10 DE ABRIL DE 2022

TP N°1 ENTREGABLE

BLANCO VALENTÍN NICOLAS, PALADINO GABRIEL AGUSTÍN
CIRCUITOS DIGITALES Y MICROCONTROLADORES

Interpretación:

Se debería realizar un proyecto en C que encienda 8 leds (todos de distinto color) siguiendo una secuencia, donde los mismos se conectan a uno de los puertos del microcontrolador Atmega328p (para ser precisos el puerto a conectar los leds es el B). La forma a realizar la secuencia viene dada por un botón conectada al primer bit de otro puerto (Puerto C) permitiéndonos alternar entre 2 secuencias preestablecidas (encendiendo los leds externos a internos y viceversa).

Consideraciones a tener en cuenta:

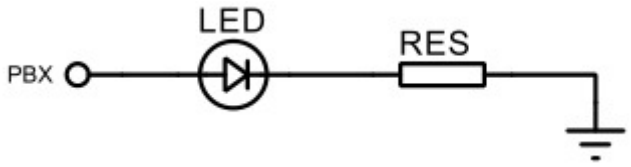
Para realizar esta práctica no se puede hacer uso de las interrupciones para detectar si se pulso el botón.

A disposición tenemos las tensiones de cada uno de los leds, pero no de sus resistencias que deben acompañarlos para limitar la corriente a 10 mA por lo que se deberá realizar el cálculo de cada una de ellas.

Resolución:

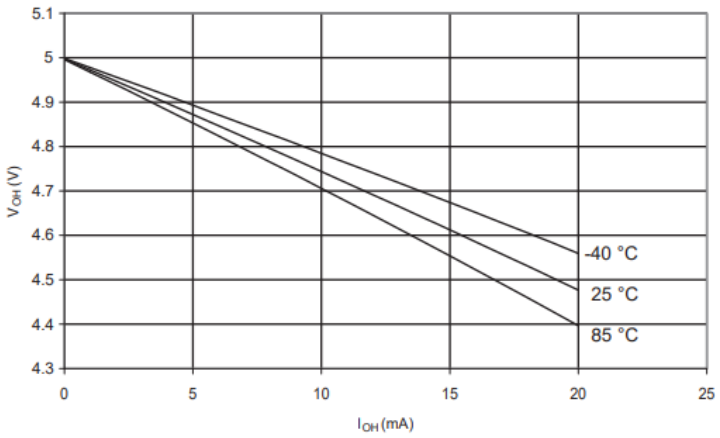
Para cada inciso se realizó lo siguiente:

Inciso 1: Para realizar los cálculos antes que nada definimos que utilizaríamos una conexión donde el led se enciende en alto (Ver figura 1)



(Figura 1: PBX hace referencia a un puerto configurado como salida)

Aplicando la segunda ley de Kirchhoff se llegó a la siguiente fórmula para calcular la resistencia en cada led: $Res = \frac{V_{OH} - V_{LED}}{I_{OH}}$ donde V_{OH} es la tensión que entrega el Atmega328p cuando la salida está en alto (el valor de este se obtuvo de la hoja de datos, ver Figura 2)



(Figura 2: tensión de salida para distintas corrientes y temperaturas entregadas por el Atmega328P)

De la figura 2 se obtuvo que V_{OH} es de 4.75 V aproximadamente para una temperatura ambiente donde I_{OH} es de 10 mA.

Para cada uno de los leds los valores fueron los siguientes:

$Color$	V_{LED}	Res
Rojo	1.8 V	295 Ω
Amarillo	2 V	275 Ω
Naranja	2 V	275 Ω
Verde	2.2 V	255 Ω
Azul	3 V	175 Ω
Blanco	3.4 V	135 Ω
Purpura	3.4 V	135 Ω
Rosa	3.4 V	135 Ω

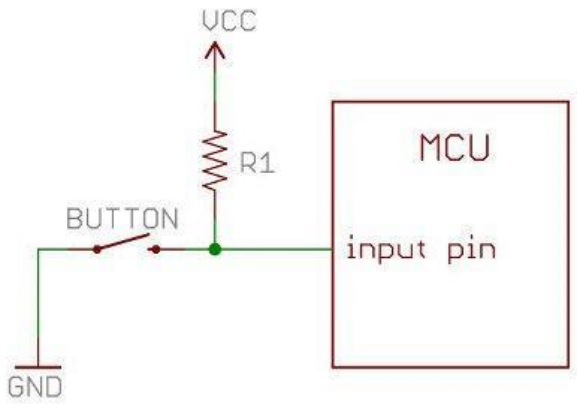
(Tabla 1: valores de resistencia para cada led)

Donde V_{LED} fueron los datos dados o buscados en internet y Res los valores obtenidos a partir de la fórmula.

Cada pin del Atmega328P soporta una corriente máxima de 40 mA sin que ocurran daños, en el caso de nuestro circuito la corriente de cada salida está limitada a 10 mA por los valores de Res calculados por lo que no debería ocurrir ningún problema.

Además, la suma de todas las corrientes entregadas no debería ser mayor a 200 mA y en nuestro caso no sería ningún problema ya que con todos los leds encendidos se obtiene un total de 80 mA.

Inciso 2: Para resolver el trabajo se optó por usar una conexión pull-up (véase figura 3) de modo tal que cuando se pulse el botón se genere un 0 lógico en la entrada.



(Figura 3: Conexión tipo pull-up)

En nuestro caso la decisión de usar pull-up sobre pull-down fue dada porque en el microcontrolador se puede configurar la entrada como pull-up para utilizar la resistencia interna que trae el MCU ahorrándonos el uso de una resistencia adicional.

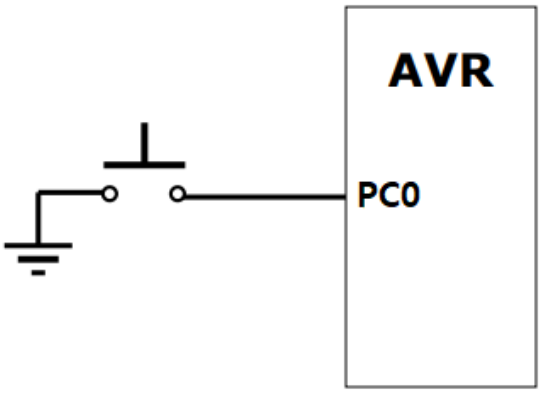
Para llevar a cabo esta configuración se accedió a la hoja de datos viendo de esta manera como se deben configurar los registros internos del MCU (Figura 4).

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

(Figura 4)

En nuestro caso debíamos usar la entrada 0 del PORTC por lo que viendo la Figura 4 para activar el modo PULL-UP de dicho PIN lo configuramos poniendo DDRC0 en 0 y PORTC0 en 1.

Por lo explicado la conexión del botón sería la siguiente:



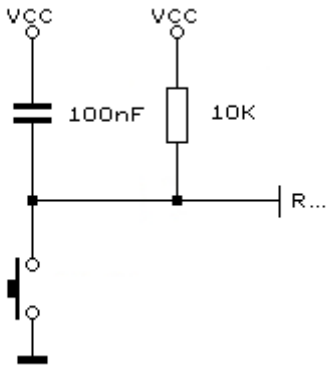
(Figura 5)

Hablando sobre el botón además puede ocurrir un efecto denominado de rebote (observar figura 6) que consiste en que cuando el botón es presionado puede provocar valores inválidos de tensión o disparos múltiples de esta debido a las características físicas del botón.

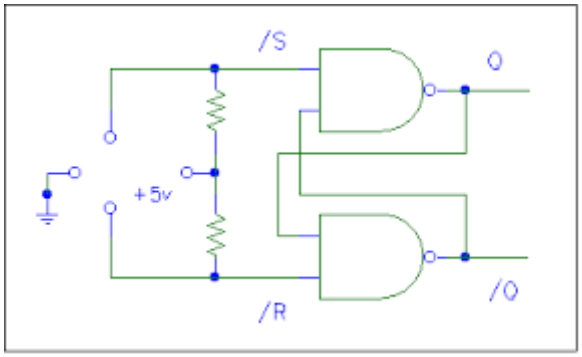
(Figura 6: efecto rebote en nuestra conexión usada)

Para solucionar este problema se puede optar por 2 grandes formas:

- Solución hardware: Se puede solucionar ya sea a partir de un capacitor (Figura 7) o con el uso de un flip flop (Figura 8).

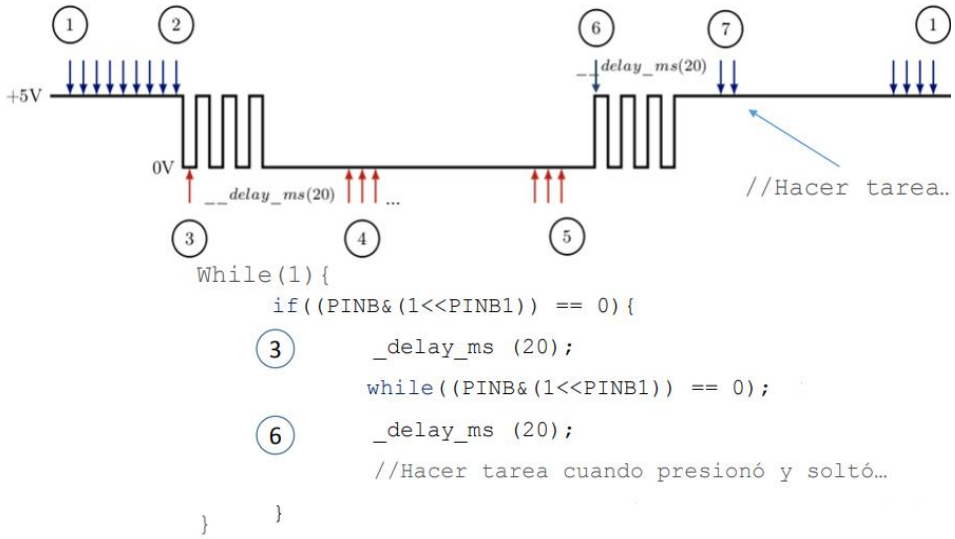


(Figura 7: Solución rebote por hardware en conexión pull-up)



(Figura 8: Solución rebote por hardware con flip flop)

- Solución software: Este es el tipo de solución que optamos por usar y consiste en el agregado de funciones delay luego de detectar algún cambio de estado en el botón ignorando los rebotes producidos. Para nuestra implementación usamos un valor habitual de 20 ms que debería ser suficiente para solucionar el problema.



(Figura 9: Solución rebote por software con delay)

Existen diversos tipos de solución por software, pero el más común y fácil de desarrollar es el anterior expuesto (Figura 9) por lo que detallamos esa.

Inciso 3: El programa consta de dos bloques mientras donde cada uno ejecuta una de las dos secuencias pedidas por el enunciado, donde para pasar de un mientras al otro se utiliza el botón y cada vez que se encienden dos leds de la secuencia se verifica si debe pasar a ejecutar la otra secuencia (Figura 10).

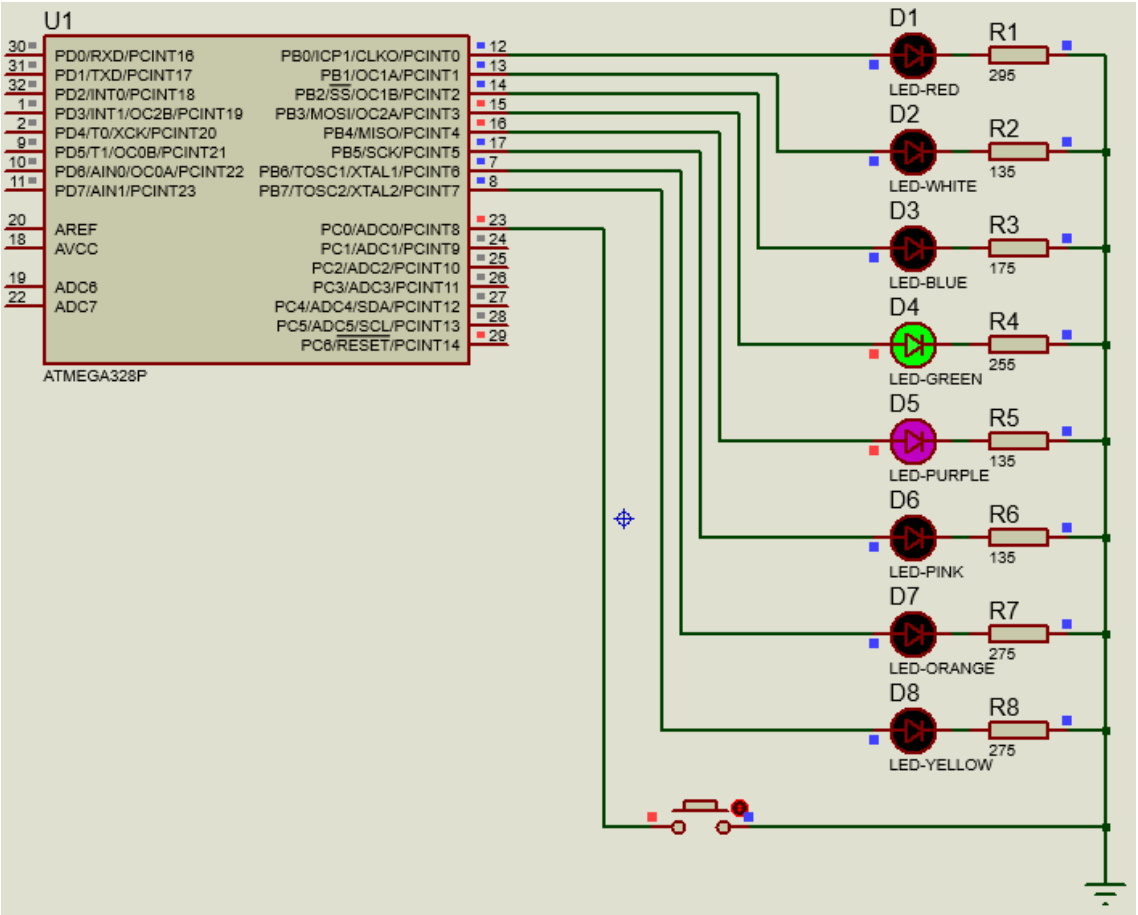
```
Inicia el programa
    mientras este seleccionada la secuencia 1
        Encender led 0 y 7
        Verificar si se pulso el boton
            Caso verdadero: cambiar de secuencia
            Caso contrario: seguir con la secuencia
        Seguir secuencia con cada uno de los led (1-6, 2-5, 3-4)
    terminar
    mientras este seleccionada la secuencia 2
        Encender led 3 y 4
        Verificar si se pulso el boton
            Caso verdadero: cambiar de secuencia
            Caso contrario: seguir con la secuencia
        Seguir secuencia con cada uno de los led (2-5, 1-6, 0-7)
    terminar
termina el programa
```

(Figura 10: Pseudocódigo del programa)

Inciso 4: De conclusión sobre la resolución podemos decir que al no hacer uso de interrupciones el programa demora un tiempo en notar que el botón fue pulsado. También se puede decir que al hacer uso de una técnica como retardo bloqueante de forma que no haya efecto rebote el programa no es del todo eficiente, ya que al agregar delay son ciclos del reloj en los que el procesador no realiza ninguna acción.

Además, si se mantiene pulsado el botón, el programa se queda en “pausa” sin seguir una secuencia por lo que no sería lo más óptimo.

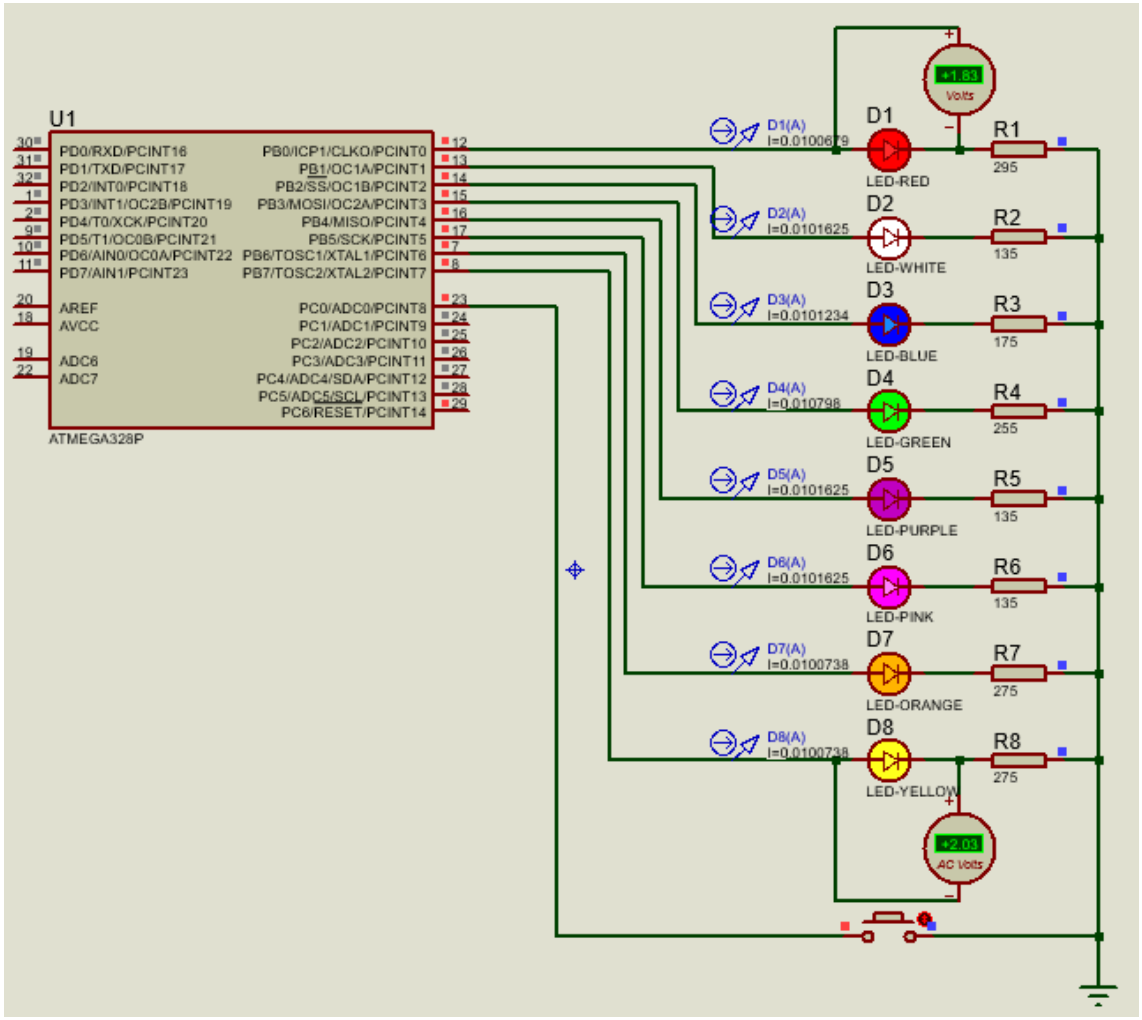
Validación:



(Figura 11: imagen del circuito usado para resolver el problema)

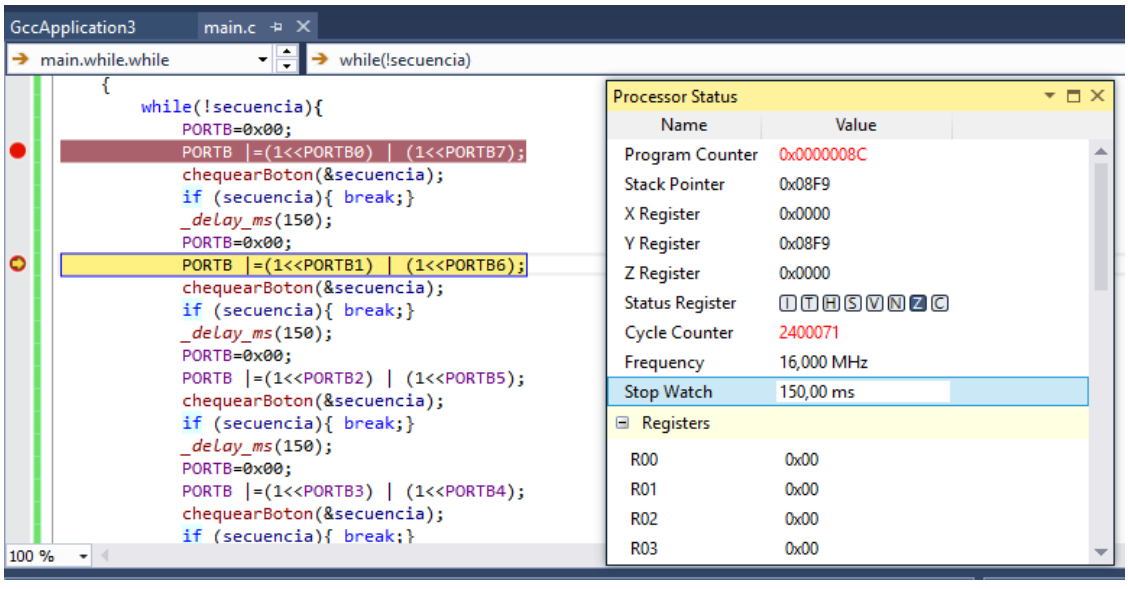
En la figura 11 se puede observar una captura del circuito a resolver. Como se puede ver, para conectar el botón no se utiliza resistencia debido a que hacemos uso de la resistencia interna del MCU por lo explicado en el punto anterior.

También se puede ver que los leds están conectados al puerto B y que los mismo se encienden cuando la señal está en alto.



(Figura 12: Validación de los cálculos realizados)

Como se puede ver en la figura 12, se midió la corriente que circula por cada led para corroborar los cálculos de resistencias hechos previamente. Además, haciendo uso de voltímetros comprobamos que la caída de tensión en el led rojo y amarillo sea la dada en la tabla 1.



(Figura 13: debugger Microchip Studio)

Para medir cuánto tiempo tarda de pasar de una secuencia a otra usamos dos break points y el debugger. Como se puede ver en la figura determinamos que tarda 150 ms en cambiar de leds que es lo correcto ya que es el valor ingresado en la función delay.

Código:

```
#include <avr/io.h>

#define F_CPU 16000000UL

#include <util/delay.h>

void chequearBoton(int *);

int main(void)

{

 PORTB =0x00; //Definimos todos las leds en estado inicial apagado

 DDRB =0xFF;  //Configuramos el puerto B como salida

 DDRC &=~(1<<PORTC0); //Configuramos el pin 0 del puerto C como entrada

 PORTC |=(1<<PORTC0);  //Configuramos el pin 0 en alto para que se comporte como pull-up interno

 int secuencia=0; // variable utilizada para alternar secuencias

 while (1)

 {

  while(!secuencia){ //Cuando secuencia vale 0 se ejecuta el encendido de leds extremos a internos

    PORTB=0x00; //se apagan todos los leds

    PORTB |=(1<<PORTB0) | (1<<PORTB7); //Se pone en alto los pines 0 y 7 del portb encendiendo de esta forma los leds 0 y 7

    chequearBoton(&secuencia); //subrutina que lee si se pulso el boton y actualiza el valor de secuencia

    if (secuencia){ break;} //si secuencia cambio se pasa al otro while (cambia de secuencia)

    _delay_ms(150);

    PORTB=0x00;

    PORTB |=(1<<PORTB1) | (1<<PORTB6);
```

```
  chequearBoton(&secuencia);
  if (secuencia){ break;}
  _delay_ms(150);
  PORTB=0x00;
  PORTB |=(1<<PORTB2) | (1<<PORTB5);
  chequearBoton(&secuencia);
  if (secuencia){ break;}
  _delay_ms(150);
  PORTB=0x00;
  PORTB |=(1<<PORTB3) | (1<<PORTB4);
  chequearBoton(&secuencia);
  if (secuencia){ break;}
  _delay_ms(150);
 }
 while(secuencia){ //si secuencia vale 1 se encienden los leds de internos a externos
  PORTB=0x00;
  PORTB |=(1<<PORTB3) | (1<<PORTB4);
  chequearBoton(&secuencia);
  if (!secuencia){ break;}
  _delay_ms(150);
  PORTB=0x00;
  PORTB |=(1<<PORTB2) | (1<<PORTB5);
  chequearBoton(&secuencia);
  if (!secuencia){ break;}
  _delay_ms(150);
  PORTB=0x00;
  PORTB |=(1<<PORTB1) | (1<<PORTB6);
  chequearBoton(&secuencia);
  if (!secuencia){ break;}
```

```
    _delay_ms(150);
    PORTB=0x00;
    PORTB |=(1<<PORTB0) | (1<<PORTB7);
    chequearBoton(&secuencia);
    if (!secuencia){ break;}
    _delay_ms(150);
   }
  }
}
void chequearBoton(int *s){
  if ((PINC & (1<<PINC0))==0){ //si se pulso el boton entra al if
   _delay_ms(20); //cancelo efecto rebote al apretar el boton
   while((PINC & (1<<PINC0)) == 0); //mientras el boton se mantenga apretado se
queda iterando
   _delay_ms(20); //cancelo efecto rebote al soltar boton
   if(*s){ //actualizo el valor de secuencia
    (*s) = 0;
   }
   else{
    (*s) = 1;
   }
  }
}
```