Circuitos Digitales y Microcontroladores - UNLP
29/7/2024

Trabajo Práctico N°4
Control de un LED RGB

Grupo 4:

Graciano Gonzalez, Fausto. Legajo: 03028/2
Zanetti, Bruno. Legajo: 02975/5

FACULTAD DE INGENIERÍA
UNIVERSIDAD NACIONAL DE LA PLATA

Índice

1. Enunciado general

1.1 Enunciado

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 220ohms 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.

1.2 Interpretación

El objetivo del programa es controlar la intensidad y el color de un LED RGB mediante la técnica de modulación por ancho de pulso (PWM). El LED RGB está conectado a los terminales PB5, PB2 y PB1 del microcontrolador (MCU) a través de resistencias limitadoras de 220ohms, y está configurado en forma de ánodo común. La simulación se realizará en el software Proteus, utilizando el dispositivo RGBLED-CA.

Se deben generar tres señales PWM en los terminales de conexión del LED (PB5, PB2 y PB1) con una frecuencia de al menos 50 Hz y una resolución de 8 bits cada una. Esto permitirá controlar la intensidad de cada uno de los colores del LED RGB (rojo, verde y azul) de manera precisa.

El programa debe permitir al usuario seleccionar la proporción de color de cada LED, variando de 0 a 255, para obtener un color resultante al combinar las intensidades de los tres colores básicos. Para ello, se implementará una interfaz serie UART0 que recibirá comandos para activar la modificación de la proporción de color deseada. Los comandos serán: 'R' para modificar el rojo, 'G' para el verde y 'B' para el azul.

Además, se utilizará un potenciómetro conectado al terminal ADC3 del kit para modificar el brillo del color seleccionado. El valor leído del potenciómetro se usará para ajustar el ciclo de trabajo (duty cycle) del canal PWM correspondiente, permitiendo así ajustar la intensidad del LED en tiempo real.

1.3 Esquema eléctrico

1.3.1 MCU

Mediante el software Proteus, se incorporaron los componentes necesarios mediante el menú de búsqueda “Library > Pick Parts”, en este caso se selecciona el MCU y se agrega en la hoja de proyecto. En este trabajo se utilizó el “ATmega328P”. Se muestra la representación del MCU en la Figura 1.3.1 a continuación.

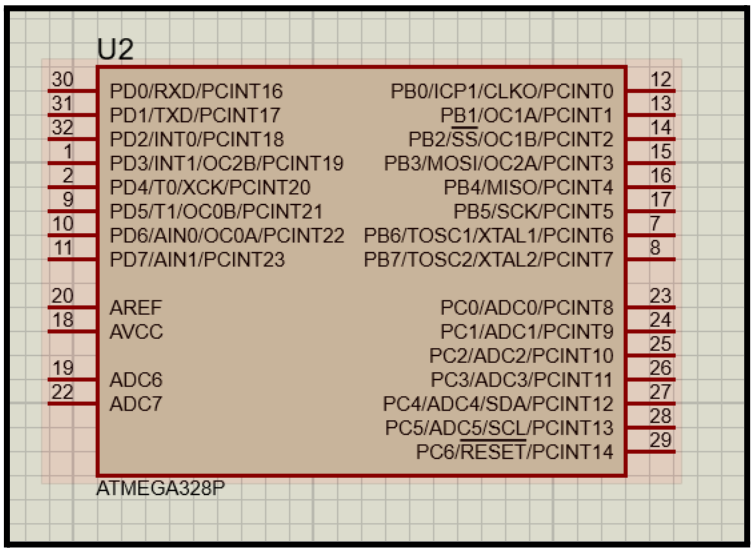

Figura 1.3.1 Microcontrolador ATmega 328P en Proteus.

1.3.2 RGBLED-CA

Se realizó el agregado del led RGBLED-CA, el cual se conectó a los terminales PB5 para el color rojo, PB2 para el color verde y PB1 para el color azul (RGB). El mismo se conector en PULLDOWN con unas resistencias externas de 220Ω. Se muestra su conexión en la Figura 1.3.2.

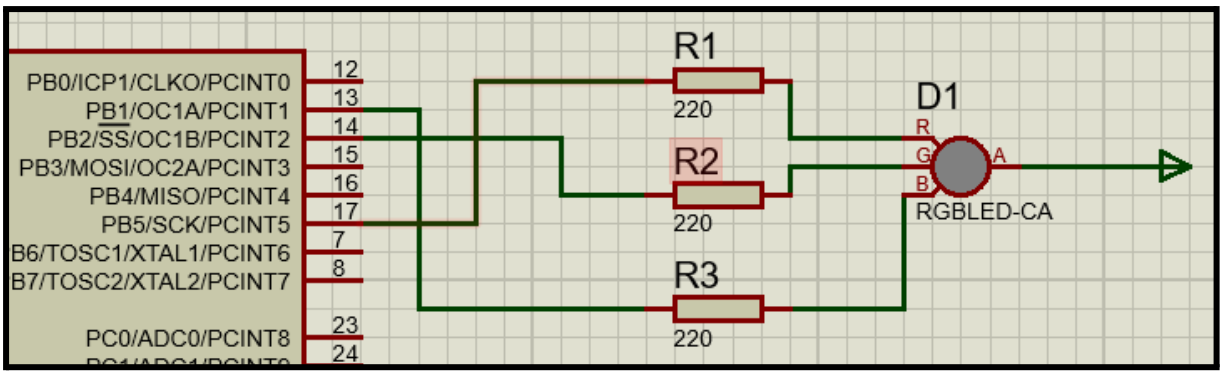

Figura 1.3.2 Led RGB en Proteus.

1.3.3 POT-HG

Se realizó el agregado del Potenciómetro, mediante el dispositivo POT-HG, el cual fue conectado al MCU mediante el terminal PC3, con sus respectivos VCC y GND en cada extremo. Se muestra su conexión en la Figura 1.3.3.

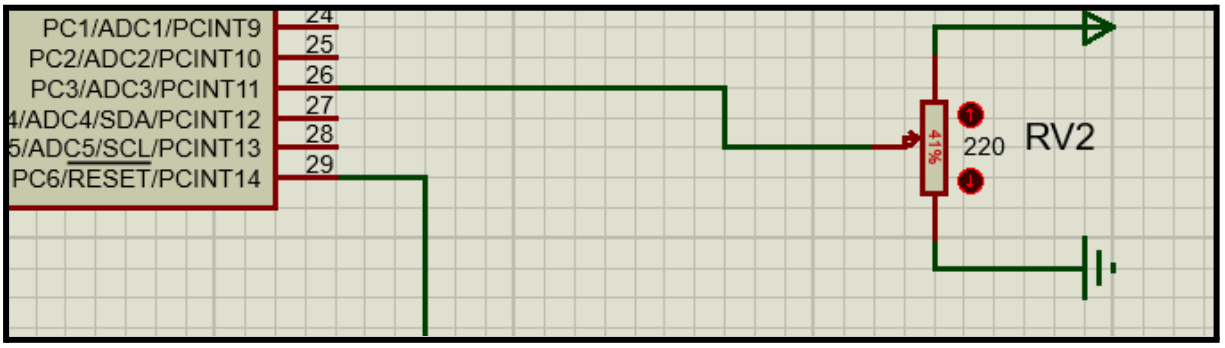

Figura 1.3.3 Potenciómetro en Proteus.

1.3.4 UART0

Se realizó el agregado del UART, mediante el dispositivo “VIRTUAL TERMINAL” del Proteus, se conectó su pin RXD con el PORTD1/TXD del MCU y su pin TXD con el PORTD0/RXD del MCU.

Para dar funcionamiento al mismo utilizamos la librería “serialPort.h” y “serialPort.c”, proporcionada por la cátedra.

Se muestra su conexión en la Figura 1.3.4.

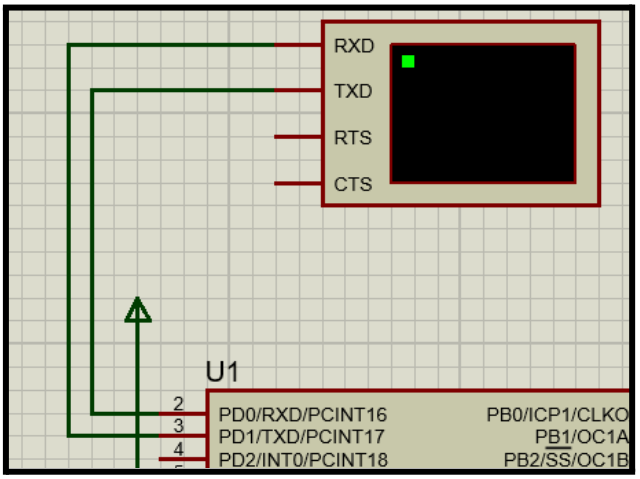

Figura 1.3.4 Conexionado del UART0.

1.4 Resolución del problema

1.4.1 Implementación PWM por software

Para generar la señal PWM del LED ROJO se creó una librería llamada “PWMsoft.c” y “PWMsoft.h”.

En la misma se implementó una función “PWM_update()”, que genera y actualiza la señal PWM, asignando un periodo y un valor de intensidad del color rojo, el cual limita el ciclo de trabajo.

Aquí un pseudocódigo de la función:

```
void PWM_update(){
    static uint16_t PWM_position=0;
    si (la posicion del PWM incrementada en 1 es mayor o igual que el periodo
de PWM){
      reinicio PWM_position en 0;
    }
    si (PWM_position es menor al valor de intensidad del led ROJO){
    se ingresa nivel bajo al puerto PB5;
}
    else{
       se ingresa nivel alto al puerto PB5;
    }

}
```

1.4.2 Configuración del TIMER0

Para la implementación del timer se utilizó el TIMER0, se configuró para el uso del modo CTC () asignando al registro TCCR0A=(1<<WGM01), usando un Preescaler de 8 asignado al registro TCCR0B=(1<<CS01), el cual fue elegido empíricamente, y suponiendo que la frecuencia del MCU es de 16Mhz.

Para generar una interrupción con una frecuencia mayor a 50hz, realizamos la siguiente ecuacion: (256 * X <= 1 / 50hz), lo que resulta en que X<= 0,0781ms.

Ciclos necesarios para generar interrupciones =($0,0781ms * 16MHz)/ Preescaler$

El resultado de la ecuación es de C = 156, para que la interrupción se de en estos ciclos, al registro OCR0A le ingresamos C-1(155).

Además se creó una ISR en la cual se insertó la función “PWM_update()” explicada anteriormente y por último para habilitar las interrupciones del timer, al registro se le asigna TIMSK0|=(1<<OCIE0A);

1.4.3 Configuración del TIMER1, LEDS VERDE y AZUL

Para la implementación del timer se utilizó el TIMER1, se configuró para el uso del modo 5 fast PWM invertido asignando al registro TCCR1A |= (1 << COM1A1) | (1 << COM1A0) | (1 << COM1B1) | (1 << COM1B0) | (1 << WGM10), usando un Preescaler de 8 asignando al registro TCCR1B |= (1 << WGM12) | (1 << CS11), el cual fue elegido empíricamente, y suponiendo que la frecuencia del MCU es de 16Mhz.

Esto fue elegido para implementar las señales de PWM de los colores verde y azul, utilizando el modo 5 de 8 bits, los que nos da un periodo de 256, lo que coincide con los niveles de intensidad requeridos.
Asignando la intensidad de los mismos a los registros 0CR1A para el color azul y OCR1B para el color verde, mediante una función “set_pwm(green, blue)”.

Aqui un pseudocódigo de la función:

```
void set_pwm(uint8_t green, uint8_t blue) {
   asignó al registro comparador OCR1B el valor de green // PB2
     asignó al registro comparador OCR1A el valor de blue  // PB1
}
```

1.4.4 Implementación del periférico ADC

Para la implementación del periférico ADC se crearon dos funciones llamadas “adc_init()” y “adc_read()”.
En la primera se realizó la configuración del mismo, asignando al registro ADMUX |= (1 << REFS0) | (1 << MUX0) | (1<<MUX1) | (1<<ADLAR), lo que habilita a usar el terminal ADC3 del MCU, da una Vref igual al Vcc y justifica a izquierda al registro ADC con ADLAR en “1”, lo que permite usar solo los 8 bits más significativos del registro (ADCH). Ademas se asigno al registro ADCSRA |= (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1)| (1 << ADPS0), lo que habilita el ADC y setea un preescaler de 128 al mismo.
En la segunda función se realizó la lectura del mismo, en la cual al ser llamada inicia la conversión, espera a que la misma termine mediante el flag ADIF, y una vez finalizada borra el flag, para poder habilitar una próxima lectura, y retorno parte alta del registro adc (ADCH).

Aquí un pseudocódigo de la función:

```
uint16_t adc_read() {
   inició la conversión
   mientras que el flag ADIF no sea “0” espero
   borro flag ADIF asignando “1” al mismo
   retorno parte alta del registro adc
}
```

1.4.5 Implementación del periférico UART

Para la implementación del periférico UART se utilizó las librerías brindadas por la cátedra “serialPort.c” y “serialPort.h”, las cuales tienen dentro distintas funciones que permiten enviarle a la consola mensajes, y otras funciones relacionadas con el control del buffer. Para poder seleccionar el color a modificar se utilizaron 3 caracteres, mediante la pulsación de la tecla “R” se cambia la intensidad del color rojo, mediante la pulsación de la tecla “G” se cambia la intensidad del color verde y mediante la pulsación de la tecla “B” se cambia la intensidad del color azul, para esto se agregó a la librería de la cátedra una interrupción “ISR(USART_RX_vect)”, la cual tiene como función asignar el carácter capturado en la UART a una variable

creada por nosotros llamada “selected_color” que luego va a ser consultada dentro del código main. A su vez, el cambio de intensidad del color seleccionado que se cambia utilizando el potenciómetro, iría modificando indefinidamente la intensidad, para solucionar esto se implementó un “enter” el cual nos sirve como barrera para no modificar valores luego de haber llegado a la intensidad deseada de un color, este se realiza presionando la tecla “E”. Aquí un pseudocódigo de cómo funciona la consulta de la variable selected_color dentro del código Main:

```
 Mientras(1)
       {
        if (la tecla presionada no es el Enter 'E' ) {

       le asignamos a pwm_value la lectura del potenciometro con el ADC.

       switch (selected_color) {
             en caso de que sea R:
             le asignamos a red el valor de pwm_value;
             sale de switch;
             en caso de que sea G:
             le asignamos a green el valor de pwm_value;
             sale de switch;
             en caso de que sea B:
             le asignamos a blue el valor de pwm_value;
             sale de switch;
       }
// Establecer el valor PWM
       set_pwm(green, blue);
       }
     }
```

Aqui un pseudocódigo de la ISR:

```
ISR(USART_RX_vect) {
       asigno a variable selectede_color el registro UDR0;
       }
}
```

2. Validación

2.1 Validación tiempo del timer

Se verificó que el tiempo de interrupción para actualizar el PWM del TIMER0 coincida con el calculado con la ecuación arriba, el cual es igual a 0,0781 ms que es aproximadamente igual a 78.10 us.

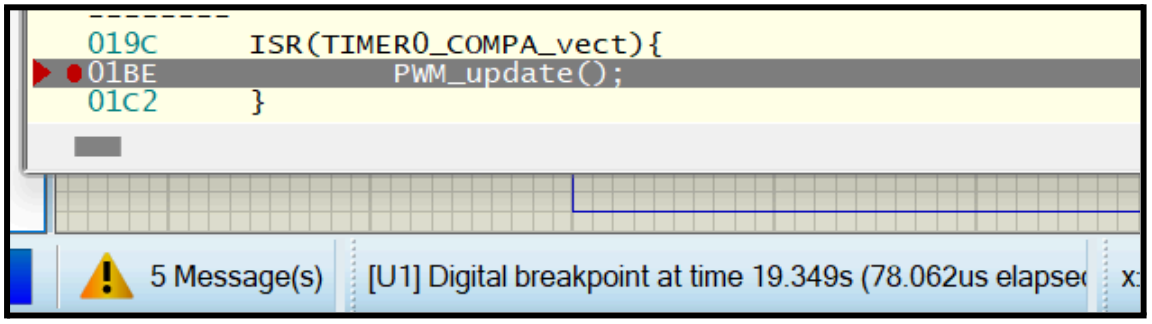

Figura 2.1.2 Tiempo del simulador Proteus.

2.2 Simulación completa con video

Adjuntamos en el siguiente link un video mostrando la utilización del LED RGB, y la variación de intensidad de los tres colores principales, al insertar la tecla requerida para modificarlos.

SIMULACION VIDEO TP4

3. Conclusión

3.1 Desarrollo de la conclusión

El programa que hicimos logró cumplir con todos los requisitos que nos pidieron. Fue un proyecto interesante porque nos permitió aplicar todo lo que aprendimos en clase para controlar la intensidad de los colores de un LED RGB, lo cual no es tan fácil de hacer. Usamos un enfoque modular para dividir el programa en partes más pequeñas, lo cual nos ayudó un montón a encontrar y arreglar errores sin tener que reescribir todo desde cero.

Durante el desarrollo, nos topamos con un "problema técnico", elegimos usar el modo Fast PWM en los timers en modo invertido. Esto hizo que el LED verde y azul nunca se apagaran en su totalidad debido a cómo estaba configurado el modo, pero como era algo relacionado a la resolución del modo en sentido al ciclo de trabajo y no por tema de implementaciones nuestras, no nos pareció algo erróneo o algo a modificar ya que no se especificó nada en los incisos.

La verificación del tiempo del Timer0 fue clave para asegurarnos de que todo funcionara según lo esperado. Nos dio la tranquilidad de que nuestra configuración estaba en línea con lo que nos pedían hacer en el proyecto.

En resumen, este proyecto no solo fue un desafío técnico divertido, sino que también nos ayudó a aprender mucho.

4. Bibliografía

4.1 Bibliografía

❖ Atmel Corporation (2015). ATmega328P Datasheet.
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

5. Anexo

5.1 Anexo

Adjuntamos en el siguiente link un PDF con el código Main C del programa.

maintp4.pdf

Adjuntamos en el siguiente link el repositorio GITHUB con todos los archivos del programa.

Link GITHUB codigo.