INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PRÁCTICA 3: CONVERTIDOR BCD A 7 SEGMENTOS

ALUMNO: MALAGON BAEZA ALAN ADRIAN
MARTINEZ CHAVEZ JORGE ALEXIS

GRUPO: 6CM3

U.A: SISTEMAS EN CHIP

PROFESOR: FERNANDO AGUILAR SÁNCHEZ

FECHA DE ENTREGA: 16 DE ABRIL DE 2023

OBJETIVO

Al término de la sesión, los integrantes del equipo contaran con la habilidad de realizar un contador BCD empleando arreglos.

INTRODUCCIÓN TEÓRICA

CÓDIGO BCD

BCD (Decimal Codificado en Binario) es un código que representa valores decimales en formato binario, para ello forma grupos de 4 bits para representar cada valor del 0 al 9. El 9 es el valor máximo que se puede representar en un dígito decimal, si recordamos los números binarios el 9 es un 1001_2, requiere 4 bits, es por eso por lo que cada valor BCD se representa con 4 bits, del 0000_2 al 1001_2 (0 – 9). Hay que destacar que BCD es un código, no un sistema de numeración, por lo que no está diseñado para hacer operaciones como sumas o restas, solo para representar valores decimales en binario.[1]

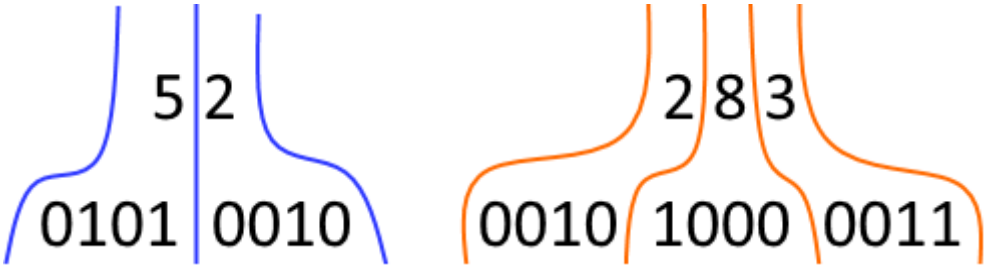


Figura 1. Ejemplo de conversión de decimal a BCD

En el ejemplo de la figura 1 se muestra como un valor 52 decimal se puede codificar en binario utilizando BCD simplemente convirtiendo cada dígito decimal a su equivalente binario (un valor entre 0000_2 y 1001_2) y unir el resultado en un solo valor binario, en el caso de la izquierda resultaría en 01010010_2. Hay que destacar que no es equivalente a convertir 52 en binario, sino que es la unión del valor binario de cada dígito. Para codificar en BCD el 283 decimal se realiza el mismo proceso.[1]

SISTEMA HEXADECIMAL

El sistema hexadecimal es aquel que utiliza entonces dieciséis dígitos, que serán los siguientes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.[2]

En dicho conjunto, las letras del alfabeto latino tienen el siguiente valor expresado en el sistema decimal: A=10, B=11, C=12, D=13, E=14 y F=15.[2]

Cabe señalar que estas letras podrían colocarse en minúsculas. Además, vale indicar que este sistema es posicional, pues el valor de cada dígito dependerá de su posición, como explicaremos continuación.[2]

Para pasar un número del sistema hexadecimal al decimal, tendría que multiplicarse cada dígito, de derecha a izquierda, por una potencia de 16, que irá de menor a mayor empezando de 0.[2]

El sistema hexadecimal tiene un uso sobre todo en temas informáticos. Esto, debido a que cada byte representa 2^8 valores posibles. Por tanto, esto sería equivalente a 100 en el sistema hexadecimal.[2]

En 1963 fue la empresa IBM la que introdujo por primera vez este sistema de numeración. La computadora Bendix G-15 hizo lo propio en 1956, pero en lugar de usar las letras de la A a la F, utilizaron las letras de la U hasta la Z. Dicho ordenador fue creado por la Corporación Bendix, de Los Ángeles, California, y estaba destinado principalmente a un uso científico e industrial.[2]

DISPLAY DE 7 SEGMENTOS

El display de 7 segmentos es un arreglo de 7 LED's rectangulares colocados formando un 8 como se muestra en la siguiente imagen:[1]

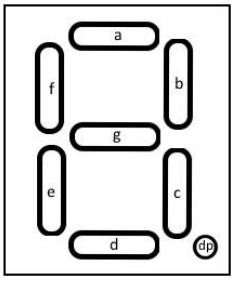


Figura 2. Formación de LED's en un display de 7 segmentos

Cada uno de esos segmentos se identifica con una letra de la "a" a la "g" como se muestra en la imagen, muchas veces se incluye un punto nombrado "dp", que también es un LED, pero circular. Con este arreglo se pueden formar los dígitos del 0 al 9 encendiendo la combinación de LED's adecuada, por ejemplo, los segmentos a, b y c pueden mostrar un 7.[1]

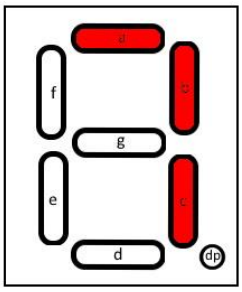


Figura 3. Display mostrando un 7

Recordando los LED's (Diodo Emisor de Luz) tienen dos terminales llamadas Ánodo (+) y Cátodo (-), si cada display tiene 7 LED's para formar los dígitos y además se tiene un LED adicional para un punto, se tendrían 8 LED's, cada uno con dos terminales, entonce el display debería tener en total 16 terminales, pero no es así. De las dos terminales que tiene cada LED, se conectan juntos ya sean todos los ánodos en un punto común y se dejan individuales los cátodos para encender individualmente cada LED, o al revés, se conectan en un punto común todos los cátodos y se dejan individuales todos los ánodos para poder encender independientes. Esto genera entonces dos tipos de display: Display de ánodo común y Display de cátodo común.[1]

Para conectar correctamente estos display, al igual que cualquier LED, se debe colocar una resistencia en serie con cada LED para limitar la corriente que circula por cada uno de ellos, en total serían 7 resistencias. Si es un display de ánodo común, el pin común se coloca a (+) y se tendría que poner (-) en los cátodos de cada uno de los segmentos (a – g). Contrario si es un display de cátodo común, el común tendría que ir a (-) y quedarían libres los ánodos que tendrían que colocarse a (+) para encender los segmentos (a – g).[1]

MATERIALES Y EQUIPO EMPLEADO

- ✓ CodeVision AVR
- ✓ AVR Studio 4
- ✓ Microcontrolador ATmega 8535
- ✓ 1 Display ánodo común
- ✓ 1 Display cátodo común
- ✓ 14 Resistores de 330 Ω a 1⁄4 W

DESARROLLO EXPERIMENTAL

1. Diseñe un convertidor BCD a 7 Segmentos para un Display Cátodo común. Observe la siguiente tabla:

Número Display	Combinaciones								Valor Hexadecimal
	.	g	f	e	d	c	b	a	
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	1	1	1	1	0x6F
E	0	1	1	1	1	0	0	1	0x79

Tabla 1. Conversiones para BCD (Cátodo común)

NOTA: Recuerde elaborar la codificación para el Display Ánodo común.

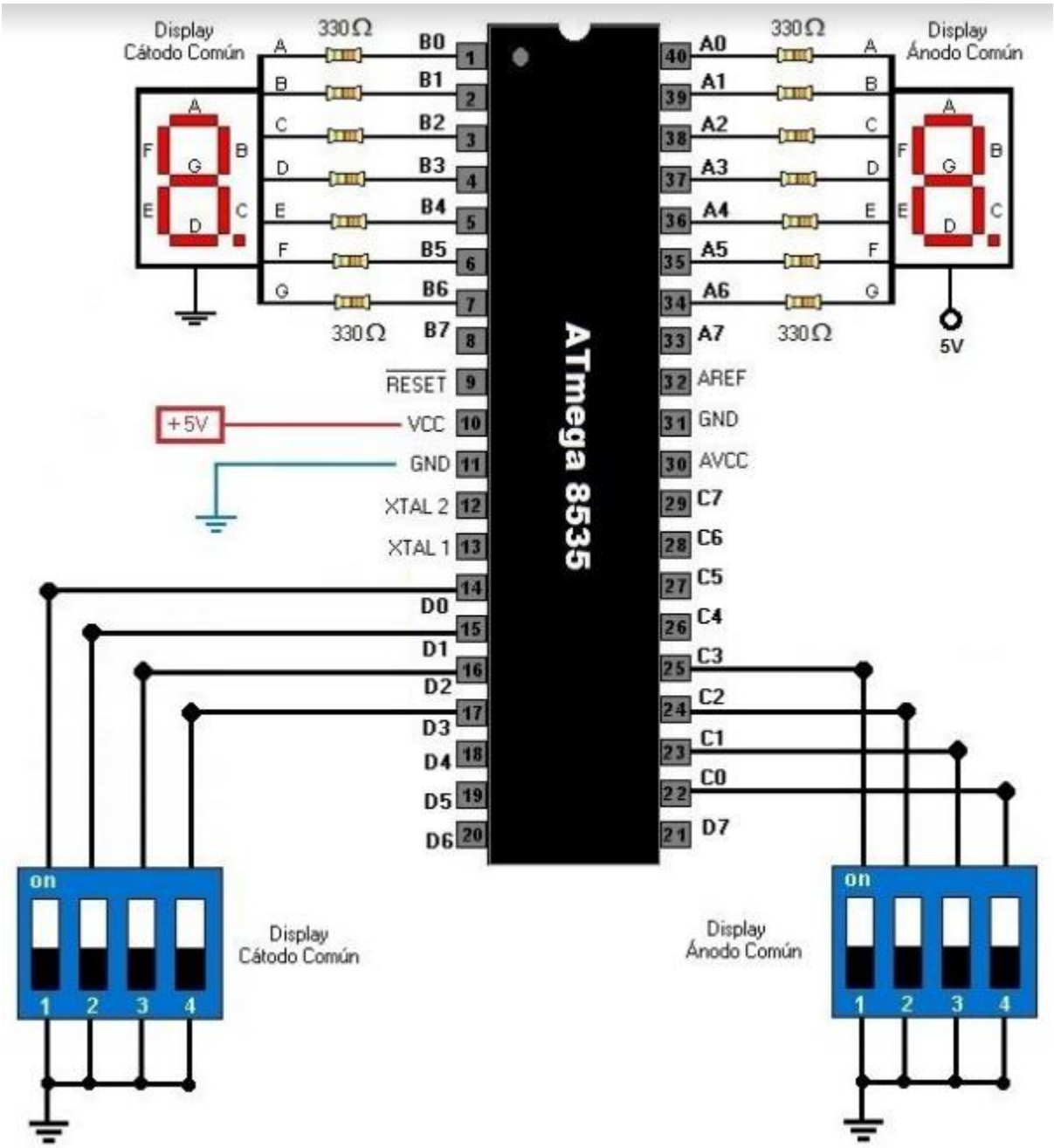


Figura 4. Circuito para el convertidor BCD a 7 segmentos con los displays ánodo y cátodo común

Número Display	Combinaciones								Valor Hexadecimal
	.	g	f	e	d	c	b	a	
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F

9	0	1	1	0	1	1	1	1	0x6F
A	0	1	1	1	0	1	1	1	0x77
B	0	1	1	1	1	1	0	0	0x7C
C	0	0	1	1	1	0	0	1	0x39
D	0	1	0	1	1	1	1	0	0x5E
E	0	1	1	1	1	0	0	1	0x79
F	0	1	1	1	0	0	0	1	0X71

Tabla 2. Conversiones para Hexadecimal (Cátodo común)

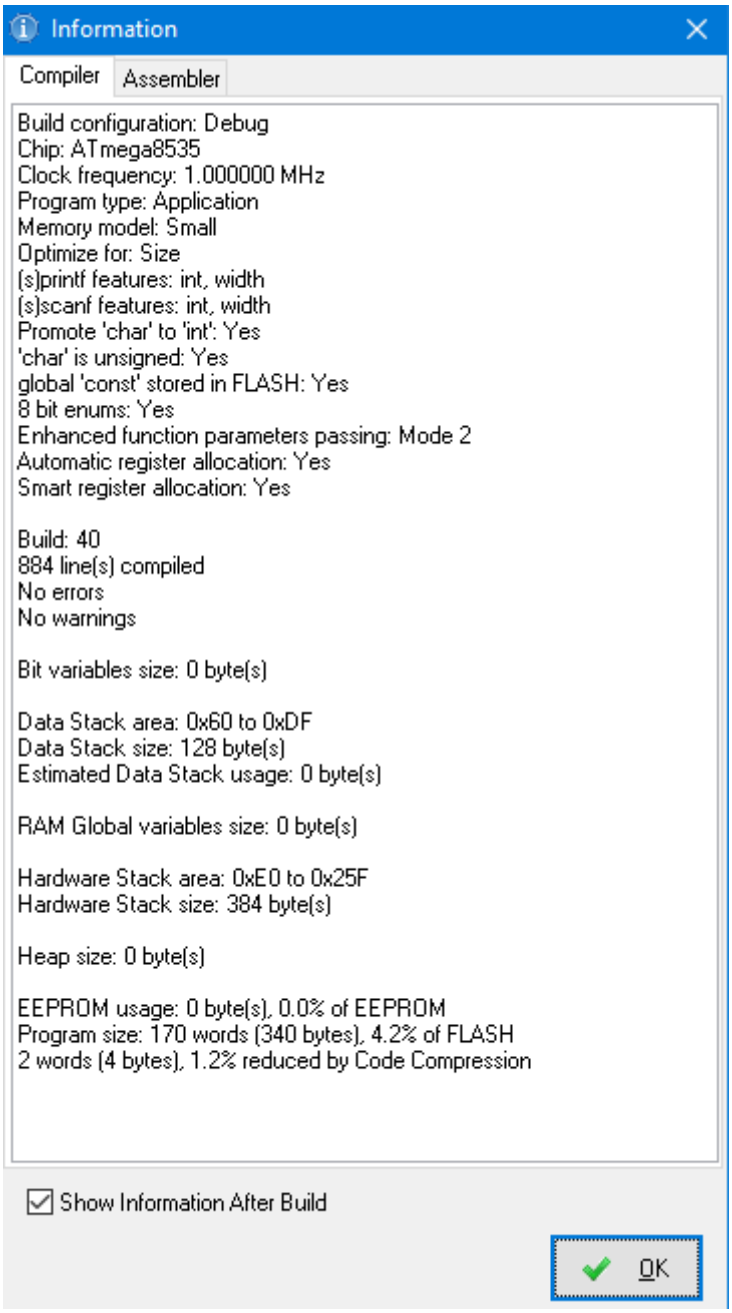


Figura 5. Compilación exitosa en CodeVision

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>

// Declare your global variables here
unsigned char anodo, catodo;
const char tablaBCD[11] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
0x07, 0x7f, 0x6f, 0x79};
const char tablaHEX[16] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};

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

// Input/Output Ports initialization
// Port A initialization
// Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out
Bit1=Out Bit0=Out
DDRA=(1<<DDA7) | (1<<DDA6) | (1<<DDA5) | (1<<DDA4) | (1<<DDA3) |
(1<<DDA2) | (1<<DDA1) | (1<<DDA0);
// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
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=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){
      //Display de 7 segmentos cátodo común
      if(!PIND.4){  //Si el pin 4 del puerto D está en 0 ->
Convertidor BCD
        catodo = PIND & 0x0f; //Enmascaramos los 4 bits menos
significativos del puerto A ya que los demás no interesan
        if(catodo < 10){
            PORTB = tablaBCD[catodo];
        }
        if(catodo >= 10){ //Si lo que leemos es mayor o igual de 10
que dibuje en el display una E de ERROR
            PORTB = tablaBCD[10];
        }
      }
      if(PIND.4){   //Si el pin 4 del puerto D está en 1 ->
Convertidor HEX
        catodo = PIND & 0x0f;
        PORTB = tablaHEX[catodo];
      }

      //Display de 7 segmentos ánodo común
      if(!PINC.4){
        anodo = PINC & 0x0f;
        if(anodo < 10){
            PORTA = ~tablaBCD[anodo];
        }
        if(anodo >= 10){
            PORTA = ~tablaBCD[10];
        }
      }
      if(PINC.4){
        anodo = PINC & 0x0f;
        PORTA = ~tablaHEX[anodo];
      }

      }
```

```
}
```

CIRCUITO ELECTRICO EN PROTEUS

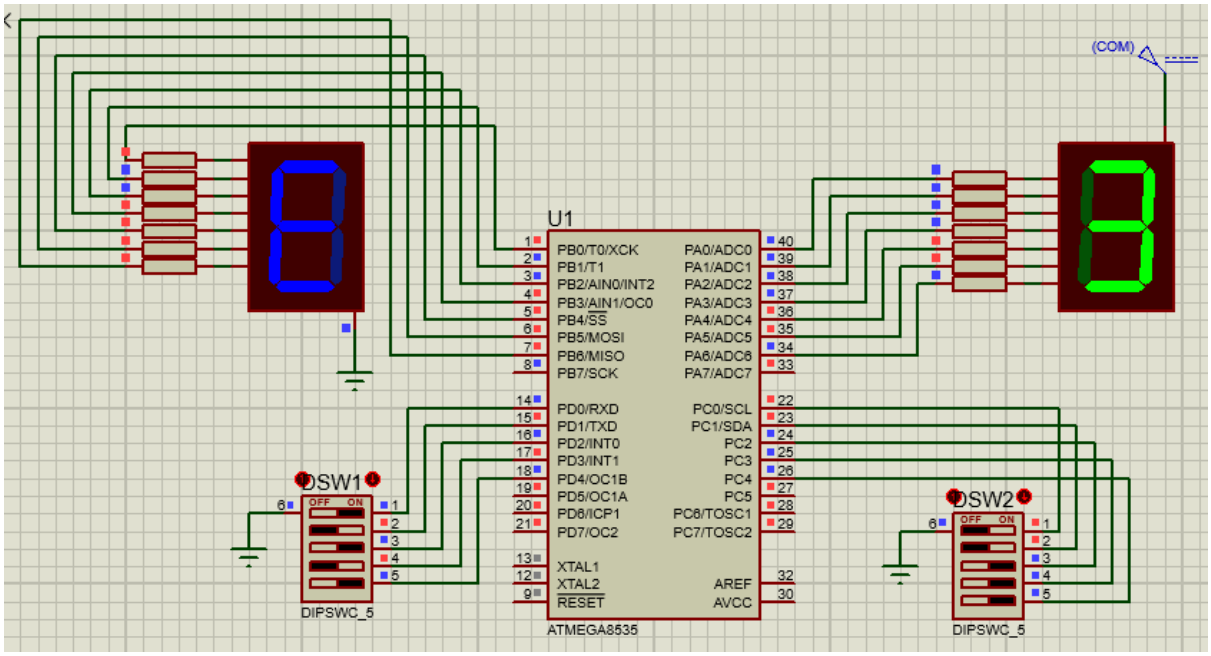


Figura 6. Circuito simulado en Proteus con convertidor BCD en ambos displays

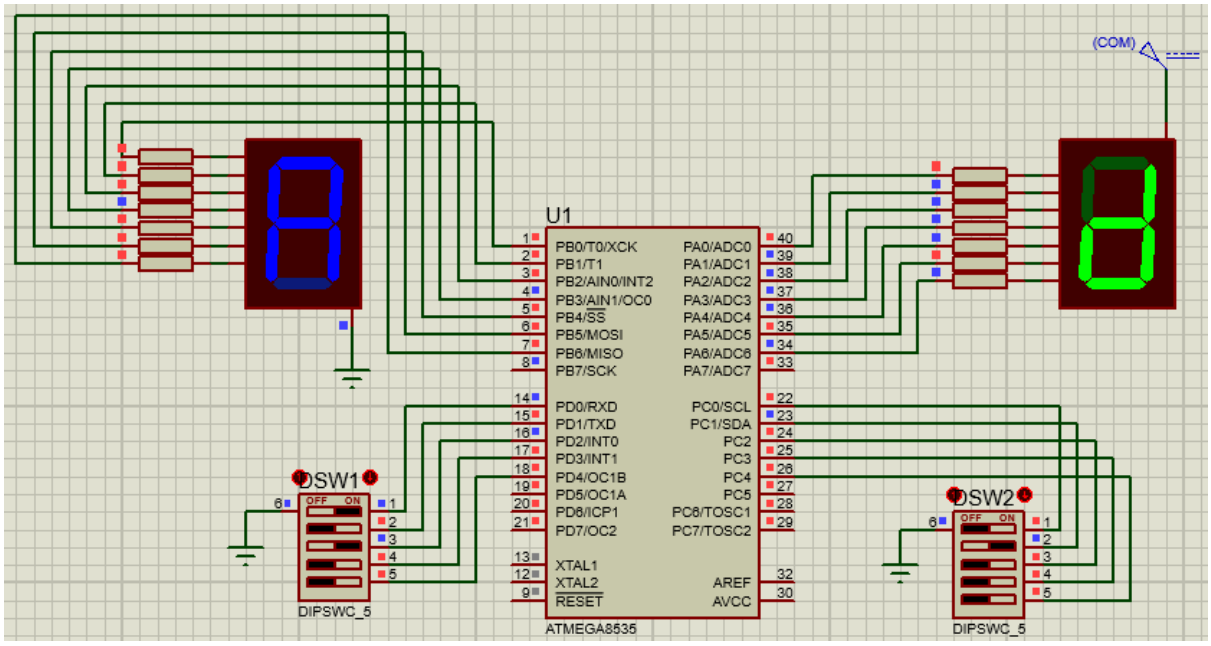
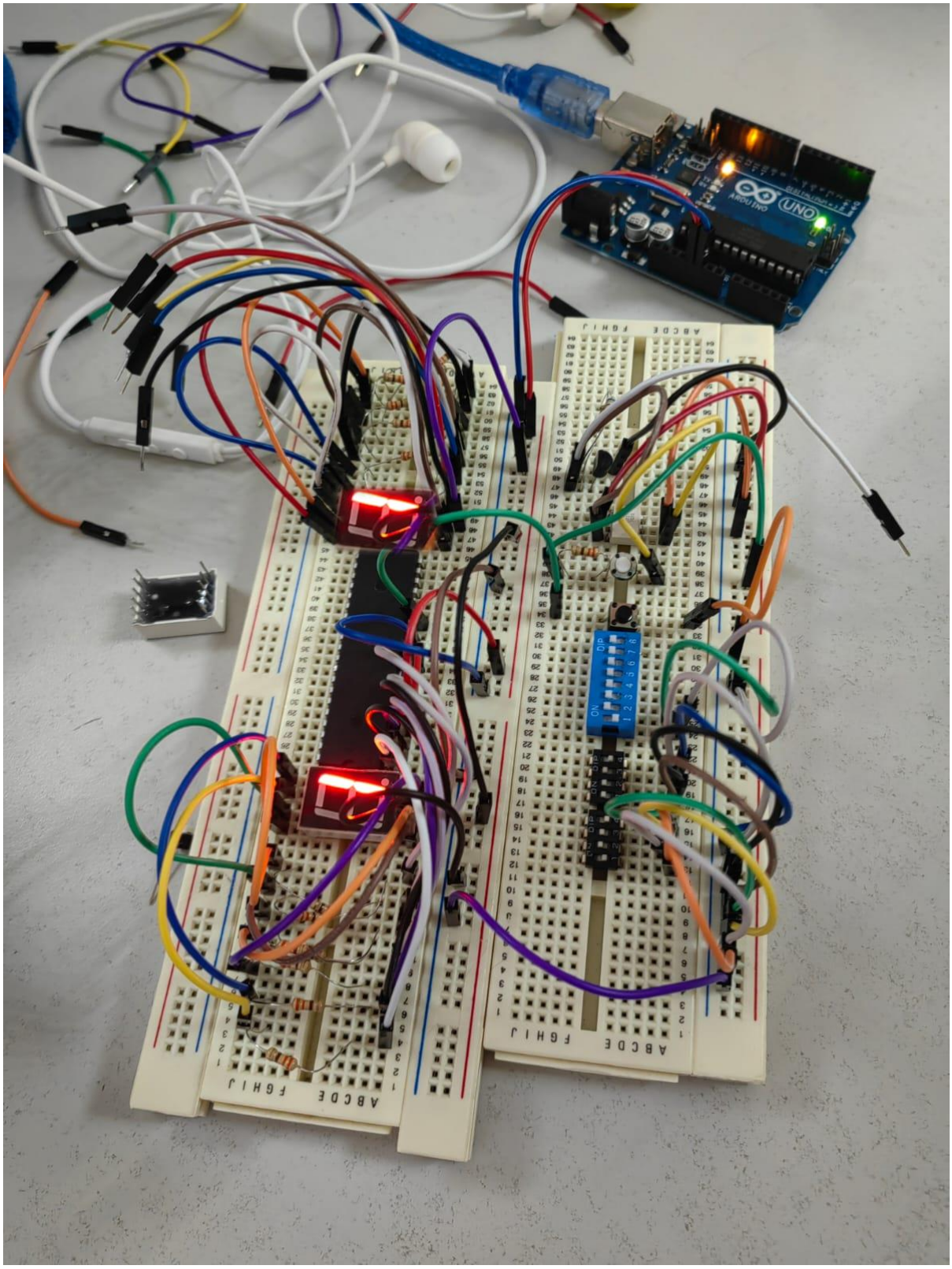


Figura 7. Circuito simulado en Proteus con convertidor Hexadecimal en ambos displays

CIRCUITO EN EL PROTO ARMADO



OBSERVACIONES Y CONCLUSIONES INDIVIDUALES

Malagón Baeza Alan Adrián

Con la realización de esta práctica pude manejar un display de 7 segmentos en su configuración de cátodo y ánodo común en el microcontrolador ATmega8535. Mediante el uso de arreglos codificados en lenguaje C se pueden cargar los datos que deseamos que maneje el controlador, en este caso fueron las combinaciones en hexadecimal de los valores de salida que se requerían para poder visualizar un número en los displays. Cabe señalar que las dos configuraciones de displays comparten la misma lógica, pero su principal diferencia es que el de cátodo común prenderá cuando el estado lógico sea 1, mientras que el de ánodo común prenderá cuando el estado lógico sea 0.

Para poder realizar las codificaciones de los números en el sistema BCD se realizó previamente una tabla con las conversiones de los estados lógicos para los segmentos del display en su configuración de cátodo común. Para realizar la codificación de los números para el display en su configuración ánodo común no es necesaria la realización de la tabla para el cátodo común; se utilizó el operador de negación (~) para el valor de la tabla de conversiones para el display cátodo común, recordando que la principal diferencia entre ambas configuraciones es que dónde había un 1 ahora se necesita un 0 y viceversa.

Martínez Chávez Jorge Alexis

Además de realizar una conversión de BCD a 7 segmentos, se realizó una conversión de Hexadecimal a 7 segmentos. Se utilizó el mismo proyecto para poder implementar ambos convertidores en un circuito. Para que el circuito pudiera tener ambos convertidores se implementó un dip switch de 5 posiciones en lugar del dip switch de 4 posiciones propuesto en el diseño de la práctica, en dónde la 5ta posición nos servirá para poder indicarle al circuito que convertidor se desea utilizar (siendo el estado 0 para el convertidor BCD y el estado 1 para el convertidor Hexadecimal).

Los convertidores de código nos son útiles para poder realizar cambios entre los sistemas de numeración que utilizan los dispositivos electrónicos para interpretar la información que viaja a través de sus puertos de entrada y salida. El display de 7 segmentos es una herramienta que nos sirve para poder visualizar la información obtenida en nuestro circuito de forma que podamos darle una interpretación con los números y letras que conocemos de nuestro lenguaje.

BIBLIOGRAFÍA

[1] J. F. Villa, “Decodificadores de BCD a 7 segmentos,” *7Robot - Mobile Education and Engineering*, Apr. 20, 2020. https://wp.7robot.net/decodificadores-de-bcd-a-7-segmentos/ (accessed Feb. 14, 2022).
[2] G. Westreicher, “Sistema hexadecimal,” *Economipedia*, Aug. 16, 2020. https://economipedia.com/definiciones/sistema-hexadecimal.html (accessed Feb. 14, 2022).