**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

**ІТРЕ**

****

**ЗВІТ**

про виконання лабораторної роботи № 3

з дисципліни «Вбудовані системи»

Виконав: студент групи ІК-32

Чик І. Д.

Прийняв: Думич С. С.

**2023**

**Мета роботи:** Навчитися використовувати SPI інтерфейс для обміну даними між пристроями.

**Завдання**

(\*) Написати програму яка виводить біжучу стрічку (з зсувам на один піксель) на матричному індикаторі (схема Рис 5).

**main.c**

#include "max7219.h"

#include <avr/pgmspace.h>

const char message[] PROGMEM = "Ura URA ura ";

int main(void)

{

initSPI();

initMatrix();

clearMatrix();

initBuffer();

const char \*messagePointer = &message[0];

uint16\_t messageSize = sizeof(message);

while (1)

{

displayMessage(messagePointer, messageSize);

}

return (0);

}

**max7219.c**

#include "max7219.h"

#include <util/delay.h>

#include <avr/pgmspace.h>

static uint8\_t buffer[NUM\_DEVICES \* 8];

const char characters[96][5] PROGMEM = {

{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000}, // space

{0b00000000, 0b00000000, 0b01001111, 0b00000000, 0b00000000}, // !

{0b00000000, 0b00000111, 0b00000000, 0b00000111, 0b00000000}, // "

{0b00010100, 0b01111111, 0b00010100, 0b01111111, 0b00010100}, // #

{0b00100100, 0b00101010, 0b01111111, 0b00101010, 0b00010010}, // $

{0b00100011, 0b00010011, 0b00001000, 0b01100100, 0b01100010}, // %

{0b00110110, 0b01001001, 0b01010101, 0b00100010, 0b01010000}, // &

{0b00000000, 0b00000101, 0b00000011, 0b00000000, 0b00000000}, // '

{0b00000000, 0b00011100, 0b00100010, 0b01000001, 0b00000000}, // (

{0b00000000, 0b01000001, 0b00100010, 0b00011100, 0b00000000}, // )

{0b00010100, 0b00001000, 0b00111110, 0b00001000, 0b00010100}, // \*

{0b00001000, 0b00001000, 0b00111110, 0b00001000, 0b00001000}, // +

{0b00000000, 0b01010000, 0b00110000, 0b00000000, 0b00000000}, // ,

{0b00001000, 0b00001000, 0b00001000, 0b00001000, 0b00001000}, // -

{0b00000000, 0b01100000, 0b01100000, 0b00000000, 0b00000000}, // .

{0b00100000, 0b00010000, 0b00001000, 0b00000100, 0b00000010}, // /

{0b00111110, 0b01010001, 0b01001001, 0b01000101, 0b00111110}, // 0

{0b00000000, 0b01000010, 0b01111111, 0b01000000, 0b00000000}, // 1

{0b01000010, 0b01100001, 0b01010001, 0b01001001, 0b01000110}, // 2

{0b00100001, 0b01000001, 0b01000101, 0b01001011, 0b00110001}, // 3

{0b00011000, 0b00010100, 0b00010010, 0b01111111, 0b00010000}, // 4

{0b00100111, 0b01000101, 0b01000101, 0b01000101, 0b00111001}, // 5

{0b00111100, 0b01001010, 0b01001001, 0b01001001, 0b00110000}, // 6

{0b00000011, 0b01110001, 0b00001001, 0b00000101, 0b00000011}, // 7

{0b00110110, 0b01001001, 0b01001001, 0b01001001, 0b00110110}, // 8

{0b00000110, 0b01001001, 0b01001001, 0b00101001, 0b00011110}, // 9

{0b00000000, 0b01101100, 0b01101100, 0b00000000, 0b00000000}, // :

{0b00000000, 0b01010110, 0b00110110, 0b00000000, 0b00000000}, // ;

{0b00001000, 0b00010100, 0b00100010, 0b01000001, 0b00000000}, // <

{0b00010100, 0b00010100, 0b00010100, 0b00010100, 0b00010100}, // =

{0b00000000, 0b01000001, 0b00100010, 0b00010100, 0b00001000}, // >

{0b00000010, 0b00000001, 0b01010001, 0b00001001, 0b00000110}, // ?

{0b00110010, 0b01001001, 0b01111001, 0b01000001, 0b00111110}, // @

{0b01111110, 0b00010001, 0b00010001, 0b00010001, 0b01111110}, // A

{0b01111111, 0b01001001, 0b01001001, 0b01001001, 0b00111110}, // B

{0b00111110, 0b01000001, 0b01000001, 0b01000001, 0b00100010}, // C

{0b01111111, 0b01000001, 0b01000001, 0b01000001, 0b00111110}, // D

{0b01111111, 0b01001001, 0b01001001, 0b01001001, 0b01001001}, // E

{0b01111111, 0b00001001, 0b00001001, 0b00001001, 0b00000001}, // F

{0b00111110, 0b01000001, 0b01001001, 0b01001001, 0b00111010}, // G

{0b01111111, 0b00001000, 0b00001000, 0b00001000, 0b01111111}, // H

{0b01000001, 0b01000001, 0b01111111, 0b01000001, 0b01000001}, // I

{0b00110000, 0b01000001, 0b01000001, 0b00111111, 0b00000001}, // J

{0b01111111, 0b00001000, 0b00010100, 0b00100010, 0b01000001}, // K

{0b01111111, 0b01000000, 0b01000000, 0b01000000, 0b01000000}, // L

{0b01111111, 0b00000010, 0b00001100, 0b00000010, 0b01111111}, // M

{0b01111111, 0b00000100, 0b00001000, 0b00010000, 0b01111111}, // N

{0b00111110, 0b01000001, 0b01000001, 0b01000001, 0b00111110}, // O

{0b01111111, 0b00001001, 0b00001001, 0b00001001, 0b00000110}, // P

{0b00111110, 0b01000001, 0b01010001, 0b00100001, 0b01011110}, // Q

{0b01111111, 0b00001001, 0b00001001, 0b00011001, 0b01100110}, // R

{0b01000110, 0b01001001, 0b01001001, 0b01001001, 0b00110001}, // S

{0b00000001, 0b00000001, 0b01111111, 0b00000001, 0b00000001}, // T

{0b00111111, 0b01000000, 0b01000000, 0b01000000, 0b00111111}, // U

{0b00001111, 0b00110000, 0b01000000, 0b00110000, 0b00001111}, // V

{0b00111111, 0b01000000, 0b00111000, 0b01000000, 0b00111111}, // W

{0b01100011, 0b00010100, 0b00001000, 0b00010100, 0b01100011}, // X

{0b00000011, 0b00000100, 0b01111000, 0b00000100, 0b00000011}, // Y

{0b01100001, 0b01010001, 0b01001001, 0b01000101, 0b01000011}, // Z

{0b01111111, 0b01000001, 0b01000001, 0b00000000, 0b00000000}, // [

{0b00000010, 0b00000100, 0b00001000, 0b00010000, 0b00100000}, // '\'

{0b00000000, 0b00000000, 0b01000001, 0b01000001, 0b01111111}, // ]

{0b00000100, 0b00000010, 0b00000001, 0b00000010, 0b00000100}, // ^

{0b01000000, 0b01000000, 0b01000000, 0b01000000, 0b01000000}, // \_

{0b00000000, 0b00000001, 0b00000010, 0b00000100, 0b00000000}, // `

{0b00100000, 0b01010100, 0b01010100, 0b01010100, 0b01111000}, // a

{0b01111111, 0b01001000, 0b01000100, 0b01000100, 0b00111000}, // 0b

{0b00111000, 0b01000100, 0b01000100, 0b01000100, 0b00100000}, // c

{0b00111000, 0b01000100, 0b01000100, 0b01001000, 0b01111111}, // d

{0b00111000, 0b01010100, 0b01010100, 0b01010100, 0b00011000}, // e

{0b00001000, 0b01111110, 0b00001001, 0b00000001, 0b00000010}, // f

{0b00001100, 0b01010010, 0b01010010, 0b01010010, 0b00111110}, // g

{0b01111111, 0b00001000, 0b00000100, 0b00000100, 0b01111000}, // h

{0b00000000, 0b01000100, 0b01111101, 0b01000000, 0b00000000}, // i

{0b00100000, 0b01000000, 0b01000100, 0b00111101, 0b00000000}, // j

{0b01111111, 0b00010000, 0b00101000, 0b01000100, 0b00000000}, // k

{0b00000000, 0b01000001, 0b01111111, 0b01000000, 0b00000000}, // l

{0b01111000, 0b00000100, 0b00001000, 0b00000100, 0b01111000}, // m

{0b01111100, 0b00001000, 0b00000100, 0b00000100, 0b01111000}, // n

{0b00111000, 0b01000100, 0b01000100, 0b01000100, 0b00111000}, // o

{0b01111100, 0b00010100, 0b00010100, 0b00010100, 0b00001000}, // p

{0b00001000, 0b00010100, 0b00010100, 0b01111100, 0b00000000}, // q

{0b01111100, 0b00001000, 0b00000100, 0b00000100, 0b00001000}, // r

{0b01001000, 0b01010100, 0b01010100, 0b01010100, 0b00100000}, // s

{0b00000100, 0b00111111, 0b01000100, 0b01000000, 0b00100000}, // t

{0b00111100, 0b01000000, 0b01000000, 0b00100000, 0b01111100}, // u

{0b00011100, 0b00100000, 0b01000000, 0b00100000, 0b00011100}, // v

{0b00111100, 0b01000000, 0b00110000, 0b01000000, 0b00111100}, // w

{0b01000100, 0b00101000, 0b00010000, 0b00101000, 0b01000100}, // x

{0b00001100, 0b01010000, 0b01010000, 0b01010000, 0b00111100}, // y

{0b01000100, 0b01100100, 0b01010100, 0b01001100, 0b01000100}, // z

{0b00000000, 0b00001000, 0b00110110, 0b01000001, 0b00000000}, // {

{0b00000000, 0b00000000, 0b01111111, 0b00000000, 0b00000000}, // |

{0b00000000, 0b01000001, 0b00110110, 0b00001000, 0b00000000}, // }

{0b00001000, 0b00000100, 0b00000100, 0b00001000, 0b00000100} // ~

};

void initSPI(void)

{

DDRB |= (1 << PB2);

PORTB |= (1 << PB2);

DDRB |= (1 << PB3);

DDRB |= (1 << PB5);

SPCR |= (1 << MSTR);

SPCR |= (1 << SPE);

}

void writeByte(uint8\_t byte)

{

SPDR = byte;

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

;

}

void writeWord(uint8\_t address, uint8\_t data)

{

writeByte(address);

writeByte(data);

}

void initMatrix()

{

uint8\_t i;

SLAVE\_SELECT;

for (i = 0; i < NUM\_DEVICES; i++)

{

writeByte(0x0A);

writeByte(0x01);

}

SLAVE\_DESELECT;

SLAVE\_SELECT;

for (i = 0; i < NUM\_DEVICES; i++)

{

writeByte(0x0B);

writeByte(0x07);

}

SLAVE\_DESELECT;

SLAVE\_SELECT;

for (i = 0; i < NUM\_DEVICES; i++)

{

writeByte(0x0C);

writeByte(0x01);

}

SLAVE\_DESELECT;

SLAVE\_SELECT;

for (i = 0; i < NUM\_DEVICES; i++)

{

writeByte(0x0F);

writeByte(0x00);

}

SLAVE\_DESELECT;

}

void clearMatrix(void)

{

for (uint8\_t x = 1; x < 9; x++)

{

SLAVE\_SELECT;

for (uint8\_t i = 0; i < NUM\_DEVICES; i++)

{

writeByte(x);

writeByte(0x00);

}

SLAVE\_DESELECT;

}

}

void initBuffer(void)

{

for (uint8\_t i = 0; i < NUM\_DEVICES \* 8; i++)

buffer[i] = 0x00;

}

void pushBuffer(uint8\_t x)

{

for (uint8\_t i = 0; i < NUM\_DEVICES \* 8 - 1; i++)

buffer[i] = buffer[i + 1];

buffer[NUM\_DEVICES \* 8 - 1] = x;

}

void pushCharacter(uint8\_t c)

{

for (uint8\_t i = 0; i < 5; i++)

{

pushBuffer(pgm\_read\_byte(&characters[c][i]));

displayBuffer();

\_delay\_us(DEL);

}

}

void displayMessage(const char \*arrayPointer, uint16\_t arraySize)

{

for (uint16\_t i = 0; i < arraySize; i++)

{

pushCharacter(pgm\_read\_byte\_near(arrayPointer + i) - 32);

pushBuffer(0x00);

displayBuffer();

\_delay\_us(DEL);

}

}

void displayBuffer()

{

for (uint8\_t i = 0; i < NUM\_DEVICES; i++)

{

for (uint8\_t j = 1; j < 9; j++)

{

SLAVE\_SELECT;

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

writeWord(0x00, 0x00);

writeWord(j, buffer[j + i \* 8 - 1]);

for (uint8\_t k = NUM\_DEVICES - 1; k > i; k--)

writeWord(0x00, 0x00);

SLAVE\_DESELECT;

}

}

}

**max7219.h**

#ifndef \_MAX7219\_H

#define \_MAX7219\_H

#include <avr/io.h>

#define SLAVE\_SELECT PORTB &= ~(1 << PB2)

#define SLAVE\_DESELECT PORTB |= (1 << PB2)

#define NUM\_DEVICES 2

#define DEL 60000

void initSPI(void);

void writeByte(uint8\_t byte);

void writeWord(uint8\_t address, uint8\_t data);

void initMatrix(void);

void clearMatrix(void);

void initBuffer(void);

void pushBuffer(uint8\_t x);

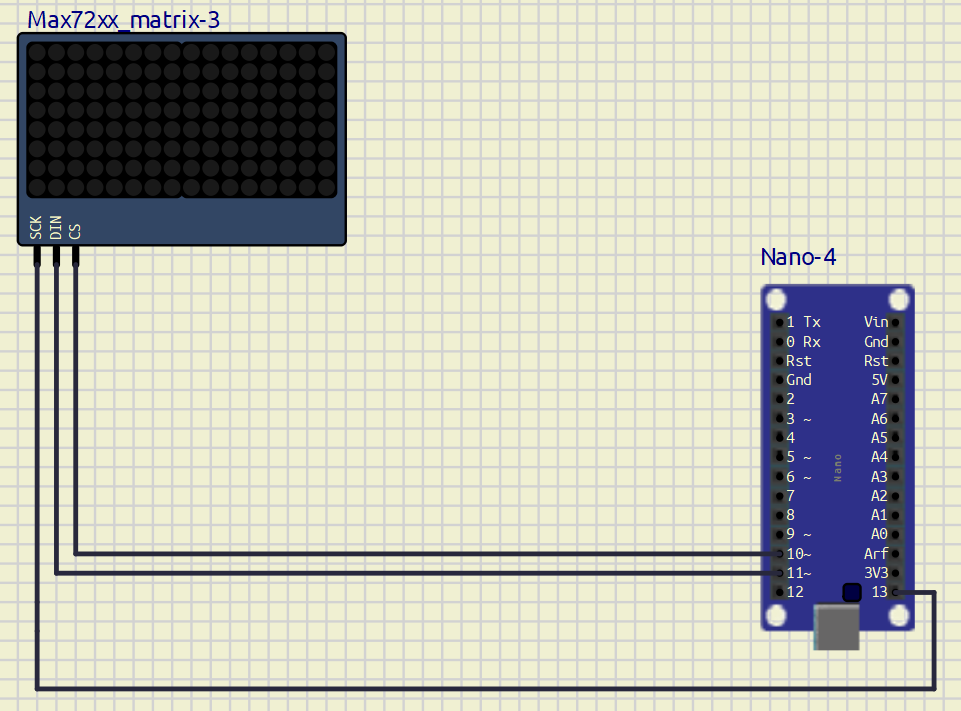
void pushCharacter(uint8\_t c);

void displayMessage(const char \*arrayPointer, uint16\_t arraySize);

void displayBuffer(void);

#endif

**Реалізація схеми в SimulIDE**

****

**Висновок:** Я навчився використовувати SPI інтерфейс для обміну даними між пристроями.