INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PRÁCTICA 5 Y 6: CONTADOR DE 0 A 9 ACTIVADO POR FLANCOS Y SIN REBOTES

ALUMNO: MALAGON BAEZA ALAN ADRIAN
MARTINEZ CHAVEZ JORGE ALEXIS

GRUPO: 6CM3

U.A: SISTEMAS EN CHIP

PROFESOR: FERNANDO AGUILAR SÁNCHEZ

FECHA DE ENTREGA: 30 DE ABRIL DE 2023

OBJETIVO

Al término de la sesión, los integrantes del equipo contaran con la habilidad de realizar un contador activado por flancos y sin rebotes de 0 a 9 mostrado en un display activado con un Push Button.

INTRODUCCIÓN TEÓRICA

FLANCOS

En electrónica digital se representa, traslada, encripta, calcula y almacena la información de forma binaria. El concepto es sencillo ya que solo tenemos dos estados 0 y 1, pero en realidad estos estados hay que pasarlos a niveles eléctricos. En la realidad trabajamos con diferencias de potencial.[1]

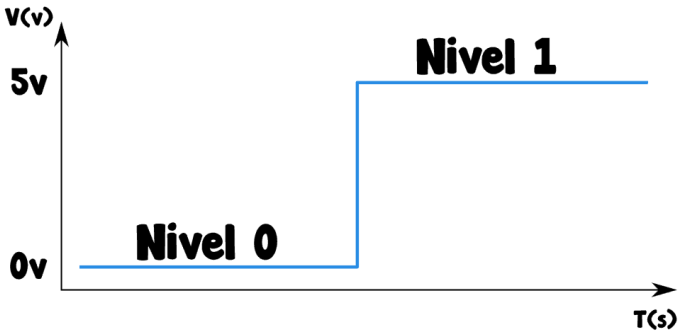


Figura 1. Niveles lógicos

Una forma clásica de definir el 0 y el 1 es con dos niveles de tensión estándar como se aprecia en la Figura 1. En este sistema el 1 se representa con 5v y el 0 con 0 voltios. Aun siendo una forma clásica de entender la transmisión de datos es una de las más usadas. En realidad, no tienen por qué ser 5v, puede ser otro valor de tensión, por ejemplo 3.3v o 1.8v.[1]

Lo más importante de este método no es el valor de tensión que representa el 1 lógico, sino que lo más importante es que el 1 se representa con un valor de tensión positivo y el 0 con la masa del circuito.[1]

Se puede decir que el diagrama anterior representa dos estados, el primero es el estado 0 que se representa con la línea horizontal baja y el segundo el estado 1 que está representado con la línea horizontal alta.[1]

La línea vertical en la Figura 1 es el FLANCO. Un flanco es la transición entre dos estados lógicos, en este caso concreto pasamos de estar en un estado bajo a un estado alto. En inglés al flanco se le denomina EDGE, cuya traducción literal podría ser borde o canto y es que es justamente esto, el borde entre dos estados, el borde entre pasar de un estado bajo a un estado alto.[1]

Un flanco de subida es el que pasa de estar en nivel bajo a estar en nivel alto. Los flancos de subida se denominan en inglés rising. Por otro lado, el flanco de bajada es aquel en el que pasa de estar en nivel alto a estar en nivel bajo. Los flancos de bajada se denominan en inglés falling.[1]

ANTI-REBOTE DEL PULSADOR

Los pulsadores tienen un efecto rebote cuando se pulsan. Es decir, cuando se presiona o suelta se produce una fluctuación en la señal que pasa por sus contactos y podría hacer que se pase de un estado alto a bajo o viceversa sin que realmente queramos que ocurra eso. Eso puede producir un efecto indeseado en el circuito dónde tengamos implementado el pulsador y que haga cosas extrañas, como activar un elemento cuando realmente lo que queríamos es apagarlo con el pulsador, etc. Eso se debe a que el microcontrolador interpreta los rebotes como si se hubiera pulsado más de una vez.[2]

Ese efecto negativo tiene solución. Para eso hay que implementar un pequeño condensador en el circuito antirrebote (método por hardware) o por software (modificando el código fuente), tanto si se ha usado una configuración pull-up como pull-down o si es NC o NA. En todos esos casos hay que implementar la solución para evitar estos rebotes.[2]

Por ejemplo, los circuitos pull-up y pull-down con el condensador antirrebote quedarían algo así:

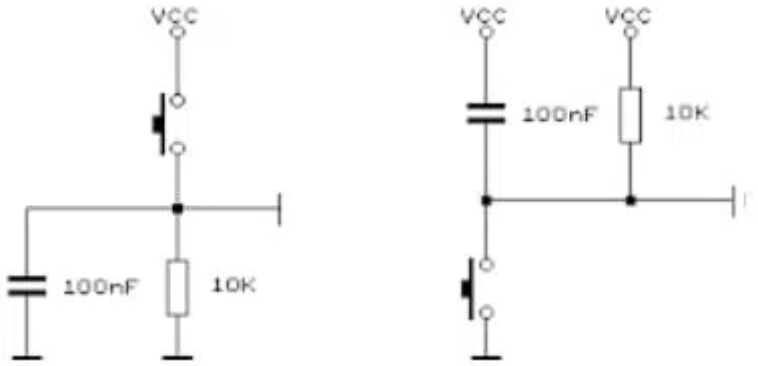


Figura 2. Solución de hardware para Anti-Rebote

Mientras que el método mediante software podría verse en este fragmento de código:

```
if (digitalRead(pulsador) == LOW) //Comprueba si el pulsador está
pulsado
{
     presionado = 1; //La variable cambia de valor
}
if (digitalRead(pulsador) == HIGH && presionado == 1)
{
     //Realiza la acción deseada
     presionado = 0; //La variable vuelve a su valor original
}
```

MATERIALES Y EQUIPO EMPLEADO

- ✓ CodeVision AVR
- ✓ AVR Studio 4
- ✓ Microcontrolador ATmega 8535
- ✓ 1 Display cátodo común

- ✓ 7 Resistores de 330 Ω a 1⁄4 W
- ✓ 1 Push Button

DESARROLLO EXPERIMENTAL

1. Diseñe un programa colocando en el Puerto B un Display. Coloque un Push Button en la terminal 0 del Puerto D para incrementar su cuenta del 0 al 9 activado por flancos y sin rebotes.

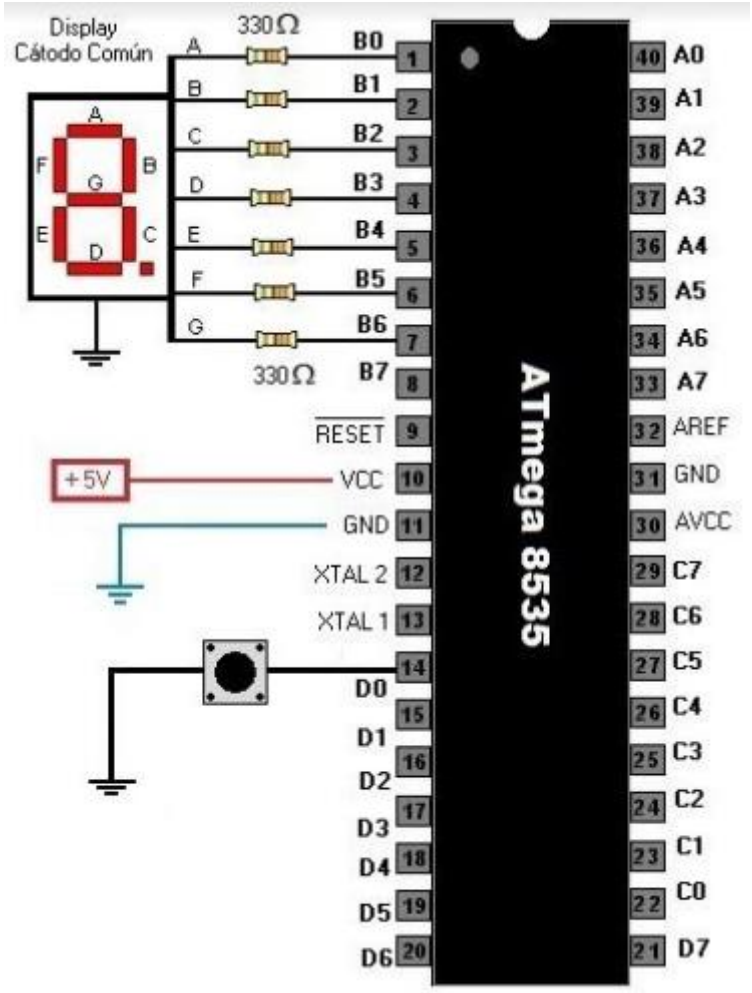


Figura 3. Circuito para el contador de 0 a 9 activado por flancos y sin rebotes

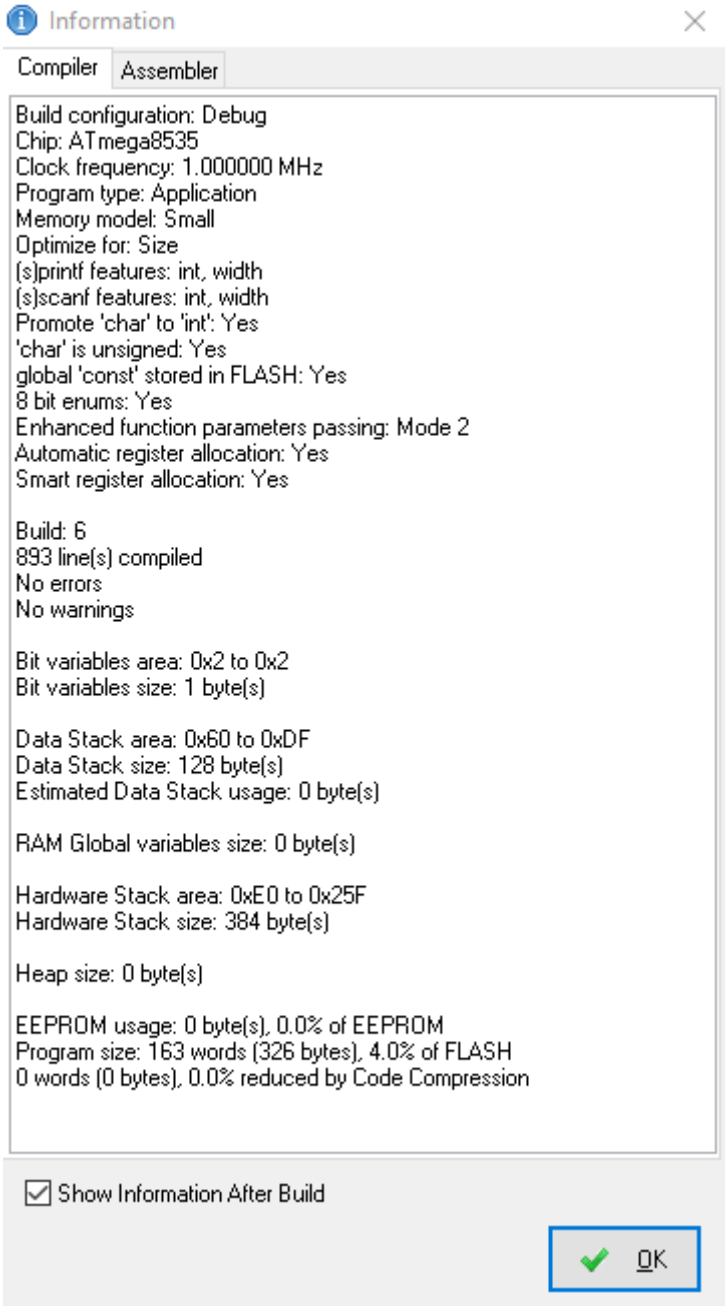


Figura 4. Compilación exitosa en CodeVision

CÓDIGO GENERADO POR CODEVISION

```
/*****************************************************
This program was created by the CodeWizardAVR V3.47
Automatic Program Generator
© Copyright 1998-2021 Pavel Haiduc, HP InfoTech S.R.L.
http://www.hpinfotech.ro

Project :
Version :
Date    :
Author  :
```

```
Company :
Comments:


Chip type               : ATmega8535
Program type            : Application
AVR Core Clock frequency: 1.000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 128
*****************************************************/

#include <mega8535.h>
#include <delay.h>
#define boton PIND.0   //Definición de un puerto mediante una
etiqueta

bit botonp;     //Botón previo
bit botona;     //Botón actual
const char tabla7segmentos[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66,
0x6d, 0x7d, 0x07, 0x7f, 0x6f};
unsigned char var;

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) |
(0<<DDA2) | (0<<DDA1) | (0<<DDA0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) |
(0<<PORTA3) | (0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);

// Port B initialization
// Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out
Bit1=Out Bit0=Out
DDRB=(1<<DDB7) | (1<<DDB6) | (1<<DDB5) | (1<<DDB4) | (1<<DDB3) |
(1<<DDB2) | (1<<DDB1) | (1<<DDB0);
// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) |
(0<<PORTB3) | (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

// Port C initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
```

```
DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) |
(0<<DDC2) | (0<<DDC1) | (0<<DDC0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) |
(0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

// Port D initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) |
(0<<DDD2) | (0<<DDD1) | (0<<DDD0);
// State: Bit7=P Bit6=P Bit5=P Bit4=P Bit3=P Bit2=P Bit1=P Bit0=P
PORTD=(1<<PORTD7) | (1<<PORTD6) | (1<<PORTD5) | (1<<PORTD4) |
(1<<PORTD3) | (1<<PORTD2) | (1<<PORTD1) | (1<<PORTD0);

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=0xFF
// OC0 output: Disconnected
TCCR0=(0<<WGM00) | (0<<COM01) | (0<<COM00) | (0<<WGM01) | (0<<CS02)
| (0<<CS01) | (0<<CS00);
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer1 Stopped
// Mode: Normal top=0xFFFF
// OC1A output: Disconnected
// OC1B output: Disconnected
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=(0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) | (0<<COM1B0) |
(0<<WGM11) | (0<<WGM10);
TCCR1B=(0<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (0<<WGM12) | (0<<CS12)
| (0<<CS11) | (0<<CS10);
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
```

```
// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=0xFF
// OC2 output: Disconnected
ASSR=0<<AS2;
TCCR2=(0<<WGM20) | (0<<COM21) | (0<<COM20) | (0<<WGM21) | (0<<CS22)
| (0<<CS21) | (0<<CS20);
TCNT2=0x00;
OCR2=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=(0<<OCIE2) | (0<<TOIE2) | (0<<TICIE1) | (0<<OCIE1A) |
(0<<OCIE1B) | (0<<TOIE1) | (0<<OCIE0) | (0<<TOIE0);

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=(0<<ISC11) | (0<<ISC10) | (0<<ISC01) | (0<<ISC00);
MCUCSR=(0<<ISC2);

// USART initialization
// USART disabled
UCSRB=(0<<RXCIE) | (0<<TXCIE) | (0<<UDRIE) | (0<<RXEN) | (0<<TXEN) |
(0<<UCSZ2) | (0<<RXB8) | (0<<TXB8);

// Analog Comparator initialization
// Analog Comparator: Off
// The Analog Comparator's positive input is
// connected to the AIN0 pin
// The Analog Comparator's negative input is
// connected to the AIN1 pin
ACSR=(1<<ACD) | (0<<ACBG) | (0<<ACO) | (0<<ACI) | (0<<ACIE) |
(0<<ACIC) | (0<<ACIS1) | (0<<ACIS0);
SFIOR=(0<<ACME);

// ADC initialization
// ADC disabled
ADCSRA=(0<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) |
(0<<ADPS2) | (0<<ADPS1) | (0<<ADPS0);

// SPI initialization
// SPI disabled
SPCR=(0<<SPIE) | (0<<SPE) | (0<<DORD) | (0<<MSTR) | (0<<CPOL) |
(0<<CPHA) | (0<<SPR1) | (0<<SPR0);

// TWI initialization
```

```
// TWI disabled
TWCR=(0<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | (0<<TWEN) | (0<<TWIE);

while (1)
      {
      //Inicializar valores del botón actual
      if(boton == 0){
        botona = 0;
      }
      else{
        botona = 1;
      }

      //Activación por flancos y eliminación de rebotes
      if((botonp == 1) && (botona == 0)){   //Hubo cambio de flanco
de 1 a 0
        var++;  //Se incrementa la variable
        if(var == 10){
            var = 0;
        }
        delay_ms(40);       //Se coloca retardo de 40ms para
eliminar rebotes
      }
      if((botonp == 0) && (botona == 1)){   //Hubo cambio de flanco
de 0 a 1
        delay_ms(40);       //Se coloca retardo de 40ms para
eliminar rebotes
      }

      PORTB = tabla7segmentos[var];
      botonp = botona;
      }
}
```

CIRCUITO ELECTRICO EN PROTEUS

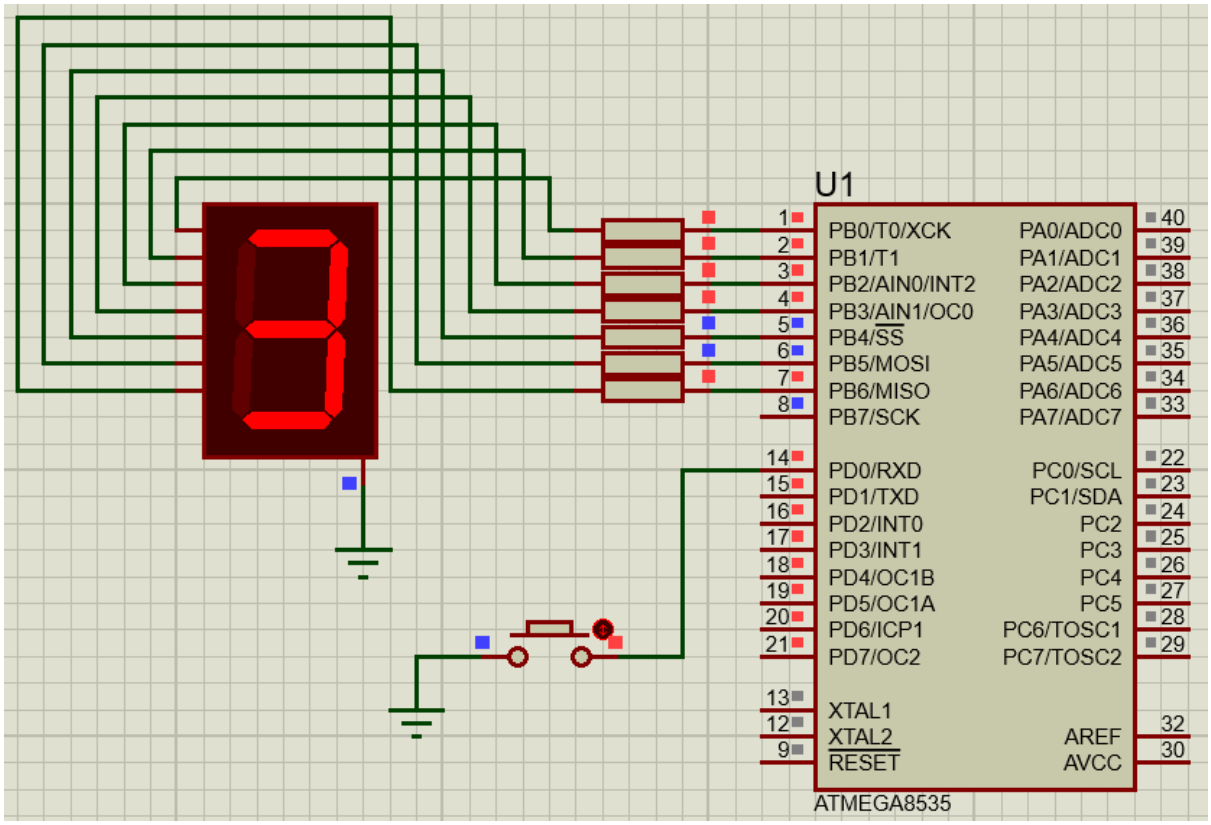
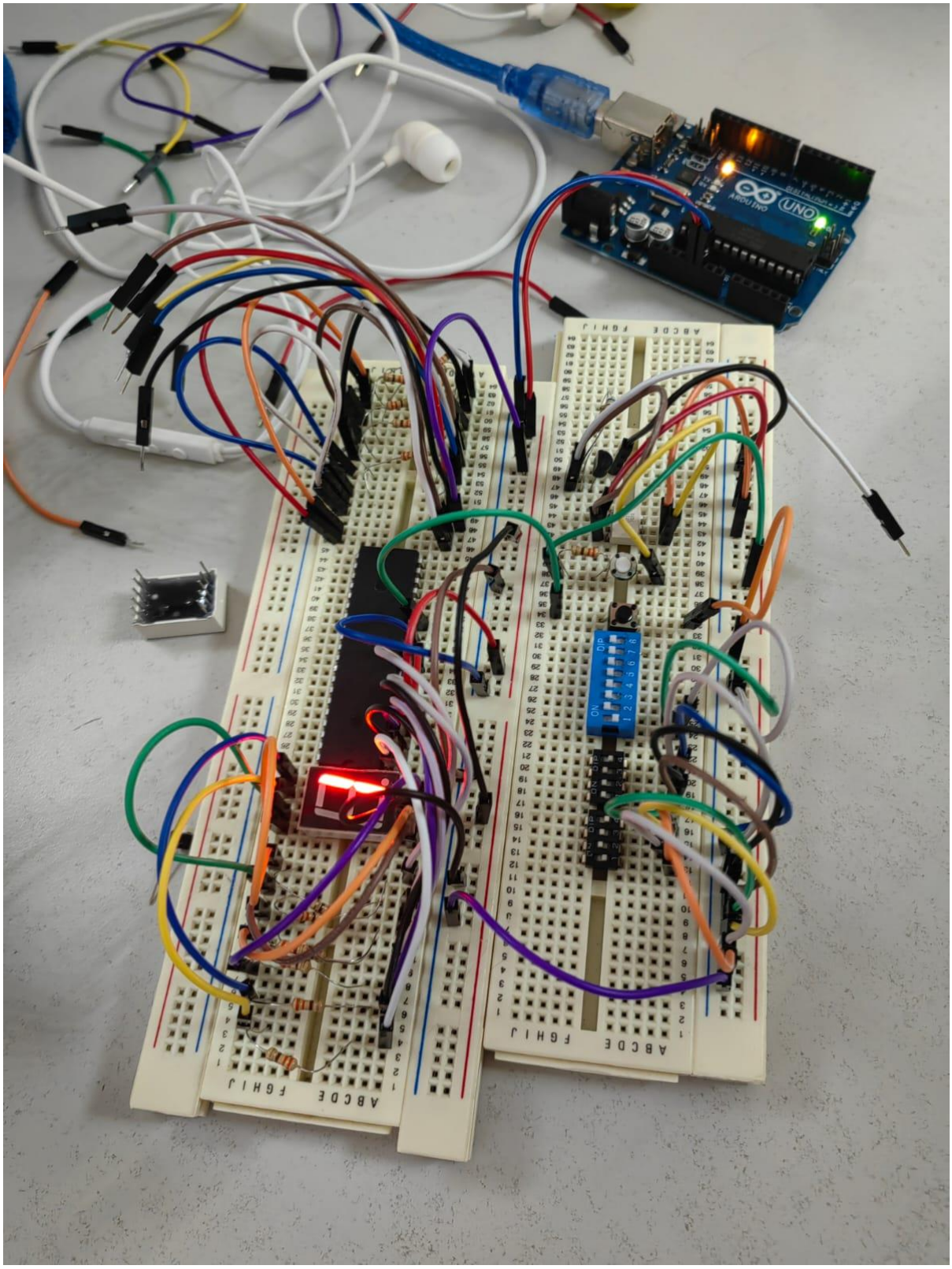


Figura 5. Simulación del circuito en Proteus

CIRCUITO EN EL PROTO ARMADO



OBSERVACIONES Y CONCLUSIONES INDIVIDUALES

Malagón Baeza Alan Adrián

El pulsador (push botón) es un dispositivo muy útil y con múltiples aplicaciones en los circuitos electrónicos y digitales. En un sistema digital el pulsador puede indicar un cambio de estado (0 o 1) que a su vez puede ayudar a describir un comportamiento específico del circuito cuando se detecte ese cambio. Hay que recordar que en un sistema digital se tienen únicamente dos valores lógicos, los cuales representan la presencia o ausencia de energía eléctrica (normalmente 5V), por lo que exclusivamente tendremos dos valores con el uso de los pulsadores: 0 para ausencia, y 1 para presencia.

Cuando implementamos los pulsadores en nuestros circuitos electrónicos se pueden presentar algunos problemas para poder interpretar la lectura de la pulsación del botón. Uno de los problemas que se puede presentar es que el botón no accione correctamente el circuito en un flanco de subida o bajada dependiendo la lógica que se desea emplear.

Martínez Chávez Jorge Alexis

Para poder asegurar que el circuito se comporte de manera adecuada y en sincronía con los flancos de subida y bajada que nos otorga el pulsador, se puede implementar un código para que el circuito se accione cuando se detecte el flanco deseado, y después de ejecutar la acción se puede regresar al flanco anterior para esperar la orden de un nuevo flanco.

Otro de los problemas que se pueden presentar en el uso de los pulsadores es la propiedad del rebote que presentan estos dispositivos. El rebote puede causar que se detecte más de una pulsación y el circuito se accione más veces de lo deseado. Estas vibraciones ocasionan ruido en el circuito y pueden interferir en los resultados esperados. Para poder solucionar este problema se puede utilizar una subrutina de retardo de tiempo después del flanco detectado para que no se registren los rebotes posteriores que puede generar el pulsador. El tiempo del retardo puede ser de 20ms, sin embargo es recomendable usar 40ms para asegurar el anti-rebote.

BIBLIOGRAFÍA

[1] E. Gómez, “Flanco de subida y bajada ¿Qué son?,” *Rincón Ingenieril*, Jan. 31, 2017. https://www.rinconingenieril.es/flanco-subida-bajada/ (accessed Feb. 18, 2022).
[2] Isaac, “Pulsador: cómo usar este simple elemento con Arduino,” *Hardware libre*, Oct. 29, 2019. https://www.hwlibre.com/pulsador/#Anti-Rebote (accessed Feb. 18, 2022).