Trabajo Práctico N°1
Control de periféricos externos con puertos de entrada/salida

Grupo 4:

Graciano Gonzalez, Fausto. Legajo: 03028/2

Zanetti, Bruno. Legajo: 02975/5

Índice

1. Esquema Eléctrico

1.1 Enunciado

Se desea conectar 8 diodos LED de diferentes colores al puerto B del MCU y encenderlos con una corriente de 10mA en cada uno. Realice el esquema eléctrico de la conexión en Proteus. Calcule la resistencia serie para cada color teniendo en cuenta la caída de tensión VLED (rojo=1.8V, verde=2.2V amarillo=2.0V azul=3.0V) Verifique que la corriente por cada terminal del MCU no supere la capacidad de corriente de cada salida y de todas las salidas del mismo puerto en funcionamiento simultáneo.

1.2 Interpretación

Utilizando el simulador Proteus y el microcontrolador ATmega328P, se deberá crear un proyecto, en el cual se insertarán 8 diodos LED de distintos colores, los cuales se deben conectar cada uno a un pin diferente del puerto B del MCU.

Para cada diodo LED, necesitaremos determinar la caída de tensión correspondiente al color. Luego, se deberá calcular el valor de la resistencia que debe estar en serie con cada diodo LED para limitar la corriente a 10mA. Esto se hace teniendo en cuenta la tensión de salida de los pines del microcontrolador y la tensión de alimentación proporcionada.

Por último se verificará que la corriente de cada terminal no supere la de cada salida.

1.3 Resolución del problema

1.3.1. MCU

Mediante el software Proteus, se incorporaron los componentes necesarios mediante el menú de búsqueda “Library > Pick Parts”, en este caso se selecciona el MCU y se agrega en la hoja de proyecto. En este trabajo se utilizó el “ATmega328P”, el cual por defecto se le proporciona una tensión de alimentación Vcc de 5V. Se muestra la representación del MCU en la Figura 1.1 a continuación.

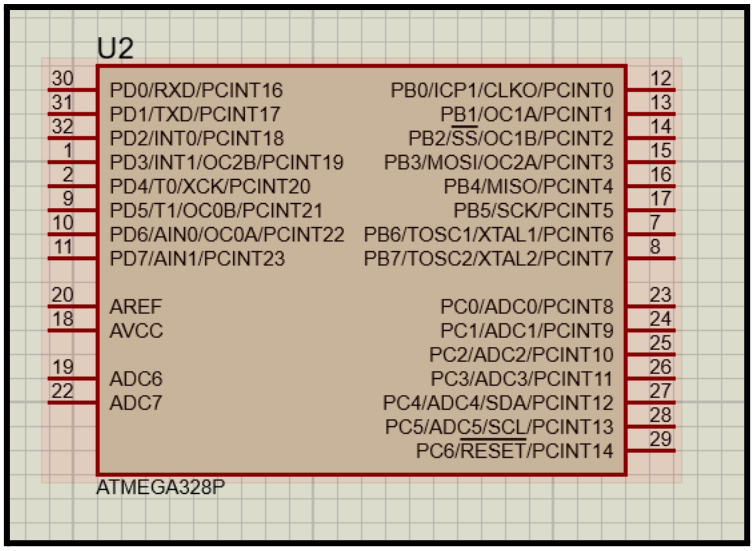

Figura 1.1. Microcontrolador ATmega 328P en Proteus.

1.3.2. LEDS

Se realizo el agregado de los diodos LEDS utilizando 2 de cada color segun las tensiones de caida “Vd” proporcionadas por el enunciado (2 azules con Vd=3.0V, 2 rojos con Vd=1.8V, 2 amarillos con Vd=2.0V y 2 verdes con Vd=2.2V).
Luego, se establecieron las propiedades de cada diodo en proteus con su correspondiente tensión de caída y además, indicando el tipo de modelo como “analógico”.

1.3.3. Resistores en serie

A cada diodo LED se conectó una resistencia en serie con el objetivo de limitar la corriente I que circula por el conductor y evitar que los componentes emisores de luz se dañen. Para finalizar el circuito estos se conectan a tierra(GROUND).
Resultando el circuito mostrado en la figura 1.2.

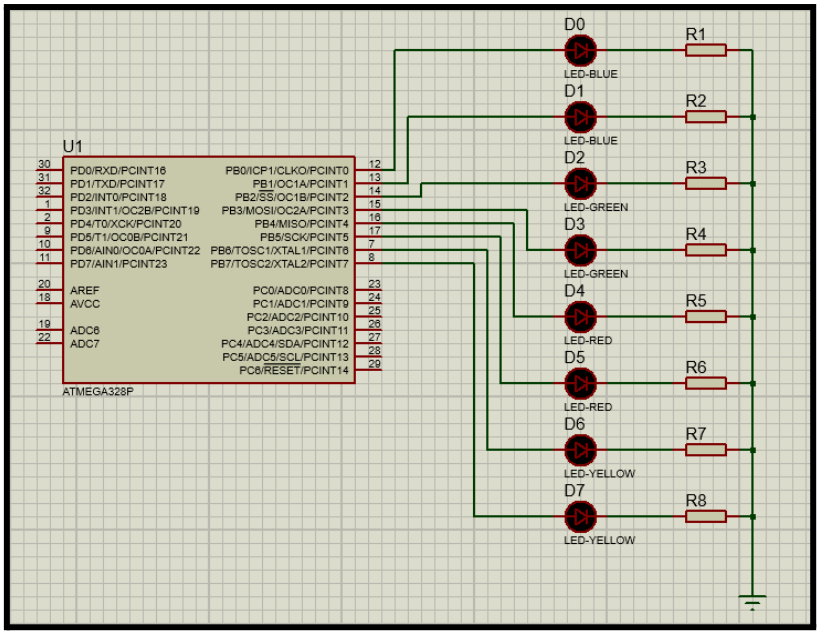

Figura 1.2. Circuito completo con MCU, LEDs y resistores.

1.4 Validación

Para la simulación del circuito completo, antes se calcularon los valores en Ohms de las resistencias en serie correspondientes a cada diodo. Los mismos se calcularon de forma que la corriente que recorre los conductores sea idealmente I=10mA, asumiendo además una tensión de alimentación Vcc de 5V, la cual no es exacta ya que se comporta como las fuentes reales de tensión, entregando menor tensión a medida que la corriente aumenta. Este efecto se indica en la hoja de datos del ATmega328P (p. 271, cap. 29, figura 29-10), cuya gráfica se muestra a continuación en la figura 1.3.

Para calcular el valor de la tensión de salida “Voh” se asume una temperatura ambiente de 25°C y como ya antes dicho una corriente de salida de 10mA, con esto se obtiene una Voh≈4,72V.

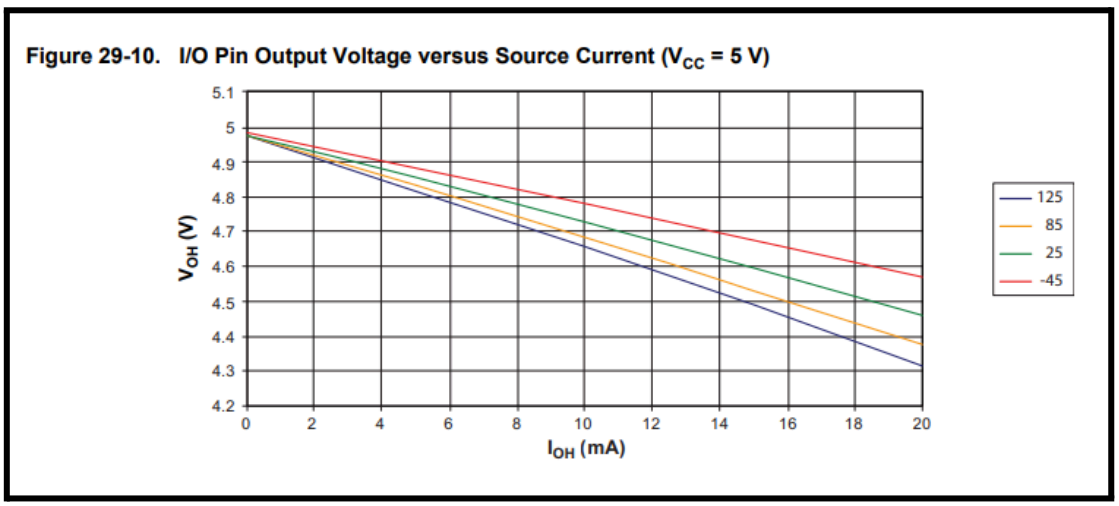

Figura 1.3. Tensión de salida versus corriente de salida en los terminales I/0 para Vcc=5V.

Finalmente se obtiene la resistencia correspondiente a cada diodo LED de la siguiente manera:

$$Rled = \frac{Voh - Vd}{I}$$

Resultando los siguiente valores para cada diodo LED:
Resistencia Led ROJO: 292 Ohms.
Resistencia Led AZUL: 172 Ohms.
Resistencia Led Verde: 252 Ohms.
Resistencia Led Amarillo: 272 Ohms.
Con estos valores se cumple que la corriente de cada terminal no supera la capacidad de corriente de los diodos.
Además, el fabricante del microcontrolador indica que la suma total de las corrientes de los pines B0 a B5 no debe exceder los 150mA. En este caso, esta restricción también se cumple, ya que la suma máxima sería de 60 mA si cada rama lleva hasta 10 mA de corriente.
Para el encendido de todos los diodos LED, se necesita configurar todos los pines del puerto B como salidas y luego escribir un valor de "1" en cada bit del registro de datos del puerto B. Esto se logra mediante las siguientes líneas:

```
DDRB = 0xFF;     // Configura todos los pines del puerto B como salidas
PORTB = 0xFF;    // Escribe un valor de "1" en todos los bits del puerto B
```

Estas sentencias se agregan al código principal (main) del programa en C. Después de escrito el código, se compila para crear un archivo con extensión ".elf". Este archivo luego se carga como Program File dentro de MCU en el simulador Proteus, donde se puede observar el resultado de la simulación, que es el encendido de todos los LEDs conectados al puerto B del microcontrolador como se muestra en la figura 1.4 a continuación.

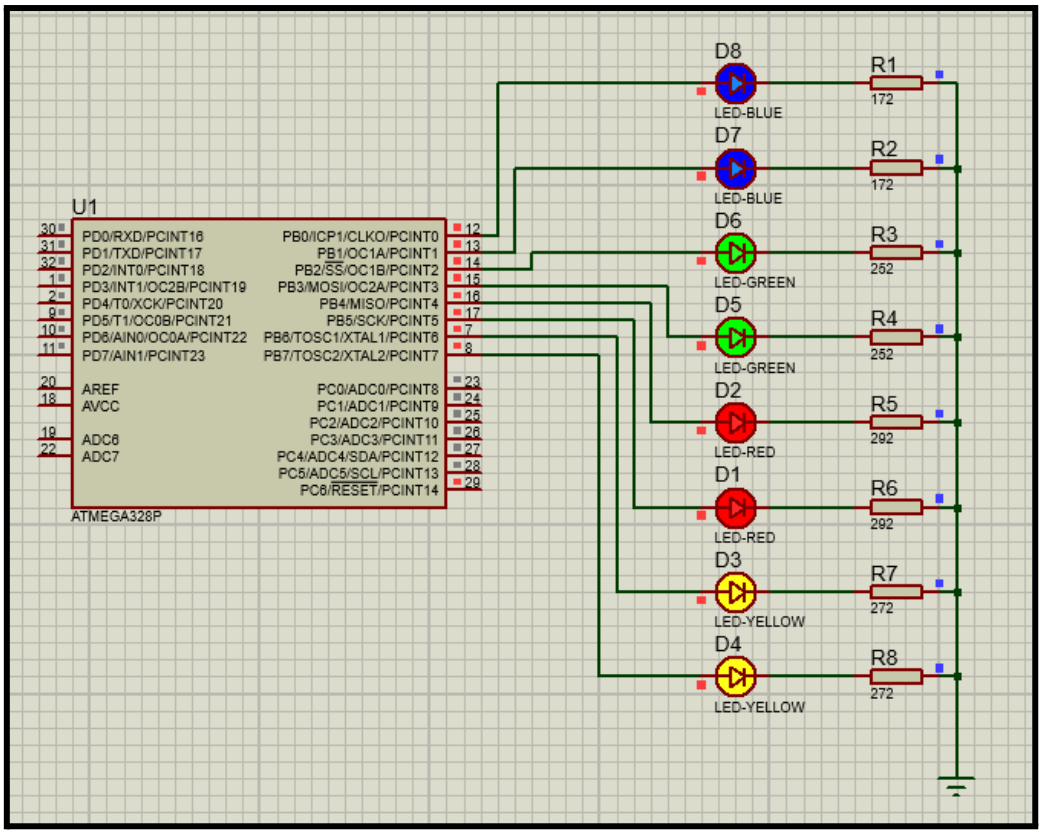

Figura 1.4. Simulación con todos los diodos encendidos.

2. Conexión de un pulsador

2.1 Enunciado

Se desea conectar un pulsador a una entrada digital del MCU y detectar cuando el usuario presiona y suelta el pulsador. Muestre el esquema de conexión y determine la configuración del MCU que corresponda. Investigue sobre el efecto de rebote que producen los pulsadores e implemente un método para eliminar este efecto en su algoritmo de detección (puede encontrar información útil en la bibliografía).

2.2 Interpretación

En el simulador Proteus, al proyecto creado en el punto anterior debemos añadirle un pulsador a un terminal que se configura como entrada del microcontrolador Atmega328P, y debemos detectar con el MCU cuando el usuario presiona y suelta el pulsador.

Se deberá realizar una investigación sobre el efecto rebote que se produce en los pulsadores e implementar un método para eliminar este efecto en el algoritmo de detección para que no haya errores de funcionamiento por parte del pulsador.

2.3 Resolución del problema

2.3.1. Esquema de la conexión del pulsador

Para la conexión del pulsador al microcontrolador se agrega en el proyecto de Proteus el elemento “button” encontrado en “Library > Pick Parts”.
Se decidió conectar el pulsador al MCU con un Pull-Up externo, donde debemos conectar el pulsador por un lado a tierra(GROUND) y del otro lado al borne del PIN de C0. A su vez, también se conecta un Vcc con su resistencia Pull-Up.
A continuación se puede observar la figura 2.1 con la conexión con pull up externo.

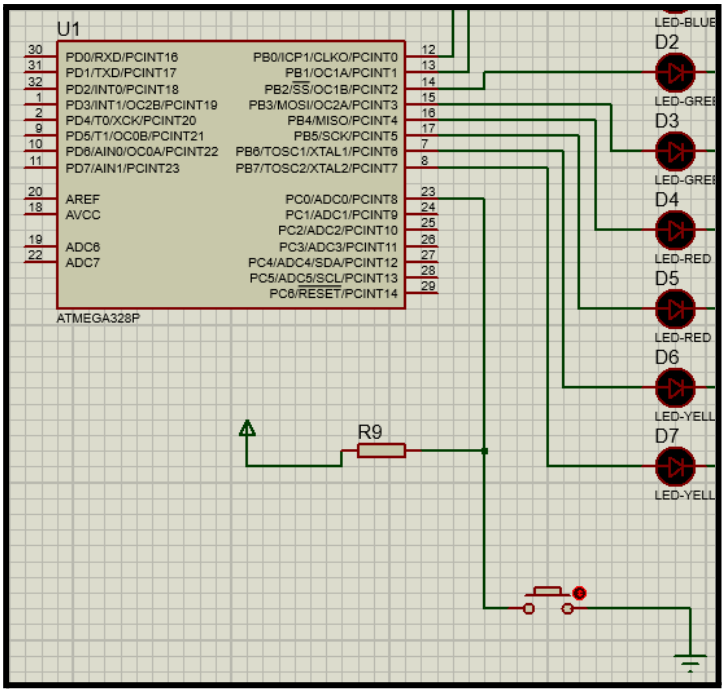

Figura 2.1. Esquema de la conexión del pulsador.

2.3.2. Configuración

Para la conexión del pulsador se decide usar al bit 0 de PORTC como entrada, usando la siguiente sentencia en el código main:

DDRC &= ~(1<<PORTC0);

Para asegurar que el pin del PORTC tenga un estado definido tanto cuando el pulsador está abierto como cuando está cerrado, y evitar que quede en un estado indefinido se usó la configuración con Pull-Up externo explicada anteriormente, mediante el cual se define que si el pulsador se encuentra abierto el estado del pin será ALTO(1) y si se encuentra cerrado(“presionado”) el estado del mismo será BAJO(0).

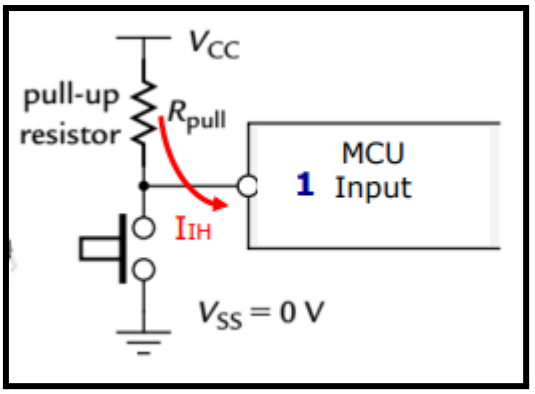

Figura 2.2 Configuración de PULL-UP externo.

Para la resistencia de Pull-Up, se usó una típica de 10KΩ.

2.3.3. Investigación Efecto rebote

El efecto de rebote en los pulsadores es un fenómeno común que ocurre cuando se presiona o se suelta un botón. Este efecto se debe a las características mecánicas del botón y puede provocar que la señal de salida del botón oscile rápidamente entre los estados de "presionado" y "no presionado" varias veces en un corto período de tiempo, antes de estabilizarse en el estado final.

Este fenómeno puede generar problemas en los circuitos electrónicos que dependen de la señal del botón para realizar acciones específicas, como cambios de estado, controles de usuario, etc. Por ejemplo, en un circuito digital, el rebote del botón puede hacer que el microcontrolador o circuito lógico interprete múltiples pulsaciones rápidas del botón como una sola pulsación larga o varias pulsaciones individuales, lo que puede causar un comportamiento no deseado en el sistema.

Figura 2.3 Gráfico del efecto rebote.

2.3.4 Resolución para el efecto rebote

Para resolver el problema del efecto rebote en los pulsadores, implementamos un algoritmo el cual consiste en chequear el estado del pulsador durante cada ciclo, analizando si se encuentra “pulsado”, o sino, lo contrario, si se mantiene “pulsado” durante un tiempo determinado, es decir, si cumple con el tiempo estipulado para asumir que verdaderamente está siendo pulsado, permitiendo mitigar el efecto rebote que puede generarse cuando se presiona o se suelta el pulsador.
Pseudocódigo del algoritmo:

```
Inicializar el estado con el valor uno (sin pulsar)
   Inicializar el contador de los ciclos en cero
   leo el estado actual del pulsador
    si el estado actual es distinto del estado {
      incremento el contador en uno
      si el contador alcanza el valor esperado {
      asigno el valor del estado actual a estado
      si estado es igual al valor uno (se dejó de pulsar)
         realizar accion segun el estado
         reiniciar el contador a cero  }
} sino
 {reiniciar el contador a cero}
} fin
```

3. Programa con secuencias

3.1 Enunciado

Realice el programa para que el MCU encienda los LEDs del puerto B con la siguiente secuencia de encendido repetitiva: b0 y b7 – b1 y b6 – b2 y b5 – b3 y b4. Luego, cuando el usuario presione y suelte el pulsador debe cambiar a la secuencia: b7-b6-b5-b4-b3-b2-b1-b0. Si presiona y suelta nuevamente vuelve a la secuencia original y así sucesivamente. Elija un retardo adecuado para la visualización en el simulador. Justifique.

3.2 Interpretación

Se debe hacer un programa en C, que permita realizar dos secuencias de encendido de los 8 diodos LED. Por defecto, se deberá dar inicio al programa con la ejecución de la secuencia b0 y b7 – b1 y b6 – b2 y b5 – b3 y b4 en donde las luces se acercan al “centro”, esto se repite de forma reiterada hasta que el usuario presione y suelte el pulsador, donde la secuencia debe cambiar a la siguiente, siendo b7-b6-b5-b4-b3-b2-b1-b0, en donde las luces se prenden en forma escalonada, si el usuario volviera a pulsar y soltar el pulsador, deberá cambiarse la secuencia nuevamente, y así realizarse el cambio consecutivamente. Si el usuario mantuviese el pulsador apretado, la secuencia deberá continuar ejecutándose sin interrupción, hasta que él mismo lo suelte, en donde efectivamente deberá cambiar. Ademas se debera elejir un retardo adecuado para la visualizacion del cambio de color de los diodos LED.

3.3 Resolucion de problema

3.3.1. Desarrollo de las secuencias

La primera secuencia de LEDs se comprende como un movimiento de dos bits “1”, uno que se desplaza a derecha y otro a izquierda, dando el efecto de que se acercan entre sí.
El que se desplegará a derecha se rotará entre los bits b0, b1, b2 y b3, y el otro se rotará entre los bits b7, b6, b5 y b4.
Por medio de dos variables llamadas valor1 y valor2 se realizó el desplazamiento de bits y para saber su posición se usó una variable aux que indica que parte de la secuencia está transcurriendo. Por ejemplo, si aux = 4, esto implica que valor1 tiene el valor alto en b4 y valor2 en b3.
La segunda secuencia a implementar es un desplazamiento de 1 bit a derecha, encendiendo un led por vez, empezando por el D7 hasta el D0 y así continuamente hasta que se presione y se suelte el pulsador.

Por medio de reutilizar valor2 y aux, se realizó el desplazamiento de bits y también se controló su posición.
Se definen dos variables globales(“secuencia” e “inicialización”) para controlar la secuencia de LEDs, al confirmar que el pulsador fue pulsado, la variable inicialización se establece en 1. Esto asegura que, al leer el valor de secuencia, que puede alternar entre 0 y 1, si inicialización es igual a 1, las variables relacionadas con la secuencia se inicializan nuevamente. De esta manera, se evitan errores que podrían surgir de valores residuales en las variables debido a cambios anteriores.

A continuación un Pseudocódigo que muestra la implementación en el Main:

```
Inicializar variable global “secuencia” en 1;
Inicializar variable global “inicialización” en 1;
Main{
  Crear variables locales aux, valor1, valor2
  Inicializar retardo en 0;
  Mientras(true) {
    Chequeamos el pulsador
    si el retardo llegó a 0 {
      reinicio el retardo a 130000;
}
    si la secuencia es igual a 1 {
      si inicialización es igual a 1 {
        inicializar valor1 = 0b10000000;
        inicializar valor2 = 0b00000001;
        inicializar aux = 0;
        inicialización pasa a 0;
}
   si aux es igual a 4 (secuencia 1 completada) {
     inicializar valor1 = 0b10000000;
     inicializar valor2 = 0b00000001;
     inicializar aux = 0;
}
  incrementamos aux en uno;
  escribir en PORTB OR entre valor1 y valor2;
  desplazar valor 1 un bit hacia la derecha;
  desplazar valor2 un bit hacia la izquierda;
```

```
}
sino(se debe hacer la otra secuencia)
{ si inicialización es igual a 1 {
        inicializar valor2 = 0b10000000;
        inicializar aux = 0;
        inicialización pasa a 0;
}
  si aux es igual a 8 (se mostraron todos los LEDs) {
    aux pasa a 0;
    inicializar valor2 = 0b10000000;
}
  PORTB es igual a valor2;
  desplazar valor2 un bit hacia la derecha;
  incremento aux en uno;
}
}
sino (si retardo no es cero) {
  decremento retardo en uno;
}
}
```

3.3.2. Tiempo de visualización

Para la elección de un retardo de visualización en el simulador y debido a que continuamente se debe chequear el estado del pulsador para el cambio de secuencia, se evitó usar la función delay_ms() ya que la misma produce un bloqueo del programa indeseable, para esto se usó un contador “retardo” que inicializa en valor y se decrementa en cada ciclo de programa hasta llegar a cero, donde se produce el avance de la secuencia.
Para el valor de esta variable “retardo”, se usó uno tal que se pueda observar una correcta visualización del programa, siendo este “130000”.
Utilizando breakpoints y suponiendo una frecuencia del MCU de 16MHz, medimos el tiempo de visualización entre cada cambio de led, siendo este aproximadamente “503.76ms” siendo este un tiempo cómodo para la correcta visualización.
A continuación se pueden observar la figura 2.4 y la 2.5 con la simulación con breakpoint en el programa Proteus de tiempo de cambio de LED.

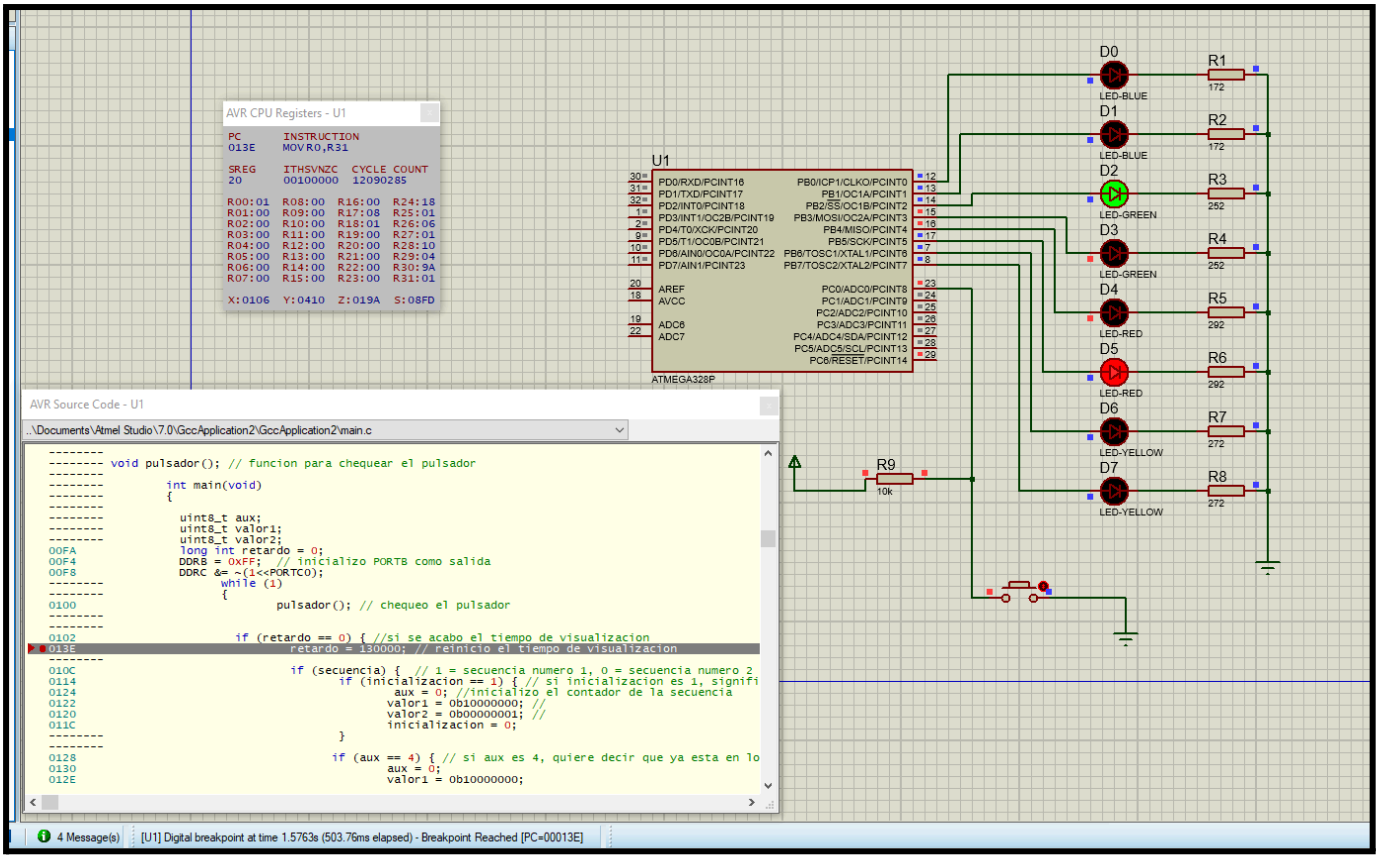

Figura 3.1 Simulación en Proteus con breakpoint, para ver el tiempo de visualización de los LEDs.

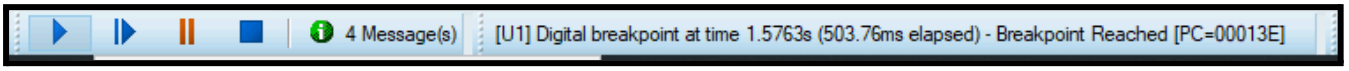

Figura 3.2 Tiempo de visualización de los LEDs.

3.3.3. Tiempo de chequeo de pulsador

Para la elección de un tiempo de chequeo en la detección antirebote, se decidió usar una variable “rebote”, la cual se refiere al tiempo de espera que se aplica después de que se detecta un cambio de estado en el botón antes de considerar que el botón ha sido presionado o soltado de manera definitiva. Mediante una variable contador dentro del chequeo del pulsador, incrementada a medida que el pulsador se encuentra presionado, cuando la misma llega al valor de la variable rebote, se establece el cambio definitivo del estado del pulsador.

El valor de “rebote” se determinó empíricamente ajustando el valor hasta que se logró un comportamiento adecuado del pulsador en el sistema, resultando este en “3000”.

A continuación se pueden observar la figura 2.6 y la 2.7 con la simulación con breakpoint en el programa Proteus y el tiempo de chequeo del pulsador.

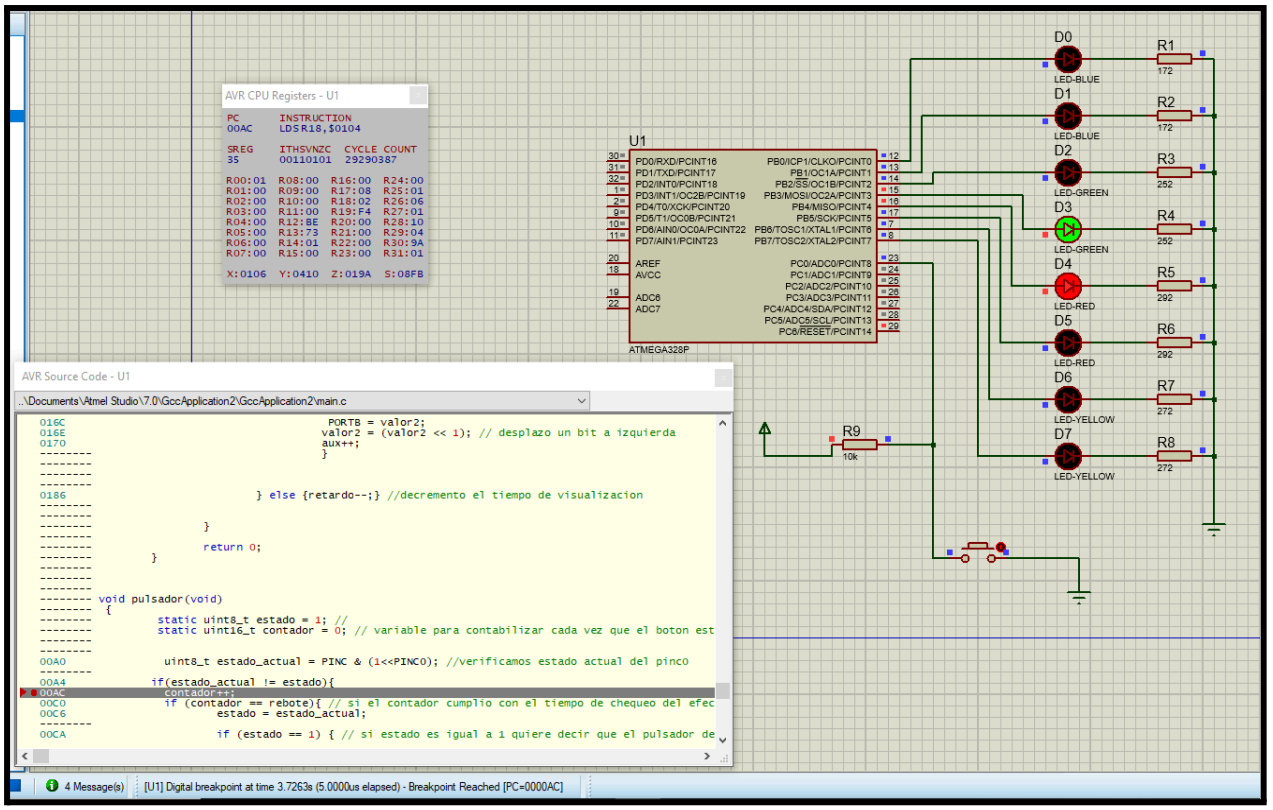

Figura 3.3 Simulación en Proteus con breakpoint, para ver el tiempo de chequeo del pulsador.

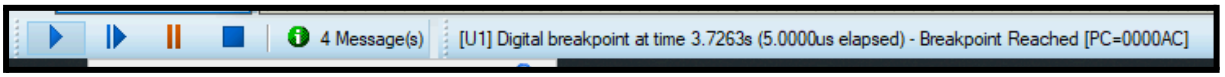

Figura 3.4 Tiempo de chequeo de pulsador.

4. Conclusión

4.1 Enunciado

Saque conclusiones sobre el funcionamiento del programa, sobre las ventajas y desventajas de utilizar retardos (delays) para temporizar acciones y cómo estos afectan el tiempo de respuesta a la acción del usuario. Hágase preguntas como por ejemplo: ¿qué sucede si se deja presionado constantemente el pulsador? ¿qué sucede si el retardo es de 1 segundo?

4.2 Desarrollo de la conclusión

El funcionamiento del programa cumple todo lo solicitado en los incisos anteriores, creemos aun así que hablando en sentido de eficiencia, se podrían encontrar diferentes formas por las cuales llegar a un resultado aún más eficiente.
Al utilizar un algoritmo de detección del pulsador con contador se evita que la ejecución del programa se paralice durante el intervalo de tiempo en el que el pulsador es presionado. De esta manera, si el usuario mantiene el pulsador, se logra que el programa continúe en ejecución con la secuencia actual de encendido de LEDs correctamente hasta que el usuario suelte el pulsador y ahí efectivamente se cambie a la otra secuencia.

Las ventajas de utilizar retardos que pausan la continuidad del programa es que el código termina siendo más simple ya que la función delay se encuentra ya creada por defecto y además se acopla bien a la frecuencia de reloj elegida.

Su desventaja principal, es que el MCU se queda ocioso esperando a que termine el tiempo de delay elegido y esto es un gran problema.
Otra de las desventajas que tiene el utilizar delays es su inflexibilidad, es decir, que los retardos fijos no pueden adaptarse bien a cambios en las condiciones del sistema o en los requisitos del tiempo. Si se dejara presionado constantemente el pulsador haciendo uso de delays, lo que ocurriría sería que la secuencia de LEDs se “congelaría” hasta que se suelte el pulsador y cambie de secuencia, es decir, no permitiría que la secuencia presente siga mientras se presiona el pulsador.

5. Bibliografía

- Atmel Corporation (2015). ATmega 328P Datasheet. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

6. Anexo

Simulación y programa completo escrito en C.

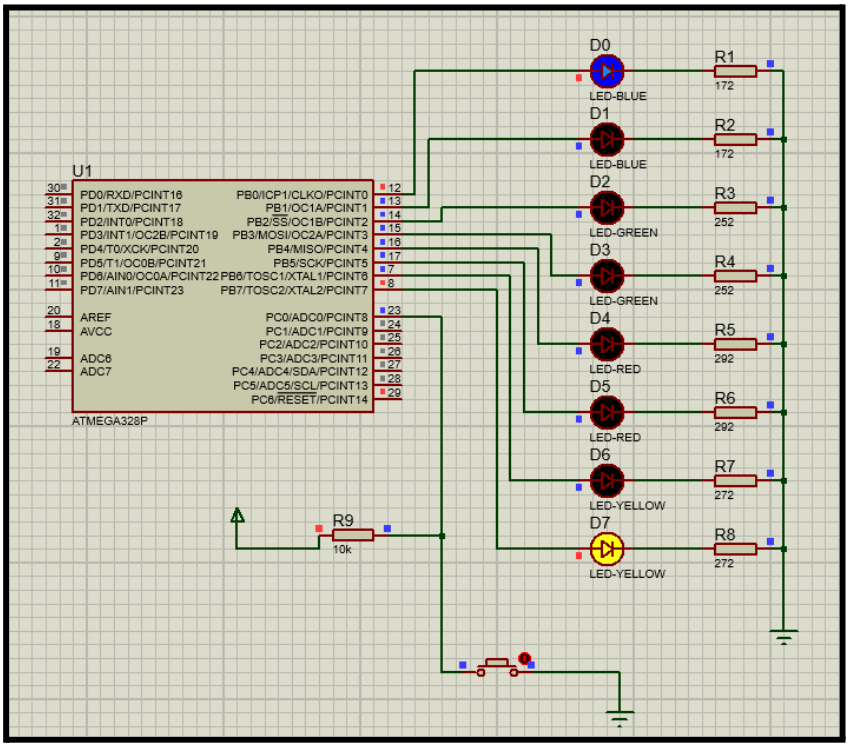

Figura 5.1 Detección de botón pulsado.

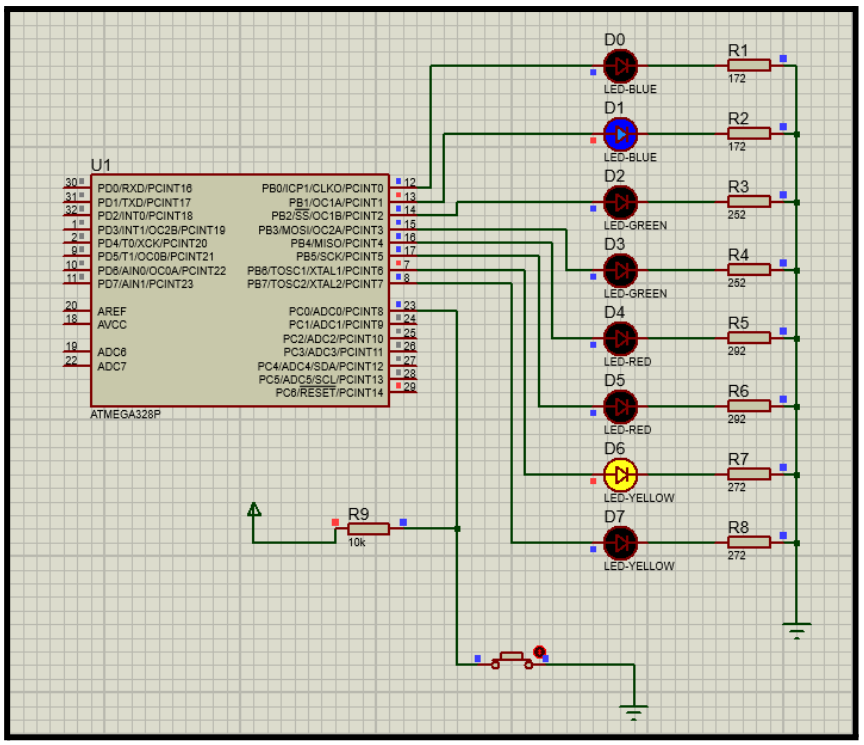

Figura 5.2. Continuación de secuencia con pulsador presionado.

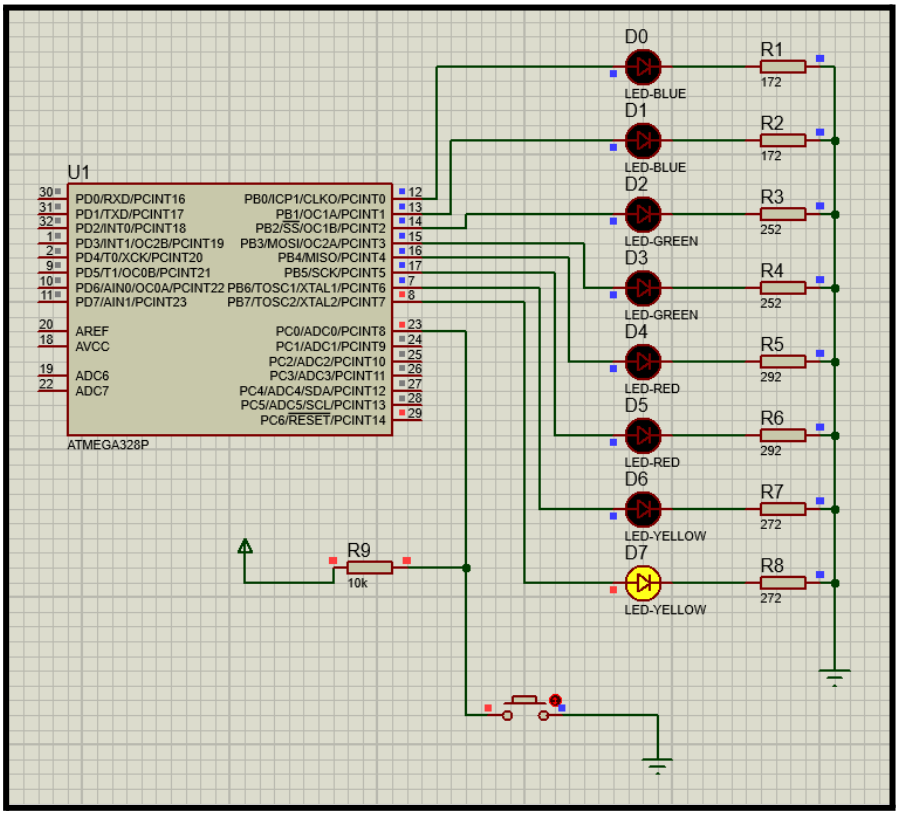

Figura 5.3. Cambio de secuencia al soltar el pulsador.

CODIGO MAIN.C

```
#include <avr/io.h>
#define F_CPU 16000000UL // Especifico la frecuencia de reloj del MCU en 16MHz
#include <util/delay.h> // Retardos por software - Macros: depende de F_CPU
#include <stdio.h>
#include <stdlib.h>
#define rebote 3000  // variable para deteccion de cambio de estado del pulsador


// variables globales
uint8_t secuencia = 1;
uint8_t inicializacion = 1;

void pulsador(); // funcion para chequear el pulsador

    int main(void)
    {

      uint8_t aux;
      uint8_t valor1;
      uint8_t valor2;
      long int retardo = 0;
      DDRB = 0xFF;  // configurado PORTB como salida
      DDRC &= ~(1<<PORTC0); //el bit0 de PORTC configurado como entrada
        while (1) // loop
        {
            pulsador(); // chequeo el pulsador


          if (retardo == 0) { //si se acabo el tiempo de visualizacion
              retardo = 130000; // reinicio el tiempo de visualizacion

              if (secuencia) {  // 1 = secuencia numero 1, 0 = secuencia numero 2
                 if (inicializacion == 1) { // si inicializacion es 1, significa
                   que la secuencia comienza.
                     aux = 0; //inicializo el contador de la secuencia
                   valor1 = 0b10000000;
                   valor2 = 0b00000001;
                   inicializacion = 0;
                 }

                if (aux == 4) { // si aux es 4, se reinicia la secuencia
                    aux = 0;
                    valor1 = 0b10000000;
                    valor2 = 0b00000001;
                }
                aux++;  //incremento  el contador de la secuencia
                PORTB = valor1 | valor2;  // Le asigno a PORTB el valor actual de
                  la secuencia
                valor1 = (valor1 >> 1);  // desplazo un bit a derecha
                valor2 = (valor2 << 1); // desplazo un bit a izquierda

              }
              else { //secuencia numero 2
```

```
                    if (inicializacion == 1) { // si inicializacion es 1, significa
                      que la secuencia comienza.
                        aux = 0;
                        valor2 = 0b10000000;
                        inicializacion = 0;
                    }

                   if (aux == 8) { // si aux llega a 8 debe volver a empezar la
                     secuencia
                       aux = 0;
                       valor2 = 0b10000000;
                   }
                    PORTB = valor2;  //le asigno a PORTB el valor actual de la
                      secuencia
                   valor2 = (valor2 >> 1); // desplazo un bit a derecha
                   aux++;
                   }


            } else {retardo--;} //decremento el tiempo de visualizacion


        }

        return 0;
    }



void pulsador(void) // funcion de chequeo del pulsador sin efecto rebote
 {
     static uint8_t estado = 1;
     static uint16_t contador = 0;


      uint8_t estado_actual = PINC & (1<<PINC0); //verificamos estado actual del
        pinc0

    if(estado_actual != estado){
        // se incrementa el contador hasta confirmar el cambio de estado
      contador++;
      if (contador == rebote){ // si el contador cumplio con el tiempo de chequeo
        del efecto rebote
          estado = estado_actual;

          if (estado == 1) { // si estado es igual a 1 quiere decir que el pulsador
             dejo de estar presionado
        secuencia = secuencia ? 0 : 1;  //cambio de secuencia
        inicializacion = 1; // inicializacion en 1 para la nueva secuencia
        contador = 0; //reinicio el contador
          }
      }
       }
       else {contador = 0;}// si no cumple contador == rebote, se reinicia a 0

}
```