Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра «Телекомунікації»

Методичні вказівки
до виконання лабораторної роботи №5
з дисципліни «Вбудовані системи»
на тему «Робота з шинами I2C та 1-Wire»

Львів 2021

Мета роботи: ознайомитися із принципами організації взаємодії мікроконтролера з різними пристроями, використовуючи інтерфейси I2C та 1-Wire.

Короткі теоретичні відомості

I2C інтерфейс

Інтерфейс **I2C** (*Inter-Integrated Circuit*) – послідовна шина даних для зв'язку інтегральних схем, що використовує дві двох-направлені лінії зв'язку ***SDA*** і ***SCL***. По лінії ***SDA*** передаються дані, а по ***SCL*** - тактовий сигнал. Обидві лінії підтягнуті через резистори 4,7 кОм до напруги живлення (Рис.1). Допустима ємність ліній з пристроями – до 400 пФ.

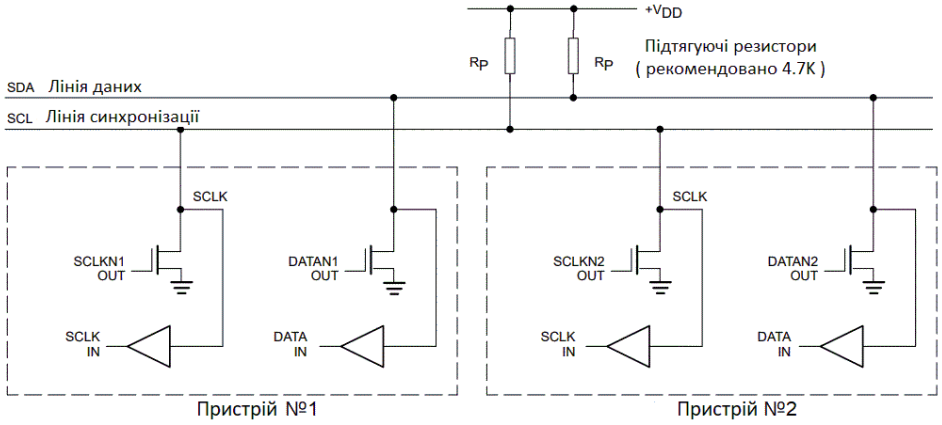

Рис 1. Схема підключення пристроїв до I2C шини.

Фірма ***Philips***, за використання назви цього інтерфейсу вимагає ліцензійних відрахувань, тому в мікроконтролерах AVR (фірми **Atmel**) використовується власна назва **TWI** –***T****wo-****W****ire* ***I****nterface*. Переваги шини **I2C**:

1. Потрібно тільки дві лінії – лінія даних (***SDA***) і лінія синхронізації (***SCL***). Кожен пристрій, підключений до шини, може бути програмно адресований за унікальною адресою. У кожен момент часу існує просте відношення ведучий/ведений: ведучі можуть працювати як ведучий-передавач і ведучий-приймач.

2. Шина дозволяє мати декілька ведучих надаючи засоби для визначення колізій і арбітражу, щоб запобігти пошкодженню даних в ситуації коли два або більше ведучих одночасно починають передавати дані. У стандартному режимі забезпечується передавання послідовних 8- бітних даних зі швидкістю до 100 кбіт/с і до 400 кбіт/с у «швидкому» режимі.

3. Вбудований в мікросхеми фільтр придушує сплески, забезпечуючи цілісність даних.

Інтерфейс I2C є синхронним, тому кожен біт даних на лінії ***SDA*** супроводжується імпульсом на лінії синхронізації ***SCL***. Рівень даних повинен бути стабільним, коли на лінії синхронізації присутня логічна 1. Винятком для цього правила є генерація умов (СТАРТ/СТОП). Ведучий пристрій ініціює і закінчує передавання даних. Передавання даних ініціюється коли ведучий формує на шині умову СТАРТ, і припиняється, коли ведучий формує умову СТОП. Формат посилки на шині ***I2C*** зображено на Рис. 2. Між умовами СТАРТ і СТОП шина вважається зайнятою і в цьому випадку інший ведучий не може здійснювати керуючий вплив на шині. Існують особливі випадки, коли нова умова СТАРТ виникає між умовами СТАРТ і СТОП. Даний випадок іменується як

"Повторний старт" і використовується при необхідності ініціювати ведучим новий сеанс зв'язку, не втрачаючи за цьому керування шиною. Відповідно після "Повторного старту" шина вважається зайнятою до наступної умови СТОП.

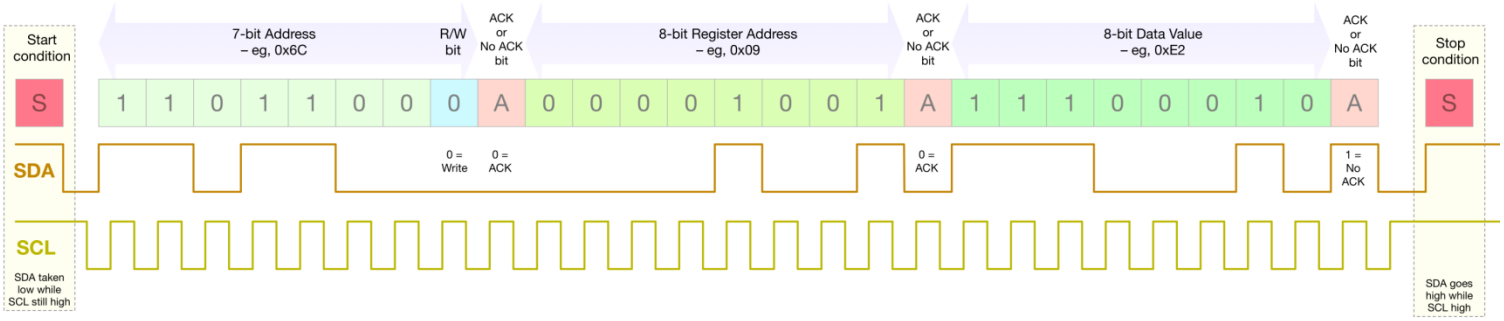

Рис. 2. Формат посилки інтерфейсу I2C.

Після передачі умови СТАРТ передається адрес підлеглого пристрою. Всі адресні пакети, які передаються по шині ***I2C***, складаються з 9 біт: 7 біт адреси пристрою до якого йде звернення, один біт для задання типу операції читання або запису (**R/W**) і один біт підтвердження (**ACK**). Якщо біт **R/W** = 1, то буде виконана операція читання, інакше – запис. Коли підлеглий розпізнає, що до нього відбувається запит (співпадіння адресу), то він повинен сформувати низький рівень на лінії ***SDA*** на 9-му циклі ***SCL*** (формування біта підтвердження – **ACK**). Якщо підлеглий пристрій з відповідним адресом відсутній на шині, або з будь-яких інших причин не може обслужити ведучий пристрій, то на лінії ***SDA*** необхідно залишити високий рівень під час циклу підтвердження (***NACK*** - *no acknowledge*). Ведучий після цього може передати умову СТОП або ПОВТОРНИЙ СТАРТ для ініціації нової передачі. Слід зазначити, що класична адресація включає 7-бітовий адресний простір з 16 зарезервованими адресами, відповідно є 112 вільних адрес для підключення периферії на одну шину. Пакети даних складаються з 8-біт даних і біта підтвердження, тобто також мають довжину 9 біт. Після прийому кожного байта даних, приймаючий пристрій (приймач) відповідає передавальному пристрою (передавачу), встановлюючи на лінії ***SDA*** низький рівень (біт підтвердження). Якщо приймаючий пристрій отримав останній байт або більше не може продовжувати прийом даних, він повинен "залишити" на лінії ***SDA*** високий рівень (***NACK***)

Шина 1-Wire

Шина **1-Wire** є основою мереж ***MicroLAN*** і розроблена наприкінці 90-х років фірмою ***Dallas Semiconductor***. В даний час фірма ***Dallas Semiconductor*** є дочірнім підприємством фірми **Maxim Integrated**. Основні характеристики цієї шини:

- максимальна протяжність шини до 300 м;
- швидкість передачі інформації 16,3 Кбіт/с;
- максимальна кількість адресованих елементів на шині 256;
- рівні напруги на шині відповідають стандартним рівням КМОП/ТТЛ;
- напруга живлення компонентів мережі 2,8...6 В;
- для з'єднання елементів мережі може застосовуватися звичайний телефонний кабель або вита пара.

Існують і модифікації шини ***1-Wire***, які підтримують швидкісний режим роботи шини (Overdrive, 142 Кбіт/с). Проте такі мікросхеми можуть працювати лише на шині малої протяжності і за умови, коли рівень зовнішніх електричних перешкод зведений до мінімуму. Шина ***MicroLAN*** побудована за технологією ***Master/Slave***. На шині має бути хоча б один ведучий пристрій (***Master***). Всі інші пристрої мають бути веденими (***Slave***). Ведучий пристрій ініціює всі процеси передачі інформації в межах шини. *Master* може прочитати дані з будь-якого ***Slave***-пристрою або записати їх туди. Передача інформації від одного *Slave* до іншого безпосередньо неможлива. Для того, щоб ***Master*** міг звертатися до будь-якого з ведених пристроїв по шині, кожен ведений пристрій містить в собі індивідуальний ID код (ROM-код). Цей код заноситься в спеціальну область мікросхеми з допомогою лазера. Фірма-виробник гарантує, що цей код ніколи не повториться і всі будь-коли виготовлені мікросхеми, що містять ***1-Wire*** інтерфейс, завжди матимуть різні коди. Протокол ***1-Wire*** включає спеціальну команду пошуку, за допомогою якої провідний пристрій (*Master*) може здійснювати автоматичний пошук ведених пристроїв. Середня швидкість пошуку елементів в мережі ***MicroLAN*** складає приблизно 75 вузлів в секунду.

Протокол **1-Wire** має декілька рівнів представлення даних. Найнижчий рівень описує як передаються окремі біти. Всі операції на шині здійснюються тільки під управлінням *Master*-пристрою, який може виконувати операції двох видів: записувати інформацію в Slave пристрій або зчитувати інформацію з нього. Інформація передається побайтно в послідовному вигляді, біт за бітом, починаючи з молодшого біта. У будь-якому з цих двох випадків для передачі інформації Master-пристрій виробляє на шині тактові імпульси, для чого він періодично «підсаджує» шину за допомогою вихідного транзистора. Корисна інформація передається завдяки зміні тривалості цих імпульсів. Причому при записі інформації тривалістю імпульсів управляє виключно *Master*-пристрій. У режимі читання *Master*-пристрій починає формування імпульсу, а *Slave*-пристрій може подовжувати тривалість будь-якого імпульсу, «підсаджуючи» у свою чергу сигнал на лінії в потрібний момент. У вихідному стані всі *Slave*-пристрої, підключені до шини, та перебувають в режимі очікування. Коли лінія «відпущена», напруга на шині визначається резистором навантаження R. Для запису даних *Master* починає формувати негативні синхроімпульси, на кожен біт формується один імпульс. Імпульси передаються «підсаджуванням» лінії до нуля. Для передачі кожного біта виділяється проміжок часу стандартної тривалості, який отримав назву «слот» (***Slot***). Якщо значення біта який передається дорівнює 0, то Master формує «довгий» імпульс, який рівний тривалості слота. Для передачі одиничного біта Master виробляє «короткий» імпульс, який, по суті, є чистим синхроімпульсом. Частина слота, що залишилася, має бути заповнена одиничним сигналом. Виявивши початок синхроімпульса, *Slave*-пристрій починає процес прийому біта інформації. Передній фронт цього імпульсу служить *Slave*-пристрою початком відліку. Витримавши паузу, рівну тривалості синхроімпульса, *Slave*-пристрій читає рівень сигналу на лінії. Протокол шини ***1-Wire*** жорстко визначає лише тривалість слота. Інтервал між слотами має обмеження лише на мінімальне своє значення. Максимальне значення інтервалу між слотами необмежене. Так можна легко регулювати швидкість передачі даних від максимального значення (16.3 Кбіт/с) до нуля. При читанні біта *Master* генерує лише синхроімпульси. Виявивши синхроімпульс *Slave*-пристрій подовжує або не подовжує цей синхроімпульс у межах слота. Якщо черговий зчитуваний біт дорівнює нулю, то синхроімпульс подовжується, якщо одиниці - подовження немає. *Master*-пристрій зчитує цю інформацію і контролює рівень сигналу всередині слота після синхроімпульсу. Для надійної роботи ***1-Wire*** інтерфейсу необхідно щоб у процесі передачі інформації всіма елементами мережі чітко дотримувалися часові параметри. Кожна мікросхема, підключена до

мережі, самостійно виробляє всі необхідні для її роботи інтервали часу. На рис. 3 наведено часові параметри протоколу ***1-Wire*** в різних режимах роботи. Величина слота для передачі одного біта інформації (Tx) має перебувати в межах від 60 до 120 мкс. Тривалість синхроімпульсу >= 1 мкс. Ведений пристрій, виявивши на шині передній фронт синхроімпульсу, має сформувати затримку мінімум у 15 мкс, і потім здійснити перевірку сигналу на шині. Допустимий розкид часу затримки для різних екземплярів мікросхем лежить в межах від 15 до 60 мкс. Цей діапазон показаний на рис. 3 у вигляді області, позначеної як «Зона перевірки рівня *Slave*». Мінімальна величина інтервалу між слотами (TREC) рівна 1 мкс, максимальна - необмежена. У режимі читання біта *Master* генерує лише синхроімпульси тривалістю 1 мкс. Якщо біт дорівнює нулю, *Slave*-пристрій подовжує тривалість синхроімпульсу. Мінімальна тривалість продовженого імпульсу складає 15 мкс. Для цього тимчасового інтервалу теж допускається досить значний розкид (ще на 45 мкс). Якщо біт дорівнює одиниці, подовження синхроімпульсу не відбувається. Аби правильно оцінити значення зчитуваного байта, *Master*-пристрій повинен прочитати рівень сигналу на шині відразу після закінчення синхроімпульсу, але не пізніше, ніж через 15 мкс. Зона перевірки рівня для *Master*-пристрою в режимі читання значно вужча аналогічної зони для *Slave*-пристрою в режимі запису.

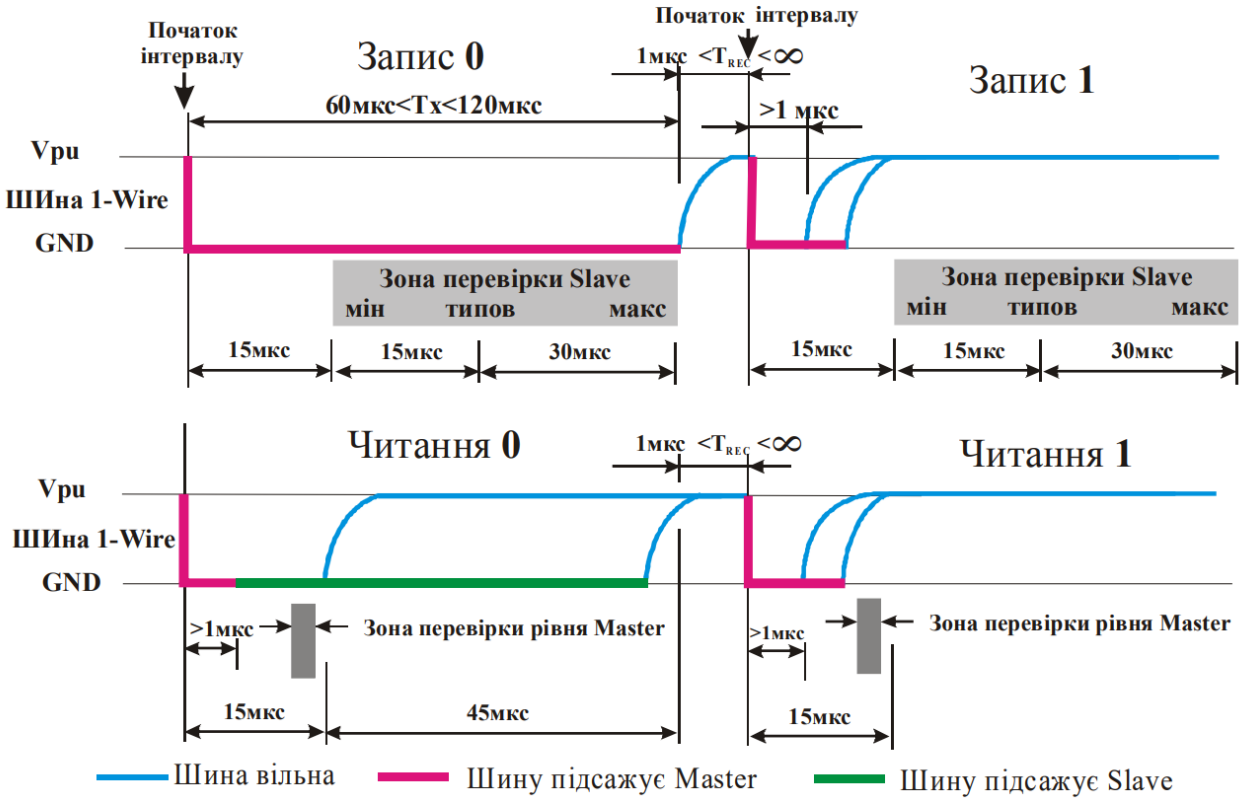

Рис 3. Часові параметри слотів прийому-передачі.

Байти передаються молодшим бітом вперед. Перші вісім бітів - це перший байт, наступні вісім бітів - другий байт, і так далі. Початок послідовності визначається сигналом скидання. Будь-який цикл обміну даними в мережі *MicroLAN* починається з імпульса скидання - наддовгого негативного імпульсу на шині ***1-Wire***, який виробляється провідним пристроєм (рис. 4).

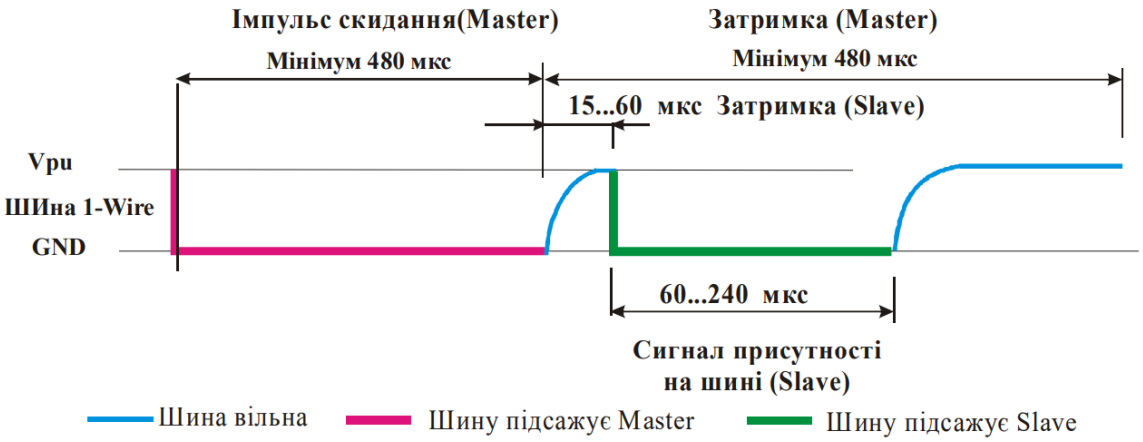

Рис 4. Часові параметри сигналу скидання та присутності.

З імпульсом скидання тісно пов'язаний ще один службовий сигнал - сигнал присутності на шині. Сигнал присутності виробляє кожний *Slave*-пристрій відразу після закінчення дії імпульсу скидання. *Master*-пристрій повинен проконтролювати наявність цього імпульсу. Якщо імпульсу присутності немає, то це означає, що на лінії немає жодного *Slave*-пристрою. Окрім ініціації імпульсів присутності імпульс скидання переводить у вихідний стан всі *Slave*-пристрої на шині. Тривалість імпульсу скидання має бути не менше 480 мкс. Процес передачі інформації по лінії повинен починатися не раніше, ніж через 480 мкс після закінчення дії імпульсу скидання. У цьому часовому інтервалі й очікується поява сигналу присутності. Для цього після закінчення імпульсу скидання *Master* «відпускає» лінію і чекає сигналу від *Slave*-пристроїв. Кожен *Slave*-пристрій, виявивши імпульс скидання, витримує паузу в 15-60 мкс, а потім «підсаджує» лінію. Тривалість імпульсу присутності складає 60-240 мкс. Сигнали присутності від всіх *Slave*-пристроїв зливаються в один загальний імпульс. Ведучий (*Master*) пристрій перевіряє наявність нульового рівня на лінії в середині цього інтервалу. Якщо сигнал виявлено, то це означає, що на лінії є хоча б один нормально працюючий пристрій. Будь-яка операція в мережі *MicroLAN* починається з команди, яка є одним байтом інформації. Кожна команда має свій власний код. Виконання команди починається з імпульсу скидання. Потім *Slave*-пристрій виробляє, а *Master* перевіряє сигнал присутності на лінії. Якщо сигнал присутності отримано, *Master* видає на лінію код ***ROM***-команди (ID).

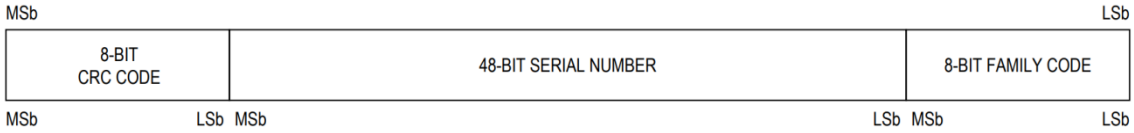

Рис 5. Формат ідентифікатора ROM (ID) 1-Wire пристрою.

Розглянута вище робота мережі *MicroLAN* базується на декількох різних рівнях, кожен з них має свою назву: - фізичний рівень - це схемне рішення шини, рівні напруги, взаємодія елементів мережі на рівні електричних сигналів; рівень зв'язку - це правила формування сигналу скидання, синхронізація і спосіб передачі бітів інформації в обох напрямах. Ці два рівні визначаються фізичними особливостями мережі *MicroLAN*. Проте при описі протоколу прийнято виділяти ще два рівні, які пов'язані з логікою роботи протоколу. Це так звані ***мережевий*** і ***транспортний (функціональний)*** рівні. Будь-яка шина ***1-Wire*** завжди перебуває на одному з цих рівнів. Причому відразу після сигналу скидання шина переходить на мережевий рівень, і лише відпрацювавши

команду мережевого рівня, шина переходить на транспортний. Команди мережевого рівня призначені для адресації *Slave*-мікросхем. Це команди типу «Вибрати мікросхему» і вони є однаковими для всіх пристроїв ***1-Wire***.

Команди мережевого рівня

Назва команди	код	Опис команди
Читання ID	**0x33**	Читання індивідуального коду веденої мікросхеми із спеціального ПЗП (Для читання ID мікросхеми DS1990A також використовується команда 0x0F).
Збіг ID	**0x55**	Активація пристрою із заданим ID. Інші елементи переходять у пасивний режим.
Пропуск ID	**0xCC**	Пропуск команди вибору пристрою. Застосовується, якщо потрібно працювати відразу зі всіма пристроями, або він лише один.
Пошук ID	**0xF0**	Дана команда дозволяє здійснювати пошук елементів, підключених до шини. Одночасно з пошуком визначаються їхні ID.
Пошук сигналізації	**0xEC**	Дана команда подібна до команди пошук ID (0xF0), за винятком того, що на неї відповідатимуть тільки пристрої з встановленим прапором тривоги.

Команди функціонального (транспортного) рівня призначені для безпосередньої передачі інформації і залежить від конкретного пристрою ***1-Wire***. Це команди типу «записати», «прочитати».

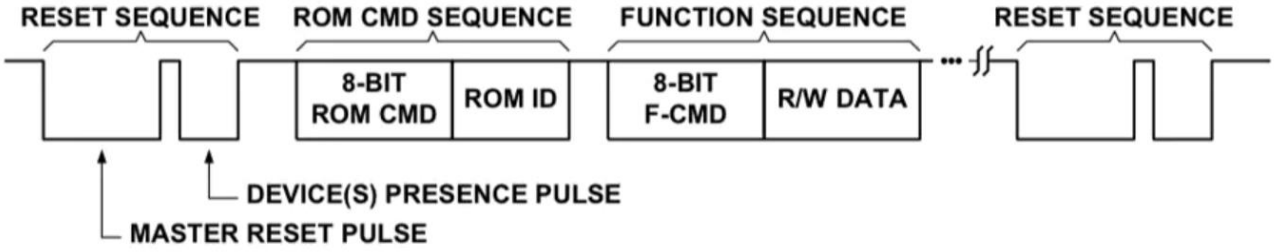

Рис 6. Діаграма передачі та прийоу даних по шині 1-Wire

Завдання до роботи:

1. Написати програму на мові Сі згідно з варіантом завдання.
2. Створити схему (Рис. 7) в програмі для моделювання (SimulIDE або Proteus 8). Провести моделювання написаної програми. Результати моделювання (скріншоти) добавити в звіт.
3. При наявності деталей зібрати відповідну до завдання частину схеми на макетній платі та запрограмувати мікроконтролер.

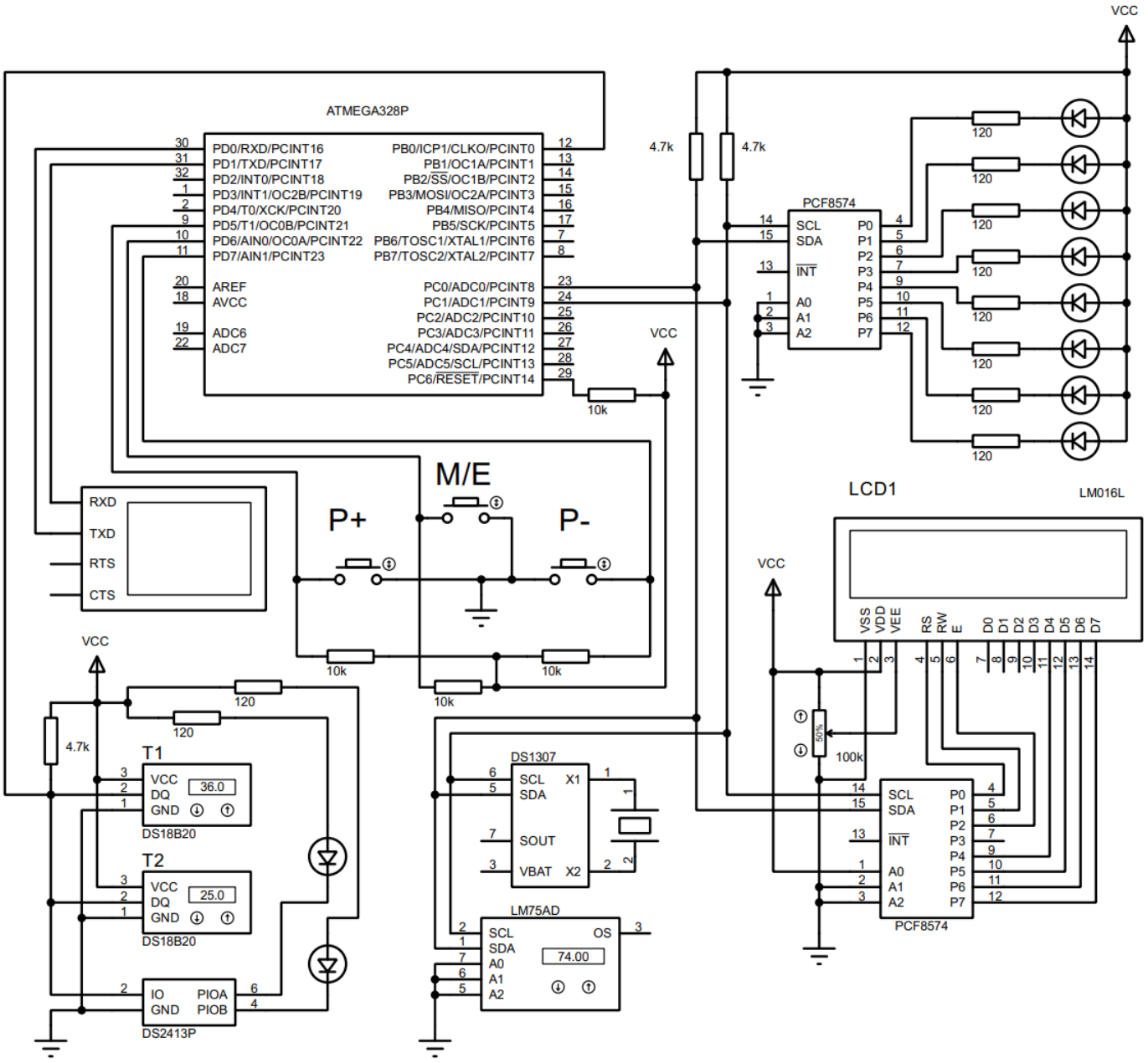

Рис 7. Тестова схема.

Демонстраційна програма для схеми Рис. 7. яка виводить на LCD індикатор строку *Hello World!* Індикатор підключений до мікроконтролера через I2C-експандер (PCF8574).

Текст файлу **main.c**

```
#include <avr/io.h>
#include <util/delay.h>
#include "i2c.h"

#define LCD_PIN_E  4
#define LCD_PIN_RS 1
#define LCD_PIN_RW 2
#define LCD_CMD  0
#define LCD_DATA 1
#define LCD_DISP_CLEAR   0x01
#define LCD_DISP_OFF     0x08
#define LCD_DISP_ON      0x0C
#define LCD_CURSOR_ON    0x0E
#define LCD_CURSOR_BLINK 0x0F
#define LCD_RETURN_HOME  0x02
#define LCD_ENTRY_MODE   0x06
#define LCD_4BIT_MODE    0x20
#define LCD_8BIT_MODE    0x30
#define LCD_2_ROWS       0x08
#define LCD_FONT_5x8     0x00
#define LCD_FONT_5x10    0x04
#define LCD_POSITION     0x80

#define LCD_I2C_ADDRESS 0x42

void LCD_E_pulse(uint8_t data)
{
  I2C_Write(data | LCD_PIN_E);
  _delay_us(1);
  I2C_Write(data & ~LCD_PIN_E);
  _delay_us(40);
}

void LCD_Send(uint8_t addr, uint8_t d, uint8_t type)
{
  uint8_t up_nibble = (d & 0xF0);
  uint8_t low_nibble = (d << 4);

    if(type) {
      up_nibble  |= LCD_PIN_RS;
      low_nibble |= LCD_PIN_RS;
    }
    I2C_Start();
    I2C_Write(addr);
    // Send upper nibble, E pulse
    I2C_Write(up_nibble);
    LCD_E_pulse(up_nibble);
    // Send lower nibble, E pulse
```

```
    I2C_Write(low_nibble);
    LCD_E_pulse(low_nibble);
    I2C_Stop();
    _delay_ms(5);
}

// Функція ініціалізації контролера LCD дисплея (HD44780)
uint8_t LCD_Init(uint8_t addr)
{
  uint8_t res, data = 0;
    I2C_Init();
    _delay_ms(16);
    I2C_Start();
    res = I2C_Write(addr);
    if(res != I2C_ACK) return 1;
    // DB7 BD6 DB5 DB4 P3 E RW RS
    // DB4=1, DB5=1 / BF cannot be checked in these instructions
    data = 0x30;
    I2C_Write(data);
    LCD_E_pulse(data);
    _delay_ms(5);  // delay > 4.1ms
    // DB4=1, DB5=1 / BF cannot be checked in these instructions
    I2C_Write(data);
    LCD_E_pulse(data);
    _delay_us(110);  // delay > 100us
    // DB4=1, DB5=1 / BF cannot be checked in these instructions
    I2C_Write(data);
    LCD_E_pulse(data);
    _delay_us(50);  // delay > 45us (=37+4 * 270/250)
    // DB5=1 / 4 bit mode 0x20 / BF cannot be checked in these instructions
    data = 0x20;
    I2C_Write(data);
    LCD_E_pulse(data);
    _delay_us(50);  // delay > 45us (=37+4 * 270/250)
    I2C_Stop();

    // 4 bit mode, 2 rows, font 5x8
    LCD_Send(addr, LCD_4BIT_MODE | LCD_2_ROWS | LCD_FONT_5x8, LCD_CMD);
    // display off 0x08 - send 8 bits in 4 bit mode
    LCD_Send(addr, LCD_DISP_OFF, LCD_CMD);
    // display clear 0x01 - send 8 bits in 4 bit mode
    LCD_Send(addr, LCD_DISP_CLEAR, LCD_CMD);
    // entry mode set 0x06 - send 8 bits in 4 bit mode
    LCD_Send(addr, LCD_ENTRY_MODE, LCD_CMD);
    LCD_Send(addr, LCD_DISP_ON, LCD_CMD);
  return 0;
}

void LCD_SetXY(uint8_t addr, uint8_t x, uint8_t y)
{
    if (y != 0) {
        x += 0x40;
    }
```

```
    LCD_Send(addr, LCD_POSITION | x), LCD_CMD);
}

void Lcd_SendData(uint8_t addr, char data)
{
  LCD_Send(addr, (uint8_t) data, LCD_DATA);
  //LCD_CheckBF(addr);
  //_delay_ms(50);
}

void LCD_Clear(uint8_t addr)
{
    LCD_Send(addr, LCD_DISP_CLEAR, LCD_CMD);
    _delay_ms(10);
}

void LCD_Print(uint8_t x, uint8_t y, char *str)
{
    LCD_SetXY(LCD_I2C_ADDRESS, x, y);
    while(*str)
    {
       Lcd_SendData(LCD_I2C_ADDRESS, *str++);
    }
}

//================================================================
int main(void)
{
  LCD_Init(LCD_I2C_ADDRESS);

  LCD_Print(0, 0,"Hello LCD!");

    // mail loop =============
    for(;;)
    {
      _delay_ms(1000);
    }
  return 0;
}
```

Текст файлу **i2c.h**

```
#ifndef __I2C_H__
#define __I2C_H__
#include <stdint.h>
#include <avr/io.h>

#define SDA_PIN 0
#define SCL_PIN 1
#define I2C_PORT PORTC
#define I2C_PIN PINC
#define I2C_DDR DDRC
```

```
#define I2C_ACK 0
#define I2C_NAK 1

void I2C_Init(void);
void I2C_Start(void);
void I2C_Stop(void);
uint8_t I2C_Write(uint8_t data);
uint8_t I2C_Read(uint8_t ack);
#endif
```

Текст файлу **i2c.c**

```
#include "i2c.h"
#include <util/delay.h>
#include <stdio.h>

#define SDA_HIGH() I2C_DDR &= ~(1<<SDA_PIN)
#define SDA_LOW() I2C_DDR |= (1<<SDA_PIN)
#define SCL_HIGH() I2C_DDR &= ~(1<<SCL_PIN)
#define SCL_LOW() I2C_DDR |= (1<<SCL_PIN)

void I2C_Init(void)
{
    // SCL = 1, SDA = 1
    I2C_DDR &= ~((1<<SDA_PIN)|(1<<SCL_PIN));
    I2C_PORT &= ~((1<<SDA_PIN)|(1<<SCL_PIN));
}

void I2C_Start(void)
{
    // SCL = 1, SDA = 1
    I2C_DDR &= ~((1<<SCL_PIN)|(1<<SDA_PIN));
    _delay_us(5); // HDEL
    SDA_LOW();    // sda = 0
    _delay_us(5); // HDEL
    SCL_LOW();    //??? scl = 0
}

void I2C_Stop(void)
{
    SDA_LOW();
    _delay_us(5); // HDEL
    SCL_HIGH();   // scl = 1;
    _delay_us(3); // QDEL
    SDA_HIGH();   // sda = 1;
    _delay_us(2);
}

uint8_t I2C_Write(uint8_t data)
{
  uint8_t i, ack;
    for (i = 0; i < 8; i++)
    {
```

```
        SCL_LOW();
        _delay_us(2);
        if (data & 0x80) {
            SDA_HIGH();
        } else {
            SDA_LOW();
        }
        _delay_us(4);
        SCL_HIGH();
        _delay_us(5);
        data <<= 1;
    }
    SCL_LOW();
    _delay_us(3);
    SDA_HIGH();
    _delay_us(5);
    SCL_HIGH();
    _delay_us(3);
    ack = I2C_PIN & (1<<SDA_PIN);
    _delay_us(2);
    SCL_LOW();
    _delay_us(5);
 return ack;
}

uint8_t I2C_Read(uint8_t ack)
{
  uint8_t i, data = 0;
    for (i = 0; i < 8; i++)
    {
        data <<= 1;
        SCL_LOW();
        _delay_us(5);
        SCL_HIGH();
        _delay_us(5); //while((I2C_PIN&(1<<SDA_PIN))==0){;}
        if (I2C_PIN & (1<<SDA_PIN)) { data |= 1; }
    }
    SCL_LOW();
    _delay_us(2);
    if (!ack) {
        SDA_LOW();
    } else {
        SDA_HIGH();
    }
    _delay_us(3);
    SCL_HIGH();
    _delay_us(5);
    SCL_LOW();
    _delay_us(5);
  return data;
}
```

Варіанти завдань

1. Написати програму яка виводить біжучий вогонь на світлодіодах підключених до I2C-експандера (PCF8574). Кнопки P+/P- збільшують/зменшують швидкість ефекту.
2. Модифікувати завдання 1. добавивши зміну ефекту кнопкою M/E та відображення на LCD індикаторі додаткової інформації (назва ефекту і його швидкість).
3. Використовуючи схему Рис 7. реалізувати термометр з виводом результату на LCD індикатор з періодом 5с. Для реалізації термометра використовувати мікросхему LM75AD.
4. Модифікувати завдання 3. замінивши мікросхему LM75AD на DS18B20.
5. Написати програму яка по черзі з періодом 1с засвічує і гасить світлодіоди підключені до мікросхеми DS2413P. (дана мікросхема підключена до мікроконтролера по шині 1-Wire)
6. Розробити 2-х канальний термометр з виводом даних на LCD індикатор використовуючи мікросхеми DS18B20. Оновлення даних відбувається кожні 2с.
7. (*****) Використовуючи мікросхему DS1307 (годинник реального часу) реалізувати «годинник» який відображає на LCD індикаторі поточну дату та час. Добавити можливість налаштування дати і часу з допомогою кнопок M/E, P+, P-.
8. (*) Написати програму яка здійснює пошук пристроїв підключених до шини 1-Wire та результат їхніх-ROM ідентифікаторів виводить в консоль (UART).