INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

INTRODUCCIÓN A LOS MICROCONTROLADORES

PROFESOR:

Aguilar Sanchez Fernando

ALUMNOS:

Arruti Sánchez Alondra
Carrillo Soto Cristian Eduardo

Grupo: 3CM14

Fecha de Entrega: 05/10/22

PRÁCTICA 3

Convertidor BCD a 7 Segmentos

Introducción

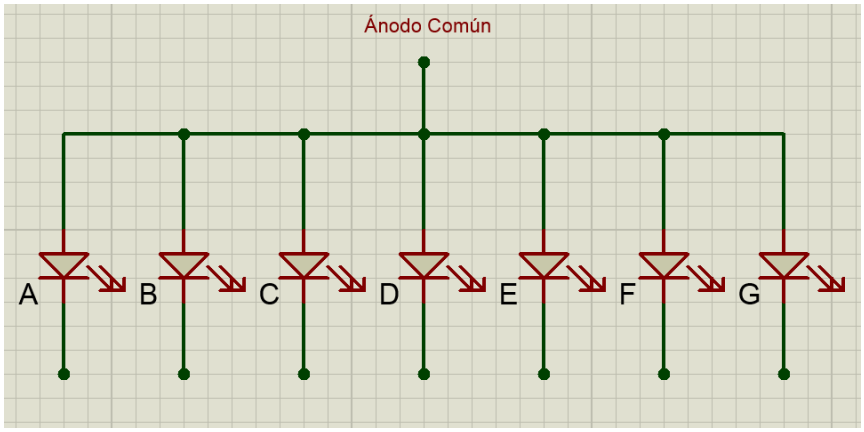
En la siguiente introducción veremos como es un convertidor BCD a 7 segmentos, BCD (Decimal Codificado en Binario) es un código que representa valores decimales en formato binario, para ello forma grupos de 4 bits para representar cada valor del 0 al 9. El 9 es el valor máximo que se puede representar en un dígito decimal, si recordamos los números binarios el 9 es un 1001_2, requiere 4 bits, es por 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.

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, entonces 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

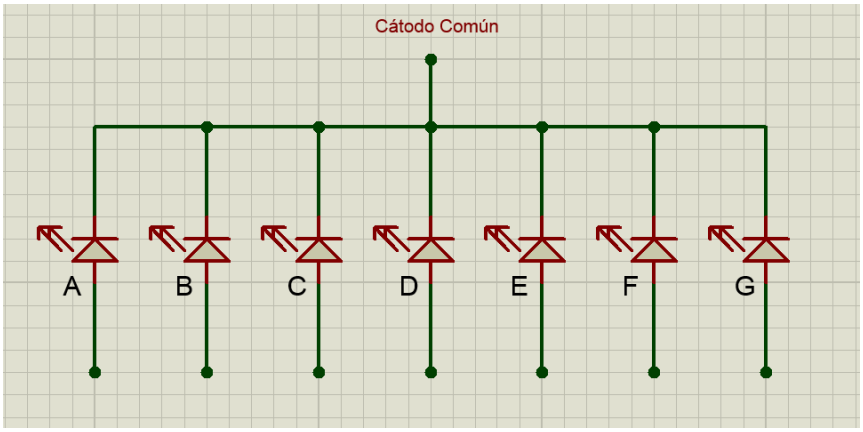


Todos los ánodos de los LED's están juntos en un punto común (Display de ánodo común)

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	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

Tabla 1 Cátodo

- Display de cátodo común:



Todos los cátodos de los LED's están juntos en un punto común (Display de cátodo común)

Número Display	Combinaciones								Valor Hexadecimal
	.	g	f	e	d	c	b	a	
0	1	1	0	0	0	0	0	0	0xC0
1	1	1	1	1	1	0	0	1	0xF9
2	1	0	1	0	0	1	0	0	0xA4
3	1	0	1	1	0	0	0	0	0xB0
4	1	0	0	1	1	0	0	1	0x99
5	1	0	0	1	0	0	1	0	0x92
6	1	0	0	0	0	0	1	0	0x82
7	1	1	1	1	1	0	0	0	0xF8
8	1	0	0	0	0	0	0	0	0x80
9	1	0	0	1	0	0	0	0	0x90

Tabla 2 Ánodo

Aunque los valores en BCD van del 0 al 9, con 4 bits se pueden lograr valores del 0 al 15. En los casos donde el valor sea mayor a 9, el decodificador mostrará en el display los símbolos arriba descritos. En el caso de tener un 15 (los 4 bits de entrada en ALTO), el display se apagará.

Códigos

1. Código de la configuración de los periféricos

```
1.  // I/O Registers definitions
2.  #include <mega8535.h>
3.  unsigned char variable;
4.  unsigned char var;
5.  const char tabCatodo [10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7c,0x07,0x7f,0x6f};
6.  const char tabAnodo [10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xf8,0x80,0x90};
7.  // Declare your global variables here
8.  void main(void)
9.  {
10. // Declare your local variables here
11. // Input/Output Ports initialization
12. // Port A initialization
13. // Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out
14. DDRA=(1<<DDA7) | (1<<DDA6) | (1<<DDA5) | (1<<DDA4) | (1<<DDA3) | (1<<DDA2) | (1<<DDA1) |
    (1<<DDA0);
15. // State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
16. PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) | (0<<PORTA3) | (0<<PORTA2) |
    (0<<PORTA1) | (0<<PORTA0);
17. // Port B initialization
18. // Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out
19. DDRB=(1<<DDB7) | (1<<DDB6) | (1<<DDB5) | (1<<DDB4) | (1<<DDB3) | (1<<DDB2) | (1<<DDB1) |
    (1<<DDB0);
20. // State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
21. PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) | (0<<PORTB2) |
    (0<<PORTB1) | (0<<PORTB0);
22. // Port C initialization
23. // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
24. DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2) | (0<<DDC1) |
    (0<<DDC0);
25. // State: Bit7=P Bit6=P Bit5=P Bit4=P Bit3=P Bit2=P Bit1=P Bit0=P
26. PORTC=(1<<PORTC7) | (1<<PORTC6) | (1<<PORTC5) | (1<<PORTC4) | (1<<PORTC3) | (1<<PORTC2) |
    (1<<PORTC1) | (1<<PORTC0);
27. // Port D initialization
28. // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
29. DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) | (0<<DDD1) |
    (0<<DDD0);
30. // State: Bit7=P Bit6=P Bit5=P Bit4=P Bit3=P Bit2=P Bit1=P Bit0=P
31. PORTD=(1<<PORTD7) | (1<<PORTD6) | (1<<PORTD5) | (1<<PORTD4) | (1<<PORTD3) | (1<<PORTD2) |
    (1<<PORTD1) | (1<<PORTD0);
32.
```

```
33. // Timer/Counter 0 initialization
34. // Clock source: System Clock
35. // Clock value: Timer 0 Stopped
36. // Mode: Normal top=0xFF
37. // OC0 output: Disconnected
38. TCCR0=(0<<WGM00) | (0<<COM01) | (0<<COM00) | (0<<WGM01) | (0<<CS02) | (0<<CS01) | (0<<CS00);
39. TCNT0=0x00;
40. OCR0=0x00;
41. // Timer/Counter 1 initialization
42. // Clock source: System Clock
43. // Clock value: Timer1 Stopped
44. // Mode: Normal top=0xFFFF
45. // OC1A output: Disconnected
46. // OC1B output: Disconnected
47. // Noise Canceler: Off
48. // Input Capture on Falling Edge
49. // Timer1 Overflow Interrupt: Off
50. // Input Capture Interrupt: Off
51. // Compare A Match Interrupt: Off
52. // Compare B Match Interrupt: Off
53. TCCR1A=(0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) | (0<<COM1B0) | (0<<WGM11) | (0<<WGM10);
54. TCCR1B=(0<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (0<<WGM12) | (0<<CS12) | (0<<CS11) | (0<<CS10);
55. TCNT1H=0x00;
56. TCNT1L=0x00;
57. ICR1H=0x00;
58. ICR1L=0x00;
59. OCR1AH=0x00;
60. OCR1AL=0x00;
61. OCR1BH=0x00;
62. OCR1BL=0x00;
63. // Timer/Counter 2 initialization
64. // Clock source: System Clock
65. // Clock value: Timer2 Stopped
66. // Mode: Normal top=0xFF
67. // OC2 output: Disconnected
68. ASSR=0<<AS2;
69. TCCR2=(0<<WGM20) | (0<<COM21) | (0<<COM20) | (0<<WGM21) | (0<<CS22) | (0<<CS21) | (0<<CS20);
70. TCNT2=0x00;
71. OCR2=0x00;
72. // Timer(s)/Counter(s) Interrupt(s) initialization
73. TIMSK=(0<<OCIE2) | (0<<TOIE2) | (0<<TICIE1) | (0<<OCIE1A) | (0<<OCIE1B) | (0<<TOIE1) | (0<<OCIE0)
    | (0<<TOIE0);
74. // External Interrupt(s) initialization
75. // INT0: Off
76. // INT1: Off
```

```
77. // INT2: Off
78. MCUCR=(0<<ISC11) | (0<<ISC10) | (0<<ISC01) | (0<<ISC00);
79. MCUCSR=(0<<ISC2);
80. // USART initialization
81. // USART disabled
82. UCSRB=(0<<RXCIE) | (0<<TXCIE) | (0<<UDRIE) | (0<<RXEN) | (0<<TXEN) | (0<<UCSZ2) | (0<<RXB8) |
    (0<<TXB8);
83. // Analog Comparator initialization
84. // Analog Comparator: Off
85. // The Analog Comparator's positive input is
86. // connected to the AIN0 pin
87. // The Analog Comparator's negative input is
88. // connected to the AIN1 pin
89. ACSR=(1<<ACD) | (0<<ACBG) | (0<<ACO) | (0<<ACI) | (0<<ACIE) | (0<<ACIC) | (0<<ACIS1) | (0<<ACIS0);
90. SFIOR=(0<<ACME);
91. // ADC initialization
92. // ADC disabled
93. ADCSRA=(0<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (0<<ADPS2) | (0<<ADPS1) |
    (0<<ADPS0);
94. // SPI initialization
95. // SPI disabled
96. SPCR=(0<<SPIE) | (0<<SPE) | (0<<DORD) | (0<<MSTR) | (0<<CPOL) | (0<<CPHA) | (0<<SPR1) | (0<<SPR0);
97. // TWI initialization
98. // TWI disabled
99. TWCR=(0<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | (0<<TWEN) | (0<<TWIE);
```

2. Código del programa principal en C

```
while (1)
      {
        variable=PIND&0x0f; //Enmascaramos los 4 bits LSB
        if (variable<10)
              PORTB=tabCatodo[variable];
        if (variable>=10) //Si es 10 o mayor da E de error
              PORTB=0x79;

        var=PINC&0x0f; //Enmascaramos los 4 bits LSB
        if (var<10)
              PORTA = tabAnodo[var];
        if (var>=10) // Si es 10 o mayor da E de error
              PORTA=0x86;
        // Place your code here
      };
```

Circuitos

1. Circuito simulado en Proteus

En la figura 1 se muestra ambos displays recibiendo un uno por parte de sus dip switch y como se puede observar en el caso del ánodo se optó por un diseño diferente al propuesto en el formato de la práctica otorgado por el profesor, puesto que se sugería el tener los valores del correspondiente dip switch invertidos.

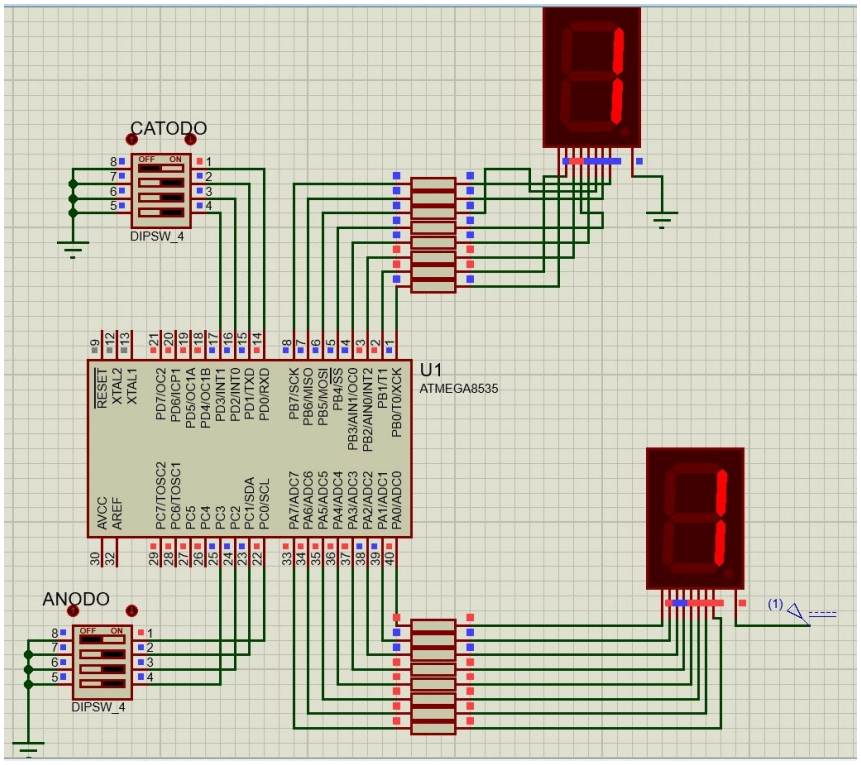


Figura. 1 Displays en 1

De igual forma en la figura anterior se puede observar el funcionamiento de los displays puesto que el cátodo es controlado por 1's lógicos como se denota con los puntos rojos que encienden los dos segmentos del display, mientras que el ánodo requiere de 0's lógicos o también llamado tierra digital y esto nuevamente se aprecia en los puntos azules que encienden los dos segmentos que forman el 1 en el display.

Es importante mencionar que la forma de diferenciar los display no solo radica en los colores que muestra el simulador, sino en el hecho de que un display 7 segmentos cátodo común debe ser conectado a tierra, mientras que un display 7 segmentos ánodo común debe recibir una alimentación de 5 volts.

En la figura 2 y figura 3, se muestra como al exceder el valor de 9 el display muestra una "E" de error, en el caso de la figura 2 se tiene un 10 y en la figura 3 se tiene un 12.

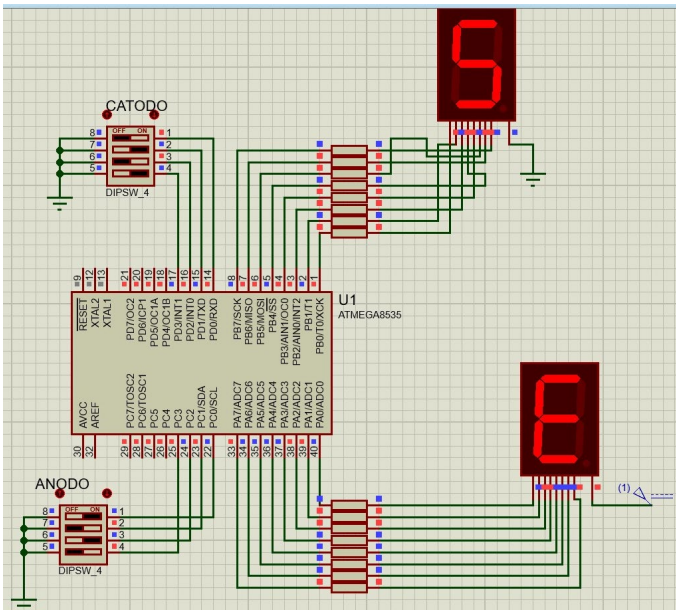


Figura. 2 Ejemplo de error en ánodo

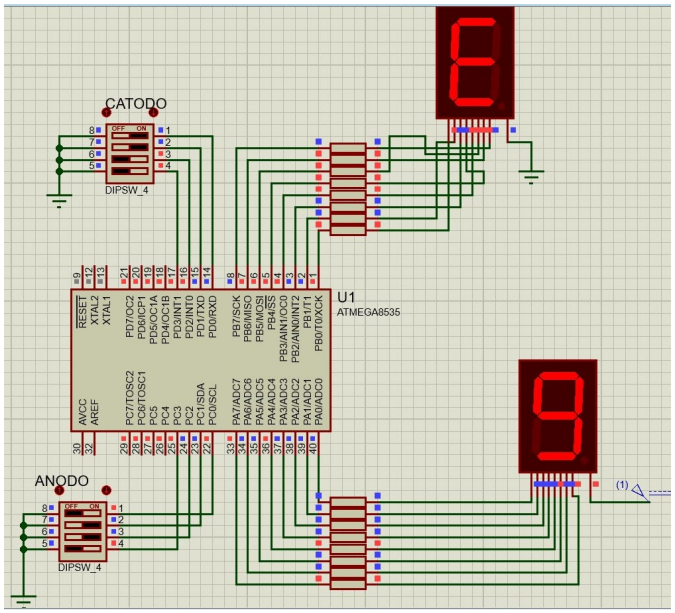


Figura. 3 Ejemplo de error en cátodo

1. Circuito armado en la Proto

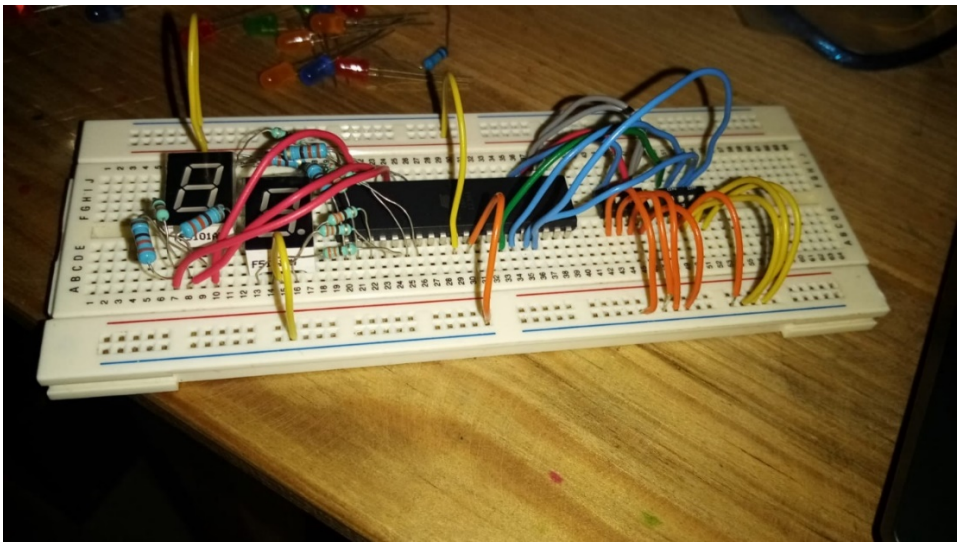


Figura. 3 Circuito Ánodo y cátodo

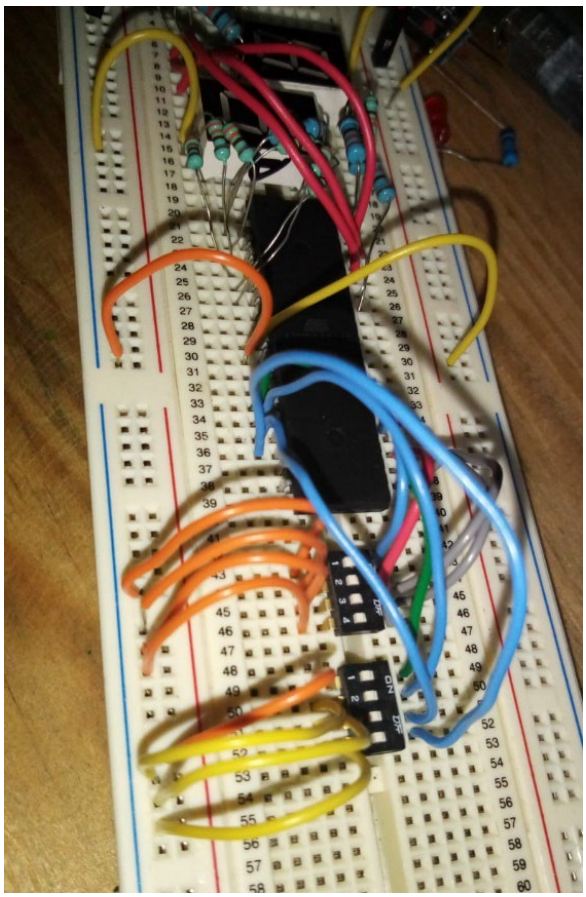


Figura. 4 Controlador del display

Conclusiones

En general el realizar la presente práctica fue sencillo puesto que se reutilizan los conocimientos adquiridos en las dos prácticas anteriores con respecto al manejo de puertos y bits, sin embargo, de forma previa a la realización del código y circuito en simulación fue necesario realizar un repaso con respecto a las diferencias y cualidades de un display cátodo común y ánodo común, resultando en la formación de tablas que posteriormente se ingresan a la implementación en forma de arreglo, durante el repaso también se adquirió conciencia de la alimentación que requerían los displays para funcionar ya que esto provocó una ligera confusión en el momento de armar el circuito y realizar las pruebas.

Por otro lado, al realizar la implementación se consideró la opción de aplicar un not para invertir los valores de uno de los arreglos y de está forma evitar el uso de dos arreglos dentro del programa (aprovechando el manejo de bits), pero esto no fue posible debido a un erro marcado en el código del cual no fue posible consultar con el profesor, debido a la situación actual.

Arruti Sánchez Alondra

En esta práctica lo que se complico en mi caso fue realizar el circuito ya que tuve que estar revisando el datasheet del cátodo y del ánodo del display ya que no me acordaba como se tenia que conectar sin embargo una vez teniendo en cuanta eso, fue más simple poder armarlo, aunque poner las resistencias en el proto fue muy difícil, ya que las dimensiones de la proto no eran muy grandes, sin embargo, se logró conectar el circuito de manera correcta.

Igual al momento de probarlo nos dimos con otro problema que el circuito no marcaba correcto ambos display como se nos había puesto en la práctica, sin embargo, rápido se soluciono porque esta mal conectado el switch de la parte del display del ánodo, pero una vez conectado bien al controlador se logró solucionar.

Carrillo Soto Cristian Eduardo

Bibliografía

[1] Decodificadores de BCD a 7 segmentos. (2020, 21 abril). 7Robot - Mobile Education and Engineering. Recuperado 5 de octubre de 2022, de https://wp.7robot.net/decodificadores-de-bcd-a-7-segmentos/