INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO


# PRÁCTICA 10: TECLADO MATRICIAL

ALUMNOS: MALAGON BAEZA ALAN ADRIAN
MARTINEZ CHAVEZ JORGE ALEXIS

GRUPO: 6CM3

U.A: SISTEMAS EN CHIP

PROFESOR: FERNANDO AGUILAR SÁNCHEZ

FECHA DE ENTREGA: 28 DE MAYO DE 2023

# OBJETIVO

Al término de la sesión, los integrantes del equipo contaran con la habilidad de realizar un teclado matricial.

# INTRODUCCIÓN TEÓRICA

## TECLADO MATRICIAL

Un teclado matricial es un dispositivo que agrupa varios pulsadores y permite controlarlos empleando un número de conductores inferior al que necesitaríamos al usarlos de forma individual. Podemos emplear estos teclados como un controlador para un autómata o un procesador.[1]

Estos dispositivos agrupan los pulsadores en filas y columnas formando una matriz, disposición que da lugar a su nombre. Es frecuente una disposición rectangular pura de NxM columnas, aunque otras disposiciones son igualmente posibles.[1]

Los teclados matriciales son frecuentes en electrónica e informática. De hecho, los teclados de ordenador normales son teclados matriciales, siendo un buen ejemplo de teclado matricial con disposición no rectangular.[1]

Una de las desventajas de usar un teclado matricial es que pueden causar problemas cuando se pulsa más de una tecla simultáneamente. Este es uno de los motivos por el que los teclados de ordenador usan una disposición no rectangular, agrupando ciertas teclas en circuitos diferentes.[1]

En el campo de la electrónica casera, se venden múltiples modelos de teclados matriciales en distintos soportes (rígidos o flexibles) y con distinto número de teclas, siendo habituales configuraciones de 3x3, 3x4, y 4x4.[1]

Podemos emplear teclados matriciales en nuestros proyectos de electrónica y robótica, por ejemplo, para cambiar el modo de funcionamiento de un montaje, para solicitar una contraseña, como teclas de dirección para controlar un brazo robótico o un vehículo, o proporcionar instrucciones a un robot.[1]

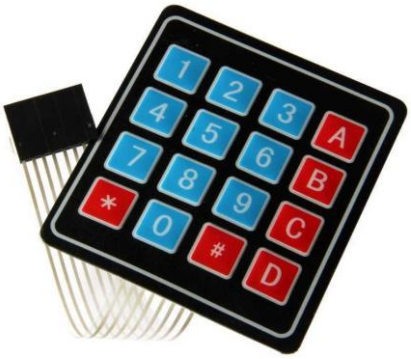



*Figura 1. Teclado matricial comercial*

**FUNCIONAMIENTO DEL TECLADO MATRICIAL**

Un teclado matricial agrupa los pulsadores en filas y columnas formando una matriz, lo que permite emplear un número menor de conductores para determinar las pulsaciones de las teclas.[1]

La siguiente imagen muestra, a modo de ejemplo, una disposición rectangular de 4x4, aunque el funcionamiento es análogo en otras disposiciones. Al detectar la pulsación en la columna X y la fila Y, sabremos que se ha pulsado la tecla (X,Y).[1]

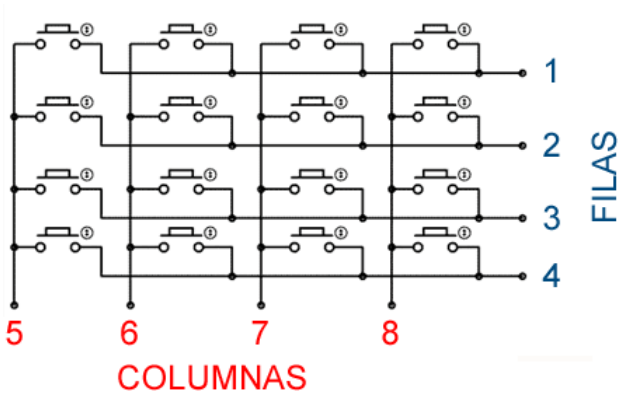



*Figura 2. Disposición de teclado matricial 4x4*

Para detectar la pulsación de una tecla actuaremos de forma similar a la lectura simple de un pulsador. En resumen, ponemos a tierra un extremo del pulsador, y el otro lo conectamos a una entrada digital con una resistencia de pull-up.[1]

Para leer todas las teclas tendremos que hacer un barrido por filas. En primer lugar, ponemos todas las filas a 5V, y definimos todas las columnas como entradas con resistencia de pull-up.[1]

Progresivamente ponemos una fila a 0V, y leemos las entradas de la columna. Una vez realizada la lectura volvemos a ponerla a 5V, pasamos a la siguiente fila, y volvemos a realizar el progreso hasta recorrer todas las filas.[1]

# 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
- ✓ 3 Resistores de 100 Ω a 1⁄4 W
- ✓ 9 Push Button

# DESARROLLO EXPERIMENTAL

1. Diseño de un teclado matricial de 3*3, con despliegue a display a 7 segmentos. Los pines C0, C1 y C2 serán los pines de salida por donde se enviarán los códigos de “saceneo”, los pines C3, C4 y C5 serán los pines de entrada donde se leerán los botones presionados.

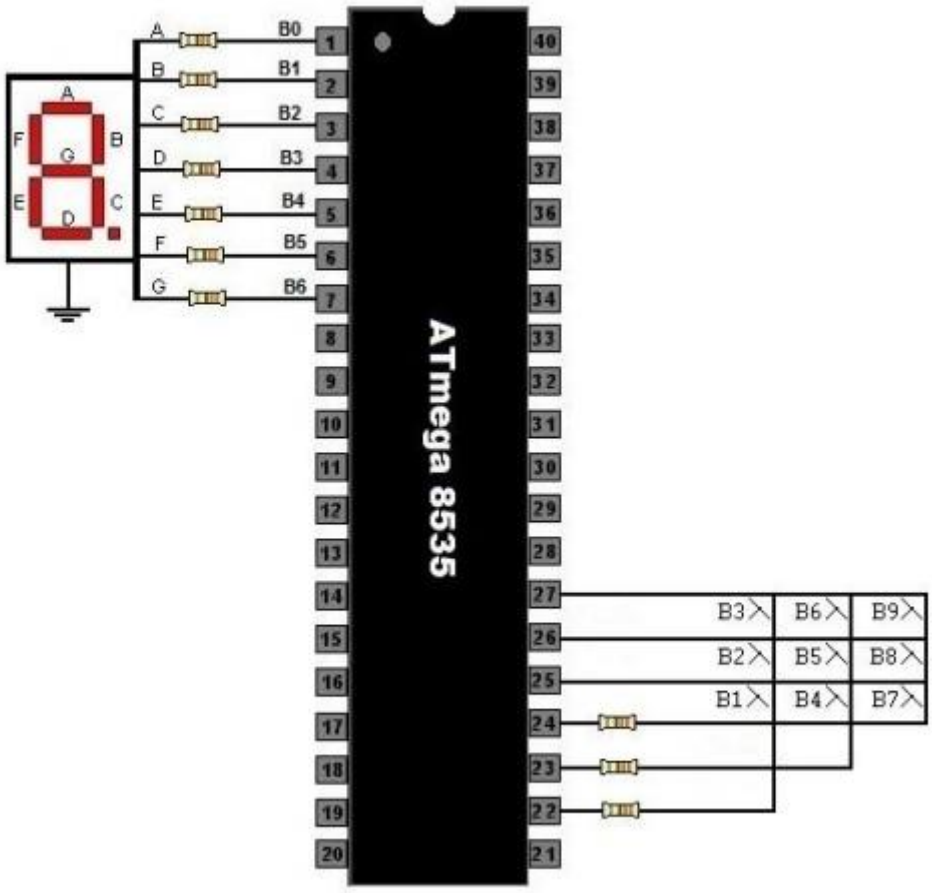



*Figura 3. Circuito para el contador teclado matricial*

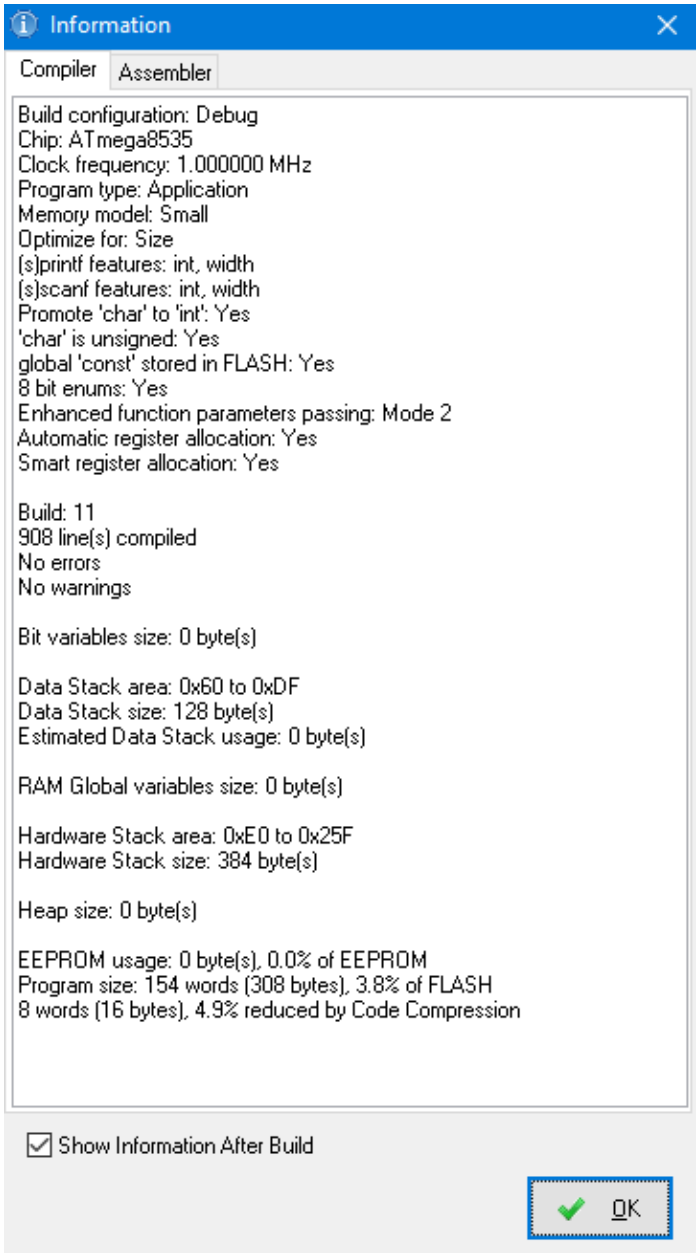



*Figura 4. Compilación exitosa en CodeVision*

# CÓDIGO GENERADO POR CODEVISION

```
/*****************************************************
This program was created by the CodeWizardAVR V3.48a
Automatic Program Generator
© Copyright 1998-2022 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
*****************************************************/

// I/O Registers definitions
#include <mega8535.h>
unsigned char tecla, lectura;
const char tabla7segmentos[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66,
0x6d, 0x7d, 0x07, 0x7f, 0x6f};

// Declare your global variables here

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=Out
Bit1=Out Bit0=Out
DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) |
(1<<DDC2) | (1<<DDC1) | (1<<DDC0);
// State: Bit7=T Bit6=T Bit5=P Bit4=P Bit3=P Bit2=0 Bit1=0 Bit0=0
PORTC=(0<<PORTC7) | (0<<PORTC6) | (1<<PORTC5) | (1<<PORTC4) |
(1<<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=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) |
(0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<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)
      {
```

```
      tecla = 0;

      //C0 A C2 son de salida y son para probar las 3 columnas
      //Se prueba la primera columna se envía 110
      PORTC=0b00111110;

      //C3, C4 y C5 son las entradas del teclado
      //Por eso se enmascaran con 00111000
      lectura=PINC&0b00111000;

      if (lectura==0b00110000){
        tecla=7;
      }
      if (lectura==0b00101000){
        tecla=8;
      }
      if (lectura==0b00011000){
        tecla=9;
      }

      //Se prueba segunda columna se envía 101
      PORTC=0b00111101;

      //C3, C4 y C5 son las entradas del teclado
      //Por eso se enmascaran con 00111000
      lectura=PINC&0b00111000;

      if (lectura==0b00110000){
        tecla=4;
      }
      if (lectura==0b00101000){
        tecla=5;
      }
      if (lectura==0b00011000){
        tecla=6;
      }

      //Se prueba tercera columna se envía 011
      PORTC=0b00111011;

      //C3, C4 y C5 son las entradas del teclado
      //Por eso se enmascaran con 00111000
      lectura=PINC&0b00111000;

      if (lectura==0b00110000){
        tecla=1;
      }
      if (lectura==0b00101000){
        tecla=2;
```

```
        }
        if (lectura==0b00011000){
          tecla=3;
        }

        PORTB=tabla7segmentos [tecla];
        }
}
```

# CIRCUITO ELECTRICO EN PROTEUS

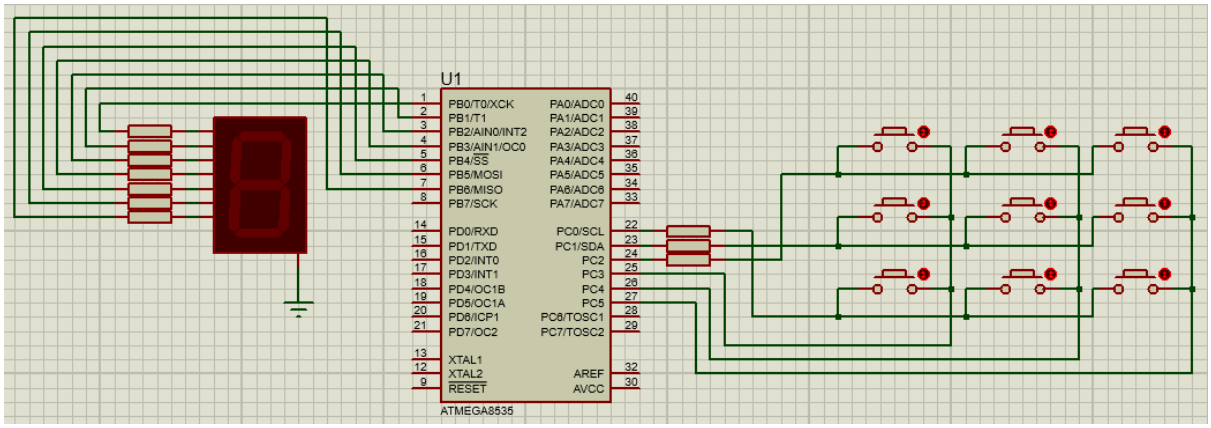
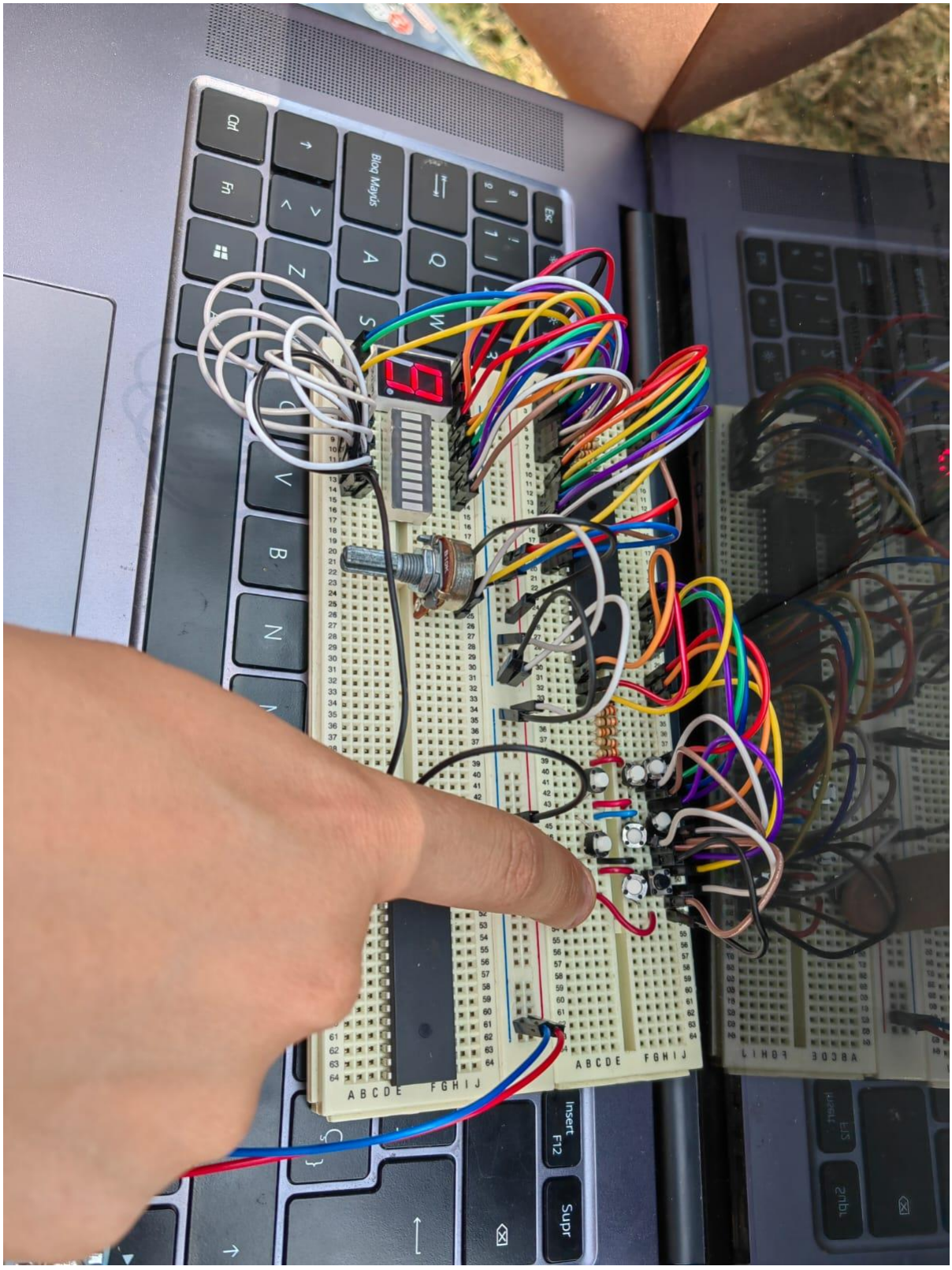



*Figura 5. Circuito simulado en Proteus*

# CIRCUITO EN EL PROTO ARMADO



# OBSERVACIONES Y CONCLUSIONES INDIVIDUALES

Malagon Baeza Alan Adrian

Los teclados matriciales tienen múltiples aplicaciones en el campo de la electrónica. En nuestros circuitos pueden realizar tareas como la lectura de múltiples datos en un número menor de entradas, lo que optimiza el número de puertos a utilizar del microcontrolador y lo que nos brinda la posibilidad de utilizar un puerto para un mayor número de entradas. Una aplicación común de los teclados matriciales en los circuitos es la de ingreso de una contraseña, por lo que con este dispositivo podremos desarrollar circuitos con ingreso de contraseñas de seguridad.

La programación del teclado matricial es muy sencilla de entender, puesto que se analiza la información por columnas y posteriormente por fila. Mediante pruebas por columnas verificamos si es que algún botón de esa columna está presionado, y en caso de estarlo posteriormente se determina, mediante sentencias if, la fila en la que se presionó el botón.

Martínez Chávez Jorge Alexis

En el mercado encontramos teclados matriciales de distintos tamaños para poder conectarlos en nuestros circuitos, incluso el simulador Proteus también cuenta con un teclado matricial para poder simular nuestros circuitos con un dispositivo que existe para circuitos físicos. Sin embargo, fue importante el entender e implementar el teclado matricial mediante el uso de pulsadores para entender la lógica que se sigue para leer los datos y que podamos desarrollar un teclado matricial del tamaño que deseemos dependiendo los requisitos de futuros proyectos.

# BIBLIOGRAFÍA

[1] L. Llamas, “Usar un teclado matricial con Arduino,” *Luis Llamas*, Oct. 02, 2016. https://www.luisllamas.es/arduino-teclado-matricial/ (accessed Mar. 21, 2022).