Trabajo Práctico N°01

Control de periféricos externos mediante puertos de E/S

Circuitos Digitales y Microcontroladores – UNLP

NOTA 9.5
redondeable a 10

CALDERÓN Sergio Leandro, ERCOLI Juan Martín

11 de abril de 2022

MUY PROLIJO EL INFORME

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 V_{LED} (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

En el programa de simulación Proteus, se deberá crear un nuevo proyecto y agregar el microcontrolador ATmega 328P y, por otra parte, 8 diodos LED de distintos colores. Cada uno de los diodos deberá conectarse a un pin diferente que se corresponda al puerto B, y además todos deberán ser conectados a masa (componente GROUND) para cerrar el circuito.

De acuerdo al color, cada diodo producirá una determinada caída de tensión que deberá averiguarse, para luego calcular el valor en Ohms de la resistencia a conectar en serie con cada uno, considerando también el dato de la corriente solicitada de 10mA y la tensión de salida en los pines del microcontrolador en concordancia a la tensión de alimentación proporcionada.

Por último, deberán utilizarse instrumentos de medición proporcionados por la herramienta Proteus para verificar que los valores medidos se correspondan con los cálculos realizados, o en su defecto, que por cada diodo no circule una corriente superior a 10mA.

1.3– Resolución

1.3.1 – Agregado del MCU

Para la incorporación de los componentes necesarios, se abre el menú de búsqueda mediante *Library > Pick Parts*, se selecciona el elemento correspondiente y se coloca el mismo en la hoja de proyecto. El nombre del microcontrolador a utilizar en este trabajo es "ATMEGA328P". Por defecto, se le proporciona una tensión de alimentación V_{CC} de 5V. La representación del mismo se muestra en la Figura 1.1 a continuación.

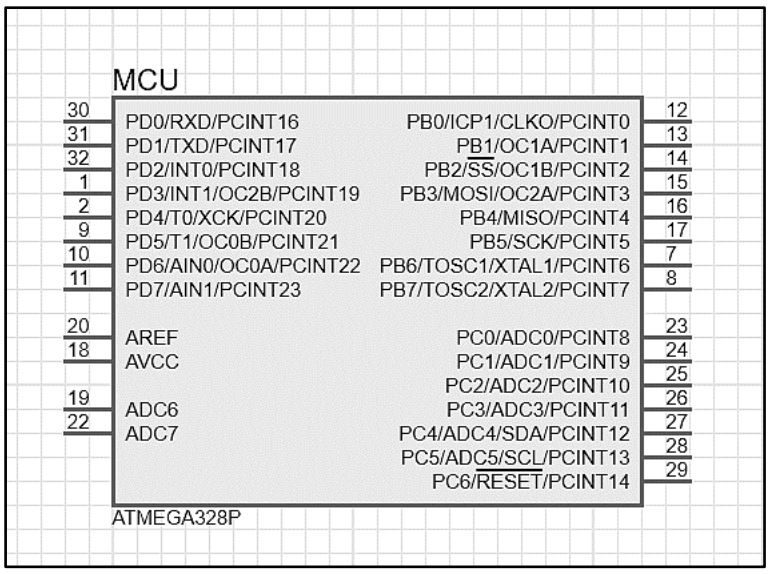

Figura 1.1. Microcontrolador ATmega 328P en el software Proteus.

Como se puede observar, los pines del puerto B, denominados PB0, PB1, …, PB7, corresponden a los terminales 12, 13, 14, 15, 16, 17, 7 y 8 respectivamente.

1.3.2 – Agregado de LEDs

Respecto a los diodos LED, se utilizan los colores rojo, naranja, amarillo, verde, turquesa, azul, rosa y blanco, cuyos valores de caída de tensión o *Forward Voltage* se encuentran tabulados a continuación. Nótese que ninguno supera los 5V (valor de V_{CC}).

Tabla 1.1. Propiedades de los diodos LED de acuerdo a su color emitido.

Description	Chemistry	# of Elements	Color Temperature (CCT Typ)	Peak Wavelength (λ / x-coord)	Dominant Wavelength (λ / y-coord)	Forward Voltage (Vf Typ)	Forward Voltage (Vf Max)
High Efficiency Red	GaP	2	~	700	660	2.0	2.5
Super Red	GaAlAs	3	~	660	640	1.7	2.2
Super Red	AlInGaP	4	~	660	640	2.1	2.5
Super High Intensity Red	AlnGaP	4	~	636	628	2.0	2.6
High Intensity Red	GaAsP	3	~	635	625	2.0	2.5
TS AlInGaP Red	AlInGaP	4	~	640	630	2.2	2.8
Super Orange	AlInGaP	4	~	610	602	2.0	2.5
Amber	GaAsP	3	~	605	610	2.0	2.5
Super Yellow	AlInGaP	4	~	590	588	2.0	2.5
TS AlInGaP Yellow	AlInGaP	4	~	590	589	2.3	2.8
Yellow	GaAsP	3	~	590	588	2.1	2.5
Super Ultra Green	AlInGaP	4	~	574	568	2.2	2.6
Green	GaP	2	~	565	568	2.2	2.6
Super Green	GaP	2	~	565	568	2.2	2.6
Pure Green	GaP	2	~	555	555	2.1	2.5
Ultra Pure Green	InGaN	3	~	525	520	3.5	4.0
Ultra Emerald Green	InGaN	3	~	500	505	3.5	4.0
Ultra Super Blue	InGaN	3	~	470	470	3.5	4.0
Ultra Violet	InGaN	3	~	410	~	3.5	4.0
Super Violet	InGaN	3	~	380	~	3.4	3.9
Turquoise	InGaN	3	~	0.19	0.41	3.2	4.0
Violet / Purple	InGaN	3	~	0.22	0.11	3.2	4.0
Pink	InGaN	3	~	0.33	0.21	3.2	4.0
Warm White	InGaN	3	3000K	~	~	3.3	4.0
Neutral White	InGaN	3	4000K	~	~	3.3	4.0
Cool White	InGaN	3	6000K	~	~	3.3	4.0

A partir de los valores de la Tabla 1.1, se establecen las propiedades de cada diodo "n" en Proteus, indicando el tipo de modelo como "Analógico" y la tensión correspondiente V_{Dn}.

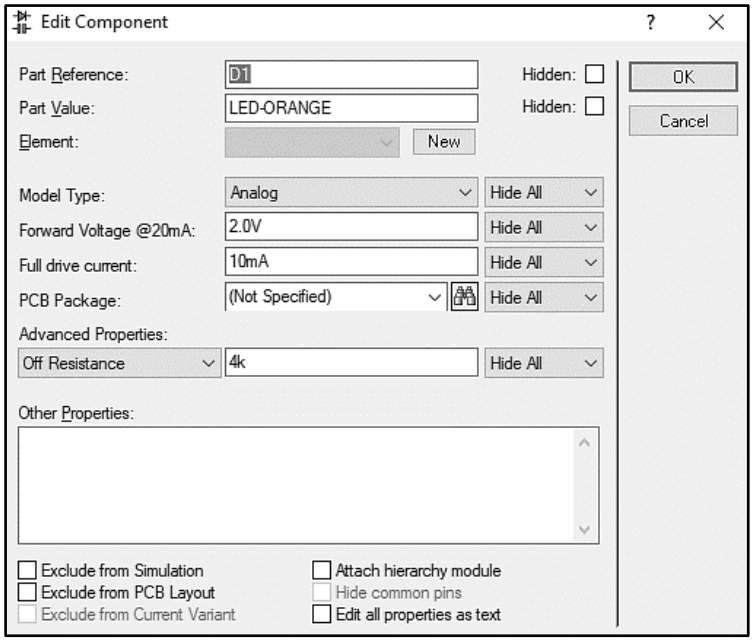

Figura 1.2. Cuadro de configuración de un componente tipo LED.

1.3.3 – Agregado de resistores

A cada diodo LED se le conecta una resistencia en serie que permita producir una segunda caída de tensión $V_{Rn} = V_{CC} - V_{Dn}$, de modo que la corriente I que circula por el conductor sea menor y evitar que el componente emisor de luz se dañe. Luego, mediante Ley de Ohm, se obtiene el valor de la resistencia $R_n = V_{Rn} \,/\, I$, siendo $I = 10mA$.

Tabla 1.2. Caída de tensión en cada diodo y su resistencia serie, y valor de dicha resistencia.[1]

n	Color	Nombre	V_{Dn} [V]	V_{Rn} [V]	R_n [Ω]
0	Rojo	RED	1.8	3.2	320
1	Naranja	ORANGE	2.0	3.0	300
2	Amarillo	YELLOW	2.0	3.0	300
3	Verde	GREEN	2.2	2.8	280
4	Turquesa	AQUA	3.2	1.8	180
5	Azul	BLUE	3.0	2.0	200
6	Rosa	PINK	3.2	1.8	180
7	Blanco	WHITE	3.3	1.7	170

[1] Los valores V_{Dn} para los colores rojo, amarillo, verde y azul son aquellos proporcionados en el enunciado.

En Proteus, se establece la resistencia en Ohms de cada resistor $R_0, R_1, \dots, R_7$ según los cálculos de la Tabla 1.2, y se conectan a masa (GROUND), resultando el siguiente circuito:

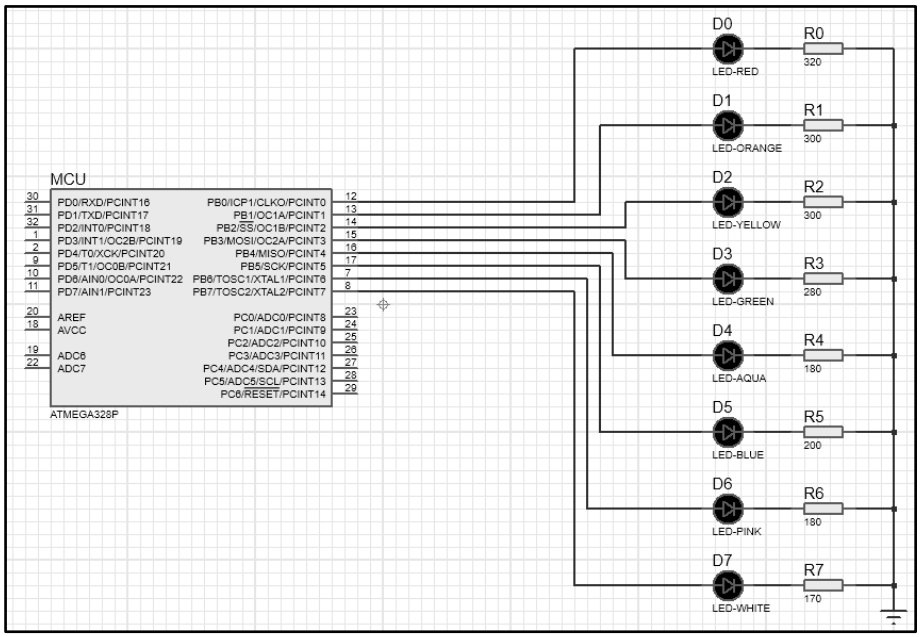

Figura 1.3. Esquema eléctrico compuesto por MCU, LEDs y resistores en serie.

1.3.4 – Agregado de instrumentos

Para la medición de corriente que atraviesa cada uno de los diodos LED, se requiere la conexión de amperímetros en serie. Su componente en Proteus se denomina “DC AMMETER” y se encuentra en el grupo “INSTRUMENTS”. El circuito final se muestra a continuación.

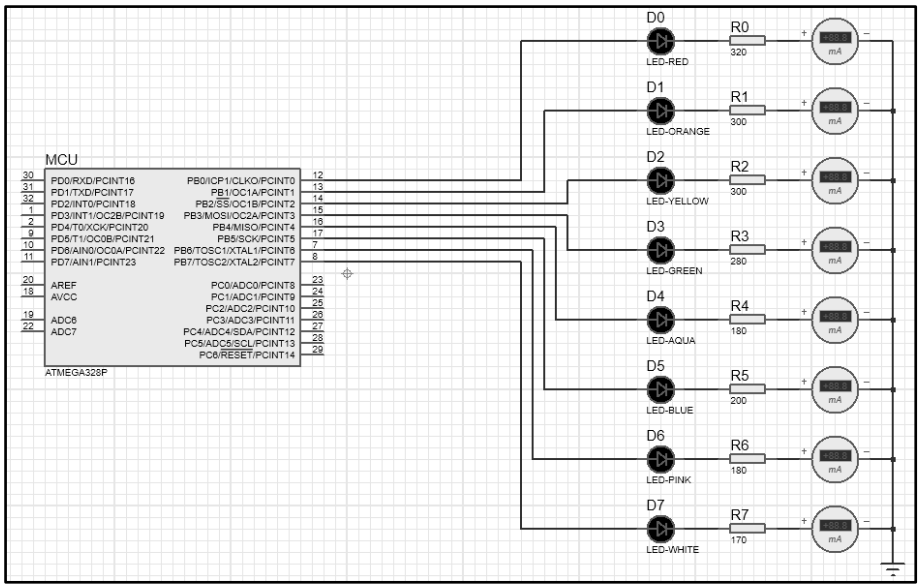

Figura 1.4. Esquema eléctrico con instrumentación añadida.

1.4 – Simulación

La visualización de las mediciones de corriente se realiza mediante la ejecución de una simulación del circuito. Sin embargo, para el encendido de todos los diodos LED se requiere:

- Que todos los terminales del puerto B estén configurados como salida.
- Que se escriba un “1” en todos los bits del registro de datos del puerto B.

Las dos condiciones anteriores se consiguen mediante las siguientes 2 sentencias:

- DDRB = 0xFF;
- PORTB = 0xFF;

Es necesario agregar dichas sentencias de C al *main* del código fuente, y luego compilar el mismo para finalmente cargar el archivo de extensión “*.elf”* como *Program File* en el microcontrolador colocado en Proteus. El resultado de la simulación es el siguiente:

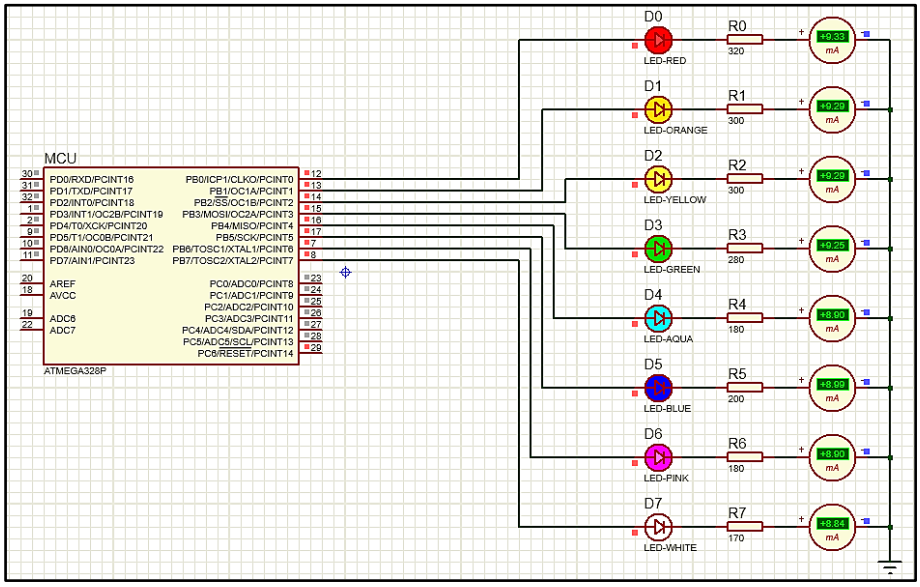

Figura 1.5. Simulación del esquema eléctrico con todos los diodos encendidos.

Como se puede observar en la Figura 1.5, la corriente que circula por cada rama no es exactamente 10mA, sino un valor hasta 12% por debajo del mismo. La explicación a esta diferencia se encuentra en que la tensión de salida del microcontrolador no es V_{cc} (5V) para todo valor de corriente, sino que se comporta como las fuentes reales de tensión, entregando menor tensión a medida que la corriente aumenta. Por ejemplo, para el LED rojo, el amperímetro arroja una medición de 9,33mA, por ende, la tensión a la salida de PB0 es:

$$V_{OH} = V_{D0} + I_0 * R_0 = 1.8V + 0.00933A * 320\Omega \cong 4.78V$$

Este efecto se indica en la hoja de datos del ATmega 328P (p. 271, cap. 29), cuya gráfica se muestra a continuación. Nótese que para 25°C y $I_{OH} = 10mA$ se tiene $V_{OH} \cong 4.75V$

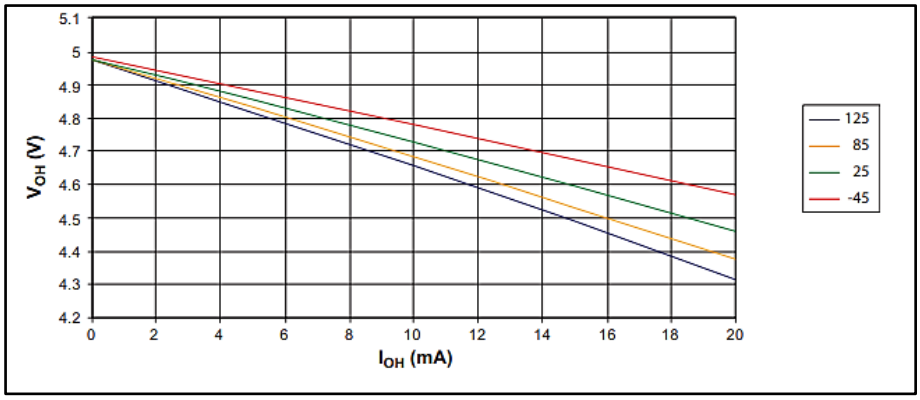

Figura 1.6. Tensión de salida versus corriente de salida en los terminales I/O para $V_{CC} = 5\ V$.

Se cumple entonces que la corriente que atraviesa cada diodo LED es menor a 10mA. Por otra parte, el fabricante del microcontrolador también especifica que la suma de las corrientes de los puertos B0 a B5 no debe superar los 150mA. Esto último también se cumple en este caso, ya que como máximo la suma es 60 mA si circula hasta 10 mA por cada rama.

$$\sum_{i=0}^{5} I_{OH}(i) = 9{,}33\ mA + 2 * 9{,}29\ mA + 9{,}25\ mA + 8{,}90\ mA + 8{,}99\ mA = 55{,}05\ mA$$

4. Although each I/O port can source more than the test conditions (20mA at V_{CC} = 5V, 10mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
ATmega328P:
1] The sum of all I_{OH}, for ports C0 - C5, D0- D4, should not exceed 150mA.
2] The sum of all I_{OH}, for ports B0 - B5, D5 - D7, XTAL1, XTAL2 should not exceed 150mA.
If I_{OH} exceeds the test condition, V_{OH} may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.

Figura 1.7. Fragmento (p. 258, cap. 28) donde se especifica la suma máxima de I_{OH}

2 – Adición 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 programa de simulación Proteus, se deberá conectar un pulsador a un terminal configurado como entrada del microcontrolador ATmega 328P, y para poder mostrar más adelante como el mismo detecta la acción de presionar y soltar producida por el usuario.

Se realizará una investigación sobre el efecto de rebote que se produce en los pulsadores y se implementará una función de C que permita eliminar el mismo y asegurar una correcta detección del cambio de estado del pulsador (pulsado o en reposo) para el correcto funcionamiento de los programas a implementar.

2.3 – Resolución

2.3.1 – Efecto rebote

El efecto rebote asociado a los pulsadores hace referencia a lo que sucede cuando los mismos son pulsados o soltados. Cuando un pulsador es accionado, los contactos deben moverse físicamente de una posición a otra, y, hasta que estos contactos se estabilicen en la nueva posición, se producirá un efecto de rebote mecánico, causando que el circuito se abra y cierre múltiples veces. Es decir, cuando un pulsador es presionado o soltado se produce una fluctuación en la señal que pasa por sus contactos internos, esto produce que se pase de un estado HIGH (1) a un estado LOW (0) o viceversa en periodos muy cortos, y al ser leída por el MCU puede producir comportamientos inesperados en el funcionamiento del mismo.

Se puede observar en la Figura 2.1 una representación del efecto rebote en un pulsador.

Figura 2.1. Cambios en la tensión debido al efecto rebote visto en un diagrama temporal.

2.3.2 – Algoritmo de detección anti-rebote

La función consiste en chequear el estado del pulsador cada cierto tiempo, se considera que el mismo se encuentra pulsado, o bien, en reposo, si se mantiene en el mismo estado durante un par de lecturas, es decir se 'espera' hasta que la señal se estabilice. Esto permite mitigar el efecto rebote en los primeros instantes cuando el usuario acaba de pulsar o soltar el botón. El pseudocódigo de su implementación se muestra a continuación.

```
Inicializar estado guardado del pulsador como reposo
Inicializar contador de ciclos en cero

Cada vez que se cumple un ciclo
    Leer estado actual del pulsador
    Si el estado actual es distinto del estado guardado
        Incrementar contador en uno
        Si contador alcanza un valor determinado
            Asignar estado actual en estado guardado
            Realizar acción según el estado guardado
            Resetear contador
    Sino
        Resetear contador
```

Pseudocódigo 2.1. Función de detección de pulsación

Como puede observarse, el contador y el estado guardado o "confirmado" deben inicializarse únicamente en el primer llamado a la función (identificador *static*). Sus valores pueden ir cambiando a medida que se realizan lecturas del estado del pulsador. Luego, sólo cuando el contador alcanza un valor establecido se realiza una acción u otra según corresponda.

Además, al no usar retardos se evita que la ejecución del programa quede pausada durante un intervalo de tiempo. Esto permite realizar otras acciones mientras se "confirma" la transición de estado, es decir, mientras se incrementa el contador hasta llegar un cierto valor.

2.3.3 – Configuración del MCU

Se decide utilizar al bit 0 de PORTC como entrada, mediante la siguiente sentencia:

```
DDRC &= ~(1<<PORTC0);
```

Se debe tener en cuenta el estado del pin cuando el pulsador se encuentra 'abierto' y 'cerrado', no debe quedar en un estado indeterminado (al aire) por lo tanto se deben usar resistencias pull-up o pull-down. Aprovechando que el MCU posee una resistencia pull-up interna se hace uso de la misma y define que si el pulsador se encuentra 'abierto', el estado del pin será ALTO (1) caso contrario si el pulsador se encuentra 'cerrado'.

Tabla 2.1. Posibles configuraciones de los terminales de un puerto (p. 60, cap. 13)

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)

Según la Tabla 2.1, para habilitar la resistencia de pull-up interna se debe establecer un valor "1" en el bit correspondiente a la entrada en PORTC, en este caso el bit 0.

```
PORTC |= 1<<PORTC0;
```

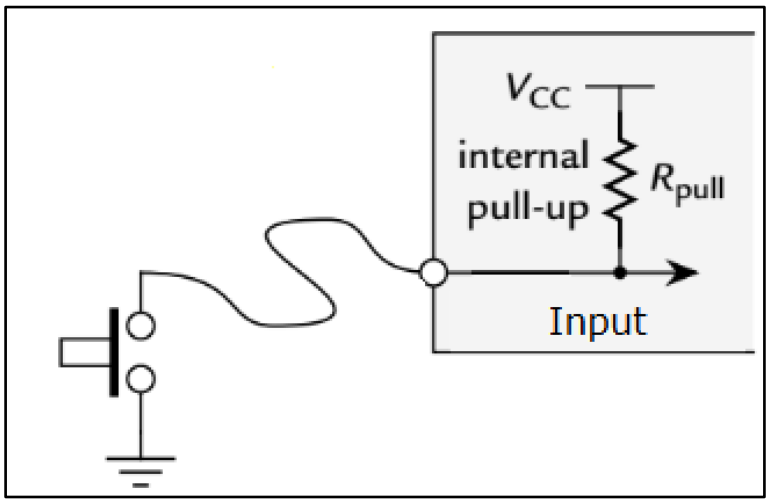

Figura 2.2. Conexión del pulsador usando resistencia de pull-up interna.

2.3.4 – Esquema de conexión

Para realizar la conexión se incorpora el pulsador al proyecto creado en el simulador Proteus, el elemento se puede encontrar como “button” en *Library > Pick Parts.*

Como se decidió utilizar la resistencia pull-up interna se debe conectar un borne del pulsador a tierra y su otro borne al terminal 23 (PC0).

A continuación, se puede observar en la Figura 2.3 el esquema de conexión.

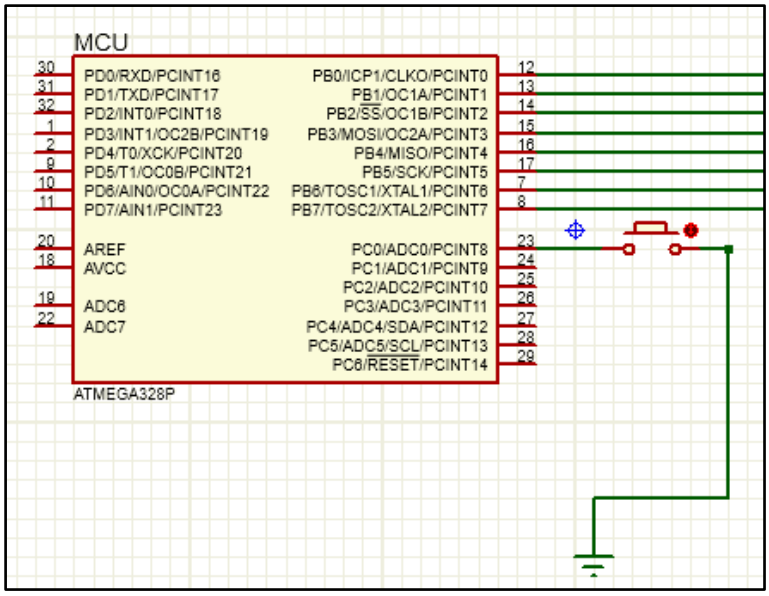

Figura 2.3. Conexión del pulsador al microcontrolador en Proteus.

3 – Programación de la secuencia

3.1 – Enunciado

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 un realizar un programa en C que permita ir encendiendo los diodos LEDs conectados previamente en el ejercicio 1 en un determinado orden. Por defecto, al iniciar la ejecución la secuencia es b0 y b7 – b1 y b6 – b2 y b5 – b3 y b4 (luces acercándose al “centro”),

la cual debe repetirse hasta que el usuario presione y suelte el pulsador. Una vez que el usuario haya soltado el pulsador, se deberá cambiar la dirección de la secuencia, es decir: b3 y b4 – b2 y b5 – b1 y b6 – b0 y b7 (luces alejándose del "centro"). Si el usuario vuelve a presionar y soltar el pulsador, deberá cambiarse a la secuencia, y así repetitivamente.

Cuando se realiza el cambio de secuencia, no se debe producir un "salto" entre luces, en otras palabras, los 2 LEDs que deben encenderse a continuación deben ser contiguos a los 2 previamente encendidos. Por ejemplo: no se puede saltar de b1 y b6 hacia b3 y b4.

Mientras el usuario se encuentra presionando el pulsador, no se afectará la secuencia que se encuentra en ejecución, hasta que el usuario haya soltado el mismo. Esto implica que no se producirá un bloqueo del programa si el usuario mantiene el botón pulsado.

Por otra parte, para una correcta visualización en la simulación, se deberá avanzar en la secuencia de luces cada un cierto intervalo de tiempo, por ejemplo, cada 500 ms o 1 segundo.

3.3 – Resolución

3.3.1 – Implementación de la secuencia

La secuencia de encendido de LEDs se puede entender como 2 bits "1" que se desplazan hacia la derecha o hacia la izquierda (dirección) según la secuencia en ejecución. Uno de los bits "1" se encuentra rotará entre los bits b0, b1, b2 y b3 (en adelante, "op1"), y el otro "1" rotará entre los bits b7, b6, b5 y b4 (en adelante, "op2").

Por medio de una variable posición (en adelante, "pos"), se indicará en cuál de los 4 estados posibles se encuentran "op1" y "op2" simultáneamente. Por ejemplo, si "pos" es igual a 0, eso implica que "op1" se encuentra en b0 y "op2" en el bit b7. El avance de la secuencia en una determinada dirección se consigue incrementando o decrementando la variable "pos" en forma cíclica, considerando que sólo puede asignarse los valores 0, 1, 2 o 3 a la misma.

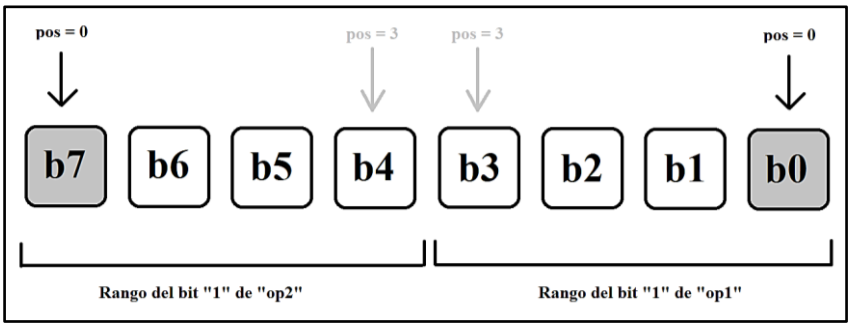

Figura 3.1. Relación entre las variables "op1", "op2" y "pos".

La decisión de incrementar o decrementar “pos” se realiza en base al valor de una variable “dirección”, que toma dos valores posibles: 0 o 1, siendo 0 el valor por defecto (encendido de LEDs desde los extremos “b0 y b7” hacia el centro “b3 y b4”). El cambio de “dirección” se produce únicamente cuando se confirma que el estado del pulsador (ejercicio 2) pasó de “pulsado” a “reposo”, en otras palabras, el usuario lo dejó de presionar. Esto implica que la secuencia continuará en la misma dirección mientras el botón esté presionado.

El pseudocódigo de implementación se muestra a continuación.

```
Inicializar “dirección” en 0
Inicializar “pos” en 0
Inicializar “op1” con su bit menos significativo en alto
Inicializar “op2” con su bit más significativo en alto
Escribir en puerto B la suma binaria de op1 y op2

Cada vez que se cumple un ciclo
    Chequear pulsador (puede modificar dirección)
    Si transcurrió un tiempo suficiente de visualización
        Si la dirección es “1”
            Decrementar “pos”
        Sino si la dirección es “0”
            Incrementar “pos”
        Asignar a “op1” un valor alto “1” solo en el bit “pos”
        Asignar a “op2” un valor alto “1” solo en el bit “7 – pos”
        Escribir en puerto B la suma binaria de op1 y op2
```

Pseudocódigo 3.1. Programa principal para representar la secuencia de encendido.

3.3.2 – Elección del tiempo de visualización

Debido a que continuamente se debe chequear el estado del pulsador para detectar si es necesario cambiar la dirección de la secuencia, no debería existir un bloqueo del programa para realizar una pausa en la visualización de la secuencia. Por tal motivo, se evitó la utilización de la función *_delay_ms()*, en reemplazo de un contador "offset" que se inicializa en un valor determinado y se decrementa en cada ciclo de programa hasta llegar a cero, donde se realiza el avance de la secuencia como se mostró en el pseudocódigo 3.1.

Mediante la herramienta de depuración de Microchip Studio, se puede contar la cantidad de ciclos de reloj entre cada decremento del contador "offset" producido. Para el programa implementado son 35 ciclos, y si la frecuencia del microcontrolador es 16 MHz:

$$T_{decremento} = \frac{35\ ciclos}{16\ MHz} = \frac{35\ ciclos}{16 * 10^6\ ciclos/s} = 2{,}1875\ \mu s$$

Para una visualización no mayor a 500 ms (medio segundo) se requiere que el contador se inicialice en el siguiente valor N (constante OFFSET_MAX en el programa):

$$T_{visualización} < N * T_{decremento} \quad \Leftrightarrow \quad N < \frac{T_{visualización}}{T_{decremento}} \cong 228571$$

Considerando que la frecuencia de reloj por defecto en Proteus es 8 MHz, si se establece un OFFSET_MAX de 200.000 resultará una visualización de 0,875 segundos por cada paso.[2]

3.3.3 – Elección del contador del pulsador

Los 35 ciclos de reloj medidos en la sección anterior también aplican para la repetición del llamado a la función de chequeo implementada en el ejercicio 2. Si consideramos nuevamente que la frecuencia de reloj en Proteus es 8 MHz, para confirmar un estado mediante la lectura de una señal estabilizada por al menos 10 ms se requiere:

$$T_{señal} > M * T_{incremento} \quad \Leftrightarrow \quad N > \frac{T_{señal}}{T_{incremento}} = \frac{10\ ms}{2 * 2{,}1875\ \mu s} \cong 2286$$

Por tal motivo, en el programa de C se decidió establecer que el contador de ciclos del pulsador alcance un valor máximo CHEQUEO = 2500 para cambiar el estado guardado.[3]

[2] Para que la variable "offset" pueda ser inicializada en 200.000, se requiere que se asignen hasta 32 bits.
[3] Para que la variable "contador" de la función de pulsador alcance el valor 2500 se deben asignar hasta 16 bits.

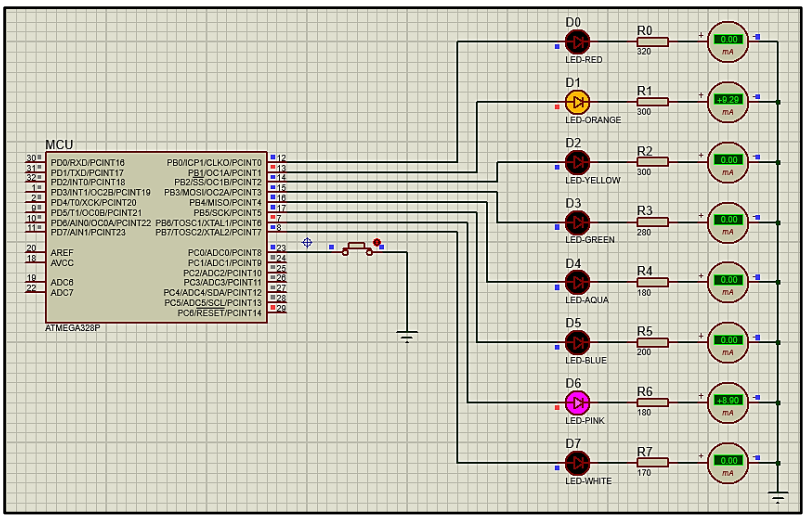

Figura 3.2. Detección de pulsador presionado desde el terminal PC0.

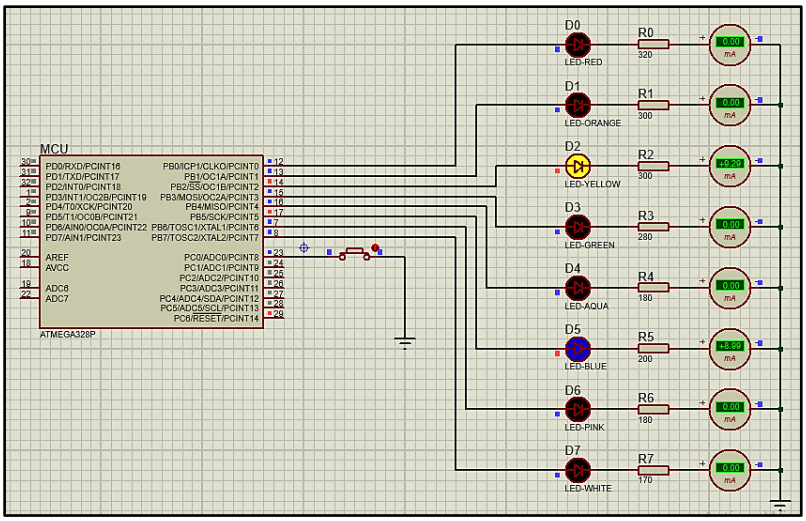

Figura 3.3. Continuación normal de la secuencia mientras se presiona el pulsador.

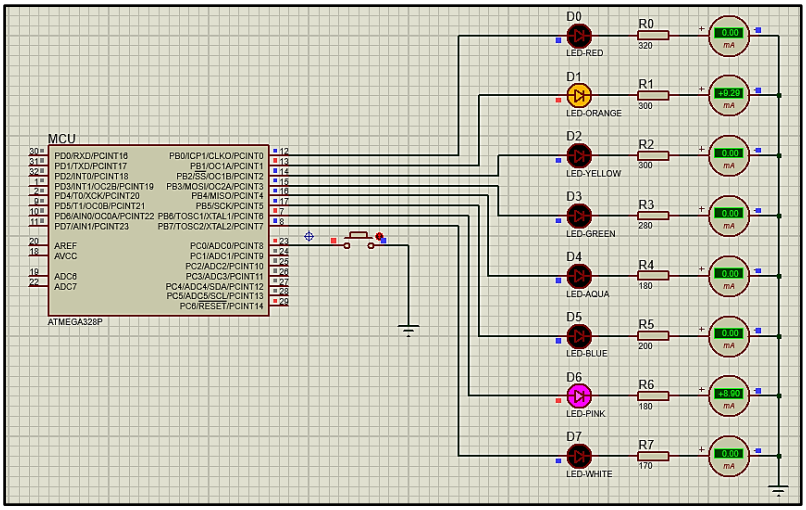

Figura 3.4. Cambio de dirección de la secuencia luego de soltar el pulsador.

4 – Conclusiones

4.1 – Enunciado

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 – Desarrollo

El funcionamiento del programa cumple con lo pedido en los enunciados. La implementación de un algoritmo de detección del pulsador con contador evita que la ejecución del programa quede pausada durante un intervalo de tiempo. Como se ha observado, si el usuario mantiene pulsado el pulsador, se logra que la secuencia de encendido de los LEDs siga funcionando normalmente, y cuando el pulsador se suelte se cambie a la secuencia opuesta.

Si se hubiese utilizado un algoritmo con retardo bloqueante se tendría la ventaja de que es una solución rápida debido a que la función de delay ya está creada y se adecúa a la frecuencia de reloj especificada. Como desventaja se encuentra el problema de que el MCU se encuentra 'atrapado' dentro de la función hasta que se cumpla el tiempo de delay. Esto último ocasiona que, si se produjese algún cambio repentino en un terminal, por ejemplo, se presiona y suelta rápidamente el pulsador, dicho evento pase desapercibido por el microcontrolador.

Otra desventaja del delay es que se establece un valor concreto de demora que según el tipo de pulsador puede resultar insuficiente o demasiado grande. En el primer caso, el efecto de rebote puede provocar que se cambie la secuencia un número par de veces cuando el usuario presiona el pulsador una única vez, teniendo como resultado que la dirección no cambie.

Respondiendo a la pregunta del enunciado, si además del delay se utilizara un *while* hasta que el usuario suelte el pulsador, también se impediría que el programa pueda realizar otras acciones, como avanzar en la secuencia de LEDs, mientras dicha condición sea falsa.

5 – Bibliografía

- LEDnique (s.f.). *LEDs and colour*. http://lednique.com/leds-and-colour.
- Atmel Corporation (2015). *ATmega 328P Datasheet*. Capítulos 13, 28 y 29. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Apéndice A – Código fuente

A continuación, se muestra el programa completo escrito en C que se compila y utiliza para el funcionamiento del microcontrolador en el simulador Proteus.

main.c

```
/* Constantes */
#define REPOSO 1
#define CHEQUEO 2500
#define OFFSET_MAX 200000

/* Cabeceras */
#include <avr/io.h>

/* Variables globales */
uint8_t direccion = 0;

/* Prototipos */
void pulsador(void);

int main(void)
{
	/* Setup */
	DDRB = 0xFF;				// Todo PORTB es salida
	DDRC &= ~(1<<PORTC0);			// Bit 0 de PORTC es entrada
	PORTC |= 1<<PORTC0;			// Seteo de resistencia pull-up interna

	uint8_t pos = 0;
	uint8_t op1 = 1<<PORTB0;		// Bit menos significativo en alto
	uint8_t op2 = 1<<PORTB7;		// Bit más significativo en alto
	uint32_t offset = OFFSET_MAX;		// Retardo de visualización

	PORTB = op2 | op1;			// Se activan b0 y b7 por defecto

	/* Loop */
	while (1)
	{
		pulsador();			// Chequear pulsador

		if (offset == 0)		// Si transcurrió el tiempo de visualización
		{
			offset = OFFSET_MAX;

			if (direccion){
				if (pos == 0) pos = 3;
				else pos--;	// Alejar del centro
			} else {
				if (pos == 3) pos = 0;
				else pos++;	// Acercar al centro
			}

			op1 = 1<<pos;		// Bit 0, 1, 2 o 3 en alto
			op2 = 1<<(7-pos);	// Bit 7, 6, 5 o 4 en alto

			PORTB = op2 | op1;	// Se muestra el paso actual de la secuencia
		} else {
			offset--;
		}
	}
}
```

```
/* Función de chequeo del estado del pulsador, con anti-rebote */
void pulsador(void)
{
	// Seguimiento del estado (anti-rebote)
	static uint8_t estado_confirmado = REPOSO;
	static uint16_t contador = 0;

	// Si se encuentra en reposo (en este momento), la entrada tiene valor 1, sino 0
	uint8_t estado_actual = PINC & (1<<PINC0);

	if (estado_actual != estado_confirmado)
	{
		// Se incrementa contador hasta confirmar cambio de estado
		contador++;
		if (contador == CHEQUEO)
		{
			estado_confirmado = estado_actual;
			// Cambia la dirección de la secuencia sólo al soltar el pulsador
			// Mientras esté presionado la secuencia seguirá ejecutándose
			if (estado_confirmado == REPOSO) direccion = direccion ? 0 : 1;
			contador = 0;
		}
	} else {
		contador = 0;
	}
}
```