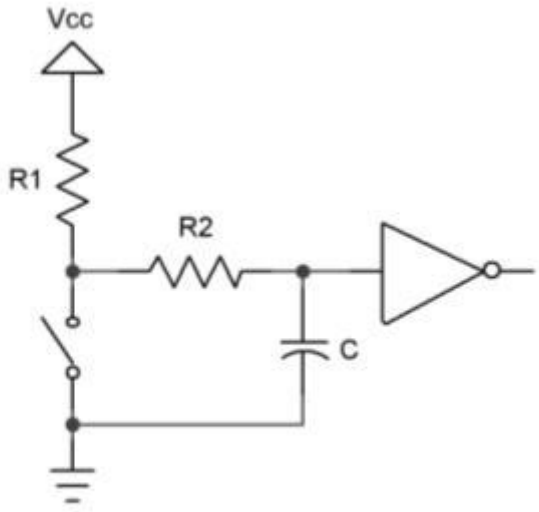


Figure 2: An RC debouncer

El circuito SR es raramente utilizado, ya que los switches bidireccionales son más caros que los comunes, y no hay manera de hacerlos lo suficientemente pequeños como para colocar en la placa

Por firmware

La más simple de todas las estrategias de eliminación de rebote, es simplemente chequear el estado del switch cada, aproximadamente, 500 ms y colocar un flag indicando el estado de la entrada. Ningún switch va a rebotar por tanto tiempo, por este motivo, la lectura lenta logra que el rebote no afecte a la lectura del switch. Una desventaja es la respuesta lenta, que en la mayoría de los casos no interesa, pero, por ejemplo, una persona que escriba muy rápido en un teclado, con este tipo de retardos, tendría muchos problemas a la hora de hacer su trabajo.

5. Cambio de secuencia de encendido de LEDs mediante el pulsador

5.1. Problema

Modifique el programa del punto 3. de manera que al presionar y soltar un pulsador (conectado a PC0) cambie el sentido de la secuencia de encendido. Es decir, mediante el pulsador el usuario puede cambiar a la secuencia 0-1-2-3-4-5-6-7 y viceversa. ¿Qué sucede si el usuario deja el pulsador constantemente presionado?

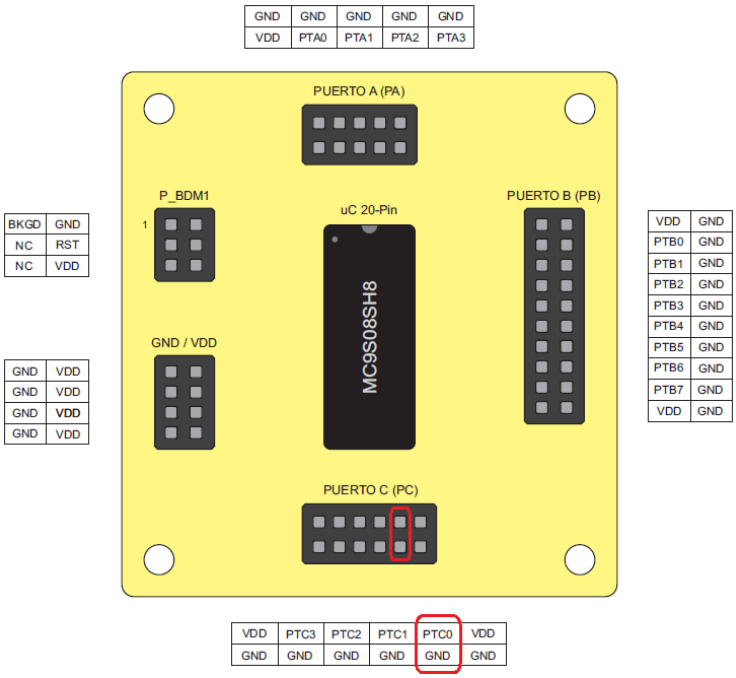
5.2. Interpretación

Dado el microcontrolador MC9S08SH8 CPJ (incluido en el kit), un conjunto de 8 LEDs y un pulsador mecánico, se deberá resolver y analizar el ejercicio provisto por la cátedra.

A diferencia del problema del punto 3., donde la secuencia de encendido de LEDs se realizaba de manera automática por software con X cantidad de tiempo por igual para cada LED, ahora la complejidad aumenta teniendo que cambiar el sentido de dicha secuencia mediante el uso de un pulsador mecánico.

5.3. Resolución

El conexionado entre el pulsador mecánico y el puerto C de la placa será en el pin 0 de la misma, como se ve en la siguiente imagen:



Donde, el pseudocódigo del programa será el siguiente:

```
Apagar todos los LEDs
      Repetir infinitamente
            Para cada LED conectado
                  Si se presionó el pulsador
                        Mientras siga presionado, espero
                        Invertir secuencia
                  encender LED
                  esperar 200mS.
                  apagar LED
```

Como podemos observar en el pseudocódigo, en este problema, a su vez, habrá que contemplar el caso en que el usuario deje constantemente el pulsador presionado, y tomar una decisión ante ello. Para este caso, optamos por que la secuencia se detenga hasta que el pulsador sea soltado, por lo que se mantendrá encendido el LED que lo estaba en el momento que se presionó el pulsador.

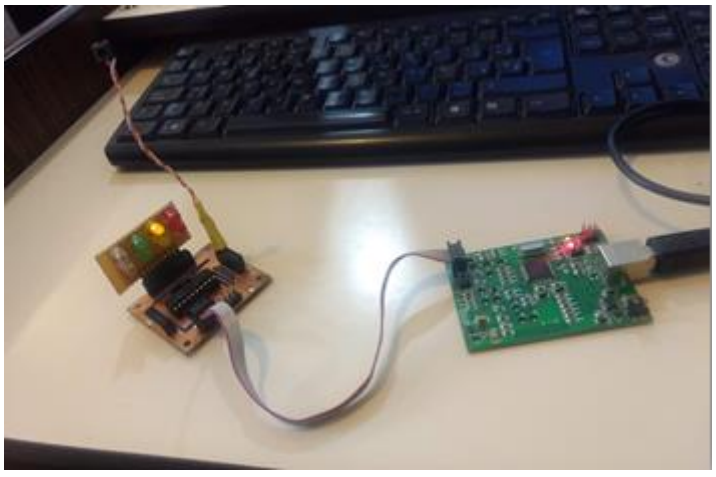


Foto del kit funcionando

6. Anexo: Main.c

6.1. Punto 2

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

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

void delay(unsigned short n){//Funcion que recibe por par□tro
la cantidad de miliseg de delay
  unsigned short temp;
  unsigned short cant;
  /*Aca sabemos que 1a ejecucion del for equivale a 23 ciclos
de reloj
  y que hacemos 8000 ciclos por milisegundo, eso equivale a mas
o menos
  348 ejecuciones del for por milisegundo*/
  for(cant=n; cant>0; cant--){
    for(temp=348; temp>0; temp--);
  }
}

void main(void) {

  PTBDD = 0xFF;
  PTBD = 0x03;

  MCU_init();   /* call Device Initialization */
  /* include your code here */


  for(;;) {
    if(PTBD == 0){
        PTBD = 0x03;
    }
    delay(200);

    PTBD=PTBD*4;
      /* __RESET_WATCHDOG(); By default COP is disabled with
device init. When enabling, also reset the watchdog. */
  } /* loop forever */
  /* please make sure that you never leave main */
}
```

6.2. Punto 3

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

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

void delay(unsigned short n){//Funcion que recibe por par□tro
la cantidad de miliseg de delay
```

```
  unsigned short temp;
  unsigned short cant;
  /*Aca sabemos que 1a ejecucion del for equivale a 23 ciclos
de reloj
  y que hacemos 8000 ciclos por milisegundo, eso equivale a mas
o menos
  348 ejecuciones del for por milisegundo*/
  for(cant=n; cant>0; cant--){
    for(temp=348; temp>0; temp--);
  }
}

void main(void) {
  PTBDD = 0xFF;
  PTBD = 0x01;

  MCU_init();   /* call Device Initialization */
  /* include your code here */



  for(;;) {
     if(PTBD == 0){
       PTBD = 0x01;
     }
     delay(200);
     PTBD = PTBD<<1;
    /*  __RESET_WATCHDOG(); By default COP is disabled with
device init. When enabling, also reset the watchdog. */
  } /* loop forever */
  /* please make sure that you never leave main */
}
```

6.3. Punto 4

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

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

void delay(unsigned short n){//Funcion que recibe por par□tro
la cantidad de miliseg de delay
  unsigned short temp;
  unsigned short cant;
  /*Aca sabemos que 1a ejecucion del for equivale a 23 ciclos
de reloj
  y que hacemos 8000 ciclos por milisegundo, eso equivale a mas
o menos
  348 ejecuciones del for por milisegundo*/
  for(cant=n; cant>0; cant--){
    for(temp=348; temp>0; temp--);
  }
}

void main(void) {
  PTBDD = 0xFF;
  PTBD = 0x01;
```

```
  MCU_init();   /* call Device Initialization */
  /* include your code here */

  PTCDD_PTCDD0 = 0;//Posicion de la memoria en la que guardo el
valor de entrada
  PTCPE_PTCPE0 = 1;//Seteo la entrada PTCD0 con una resistencia
pull up interna

  for(;;) {
    delay(10);//Delay para el polling del boton (No seria
necesario en este caso por el delay de los led)
    if(PTBD == 0x00){
      PTBD = 0x01;
      delay(500);//Un delay para cuando el led esta en 1
    }
    //Aca se lee a la entrada para el pulsador
    if(PTCD_PTCD0 == 0){
      PTBD = PTBD<<1;
      if(PTBD != 0){//Para que no tenga un delay de un segundo
con los led apagados
        delay(500);
      }
    }
    /* __RESET_WATCHDOG(); By default COP is disabled with
device init. When enabling, also reset the watchdog. */
  } /* loop forever */
  /* please make sure that you never leave main */
}
```

6.4. Punto 5

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

#ifdef __cplusplus
extern "C"
#endif

#define tiempo_leds 150 //El tiempo que queda prendido cada led
en milisegundos

void MCU_init(void); /* Device initialization function
declaration */

void delay(unsigned short n){//Funcion que recibe por par□tro
la cantidad de miliseg de delay
  unsigned short temp;
  unsigned short cant;
  /*Aca sabemos que 1a ejecucion del for equivale a 23 ciclos
de reloj
  y que hacemos 8000 ciclos por milisegundo, eso equivale a mas
o menos
  348 ejecuciones del for por milisegundo*/
  for(cant=n; cant>0; cant--){
    for(temp=348; temp>0; temp--);
  }
}

void main(void) {
  unsigned char estado = 0;
```

```
  PTBDD = 0xFF;
  PTBD = 0x01;

  MCU_init();   /* call Device Initialization */
  /* include your code here */

  PTCDD_PTCDD0 = 0;//Posicion de la memoria en la que guardo el
valor de entrada
  PTCPE_PTCPE0 = 1;//Seteo la entrada PTCD0 con una resistencia
pull up interna

  for(;;) {
    //Aca se lee a la entrada para el pulsador
    if(PTCD_PTCD0 == 0){
        delay(20);
        while(PTCD_PTCD0 == 0);
        delay(20);
        estado = ~estado;
    }
    if(estado){
      if(PTBD == 0x00){
        PTBD = 0x01;
        delay(tiempo_leds);//Un delay para cuando el led esta
en 1
      }
      PTBD = PTBD<<1;
      if(PTBD != 0){//Para que no tenga un delay de un segundo
con los led apagados
        delay(tiempo_leds);
      }
    }
    else{
      if(PTBD == 0x00){
        PTBD = 0x0080;
        delay(tiempo_leds);//Un delay para cuando el led esta
en 1
      }
      PTBD = PTBD>>1;
      if(PTBD != 0){//Para que no tenga un delay de un segundo
con los led apagados
        delay(tiempo_leds);
      }
    }
    /* __RESET_WATCHDOG(); By default COP is disabled with
device init. When enabling, also reset the watchdog. */
  } /* loop forever */
  /* please make sure that you never leave main */
}
```