МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

Інститут комп'ютерних наук та інформаційних технологій

Кафедра інформаційних систем та мереж



ЗВІТ

про виконання лабораторної роботи № 4

«Робота з інтерфейсом UART»

з дисципліни

«Вбудовані системи»

Виконала:

студентка групи ІТ-31

Щербак Л. В. Прийняв викладач:

Думич С.С.

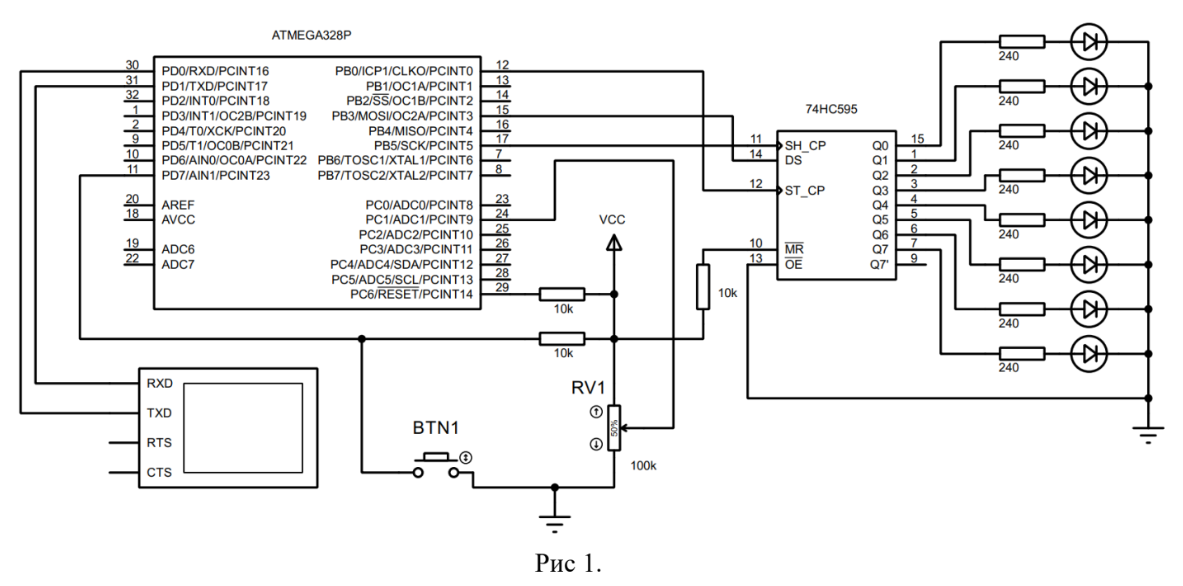
Львів 2022

Мета роботи: ознайомитися із принципами організації взаємодії мікроконтролера з персональним комп’ютером. Вивчити функціонування і програмування інтерфейсу UART.

Завдання до роботи:

1. Написати програму на мові Сі згідно з варіантом завдання.

2. Створити схему (Рис. 1) в програмі для моделювання (SimulIDE або Proteus 8). Провести моделювання написаної програми.



3. При наявності деталей зібрати схему на макетній платі та запрограмувати мікроконтролер.

Варіанти завдань

4. (\*) Використовуючи схему Рис 1. написати програму «автомат світлових ефектів». Керування ефектами (вибір ефекту, зміна швидкості, вивід справки, ...) відбувається введенням відповідних команд в терміналі.

Хід роботи:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

#include <util/delay.h>

#include <avr/io.h>

#define MAX\_NUMBER\_SYMBOLS\_IN\_COMMAND 255

#define MAX\_NUMBER\_SYMBOLS\_IN\_ARRAY 100

#define BAUD\_RATE 9600

#define OUT\_REG\_CS PB0

#define MOSI PB3

#define SCK PB5

#define BUTTON PD7

#define NUMBER\_COMMANDS 4

#define NUMBER\_EFFECTS 3

#define NUMBER\_SPEED 3

#define NOTATION 10

#define SPEED\_LOW 310

#define SPEED\_MEDIUM 150

#define SPEED\_HIGH 90

#define VALUE\_RIGHT\_CENTER\_LED 16

#define VALUE\_LEFT\_CENTER\_LED 8

#define VALUE\_FIRST\_LED 1

#define VALUE\_LAST\_LED 128

#define HELP\_COMMAND\_STRING "Commands: \r\n" \

"--- help\r\n" \

"--- info\r\n" \

"--- effect\r\n" \

"--- speed\r\n\r\n"

enum Commands

{

COMMANDS\_HELP,

COMMANDS\_INFORMATION,

COMMANDS\_EFFECT\_SELECTION,

COMMANDS\_SPEED\_CHANGE

};

enum Effects

{

FIRST\_EFFECT = 1,

SECOND\_EFFECT,

THIRD\_EFFECT

};

enum Speed

{

LOW = 1,

MEDIUM,

HIGH

};

typedef struct LedsConfiguration

{

uint32\_t speed;

uint8\_t effect;

} LedsConfiguration;

LedsConfiguration leds\_configuration;

const char \*commands[NUMBER\_COMMANDS];

const uint32\_t speeds\_numbers[NUMBER\_SPEED];

const uint32\_t speeds\_numbers[NUMBER\_SPEED];

const char \*commands[NUMBER\_COMMANDS];

LedsConfiguration leds\_configuration = { 1, 1 };

const char \*commands[NUMBER\_COMMANDS] =

{

"help",

"info",

"effect",

"speed"

};

const uint32\_t speeds\_numbers[NUMBER\_SPEED] =

{

SPEED\_LOW,

SPEED\_MEDIUM,

SPEED\_HIGH

};

void ProgramDelay(void)

{

switch(leds\_configuration.speed) {

case LOW:

\_delay\_ms(SPEED\_LOW);

break;

case MEDIUM:

\_delay\_ms(SPEED\_MEDIUM);

break;

case HIGH:

\_delay\_ms(SPEED\_HIGH);

break;

default:

\_delay\_ms(0);

}

}

void UsartWritingStringInTerminal(const char \*string)

{

while(\*string) {

//Writing Symbol In Terminal

//Start sending data

UDR0 = \*string++;

//Wait for complete sending data

while ( !(UCSR0A & (1 << UDRE0)) ) {}

}

}

bool SpeedProcessingForLeds(const char \*speed\_string)

{

uint8\_t speed\_number = atoi(speed\_string);

if( ('0' > \*speed\_string) || (\*speed\_string > '9') ||

( 0 > speed\_number) || ( speed\_number > 3) )

{

UsartWritingStringInTerminal("Incorrect speed!\r\n\r\n");

UsartWritingStringInTerminal("Enter the speed(1-3): ");

return false;

}

leds\_configuration.speed = speed\_number;

return true;

}

void StringClearing(char \*string, const uint8\_t size)

{

for(uint8\_t i = 0; i < size; ++i) string[i] = '\0';

}

void Atmega328DataInitialization(void)

{

//Init Spi

DDRB = (1 << OUT\_REG\_CS) | (1 << MOSI) | (1 << SCK );

SPCR = (1 << SPE ) | (1 << MSTR) | (1 << SPR0);

//

DDRC = 0;

DDRD = 0;

}

void StartSendDataLeds(uint8\_t data)

{

PORTB |= 1 << OUT\_REG\_CS;

//Start sending data

SPDR = data;

//Wait for complete sending data

while (!(SPSR & (1<<SPIF)));

PORTB &= ~(1 << OUT\_REG\_CS);

}

bool EffectProcessingForLeds(const char \*effect\_string)

{

uint8\_t effect\_number = atoi(effect\_string);

if( ('0' > \*effect\_string) || (\*effect\_string > '9') ||

( 0 > effect\_number) || ( effect\_number > 3) )

{

UsartWritingStringInTerminal("Incorrect effect!\r\n\r\n");

UsartWritingStringInTerminal("Enter the effect(1-3): ");

return false;

}

leds\_configuration.effect = effect\_number;

return true;

}

void StartFirstEffectForLeds(void)

{

static uint16\_t value\_led\_right = VALUE\_RIGHT\_CENTER\_LED;

static uint16\_t value\_led\_left = VALUE\_LEFT\_CENTER\_LED ;

StartSendDataLeds(value\_led\_right | value\_led\_left);

value\_led\_right = value\_led\_right << 1;

value\_led\_left = value\_led\_left >> 1;

if( (value\_led\_left < VALUE\_FIRST\_LED) &&

(value\_led\_right > VALUE\_LAST\_LED) )

{

value\_led\_right = VALUE\_RIGHT\_CENTER\_LED;

value\_led\_left = VALUE\_LEFT\_CENTER\_LED ;

}

ProgramDelay();

}

void StartSecondEffectForLeds(void)

{

static uint16\_t value\_previous\_led = VALUE\_FIRST\_LED;

static uint16\_t value\_next\_led = VALUE\_FIRST\_LED << 1;

StartSendDataLeds(value\_previous\_led | value\_next\_led);

static bool direction = true;

if(value\_next\_led == VALUE\_LAST\_LED)

{

direction = false;

}

else if(value\_previous\_led == VALUE\_FIRST\_LED)

{

direction = true;

}

if(direction)

{

value\_previous\_led = value\_previous\_led << 1;

value\_next\_led = value\_next\_led << 1;

}

else

{

value\_previous\_led = value\_previous\_led >> 1;

value\_next\_led = value\_next\_led >> 1;

}

ProgramDelay();

}

void StartThirdEffectForLeds(void)

{

static uint16\_t value\_previous\_led = VALUE\_FIRST\_LED;

static uint16\_t value\_next\_led = VALUE\_FIRST\_LED << 1;

static uint16\_t value\_last\_led = VALUE\_LAST\_LED;

static uint16\_t value\_penultimate\_led = VALUE\_LAST\_LED >> 1;

StartSendDataLeds(value\_previous\_led | value\_next\_led |

value\_last\_led | value\_penultimate\_led);

value\_previous\_led = value\_previous\_led << 1;

value\_next\_led = value\_next\_led << 1;

value\_last\_led = value\_last\_led >> 1;

value\_penultimate\_led = value\_penultimate\_led >> 1;

if( (value\_previous\_led > VALUE\_LAST\_LED ) &&

(value\_last\_led < VALUE\_FIRST\_LED) )

{

value\_previous\_led = VALUE\_FIRST\_LED;

value\_next\_led = value\_previous\_led << 1;

value\_last\_led = VALUE\_LAST\_LED;

value\_penultimate\_led = value\_last\_led >> 1;

}

ProgramDelay();

}

void StartEffectForLeds(void)

{

switch(leds\_configuration.effect)

{

case FIRST\_EFFECT:

StartFirstEffectForLeds();

break;

case SECOND\_EFFECT:

StartSecondEffectForLeds();

break;

case THIRD\_EFFECT:

StartThirdEffectForLeds();

break;

}

}

void ButtonHandler(void)

{

++leds\_configuration.speed;

if(leds\_configuration.speed > NUMBER\_SPEED)

{

leds\_configuration.speed = 1;

++leds\_configuration.effect;

}

if(leds\_configuration.effect > NUMBER\_EFFECTS)

{

leds\_configuration.speed = 1;

leds\_configuration.effect = 1;

}

}

void StartButtonScan(void)

{

static uint8\_t shreg;

shreg <<= 1;

if( (PIND & (1 << BUTTON)) != 0 )

{

shreg |= 1;

}

if( (shreg & 0x07) == 0x04 )

{

ButtonHandler();

}

}

uint8\_t UsartReadingSymbolFromTerminal(void)

{

//Wait for complete sending data

while ( !(UCSR0A & (1 << RXC0)) )

{

StartButtonScan ();

StartEffectForLeds ();

}

//Return received data

return UDR0;

}

bool UsartReadingStringFromTerminal(char \*string, uint8\_t \*size, const char \*message)

{

char symbol = UsartReadingSymbolFromTerminal();

if(symbol == '\r' || symbol == '\n')

{

if(strlen(string) != 0)

{

return true;

}

UsartWritingStringInTerminal(message);

}

else

{

string[(\*size)++] = symbol;

}

return false;

}

void UsartInterfaceInitialization(uint32\_t baud\_rate)

{

//Speed calculation(U2X = 0)

UBRR0H = (uint8\_t)((F\_CPU / (16 \* baud\_rate) - 1) >> 8);

UBRR0L = (uint8\_t) (F\_CPU / (16 \* baud\_rate) - 1) ;

UCSR0A = 0;

//Turn on the transmitter and receiver

UCSR0B = (1 << RXEN0) | (1 << TXEN0);

//Setting the data transmission/recieve frame

//8 data bits, 1 stop bit, no parity check

UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);

}

void ToProcessCommandHelp(void)

{

UsartWritingStringInTerminal(HELP\_COMMAND\_STRING);

}

void PrintInformationInTerminal(const char \*string\_number, const char \*data)

{

UsartWritingStringInTerminal(data);

UsartWritingStringInTerminal(string\_number);

UsartWritingStringInTerminal("\r\n");

}

void ToProcessCommandInformation(void)

{

char string\_number[MAX\_NUMBER\_SYMBOLS\_IN\_ARRAY];

itoa(leds\_configuration.speed, string\_number, NOTATION);

PrintInformationInTerminal(string\_number, "Speed: ");

itoa(leds\_configuration.effect, string\_number, NOTATION);

PrintInformationInTerminal(string\_number, "Effect: ");

UsartWritingStringInTerminal("\r\n");

}

void ToProcessCommandEffectSelection(void)

{

UsartWritingStringInTerminal("Enter the effect(1-3): ");

for( ; ; )

{

static uint8\_t size = 0;

static char effect\_string[MAX\_NUMBER\_SYMBOLS\_IN\_ARRAY];

bool check = UsartReadingStringFromTerminal(effect\_string, &size, "Enter the effect(1-3): ");

if(check)

{

bool successfully = EffectProcessingForLeds(effect\_string);

StringClearing(effect\_string, size);

size = 0;

if(successfully)

{

return;

}

}

}

}

void ToProcessCommandSpeedChange(void)

{

UsartWritingStringInTerminal("Enter the speed(1-3): ");

for( ; ; )

{

static uint8\_t size = 0;

static char speed\_string[MAX\_NUMBER\_SYMBOLS\_IN\_ARRAY];

bool check = UsartReadingStringFromTerminal(speed\_string, &size, "Enter the speed(1-3): ");

if(check)

{

bool successfully = SpeedProcessingForLeds(speed\_string);

StringClearing(speed\_string, size);

size = 0;

if(successfully)

{

return;

}

}

}

}

uint8\_t DetermineCommand(const char \*command)

{

for(uint8\_t i = 0; i < NUMBER\_COMMANDS; ++i)

{

if( strcmp(command, commands[i]) == 0)

{

return i;

}

}

return NUMBER\_COMMANDS + 1;

}

void CommandsProcessing(const char \*command)

{

uint8\_t action = DetermineCommand(command);

switch(action)

{

case COMMANDS\_HELP:

ToProcessCommandHelp();

UsartWritingStringInTerminal("Enter the command: ");

break;

case COMMANDS\_INFORMATION:

ToProcessCommandInformation();

UsartWritingStringInTerminal("Enter the command: ");

break;

case COMMANDS\_EFFECT\_SELECTION:

ToProcessCommandEffectSelection();

UsartWritingStringInTerminal("Enter the command: ");

break;

case COMMANDS\_SPEED\_CHANGE:

ToProcessCommandSpeedChange();

UsartWritingStringInTerminal("Enter the command: ");

break;

default:

UsartWritingStringInTerminal("Incorrect command!\r\n\r\n");

UsartWritingStringInTerminal("Enter the command: ");

break;

}

}

void StartTerminalProcess(void)

{

static uint8\_t size = 0;

static char command[MAX\_NUMBER\_SYMBOLS\_IN\_COMMAND];

bool check = UsartReadingStringFromTerminal(command, &size, "Enter the command: ");

if(check)

{

CommandsProcessing(command);

StringClearing(command, size);

size = 0;

}

}

void StartDataInitialization(void)

{

UsartInterfaceInitialization(BAUD\_RATE);

CommandsProcessing("help");

Atmega328DataInitialization();

}

extern void StartDataInitialization(void);

extern void StartTerminalProcess (void);

int main(void)

{

StartDataInitialization();

for( ; ; )

{

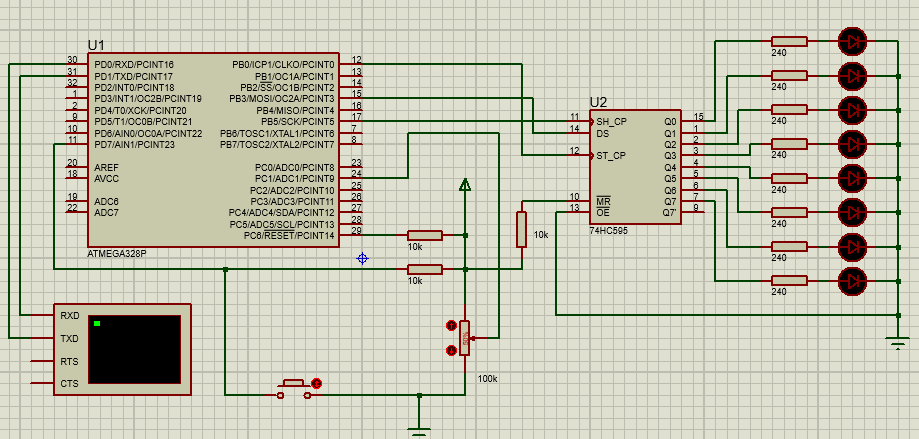
StartTerminalProcess();

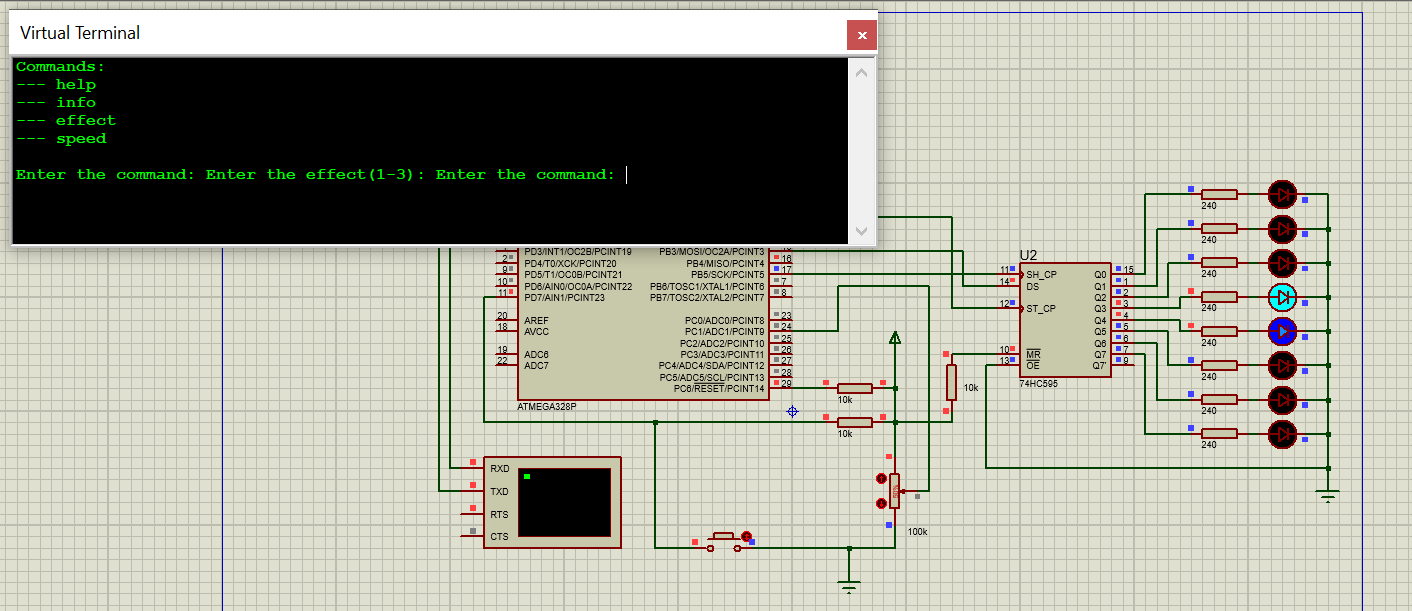
}

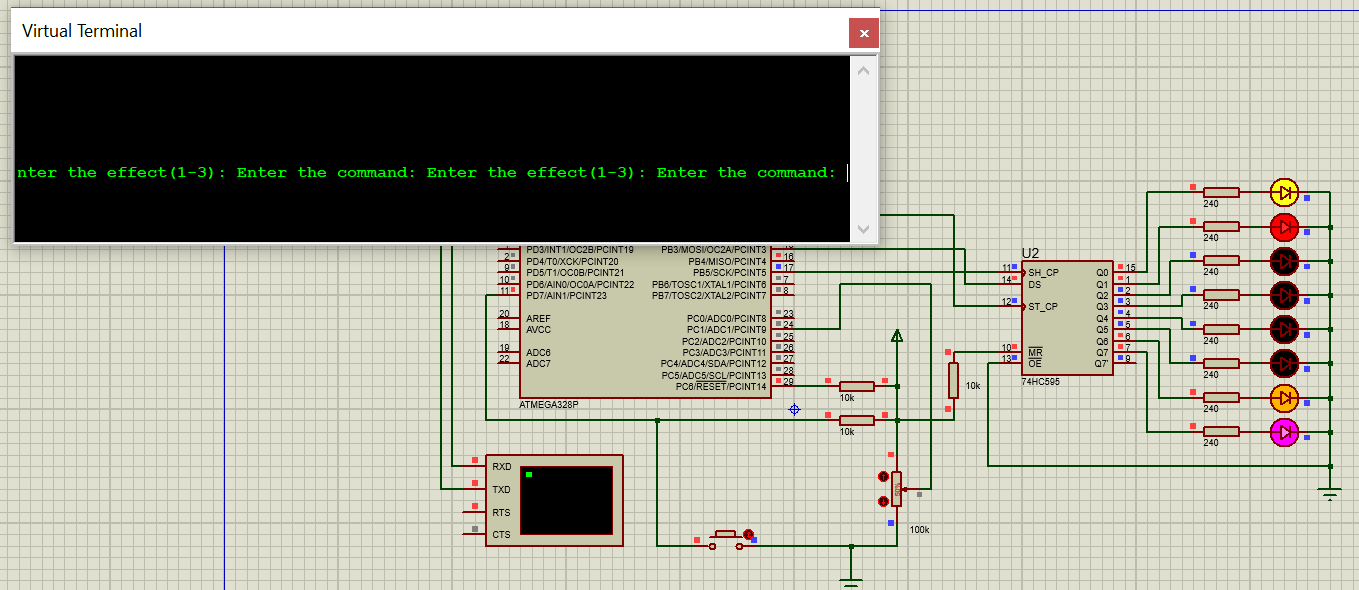
return 0;

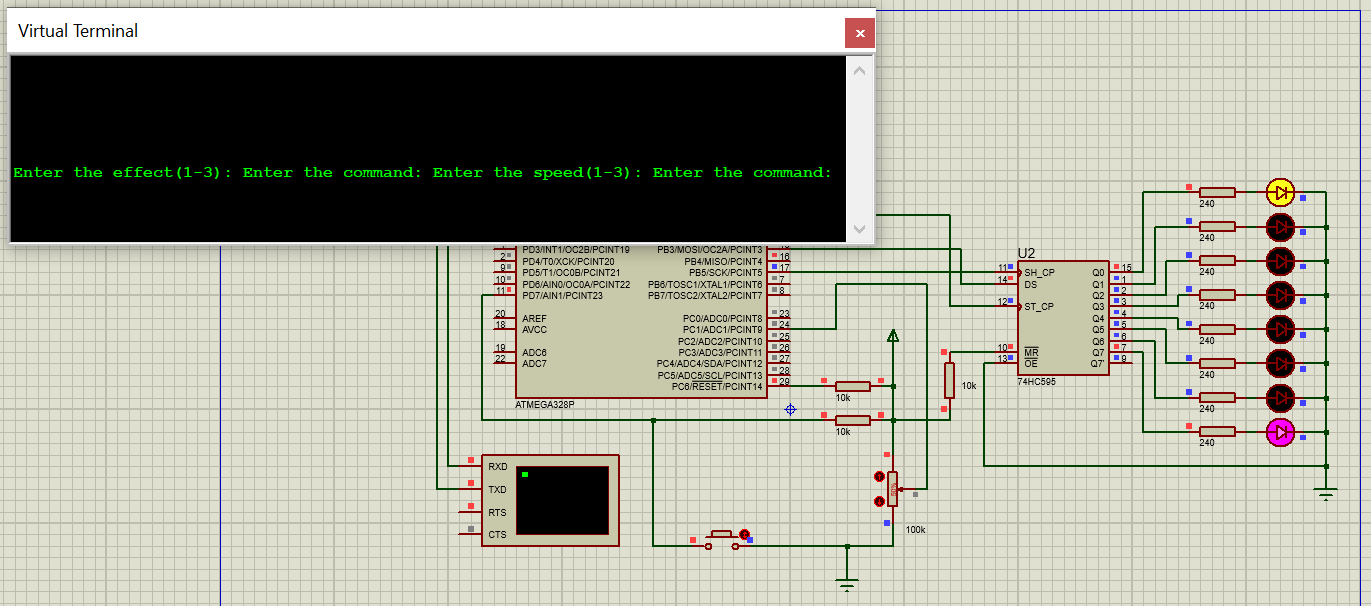
}

Результат:









Висновок: ознайомилася із принципами організації взаємодії мікроконтролера з персональним комп’ютером. Вивчити функціонування і програмування інтерфейсу UART.