INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Introducción a los microcontroladores

3CM16

Practica 02

"Contador Ascendente y descendente"

Equipo 7

Integrantes:

- Bocanegra Heziquio Yestlanezi
- Dominguez Durán Alan Axel
- Hernandez Mendez Oliver Manuel
- Martinez Cruz José Antonio

Profesor: Aguilar Sanchez Fernando

Índice

Índice de imágenes 2
Introducción 3
Contador 3
Contador ascendente 3
Contador descendente 3
Objetivo 4
Material y equipo empleado 4
Desarrollo 5
Programa en CodeVisionAVR 5
Simulación 10
Proteus 10
Estructura del programa 11
Contador ascendente 11
Contador descendente 15
Conclusiones 19
Bocanegra Heziquio Yestlanezi 19
Dominguez Durán Alan Axel 19
Hernandez Mendez Oliver Manuel 19
Martinez Cruz José Antonio 19
Referencias 20

Índice de imágenes

Imagen 1 Crear un nuevo proyecto en CodeVision 5
Imagen 2 Clock 6
Imagen 3 Puerto B 6
Imagen 4 Puerto D 7
Imagen 5 Guardar Proyecto 7
Imagen 6 Guardar Proyecto con el mismo nombre 3 veces 8
Imagen 7 Circuito armado 8
Imagen 8 contador ascendente 9
Imagen 9 Contador descendente 9
Imagen 10 Circuito simulado con Proteus 10

Introducción

Contador

Un contador es un dispositivo electrónico o mecánico que se utiliza para contar eventos o pulsos. Los contadores pueden contar eventos en una dirección, ya sea ascendente o descendente, dependiendo de su diseño y configuración [1].

Contador ascendente

es un dispositivo que cuenta los pulsos en orden creciente, de modo que el valor de salida del contador aumenta con cada pulso de entrada. Por ejemplo, si el contador se configura para contar hasta 9, después de recibir un pulso de entrada, el contador mostrará el valor "1". Después del segundo pulso, mostrará el valor "2", y así sucesivamente hasta llegar a "9" [2].

Un contador ascendente comienza en un valor determinado y cuenta en incrementos sucesivos hasta alcanzar un valor máximo. Por ejemplo, un contador binario de 3 bits comienza en 000 y cuenta en incrementos de 1 para llegar a 111 (el valor máximo). Luego, el contador vuelve a comenzar desde 000 [2].

Contador descendente

un contador descendente cuenta los pulsos en orden decreciente, de modo que el valor de salida del contador disminuye con cada pulso de entrada. Por ejemplo, si el contador se configura para contar desde 9 hasta 0, después de recibir un pulso de entrada, el contador mostrará el valor "9". Después del segundo pulso, mostrará el valor "8", y así sucesivamente hasta llegar a "0" [3].

un contador descendente comienza en un valor máximo y cuenta en decrementos sucesivos hasta alcanzar un valor mínimo. Por ejemplo, un contador binario de 3 bits comienza en 111 y cuenta en decrementos de 1 para llegar a 000 (el valor mínimo). Luego, el contador vuelve a comenzar desde 111 [3].

Objetivo

Al término de la sesión, los integrantes del equipo contaran con la habilidad de realizar un contador descendente y ascendente utilizando funciones de retardo.

Material y equipo empleado

- ⇒ CodeVision AVR
- ⇒ Proteus
- ⇒ AVR Studio 4
- ⇒ Microcontrolador ATmega 8535
- ⇒ 16 LED's
- ⇒ 16 Resistores de 330 Ω a 1⁄4 W

Desarrollo

Diseñe un contador binario ascendente en el Puerto B (0-255), y un contador descendente en el Puerto D (255-0). Use un retardo de un segundo para visualizar la información.

Programa en CodeVisionAVR

Realizamos un nuevo programa para nuestra practica 02 como se muestra en la imagen 1.

⇒ File

⇒ New

⇒ Project

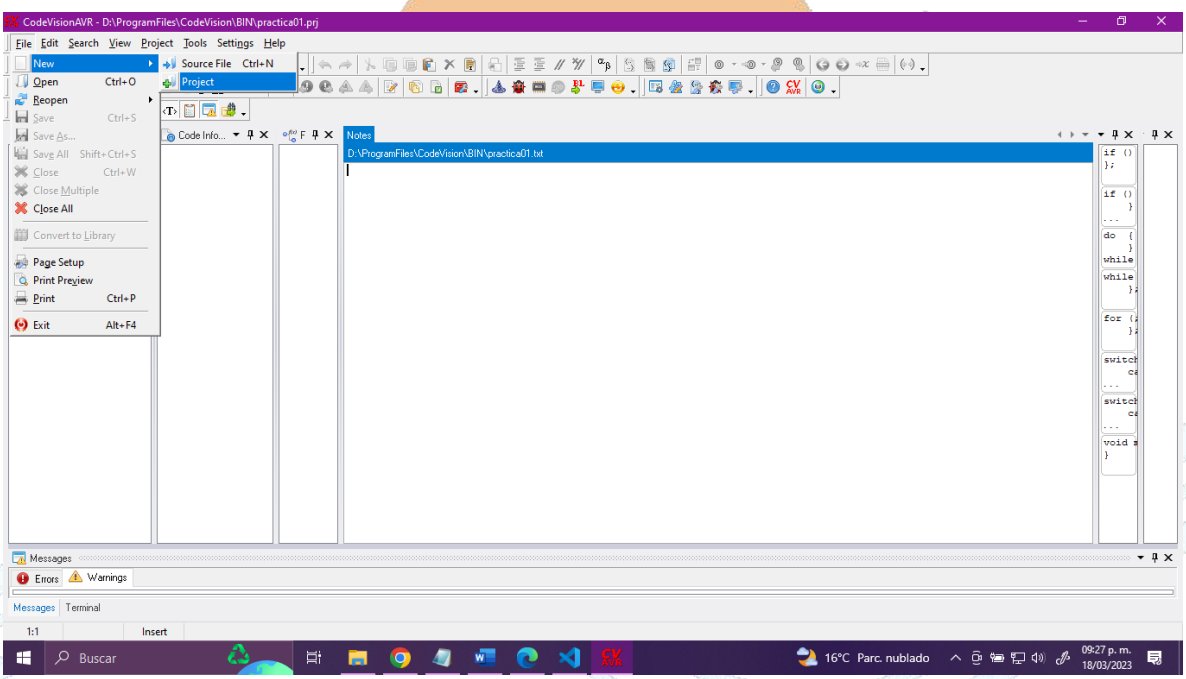

Imagen 1 Crear un nuevo proyecto en CodeVision

Recuerda siempre dejar reloj en 1Mhz como se muestra en la imagen 2.

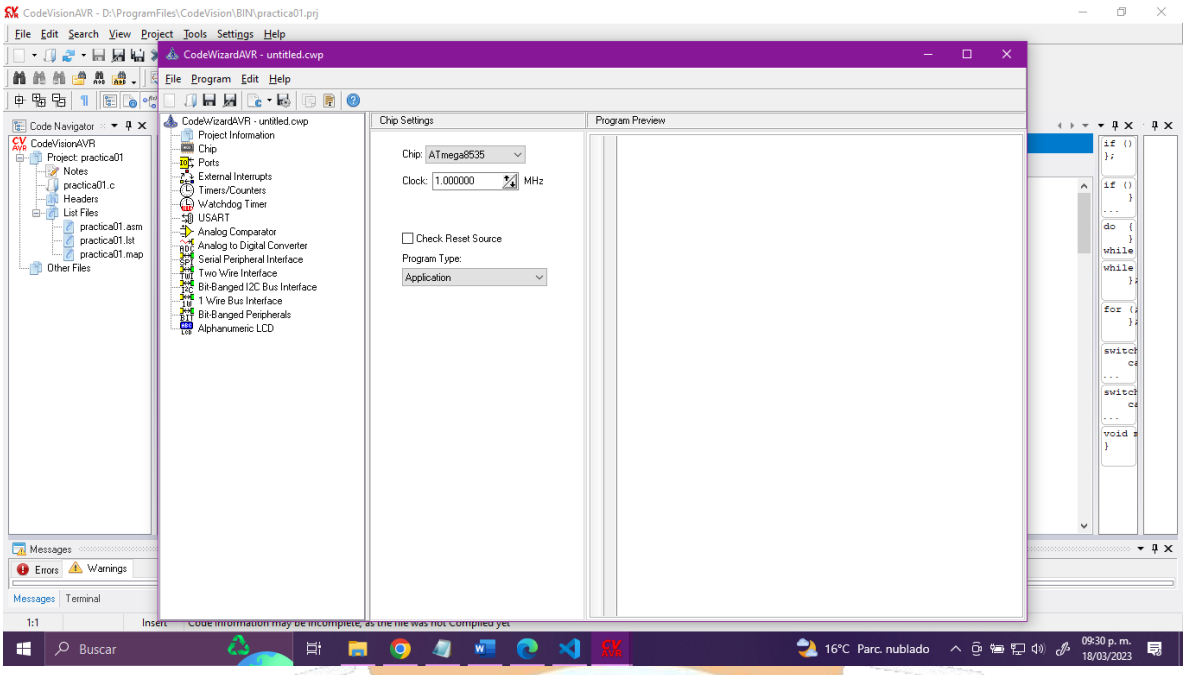

Imagen 2 Clock

en la parte ports cambiamos la configuración de los puertos B y D como en la practica 01 como se muestra en la imagen 3 y 4.

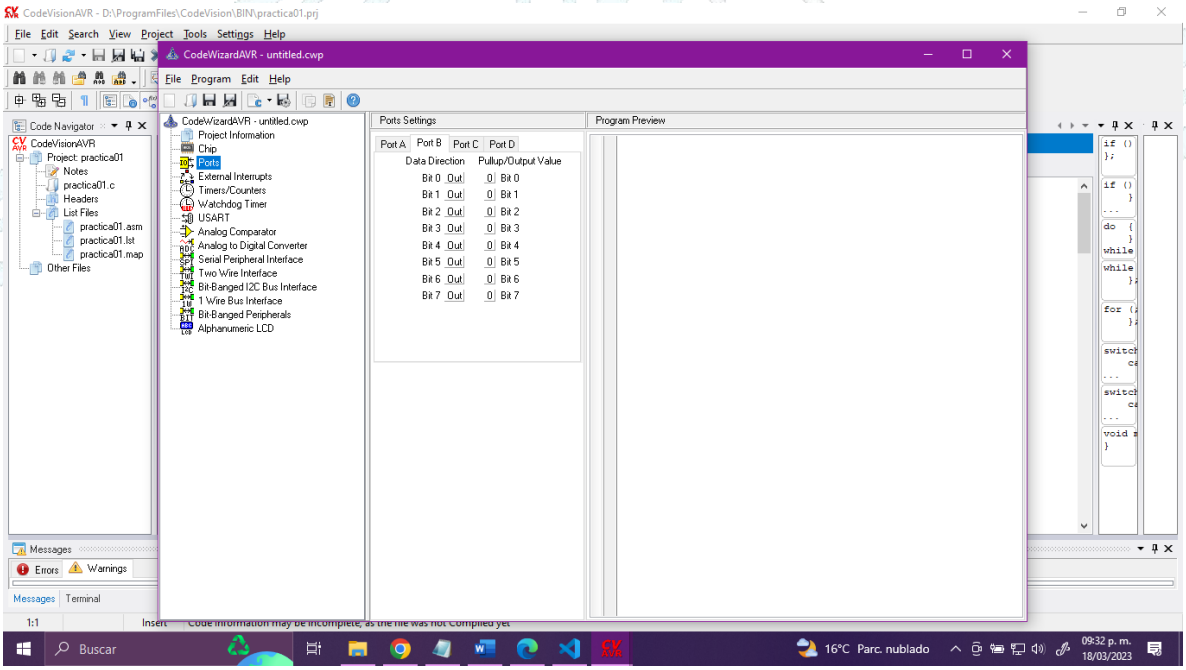

Imagen 3 Puerto B

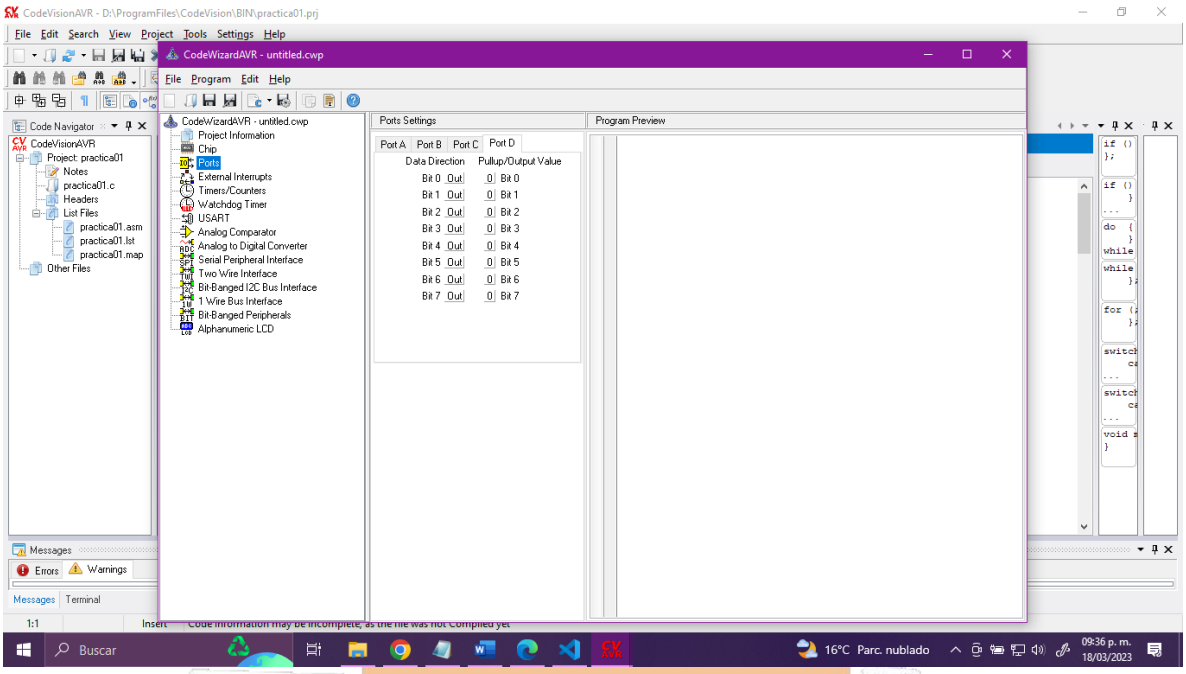

Imagen 4 Puerto D

por ultimo guardamos el proyecto 3 veces con el mismo nombre como se muestra en la imagen 5 y 6.

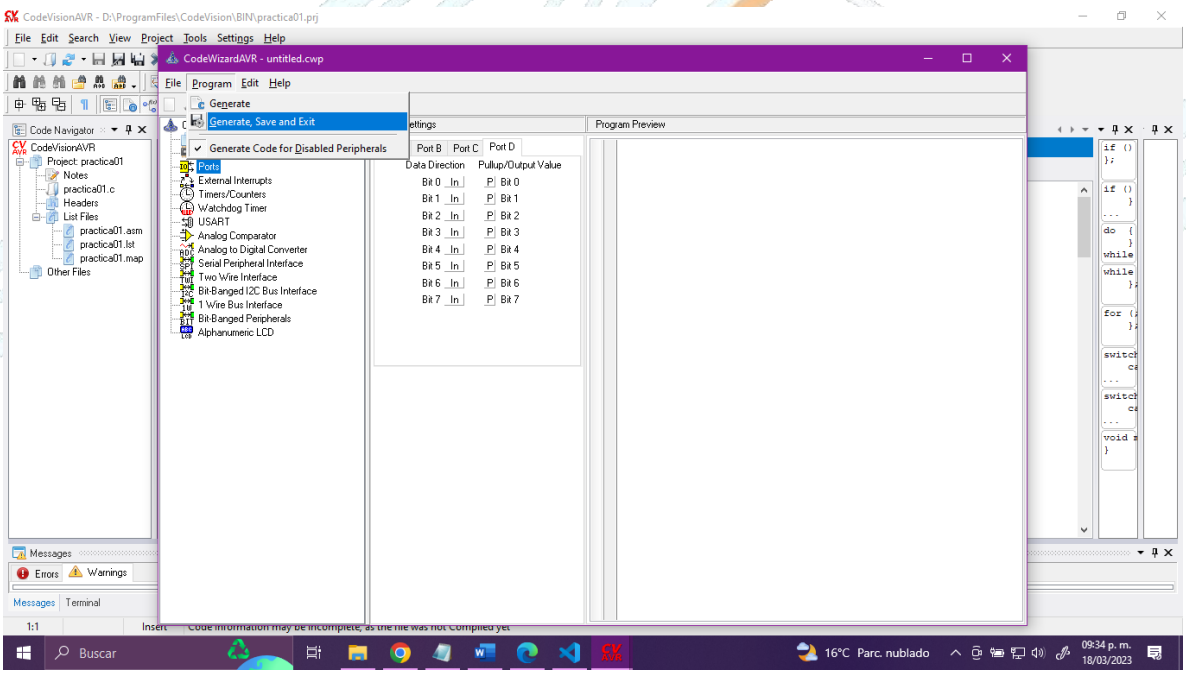

Imagen 5 Guardar Proyecto

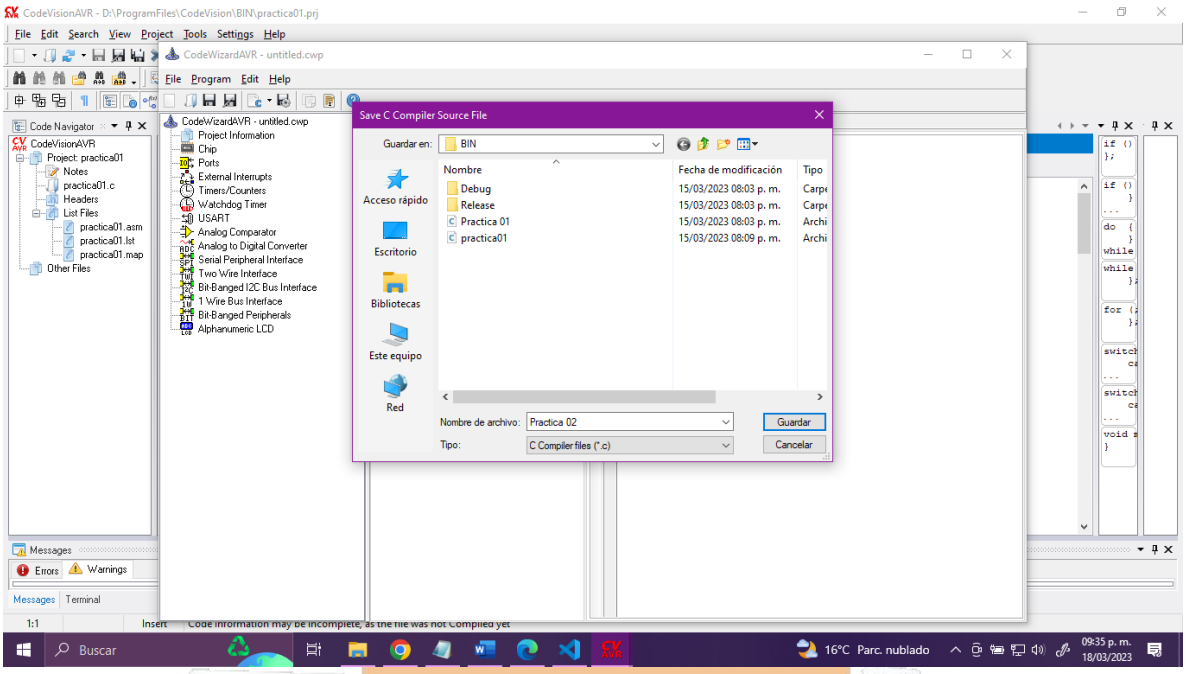

Imagen 6 Guardar Proyecto con el mismo nombre 3 veces

a continuación, realizamos el código para realizar un contador ascendente y uno descendente, en nuestro caso como no contábamos con los LEDS suficientes para realizarlo junto, lo dividimos en dos partes, a continuación, en la imagen 7 se puede apreciar el circuito armado.

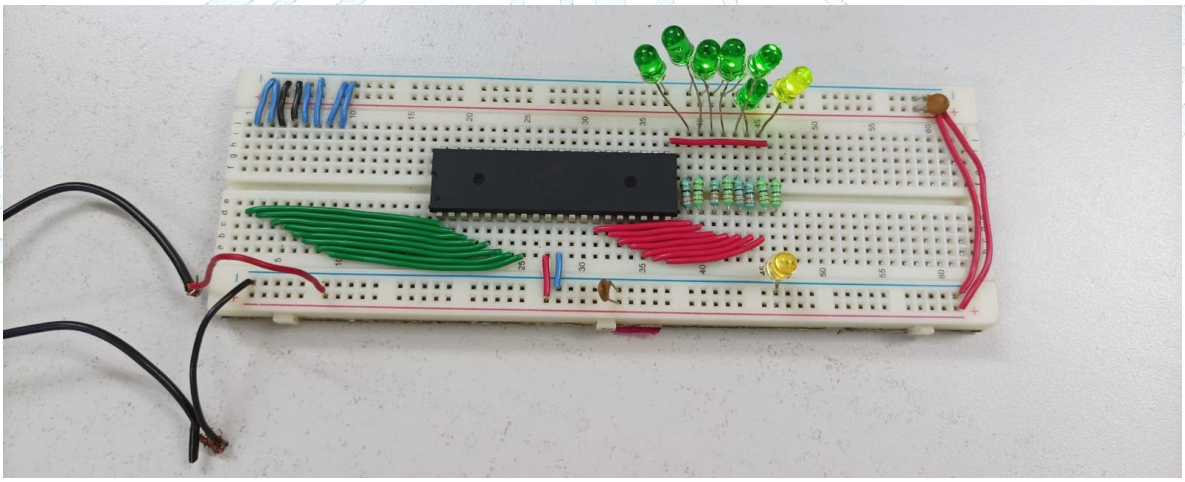

Imagen 7 Circuito armado

como podemos observar, solo contamos con 8 LEDS por lo cual dividimos el código en dos, uno donde cuenta ascendente y otro descendente utilizando el mismo circuito, utilizando únicamente el puerto D.

En la imagen 8 podemos apreciar el circuito funcionando con la operación ascendente.

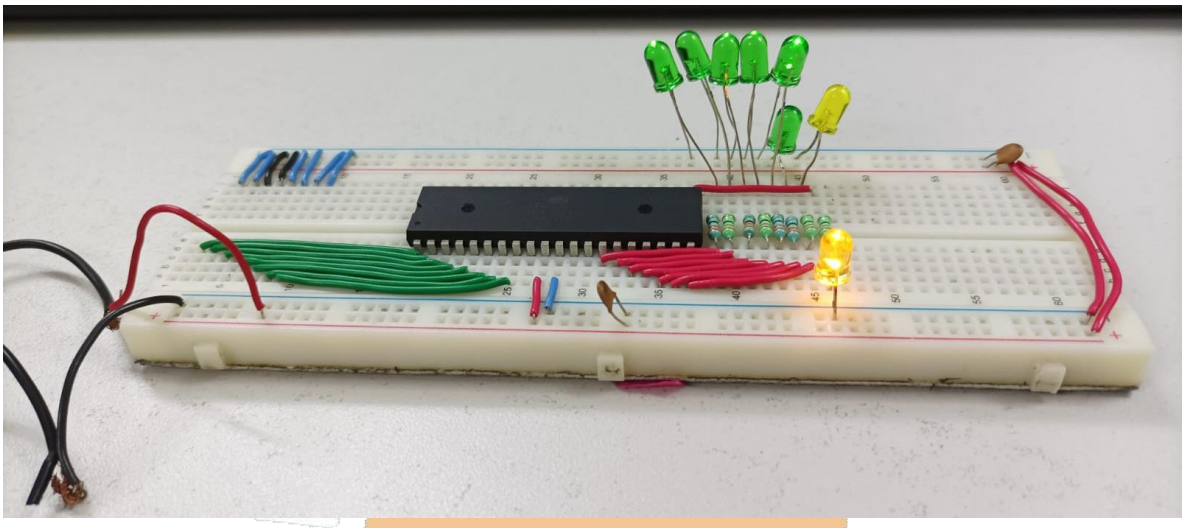

Imagen 8 contador ascendente

en la imagen 9 podemos apreciar el mismo circuito realizando la operación descendente, en la imagen no es visible ya que se utiliza el mismo circuito para las dos operaciones, pero al momento de hacer el código, cambiamos la función para que este realice el contador descendente, se puede notar el cambio en el apartado "estructura del programa – contador descendente".

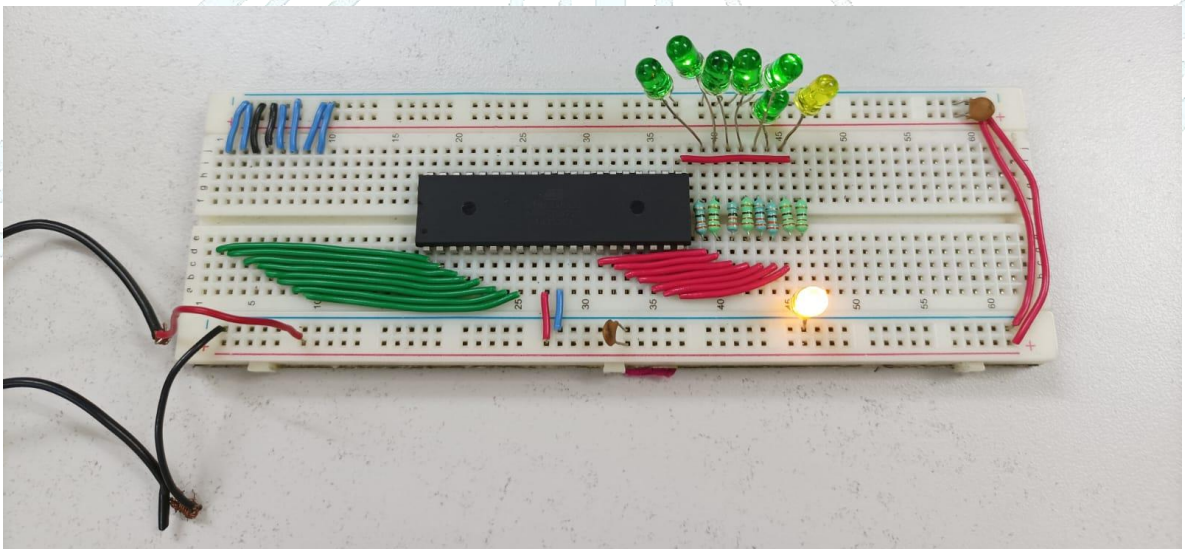

Imagen 9 Contador descendente

Simulación

Proteus

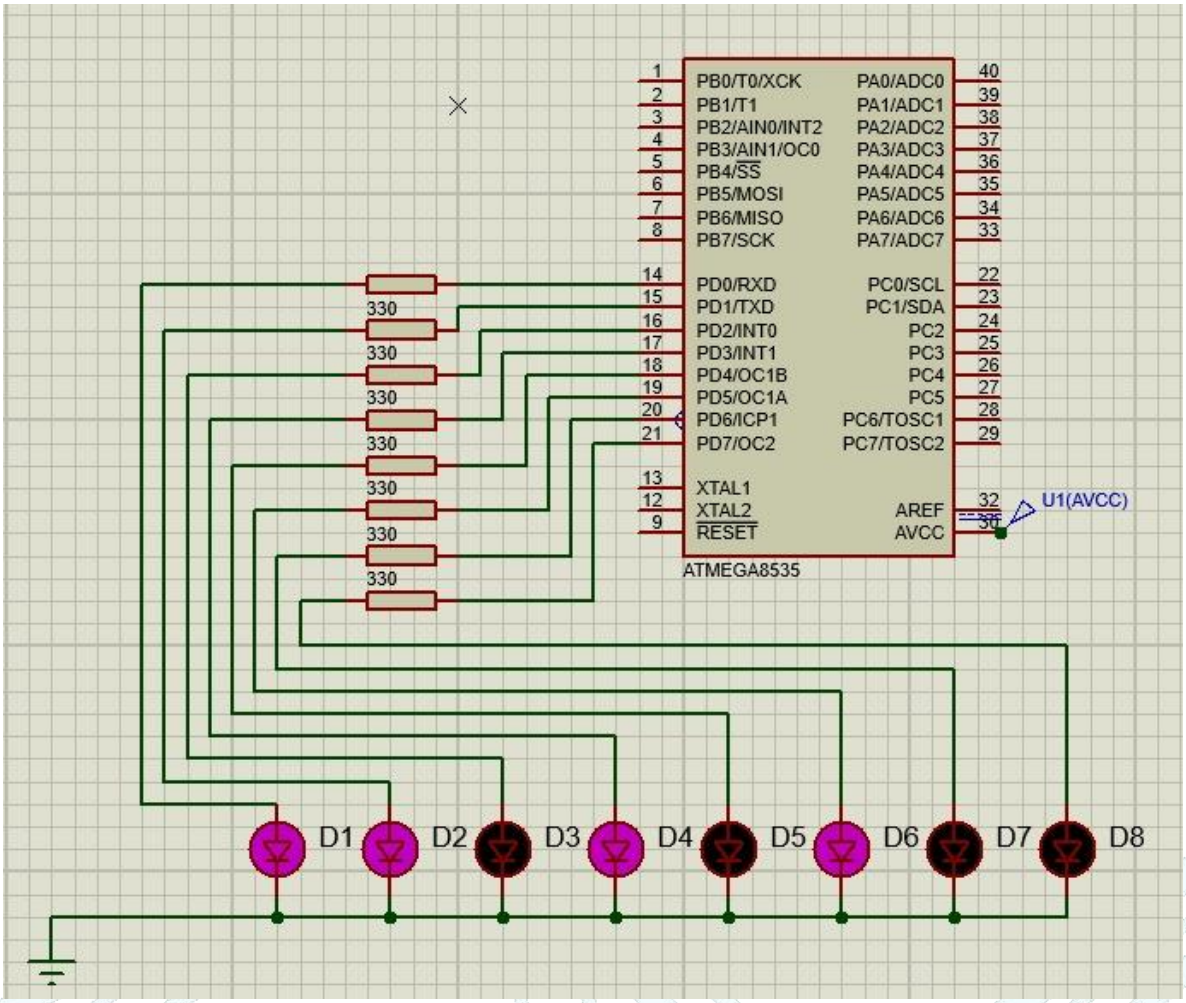

Imagen 10 Circuito simulado con Proteus

Como podemos observar en la imagen 10 la simulación en proteus fue realizada tanto para el contador ascendente y descendente, al igual que en las imágenes 8 y 9 del circuito realizado en laboratorio, el cambio no es notorio, este cambio donde cuenta en forma ascendente y descendente solo se puede observar en el código en el apartado "estructura del programa".

Utilizamos la misma estructura de la practica 01 del circuito para realizar la practica 02, en este caso solo quitamos el del switch.

Estructura del programa

Contador ascendente

```
/*****************************************************
This program was created by the CodeWizardAVR V3.51
Automatic Program Generator
© Copyright 1998-2023 Pavel Haiduc, HP InfoTech S.R.L.
http://www.hpinfotech.ro

Project :
Version :
Date    : 18/03/2023
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>

// 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=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
```

```
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) |
(0<<DDB1) | (0<<DDB0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
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=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=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out
Bit0=Out
DDRD=(1<<DDD7) | (1<<DDD6) | (1<<DDD5) | (1<<DDD4) | (1<<DDD3) | (1<<DDD2) |
(1<<DDD1) | (1<<DDD0);
// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
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);

PORTD =0xFF;
while (1)
      {
      // Place your code here
      PORTD++;
      delay_ms(250);

      }
}
```

Contador descendente

```
/*****************************************************
This program was created by the CodeWizardAVR V3.51
Automatic Program Generator
© Copyright 1998-2023 Pavel Haiduc, HP InfoTech S.R.L.
http://www.hpinfotech.ro

Project :
Version :
Date    : 18/03/2023
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>

// 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=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
```

```
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) |
(0<<DDB1) | (0<<DDB0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
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=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=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out
Bit0=Out
DDRD=(1<<DDD7) | (1<<DDD6) | (1<<DDD5) | (1<<DDD4) | (1<<DDD3) | (1<<DDD2) |
(1<<DDD1) | (1<<DDD0);
// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
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);

PORTD =0x00;
while (1)
      {
      // Place your code here
      PORTD--;
      delay_ms(250);

      }
}
```

Conclusiones

Bocanegra Heziquio Yestlanezi

Para la practica 02 utilizamos la misma estructura del circuito que ya se tenia armado de la practica 01, en este caso como no contábamos con 16 LEDS decidimos dividir el código en dos partes como ya se mencionó, haciendo que primero realizara el contador ascendente y después el contador descendente, teniendo así que programar dos veces el ATmega8535.

Dominguez Durán Alan Axel

Al finalizar esta práctica, se puede notar el contraste que hay entre armar un circuito secuencial y utilizar el microcontrolador, ya que es mucho más sencillo de implementar un contador en el micro, aparte de que detectar fallas o errores toma menos tiempo que en un circuito secuencial convencional. Poco a poco podemos notar el poder y las facilidades que nos da el microcontrolador para los diferentes casos de uso de cada práctica.

Hernandez Mendez Oliver Manuel

Gracias a esta práctica y a la elaboración previa de circuitos que cumplían está tarea. Pudimos observar que con algunas líneas de código podemos realizar implementaciones de contadores ascendentes y descendientes de forma más sencilla y sintetizada. El uso de un Microcontrolador nos permite enfocarnos en tareas más complejas, ya que los conceptos básicos son relativamente simples de implementar.

Martinez Cruz José Antonio

Retomando el circuito de la anterior practica logramos demostrar el funcionamiento de un contador ascendente y descendente de forma binaria, mediante el uso de diodos leds. Este proceso se pudo apreciar de manera clara, sin embargo, se le agrego un valor inicial de cero para mostrar una transición correcta desde su inicio a final.

Referencias

[1] S. Brown and Z. Vranesic, "Contadores", en Fundamentals of Digital Logic with VHDL Design, 3rd ed., Nueva York, NY, EE. UU.: McGraw-Hill Education, 2009, pp. 301-329. DOI: 10.1007/978-1-4615-3788-4_11.

[2] J. Wakerly, "Contadores y registros", en Digital Design: Principles and Practices, 4th ed., Upper Saddle River, NJ, EE. UU.: Pearson Education, 2006, pp. 336-375. DOI: 10.1109/MAHC.1984.10028.

[3] M. T. Hossain, S. S. Haque, and M. A. Islam, "Design and Implementation of Asynchronous Up-Down Counter Circuit Using JK Flip-Flops," 2021 International Conference on Electrical, Electronics and Communication Engineering (ICEECE), Dhaka, Bangladesh, 2021, pp. 1-5. doi: 10.1109/ICEECE51825.2021.9470366.