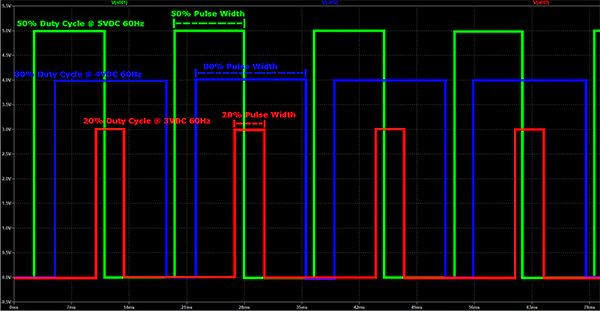
**Interpretación**

tp N°4 ENTREGABLE

1 de agosto de 20220



BLANCO Valentin Nicolas, PALADINO gABRIEL AGUSTIN

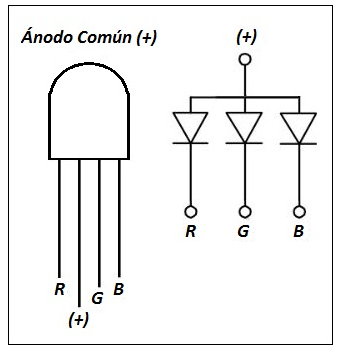
CIRCUITOS DIGITALES Y MICROCONTROLADORES

Se debería realizar un programa en c donde conectando un LED RGB al Arduino y dando a este distintas señales en cada una de sus entradas se conseguiría encenderlo de distintos colores. Dicho LED cuenta con 3 pines para su control (uno para cada color) y la conexión de los mismo con el MCU debe ser PB5 para el rojo, PB2 para verde y PB1 para azul.

Además de este periférico se debe utilizar un potenciómetro para controlar la intensidad de cada color del LED y una terminal para definir a cuál color se le va a realizar dicha modificación. Para hacer funcionar la terminal se utiliza el periférico UART que trae incorporado el MCU y para leer los valores del potenciómetro se hace uso de un conversor analógico digital (ADC3) que también tiene incorporado el MCU.

**Resolución del Problema:**

**Corriente máxima para cada LED**

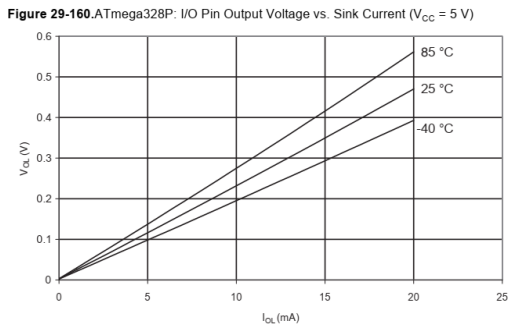


Ya que el led es de tipo ánodo común (como se observa en la imagen superior) la intensidad de los colores serán activos en bajo por lo que la manera en la que se calcula la corriente máxima es la siguiente:



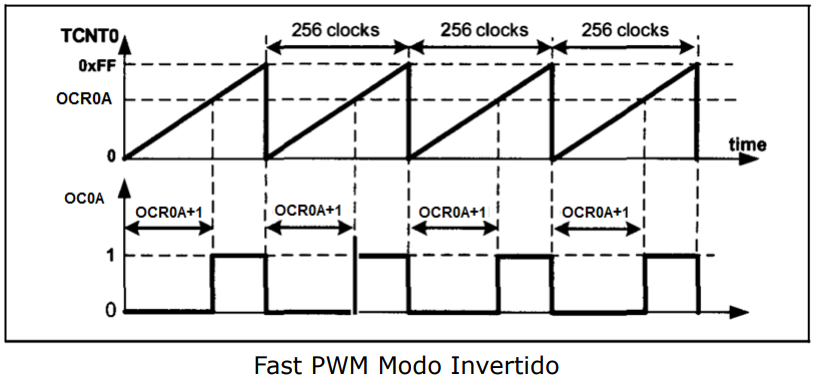
A partir de esta fórmula y siguiendo la figura 29-160 de la hoja de datos donde se observa que a temperaturas normales de operación el Vol maximo sera de aproximadamente 0.47 V y siendo Rled 220 Ω en los 3 casos se consiguen las siguientes corrientes maximas para cada color:

|  |  |  |
| --- | --- | --- |
| Color | Voltaje de color | Corriente maxima |
| Rojo | 2.049V | 11.27mA |
| Verde | 2.81V | 7.81mA |
| Azul | 2.871V | 7.54mA |

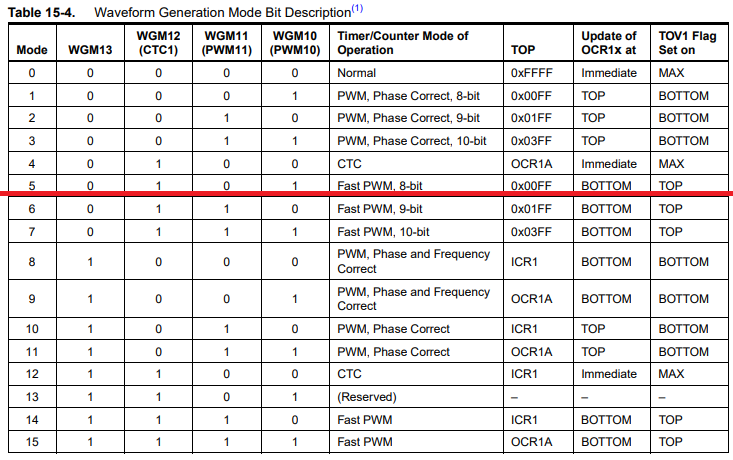


**Generación de señales**

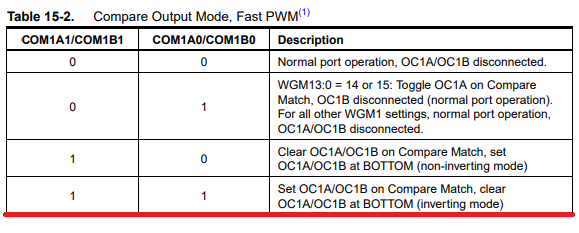
Para generar distintos ciclos de trabajo de tal manera que varíen los colores del led se utilizó la técnica PWM (Pulse Width Modulation) lo que permite que se pueda obtener una señal analógica a partir de una señal digital. Como los colores son activos en bajo para solucionar el problema escogimos PWM invertido de modo tal que al poner el registro comparador del timer en 0 el LED este apagado y en 255 este a la máxima intensidad. A continuación se muestra un gráfico de cómo se generaría una señal PWM de 8 bits invertida con timer0.



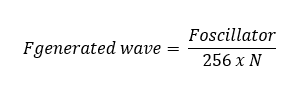
**Generación de PWM para azul y verde (Timer1):** Como los colores azul y verde corresponden a los puertos PB1 y PB2 respectivamente se puede utilizar la función PWM del timer1 ya que la señal generada por el mismo se refleja en estos puertos. El enunciado solicita una resolución de 8 bits por lo que se puede configurar el timer en modo 5 (Fast PWM de 8 bits). Para configurarlo de dicho modo seguimos la tabla de configuración del timer1:



Luego de seleccionar dicho modo, siguiendo la tabla inferior se configuro el mismo para que la señal sea dada de manera invertida (COM1A1 y COM1A0 en 1 para el color azul, COM1A0 y COM1A0 en 1 para el verde) ya que como se explico anteriormente la conexión del LED era ánodo común de modo tal que 0 indica apagado y 255 la intensidad máxima del color.

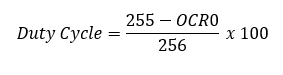


El enunciado solicita una frecuencia de 50 hz o mayor por lo que a partir de la siguiente fórmula se puede calcular la frecuencia de la señal generada por el timer1



De esta cuenta reemplazando Foscillator=16Mhz y N=1024 (siendo este el preescaler) se consiguen 61.03 Hz.

Con la siguiente formula se obtendria la intensidad del color ya que la misma representa cuanto tiempo del periodo estara en alto la señal.



Si queremos encender un color al máximo OCR0 debería valer 255 de modo tal que duty cycle sea 0 y la señal se encuentre siempre en bajo.

**Generación de PWM para rojo (por software):** Para este color se debe utilizar la salida PB5 del MCU y a diferencia de los otros colores esta salida no cuenta con PWM, por lo que la señal se debe generar por software de manera bloqueante.

Para realizar la temporización de la señal se usó el timer0 en modo CTC. Por simplicidad como periodo máximo fue utilizado el contador máximo es decir OCR0A=255 resultando una señal de periodo 16.384ms y frecuencia 61.03Hz como las generadas por timer1 en modo PWM 8 bits.

Para saber cuánto tiempo se debería dejar en bajo y en alto la señal (según la intensidad deseada) se utilizó la regla de tres simple obteniendo las siguientes formulas:



Donde T hace referencia a el periodo total de la señal y rojo a la intensidad del color.

Y para conseguir los valores de OCRA0 también se usó regla de tres simple y se llegó a las siguientes formulas:

Señal OCRA0 en bajo:



Reemplazando la formula de t\_bajo se obtuvo



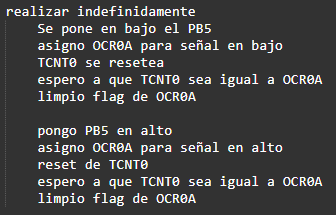
Señal OCRA0 en alto:



Reemplazando la fórmula de t\_bajo se obtuvo



De esta forma asignando los valores de OCR0A calculados y reiniciando TCNT0 en el momento indicado se obtiene la señal con intensidad deseada. Se adjunta pseudocodigo de los descripto anteriormente en la imagen inferior.



**Uso de terminal**

Para esto se creó la librería terminal.c la cual para funcionar hace uso de la librería del anterior TP (SerialPort.c). Por lo que para realizar la comunicación entre la terminal y el MCU se hace uso del periférico UART0.

Para enviar el menú de opciones a la consola utilizamos del método Terminal\_imprimirMenu() que para funcionar hace uso del método SerialPort\_Send\_String(Char \*). El menú se imprimirá al iniciar el programa y luego de realizar modificaciones en los colores.

Una vez que se entra al modo de configuración de algún color se debe imprimir el valor del potenciómetro en pantalla (decidimos imprimir dicho valor cada 1 segundo). Para esto primero se debe obtener el valor del potenciómetro haciendo uso del conversor analógico digital (ADC3) por lo que se creó el método Terminal\_leerPotenciometro() que retorna la posición del potenciómetro en un numero binario de 8 bits sin signo (uint8\_t). Una vez obtenida la posición se debe imprimir la misma en la terminal por lo que se creó la función Terminal\_imprimirValor() que lo que hace es convertir el numero de 8 bits sin signo a el conjunto de caracteres ASCII correspondientes a los dígitos de dicho número. A medida que se obtienen los dígitos se espera a que el registro interno UDR de la UART este vacío con el método SerialPort\_Wait\_For\_TX\_Buffer\_Free() y una vez que sucede esto se manda el digito a la terminal haciendo uso del método SerialPort\_Send\_Data(). Si el número es de un solo digito imprimirá solo un digito, si es de dos imprimirá solo dos dígitos y si es de tres imprimirá los tres.

La transmisión de datos desde el MCU hacia la terminal lo hacemos de manera bloqueante ya que no se creyó necesario el uso de interrupciones para la transmisión debido a que no afecta a la funcionalidad y simplifica el programa. En cambio, para la lectura de los comandos ingresados se utilizaron las interrupciones como en el anterior TP para que los mismos sean detectados lo más rápido posible. Al igual que el TP anterior también se hizo uso de la estructura foreground/background para realizar las tareas correspondientes cada vez que se ingresa un nuevo comando valido.

Los comandos programados para usar la terminal fueron los siguientes:

R <Enter>: configuración del color rojo.

V <Enter>: configuración del color verde.

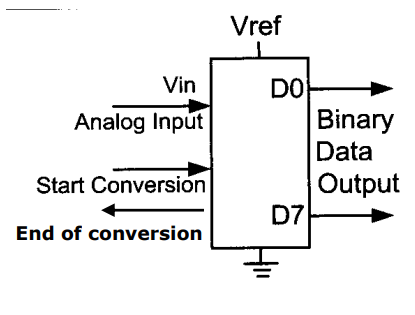
A<Enter>: configuración del color azul.

S<Enter>: asigno valor elegido por el potenciómetro en caso de encontrarse en alguno de los casos anteriores.

En caso de encontrarse en el modo R, V o A se imprime cada 1 segundo el valor del potenciómetro. Y en caso de no recibir ningún comando correcto se ignora el mismo siguiendo en el modo en el que se encuentre.

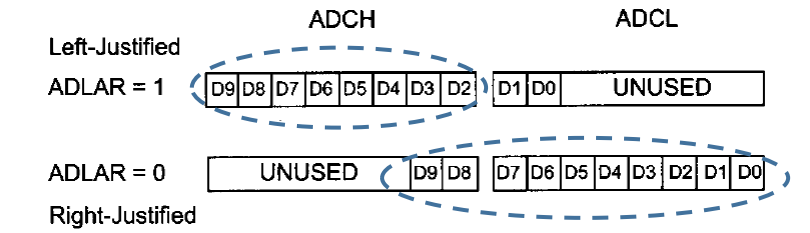
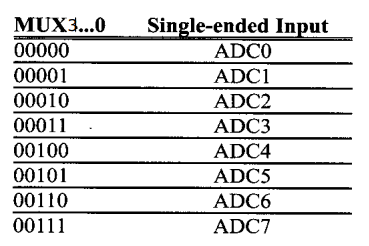
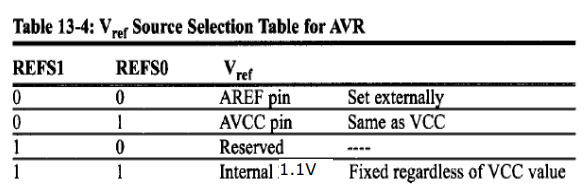
**Uso de Periferico ADC**

Para la utilización del potenciómetro se hizo uso de un conversor Analógico - Digital que nos permitió convertir magnitudes analógicas a valores digitales para poder procesarlos en el MCU, poder enviarlos por la terminal y generar las señales correspondientes para encender el LED. Se puede observar un esquema del periférico en cuestión en la imagen inferior.

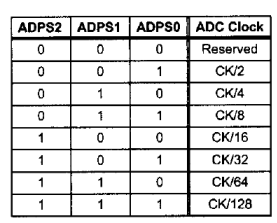


En el enunciado se pidió el uso del ADC3 y para ello se configuro el mismo de la siguiente manera:

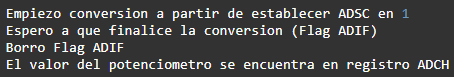
En el registro ADMUX se configura en 1 ADLAR (esto significa que se utilizan solo 8 bits para la conversión siendo estos la parte alta de ADC), en 1 MUX1 y en 1 MUX0 (Seleccionando de esta manera ADC3), y poniendo REFS0 en 1 y REFS1 en 0 para utilizar el voltaje VCC como referencia. En las siguientes imágenes se puede observar las diferentes configuraciones para los registros y también como se utilizó la parte alta del registro ADC para simplificar la codificación de los datos.



Por ultimo se configuro tambien el registro ADCSRA en el cual se establecio el bit ADEN en 1 (con tal de habilitar el conversor analogico-digital) y los bits ADPS2, ADPS1 y ADPS0 en 1 para establecer el reloj con un preescalador de 128 consiguiendo una frecuencia de 125KHz (para este valor se considera que el conversor tiene una buena performance). En la siguiente imagen se observa las distintas configuraciones de preescalador.



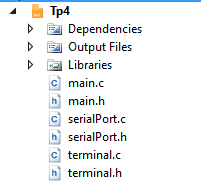
Las configuraciones previamente descriptas se incorporaron en el método Terminal\_configurarRegs() de la librería terminal.c por lo que al invocar el mismo el conversor ya estará listo para ser usado. A continuacion se muestra el pseudocodigo de la rutina usada para obtener la posición del potenciometro:



La rutina anterior se encuentra implementada en el metodo Terminal\_leerPotenciometro() de la librería terminal.c.

**Modularización**

El programa fue modularizado como se puede observar en la siguiente imagen:



**main:** hace referencia al programa principal donde se realizan las tareas background, es decir, al llegar un comando válido se realizan las tareas necesarias.

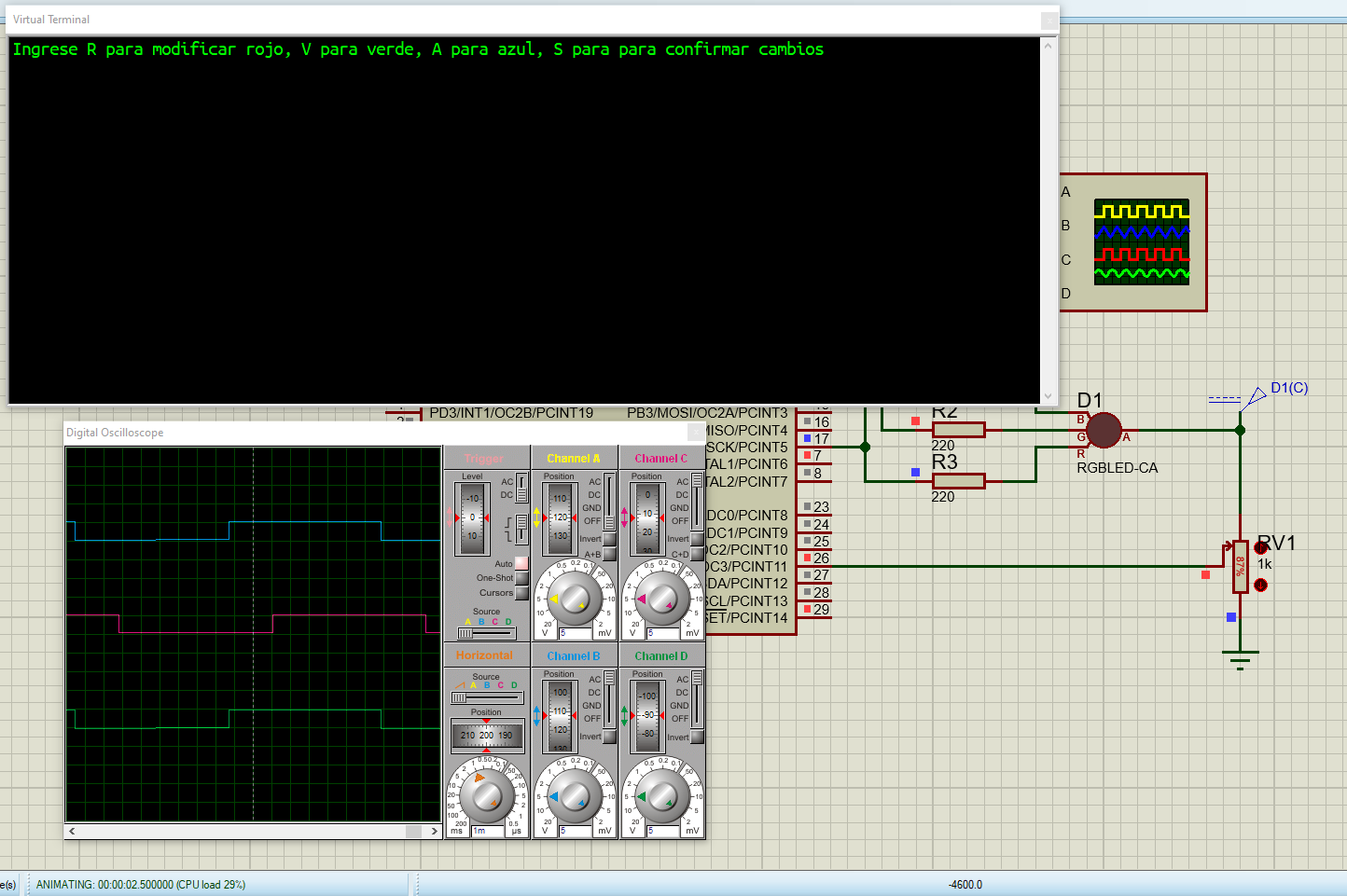
**serialPort:** es la librería brindada por la catedra para configurar y utilizar el puerto serie del MCU.

**terminal:** es una librería que en conjunto con serialPort se encargan de realizar la comunicación entre el MCU y la terminal. Los métodos prensentes en el mismo son los siguientes:

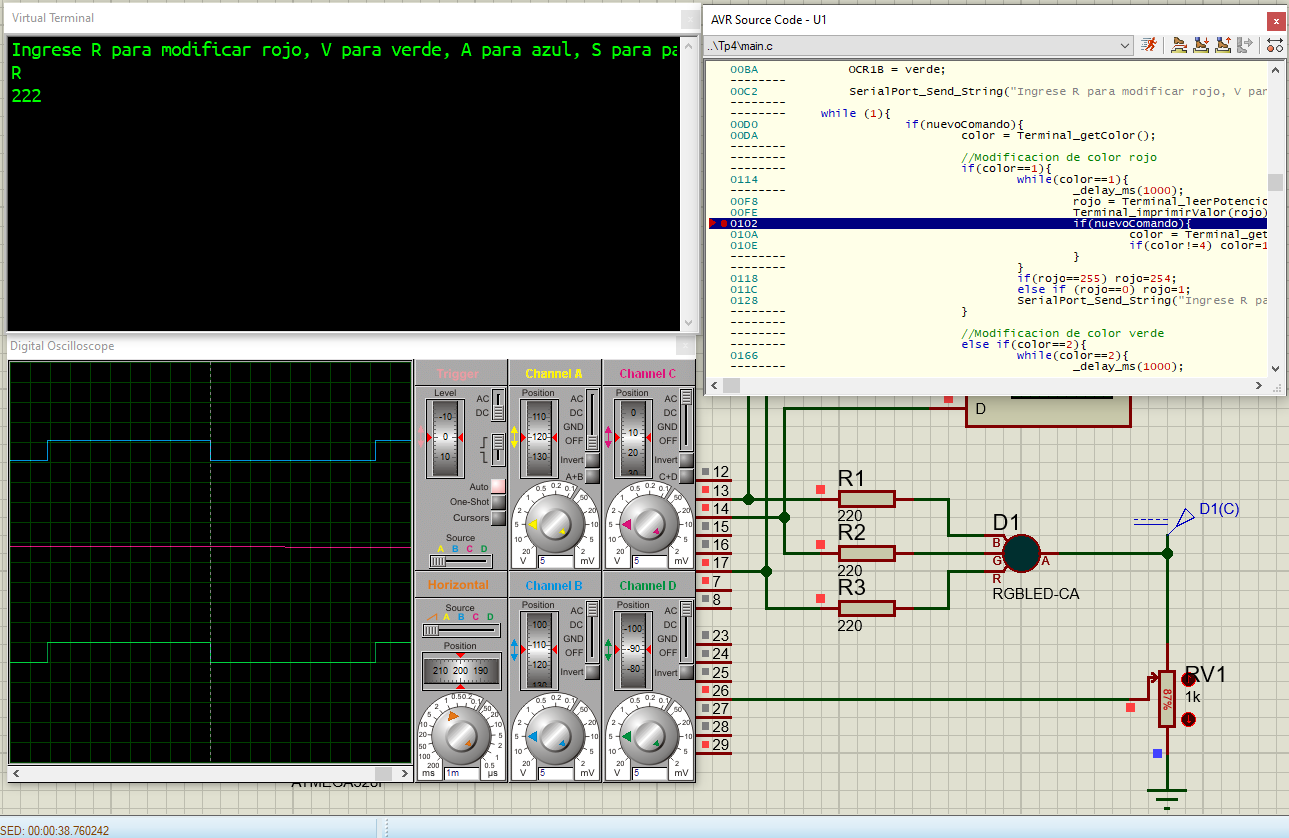
* Terminal\_leerComando(): se encarga de procesar el comando presente en el buffer.
* Terminal\_iniciarPuertoSerie(): configura e inicia el puerto serie.
* Terminal\_configurarRegs(): inicializa los registros del MCU (timer 0, timer 1 y ADC3).
* Terminal\_imprimirMenu(): imprimer el menu de opciones en la terminal.
* Terminal\_leerPotenciometro(): retorna un numero de 8 bits sin signo que representa la posición del potenciómetro.
* Terminal\_imprimirValor(uint8\_t): recibe como parametro la posicion del potenciometro y lo imprime en la terminal.
* Terminal\_getColor(): retorna un numero que indica que color se está modificando.
* ISR(USART\_RX\_vect): rutina de interrupcion del receptor del puerto serie. En esta misma se activan los flags (tareas de foreground) para que el programa principal realice las tareas necesarias (tareas de background).

**validación**

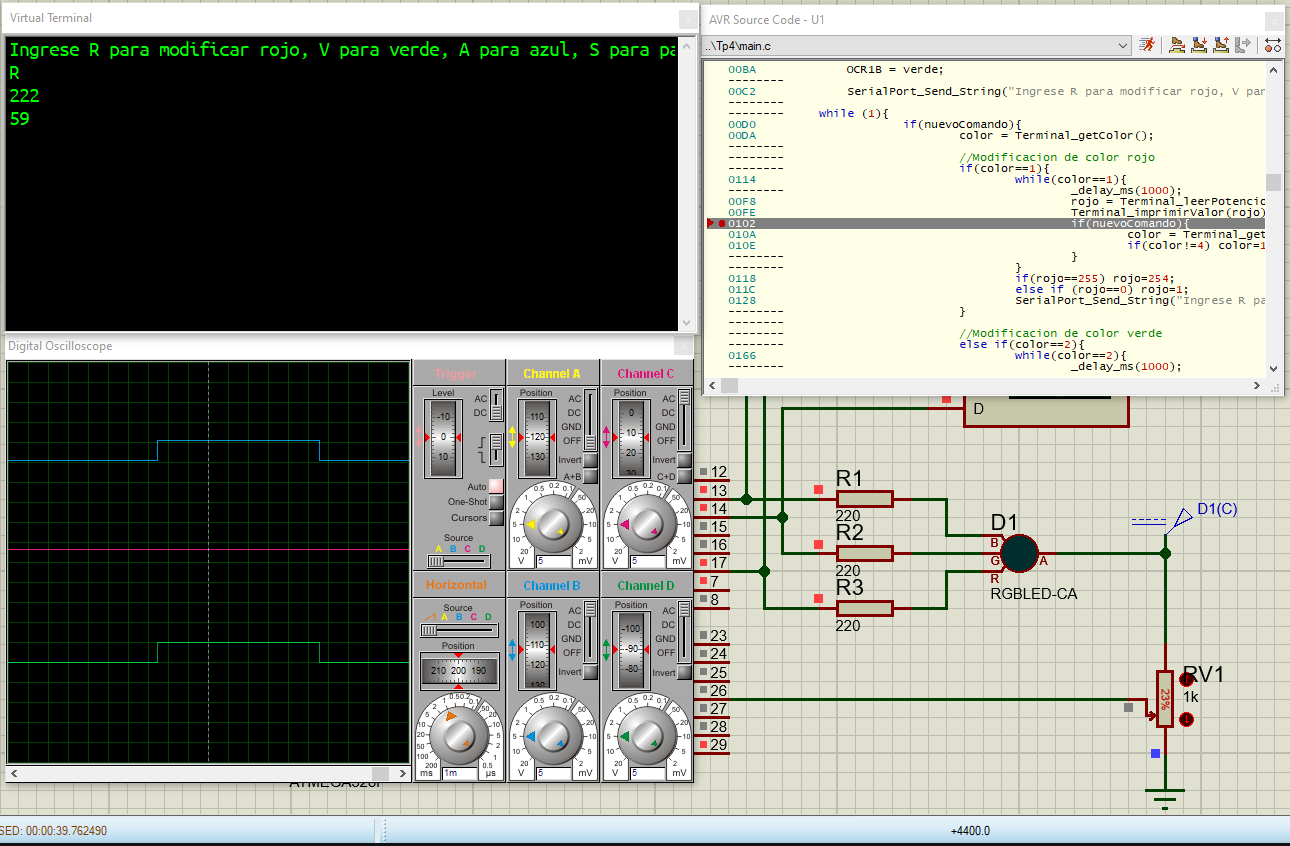
Apenas iniciamos la simulación en Proteus la terminal muestra el menú de opciones y el osciloscopio muestra que las señales generadas tienen el mismo ciclo de trabajo, esto tiene sentido ya que todos los colores inicialmente están configurados para brillar a mitad de intensidad. Se puede notar un desfasaje de la señal roja con respecto a las otras. Esto se debe a que la misma es generada por software en comparación a las otras que son generadas con timer1. El resultado producido por estas señales se puede ver en el color de led.



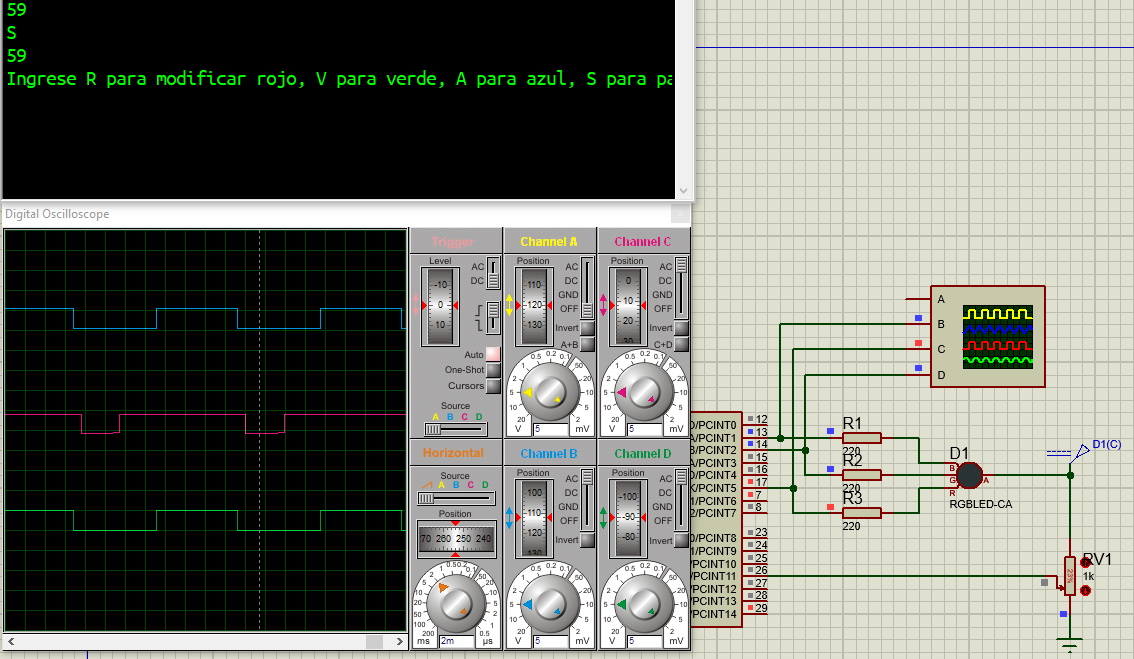
A continucacion se ingreso el comando R para entrar en la configuracion del color Rojo. Se puede ver que al ingresar el mismo en la terminal se imprimio el valor en el que esta el potenciometro. Se uso un breackpoint para que la simulacion se frene en dicho punto. En el osciloscopio se puede notar que la señal del color rojo se desactiva, esto se debe a que la señal se genera por software, por lo que mientras se este configurando el mismo la señal se desactivara. Para el caso de los otros dos colores esto no sucede ya que la mismas son generadas internamente por el MCU.



Luego movimos el potenciometro hacia abajo para que al avanzar al siguiente breakpoint se muestre el nuevo valor del potenciometro. Tambien en la parte inferior se puede ver que el tiempo de simulacion entre el primer valor impreso y el segundo es de aproximadamente un segundo, por lo que seria correcto.



Por ultimo se ingreso el comando S para asignale el valor del potenciometro como intensidad al color rojo. Como se puede observar en el osciloscopio el ciclo de trabajo de la señal para el color rojo se redujo considerablemte y el color del led cambio.



Para el resto de colores se hizo lo mismo y se observo el correcto funcionamiento del programa.

**Código**

**main.h**

#ifndef MAIN\_H\_

#define MAIN\_H\_

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <util/delay.h>

#include "serialPort.h"

#include "terminal.h"

#endif /\* MAIN\_H\_ \*/

**main.c**

#include "main.h"

volatile *uint8\_t* nuevoComando=0;

int main(void){

Terminal\_iniciarPuertoSerie(); //Se configura el puerto serie para poder usar la terminal

Terminal\_configurarRegs(); //Se configura timer 0, timer 1 y ADC3

*uint8\_t* color=0; //variable que indica el color que ese esta modificando, 1-Rojo, 2-Verde, 3-Azul, 4-Terminar

*uint8\_t* rojo=128; //Se configura la intensidad de los colores al iniciar el programa

*uint8\_t* verde=128;

*uint8\_t* azul=128;

if(rojo==255) rojo=254;

else if (rojo==0) rojo=1;

OCR1A = azul;

OCR1B = verde;

Terminal\_imprimirMenu();

while (1){

if(nuevoComando){ //Se activa unicamente si se ingreso un comando valido

color = Terminal\_getColor(); //se obtiene el color a modificar

//Modificacion de color rojo

if(color==1){

while(color==1){

*\_delay\_ms*(1000);

rojo = Terminal\_leerPotenciometro(); //se obtiene la posicion del potenciometro

Terminal\_imprimirValor(rojo); //se imprime en la terminal cada un segundo

if(nuevoComando){

color = Terminal\_getColor();

if(color!=4) color=1; //Si se quiere entrar a configurar otro color se debe terminar de configurar el actual

}

}

if(rojo==255) rojo=254;

else if (rojo==0) rojo=1;

Terminal\_imprimirMenu(); //una vez configurado el color volvemos a imprimir el menu

}

//Modificacion de color verde

else if(color==2){

while(color==2){

*\_delay\_ms*(1000);

verde = Terminal\_leerPotenciometro();

Terminal\_imprimirValor(verde);

if(nuevoComando){

color = Terminal\_getColor();

if(color!=4) color=2;

}

}

OCR1B = verde;

Terminal\_imprimirMenu();

}

//Modificacion de color azul

else if(color==3){

while(color==3){

*\_delay\_ms*(1000);

azul = Terminal\_leerPotenciometro();

Terminal\_imprimirValor(azul);

if(nuevoComando){

color = Terminal\_getColor();

if(color!=4) color=3;

}

}

OCR1A = azul;

Terminal\_imprimirMenu();

}

}

//Pongo en bajo PB5 el tiempo necesario para cumplir con el ciclo de trabajo

PORTB &= ~(1<<PORTB5);

OCR0A = rojo;

TCNT0 = 0;

while((TIFR0&(1<<OCF0A))==0);

TIFR0 |= (1<<OCF0A);

//Pongo en alto PB5 el tiempo necesario para completar el periodo

PORTB |= (1<<PORTB5);

OCR0A = 255-rojo;

TCNT0 = 0;

while((TIFR0&(1<<OCF0A))==0);

TIFR0 |= (1<<OCF0A);

}

}

**terminal.h**

#ifndef TERMINAL\_H\_

#define TERMINAL\_H\_

#include <stdint.h>

#include <avr/io.h>

#include "serialPort.h"

#define BR9600 (0x67) // 0x67=103 configura BAUDRATE=9600@16MHz

void Terminal\_iniciarPuertoSerie(void);

void Terminal\_configurarRegs(void);

void Terminal\_imprimirMenu(void);

*uint8\_t* Terminal\_leerPotenciometro(void);

void Terminal\_imprimirValor(*uint8\_t*);

*uint8\_t* Terminal\_leerComando();

*uint8\_t* Terminal\_getColor(void);

static *uint8\_t* menu[]="Ingrese R para modificar rojo, V para verde, A para azul, S para para confirmar cambios \r\n";

#endif /\* TERMINAL\_H\_ \*/

**terminal.c**

#include "terminal.h"

*uint8\_t* color=0;

*uint8\_t* RX\_Buffer;

*uint8\_t* bufferRX[10];

*uint8\_t* cant=0;

extern *uint8\_t* nuevoComando;

static *uint8\_t* menu[]="Ingrese R para modificar rojo, V para verde, A para azul, S para para confirmar cambios \r\n";

void Terminal\_iniciarPuertoSerie(void){

SerialPort\_Init(BR9600); // Configuro tramas 8N1 a 9600bps

SerialPort\_TX\_Enable(); // Activo el transmisor del puerto serie

SerialPort\_RX\_Enable(); // Activo el Receptor del puerto serie

SerialPort\_RX\_Interrupt\_Enable(); // Activo Interrupción de recepción

sei(); // Activo la máscara global de interrupciones

}

void Terminal\_configurarRegs(void){

TCCR0A = (1<<WGM01); //Modo CTC timer 0

TCCR0B = (1<<CS02) | (1<<CS00); //prescaler 1024

DDRB = (1<<PORTB1) | (1<<PORTB2)| (1<<PORTB5); //se configura PB1, PB2 y PB5 como salida

TCCR1A = (1<<COM1A0) | (1<<COM1A1) | (1<<COM1B0) | (1<<COM1B1) | (1<<WGM10); //PWM invertido en PB1 y PB2

TCCR1B = (1<<CS10)| (1<<CS12) | (1<<WGM12); //Modos fast pwm 8-bits con prescaler 1024

DIDR0 = (1<<ADC3D); //se configura el pin del ADC3 como entrada analogica

ADCSRA= 0x87; //habilitamos el adc y seleccionamos ck/128

ADMUX= (1 << ADLAR) | (1 << MUX1) | (1 << MUX0) | (1 << REFS0); //se elige Vref=AVCC, justificado a la izquierda

}

void Terminal\_imprimirMenu(){

SerialPort\_Send\_String(menu);

}

*uint8\_t* Terminal\_leerPotenciometro(){

ADCSRA |= (1<<ADSC); //Empezar conversion

while((ADCSRA&(1<<ADIF))==0); //Esperar a que termine la conversion

ADCSRA |= (1<<ADIF); //Limpiar flag

return ADCH;

}

void Terminal\_imprimirValor(*uint8\_t* num){

static *uint8\_t* c, aux;

aux=0;

c = num/100;

if(c!=0){

aux=1;

SerialPort\_Wait\_For\_TX\_Buffer\_Free();

SerialPort\_Send\_Data('0'+c);

}

c = (num%100)/10;

if(c!=0){

SerialPort\_Wait\_For\_TX\_Buffer\_Free();

SerialPort\_Send\_Data('0'+c);

}else if(aux==1){

SerialPort\_Wait\_For\_TX\_Buffer\_Free();

SerialPort\_Send\_Data('0'+c);

aux=0;

}

c = num%100%10;

SerialPort\_Wait\_For\_TX\_Buffer\_Free();

SerialPort\_Send\_Data('0'+c);

SerialPort\_Wait\_For\_TX\_Buffer\_Free();

SerialPort\_Send\_String("\r\n");

}

*uint8\_t* Terminal\_leerComando(){

if((bufferRX[0]=='R') && (bufferRX[1]=='\n'))

return 1;

else if((bufferRX[0]=='V') && (bufferRX[1]=='\n'))

return 2;

else if((bufferRX[0]=='A') && (bufferRX[1]=='\n'))

return 3;

else if((bufferRX[0]=='S') && (bufferRX[1]=='\n'))

return 4;

else

return 0;

}

*uint8\_t* Terminal\_getColor(void){

return color;

}

ISR(USART\_RX\_vect){

RX\_Buffer = SerialPort\_Recive\_Data();

if(RX\_Buffer!='\r'){

bufferRX[cant]=RX\_Buffer;

cant++;

}else{

bufferRX[cant]='\n';

color = Terminal\_leerComando();

if(color!=0){

nuevoComando=1;

}

cant=0;

}

}

**serialPort.h**

#ifndef SERIALPORT\_H\_

#define SERIALPORT\_H\_

// ------------------- Includes ----------------------------

// Archivo de cabecera del Microcontrolador

#include <avr/io.h>

// Interrupciones del Microcontrolador

#include <avr/interrupt.h>

// -------- Prototipos de funciones Publicas ---------------

// Inicializacion de Puerto Serie

void SerialPort\_Init(*uint8\_t*);

// Inicializacion de Transmisor

void SerialPort\_TX\_Enable(void);

void SerialPort\_TX\_Interrupt\_Enable(void);

void SerialPort\_TX\_Interrupt\_Disable(void);

// Inicializacion de Receptor

void SerialPort\_RX\_Enable(void);

void SerialPort\_RX\_Interrupt\_Enable(void);

// Transmision

void SerialPort\_Wait\_For\_TX\_Buffer\_Free(void); // Pooling - Bloqueante hasta que termine de transmitir.

void SerialPort\_Send\_Data(char);

void SerialPort\_Send\_String(char \*);

void SerialPort\_Send\_uint8\_t(*uint8\_t*);

void SerialPort\_send\_int16\_t(int val,unsigned int field\_length); //This function writes a integer type value to UART

//-32768 y 32767

// Recepcion

void SerialPort\_Wait\_Until\_New\_Data(void); // Pooling - Bloqueante, puede durar indefinidamente!

char SerialPort\_Recive\_Data(void);

//Driver P.C.

void SerialPort\_Write\_Char\_To\_Buffer ( char Data );

void SerialPort\_Write\_String\_To\_Buffer( char \* STR\_PTR );

void SerialPort\_Send\_Char (char dato);

void SerialPort\_Update(void);

char SerialPort\_Get\_Char\_From\_Buffer (char \* ch);

char SerialPort\_Get\_String\_From\_Buffer (char \* string);

char SerialPort\_Receive\_data (char \* dato);

#endif /\* SERIALPORT\_H\_ \*/

**serialPort.c**

#include "SerialPort.h"

#define TX\_BUFFER\_LENGTH 32

#define RX\_BUFFER\_LENGTH 32

volatile static unsigned char TXindice\_lectura=0, TXindice\_escritura=0;

volatile static unsigned char RXindice\_lectura=0, RXindice\_escritura=0;

static char TX\_Buffer [TX\_BUFFER\_LENGTH];

static char RX\_Buffer [RX\_BUFFER\_LENGTH];

// Inicialización de Puerto Serie

void SerialPort\_Init(*uint8\_t* config){

// config = 0x67 ==> Configuro UART 9600bps, 8 bit data, 1 stop @ F\_CPU = 16MHz.

// config = 0x33 ==> Configuro UART 9600bps, 8 bit data, 1 stop @ F\_CPU = 8MHz.

// config = 0x25 ==> Configuro UART 9600bps, 8 bit data, 1 stop @ F\_CPU = 4MHz.

UCSR0B = 0;

UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);

//UBRR0H = (unsigned char)(config>>8);

UBRR0L = (unsigned char)config;

}

// Inicialización de Transmisor

void SerialPort\_TX\_Enable(void){

UCSR0B |= (1<<TXEN0);

}

void SerialPort\_TX\_Interrupt\_Enable(void){

UCSR0B |= (1<<UDRIE0);

}

void SerialPort\_TX\_Interrupt\_Disable(void)

{

UCSR0B &=~(1<<UDRIE0);

}

// Inicialización de Receptor

void SerialPort\_RX\_Enable(void){

UCSR0B |= (1<<RXEN0);

}

void SerialPort\_RX\_Interrupt\_Enable(void){

UCSR0B |= (1<<RXCIE0);

}

// Transmisión

// Espera hasta que el buffer de TX este libre.

void SerialPort\_Wait\_For\_TX\_Buffer\_Free(void){

// Pooling - Bloqueante hasta que termine de transmitir.

while(!(UCSR0A & (1<<UDRE0)));

}

void SerialPort\_Send\_Data(char data){

UDR0 = data;

}

void SerialPort\_Send\_String(char \* msg){ //msg -> "Hola como andan hoy?" 20 ASCII+findecadena, tardo=20ms

*uint8\_t* i = 0;

//'\0' = 0x00

while(msg[i]){ // \*(msg+i)

SerialPort\_Wait\_For\_TX\_Buffer\_Free(); //9600bps formato 8N1, 10bits, 10.Tbit=10/9600=1ms

SerialPort\_Send\_Data(msg[i]);

i++;

}

}

// Recepción

// Espera hasta que el buffer de RX este completo.

void SerialPort\_Wait\_Until\_New\_Data(void){

// Pooling - Bloqueante, puede durar indefinidamente!

while(!(UCSR0A & (1<<RXC0)));

}

char SerialPort\_Recive\_Data(void){

return UDR0;

}

void SerialPort\_Send\_uint8\_t(*uint8\_t* num){

SerialPort\_Wait\_For\_TX\_Buffer\_Free();

SerialPort\_Send\_Data('0'+num/100);

num-=100;

SerialPort\_Wait\_For\_TX\_Buffer\_Free();

SerialPort\_Send\_Data('0'+num/10);

SerialPort\_Wait\_For\_TX\_Buffer\_Free();

SerialPort\_Send\_Data('0'+ num%10);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This function writes a integer type value to UART

Arguments:

1)int val : Value to print

2)unsigned int field\_length :total length of field in which the value is printed

must be between 1-5 if it is -1 the field length is no of digits in the val

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SerialPort\_send\_int16\_t(int val,unsigned int field\_length)

{

char str[5]={0,0,0,0,0};

int i=4,j=0;

while(val)

{

str[i]=val%10;

val=val/10;

i--;

}

if(field\_length==-1)

while(str[j]==0) j++;

else

j=5-field\_length;

if(val<0) {

SerialPort\_Wait\_For\_TX\_Buffer\_Free();

SerialPort\_Send\_Data('-');

}

for(i=j;i<5;i++)

{

SerialPort\_Wait\_For\_TX\_Buffer\_Free();

SerialPort\_Send\_Data('0'+str[i]);

}

}

//\*\*\*\*\*\*\*\*\*

void SerialPort\_Write\_Char\_To\_Buffer ( char Data )

{

// Write to the buffer \*only\* if there is space

if (TXindice\_escritura < TX\_BUFFER\_LENGTH){

TX\_Buffer[TXindice\_escritura] = Data;

TXindice\_escritura++;

}

else {

// Write buffer is full

//Error\_code = ERROR\_UART\_FULL\_BUFF;

}

}

void SerialPort\_Write\_String\_To\_Buffer( char \* STR\_PTR )

{

unsigned char i = 0;

while ( STR\_PTR [ i ] != '\0')

{

SerialPort\_Write\_Char\_To\_Buffer ( STR\_PTR [ i ] );

i++;

}

}

void SerialPort\_Send\_Char (char dato)

{

SerialPort\_Wait\_For\_TX\_Buffer\_Free(); // Espero a que el canal de transmisión este libre (bloqueante)

SerialPort\_Send\_Data(dato);

}

void SerialPort\_Update(void)

{

static char key;

if ( UCSR0A & (1<<RXC0) ) { // Byte recibido. Escribir byte en buffer de entrada

if (RXindice\_escritura < RX\_BUFFER\_LENGTH) {

RX\_Buffer [RXindice\_escritura] =UDR0; // Guardar dato en buffer

RXindice\_escritura++; // Inc sin desbordar buffer

}

//else

//Error\_code = ERROR\_UART\_FULL\_BUFF;

}

// Hay byte en el buffer Tx para transmitir?

if (TXindice\_lectura < TXindice\_escritura){

SerialPort\_Send\_Char ( TX\_Buffer [TXindice\_lectura] );

TXindice\_lectura++;

}

else {// No hay datos disponibles para enviar

TXindice\_lectura = 0;

TXindice\_escritura = 0;

}

}

char SerialPort\_Get\_Char\_From\_Buffer (char \* ch)

{

// Hay nuevo dato en el buffer?

if (RXindice\_lectura < RXindice\_escritura){

\*ch = RX\_Buffer [RXindice\_lectura];

RXindice\_lectura++;

return 1; // Hay nuevo dato

}

else {

RXindice\_lectura=0;

RXindice\_escritura=0;

return 0; // No Hay

}

}

char SerialPort\_Get\_String\_From\_Buffer (char \* string)

{

char rxchar=0;

do{

if(SerialPort\_Get\_Char\_From\_Buffer (&rxchar)){

\*string=rxchar;

string++;

}

else{

rxchar='\n'; //empty string

}

}while(rxchar!='\n');

\*string='\0'; //End of String

return 1;

}

char SerialPort\_Receive\_data (char \* dato)

{

if ( (UCSR0A & (1<<RXC0))==1) {

\*dato=UDR0;

return 1;

}

return 0; //no data

}