Universidad Nacional de La Plata

Trabajo Práctico 4 - Grupo 20

Control de un LED RGB

Circuitos Digitales y Microcontroladores

Tarifa, Carla.
Carballo Ormaechea, Lucas.

Índice

1.Problema

Realizar un programa para controlar la intensidad y el color del LED RGB con la técnica de PWM. En el kit de clases el mismo se encuentra conectado a los terminales PB5, PB2 y PB1 (RGB) a través de resistencias de limitación de 220 ohms y en forma ánodo común. Para la simulación del modelo en Proteus puede utilizar RGBLED-CA.
Requerimientos detallados:
1. Genere en los tres terminales de conexión del LED, tres señales PWM de frecuencia mayor o igual a 50Hz y con una resolución de 8 bits cada una.
2. Seleccione la proporción de color de cada LED (de 0 a 255) para obtener un color resultante.
3. Mediante un comando por la interfaz serie UART0 deberá activar cual proporción de color desea modificar. Por ejemplo, envíe ´R´ para modificar el rojo, ´G´ para el verde y ´B´ para el azul.
4. Utilice el potenciómetro (resistencia variable) del kit conectado al terminal ADC3 para modificar el brillo del color seleccionado.

1.1Interpretación

Se deberá realizar un program en c , donde se conectara un LED RGB a un MCU y se le dará distintas señales para que este encienda distintos colores.Dicho led cuenta con 3 pines para su control(uno para cada color) y la conexión con el MCU será en PB5 para el rojo,PB2 para el verde y PB1 para el azul.

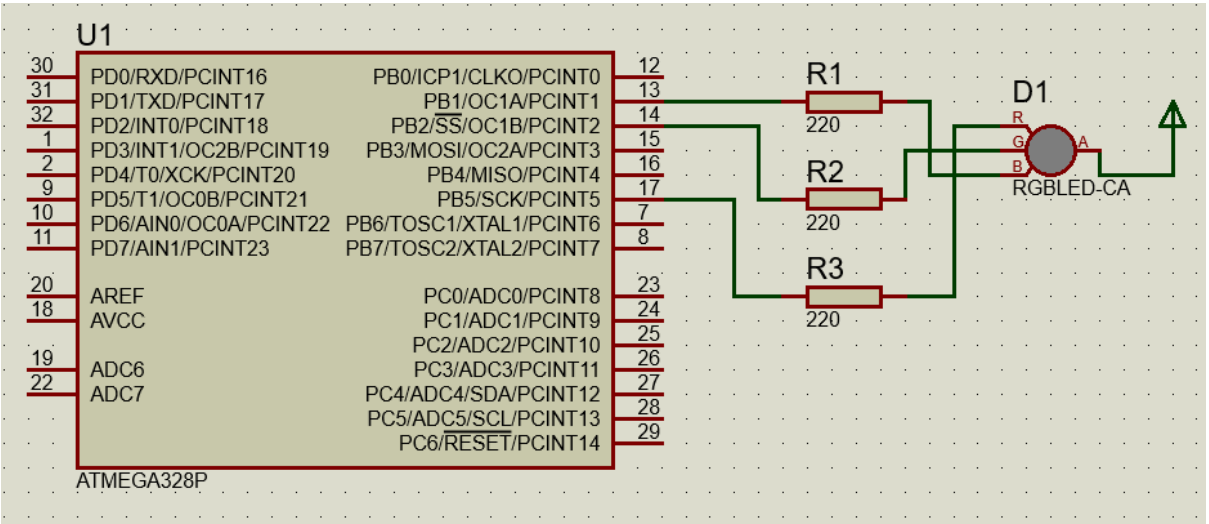
Requerimientos

1) Para generar señales PWM , se debe configurar el timer 1 en modo PWM debido a que PB1 y PB2 concuerdan con la salida de estas . Para generar una señal PWM en PB5 se debe realizar por software.
2) Realizar una funcion para seleccionar una proporcion de color de cada LED en un rango de 0 a 255.
3) Utilizar la interfaz serie UART0 para permitir al usuario seleccionar el color del led a modificar,para esto se utilizará la librería serialPort provista por la cátedra en el tp anterior.
4) Conectar el potenciómetro al terminal ADC3 , leer su valor y ajustar su intensidad.

2.Solución

2.1 Led RGB

El led RGB es un componente electrónico que emite luz en los tres colores primarios (Rojo, verde y azul) donde, estos tres colores combinados en diversas proporciones crean una gama de colores visibles .
El led está compuesto por 3 diodos, uno para cada color, el cual se activan individualmente, la forma en que está conectado es en ánodo común, es decir que , tiene un terminal común positivo que se conecta a la fuente de voltaje positiva (VCC). Los tres pines restantes del LED se utilizan para controlar la intensidad de los colores y cada pin está conectado a un diodo emisor de luz de un color específico.Cuando se desea encender uno de los colores, el pin correspondiente se conecta a la tierra (GND). Al conectar el pin a GND, se establece un camino de corriente para el diodo emisor de luz del color correspondiente, y este se enciende.Por último cada uno de estos diodos tiene una resistencia de 220Ω.
La intensidad del color se controla ajustando el ciclo de trabajo de la señal PWM que se aplica al pin del color correspondiente.
A continuación se muestra su conexión al MCU.



Conexión del led RGB y sus resistencias al MCU

2.2 Generar Señales

2.2.1 Introducción

La señal PWM es una técnica que permite controlar el valor medio de una señal digital periódica, por ende controlando la intensidad de un LED variando el ancho de los pulsos de

la señal eléctrica que se le aplica, manteniendo fijo el período. De esta forma lo que estamos haciendo es modificar el ciclo de trabajo de la señal generada.

Las señales PWM pueden dividirse en dos tipos de funcionamiento :

1) **Fase Correcta**

   En este modo de trabajo, poseemos un registro llamado TCNT el cual es un contador y varía en un rango de 0 a TOP. El contador suma hasta llegar al valor máximo, una vez alcanzado este valor, decrece hasta 0 y se cumple un período de la señal.
   Existe otro registro llamado OCRx (el valor de x viene determinado por el TIMER elegido, por ejemplo si fuese TIMER 0 el nombre de dicho registro sería OCR0) en el cual se puede configurar un valor determinado y la existencia de este registro nos ayuda a generar la señal deseada en OCx.
   La frecuencia de la señal generada puede calcularse de la siguiente forma:

   Fpwm = Foscilador / (2*TOP*N)

$$DC = \frac{2 * OCRnA}{2 * TOP} * 100$$

Ciclo de trabajo para el modo NO invertido

$$DC = \frac{TOP - OCRnA}{TOP} * 100$$

Ciclo de trabajo para el modo invertido

   Cabe destacar que, la señal generada en OCx puede ser en modo invertido o modo NO invertido, como se muestra en la figura. Y para cualquiera de los dos modos, es posible tener 0% o 100% del ciclo de trabajo.

2) **Fast - PWM**

El modo de funcionamiento Fast - PWM es idéntico al anterior en cuanto a registros ya que también poseemos el contador TCNT, el registro OCRx y la señal generada en OCx. La gran diferencia en este caso es que el contador TCNT cuenta desde 0 hasta el valor máximo pero, en lugar de decrementar nuevamente hasta 0, genera un toggle que lleva el valor de TCNT del máximo a 0 sin escalas. La frecuencia de la señal generada puede calcularse de la siguiente forma:

Fpwm= Foscilador / (TOP+1)*N

$$DC = \frac{OCR + 1}{TOP + 1} * 100$$

Ciclo de trabajo para modo no invertido

$$DC = \frac{TOP - OCR}{TOP + 1} * 100$$

Ciclo de trabajo para modo invertido

Para el modo NO INVERTIDO es posible tener MÁS del 0% de ciclo de trabajo (mayor estricto > 0%). Análogamente para el modo INVERTIDO es posible obtener MENOS que el 100% de ciclo de trabajo (menor estricto < 100%).

2.2.2 Timer 1

Los colores azul y verde corresponden a los puertos PB1 y PB2 se puede utilizar el modo PWM del timer 1, ya que la señal generada por los mismos se refleja en esos puertos.
El problema propuesto solicita 8 bits por lo tanto se deberá configurar el timer1 en modo 5 según la tabla y luego se deberá poner en 1 los registros de configuración (COM1A1 Y COM1A0) para ponerlos en modo invertido.

- Cálculos

  Para obtener el preescaler(N) correcto se utilizará la siguiente fórmula , teniendo en cuenta los siguiente datos : Foscillator = 16MHz , (TOP+1)=256 y Fgenerated=50Hz

  $$F_{generated\ wave} = \frac{F_{oscillator}}{(Top + 1) \times N}$$

  Despejando N y reemplazando los datos se obtiene N=1250 , como este no es un valor estándar , se elegirá el valor estándar más próximo que sería N=1024 .Obteniendo de esta manera Fgenerated wave= 61.06Hz
  Para obtener la intensidad del color se utilizará la siguiente fórmula que representa cuánto tiempo del periodo la señal estará en alto

  $$\text{Duty Cycle} = \frac{Top - OCR1x}{Top + 1} \times 100$$

  OCR1x debería valer 255 para encender un color al máximo , de esta manera la señal se mantendrá en bajo porque duty cycle sería 0.

- Configuración

  Se utilizarán los registros TCCR1A y TCCR1B que sirven para configurar el modo de funcionamiento del timer, dado que el TIMER 1 es de 16 bits, tiene disponibles 16 modos de funcionamiento detallados a continuación:

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

El modo a utilizar es el 5 Fast PWM con una resolución de 8 bits y un tope de 255. Además como explicamos anteriormente el modo Fast PWM puede generar señales en modo invertido y no invertido. Para seleccionar el modo debemos configurar el registro TCCR1A:

COM1A1	COM1A0	Description
0	0	Normal port operation, OC1A disconnected
0	1	Toggle OC1A on compare match
1	0	Clear OC1A on compare match
1	1	Set OC1A on compare match

Se utilizará el modo invertido, por lo que se debe setear los bits COM1A1 y COM1A0 en 1, lo mismo debemos hacer para el canal B.
Luego el valor del preescalador a utilizar es el de 1024.

- Pseudocódigo Timer1(){
  //Para puertos PB1 y PB2
  Configurar Timer1 en modo PWM fast 8bits
  Configurar modo invertido
  Configurar preescaler en 1024
  }

2.2.2 PWM por software para color rojo

Se debe utilizar la salida PB5 del MCU pero esta salida no cuenta con PWM , por lo tanto se debe generar por software de manera bloqueante

- Cálculos

Para realizar la temporización de la señal se utilizara el TIMER0 en modo ctc. Para calcular cual es el preescaler adecuado , se utilizará la siguiente fórmula :

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

Despejando N y reemplazando los datos(Foscilador=16MHz y Fgenerated=50 Hz) se obtiene N=1250 , como este no es un valor estándar , se elegirá el valor estándar más próximo que sería N=1024 .Obteniendo de esta manera Fgenerated wave= 61 Hz. Y por simplicidad se utilizará el contador OCR0A=255.

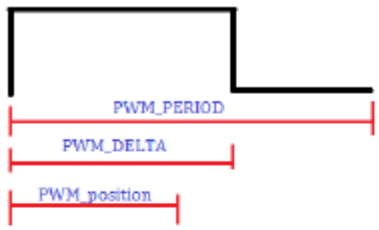
$$\text{Duty Cycle} = \frac{255 - \text{OCR0}}{256} \times 100$$

OCR0 deberá valer 255 para encender un color al máximo , de esta manera la señal se mantendrá en bajo porque duty cycle sería 0.

- Configuración

  Para la configuración del modo PWM por software se utilizó el código proporcionado en la teoría ,cambiando PWM_DELTA por una variable uint8_t para poder modificar su valor de acuerdo a la lectura del potenciómetro y también se realizará el cambio en PWM_PERIOD por 255 en vez de 100.

  A continuación se muestra el código proporcionado en la teoría.




```
#define PWM_PERIOD   100
#define PWM_DELTA    25
#define PWM_OFF      PORTB &=~(1<<PORTB0)
#define PWM_ON       PORTB |=(1<<PORTB0)
#define PWM_START    DDRB |=(1<<PORTB0)

 // interrupción periodica cada 1ms
ISR (TIMER0_COMPA_vect)
{
  PWM_soft_Update();
}
```

```
void PWM_soft_Update(void){
static uint16_t PWM_position=0;
    if (++PWM_position>=PWM_PERIOD){
        PWM_position=0;
    }
    if (PWM_position<PWM_DELTA){
    PWM_ON;
    } else{
    PWM_OFF;
    }
}
```

- Pseudocódigo Timer0_Init(){

  Configurar timer0 en modo ctc para ser utilizado en PWM por software

  Establecer preescaler 1024

  Inicializar OCR0A

  Habilitar interrupción por comparación

  Habilitar interrupciones globales

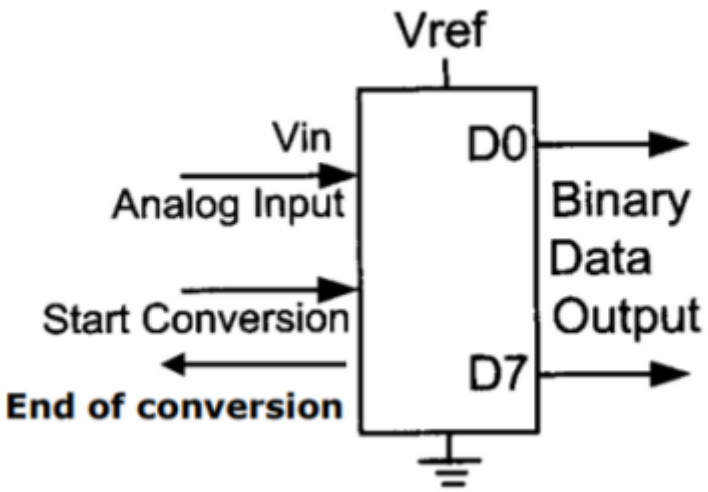
  }

- Pseudocodigo PWM_soft_Update(){
  Declarar variable estatica de 16 bits para mantener la posición del PWM
  Incrementar en 1 la posición PWM
  Si la posición es mayor o igual al periodo
  Reiniciar posición de PWM
  Si la posición es menor al valor seteado
  Encender led rojo
  sino
  Apagar led rojo
  }

2.3 Periférico ADC

2.3.1 Introducción

El periférico ADC permite convertir magnitudes analógicas a valores digitales (números binarios) en registros de 10 bits o de 8 ,que luego pueden ser procesados.

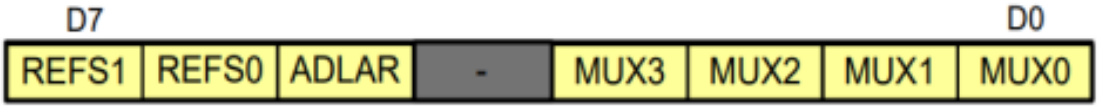


2.3.2 Implementación ADC

Para poder realizar la implementación del ADC, hay que tener en cuenta que el resultado es un valor de 10 bits que se encuentra en el registro ADC y primero se debe inicializarlo seteando ciertos valores en sus registros de configuración.

El primer registro a configurar será el registro **ADMUX**:

ADMUX – ADC Multiplexer Selection Register



Este registro del ADC nos permitirá configurar cosas como, la fuente de voltaje de referencia y el canal analógico en el que se va a convertir.
Esta configuración nos permitirá tener una **tensión** de referencia igual a **VCC** y el **canal** análogico será el número **tres**.

El ADC nos permite configurar también un tipo de justificación.
La justificación es la alineación del resultado del ADC, si ADLAR es igual a 1 (Left - Justified), el resultado de la conversión se almacenará en los 8 bits más significativos (ADCH) y los 2 bits menos significativos se desecharán. Si ADLAR es igual a 0 (Right - Justified) el resultado se almacenará en los 2 bits menos significativos (ADCL) y los 8 bits más significativos se almacenarán en ADCH.

Otro registro a tener en consideración a la hora de utilizar el ADC es el **ADCSRA**, el cual nos sirve para poder configurar un preescalador a elección y habilitar el ADC.

ADCSRA

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN- Bit7 ADC Enable
This bit enables or disables the ADC. Writing this bit to one will enable and writing this bit to zero will disable the ADC even while a conversion is in progress.

ADSC- Bit6 ADC Start Conversion
To start each coversion you have to write this bit to one.

ADATE- Bit5 ADC Auto Trigger Enable
Auto Triggering of the ADC is enabled when you write this bit to one.

ADIF- Bit4 ADC Interrupt Flag
This bit is set when an ADC conversion completes and the Data Registers are updated

ADIE- Bit3 ADC Interrupt Enable
Writing this bit to one enables the ADC Conversion Complete Interrupt.

ADPS2:0- Bit2:0 ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

Los niveles del preescalador que posee el ADC son los siguientes:

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

Habilitamos el ADC estableciendo un 1 en el bit más significativo del registro ADCSRA y lo configuramos con un preescalador de 128.

2.3.3 Implementación del código

Existen dos opciones para programar el ADC, por interrupción o por polling,en este proyecto se eligió programarlo utilizando la técnica de polling implementándolo en una función llamada ADC_LEER().

La misma funciona de la siguiente forma:

● Se inicializa la conversión
● Se espera que la misma finalice (este paso es bloqueante ya que el programa debe quedarse esperando)
● Se borra el flag ADIF, el cual si está activo indica que se está realizando una conversión. Por eso es importante para el programador borrarlo de manera manual cuando se utiliza una programación de ADC por polling.
● El resultado de la conversión es almacenado en una variable para su posterior uso.

Pseudocódigo del ADC

```
ADC_init {
        Setear el puerto como entrada analogica
        Vref= Avcc, Selecciono ADC3
        Habilitar ADC
        Configura el prescaler en 128
        Ajuste a la izquierda del resultado del ADC
}

uint8_t ADC_LEER() {
        Iniciar conversión
        Esperar a que termine la conversación
        Reiniciar el flag
        Retornar el valor
}
```

2.4 Terminal

Este archivo es el encargado de inicializar UART0 para empezar con la transmisión y recepción de datos , también de clasificar que tipo de comando fue ingresado en la terminal virtual y de imprimir el valor del potenciómetro a modo informativo.

2.4.1 Pseudocodigos

- Terminal_Init(){
  Declarar 4 variables (colorActual,comando[2],RX_Buffer y nuevoComando);
  Establecer un baud rate de 9600 bps con formato de datos 8N1.
  Habilitar tanto la transmisión como la recepción de datos en el puerto serie
  Habilitar la interrupción para la recepción de datos.
  }
- Terminal_LeerComando(){
  Si se ingresó R por led rojo

Retorna 1
Sino si se ingresó G por el led verde
Retorna 2
Sino si se ingresó B por el led azul
Retorna 3
}

- Terminal_ImprimirValor(num){
  Declarar dos variables e inicializarlas en cero(c y aux)
  Guardar el primer dígito del número pasado por parámetro
  Si c es distinto de cero
  Poner aux en 1
  Esperar a que el buffer esté libre
  Enviar c convertida a string para no perder su valor
  Guardar el segundo dígito del número pasado por parámetro
  Si c es distinto de cero
  Poner aux en 1
  Esperar a que el buffer esté libre
  Enviar c convertida a string para no perder su valor
  si no si aux=1
  Esperar a que el buffer esté libre
  Enviar c convertida a string para no perder su valor
  poner aux en cero
  Guardar el tercer dígito
  Esperar a que el buffer esté libre
  Enviar c convertida a string para no perder su valor

2.5 Main

En este archivo se llaman a todas las librerías desarrolladas anteriormente. Este es el encargado de inicializar todos los periféricos y timers . Además se encarga de actualizar el PWM de cada color

2.5.1 Pseudocódigo

Configurar como salida el led RGB
Inicializar UART0
Inicializar Timer0 y Timer1
Inicializar ADC
Declarar variables para poder setear OCR1A, OCR1B y el delta de PWM por software
Bucle infinito
Si nuevoComando {
Obtener el color a modificar (es un uint8_t)

Sí color es 1 significa que es rojo

Leer la posición del potenciómetro

Mostrar en pantalla dicha posición

Setear el valor del delta con la posición leída y guardada

Reiniciar nuevoComando para que no se imprima varias veces lo mismo

Mostrar menu

Sí color es 2 significa que es verde

Leer la posición del potenciómetro

Mostrar en pantalla dicha posición

Setear el valor de OCR1B con la posición leída

Reiniciar nuevoComando para que no se imprima varias veces lo mismo

Mostrar menu

Sí color es 2 significa que es azul

Leer la posición del potenciómetro

Mostrar en pantalla dicha posición

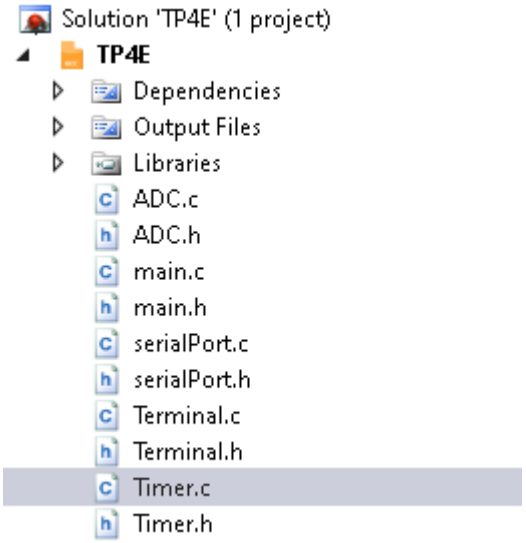
Setear el valor de OCR1A con la posición leída

Reiniciar nuevoComando para que no se imprima varias veces lo mismo

Mostrar menu

2.6 Modularización

A continuación se muestra una imagen donde se podrán observar los distintos archivos utilizados para la solución del problema.

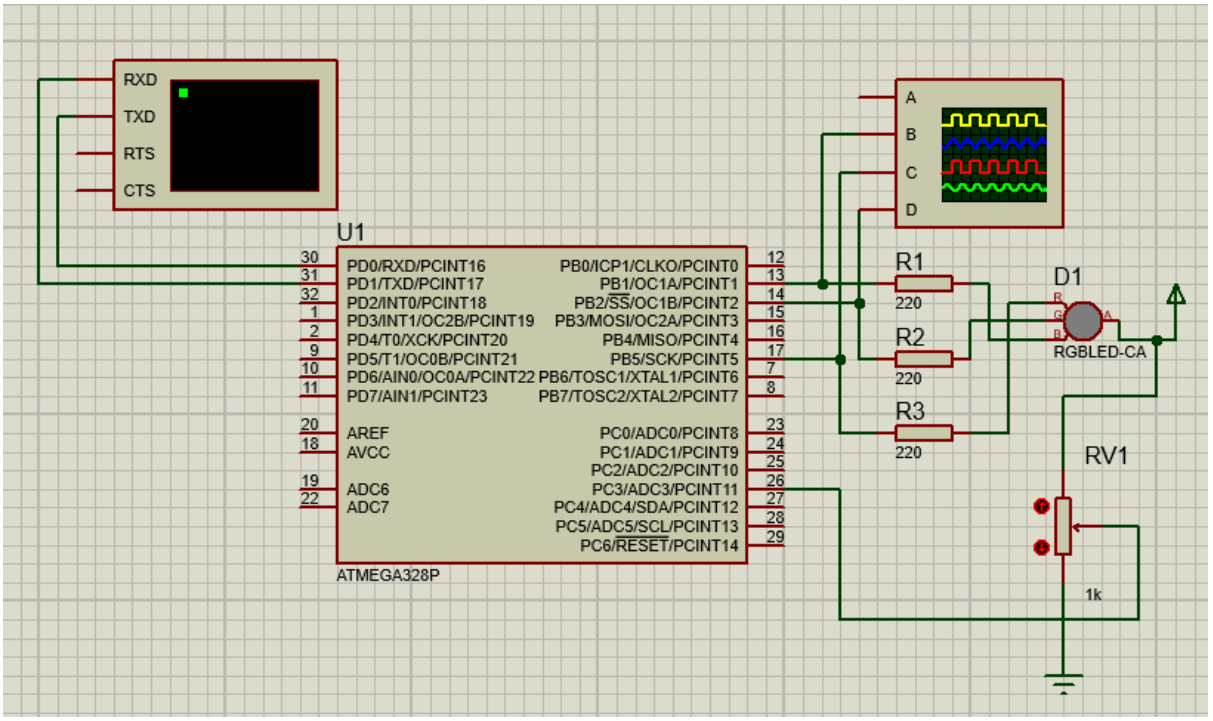


- serialPort es la librería brindada por la cátedra para configurar y utilizar el puerto serie del MCU
- ADC cuenta con dos funciones , la primera ADC_Init() es la encargada de configurar el periférico ADC y la segunda función es ADC_LEER() , esta es la encargada de retornar los datos del potenciómetro
- Timer se encarga de configurar el timer 0 en modo ctc (Timer0_Init()) , timer 1 en modo PWM (Timer1_Init()),actualizar el valor delta del modo PWM por software(PWM_SET_ROJO(delta) y por último implementa el modo PWM por software (PWM_soft_Update()).
- Terminal utiliza la librería serialPort.h para poder inicializar UART0(Terminal_Init()) , leer el comando ingresado (Terminal_LeerComando()) y retornar el valor asignado para dicho comando , convertir el valor del potenciómetro en ascii y mostrarlo en la terminal (Terminal_ImprimirValor(num)) y por ultimo devuelve el valor del comando ingresado(Terminal_Color())

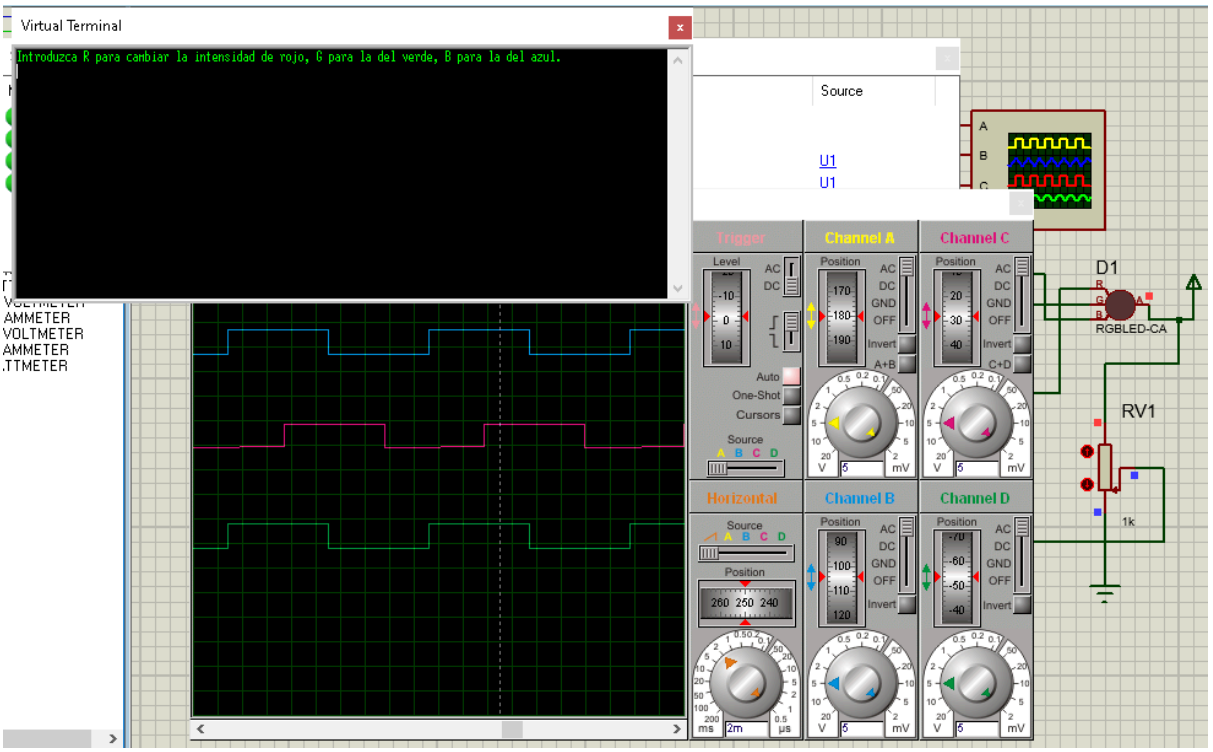
3.Validación y conclusiones

3.1 Validación

A continuación se muestra la conexión final entre el MCU , terminal virtual, led RGB y potenciómetro.Además se utilizó un oscilador para poder observar el funcionamiento de las señales PWM .

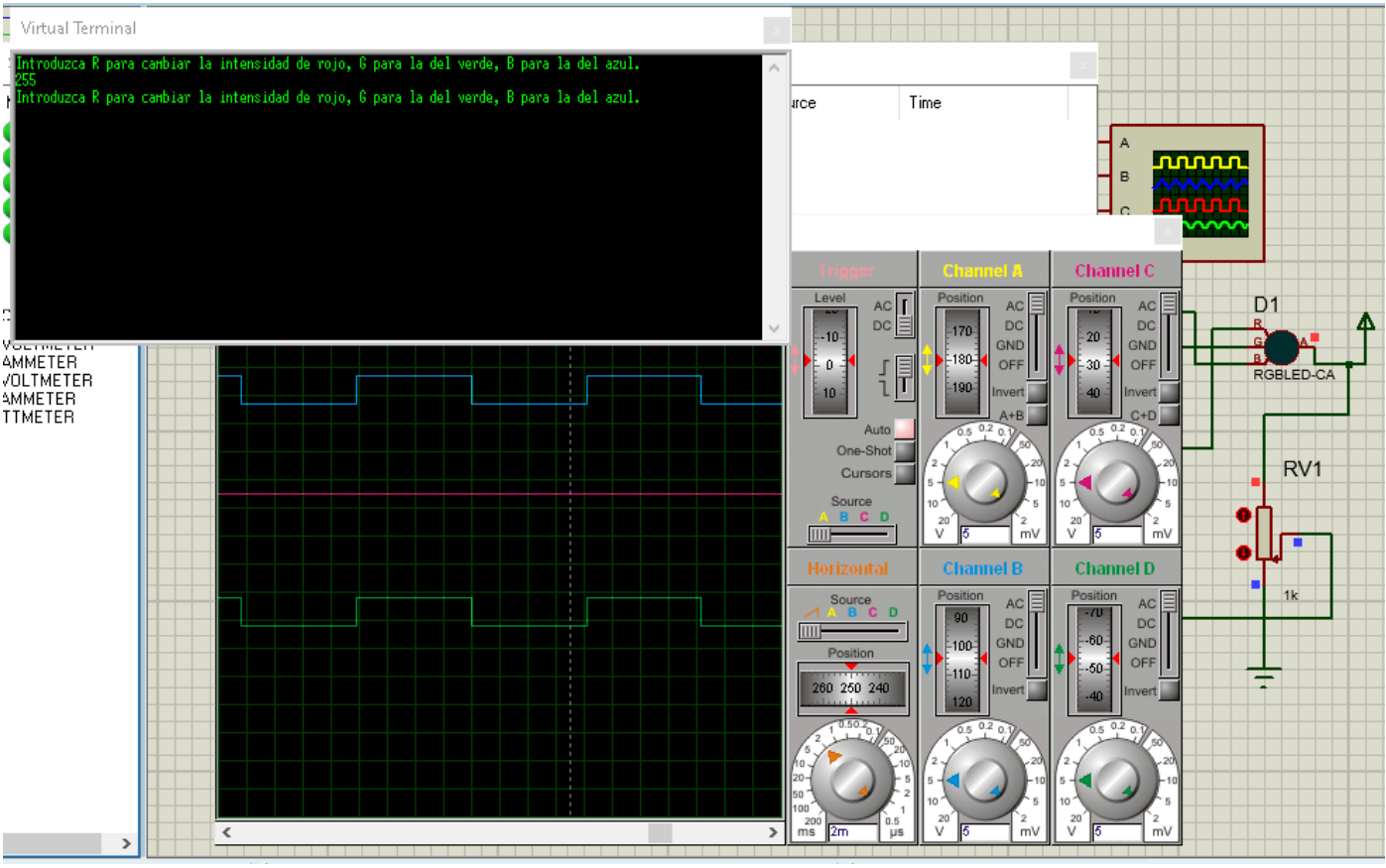


Apenas iniciamos la simulación en Proteus la terminal muestra el menú de opciones y el osciloscopio muestra que las señales generadas tienen el mismo ciclo de trabajo, esto tiene sentido ya que todos los colores inicialmente están configurados para brillar a mitad de intensidad. Se puede notar un desfasaje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

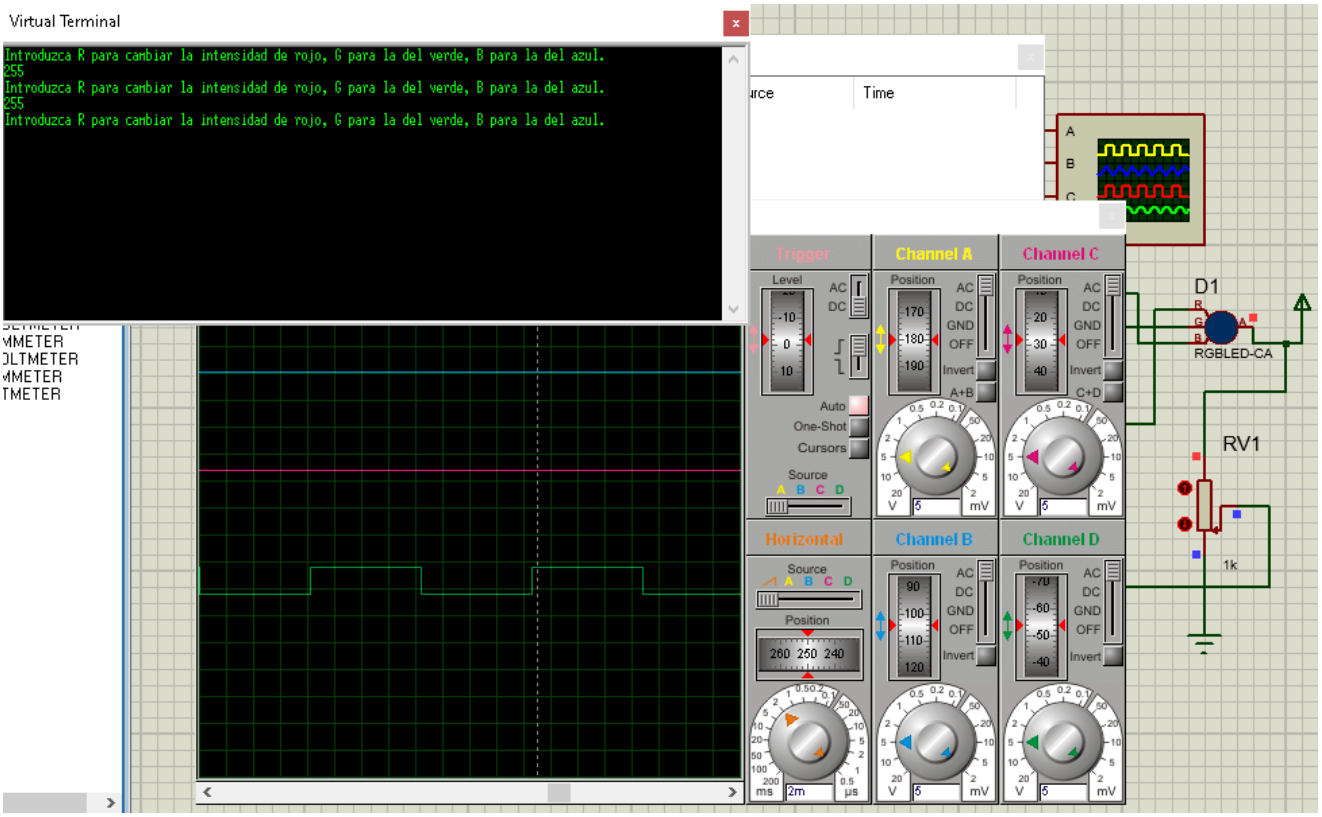


A continuación se ingresó el comando R para entrar en la configuración del color Rojo . Se puede observar en la terminal que el valor del potenciómetro es 255 y según los cálculos

realizados anteriormente esta se pondrá en bajo. Los otros colores no se ven afectados ya que su señal es generada internamente por el MCU.



Si apretamos el comando B obtendremos la señal en bajo por el mismo motivo que en el led rojo



Debido a un error en proteus o en el código realizado no se pudo realizar más pruebas ya que el potenciómetro solo imprime la posición 255.

3.2 Conclusión

El código realizado cumple parcialmente con la consigna , ya que , como se dijo anteriormente , se encuentra un error ya sea en el simulador o en el código que impide su correcto funcionamiento.

4.GitHub

https://github.com/LucasCarba/CDyM-TP4