МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра **САП**

ЗВІТ

до виконання лабораторної роботи №3

На тему: ”Робота з RGB світлодіодом ”

з курсу “Мікропроцесорні системи ”

Варіант - 4

Виконав:

Студент гр.ПП-31

Гаврилюк Н. О.

Прийняв:

Доцент кафедри САП

Головатий А.І.

ЛЬВІВ - 2025

Мета: дослідити роботу RGB світлодіода, закріпити навички роботи з цифровими портами, тактовими кнопками, формуванням ШІМ; навчитися програмувати режими роботи Arduino з використанням масивів.

Теоретичні відомості

Масив. це набір змінних, доступ до яких здійснюється за індексом. Для роботи з масивом його потрібно створити (оголосити). Способи створення (оголошення) масивів:

int myInts[6];

int myPins[] = {2, 4, 8, 3, 6};

int mySensVals[6] = {2, 4, -8, 3, 2}; char message[6] = "hello";

Можна оголосити масив без його ініціалізації, як *myInts*. У *myPins* оголошений масив без прямої вказівки його розміру. Компілятор сам порахує елементи та створить масив відповідного розміру. При створенні (ініціалізації) масиву можна вказати його розмір, як *mySensVals*. При оголошенні масиву типу *char*, в ньому необхідно місце для зберігання обов'язкового нульового символу, тому розмір масиву повинен бути на один символ більше.

Нумерація елементів масиву починається з нуля , тобто перший елемент масиву має індекс 0. Так *mySensVals[0] ==2*, *mySensVals[1] ==4*.

Це також означає, що в масиві з 10 елементів, останній елемент має індекс 9. Отже:

int myArray[10]={9,3,2,4,3,2,7,8,9,11};

// myArray[9] дорівнює 11

// myArray[10] буде помилка

Тому, необхідно бути уважним при зверненні до масивів. Звернення до елементу за межами масиву (коли зазначений індекс більше, ніж оголошений розмір масиву. 1) призведе до читання даних з комірки пам'яті, що використовується для інших цілей. Зчитування з цієї області призведе до збою в роботі програми. На відміну від BASIC або JAVA, компілятор C не перевіряє правильність індексів при зверненні до елементів масиву.

Запис значення до масиву. *mySensVals[0] = 10*. Зчитування запису з масиву . *x = mySensVals[4]*.

Робота з масивами часто здійснюється всередині циклів FOR, в яких лічильник циклу використовується як індексу кожного елемента масиву. **Наприклад**, програма налаштування режимів роботи портів Arduino може виглядати так:

```
const byte rgbPins[3] = {11,10,9}; void setup() {

for( byte i=0; i < 3; i++ ) pinMode( rgbPins[i], OUTPUT );

}
```

В одномірних масивах елементи визначаються просто порядковим номером. У двовимірних масивах (матриця або таблиця) кожен елемент має номер рядка та стовпця. Задається такий масив ось так:

```
// двовимірний масив, 5 рядків 10 стовпців byte myTable [5][10] ;

// матриця 3x3

byte myMatrix [3][3] = {

{   10,   11,   12   }   ,

{   13,   14,   15   }   ,

{   16,   17,   18   }   ,

} ;
```

Після останнього члена масиву можна ставити кому, це не призведе до помилки (приклад коду вище). У розглянутому вище двовимірному масиві елемент з адресою 0, 2 (рядок 0 стовпець 2) *myMatrix [0] [2]* має значення 12.

Дребезг контактів. це явище, що відбувається в електромеханічних пристроях (кнопках, реле, герконах, перемикачах, контакторах), що триває деякий час після замикання електричних контактів. Після замикання (натискання кнопки, включення реле і т.д.) відбуваються багаторазові неконтрольовані замикання та розмикання контактів за рахунок пружності матеріалів та деталей контактної системи. Перехідні процеси протікають дуже швидко (від 0,5 до декількох сотень мілісекунд). Тому їх не помічаємо, наприклад, коли включаємо світло в кімнаті. Лампа розжарювання не може змінювати свою яскравість з такою швидкістю. Але, обробляючи сигнал від кнопки на швидкому пристрої, як Arduino, повинні враховувати при програмуванні це явище.

Найпростішим способом боротьби з дребезгом кнопки є витримування паузи. Для цього робимо паузу 10-50 мілісекунд, як наведено у **прикладі** програми нижче.

```
int currentValue, prevValue; void loop ( ) {
```

currentValue = digitalRead (PIN_BUTTON) ; if (currentValue! = prevValue) {

delay (10) ;

currentValue = digitalRead (PIN_BUTTON) ;

}

prevValue = currentValue;

Serial. println (currentValue) ; }

Проблема з дребезгом настільки актуальна, що є спеціальні бібліотеки, в яких не потрібно організовувати очікування та паузи вручну. це все робиться всередині спеціального класу. Приклад популярної бібліотеки для боротьби з дребезгом кнопок. бібліотека Bounce.

Більш правильним способом боротьби з дребезгом є використання апаратного рішення, що згладжує імпульси з кнопки. Апаратний спосіб усунення дребезгу заснований на використанні RC фільтру або тригера Шмідта [9, 10].

RGB світлодіод на схемі лабораторного макету позначений HL7 (рис.3.1) та підключений за схемою з загальним катодом. Для його з'єднання з Arduino Nano використовується з'єднувач X6 (рис.3.7).

Хід виконання роботи

Підключити схему до комп'ютера через USB порт плати Arduino та/або запустити віртуальний стенд у середовищі Proteus VSM.

Завантажити програму RGB1 (додаток Г) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання RGB світлодіода у відповідність до програми. Дослідити роботу програми.

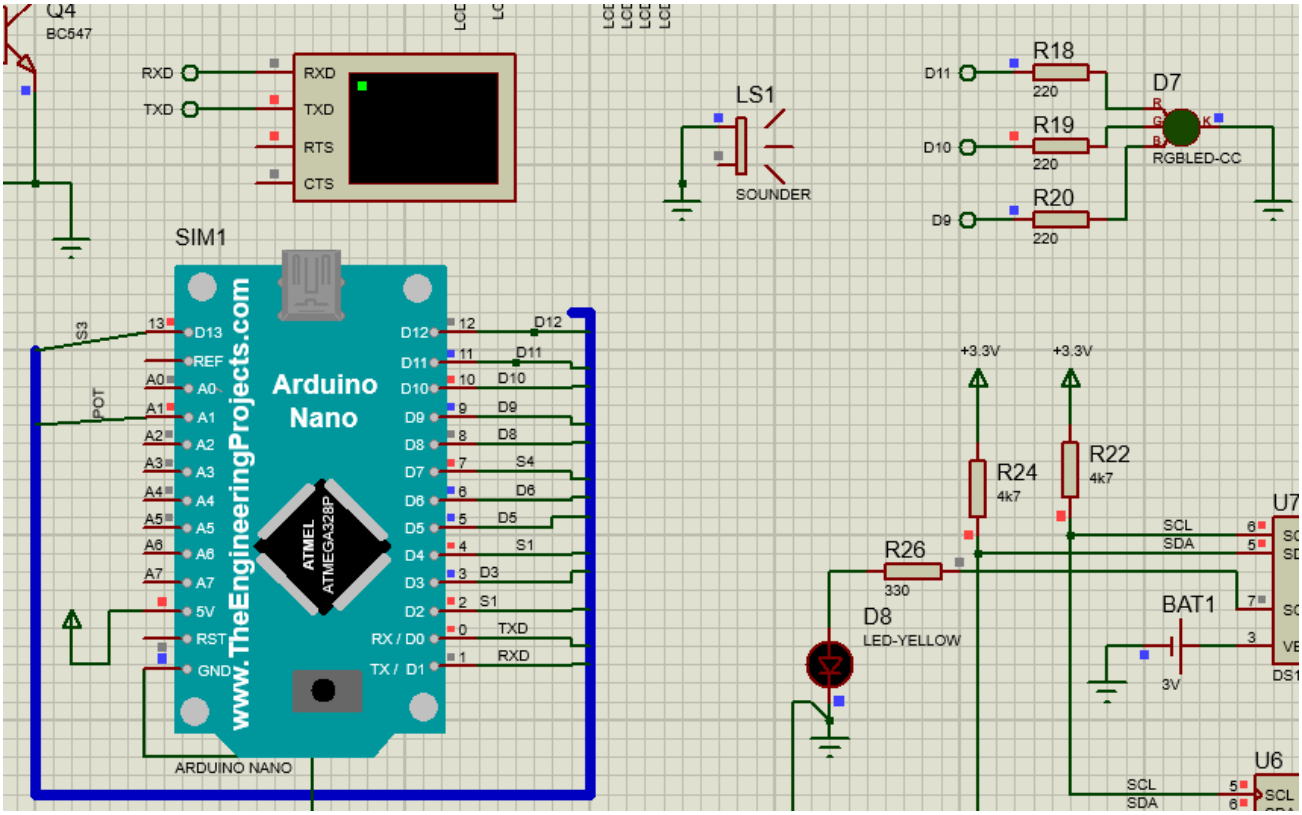

Рис. 1. Результат роботи

Завантажити програму RGB2 (додаток Г) до лабораторного макета /віртуального стенду. Дослідити роботу програми.

Завантажити програму RGB3 (додаток Г) до лабораторного макета / віртуального стенду. Дослідити роботу програми.

Завантажити програму RGB4 (додаток Г) до лабораторного макета / віртуального стенду. Дослідити роботу програми.

Завантажити програму RGB5 (додаток Г) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання RGB світлодіода та тактової кнопки у відповідність до програми. Дослідити роботу програми.

Індивідуальне завдання

Внести зміни до програми RGB5 (додаток Г). Для опрацювання моментів натиснення кнопки використати зовнішнє переривання. Для вибору режиму роботи RGB світлодіода використати оператор SWITCH.

Код:

```
#define BLED 9

#define GLED 10

#define RLED 11
```

```
#define BUTTON 2

boolean lastButton = LOW;
boolean currentButton = LOW;
int ledMode = 0;

void setup() {
  pinMode(BLED, OUTPUT);
  pinMode(GLED, OUTPUT);
  pinMode(RLED, OUTPUT);
  pinMode(BUTTON, INPUT_PULLUP);
 attachInterrupt(0, blink, CHANGE);

}

void loop() {
  if (lastButton == LOW && currentButton == HIGH) {
    ledMode++;
  }
  lastButton = currentButton;
  if (ledMode == 8) ledMode = 0;
  setMode(ledMode);
}

void blink()
{
    currentButton = debounce(lastButton);
}
```

```
boolean debounce(boolean last) {
  boolean current = digitalRead(BUTTON);
  if (last != current) {
    delay(5);
    current = digitalRead(BUTTON);
  }
  return current;
}

void setMode(int mode) {
  switch (mode){
    case 1:
    digitalWrite(RLED, HIGH);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, LOW);
    break;
    case 2:
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, HIGH);
    digitalWrite(BLED, LOW);
    break;
    case 3:
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, HIGH);
    break;
    case 4:
```

```
    analogWrite(RLED, 127);
    analogWrite(GLED, 0);
    analogWrite(BLED, 127);
    break;
    case 5:
    analogWrite(RLED, 0);
    analogWrite(GLED, 127);
    analogWrite(BLED, 127);
    break;
    case 6:
    analogWrite(RLED, 127);
    analogWrite(GLED, 127);
    analogWrite(BLED, 0);
    break;
    case 7:
    analogWrite(RLED, 85);
    analogWrite(GLED, 85);
    analogWrite(BLED, 85);
    break;
    default:
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, LOW);
    break;
  }
}
```

Код для МК AVR:

```
#include <avr/io.h>
#include <util/delay.h>
```

```
#include <avr/interrupt.h>

#define F_CPU 16000000UL

#define RLED PB3  // Timer0 OC0
#define GLED PB2  // Timer1B OC1B
#define BLED PB1  // Timer1A OC1A
#define BUTTON PD2

volatile uint8_t ledMode = 0;
volatile uint8_t lastButton = 0;
volatile uint8_t currentButton = 0;

void PWM_init(void) {
    // Виходи PWM
    DDRB |= (1<<RLED)|(1<<GLED)|(1<<BLED);

    // Timer0 Fast PWM для RLED
    TCCR0 = (1<<WGM00)|(1<<WGM01)|(1<<COM01)|(1<<CS01);

    // Timer1 8-bit Fast PWM для GLED і BLED
    TCCR1A = (1<<WGM10)|(1<<COM1A1)|(1<<COM1B1);
    TCCR1B = (1<<WGM12)|(1<<CS11);
}

void setPWM(uint8_t r, uint8_t g, uint8_t b) {
    OCR0 = r;    // RLED
    OCR1AL = b;  // BLED
```

```
    OCR1BL = g;  // GLED
}

void BUTTON_init(void) {
    DDRD &= ~(1<<BUTTON);    // PD2 як вхід
    PORTD |= (1<<BUTTON);    // внутрішня підтяжка

    // Переривання INT0 по зміні рівня
    MCUCR |= (1<<ISC00);  // CHANGE
    GICR |= (1<<INT0);    // увімкнути INT0
    sei();                // глобальні переривання
}

// debounce
uint8_t debounce(uint8_t last) {
    uint8_t current = PIND & (1<<BUTTON);
    if((last && !current) || (!last && current)) {
        _delay_ms(5);
        current = PIND & (1<<BUTTON);
    }
    return current;
}

ISR(INT0_vect) {
    currentButton = debounce(lastButton);
}

void setMode(uint8_t mode) {
```

```
    switch(mode) {
        case 1: setPWM(255,0,0); break;
        case 2: setPWM(0,255,0); break;
        case 3: setPWM(0,0,255); break;
        case 4: setPWM(127,0,127); break;
        case 5: setPWM(0,127,127); break;
        case 6: setPWM(127,127,0); break;
        case 7: setPWM(85,85,85); break;
        default: setPWM(0,0,0); break;
    }
}

int main(void) {
    PWM_init();
    BUTTON_init();

    while(1) {
        if((lastButton == 0) && (currentButton != 0)) {
            ledMode++;
            if(ledMode >= 8) ledMode = 0;
        }
        lastButton = currentButton;
        setMode(ledMode);
        _delay_ms(50);
    }
}
```

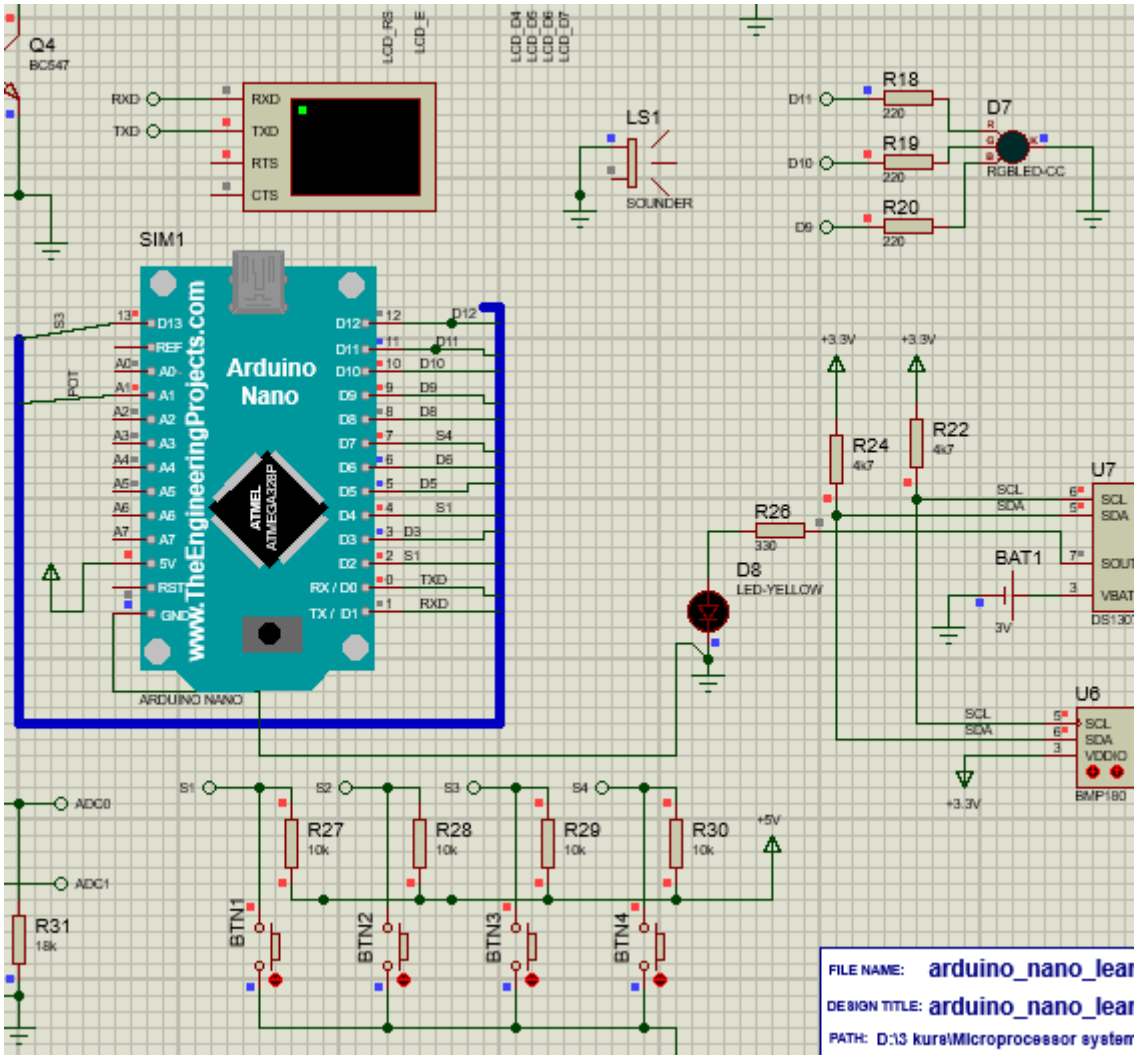

Рис. 2. Результат роботи

Реалізувати програму з застосуванням переривань INT0, INT1. При натисканні кнопки, що підключена до INT0 змінюється колір світлодіода, а при натисканні кнопки, що підключена до INT1 змінюється частота мигання світлодіода даним кольором.

Код:

```
#define BLED 9

#define GLED 10

#define RLED 11

#define BUTTON 2


bool button [2][2]={{0,0},{0,0}};

int ledMode = 0;

int ledSpead = 100;
```

```
const byte rgbPins[3] = {11,10,9};
const byte rainbow[6][3] = {
{1,0,0}, // red
{1,1,0}, // yellow
{0,1,0}, // green
{0,1,1}, // light blue
{0,0,1}, // blue
{1,0,1}, // purple
};

void setup() {
  pinMode(BLED, OUTPUT);
  pinMode(GLED, OUTPUT);
  pinMode(RLED, OUTPUT);
  pinMode(BUTTON, INPUT_PULLUP);
  attachInterrupt(0, blink0, CHANGE);
  attachInterrupt(1, blink1, CHANGE);
  Serial.begin(9600);
}

void loop() {
  if (button[0][0] == LOW && button[0][1] == HIGH) {
    ledMode++;
  }

  if (button[1][0] == LOW && button[1][1] == HIGH) {
    ledSpead=ledSpead+10;
```

```
  }

  for(int i=0;i<2;i++)
  {
    button[i][1]=button[i][0];
  }
  if (ledMode == 8) ledMode = 0;
  if (ledSpead == 255) ledSpead = 0;
  setMode(ledMode);
  Serial.print(ledMode);
  Serial.print(" ");
  Serial.println(ledSpead);
}

void blink0()
{
    button[0][0] = debounce(0,button[0][1]);
}

void blink1()
{
    button[1][0] = debounce(1,button[1][1]);
}

boolean debounce(int i, boolean last) {
  boolean current = digitalRead(i+2);
  if (last != current) {
    delay(5);
```

```
    current = digitalRead(i+2);
  }
  return current;
}

void setMode(int mode) {
  for(int k=0; k<3; k++) {
    digitalWrite(rgbPins[k], rainbow[mode][k]);
  }
  delay(ledSpead);
  for(int k=0; k<3; k++) {
    digitalWrite(rgbPins[k], LOW);
  }
  delay(ledSpead);



}
```

Код для МК AVR:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define F_CPU 16000000UL

#define RLED PB3   // PWM вихід Timer0
#define GLED PB2   // PWM вихід Timer1B
#define BLED PB1   // PWM вихід Timer1A
#define BUTTON0 PD2
#define BUTTON1 PD3
```

```
volatile uint8_t button[2][2] = {{0,0},{0,0}};
volatile uint8_t ledMode = 0;
volatile uint8_t ledSpeed = 100;

// Веселка: R,G,B
const uint8_t rainbow[6][3] = {
    {1,0,0}, // red
    {1,1,0}, // yellow
    {0,1,0}, // green
    {0,1,1}, // light blue
    {0,0,1}, // blue
    {1,0,1}  // purple
};

void PWM_init(void) {
    DDRB |= (1<<RLED)|(1<<GLED)|(1<<BLED);

    // Timer0 Fast PWM для RLED
    TCCR0 = (1<<WGM00)|(1<<WGM01)|(1<<COM01)|(1<<CS01);

    // Timer1 8-bit Fast PWM для GLED і BLED
    TCCR1A = (1<<WGM10)|(1<<COM1A1)|(1<<COM1B1);
    TCCR1B = (1<<WGM12)|(1<<CS11);
}

void setPWM(uint8_t r, uint8_t g, uint8_t b) {
    OCR0 = r?255:0;
```

```
    OCR1BL = g?255:0;
    OCR1AL = b?255:0;
}

void BUTTON_init(void) {
    DDRD &= ~((1<<BUTTON0)|(1<<BUTTON1)); // входи
    PORTD |= (1<<BUTTON0)|(1<<BUTTON1);   // підтяжка

    MCUCR |= (1<<ISC00)|(1<<ISC10); // CHANGE
    GICR |= (1<<INT0)|(1<<INT1);
    sei();
}

// debounce
uint8_t debounce(uint8_t btn) {
    uint8_t current = PIND & (1<<btn);
    _delay_ms(5);
    current = PIND & (1<<btn);
    return current ? 1:0;
}

ISR(INT0_vect) {
    button[0][0] = debounce(BUTTON0);
}

ISR(INT1_vect) {
    button[1][0] = debounce(BUTTON1);
}
```

```
void setMode(uint8_t mode) {
    for(uint8_t k=0;k<3;k++) {
        setPWM(rainbow[mode][0], rainbow[mode][1], rainbow[mode][2]);
    }
    _delay_ms(ledSpeed);
    setPWM(0,0,0);
    _delay_ms(ledSpeed);
}

int main(void) {
    PWM_init();
    BUTTON_init();

    while(1) {
        for(uint8_t i=0;i<2;i++)
            button[i][1] = button[i][0];

        if(button[0][1] && !button[0][0]) ledMode++;
        if(button[1][1] && !button[1][0]) ledSpeed += 10;

        if(ledMode >= 6) ledMode = 0;
        if(ledSpeed >= 255) ledSpeed = 0;

        setMode(ledMode);
    }
}
```

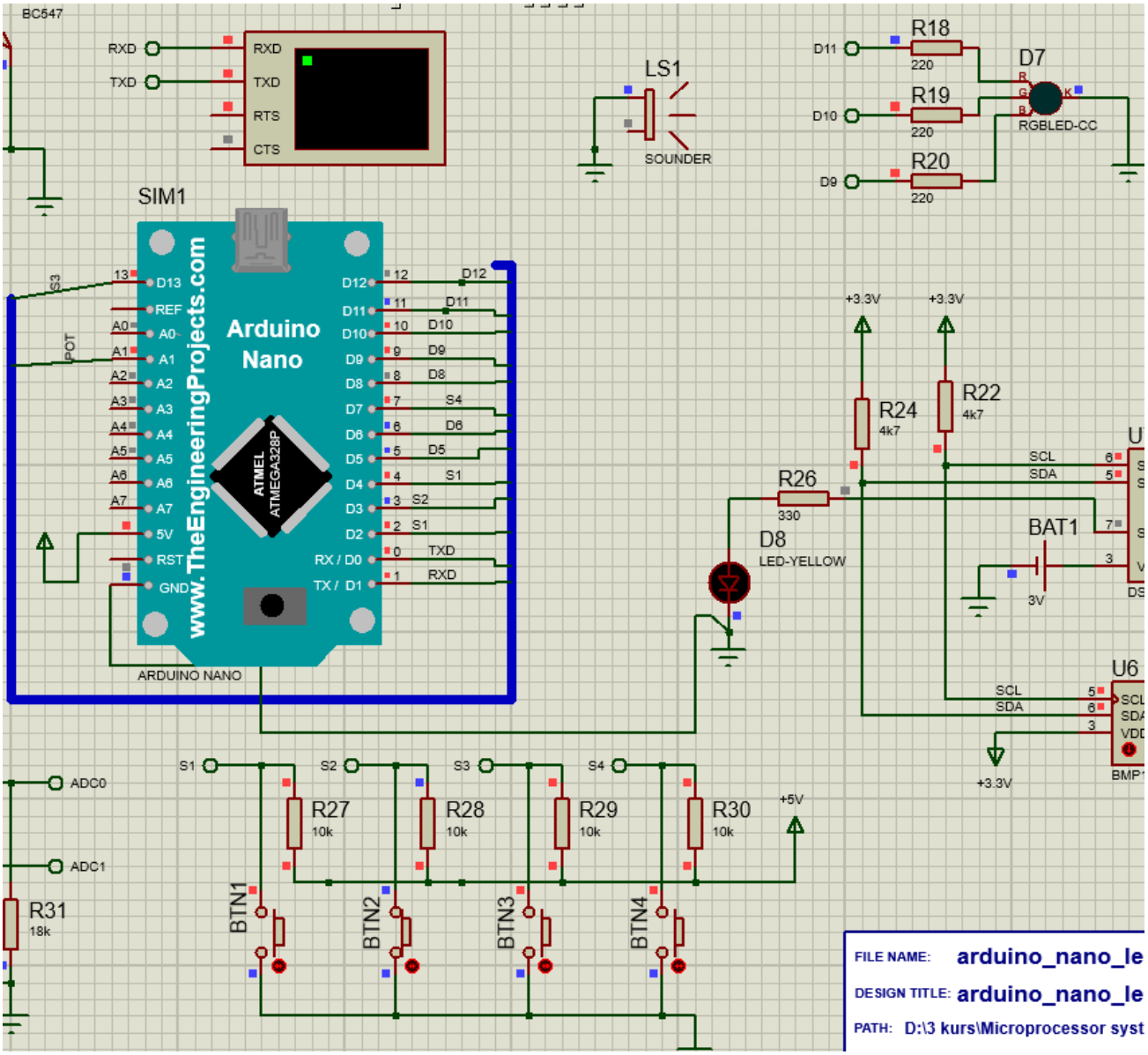

Рис. 3. Результат роботи

Реалізувати програму, у якій колір RGB світлодіода змінюється потенціометром RV1.

Код:

```
#define BLED 9

#define GLED 10

#define RLED 11


int ledMode = 0;


void setup() {
```

```
  pinMode(BLED, OUTPUT);
  pinMode(GLED, OUTPUT);
  pinMode(RLED, OUTPUT);
  pinMode(A1,INPUT_PULLUP);
}

void loop() {
  ledMode=analogRead(A1)/129;
  setMode(ledMode);
}

void setMode(int mode) {
  switch (mode){
    case 1:
    digitalWrite(RLED, HIGH);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, LOW);
    break;
    case 2:
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, HIGH);
    digitalWrite(BLED, LOW);
    break;
    case 3:
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, HIGH);
    break;
```

```
    case 4:
    analogWrite(RLED, 127);
    analogWrite(GLED, 0);
    analogWrite(BLED, 127);
    break;
    case 5:
    analogWrite(RLED, 0);
    analogWrite(GLED, 127);
    analogWrite(BLED, 127);
    break;
    case 6:
    analogWrite(RLED, 127);
    analogWrite(GLED, 127);
    analogWrite(BLED, 0);
    break;
    case 7:
    analogWrite(RLED, 85);
    analogWrite(GLED, 85);
    analogWrite(BLED, 85);
    break;
    default:
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, LOW);
    break;
  }
}
```

Код для МК AVR:

```
#include <avr/io.h>
```

```
#include <util/delay.h>

#define F_CPU 16000000UL

#define RLED PB3   // Timer0 OC0
#define GLED PB2   // Timer1B OC1B
#define BLED PB1   // Timer1A OC1A
#define ADC_CHANNEL 1  // A1 -> ADC1

void PWM_init(void) {
   DDRB |= (1<<RLED)|(1<<GLED)|(1<<BLED);

   // Timer0 Fast PWM non-inverting для RLED
   TCCR0 = (1<<WGM00)|(1<<WGM01)|(1<<COM01)|(1<<CS01);

   // Timer1 8-bit Fast PWM non-inverting для GLED і BLED
   TCCR1A = (1<<WGM10)|(1<<COM1A1)|(1<<COM1B1);
   TCCR1B = (1<<WGM12)|(1<<CS11);
}

void ADC_init(void) {
   ADMUX = (1<<REFS0) | (ADC_CHANNEL & 0x0F); // AVCC, ADC1
   ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); // prescaler 128
}

uint16_t ADC_read(uint8_t channel) {
   ADMUX = (ADMUX & 0xF0) | (channel & 0x0F);
   ADCSRA |= (1<<ADSC);
```

```
    while(ADCSRA & (1<<ADSC));
    return ADC;
}

void setMode(uint8_t mode) {
    switch(mode) {
        case 1: OCR0=255; OCR1BL=0; OCR1AL=0; break;
        case 2: OCR0=0; OCR1BL=255; OCR1AL=0; break;
        case 3: OCR0=0; OCR1BL=0; OCR1AL=255; break;
        case 4: OCR0=127; OCR1BL=0; OCR1AL=127; break;
        case 5: OCR0=0; OCR1BL=127; OCR1AL=127; break;
        case 6: OCR0=127; OCR1BL=127; OCR1AL=0; break;
        case 7: OCR0=85; OCR1BL=85; OCR1AL=85; break;
        default: OCR0=0; OCR1BL=0; OCR1AL=0; break;
    }
}

int main(void) {
    PWM_init();
    ADC_init();

    uint8_t ledMode = 0;

    while(1) {
        uint16_t adcVal = ADC_read(ADC_CHANNEL);
        ledMode = adcVal / 129; // 0-7
        setMode(ledMode);
        _delay_ms(50);
```

```
    }
}
```

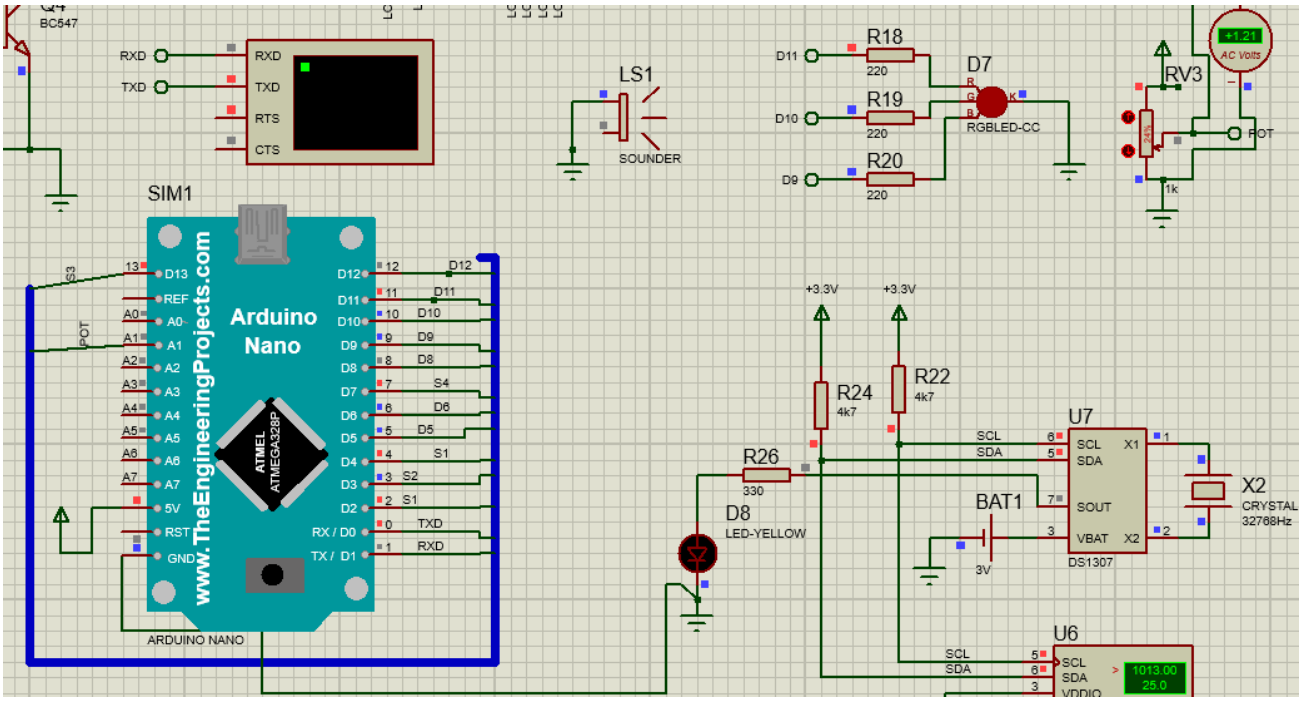

Рис. 4. Результат роботи

Висновок: У ході виконання лабораторної роботи я дослідив роботу RGB світлодіода, закріпив навички роботи з цифровими портами, тактовими кнопками, формуванням ШІМ; навчився програмувати режими роботи Arduino з використанням масивів.