INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PRÁCTICA 11: MEMORIA EEPROM

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 hacer uso de la memoria EEPROM del microcontrolador.

INTRODUCCIÓN TEÓRICA

MEMORIA EEPROM

EEPROM responde a “Erasable Programmable Read Only Memory” que se puede traducir como Memoria programable borrable de solo lectura. También se la conoce como E-2-PROM. Como su nombre sugiere, una EEPROM puede ser borrada y programada con impulsos eléctricos. Al ser una pieza que se puede gestionar por estos impulsos eléctricos, podemos realizar todas estas operaciones de reprogramación sin tener que desconectarla de la placa a la cual va conectada.[1]

La EEPROM también se conoce como “non-volatile memory” o memoria no volátil y es debido a que cuando se desconecta la energía, los datos almacenados en la EEPROM no serán eliminados quedando intactos.[1]

Las EEPROM más nuevas no tiene datos almacenados en ellas y deben ser primero configuradas con un programador antes de ser usadas. La información almacenada dentro de este dispositivo puede permanecer durante años sin una fuente de energía eléctrica.[1]

Figura 1. Memoria EEPROM comercial

FUNCIÓN DE UNA EEPROM

Son usadas para almacenar información programable de usuario, como, por ejemplo:

- Información de programación VCR
- Información de programación de CD
- Información de usuario de productos instalados en el equipo

La EEPROM en el monitor realiza dos funciones:

- ✓ Cuando encendemos un monitor se copiarán todos los datos o información desde la EEPROM al microprocesador. Por ejemplo, la EEPROM dejará al microprocesador conocer las frecuencias en las cuales el monitor funcionará.[1]

- ✓ La EEPROM se utiliza para guardar la configuración más reciente del monitor. La configuración del monitor no desaparecerá, aunque el monitor sea apagado. Cuando se haga un cambio en dicha configuración, el microprocesador actualiza estos cambios en la EEPROM. Cuando el monitor vuelve a encenderse, los datos ya actualizados son usados para poner el monitor operativo.[1]

Las EEPROM raramente fallan, y cuando lo hacen suele ser por picos eléctricos y sobrecargas de energía, provocando perdida de datos o que estos datos queden dañados. Las EEPROM más modernas viene vacías y necesitan que la información les sea cargada para funcionar. El trabajo de copiar los datos en una EEPROM se hace mediante un programador o copiador. Estos programas vienes en todos los tamaños y formas. Se componen de una parte hardware donde se conecta la EEPROM y luego existen muchos softwares que harán la descarga al dispositivo.[1]

Hay que tener en cuenta que las EEPROM tiene un tiempo limitado de vida, es decir, las veces que se pueden reprogramar puede ser de cientos o miles de veces, pero no son infinitas.[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
- ✓ 2 Push Button

DESARROLLO EXPERIMENTAL

1. Haga un programa en el cual con un botón conectado al pin C0 incrementará el valor en el display conectado en el puerto B y cuando se presione el botón C1 lo guarde en la memoria EEPROM. Después desconecte el microcontrolador de la energía eléctrica y vuélvalo a conectar para que observe que el dato se quedó guardado.

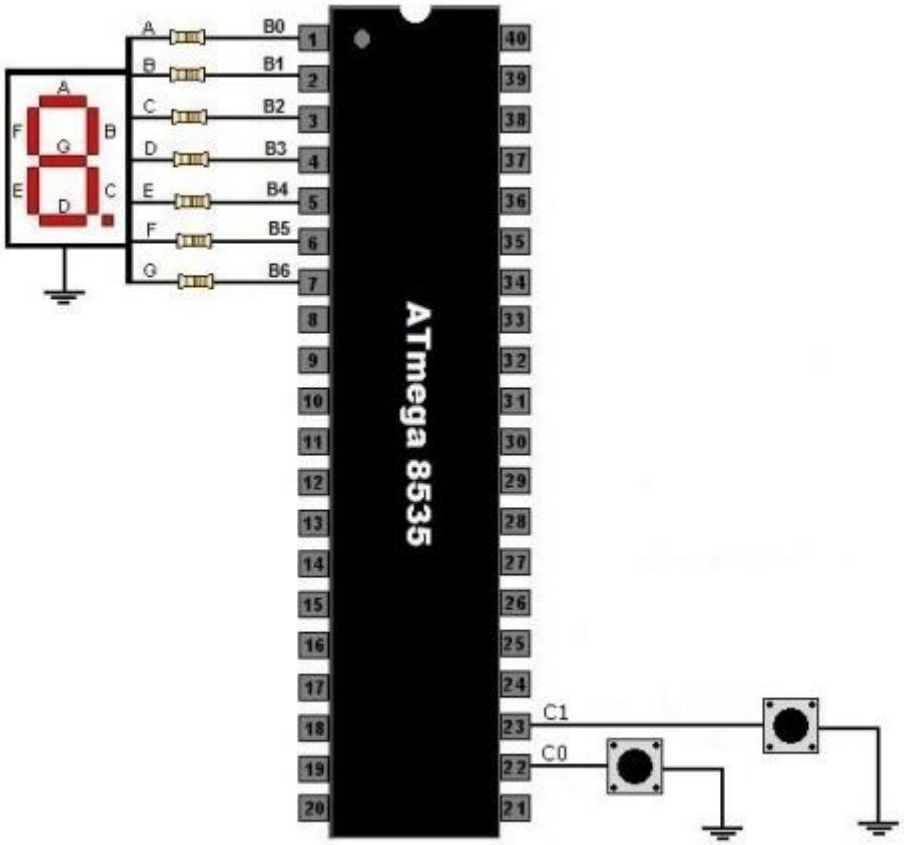


Figura 2. Circuito para el uso de la memoria EEPROM

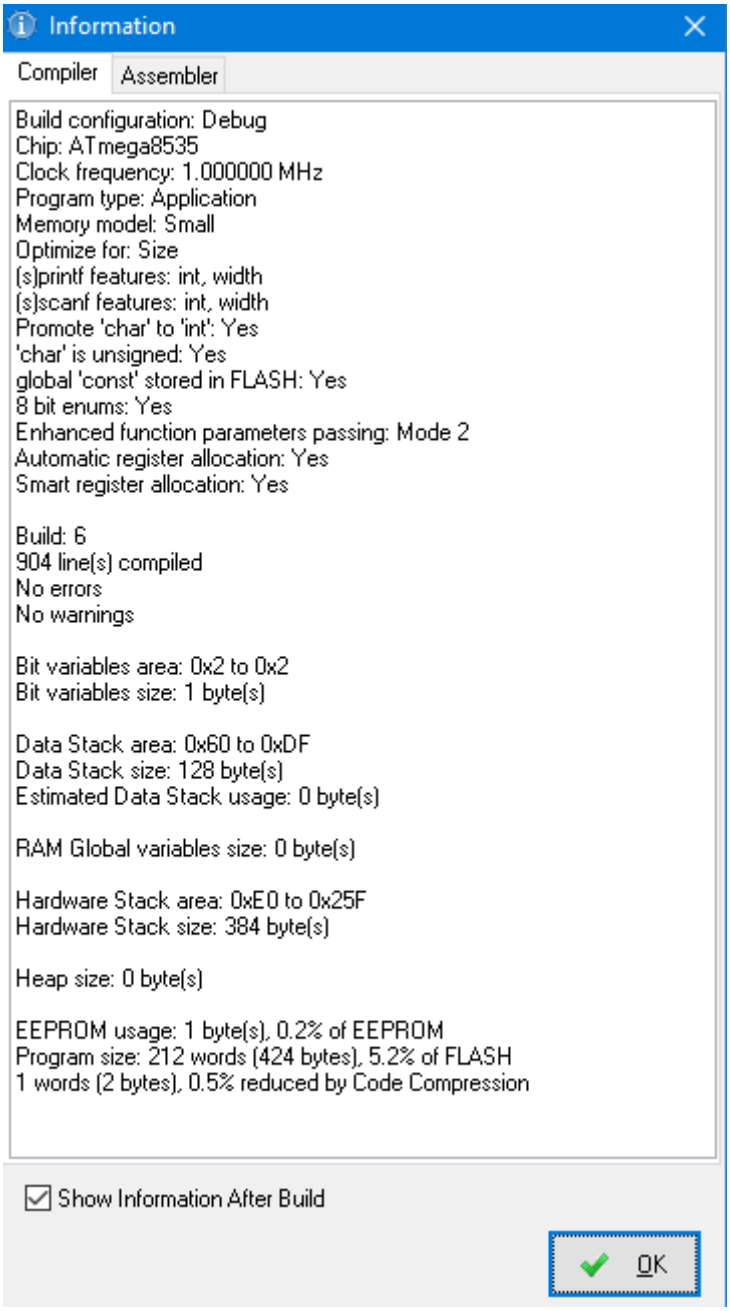


Figura 3. Compilación exitosa en CodeVision

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>
#include <delay.h>
#define boton PINC.0
#define boton_guarda PINC.1
bit botonp;
bit botona;
unsigned char var; //Ahora la variable se guarda en eeprom
const char tabla7segmentos
[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
eeprom char datoaguardar;

void checa_boton(void); // Aquí se declaran todas las funciones que
se van usar

// 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=In Bit1=In
Bit0=In
DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) |
(0<<DDC2) | (0<<DDC1) | (0<<DDC0);
// State: Bit7=P Bit6=P Bit5=P Bit4=P Bit3=P Bit2=P Bit1=P Bit0=P
PORTC=(1<<PORTC7) | (1<<PORTC6) | (1<<PORTC5) | (1<<PORTC4) |
(1<<PORTC3) | (1<<PORTC2) | (1<<PORTC1) | (1<<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);

    if (datoaguardar>10){
        datoaguardar=0;
    }
    var=datoaguardar;

    while (1)
    {
        checa_boton();
        PORTB=tabla7segmentos [var];
        if (boton_guarda==0) //Si se presiona el botón de guardar
            datoaguardar=var; //se grabara la eeprom con el valor de
var
    };

}

void checa_boton (void){
    if (boton==0)
        botona=0;
    else
        botona=1;
    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
    botonp=botona;
}
```

CIRCUITO ELECTRICO EN PROTEUS

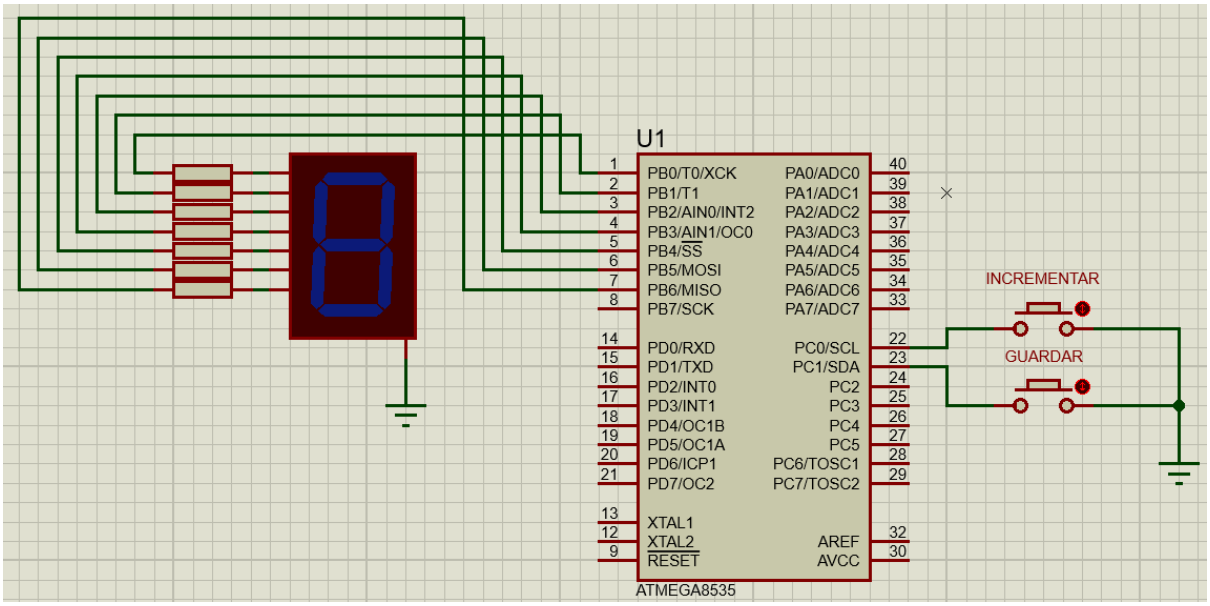
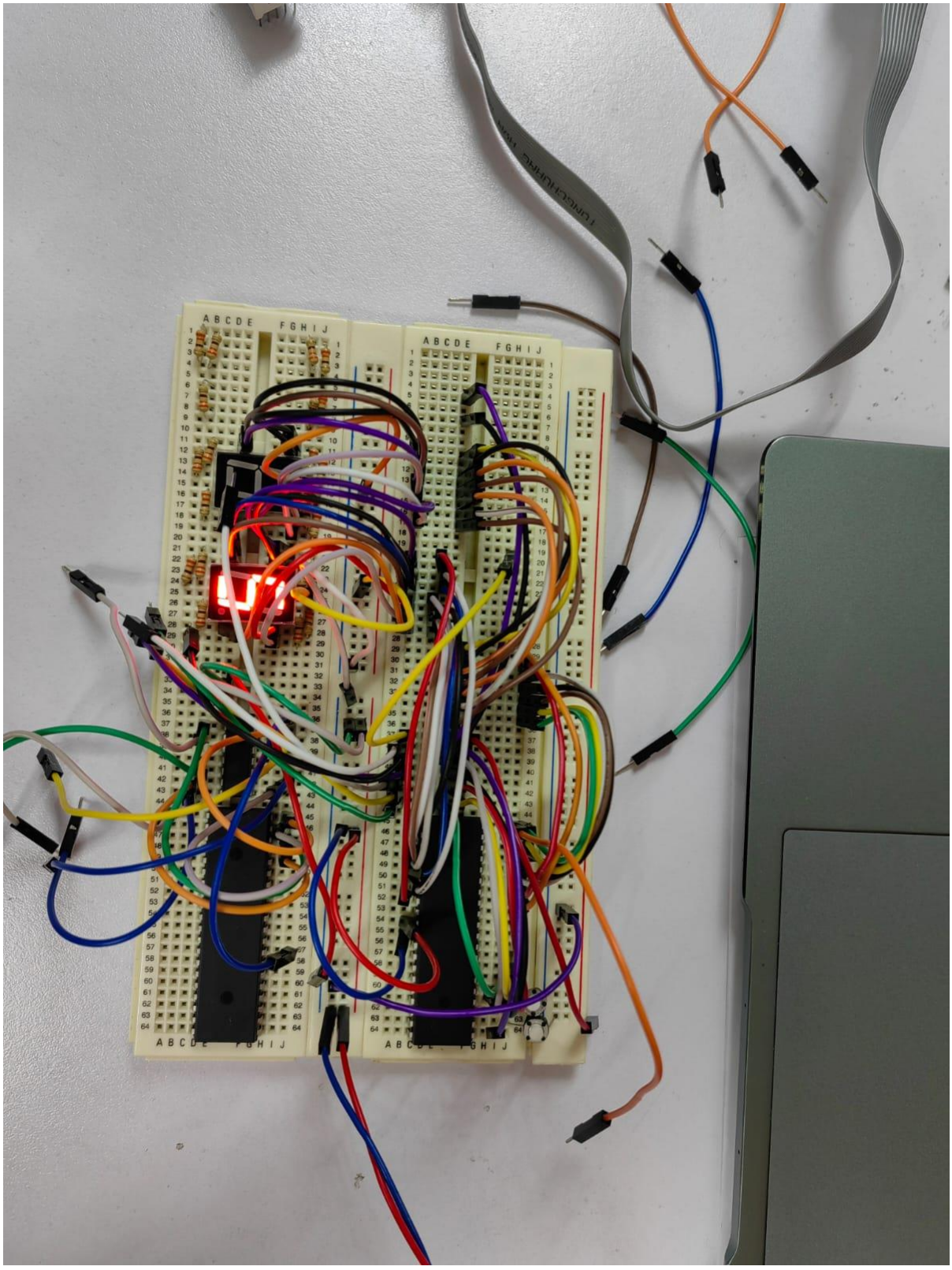


Figura 4. Circuito simulado en Proteus

CIRCUITO EN EL PROTO ARMADO



OBSERVACIONES Y CONCLUSIONES INDIVIDUALES

Malagon Baeza Alan Adrian

Las memorias EEPROM son dispositivos que nos permiten el almacenar información de tal manera que, si el circuito en el que está implementada la memoria deja de recibir alimentación eléctrica, la memoria no borrará el dato que se guardó en ella. Las memorias EEPROM destacan por ser fáciles de grabar y borrar información mediante pulso eléctricos, además de que esta información puede durar años almacenada dentro de la memoria sin sufrir daños.

Las memorias EEPROM se pueden conseguir en el mercado como dispositivos electrónicos listos para ser programados e implementados en nuestros circuitos electrónicos; sin embargo, con ayuda del lenguaje C y un microcontrolador

Martínez Chávez Jorge Alexis

Podemos desarrollar una memoria EEPROM dentro de nuestro microcontrolador mediante dos pulsadores los cuales podrán controlar el dato que se desea almacenar y la acción de guardado para que al quitar la energía del circuito y volver a conectar el circuito a la corriente eléctrica, se pueda apreciar el dato que se solicitó guardar en la memoria.

Para desarrollar el código de esta práctica no se presentó ninguna complicación; el código es muy similar a los códigos desarrollados previamente para los contadores con displays de 7 segmentos, sólo que, a diferencia de esos, en ésta práctica se hizo uso de la memoria EEPROM interna del microcontrolador para guardar el dato deseado y mostrarlo cada vez que se energiza el circuito.

BIBLIOGRAFÍA

[1] Beckerola, “¿Qué es una EEPROM?,” *Ordenadores y Portátiles*, May 01, 2020. https://www.ordenadores-y-portatiles.com/eeprom/ (accessed Mar. 22, 2022).