INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO.

INTRODUCCIÓN A LOS MICROCONTROLADORES.

REPORTE PRÁCTICA 3.

PROFESOR: AGUILAR SANCHEZ FERNANDO

Grupo: 3CM6

Alumno: Rojas Alvarado Luis Enrique

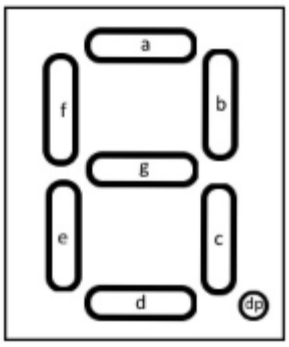
Práctica 3. Convertidor BCD a 7 segmentos.

Introducción teórica.

Un decodificador es un elemento digital que funciona a base de estados lógicos, con los cuales indica una salida determinada basándose en un dato de entrada característico, su función operacional se basa en la introducción a sus entradas de un número en código binario correspondiente a su equivalente en decimal para mostrar en los siete pines de salida establecidos para el integrado, una serie de estados lógicos que están diseñados para conectarse a un elemento alfanumérico en el que se visualizará el número introducido en las entradas del decodificador. [1]

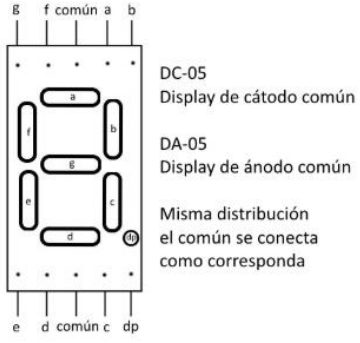
Código BCD: (Decimal Codificado en Binario) es un código que representa valores decimales en formato binario, para ello forma grupos de 4 bits para representar cada valor del 0 al 9. El 9 es el valor máximo que se puede representar en un dígito decimal, si recordamos los números binarios el 9 es un 10012, requiere 4 bits, es por eso que cada valor BCD se representa con 4 bits, del 00002 al 10012 (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. [2]

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



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. Existen diferentes modelos de display de 7 segmentos, cambian en tamaño y en la distribución de las terminales, aunque todos deben ser de uno de estos dos tipos, ya sea de ánodo común o cátodo común. La distribución de las terminales varía de un modelo a otro, se tiene que consultar su hoja de datos para identificar cada terminal. Un ejemplo de display típico es el DA-05 o DC-05:



Pines de un DC-05 y DA-05

jExisten diferentes modelos de display de 7 segmentos, cambian en tamaño y en la distribución de las terminales, aunque todos deben ser de uno de estos dos tipos, ya sea de ánodo común o cátodo común. La distribución de las terminales varía de un modelo a otro, se tiene que consultar su hoja de datos para identificar cada terminal. Un ejemplo de display típico es el DA-05 o DC-05: [2]

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	0x7C
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

Código:

```
/*****************************************************
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    : 19/10/2020
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 variable, variable2;
const char tabla7segmentos [10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7c,0x07,0x7f,0x6f
};

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=1 Bit6=1 Bit5=1 Bit4=1 Bit3=1 Bit2=1 Bit1=1 Bit0=1
PORTA=(1<<PORTA7) | (1<<PORTA6) | (1<<PORTA5) | (1<<PORTA4) | (1<<PORTA3) | (1<<PORTA2) | (1<<PORTA1) | (1<<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
      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){
          variable=PIND&0x0f; //Enmascaramos los 4 bits menos significativos del puerto A ya que los demás no interesan.
          variable2=PINC&0x0f;

          if (variable<10){
```

```
137.                PORTB=tabla7segmentos[variable];
138.            }
139.            else if (variable>=10){ //Si lo que leemos es mayor o igual de 10 que
    dibuje en el display una E de ERROR
140.                PORTB=0x79;
141.            }
142.            if (variable2<10){
143.                PORTA=~(tabla7segmentos[variable2]);
144.            }
145.            else if (variable2>=10){ //Si lo que leemos es mayor o igual de 10 que
     dibuje en el display una E de ERROR
146.                PORTA=~(0x79);
147.            }
148.          }
149.        }
```

Simulación:

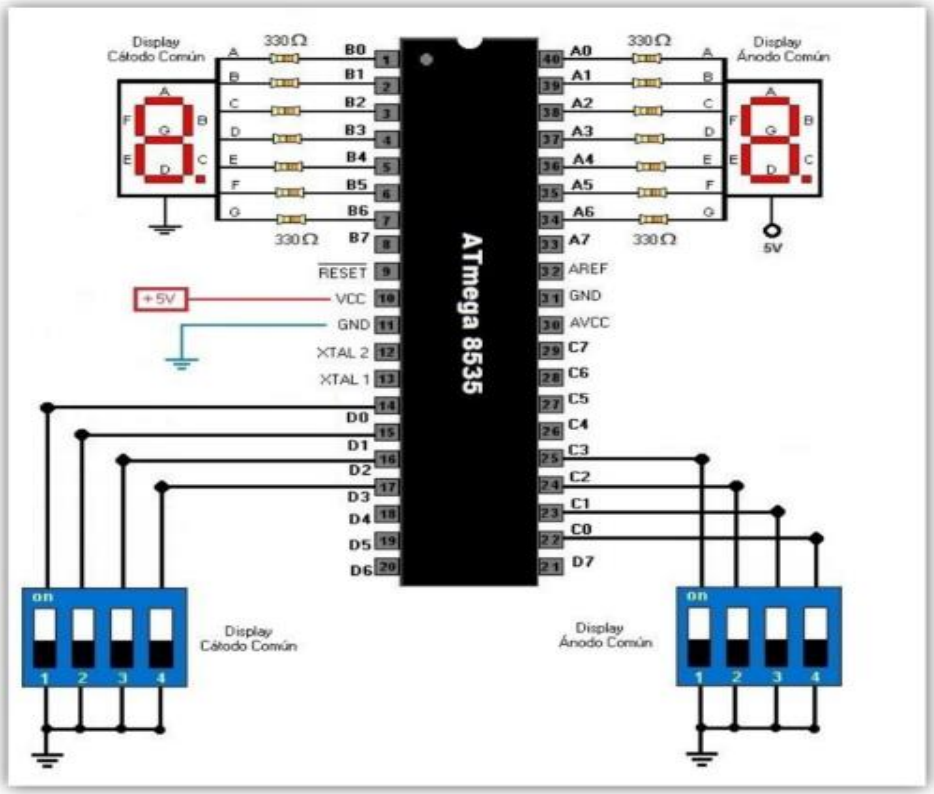
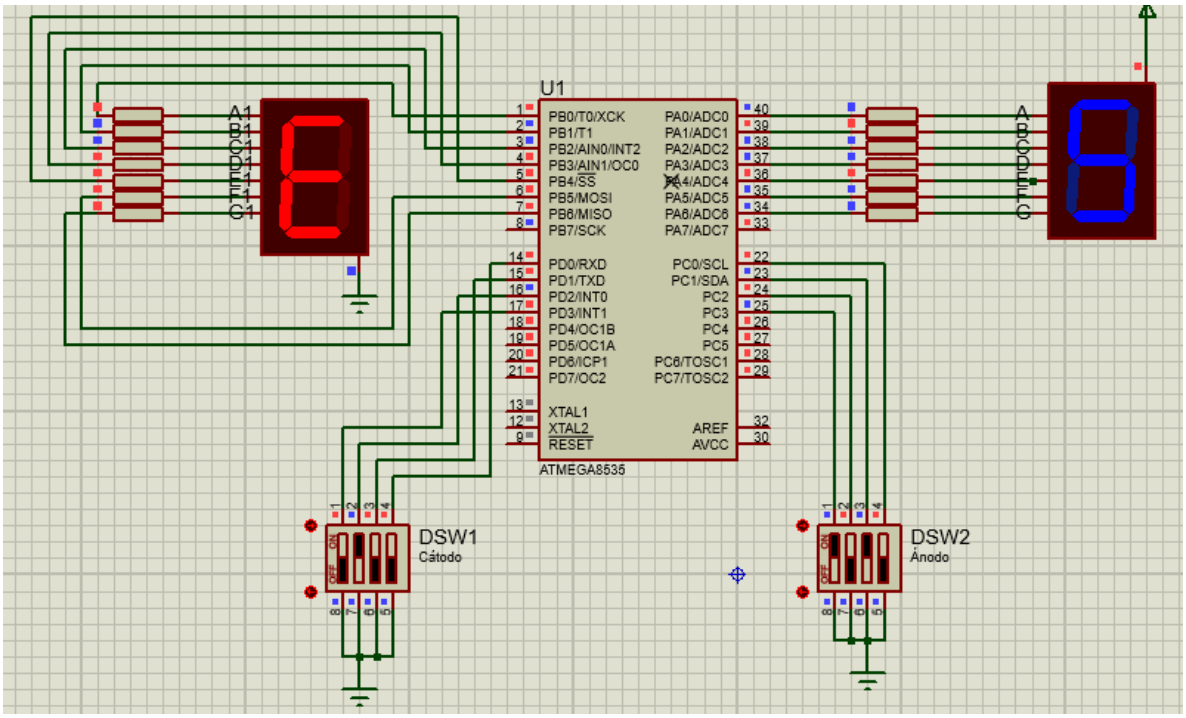


Figura 1. Circuito para el convertidor BCD a 7 segmentos con los displays ánodo y cátodo común.



Conclusiones

En esta práctica pudimos observar el comportamiento del ATMEGA 8535 cuando lo programamos para ser un decodificador BCD a 7 segmentos, cabe destacar que ya no utilizamos en decimal el número a convertir (o el número que queremos visualizar en el display) y por casos como hacíamos con una GAL, en lenguaje VHDL. En este caso usamos el registro hexadecimal de cada número a convertir y con ayuda del leguaje C, lo colocamos en un arreglo al que posteriormente con un índice y un condicional podemos acceder a ese número. Para el caso del display de ánodo común, era necesario negar toda la información que utilizamos con el cátodo común, utilizando un operador " ~ " el cual saca el complemento a 2 de lo que venga después de ese signo, haciendo que ya no tengamos que declarar los valores contrarios que ya habíamos calculado, sino simplemente los niega y podemos utilizarlo con el display de ánodo común.

Bibliografía:

[1] Electrónica digital Circuitos, Decodificador BCD a 7 segmentos, Tomado de: https://sites.google.com/site/electronicadigitalmegatec/home/deccoder-bcd-a-7-segmentos,. Recuperado el 19/10/2020

[2] jfvilla, "Decodificadores BCD a 7 segmentos", 20 de abril del 2020 Recuperado de: https://wp.7robot.net/decodificadores-de-bcd-a-7-segmentos/, Recuperado el 19/10/2020