ECE 263: Embedded System Design

LAB 8: USART/PWM

We certify that this work is original and not a product of anyone's work but our own.

<Bishoy Mikhail>:_____

<Leandro Neves>:_____

Submitted: April 18th, 2023

Due by: April 18th, 2023

Graded by: _____ Date: _____

Contents

1 Abstract **2**

2 Introduction **2**

3 Methods **2**

3.1 Interface With puTTY 2

3.2 Control of The Built-in RGB LED 2

3.3 Important Functions 3

3.3.1 PWM (Pulse Width Modulation) 3

3.3.2 SetLED( ) Function 3

3.3.3 uart_putchar( ) Function 3

3.3.4 uart_getchar( ) Function 3

3.3.5 Command Processing 4

4 Results and Discussion **4**

5 Conclusion **5**

6 Recommendations **5**

7 Reflection **5**

8 Appendix **6**

List of Figures

1 LAB 8 Project Testing 4

List of Tables

List of Files

1 LAB 8 main.c File 6

2 LAB 8 mBuffer.c File 11

3 LAB 8 mBuffer.h File 13

1 Abstract

This lab report details the design and implementation of LAB 8 which aimed to control an RGB LED using a microcontroller-based system. The system includes an ATmega328PB microcontroller, an RGB LED, and a terminal emulation software called PuTTY for communication. The microcontroller is programmed to generate control signals for the RGB LED based on user input received through PuTTY. The RGB LED emits red, green, and blue light, which can be combined to create various colors. The project demonstrates the implementation of a microcontroller-based RGB LED system and the control of LED colors using terminal emulation software, providing insights into embedded system design.

2 Introduction

In this report, the results of Lab 8 will be presented and analyzed. The objective of the lab was to implement and control an RGB LED using a microcontroller-based system. This system received input from the user using PuTTY, a terminal emulation software used for communication. The microcontroller was programmed to generate control signals for the RGB LED based on user input received through PuTTY. This report presents the design and implementation of the microcontroller-based RGB LED system, including the hardware and software aspects, and discusses the results and insights gained from the project.

3 Methods

3.1 Interface With puTTY

The communication between the ATmega328PB microcontroller and the host computer was established using PuTTY, a terminal emulator software. PuTTY was configured with the appropriate serial communication settings, such as the baud rate, data bits, stop bits, and parity, to establish a reliable serial connection between the microcontroller and the computer. The microcontroller transmitted data to PuTTY, which was displayed on the computer's screen, allowing for real-time monitoring and control of the project.

3.2 Control of The Built-in RGB LED

The built-in RGB LED on the microcontroller's "shield"/accessory board was controlled using the microcontroller's PWM functionality. The red, green, and blue channels of the LED were

connected to PD6, PB1, and PB3 pins, respectively, while the common channel was connected to the PC3 pin. PWM settings, such as frequency, duty cycle, and waveform generation mode, were configured in the code to control the brightness and color output of the LED. Functions or routines were implemented in the code to update the duty cycle of the PWM signals for each color channel based on user input or other external factors, allowing for dynamic control of the LED's brightness and color output.

3.3 Important Functions

Several key functions were implemented in the software to control the RGB LED and interface with PuTTY. These functions were written in C language and utilized the ATmega328PB microcontroller.

3.3.1 PWM (Pulse Width Modulation)

PWM is a technique used to control the intensity of LEDs by varying the duty cycle of a square wave signal. In this code, PWM is used to control the intensity of the Red, Green, and Blue LEDs using three different timers (TCCR0A, TCCR1A, and TCCR2A) with Fast PWM mode. The function InitPWM() initializes the timers with appropriate settings for PWM operation.

3.3.2 SetLED( ) Function

This function sets the intensity of the Red, Green, and Blue LEDs by updating the Output Compare Registers (OCR0A, OCR1A, and OCR2A) with the desired values for each color. The higher the value, the higher the intensity of the corresponding LED. This function is called with the values of intensity for each color (r, g, b) as arguments, and it sets the PWM output for each LED accordingly.

3.3.3 uart_putchar( ) Function

This function is called by the standard output stream to send characters to the PuTTY terminal via UART (Universal Asynchronous Receiver/Transmitter). It disables interrupts, enqueues the character to the transmit queue (TXQ), enables the UART data register empty interrupt (UDRIE0) and returns 0.

3.3.4 uart_getchar( ) Function

This function is called by the standard input stream to receive characters from the PuTTY terminal via UART. It waits for a character to be available in the receive queue (RXQ) and then dequeues

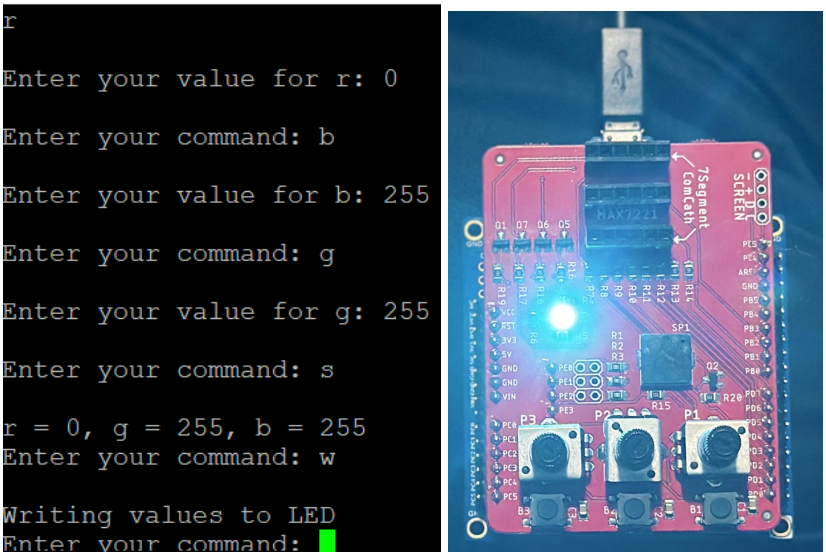
and returns the character. If the queue is empty, it spins in a loop until a character is available.

3.3.5 Command Processing

After reading the command from the user using fgets(), the code uses strcmp() function to compare the entered command with predefined commands (e.g., ”r”, ”g”, ”b”, etc.) to determine the desired action. Based on the command, the code sets the appropriate values for the Red, Green, and Blue intensities using the SetLED() function, and performs other actions accordingly.

4 Results and Discussion

After writing the full program and connecting the microcontroller to the computer via the debug wire, the code was successfully compiled and run. To test the functionality of the constructed project, a window was opened in puTTY where the user could control the behavior of the built-in LED. As seen in Figure 1a, red is set to 0, green is set to 255, and blue is set to 255 which would ideally result in a cyan color. This information was successfully written to the microcontroller where the color of the LED was indeed cyan as shown in Figure 1b. This test was repeated multiple times with different color combinations. Nevertheless, the project was a success and deemed fully functional. The next logical step with this project is to work on optimizing it and increasing its accuracy.



(a) puTTY input for cyan LED (b) Cyan Output on RGB LED

Figure 1: LAB 8 Project Testing

5 Conclusion

In conclusion, this project utilized an ATmega328PB microcontroller, a built-in RGB LED, and PuTTY as the interface for controlling the LED's color and brightness. The code was developed to configure the PWM pins for the Red (PD6), Green (PB1), and Blue (PB3) channels of the LED, allowing for precise control of the LED's color and intensity. The communication with PuTTY facilitated the user-friendly interface for sending commands to the microcontroller and controlling the LED's behavior. Important functions, such as the PWM configuration and LED control, were implemented to ensure the functionality and performance of the project. Overall, this project demonstrated the successful integration of the ATmega328PB microcontroller, built-in RGB LED, and PuTTY for controlling and manipulating the LED's behavior, showcasing the potential of embedded systems in practical applications.

6 Recommendations

I do not have many recommendations for someone doing Lab 8 for the first time. What I can say is to get very familiar with the functions, techniques, and code that are being used in this project. The code in Lab 8 is considerably advanced so it is easy to get lost and fall behind.

7 Reflection

Reflecting on this lab, it was very useful and a good learning experience. A lot of skills were learned and developed in this lab, which is great. Not only did it strengthen my programming skills, but it also gave me the opportunity to get more familiar with interfacing with puTTY.

8 Appendix

The following is the main.c file for LAB 8 written in C language

File 1: LAB 8 main.c File

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>  // for printf/fgets
#include <stdlib.h> // for atoi()
#include <string.h> // for strlen()
#include <ctype.h>  // for toupper()
#include "mBuffer.h"

#define BUFFSIZE 64

uint8_t txBuffer[BUFFSIZE];
uint8_t rxBuffer[BUFFSIZE];

MBUFFER TXQ;
MBUFFER RXQ;

// see page 146 of avr-libc-user-manual
// for info on why int and char are used rather than uint8_t
// and for more info on FDEV_SETUP_STREAM()

int uart_putchar(char c, FILE* stream);
int uart_getchar(FILE* stream);

FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_WRITE
    ↪ );
FILE mystdin  = FDEV_SETUP_STREAM(NULL,uart_getchar, _FDEV_SETUP_READ);

void uart_9600();

int uart_putchar(char c, FILE* stream)
{
  CLI();
```

```
  enqueue(&TXQ, c);
  SEI();
  UCSR0B |= 1<<UDRIE0;
  return 0;
}

int uart_getchar(FILE* stream)
{
  uint8_t t;
  while (isEmpty(&RXQ)) // while empty, spin
  ;
  CLI();
  t = dequeue(&RXQ);
  SEI();
  return t;
}

ISR (USART0_UDRE_vect)
{
  char t;
  if (isEmpty(&TXQ))
  {
    UCSR0B &= ~(1<<UDRIE0);
  }
  else
  {
    t = dequeue(&TXQ);  // could say UDR0 = dequeue(&TXQ);
    UDR0 = t;
  }
}

ISR (USART0_RX_vect)
{
  char t;
  uint8_t count;
  count = available(&RXQ);
  if (count > 3)
  {
```

```
    t = UDR0;
    if (t=='\r')
    {
      enqueue(&RXQ,'\r');
      enqueue(&RXQ,'\n');
      uart_putchar('\r',stdout);
      uart_putchar('\n',stdout);
    }
    else
    {
      enqueue(&RXQ,t);
      uart_putchar(t,stdout);
    }
  }
  else
  uart_putchar('\a',stdout);
}

void uart_9600()
{
  // 9600, 8, no parity, 1 stop
  UBRR0 = 103;
  UCSR0B = 1<<RXCIE0 | 0<<TXCIE0 | 1<<UDRIE0 | 1<<RXEN0 | 1<<TXEN0 |
     ↪ 0<<UCSZ02 ;
  UCSR0C = 0b00<<UPM00 | 0b00<<UPM00 | 0<<USBS0 | 0b11<<UCSZ00;
}

void InitPWM(){
  TCCR0B = (0b0<<WGM02)|(0b011<<CS00);//Pre-Scalar
  TCCR1B = (0b0<<WGM12)|(0b011<<CS10);//Pre-Scalar
  TCCR2B = (0b0<<WGM22)|(0b011<<CS20);//Pre-Scalar
  // Red
  TCCR0A = (0b11<<WGM00)|(0b10<<COM0A0);// Fast PWM
  DDRD |= (1<<6); // PD6
  // Green
  TCCR1A = (0b01<<WGM10)|(0b10<<COM1A0);// Fast PWM
  DDRB |= (1<<1); // PB1
  // Blue
```

```
  TCCR2A = (0b11<<WGM20)|(0b10<<COM2A0);// Fast PWM
  DDRB |= (1<<3); // PB3
}
void SetLED(uint8_t r, uint8_t g, uint8_t b)
{
  OCR0A = 255-r;
  OCR1A = 255-g;
  OCR2A = 255-b;
}


int main()
{

  InitPWM();
  DDRC = 0xFF;
  PORTC = 0xFF;
  char cmd[40];
  int val;
  bufInit(&TXQ, txBuffer, BUFFSIZE);
  bufInit(&RXQ, rxBuffer, BUFFSIZE);
  stdout = &mystdout;
  stdin = &mystdin;
  uart_9600();
  int cmdStatus = 0;
  printf("Welcome␣to␣COLOR␣Display");




  //_delay_ms(1000);
  /* Replace with your application code */
  while (1)
  {
    volatile int cmdStatus = 0;
    printf("\r\nEnter␣your␣command:␣");
    fgets(cmd, 40-1, stdin);
```

```
    cmd[strlen(cmd)-2] = '\0'; // get rid of crlf

    flush(&TXQ);
    if(strcmp(cmd,"r")==0) {cmdStatus = 1;}

    if(strcmp(cmd,"g")==0) {cmdStatus = 2;}

    if(strcmp(cmd,"b")==0) {cmdStatus = 3;}

    if(strcmp(cmd,"w")==0) {cmdStatus = 4;}

    if(strcmp(cmd,"s")==0) {cmdStatus = 5;}

    if(strcmp(cmd,"?")==0) {cmdStatus = 6;}


    switch(cmdStatus)
    {

      case 1:
        printf("\r\nEnter␣your␣value␣for␣r:␣");
        scanf("%d", &val);
        printf("\r\nYour␣value␣for␣r␣is:␣%d\r\n", val);
        int r = val;

        BREAK;

      case 2:
        printf("\r\nEnter␣your␣value␣for␣g:␣");
        scanf("%d", &val);
        printf("\r\nYour␣value␣for␣g␣is:␣%d", val);
        int g = val;

        BREAK;

      case 3:
        printf("\r\nEnter␣your␣value␣for␣b:␣");
        scanf("%d", &val);
```

```
        printf("\r\nYour value for b is: %d\r\n", val);
        int b = val;


        BREAK;

      case 4:
        printf("\r\nWriting values to LED");
        SetLED(r,g,b);

        BREAK;

      case 5:
        printf("\r\nr = %d, g = %d, b = %d", r, g, b);

        BREAK;

      case 6:
          printf("\r\n r - set red value \r\n g - set green value \r\n
            ↪ b - set blue value");
        flush(&RXQ);
        printf("\r\n w - write value to LED \r\n s - status \r\n ? -
          ↪ display commands");
        BREAK;


    }

  }
}
```

File 2: LAB 8 mBuffer.c File

```
// written by Ben Viall
// licensed for use by currently enrolled ECE263 students
// for class projects only.

#include "mBuffer.h"
```

```
void bufInit(MBUFFER *m,uint8_t *buf,uint8_t size)
{
  m->size=size;
  m->rxBuffer=buf;
  m->tail=m->head=m->bufferOverflow=0;
}

//free space available in buffer
uint8_t available(MBUFFER* m)
{
  if (m->bufferOverflow) return -1;
  return (m->tail+m->size-m->head)%m->size;
}

uint8_t isEmpty(MBUFFER* m)
{
  return (m->head==m->tail);
}

uint8_t peek(MBUFFER* m)
{
  if (m->head==m->tail) return -1;
  return m->rxBuffer[m->head];
}

void flush(MBUFFER *m)
{
  m->head=m->tail=m->bufferOverflow=0;
}

uint8_t dequeue(MBUFFER *m)
{
  m->bufferOverflow=0;
  if (m->head==m->tail) return -1;
  uint8_t nextChar = m->rxBuffer[m->head];
  m->head=(m->head+1)%m->size;
  return nextChar;
}
```

```
45
   void enqueue(MBUFFER *m,uint8_t newChar)
   {
      if ((m->tail + 1) % m->size == m->head)
      {
50      m->bufferOverflow = 1;
      }
      else
      {
        m->rxBuffer[m->tail] = newChar;
55      m->tail = (m->tail + 1) % m->size;
      }
   }
```

File 3: LAB 8 mBuffer.h File

```
   //written by Ben Viall
   // liceneced for use by currently enrolled ECE263 students
   // for class projects only.

5

   #ifndef MBUFFER_H
   #define MBUFFER_H
   #include <stdint.h>
   // buffer for at most 255 bytes.
10 typedef struct _mbuffer
   {
     uint8_t size;
     uint8_t *rxBuffer;
     uint8_t tail;
15   uint8_t head;
     uint8_t bufferOverflow;
   } MBUFFER;



20 void bufInit(MBUFFER *m,uint8_t *buf,uint8_t size);
   uint8_t available(MBUFFER* m);
   uint8_t isEmpty(MBUFFER* m);
   uint8_t peek(MBUFFER* m);
```

```
void flush(MBUFFER *m);
uint8_t dequeue(MBUFFER *m);
void enqueue(MBUFFER *m,uint8_t newChar);

#endif
```