UNIVERSIDAD
NACIONAL
DE LA PLATA

Facultad de Ingeniería - UNLP
Circuitos Digitales y Microcontroladores (E1305 / E0305)
Curso 2022 - Trabajo Práctico Nº1

Grupo 2

Alumnos: Santana Valentin, Guerrero Nicolás

Control de periféricos externos con puertos de entrada/salida.
1. Esquema eléctrico de la conexión en Proteus

1.1 Problema

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 Resolución del problema

En cuanto a la arquitectura, utilizamos el microcontrolador ATMega328p, cuya frecuencia de reloj establecimos en 16MHz. Los periféricos requeridos para resolver este inciso fueron leds de distintos colores y voltajes, resistencias y amperímetros. A fin de calcular la resistencia serie de cada led particular para que no supere la capacidad de corriente de 10mA, utilizamos la siguiente fórmula:

$$\frac{U_{MCU} - U_{LED}}{I_{CC}} = R_{LED}$$

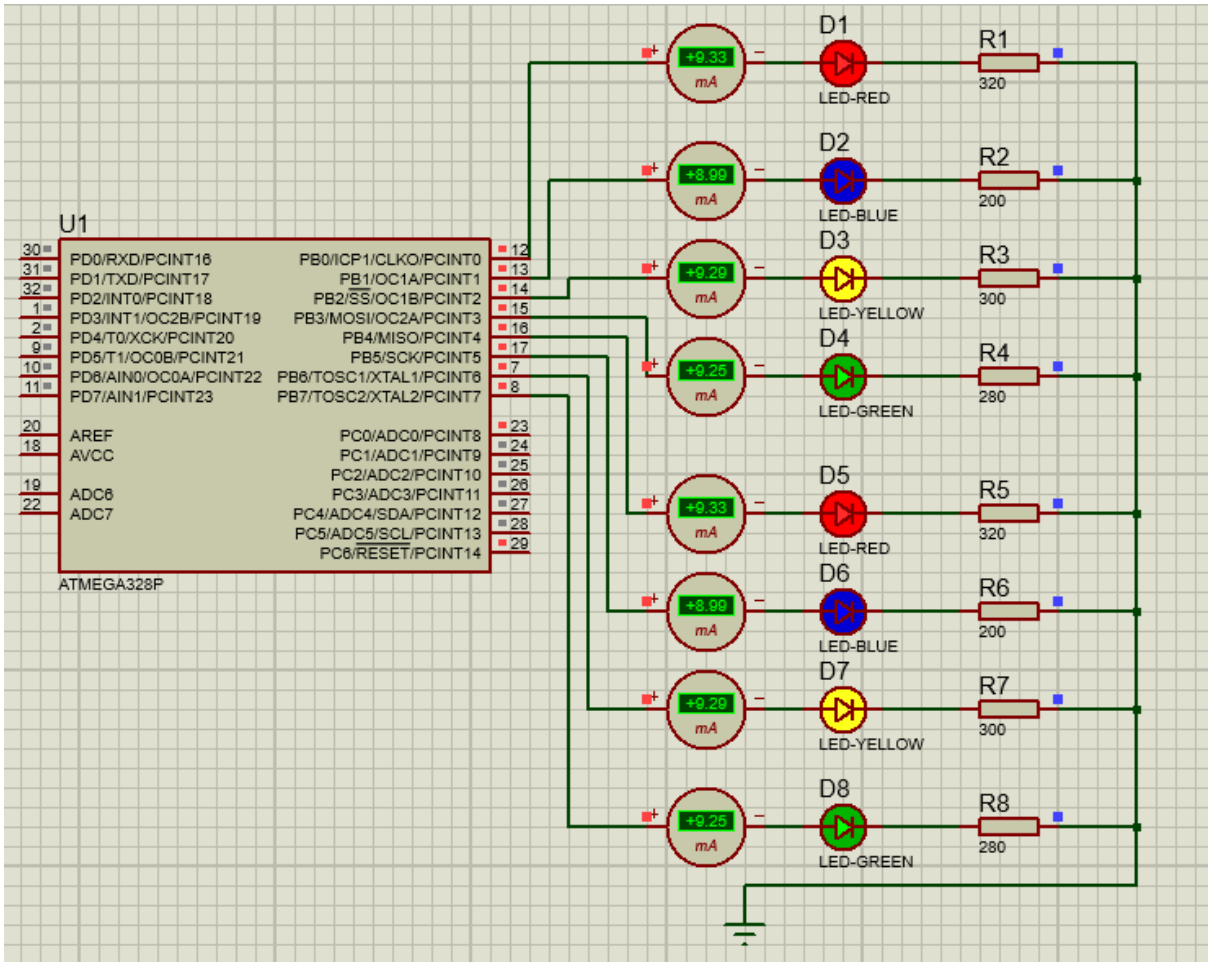
Luego, los resultados obtenidos fueron:

Resistencia led rojo: 320Ω
Resistencia led azul: 200Ω
Resistencia led amarillo: 300Ω
Resistencia led verde: 280Ω

Utilizando estos valores para las resistencias pudimos observar mediante los amperímetros que la corriente por cada terminal del MCU no supera la capacidad de corriente de cada salida. Así los valores de la misma, nunca superaron los 10 mA, siendo los valores de corriente de que pasan por cada led:
Led rojo: 9,33 mA
Led azul: 8,99 mA
Led amarillo: 9,29 mA
Led verde: 9,25 mA
En cuanto a la parte de software, el único código necesario fue:
Poner el puerto B como salida.
Poner todos los bits del puerto B en 1.

En C:
DDRB = 0xFF; // Puerto B configurado para salida.
PORTB = 0xFF // Enciendo todos los leds

1.3 Validación



2. Pulsador y efecto rebote

2.1 Problema

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

Se conecta un pulsador a la entrada del MCU para que el usuario pueda utilizarlo. Para esto será necesario habilitar una entrada en el MCU que reciba una señal

cuando el botón esté pulsado y cuando el mismo se deje de pulsar. Se debe tener en cuenta que para poder evitar un efecto rebote al momento de usar el pulsador será necesario tener un algoritmo en el código que resuelva este problema

2.3 Resolución del problema

Se habilitó la entrada del puerto PC0 del MCU para que reciba una señal en alto y que cuando el pulsador se oprima reciba una señal en bajo (resistencia pull-up). Como el Atmega328P ya cuenta con una resistencia interna, no fue necesario agregar una externa entre la entrada del PC0 y el pulsador.

Luego se creó un algoritmo que permite saber si la señal recibida es verdadera, generando un retardo en el programa para mitigar el efecto rebote generado al pulsar y soltar el botón. Este algoritmo cuenta con dos **if anidados**, uno para saber si el botón está presionado y otro para saber si no está presionado. Si se recibe una señal, el programa tendrá un retardo de 20 ms (se usó por recomendación de la cátedra) y volverá a preguntar si esa señal sigue siendo la misma, con esto podremos asegurar que la señal era verdadera y no una derivación del efecto rebote producido por el botón.

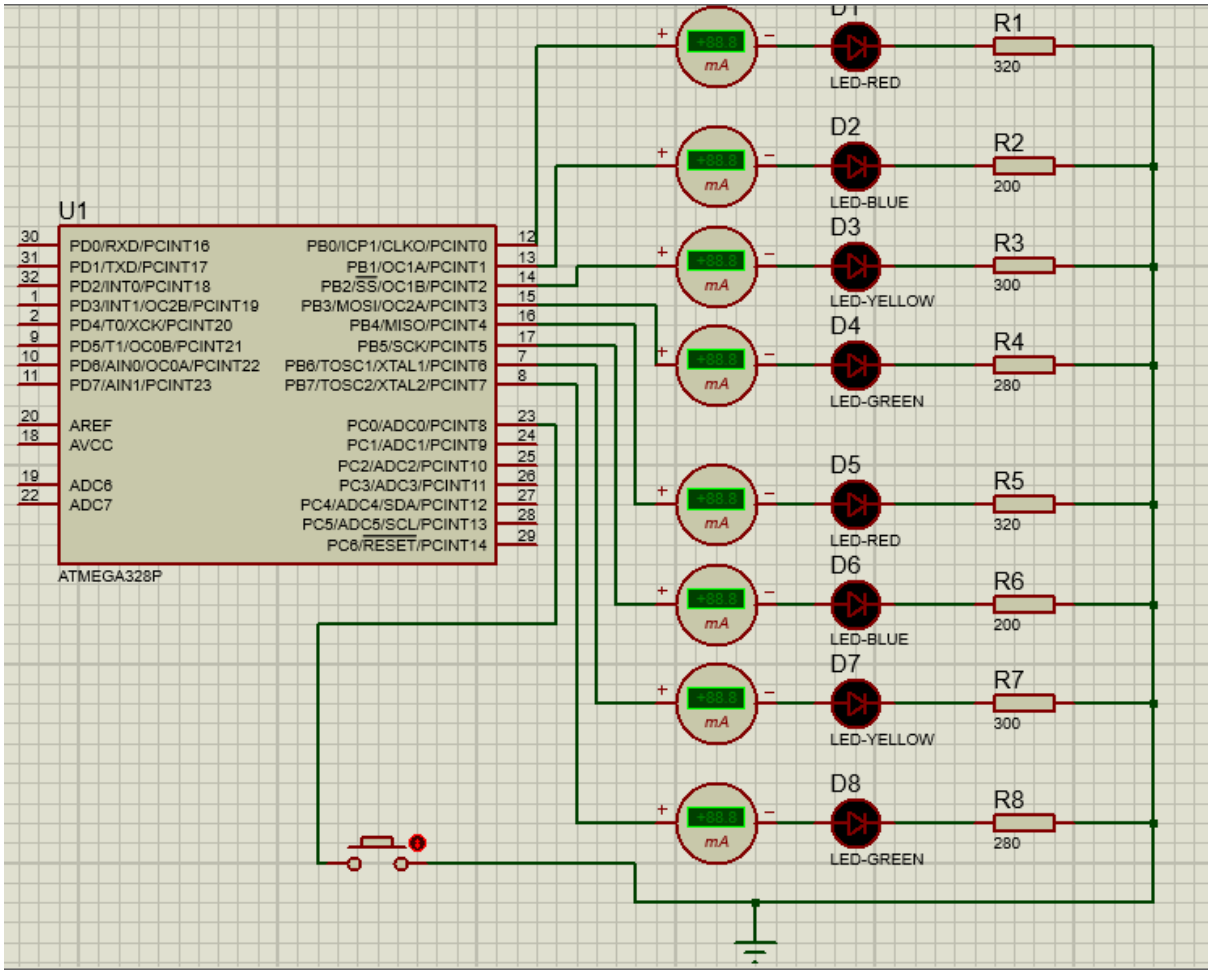
Pseudocódigo módulo comprobar:

Se tiene una variable global condición que señala si el estado anterior del botón.

Si el botón está presionado
espero un tiempo por el efecto rebote(delay).
Si el botón sigue presionado (señal verdadera) y anteriormente el botón no estaba presionado
Cambio el estado de la condición.

Si el botón no está presionado
espero un tiempo por el efecto rebote(delay).
Si el botón no está presionado (señal verdadera) y anteriormente el botón estaba presionado
Cambio el estado de la condición.

2.4 Validación



3.1 Problema

Realice el programa para que el MCU encienda los LEDs del puerto B con la siguiente secuencia 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: b3 y b4 – b2 y b5 – b1 y b6 – b0 y b7. Si presiona y suelta nuevamente vuelve a la secuencia original y así sucesivamente. Elija un retardo adecuado para la visualización. Justifique.

3.2 Interpretación

Se deberá agregar una variable al algoritmo creado para la pulsación del botón para que se sepa cuando el usuario dejó de presionar el botón, esto será necesario para cambiar el patrón de los leds.

También se tendrá que tener en cuenta que, si el usuario deja presionado el botón más de la cuenta, el programa debería seguir con su funcionamiento normal hasta que esté lo suelte, evitando así un bloqueo del mismo.

3.3 Resolución del problema

Se creó un módulo denominado **Inicializar**, el cual recibe como parámetro una variable condición, que será la responsable de decidir qué patrón de encendido se utilizará en los leds. Para lograrlo el módulo además recibe dos variables int por referencia que serán las que se inicializarán de acuerdo con el patrón a seguir.

El programa principal será donde estas variables se incrementen o decrementen para después enviarlas como señal al PORTB.

Pseudocódigo módulo Inicializar:

Se tiene una variable global condición que señala si el estado anterior del botón.

Si la condición es verdadera
  Inicializo el patrón A (**variableA** y **variableB**) Ver Fig. 3.4.1
Sino
  Inicializo el patrón B (**variableA** y **variableB**) Ver Fig. 3.4.1

Pseudocódigo programa principal:

Se tiene una variable global condición que señala si el estado anterior del botón.
Se tiene una variable global contador que empieza en 4.

Si el contador es mayor o igual a 4
  Llamo al módulo inicializar
  Reinicio el contador
Llamo el módulo comprobar (Previamente explicado en el inciso 2.3)
Prendo los leds correspondientes (PORTB=(1<<**variableA**)+(1<<**variableB**))
Realizo un retardo para la correcta visualización de los leds
Sumo el contador y la **variableA**
Resto la **variableB**

3.4 Validación

```
if(condicion){
     *i=0;
     *j=7;
}
else{
     *i=4;
     *j=3;
}
```

Fig 3.4.1

4.1 Problema

Saque conclusiones sobre el funcionamiento del programa, sobre las ventajas y desventajas de utilizar un retardo bloqueante y cómo este afecta el tiempo de respuesta del programa ¿qué sucede si se deja presionado constantemente el pulsador?

4.2 Resolución del problema

Una posible solución para contrarrestar el efecto rebote era utilizar un retardo bloqueante en el programa que consistía en que, mientras se pulsara el botón, un bucle **while** generaría un delay de 20 ms hasta que este se soltara. Una ventaja de este tipo de código es que es sencillo por lo que podría decirse que el mantenimiento del código es fácil. Sin embargo, es poco eficiente y hay una pérdida de rendimiento porque el programa no puede continuar a menos que se de la situación esperada en el **while**, además de generar una experiencia poco placentera para el usuario.

Debido a lo anteriormente explicado, realizamos la solución de manera que al mantener presionado el botón el programa continúe con su ejecución y su mismo patrón de leds hasta que deje de ser presionado, con esto se logró una mayor eficiencia en el programa.

Anexo

main.c

```
#include <avr/io.h>
#define F_CPU 16000000UL          // Se trabaja con un microprocesador de 16 MHz.
#include <util/delay.h>

void comprobar(int *,int *,int *);
void inicializar(int ,int *,int *);

int main(void){

	DDRB = 0xFF;                              // Puerto B configurado para salida.
	DDRC &=~(1<<PC0);          // Bit 0 del Puerto C configurado para entrada.
	PORTC |= (1<<PC0);         // Resistencia pull-up (se actica señal en bajo).

	int condicion = 1;         // Condicion inicial
	int i,j,ok=1,cont=4;

	while (1)
	{
		if(cont>=4 ){
			inicializar(condicion,&i,&j);     // Inicializo valores de i y j
                                                  // para enceder leds.
			cont=0;
```

```
                }
                // Ciclo de prendido de luces leds
                comprobar(&ok,&condicion,&cont);   // Compruebo si el boton se pulso.
                PORTB = (1<<i) + (1<<j);           // Prendo leds correspondientes.
                _delay_ms(50);
                i++;
                j--;
                cont++;
        }
}


// Comprobacion del estado del pulsador.

void comprobar(int *ok, int *condicion,int *cont){
        if((PINC & (1<<PINC0))==0){              // Pregunto por estado del boton.
                _delay_ms (20);                  // Efecto rebote (de 1 a 0).

                // Pregunto si el botón se pulsó (si la señal esta en bajo y “ok”
                // es verdadero significa que antes la señal era alta)
                if(((PINC&(1<<PINC0))==0) &(*ok)){
                        *ok=0;
                }
        }
        if((PINC&(1<<PINC0))==1){                // Pregunto por estado del boton.
                _delay_ms(20);                   // Efecto rebote (de 0 a 1).

                // Pregunto si el botón se pulsó (si la señal esta en alto y “ok”
                // es falso significa que antes la señal era baja)
                if(((PINC&(1<<PINC0))==1) &(!*ok)){
                        *ok=1;
                        *condicion=!*condicion;       // Cambio condición para alterar el
                                                      // patron
                        *cont=4;      // Cambio contador a 4 para que termine
                                      //el patrón de leds y comience el otro.
                }
        }
}

// Alteración del patrón de los leds.

void inicializar (int condicion, int *i,int *j){
        if(condicion){          // Si la condición es verdadera modifico i y j para que el
                                // patrón sea el de: b0-b7, b1-b6, b2-b5, b3-b4.
                *i=0;
                *j=7;
        }
        else{                   // Si la condición es falsa modifico i y j para que
                                // el patrón sea el de: b3-b4, b2-b5, b1-b6, b0-b7.
                *i=4;
                *j=3;
        }
}
```