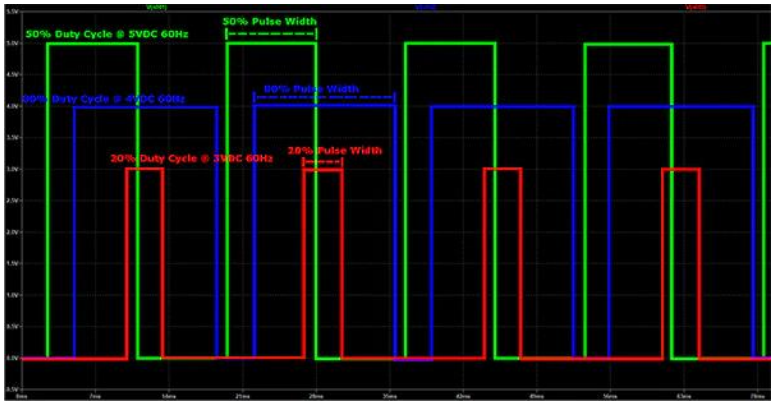
1 DE AGOSTO DE 20220



TP N°4 ENTREGABLE

BLANCO VALENTIN NICOLAS, PALADINO GABRIEL AGUSTIN
CIRCUITOS DIGITALES Y MICROCONTROLADORES

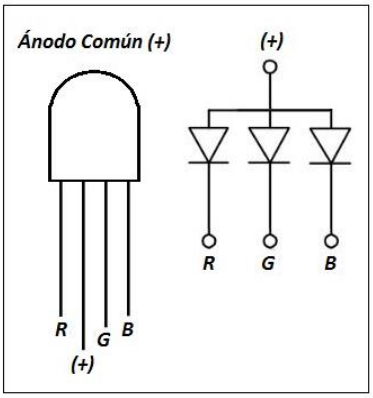
Interpretación

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



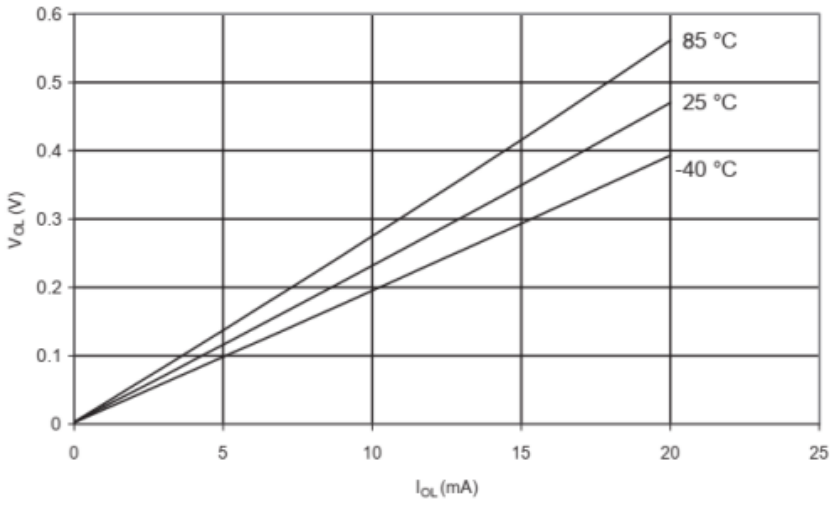
Ya que el led es de tipo ánodo común (como se observa en la imagen superior) la intensidad de los colores serán activos en bajo por lo que la manera en la que se calcula la corriente máxima es la siguiente:

$$I_{OL} = (V_{CC} - V_{OL} - V_{LED}) / R_{LED}$$

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:

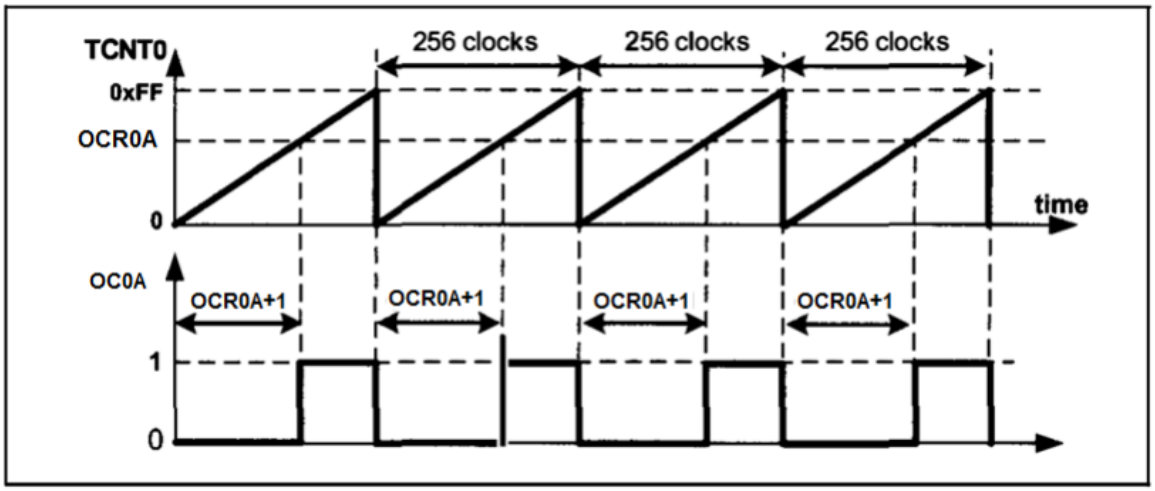
Color	Voltaje de color	Corriente maxima
Rojo	2.049V	11.27mA
Verde	2.81V	7.81mA
Azul	2.871V	7.54mA

Figure 29-160.ATmega328P: I/O Pin Output Voltage vs. Sink Current (V_{CC} = 5 V)



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.



Fast PWM Modo Invertido

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:

Table 15-4. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

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.

Table 15-2. Compare Output Mode, Fast PWM[1]

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM (non-inverting mode)
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at BOTTOM (inverting mode)

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

$$Fgenerated\ wave = \frac{Foscillator}{256\ x\ N}$$

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.

$$Duty\ Cycle = \frac{255 - OCR0}{256}\ x\ 100$$

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:

$$t_{bajo} = \frac{rojo * T}{255}$$

$$t_{alto} = T - t_{bajo}$$

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:

$$OCRA0\_bajo = \frac{t\_bajo * 255}{T}$$

Reemplazando la formula de t_bajo se obtuvo

$$OCRA0_{bajo} = \frac{\frac{rojo * T}{255} * 255}{T} = rojo$$

Señal OCRA0 en alto:

$$OCRA0_{alto} = \frac{(T - t\_bajo) * 255}{T}$$

Reemplazando la fórmula de t_bajo se obtuvo

$$OCRA0_{alto} = \frac{T(1 - \frac{rojo}{255}) * 255}{T} = 255 - rojo$$

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.

```
realizar indefinidamente
    Se pone en bajo el PB5
    asigno OCR0A para señal en bajo
    TCNT0 se resetea
    espero a que TCNT0 sea igual a OCR0A
    limpio flag de OCR0A

    pongo PB5 en alto
    asigno OCR0A para señal en alto
    reset de TCNT0
    espero a que TCNT0 sea igual a OCR0A
    limpio flag de OCR0A
```

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.

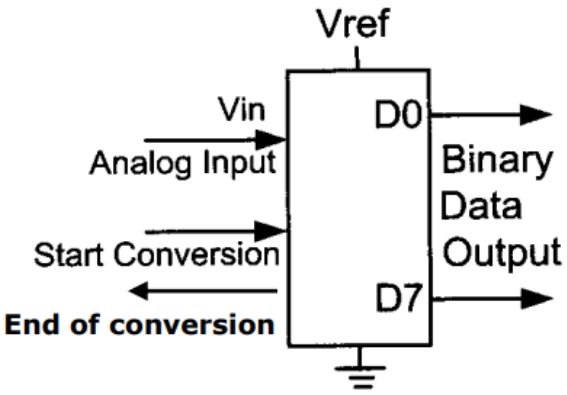
A<Enter>: configuración del color azul.

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.



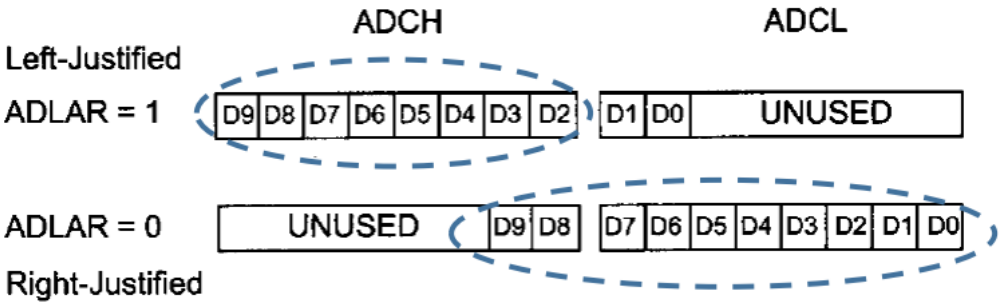
En el enunciado se pidió el uso del ADC3 y para ello se configuro el mismo de la siguiente manera:

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.

Table 13-4: V_{ref} Source Selection Table for AVR

REFS1	REFS0	V_{ref}	
0	0	AREF pin	Set externally
0	1	AVCC pin	Same as VCC
1	0	Reserved	----
1	1	Internal 1.1V	Fixed regardless of VCC value

MUX3...0	Single-ended Input
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5
00110	ADC6
00111	ADC7



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.

ADPS2	ADPS1	ADPS0	ADC Clock
0	0	0	Reserved
0	0	1	CK/2
0	1	0	CK/4
0	1	1	CK/8
1	0	0	CK/16
1	0	1	CK/32
1	1	0	CK/64
1	1	1	CK/128

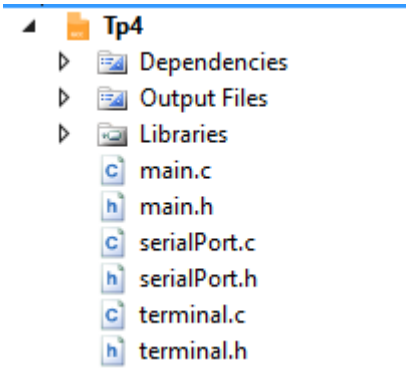
Las configuraciones previamente descriptas se incorporaron en el método Terminal_configurarRegs() de la librería terminal.c por lo que al invocar el mismo el conversor ya estará listo para ser usado. A continuacion se muestra el pseudocodigo de la rutina usada para obtener la posición del potenciometro:

```
Empiezo conversion a partir de establecer ADSC en 1
Espero a que finalice la conversion (Flag ADIF)
Borro Flag ADIF
El valor del potenciometro se encuentra en registro ADCH
```

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.

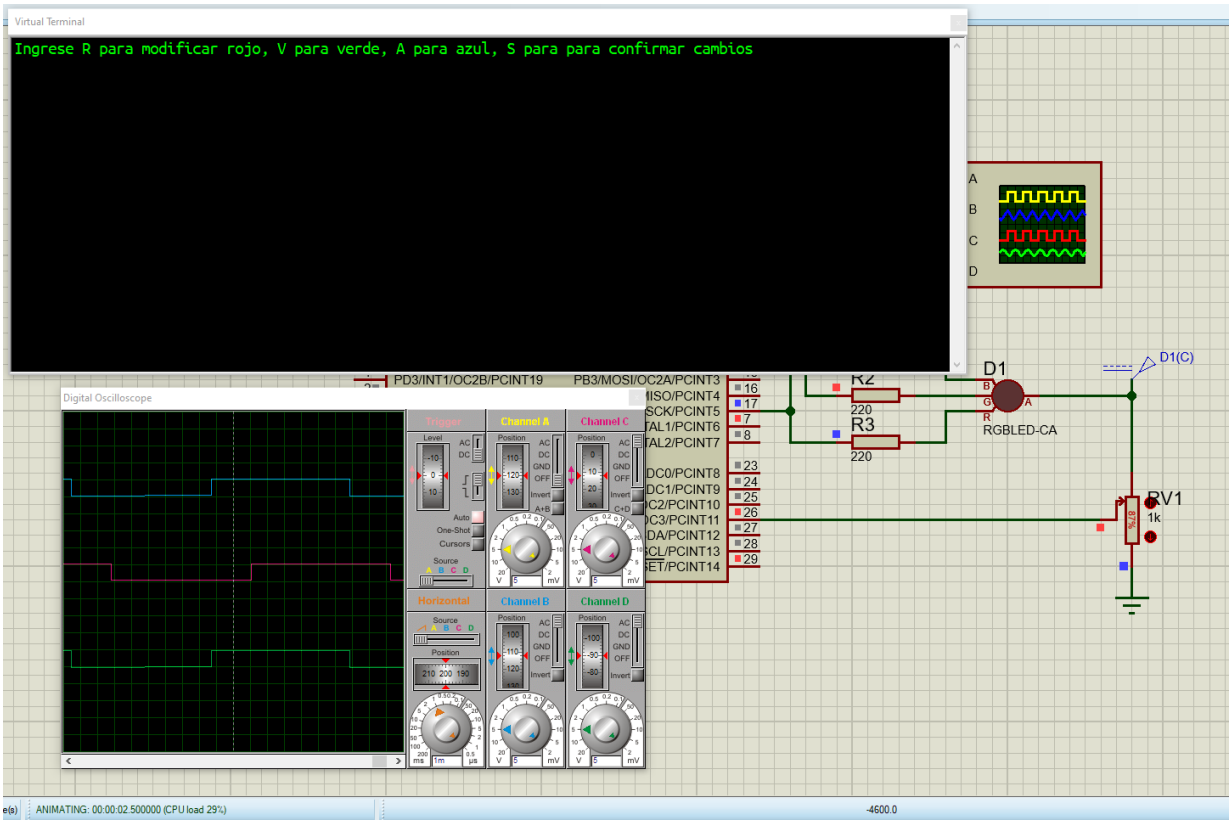
serialPort: es la librería brindada por la catedra para configurar y utilizar el puerto serie del MCU.

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:

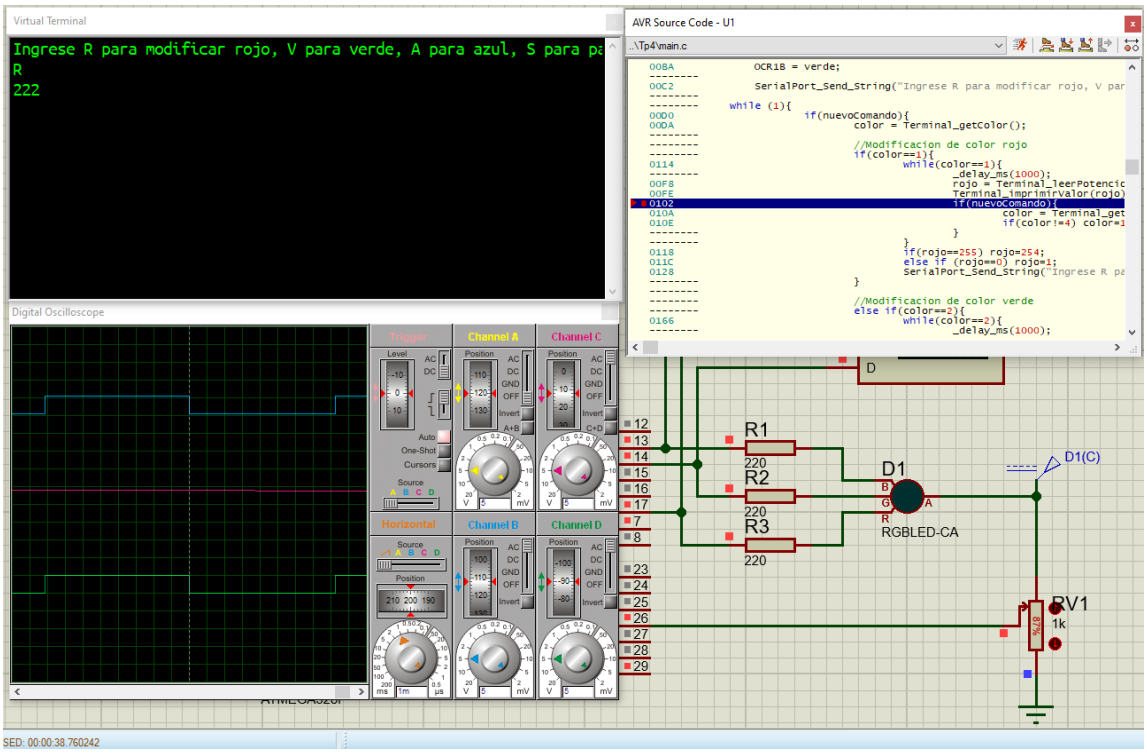
- Terminal_leerComando(): se encarga de procesar el comando presente en el buffer.
- Terminal_iniciarPuertoSerie(): configura e inicia el puerto serie.
- Terminal_configurarRegs(): inicializa los registros del MCU (timer 0, timer 1 y ADC3).
- Terminal_imprimirMenu(): imprimer el menu de opciones en la 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

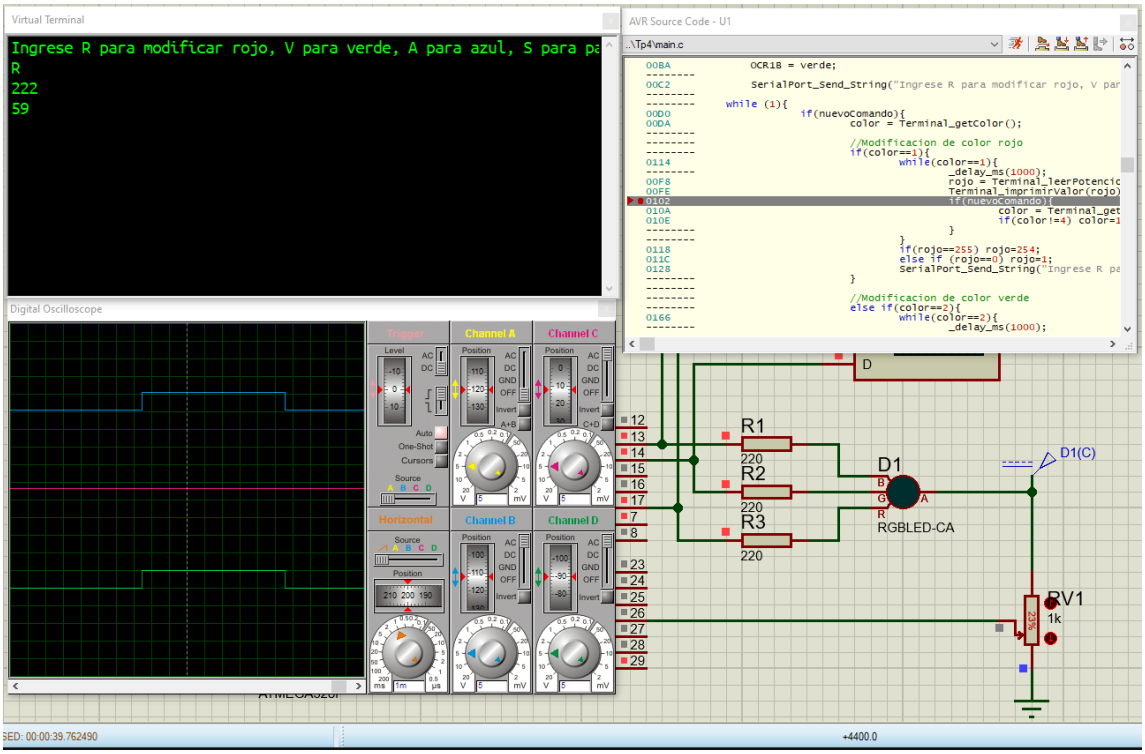
Apenas iniciamos la simulación en Proteus la terminal muestra el menú de opciones y el osciloscopio muestra que las señales generadas tienen el mismo ciclo de trabajo, esto tiene sentido ya que todos los colores inicialmente están configurados para brillar a mitad de intensidad. Se puede notar un desfasaje 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.



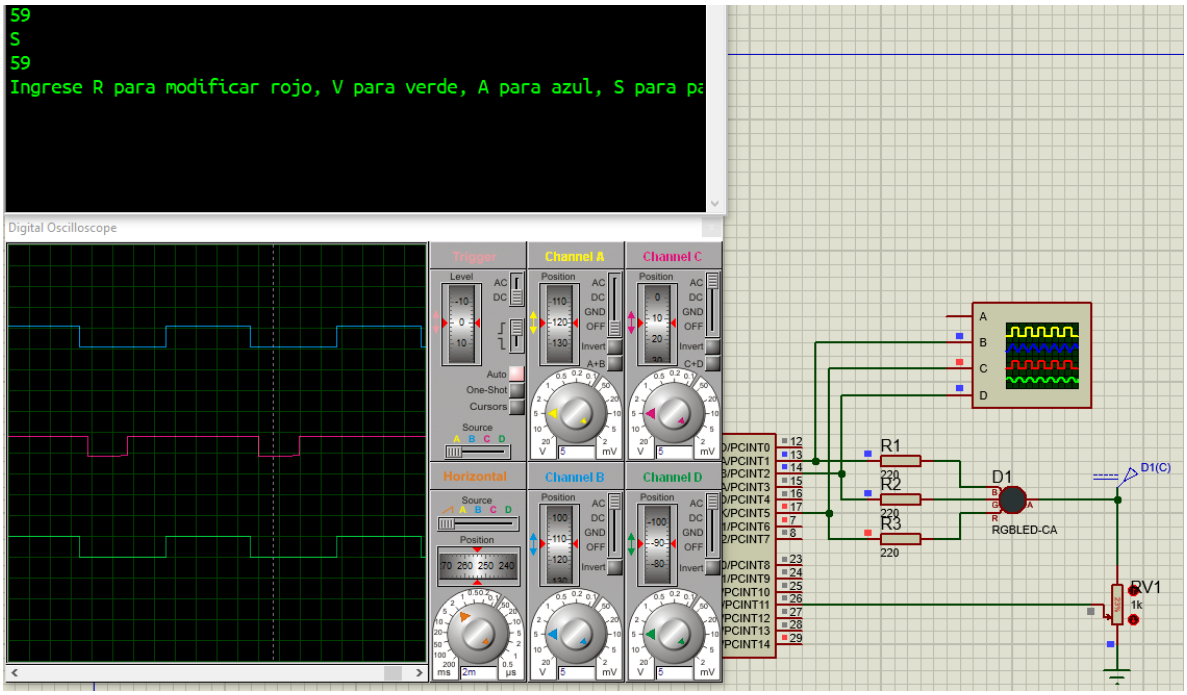
A continucacion se ingreso el comando R para entrar en la configuracion del color Rojo. Se puede ver que al ingresar el mismo en la terminal se imprimio el valor en el que esta el potenciometro. Se uso un breackpoint para que la simulacion se frene en dicho punto. En el osciloscopio se puede notar que 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
}
```