Лабораторна робота № 8.

Моделювання інтерфейсу TWI (I2C)

Тема: Моделювання інтерфейсу TWI (I2C)

Мета: Користуючись пакетом PROTEUS дослідити моделювання інтерфейсу TWI (I2C)

1. Порядок виконання роботи

- створити модель пристрою в пакеті Proteus 8.6
- розробити схему алгоритму роботи моделі та робочу програму
- створити hex-файл та підключити його до мікроконтролера
- запустити модель та виконати її дослідження згідно методичних вказівок
- зробити відповідні висновки.

2. Стислі теоретичні відомості

2.1 Схема моделі

Схему моделювання у пакеті Proteus 6 наведено на рисунку 1.

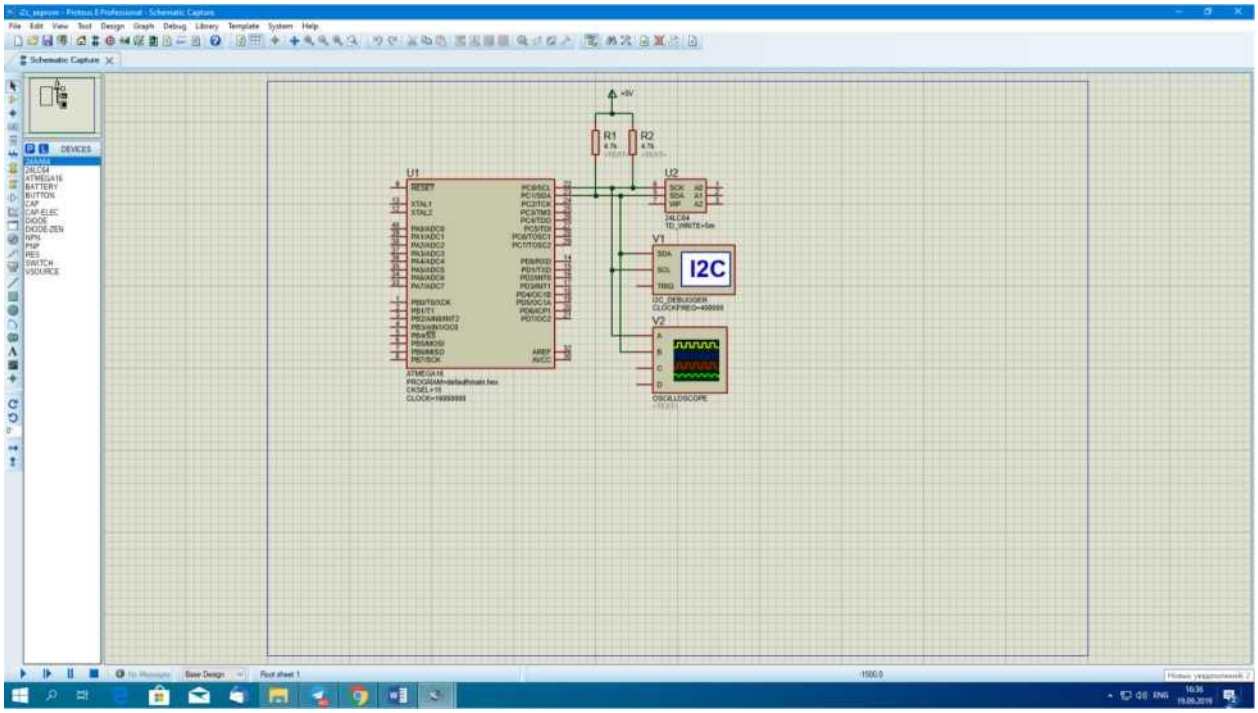

Рисунок 1 - Схема моделювання інтерфейсу I2C у пакеті Proteus 8.6

Опис моделі

У моделі використано мікроконтролер ATmega16. До нього додаються 2 резистори: R1 та R2 для підключення шини I^2C до джерела живлення. З вкладки «Віртуальні інструменти» береться осцилограф, I^2C-відлагоджувач та EEPROM-пам'ять даних із вбудованим інтерфейсом I^2C - 24LC64. Ця мікросхема має 8 виводів, які, крім живлення і землі, відображено на моделі (U2).

У налаштуваннях мікроконтролера вказано прошивку, частоту тактування: 16 МГц і виставлено CKSEL-фьюзи для кристала. У налаштуваннях I^2C-відлагодувача вказано, що тактова частота імпульсів шини дорівнює 400 КГц.

2.2. Налаштування частоти тактових імпульсів лінії SCL

Періодом тактових імпульсів на лінії SCL керує блок генератора швидкості зв'язку ведучого мікроконтролера. Період SCL змінюється шляхом програмування регістра швидкості зв'язку TWI (TWBR) і бітів переддільника у регістрі статусу TWI (TWSR).

Частота SCL визначається за формулою:

$$f_{SCL} = f_{CLK} / [16+2(TWBR) \cdot 4TWPS], \quad (1)$$

де:

TWBR - значення регістра швидкості зв'язку TWI;

TWPS - значення бітів переддільника в регістрі статусу TWI (TWSR);

f_{CLK} - тактова частота ведучого мікроконтролера;

f_{SCL} - частота тактових імпульсів лінії SCL.

Налаштування TWBR потрібно, тому що ведена мікросхема може обмінюватися даними на певній частоті. Наприклад, якщо в Proteus, ввести пошук I^2C MEM, то побачите велику кількість відповідних мікросхем пам'яті, для яких в основному вказані частоти 100 і 400КГц. Для обраної в моделі мікросхеми EEPROM-пам'яті тактова частота імпульсів шини дорівнює 400 КГц.

Підставляючи у формулу частоти f_{CLK} та f_{SCL}, ми зможемо знайти

оптимальне значення для регістра TWBR.

TWPS - це 2-бітове число [TWPS1: TWPS0], перший біт - це розряд TWPS0, другий - TWPS1 в регістрі статусу TWSR. Задаючи значення переддільника швидкості зв'язку 4TWPS, ми можемо зменшити значення TWBR, тому що TWBR має довжину вісім біт, а відповідно максимальне значення 255. Але зазвичай це не потрібно, тому TWPS задається 0 і 4 TWPS = 1. Набагато частіше, навпаки, важливе значення нижнього діапазону регістра TWBR. Якщо TWI працює у ведучому режимі, то значення TWBR повинно бути не менше 10. Якщо значення TWBR менше 10, то ведучий пристрій шини може генерувати не коректні сигнали на лініях SDA і SCL під час передачі байта [2].

Таблиця 1

TWPS1	TWPS0	$TWPS_{10}$	Значення переддільника 4TWPS
0	0	0	1
0	1	1	4
1	0	2	16
1	1	3	64

Таблиця 2

Частота МК, МГц	TWBR	TWPS	Частота SCL, КГц
16.0	12	0	400
16.0	72	0	100
14.4	10	0	400
14.4	64	0	100
12.0	52	0	100
8.0	32	0	100
4.0	12	0	100
3.6	10	0	100

Таким чином для налаштування TWI необхідно:

- завдати значення переддільника [TWP81: TWPS0] у регістрі статусу TWSR;
- завдати швидкість зв'язку у регістрі TWBR.

2.3. Запуск процесу моделювання

На рисунку 2 наведено стан моделі після запуску процесу моделювання (після натискання клавіши «START»).

Як бачимо, що в PORTA записалося число 0b0000’1010 або 0x0A або 10, тобто змінна uint8_t byte містить вірне значення після читання пам’яті 24LC64 за адресою 0x19.

Нижче наведено пояснення окремих етапів обміну TWI (I2C) - шиною, які відображено у вікні I^2C Debug (рисунок 3), та часових діаграм обміну між мікроконтролером та EEPROM-пам’яттю, отриманих на екрані осцилографа (рисунок 4).

Як видно з цих рисунків, результати моделювання підтверджують теоретичні відомості, які наведено на рисунку 5. Адреса веденого пристрою в нашому прикладі дорівнює: 0x19, а байт даних, який спочатку записується, а потім зчитується за цією адресою, дорівнює: 0x0A.

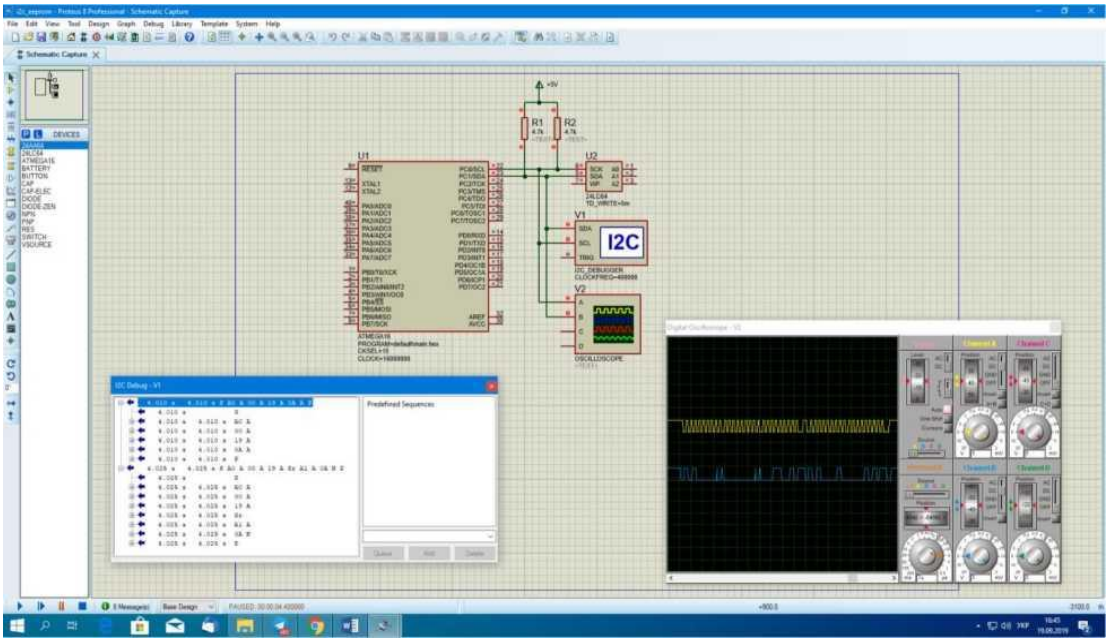

Рисунок 2 - Стан моделі після запуску процесу моделювання

A - підтвердження від ведучого або веденого, N - відсутність підтвердження

Рисунок 3 - Пояснення окремих етапів обміну шиною TWI (I2C)

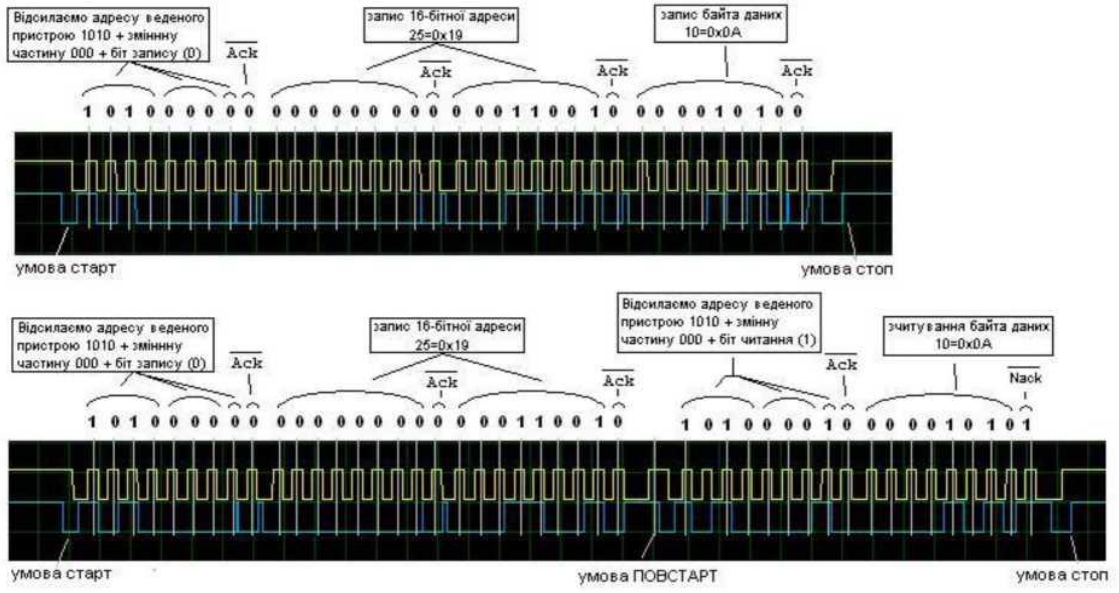

Рисунок 4 - Пояснення часових діаграм обміну між мікроконтролером та EEPROM - пам’яттю, отриманих на екрані осцилографа

Рисунок 5 - Приклад звернення до зовнішньої пам’яті даних типу EEPROM

2.4. Схема алгоритму роботи

Схема алгоритму роботи моделі наведено на рисунках 6 … 8.

Згідно з описом роботи моделі, який наведено вище, спочатку МК передає веденому пристрою (EEPROM -пам’яті) байт даних 0x10 за адресою 0x19, а потім читає вміст комірки EEPROM -пам’яті з тією самою адресою. Отримання в цьому випадку значення 0x10 говорить про те, що модель працює правильно. Через те, що МК моделі спочатку працює як ведучий передавач, а потім, як ведений приймач, а EEPROM -пам’ять спочатку виконує функцію веденого приймача, а потім ведучого передавача, нижче на рисунках 6 ... 8 наведено схеми алгоритмів роботи МК та EEPROM -пам’яті у цих режимах.

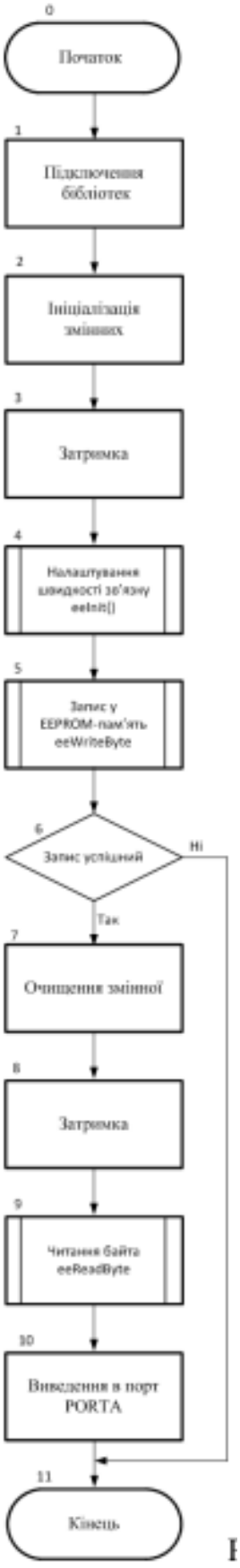

Рисунок 6 – Схема алгоритму роботи моделі

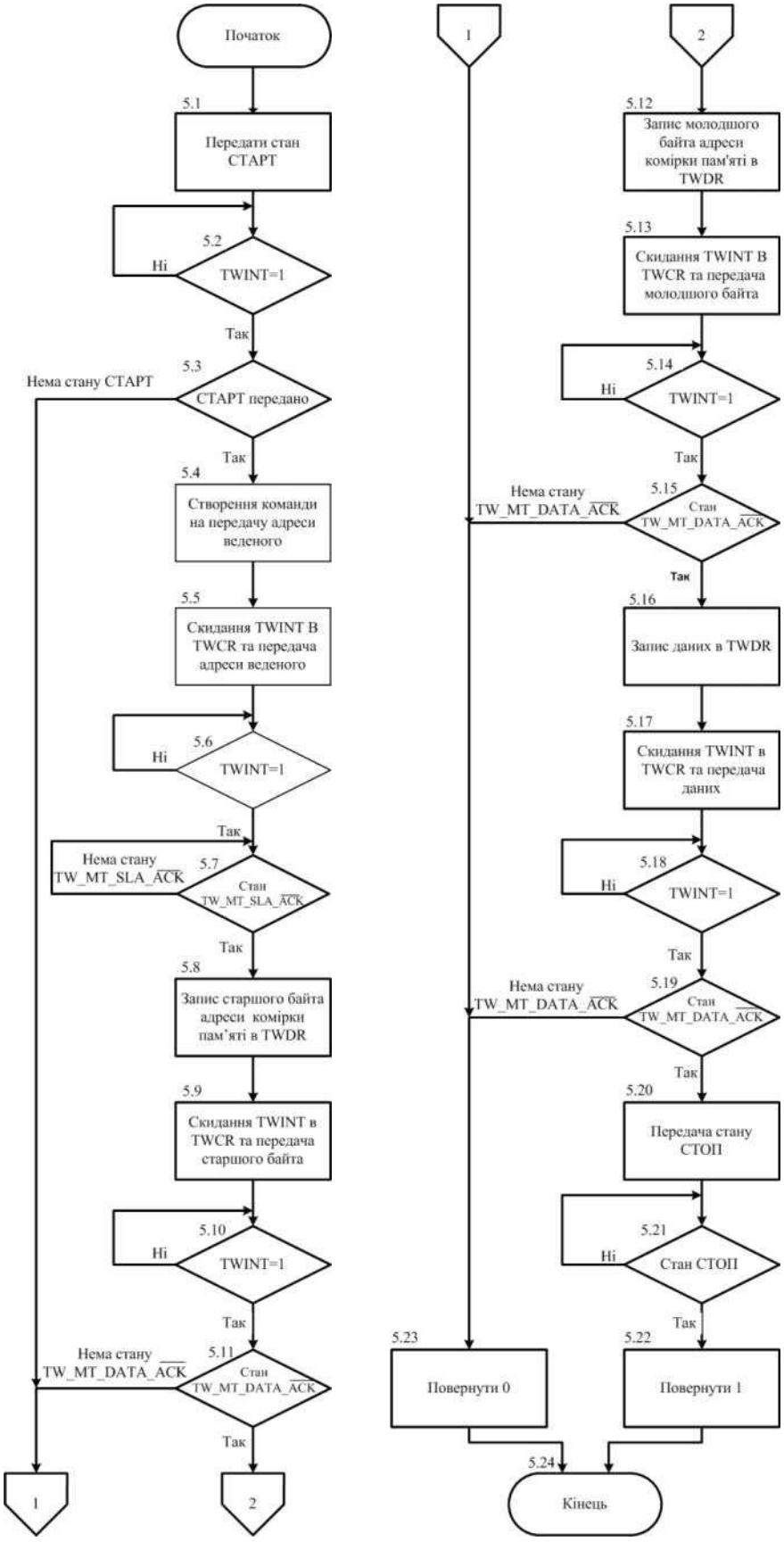

Рисунок 7 - Схема алгоритму запису у EEPROM -пам’ять

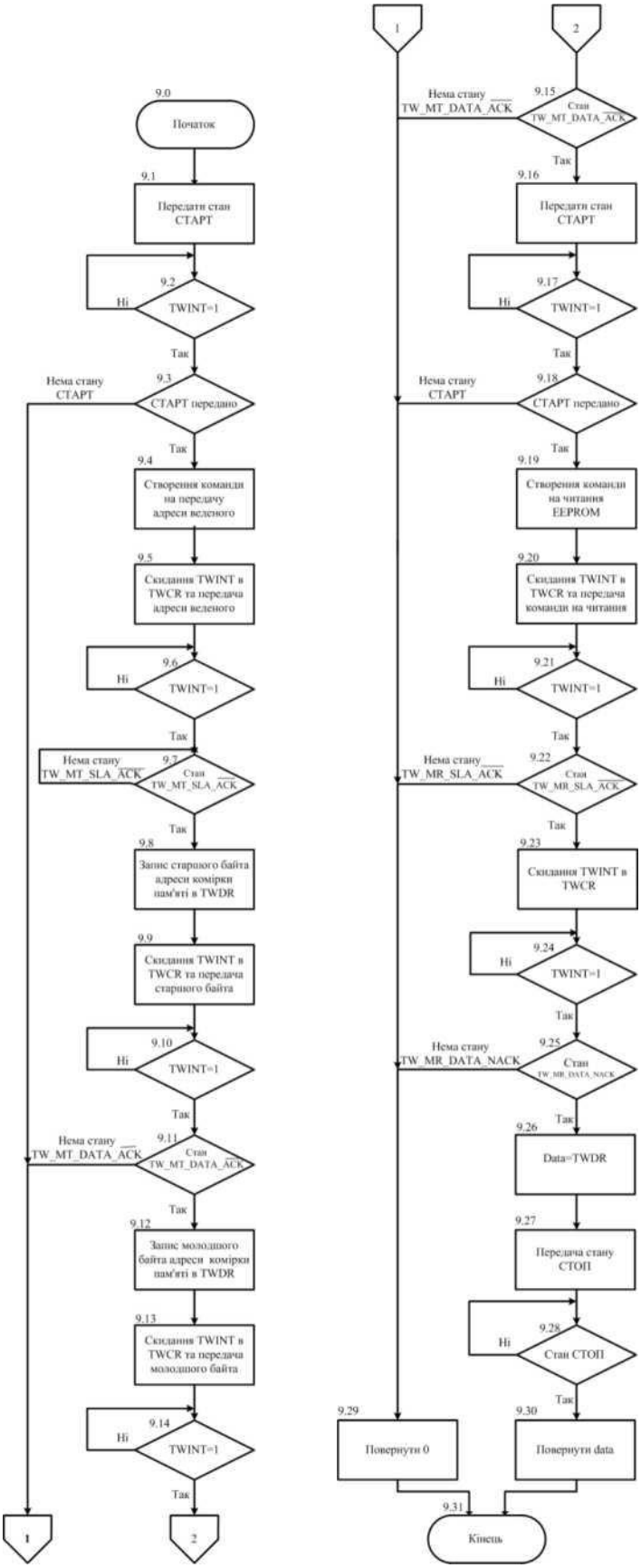

Рисунок 8- Схема алгоритму читання EEPROM -пам’яті

Робоча програма

Всі функції (ініціалізація TWI, читання, запис зовнішньої пам'яті) винесено в окремі файли i2c_eeprom.c і i2c_eeprom.h.

Зміст i2c_eeprom.h

```
#define false 0
#define true 1 #define slaveF_SCL 400000 //400 Khz

#define slaveAddressConst 0b1010
#define slaveAddressVar 0b000

#define READFLAG 1 // read
#define WRITEFLAG 0 // write

void eelnit(void);
uint8_t eeWriteByte(uint16_t address,uint8_t data);
uint8_t eeReadByte(uint16_t address);

                                    // Master
#define TW_START              0x08 //Було сформовано стан СТАРТ
#define TW_REP_START          0x10 //Було сформовано стан ПОВСТАРТ
// Master Transmitter
#define TW_MT_SLA_ACK              0x18 //було передано стан SLA+W і
                                        //прийнято підтвердження ACK
#define TW_MT_SLA_NACK        0x20 //Було передано пакет SLA+W і //прийнято
                                    непідтвердження NACK
#define TW_MT_DATA_ACK          0x28 //Було передано пакет даних і //прийнято
                                    підтвердження ACK
#define TW_MT_DATA_NACK     0x30 //Було передано пакет даних і //прийнято
                                    непідтвердження NACK
#define TW_MT_ARB_LOST          0x38 //Втрата пріоритету при передачі
                                    //пакета адреси або даних
// Master Receiver
#define TW_MR_ARB_LOST          0x38 //Втрата пріоритету при передачі
                                    //пакета адреси або даних
#define TW_MR_SLA_ACK           0x40 //Було передано пакет SLA+R і
                                        //прийнято підтвердження ACK
#define TW_MR_SLA_NACK        0x48 //Було передано пакет SLA+R і //прийнято
                                    непідтвердження NACK
#define TW_MR_DATA_ACK          0x50 //Було прийнято байт даних і //передано
                                    підтвердження ACK
#define TW_MR_DATA_NACK         0x58 //Було прийнято байт даних і //передано
                                    не підтвердження NACK
// Slave Transmitter
#define TW_ST_SLA_ACK       0xA8 //Було прийнято SLA+R з власною
                                                    // адресою і послано підтвердження ACK
#define TW_ST_ARB_LOST_SLA_ACK 0xB0 //Втрата пріоритету в
```

```
                                                    //режимі ведучого
    //під час передачі SLA+R/W
#define TW_ST_DATA_ACK          0xB8 //Було передано байт даних і
                                    //передано підтвердження ACK
#define TW_ST_DATA_NACK         0xC0 //Було передано останній байт
                                    //даних і отримано
                                //непідтвердження NACK
#define TW_ST_LAST_DATA         0xC8 //Було передано останній байт даних
                                // і отримано підтвердження ACK

// Slave Receiver
#define TW_SR_SLA_ACK 0x60 //Було прийнято SLA+W з власною
                                //адресою і послано підтвердження ACK
#define TW_SR_ARB_LOST_SLA_ACK 0x68 //Втрата пріоритету в
                                    //режимі ведучого
//під час передачі SLA+R/W
#define TW_SR_GCALL_ACK 0x70 //Було прийнято загальний виклик і
                                //послано підтвердження ACK
#define TW_SR_ARB_LOST_GCALL_ACK 0x78 //Втрата пріоритету в
                                    //режимі ведучого
    //під час передачі SLA+R/W
#define TW_SR_DATA_ACK          0x80 //Пристрій уже адресовано: було
                                    //прийнято байт даних і
                                //послано підтвердження ACK
#define TW_SR_DATA_NACK         0x88 //Пристрій уже адресовано:
                                    //було прийнято байт даних і
                                    //послано непідтвердження NACK
#define TW_SR_GCALL_DATA_ACK 0x90 //Пристрій уже адресовано
                                    //(загальний виклик):
                                    //було прийнято байт даних і
                                    //послано підтвердження ACK
#define TW_SR_GCALL_DATA_NACK 0x98 //Пристрій уже адресовано
                                    //(загальний виклик): було прийнято
//байт даних та послано //непідтвердження NACK
#define TW_SR_STOP 0xA0 //Було виявлено стан СТОП або ПОВСТАРТ
                                //у той час, коли пристрій було
                                //адресовано в якості веденого
// Mise
#define TW_NO_INFO              0xF8 //Немає інформації TWINT = 0
#define TW_BUS_ERROR            0x00 //Помилка на шині в результаті
//некоректного формування стану //СТАРТ або СТОП
```

Лістинг файлу i2c_eeprom.c

При написанні програми мовою С у змінні: TW_MT_SLA_ACK; TW_MT_DATA_ACK та TW_MR_SLA_ACK входить скорочення ACK (acknowledge) - підтвердження. Це скорочення відображає отримання сигналу низького рівня (підтвердження) після передачі адреси або даних (рис. 7, 8).

Щоб підкреслити активний низький рівень цього сигналу на рисунках 7, 8 він помічений інверсією. Але при написанні програм мовою С використання інверсії заборонено. Тому названі вище змінні мають скорочення ACK без інверсії.

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

#include "i2c_eeprom.h"

void eelnit(void)
{
  /* Налаштування генератора швидкості зв’язку */
  TWBR = (F_CPU/slaveF_SCL - 16)/(2 * /* TWI_Prescaler= 4^ATWPS */1);

/*
Якщо TWI працює у ведучому режимі, то значення TWBR повинно бути не менше 10. Якщо значення TWBR менше 10, то ведучий пристрій шини може генерувати некоректні сигнали на лініях SDA і SCL під час передачі байта.*/

  if(TWBR < 10)
      TWBR = 10;

   /*
   Налаштування переддільника у регістрі статусу блоку управління.
   Очищаються біти TWPS0 і TWPS1 регістра статусу, встановлюючи тим самим, значення переддільника = 1.
   */
  TWSR &= (~((1<<TWPS1)|(1<<TWPS0)));
}

uint8_t eeWriteByte(uint16_t address,uint8_t data)
{
  do
  {
        TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN); //5.1 Передача стану
                                              //СТАРТ

        while(!(TWCR & (1<<TWINT))); //5.2 Очікування встановлення TWINT

        if((TWSR & 0xF8) != TW_START)         //5.3 Перевірка регістру стану
                                          //5.3 return false;

TWDR = (slaveAddressConst<<4) + //5.4 Створення команди на (slaveAddressVar<<1) + (WRITEFLAG); //5.4 передачу

TWCR=(1<<TWINT)|(1<<TWEN); //5.5 Скидання TWINT в TWCR та
                                    // передача адреси веденого
```

```
while(!(TWCR & (1<<TWINT))); //5.6 Очікування встановлення TWINT

   }while((TWSR & 0x18) != TW_MT_SLA_ACK); //5.7 Перевірка стану
                                                    //5.7 TW_MT_SLA_ACK
TWDR=(address>>8);          //5.8 Завантаження старшого
                           //байта адреси комірки пам'яті у TWDR

TWCR=(1<<TWINT)|(1<<TWEN);          //5.9 Скидання TWINT та
                                     //передача старшого байта адреси
                                     //комірки в EEPROM

while(!(TWCR & (1<<TWINT)));     //5.10 Очікування встановлення TWINT


if((TWSR & 0x28) != TW_MT_DATA_ACK) //5.11 Перевірка стану
                                          //TW_MT_DATA_ACK

return false;                              //5.23

  TWDR=(address);                  //5.12 Завантаження молодшого
                                    // байта адреси комірки пам'яті у TWDR

  TWCR=(1<<TWINT)|(1<<TWEN);          //5.13 Скидання TWINT та
                                       // передача молодшого байта
                                       //адреси комірки в EEPROM
  while(!(TWCR & (1<<TWINT)));   //5.14 Очікування встановлення TWINT
    if((TWSR & 0x28) != TW_MT_DATA_ACK) //5.15 Перевірка стану
                                          // TW_MT_DATA_ACK return
          false;                          //5.23

                /***** Передача даних в EEPROM ********/


    TWDR=(data);                  //5.16 Завантаження даних у TWDR

    TWCR=(1<<TWINT)|(1<<TWEN);          //5.17 Скидання TWINT та
                                         // передача даних в EEPROM
    while(!(TWCR & (1<<TWINT))); //5.18 Очікування встановлення TWINT

    if((TWSR & 0x28) != TW_MT_DATA_ACK) //5.19 Перевірка стану
                                          // TW_MT_DATA_ACK
return false;                              //5.23


TWCR=(1<<TWINT)|(1<<TWEN)|(1<<TWSTO);//5.20 Передача умови СТОП while(TWCR &
    (1<<TWSTO));                  //5.21 Очікування передачі умови СТОП

    return true;                  //5.22
}
```

```
uint8_t eeReadByte(uint16_t address)
{   .                .
   uint8_t data; // Змінна, до якої записується прочитаний байт
                /***** Встановлення зв'язку із веденим ********/
   do
   {
      TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN); //9.1 Передача стану СТАРТ

      while(!(TWCR & (1<<TWINT)));   //9.2 Очікування встановлення TWINT

      if((TWSR & 0xF8) != TW_START)        //9.3 Перевірка регістру стану
         return false;                     //9.29

         //9.4 Створення команди на передачу
      TWDR = (slaveAddressConst<<4) +(slaveAddressVar<<1) + WRITEFLAG;

      TWCR=(1<<TWINT)|(1<<TWEN); //9.5 Скидання TWINT в TWCR та
                                 // передача адреси веденого
      while(!(TWCR & (1<<TWINT))); //9.6 Очікування встановлення TWINT

   }while((TWSR & 0x18) != TW_MT_SLA_ACK); //9.7 Перевірка стану
                                           // TW_MT_SLA_ACK

                /*****Передача адреси читання********/
   TWDR=(address>>8);                 //9.8 Завантаження старшого
                                      // байта адреси комірки EEPROM у TWDR
TWCR=(1<<TWINT)|(1<<TWEN);            //9.9 Скидання TWINT та
//передача старшого байта адреси комірки EEPROM
   while(!(TWCR & (1<<TWINT)));    //9.10 Очікування встановлення TWINT

   if((TWSR & 0x28) != TW_MT_DATA_ACK)       //9.11 Перевірка стану
                                             // TW_MT_DATA_ACK
return false;                                //9.29
   TWDR=(address);                    //9.12 Завантаження молодшого байта
                                      // адреси комірки пам'яті у TWDR
   TWCR=(1<<TWINT)|(1<<TWEN); //9.13 Скидання TWINT та передача
                              // молодшого байта адреси комірки
                              // в EEPROM
   while(!(TWCR & (1<<TWINT)));    //9.14 Очікування встановлення TWINT

   if((TWSR & 0xF8) != TW_MT_DATA_ACK) //9.15 Перевірка стану
                                       //TW_MT_DATA_ACK
      return false;                    //9.29

                /*****Перехід у режим читання********/
/* Знову зв'язок з веденим, тому що раніше було передано адресний пакет (slaveAddressConst
<<4) + (slaveAddressVar <<1) + WRITEFLAG, щоб записати адресу читання байта даних. Тепер
потрібно перейти у режим читання, щоб прочитати байт даних), для цього передаємо новий
пакет                (slaveAddressConst <<4) + (slaveAddressVar <<1) + READFLAG .*/
```

```
   //Повторення умови початку передачі
   T\CE=(1<<T\IKT)|(1<<T\8TA)|(1<<T\EX);          //9.16 Передача стану //Повторний
 старт
     while(!(TWCR & (1<<TWINT)));                    //9.17 Очікування TWINT=1

     if((TWSR & 0x10) != TW_REP_START)            //9.18 Перевірка регістру стану
        return false;                                 //9.29

   //9.29 Створення команди на читання
    T\\1Ж = (8IayeAййге88Соп81<<4) + (8IayeAййге88Уаг<<1) + КЕЛОЕІ.ЛСт;

          //Передача
T\CE=(1<<T\IXT)|(1<<T\EX);              //9. 20 Скидання TWINT та передача
                                                  //адреси веденого
    while(!(TWCR & (1<<T\ЮТ)));          //9. 21 Очікування встановлення T\\I\T

/* Перевірка, що знайшовся ведений з адресою 1010’000 і він готовий до читання */
     if((TWSR & 0x40) != TW_MR_SLA_ACK)       //9. 22 Перевірка стану
                                                        // TW_MR_SLA_ACK
 return false;                                        //9. 29


                /*****Прийом байта даних********/

/* Початок прийому даних шляхом очищення прапорця переривання ТАЮТ. Байт, який буде
прийнятий, записується у регістр TWDR.*/
    TACE (I ГАІ\T) (I· ■ TAE\);                    //9.23 Скидання T\¥I\T та
                                                        //встановлення TAE^I

   //Чекання закінчення прийому.
   while(!(TWCR & (1<<TWINT)));              //9.24 Чекання закінчення прийому

/* Перевірка регістра статусу. За протоколом, прийом даних повинен закінчуватися без
підтвердження з боку ведучого (TW_MR_DATA_NACK = 0x58)*/
    if((TWSR & 0x58) != TW_MR_DATA_NACK) //9.25 Перевірка регістра статусу
       return false;                                  //9.29

   data=TWDR;     //9.26 Надання змінній data значення TWDR

T\\44< (Ч      ІЛСІХТ) (І     ІЛ\4Л) (І      ІЛХАТО); //9.27 Встановлення умови СТОП

   \\'1iiIe(T\\CI< & (1<<T\8TO));      //9.28 Чекання встановлення умови СТОП

   return data;          //9.30 Повернення прочитаного байта
                                                  //9.31
}
```

Створення проекту в AVR STUDIO

У налаштуваннях проекту вказують ATmega16 та підключають файли i2c_eeprom.c та i2c_eeprom.h (рис. 9).

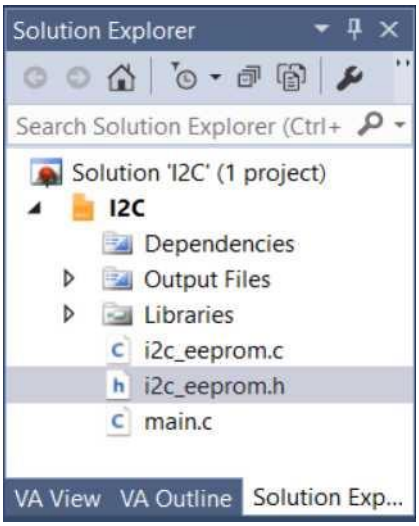

Рисунок 9 - Структура проекту

Лістинг файлу main.c

```
#define F_CPU 16000000UL
#include <avr/io.h>                                    //1
#include "util/delay.h"
#include "i2c_eeprom.h"
int main(void)
{
        uint8_t byte = 10;                             //2
        uint16_t address = 25;                         //2

/*Треба почекати, тому що віртуальний осцилограф у Proteus включається із затримкою */
        _delay_ms(300);                                //3

        //Налаштування TWI
        eeInit();                                      //4

        // Запис байта даних 10 = 0xA0 за адресою 25 = 0x19
        // Якщо запис успішний, то повернеться true
        if(eeWriteByte(address, byte))                 //5 ; 6
        {                          .
                // Очищення змінної
                byte = 0;                              //7

                // Чекаєння 15 с, щоб візуально розрізнити осцилограми
                _delay_ms(15);                         //8

                // Читання байта, розташованого за адресою 25 = 0x19
```

```
            byte = eeReadByte(address);          //9 // Перевірка виведення цього байта у порт
            виведення (PORTA)

            PORTA = byte;                        //10
        }
}                                                //11
```

3 Зміст звіту

Звіт по роботі повинен містити:

- схему моделі;
- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

Контрольні запитання

1. Якою фирмою було розроблено інтерфейс I^2C та для чого він призначений?
2. Чим відрізняється інтерфейс I^2C від інтерфейсу TWI?
3. З яких двох ліній складається шина інтерфейсу I^2C (TWI) та як вони повинні бути підключені в мережі?
4. Поясніть призначення функції «монтажне АБО/І» та як вона використовується в I^2C (TWI)-мережі?
5. Наведіть та поясніть структурну схему типової мережі, яка використовує для обміну даними шину I^2C (TWI).
6. Опишіть роботу пристроїв, які використовують для обміну інтерфейс I^2C (TWI).
7. Опишіть чотири можливі режими обміну даними інтерфейсом I^2C (TWI).
8. Поясніть призначення станів «СТАРТ» і «СТОП» шини TWI. Яким чином вони формуються?
9. У чому полягає задача розподілу пріоритетів при підключенні до шини TWI декількох ведучих пристроїв? Як вона вирішується?
10. Опишіть формат адресного пакету та пакету даних.
11. Наведіть та поясніть структурну схему типового модуля TWI.
12. Що задає контролер швидкості передачі?
13. Які два вузли входять до складу блоку шинного інтерфейсу? Які їх функції?

14. Що перевіряє блок контролю адреси?
15. При виникненні яких подій блок керування здійснює формування запиту на переривання?
16. Назвіть та опишіть використання регістрів керування TWI-модулем.
17. На чому базується взаємодія прикладної програми з модулем TWI?
18. З яких трьох частин складається будь-який етап взаємодії прикладної програми з модулем TWI? Поясніть цю взаємодію.
19. Назвіть деякі пристрої, які підтримують інтерфейс TWI.
20. У яких режимах може працювати модуль TWI, реалізований у мікроконтролерах сімейства Mega?
21. Опишіть роботу модуля TWI у режимі «Ведучий передавач».
22. Опишіть роботу модуля TWI у режимі «Ведучий приймач».
23. Опишіть роботу модуля TWI у режимі «Ведений приймач».
24. Опишіть роботу модуля TWI у режимі «Ведений передавач».
25. Які можливості надає використання стану «ПОВСТАРТ»?
26. Що трапиться, якщо адресація пристрою відбудеться при перебуванні мікроконтролера у «сплячому» режимі?
27. За якими сценаріями може розвиватися процес арбітражу?
28. З якою швидкістю може виконуватися обмін інформацією в I^2C (TWI)-мережі?
29. На яку відстань може передаватися інформація в I^2C (TWI)-мережі?
30. Скільки ведучих пристроїв може одночасно бути I^2C (TWI)-мережі?
31. Опишіть схему моделювання інтерфейсу TWI у пакеті Proteus 8.6.
32. Опишіть схему алгоритму роботи моделі.
33. Опишіть схему алгоритму запису в EEPROM-пам'ять.
34. Опишіть схему алгоритму читання EEPROM-пам'яті.
35. Поясніть окремі етапи обміну шиною TWI (I^2C), які отримано у вікні I^2C Debug після запуску моделювання.
36. Поясніть часові діаграми обміну між мікроконтролером та EEPROM-пам'яттю, отриманих на екрані осцилографа після запуску моделювання.