INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PRÁCTICA 19: INTERRUPCIONES EXTERNAS

ALUMNOS: MALAGON BAEZA ALAN ADRIAN
MARTINEZ CHAVEZ JORGE ALEXIS

GRUPO: 6CM3

U.A: SISTEMAS EN CHIP

PROFESOR: FERNANDO AGUILAR SÁNCHEZ

FECHA DE ENTREGA: 18 DE JUNIO DE 2023

OBJETIVO

Al término de la sesión, los integrantes del equipo contaran con la habilidad para manejar las interrupciones del Microcontrolador.

INTRODUCCIÓN TEÓRICA

INTERRUPCIONES EXTERNAS

Las interrupciones externas son activadas por los pines INT1 y INT0. Si son habilitadas, las interrupciones se activarán aun cuando los pines INT0/INT1 se configuren como salidas.[1]

Esta característica proporciona una manera de generar una interrupción por software. Las interrupciones externas pueden ser activadas por un flanco de bajada, subida o por un nivel bajo. Esto es establecido como se indica en la especificación para el registro de control MCU (MCUCR). Cuando la interrupción externa se habilita y se configura como activo por nivel, la interrupción se activará mientras el pin se mantenga a nivel bajo.[1]

Las interrupciones externas se establecen como se describe en la especificación para el registro de control MCU (MCUCR).[1]

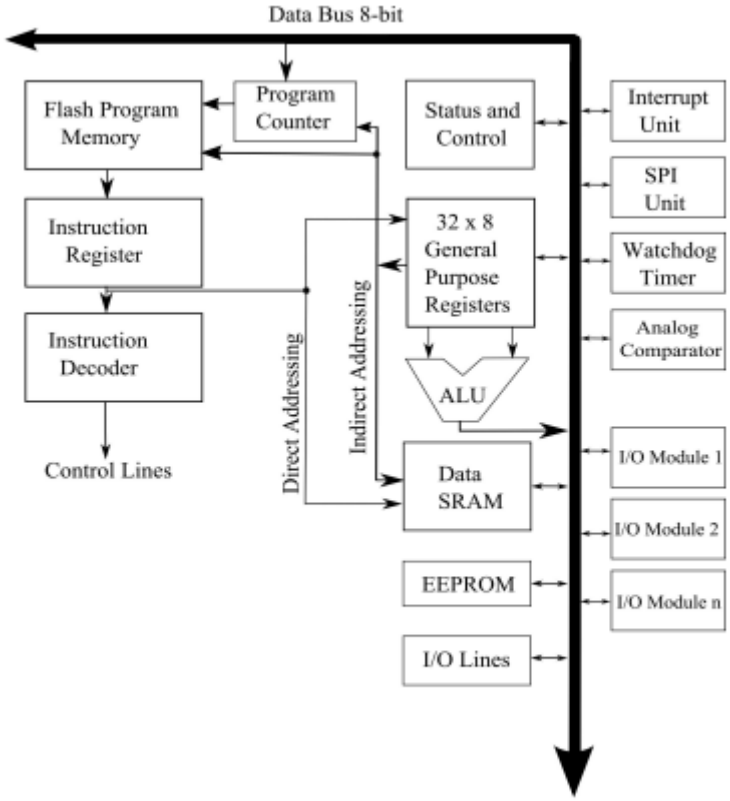


Figura 1. Diagrama a bloques del núcleo de microcontroladores AVR Atmega.

Tiempo de respuesta de interrupción

La respuesta de ejecución de interrupción para todas las interrupciones de AVR habilitadas es de cuatro ciclos de clock como mínimo. Cuatro ciclos de clock después de que el flag de interrupción se ponga a set, se ejecuta la dirección del vector de programa para la rutina de manejo de la interrupción. Durante este 4º periodo de ciclo de clock, el Contador de Programa

(2 bytes) es introducido en la pila y el puntero de pila es decrementado en 2. El vector es normalmente un salto relativo a la rutina de interrupción y este salto toma dos ciclos de clock. Si ocurre una interrupción durante la ejecución de una instrucción multi-ciclo, antes de que la interrupción sea servida, se completa esta instrucción.[1]

Un retorno de una rutina de manejo de interrupción (el mismo que una rutina de llamada a subrutina) toma cuatro ciclos de clock. Durante estos cuatro ciclos de clock, el Contador de Programa (2 bytes) se extrae de la pila, el puntero de pila es incrementado en 2 y el flag I en SREG está a set. Cuando el AVR termina una interrupción, siempre volverá al programa principal y ejecutará una instrucción más antes de que cualquier interrupción pendiente se sirva.[1]

Observe que el registro de estado (SREG) no es manejado por el hardware del AVR, para ninguna interrupcion ni subprograma. Para la rutina de manejo de interrupción se requiere un almacenamiento del SREG, esto debe ser realizado por el software del usuario.[1]

Para interrupciones activadas por eventos que pueden permanecer estáticos, cuando el evento ocurre, el flag de interrupción está a set. Si el flag de interrupción es borrado y la condición de interrupción persiste, el flag no se pondrá a set hasta que el evento ocurra la próxima vez. Observe que una interrupción de nivel externo sólo se recordará mientras que la condición de interrupción esté activa.[1]

MATERIALES Y EQUIPO EMPLEADO

- ✓ CodeVision AVR
- ✓ AVR Studio 4
- ✓ Microcontrolador ATmega 8535
- ✓ 1 Display Cátodo Común
- ✓ 8 Resistor de 1kΩ
- ✓ 2 Push Botón

DESARROLLO EXPERIMENTAL

1. Diseñe un programa que cuando haya un flanco de subida en INT0 se incremente el conteo del display que podrá contar de 0 a 9, y que cuando haya un nivel lógico de 0 en INT1 se decremente el valor del display.

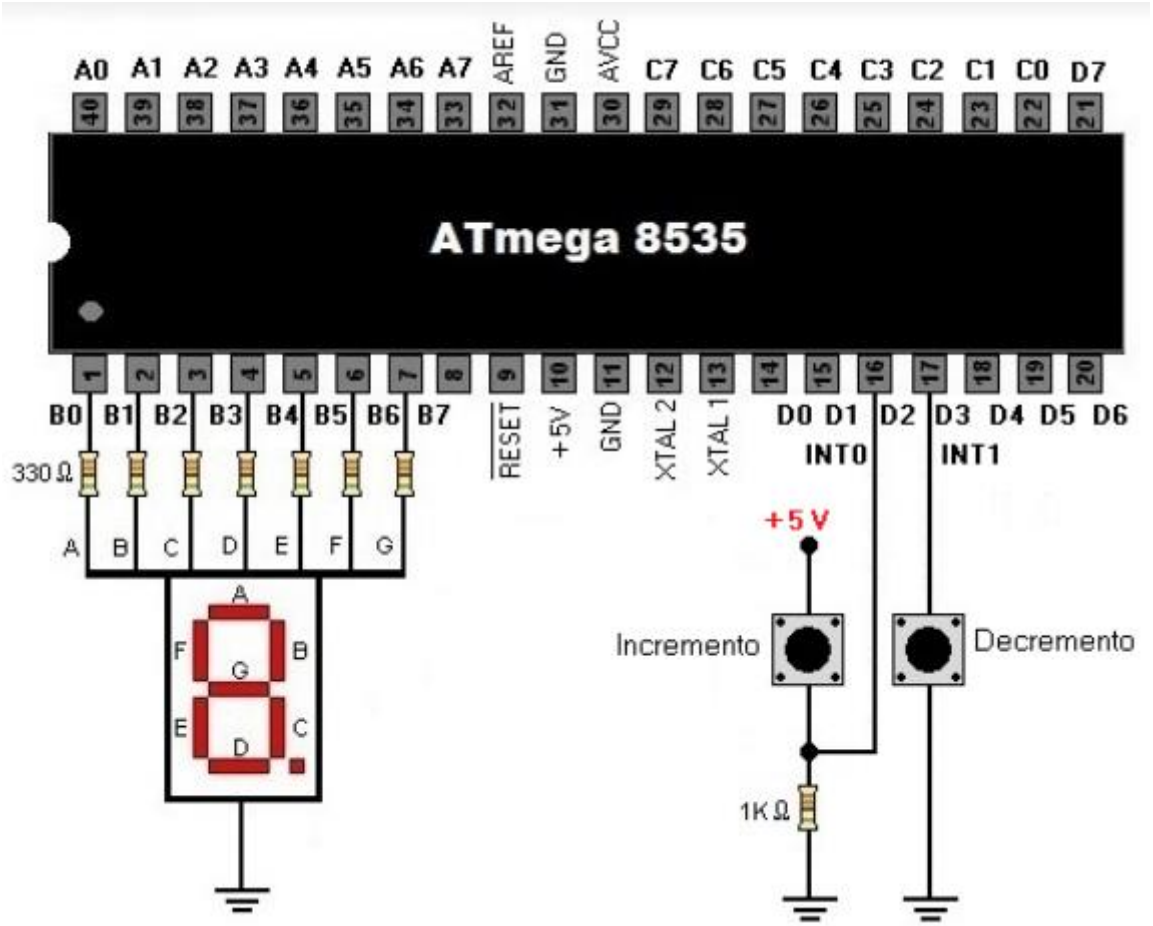


Figura 2. Circuito para las interrupciones INT0 y INT1.

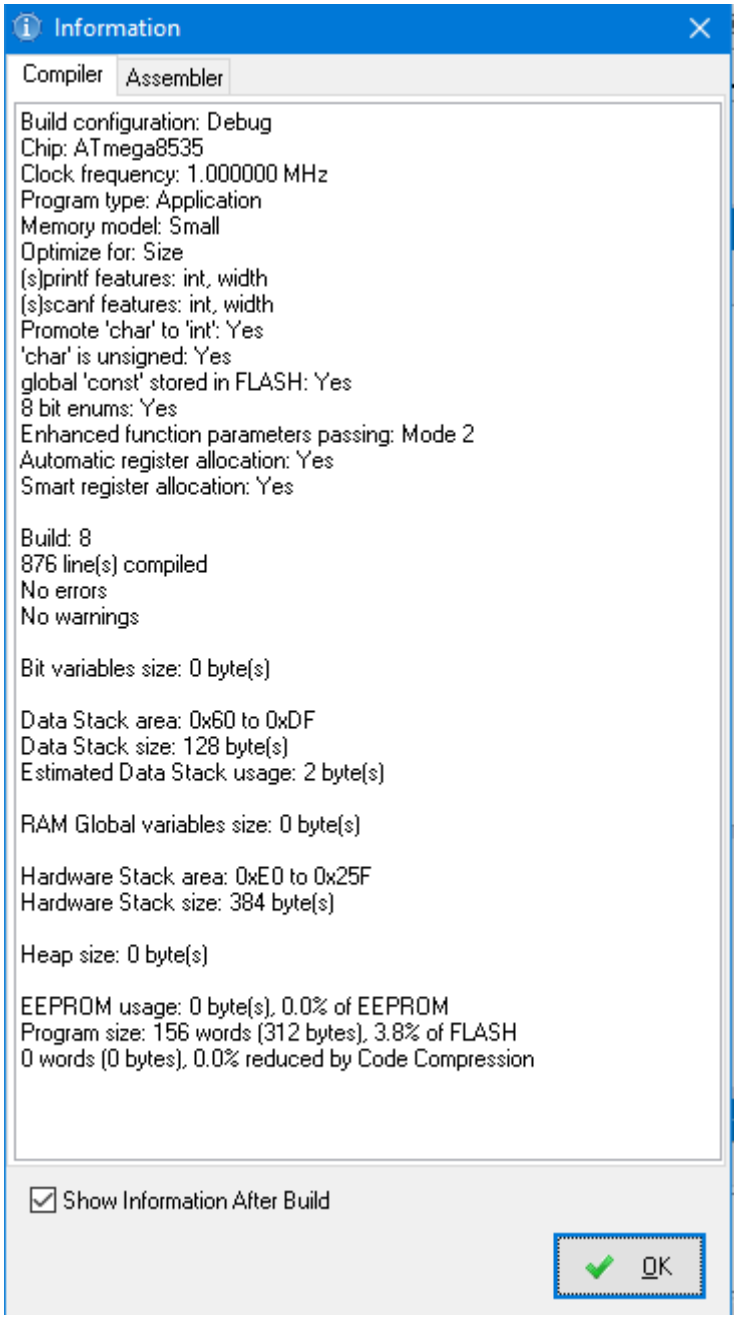


Figura 3. Compilación exitosa en CodeVision.

CÓDIGO GENERADO POR CODEVISION

```
/*****************************************************
This program was created by the
CodeWizardAVR V2.60 Evaluation
Automatic Program Generator
? Copyright 1998-2012 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

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>
int var=0;
const char tabla7segmentos
[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7c,0x07,0x7f,0x6f};
// Declare your global variables here

// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
    var++;
    delay_ms(77);
// Place your code here

}

// External Interrupt 1 service routine
interrupt [EXT_INT1] void ext_int1_isr(void)
{
// Place your code here
    var--;
    delay_ms(77);
}
```

```
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=1 Bit6=1 Bit5=1 Bit4=1 Bit3=1 Bit2=1 Bit1=1 Bit0=1
PORTB=(1<<PORTB7) | (1<<PORTB6) | (1<<PORTB5) | (1<<PORTB4) |
(1<<PORTB3) | (1<<PORTB2) | (1<<PORTB1) | (1<<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=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=P Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) |
(1<<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: On
// INT0 Mode: Rising Edge
// INT1: On
// INT1 Mode: Falling Edge
// INT2: Off
GICR|=(1<<INT1) | (1<<INT0) | (0<<INT2);
```

```
MCUCR=(1<<ISC11) | (0<<ISC10) | (1<<ISC01) | (1<<ISC00);
MCUCSR=(0<<ISC2);
GIFR=(1<<INTF1) | (1<<INTF0) | (0<<INTF2);

// 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
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);

// Global enable interrupts
#asm("sei")

while (1)
      {
      // Place your code here
      if(var==-1)
        var=9;
      else if(var>9)
        var=0;


      PORTB=~(tabla7segmentos [var]);
      delay_ms(77);
      }
}
```

CIRCUITO ELECTRICO EN PROTEUS

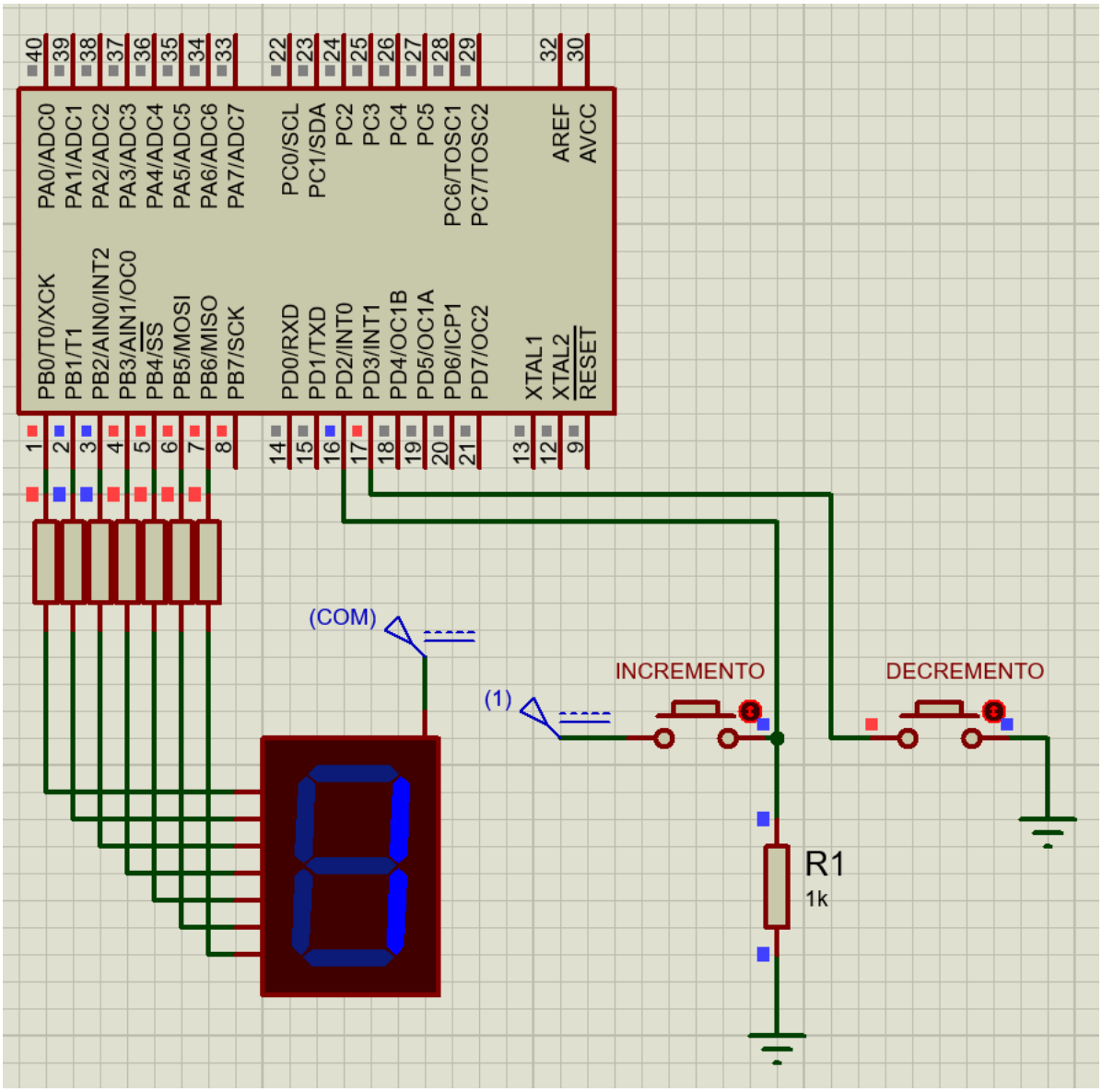
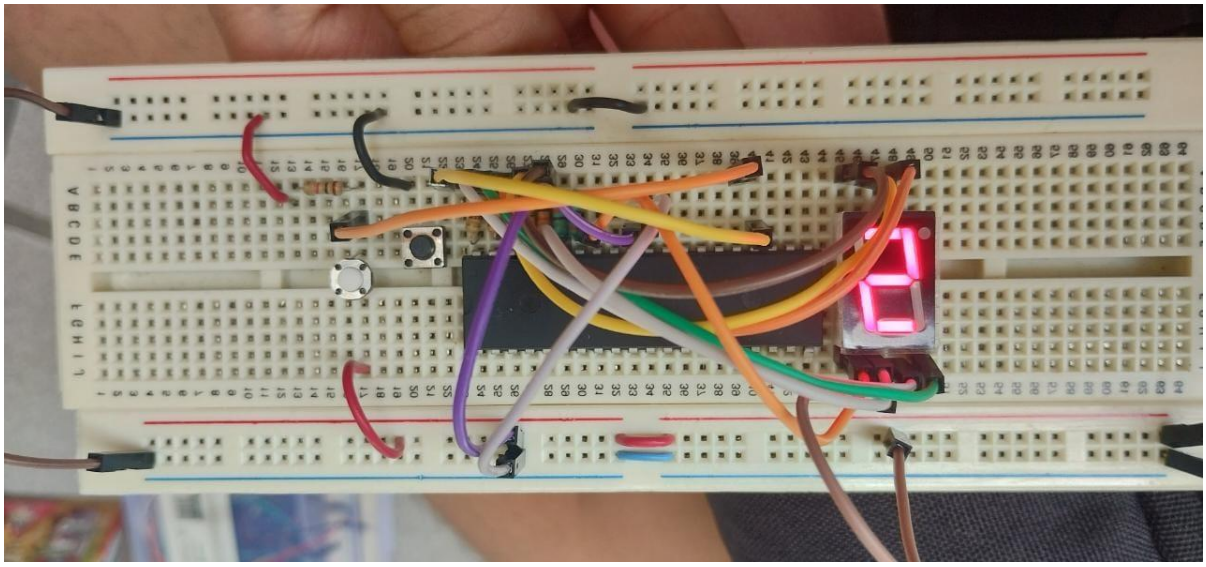


Figura 4. Circuito simulado en Proteus.

CIRCUITO EN EL PROTO ARMADO



OBSERVACIONES Y CONCLUSIONES INDIVIDUALES

Malagón Baeza Alan Adrian

Anteriormente habíamos visto como eliminar el rebote y detectar los flancos de subida y bajada en los push botón conectados a nuestros microcontroladores; si bien esas líneas de código desarrolladas anteriormente cumplían con la tarea establecida, cada vez que se necesitaba implementar un botón en nuestros circuitos era necesario colocar esas líneas de código. Este proceso se tenía que hacer con cada botón implementado y se requería de declarar las variables de botón previo y botón actual por cada uno, lo que hace que el programa sea más complejo y utilice más variables.

Martínez Chávez Jorge Alexis

Con las interrupciones externas ya no es necesario el implementar esas líneas de código en los botones necesarios en nuestros circuitos. Las interrupciones externas sirven para detectar un estado lógico o un cambio de estado en alguna de las terminales de entrada del microcontrolador. Son útiles para monitorear interruptores, botones o sensores con salida a relevador.

Su configuración en CodeVision es muy sencilla; utilizando el asistente para crear el proyecto se puede declarar el uso de las interrupciones externas, así como el nivel que se desea utilizar en cada una, y con esto CodeVision nos creará la sección del código para programar el comportamiento que se quiere realizar cuando se detecte la interrupción.

BIBLIOGRAFÍA

[1] “Interrup externas y tiempo respuesta int,” *www.sc.ehu.es*. http://www.sc.ehu.es/sbweb/webcentro/automatica/web_avr/archivos/Manual_AT90S8515/Arquitectura/interrup_externas&tiempo_respu.htm (accessed May 28, 2022).