Вбудовані системи

I2C Шина

I2C Шина

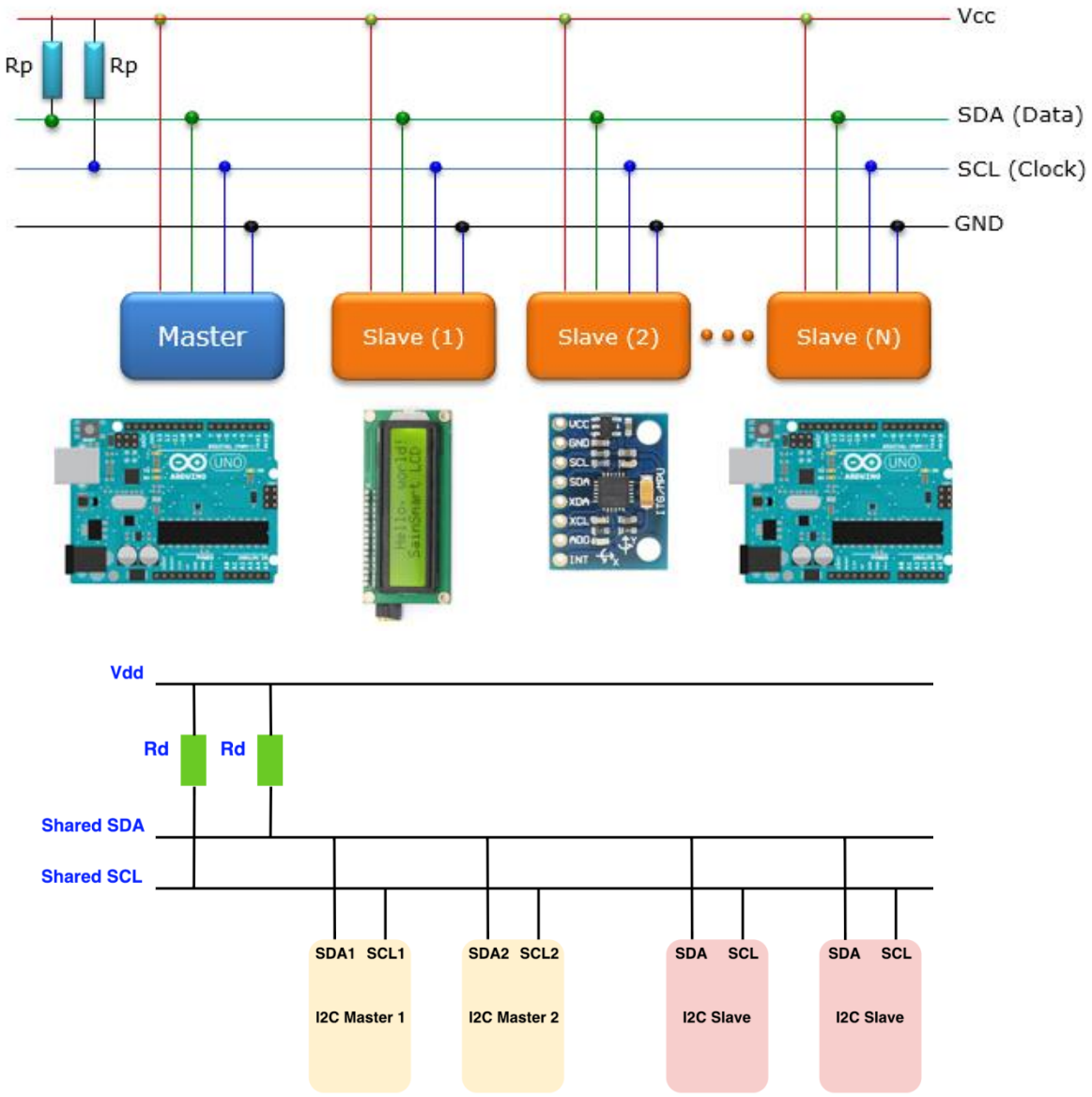
I2C (**I**nter-**I**ntegrated **C**ircuit) — послідовна шина даних для зв'язку інтегральних схем, розроблена фірмою **Philips** на початку 1980-х як проста шина для створення внутрішнього зв'язку. Використовується для з'єднання низькошвидкісних периферійних компонентів з материнською платою, вбудованими системами та різними мобільними пристроями.

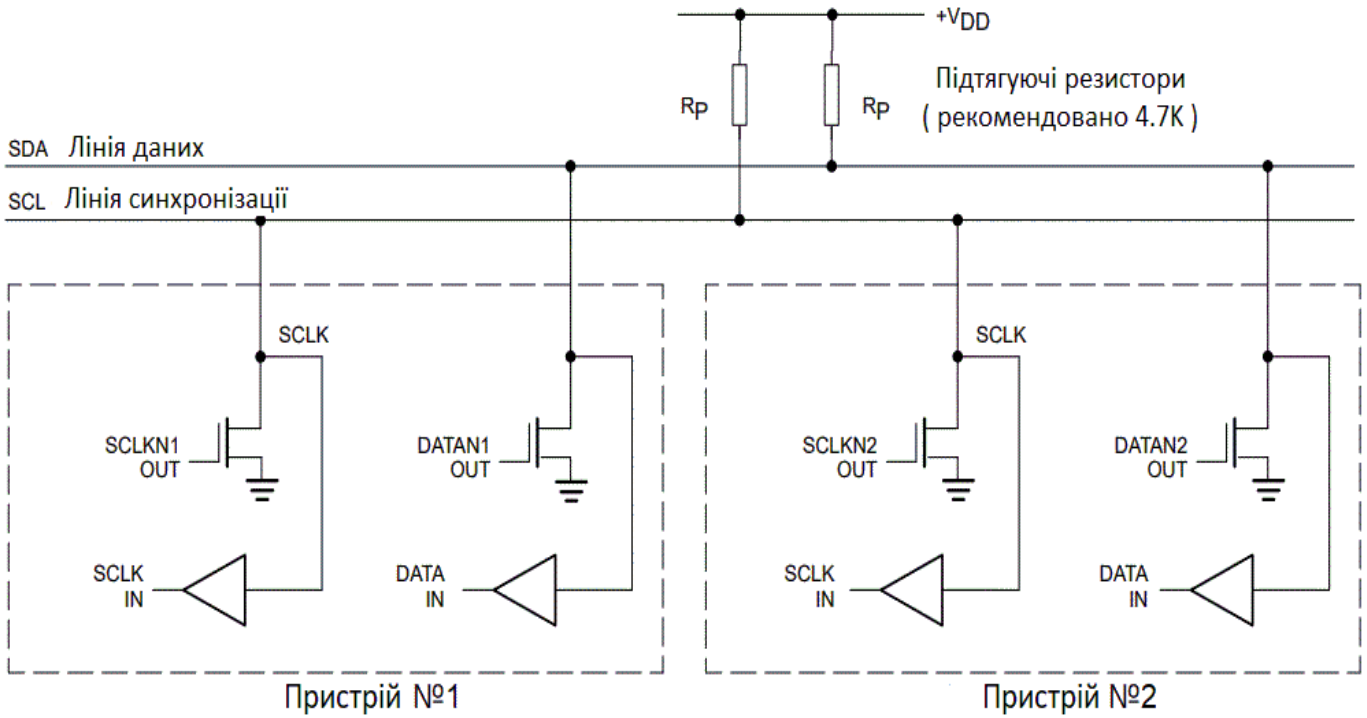
I2C використовує дві двохнаправлені лінії підтягнуті до напруги живлення, які керуються через відкритий колектор або відкритий стік — послідовна лінія даних (**SDA** - ***S**erial **D**ata*) і послідовна лінія тактування (**SCL** - ***S**erial **C**lock*). Стандартні напруги +5 В або +3,3 В, проте допускаються й інші.

Класична адресація включає 7-бітовий адресний простір з 16 зарезервованими адресами. Це означає до 112 вільних адрес для підключення периферії на одну шину. Максимальна допустима кількість мікросхем, приєднаних до однієї шини, обмежується максимальною ємністю шини в 400 пФ. Швидкість обміну в основному режимі роботи рівна 100 кбіт/с, або 10 кбіт/с в режимі роботи із зниженою швидкістю.

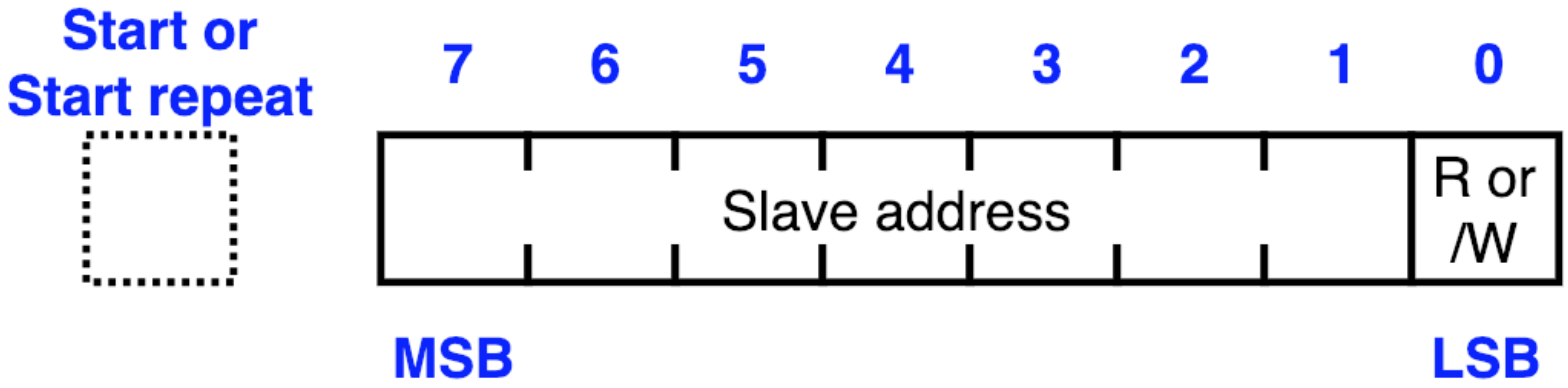
Після перегляду стандарту 1992 р. стає можливим підключення ще більшої кількості пристроїв на одну шину (за рахунок можливості 10-бітної адресації), а також велику швидкість до 400 кбіт/с у швидкісному режимі. Відповідно, доступна кількість вільних вузлів зросла до 1008. Версія стандарту 2.0, випущена 1998 р. представила високошвидкісний режим роботи зі швидкістю до 3,4 Мбіт/с зі зниженим енергоспоживанням.

Підключення пристроїв до шини I2C

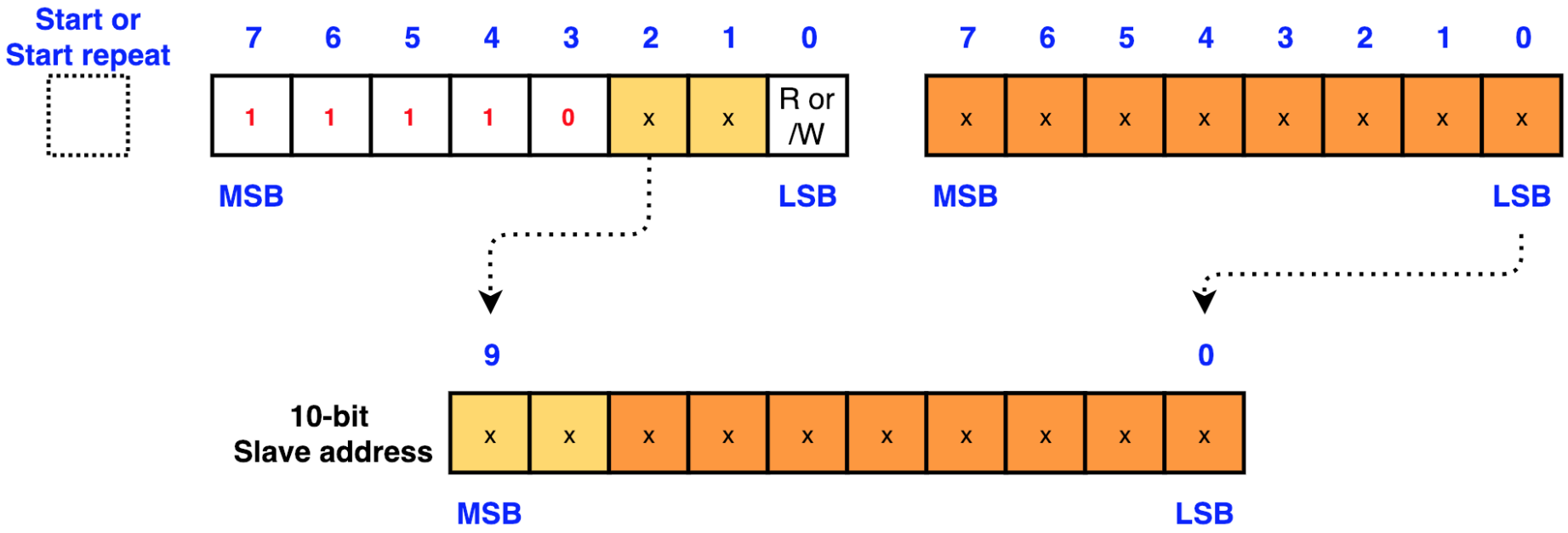




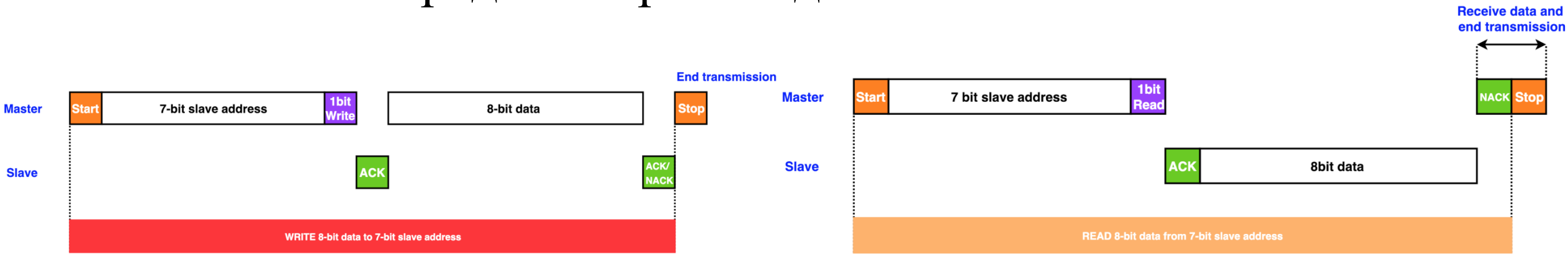
Адресація пристроїв на шині I2C

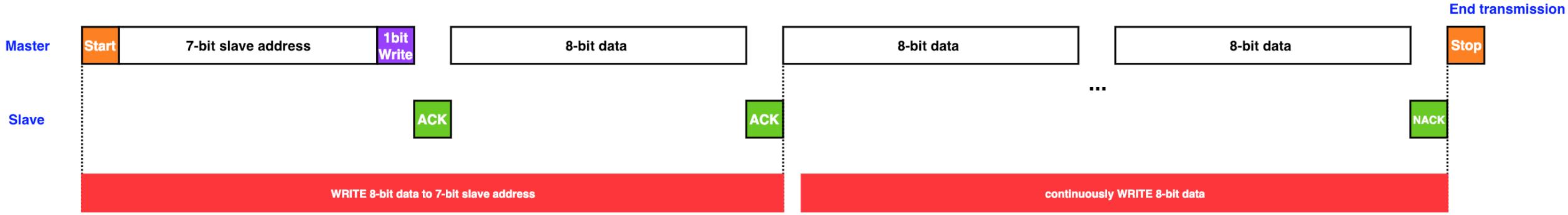


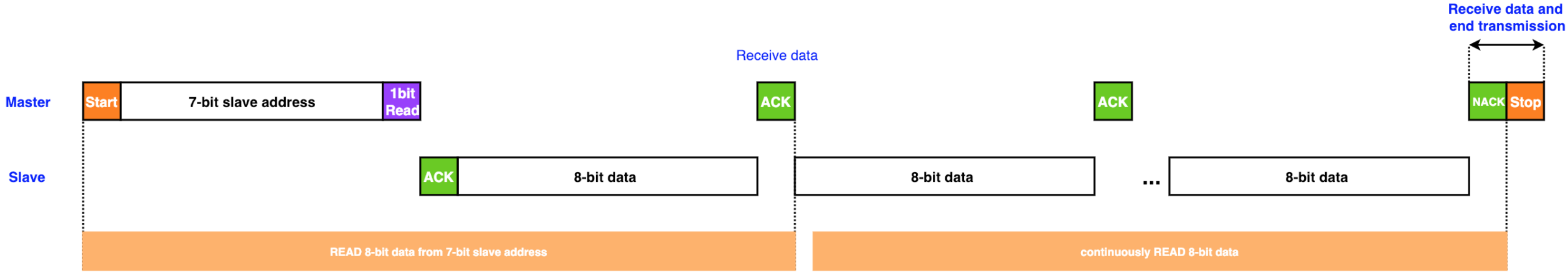
Bit 0 = 0: Master **writes** to **"Slave address"**
Bit 0 = 1: Master **reads** from **"Slave address"**



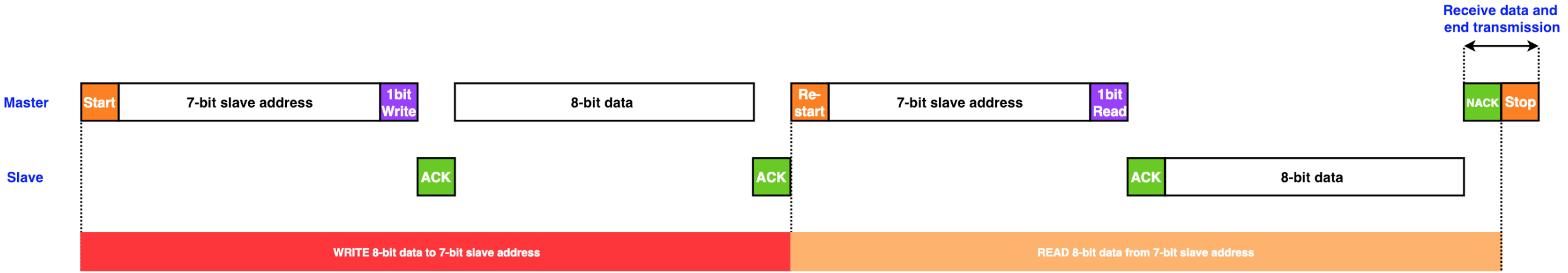
Передача і прийом даних по шині I2C

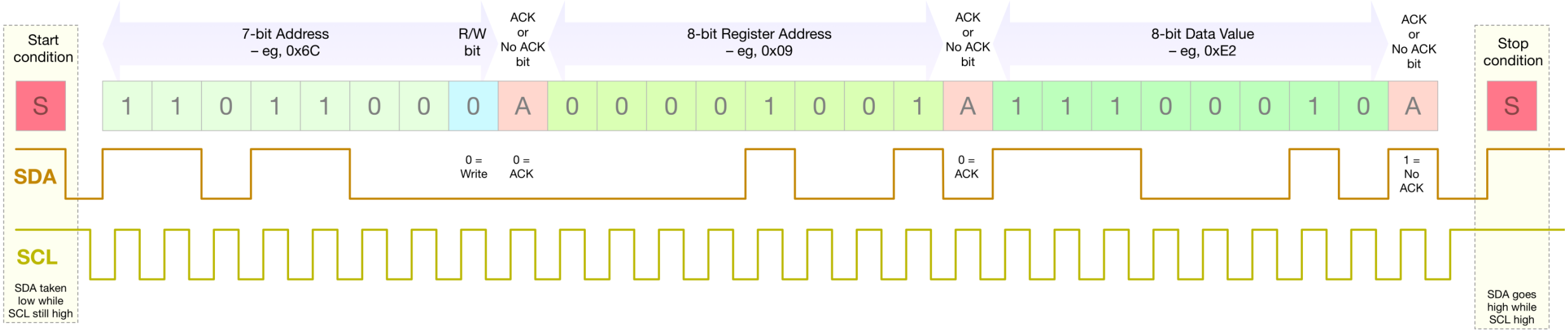




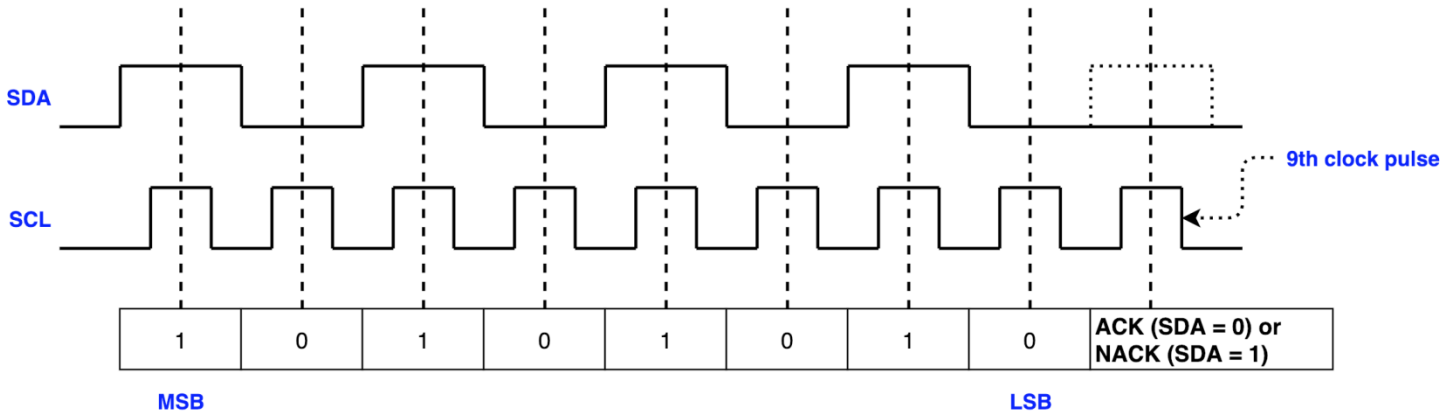


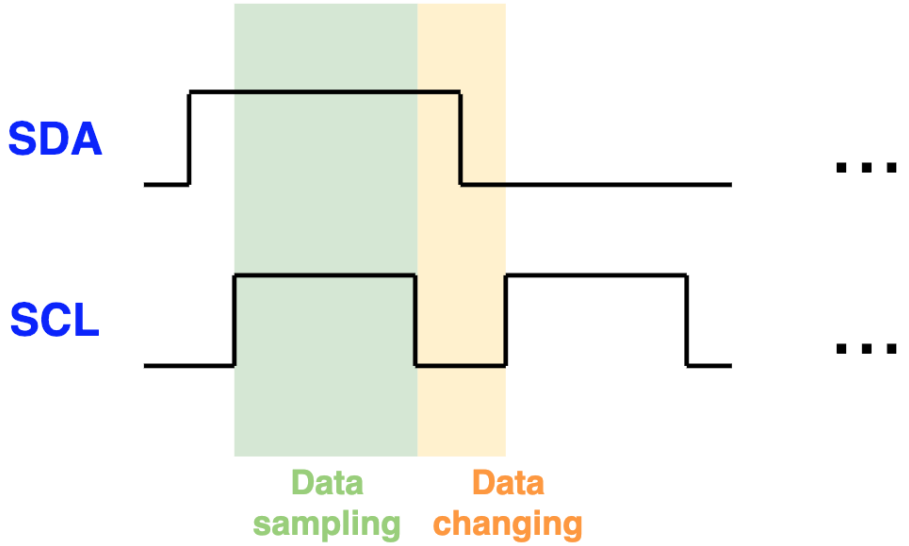
Передача і прийом даних по шині I2C (продовження)

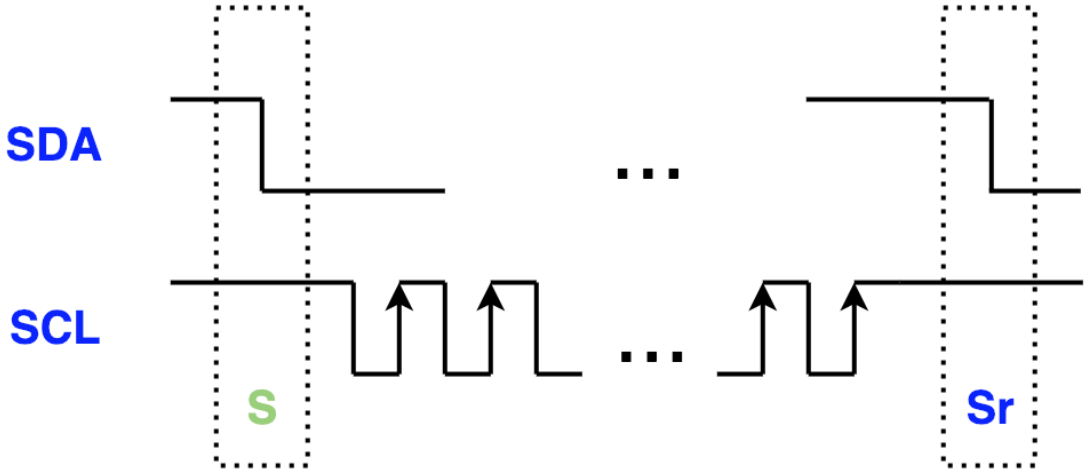




Формат посилки та умови початку і кінця транзакції

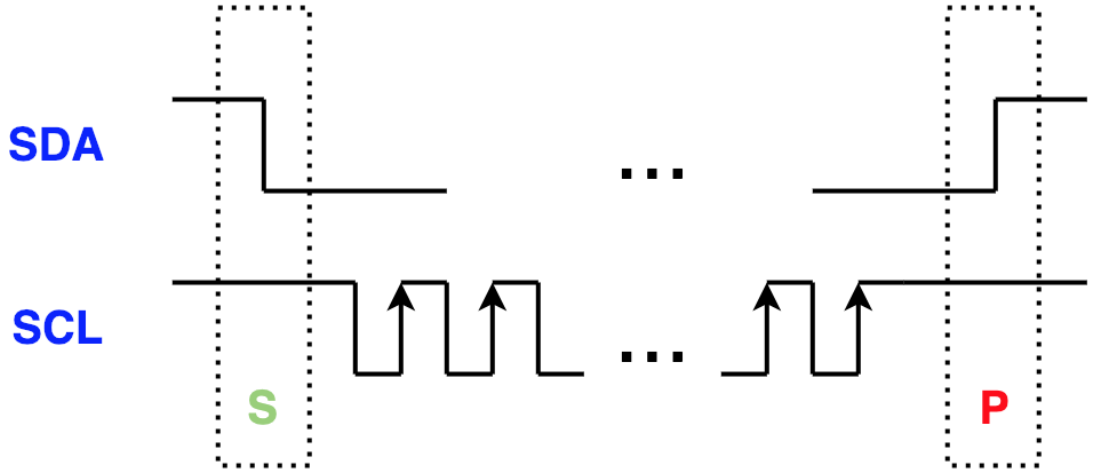






START
condition

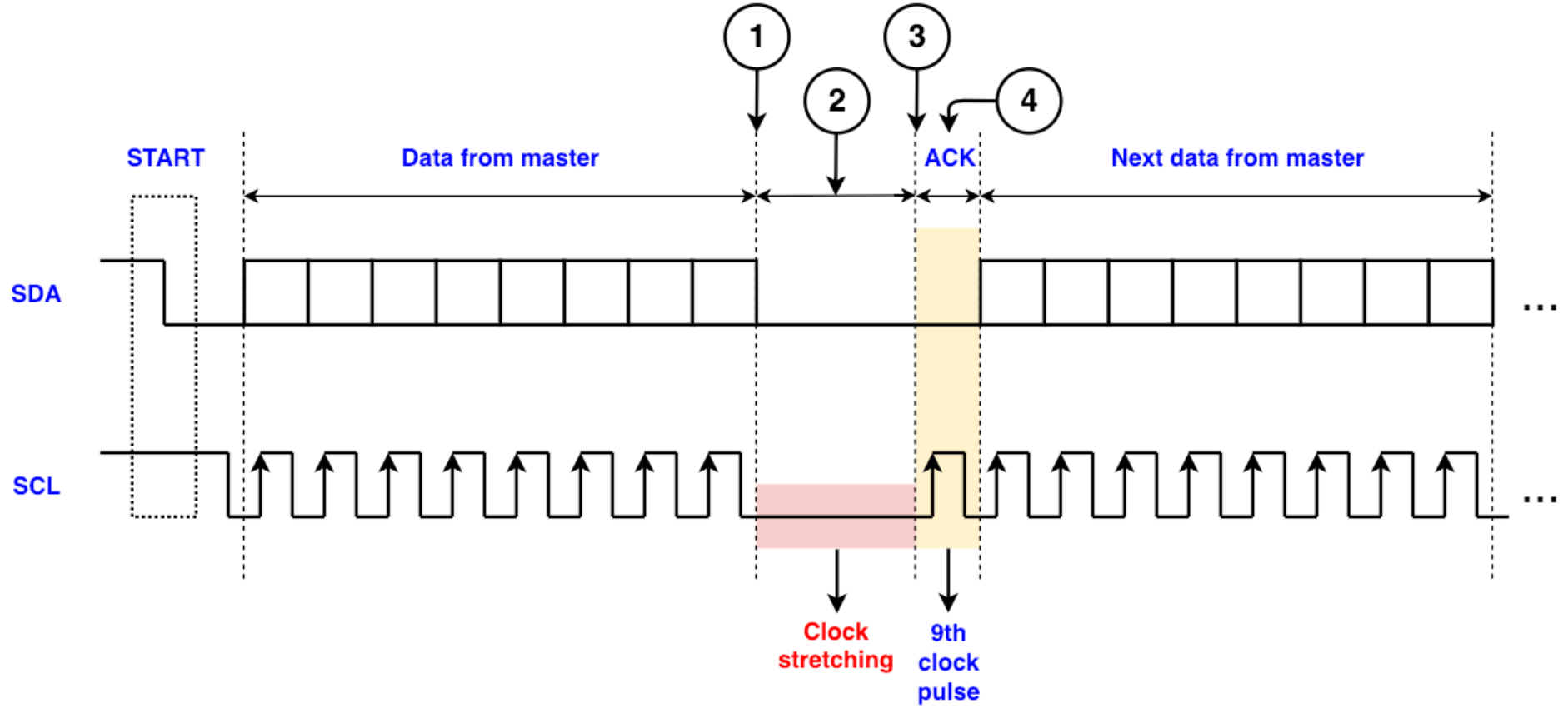
START
repeat



START
condition

STOP
condition

Розтягування сигналу синхронізації



1. До розтягування сигналу синхронізації: Після завершення надсилання 8 бітів ведучий “відпускає” (встановлює в 1) **SDA** та **SCL**. Ведучий повинен продовжувати читати стан лінії **SCL**, поки він не стане ВИСОКИМ.
2. Під час розтягування сигналу синхронізації: Ведений розтягує сигнал синхронізації та встановлює **SDA** в “0” (для ACK), але продовжує утримувати **SCL**.
3. Завершення розтягування сигналу синхронізації: Ведений відпускає **SCL** і він пасивно переходить у ВИСОКИЙ рівень.
4. Після розтягування сигналу синхронізації : Ведучий виявляє 9-й тактовий імпульс. Якщо на **SDA** НИЗКИЙ рівень, то це відповідає сигналу ***ACK***.

Програмна реалізація протоколу I2C

```
#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

#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);
}
```

Функції I2C_Write() та I2C_Read()

```
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); //while((I2C_PIN&(1<<SDA_PIN))==0){;}
        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;
}
```

PCF8574 (8-бітний I/O експандер)

Структурна схема мікросхеми PCF8574

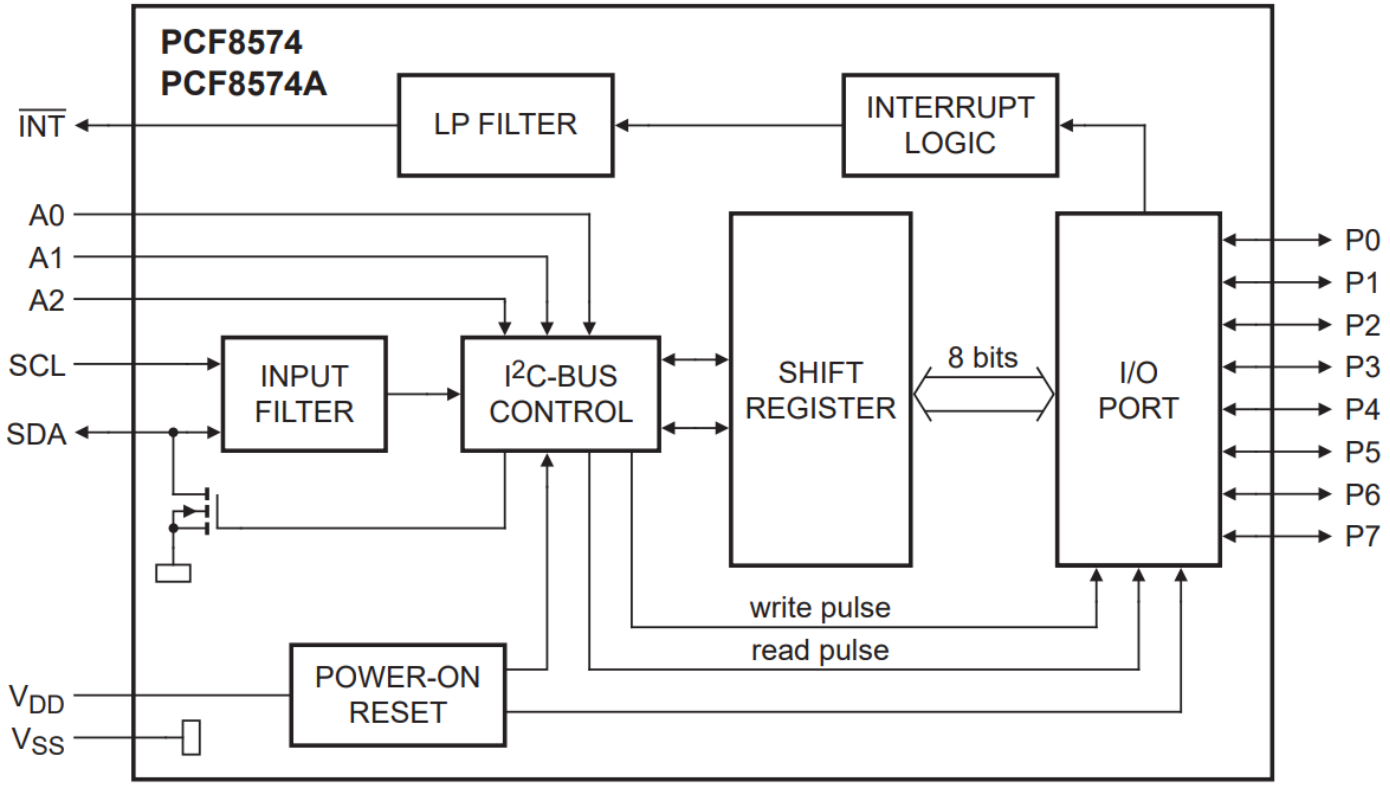
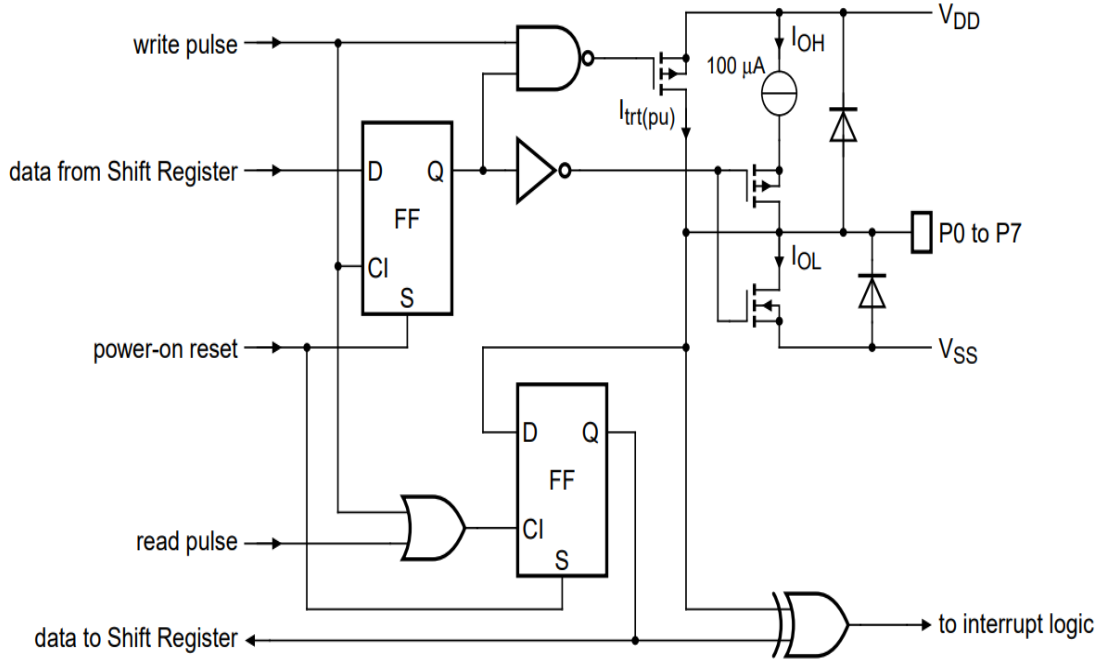
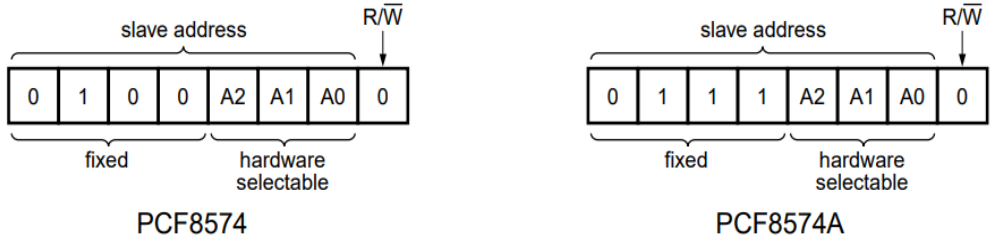


Схема вихідного драйвера порта вводу/виводу



Формат адресного байта



Транзакція запису даних

<S> <slave address + write> <ACK> <data out> <ACK> <P>

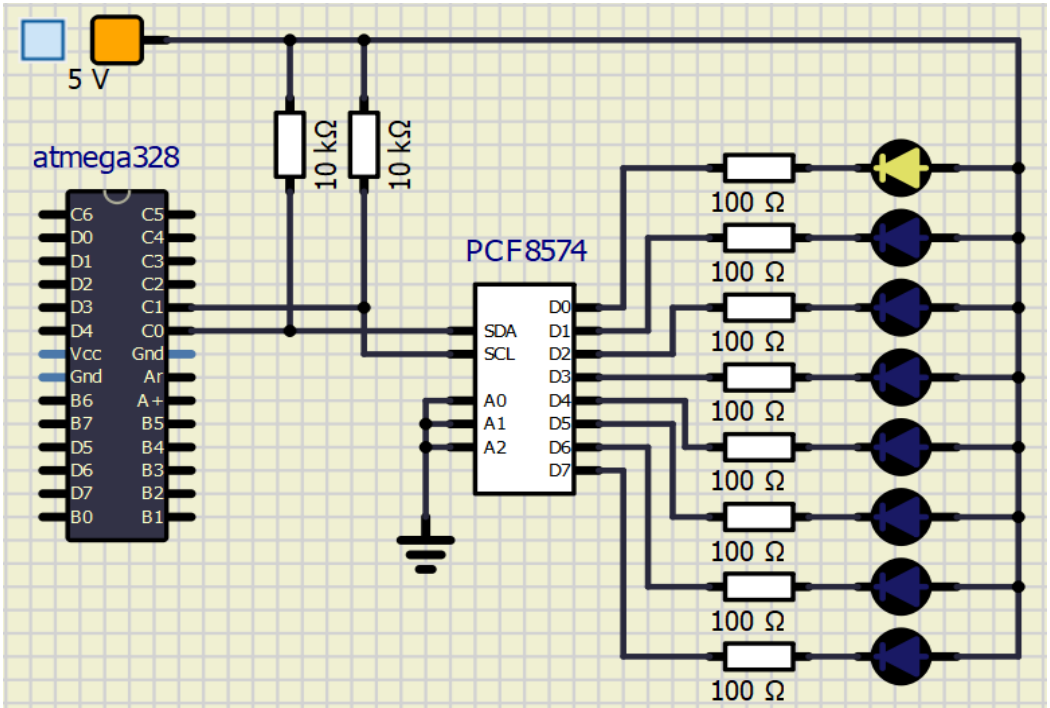
Транзакція читання даних

<S> <slave address + read> <**ACK**> <**data in1**><ACK> <**data in2**> <NACK> <P>

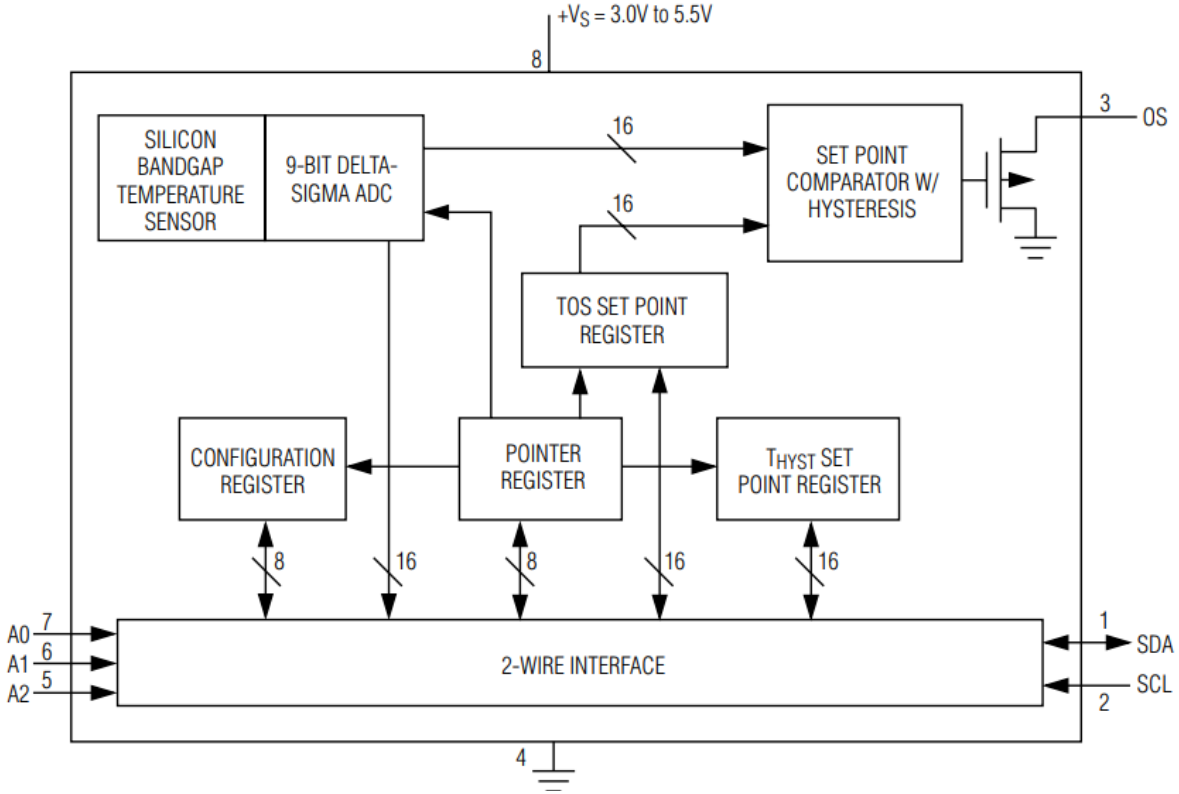
Тестова програма для PCF8574

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

// defines ...

void I2C_Init(void);  //...
void I2C_Start(void); //...
void I2C_Stop(void);  //...
uint8_t I2C_Write(uint8_t data); //...
uint8_t PCF8574_Write(uint8_t adr, uint8_t data)

//-------------------------------------------------
int main(void)
{
  uint8_t data = 1;
  I2C_Init();

    // mail loop =============
    for(;;)
    {
        PCF8574_Write(0x40, data);
        data = (data == 0)? 1: data << 1;
        _delay_ms(500);
    }

  return 0;
}
```

```c
// Функція запису даних в PCF8574
uint8_t PCF8574_Write(uint8_t adr, uint8_t data)
{
    I2C_Start();
    if(I2C_Write(adr) != I2C_ACK) {
      I2C_Stop();
      return 1;
    }
    I2C_Write(~data);
    I2C_Stop();
  return 0;
}
```



LM75 (Цифровий термометр)



Temperature, T_{HYST}, and T_{OS} Register Definition

UPPER BYTE								LOWER BYTE							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Sign bit 1= Negative 0 = Positive	MSB 64°C	32°C	16°C	8°C	4°C	2°C	1°C	LSB 0.5°C	X	X	X	X	X	X	X

Configuration Register Definition

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	Fault Queue	Fault Queue	OS Polarity	Comparator/ Interrupt	Shutdown

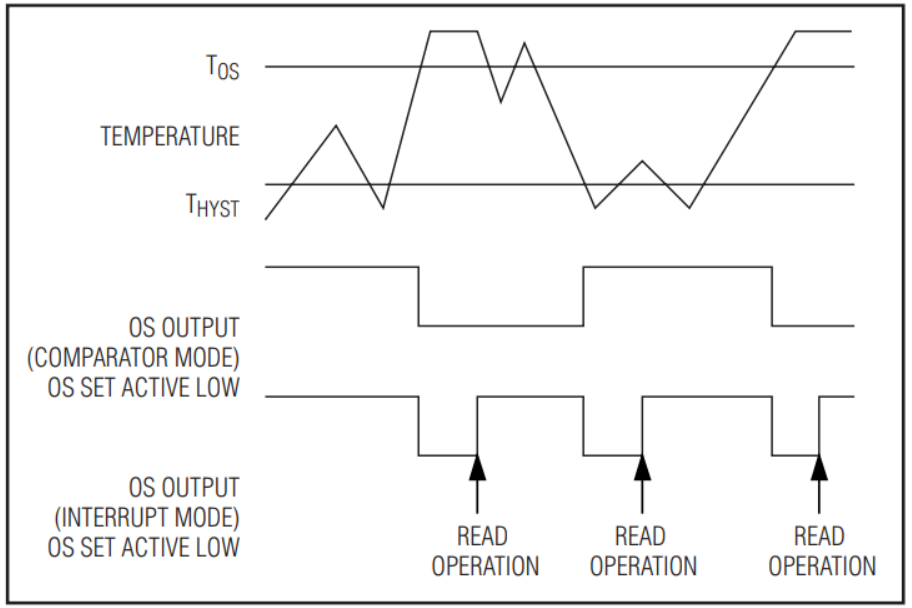
Slave Address

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
1	0	0	1	A2	A1	A0	R/$\overline{W}$

Register Functions

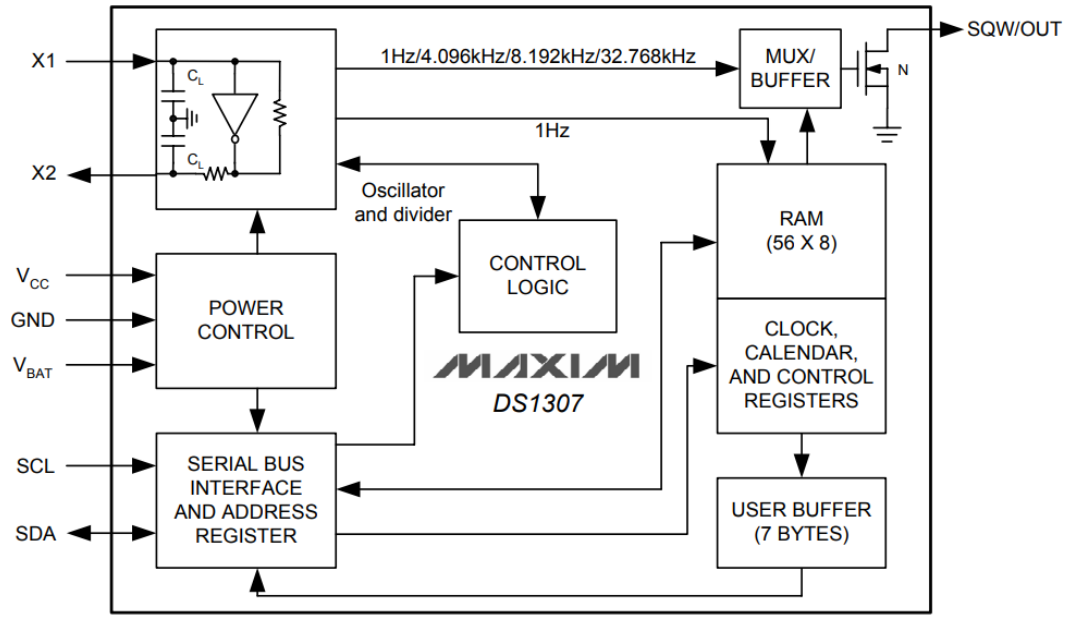
REGISTER NAME	ADDRESS (hex)	POR STATE (hex)	POR STATE (binary)	POR STATE (°C)	READ/ WRITE
Temperature	00	000X	0000 0000 0XXX XXXX	—	Read only
Configuration	01	00	0000 0000	—	R/W
T_{HYST}	02	4B0X	0100 1011 0XXX XXXX	75	R/W
T_{OS}	03	500X	0101 0000 0XXX XXXX	80	R/W

X = Don't care.



OS Output Temperature Response Diagram

Годинник реального часу DS1307

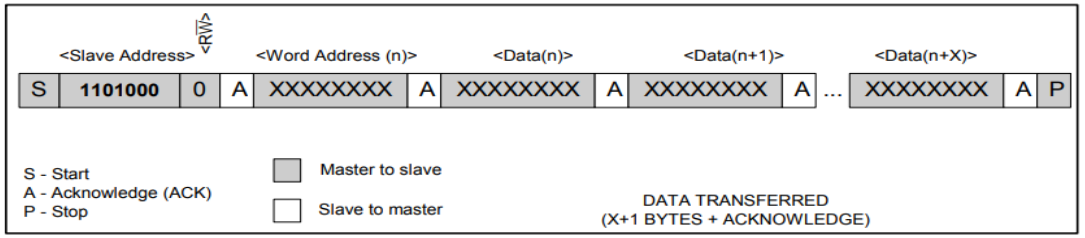


Опис регістрів для роботи з датою і часом

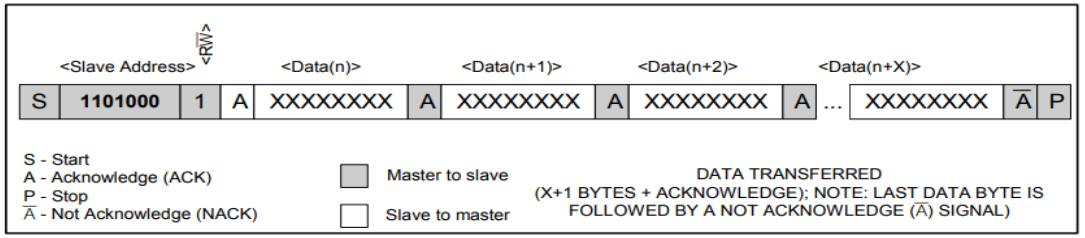
ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

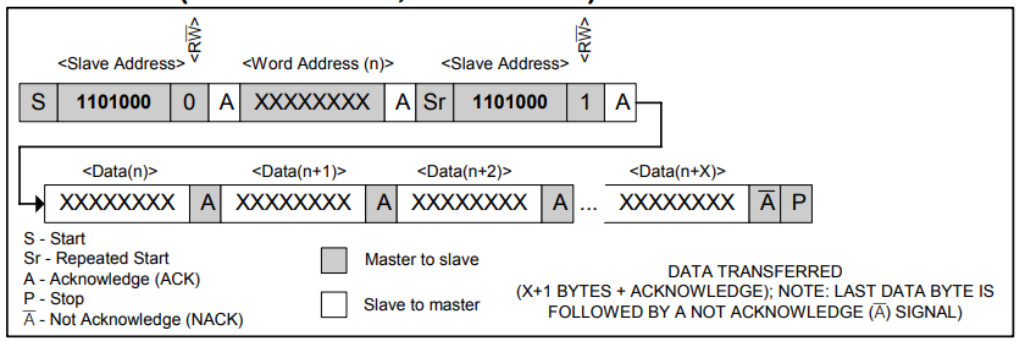
Data Write—Slave Receiver Mode



Data Read—Slave Transmitter Mode



Data Read (Write Pointer, Then Read)—Slave Receive and Transmit

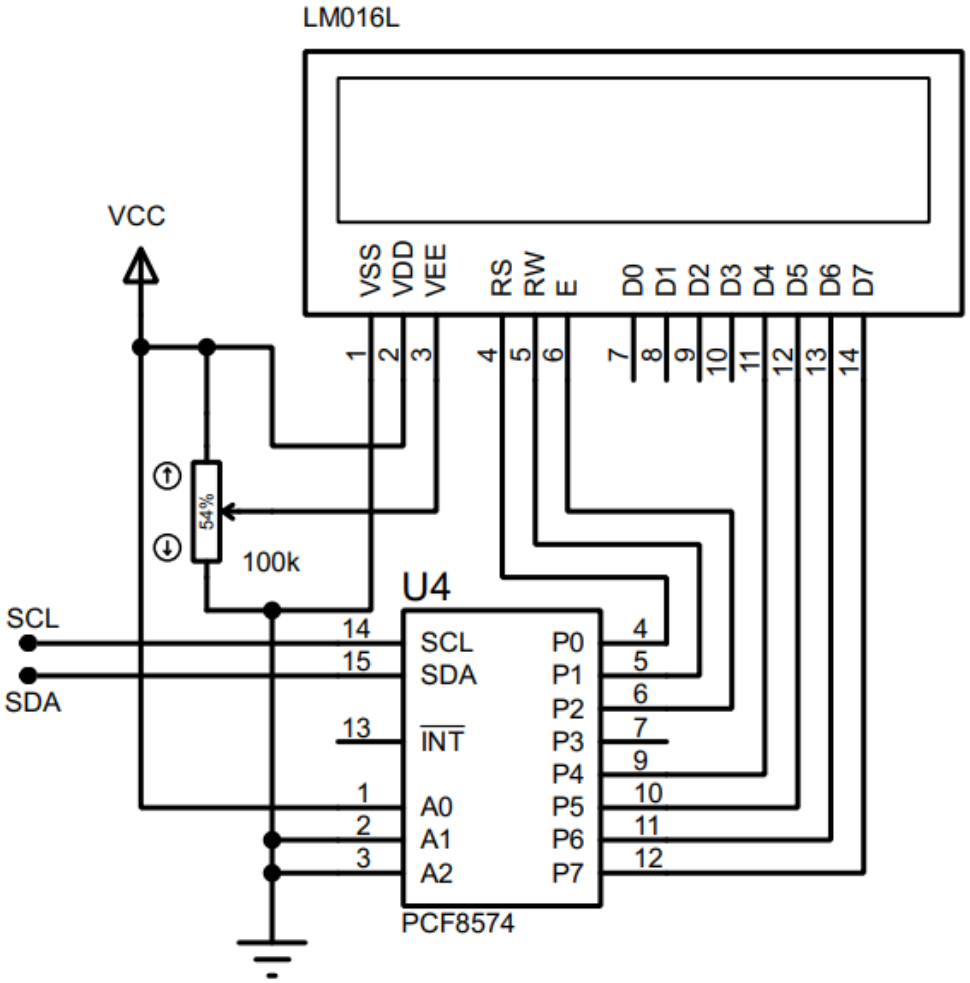


CONTROL REGISTER

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	0	0	SQWE	0	0	RS1	RS0

RS1	RS0	SQW/OUT OUTPUT	SQWE	OUT
0	0	1Hz	1	X
0	1	4.096kHz	1	X
1	0	8.192kHz	1	X
1	1	32.768kHz	1	X
X	X	0	0	0
X	X	1	0	1

Підключення символьного LCD (HD44780) дисплея до I2C шини




```
//                                  Initialize LCD (HD44780)
// +----------------------------+
// |         Power on           |
// +----------------------------+
//              |
// +----------------------------+
// |  RS R/W DB7 DB6 DB5 DB4    |
// |   0   0   0   0   1   1    |   // Initial sequence 0x30
// | Wait for more than 4.1 ms  |   // 4.1 ms us writing DATA into DDRAM or CGRAM
// +----------------------------+
//              |
// +----------------------------+
// |  RS R/W DB7 DB6 DB5 DB4    |
// |   0   0   0   0   1   1    |   // Initial sequence 0x30
// | Wait for more than 0.1 ms  |   // 100 us writing DATA into DDRAM or CGRAM
// +----------------------------+
//              |
// +----------------------------+
// |  RS R/W DB7 DB6 DB5 DB4    |   // Initial sequence 0x30
// |   0   0   0   0   1   1    |   // 37 us writing DATA into DDRAM or CGRAM 4 us add - time after busy flag disapeared
// | Wait for more than 45 us   |   // 37 us + 4 us = 41 us * (270/250) = 45us
// +----------------------------+
//              |
// +----------------------------+   // 4bit mode 0x20 !!! MUST BE SET TIME, BF CHECK DOESN'T WORK CORRECTLY !!!
// |  RS R/W DB7 DB6 DB5 DB4    |   //
// |   0   0   0   0   1   0    |   // 37 us writing DATA into DDRAM or CGRAM 4 us add - time after busy flag disapeared
// | Wait for more than 45 us   |   // !!! MUST BE SET DELAY TIME, BUSY FLAG CHECK DOESN'T WORK CORRECTLY !!!
// +----------------------------+
//              |
// +----------------------------+
// |  RS R/W DB7 DB6 DB5 DB4    |   // Display off 0x28
// |   0   0   0   0   1   0    |   //
// |   0   0   1   0   0   0    |   //
// |    Wait for BF Cleared     |   // Wait for 50us
// +----------------------------+
//              |
// +----------------------------+
// |  RS R/W DB7 DB6 DB5 DB4    |   // Display clear 0x01
// |   0   0   0   0   0   0    |   //
// |   0   0   0   0   0   1    |   //
// |    Wait for BF Cleared     |   // Wait for 50us
// +----------------------------+
//              |
// +----------------------------+
// |  RS R/W DB7 DB6 DB5 DB4    |   // Entry mode set 0x06
// |   0   0   0   0   0   0    |   //
// |   0   0   0   1   1   0    |   // shift cursor to the left, without text shifting
// |    Wait for BF Cleared     |   // Wait for 50us
// +----------------------------+
```

Функції для роботи з LCD

```
#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

void LCD_E_pulse(uint8_t data)
{
  I2C_Write(data | LCD_PIN_E);
  _delay_us(0.5);  // PWeh delay time > 450ns
  I2C_Write(data & ~LCD_PIN_E);
  _delay_us(0.5);  // PWeh delay time > 450ns
}
```

```
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(50);
}

void LCD_SetXY (uint8_t addr, uint8_t x, uint8_t y)
{
  if (y == 0) {
    // send instruction 1st row (0)
    LCD_Send(addr, LCD_POSITION | (x), LCD_CMD);
  } else if (y == 1) {
    // send instruction 2nd row (0x40)
    LCD_Send(addr, LCD_POSITION | (0x40 + x), LCD_CMD);
  }
}
```

продовження...

```
uint8_t LCD_Init(uint8_t addr)
{
  uint8_t res, data = 0;

  _delay_ms(20);
  I2C_Start();
  res = I2C_Write(addr);
  if(res != I2C_ACK) {
    I2C_Stop();
    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();
```

```
 //продовження...

  // 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_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);
}
```

Правила написання коду на мові С

У 1998 році організація **MISRA** (***M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation*) випустила перший документ регламентує написання коду для автомобільної промисловості. Згодом стандарт був оновлений кілька разів (2004, 2012) і кількість правил значно збільшилася; ***MISRA* C** 2012 містить 143 правила, згруповані по секціях - обов'язкові, необхідні і рекомендовані. Однак, є більш короткий список, який насправді є доповненням до ***MISRA***. Його склав в 2006 році співробітник НАСА, Геральд Хольцман (Gerard J. Holzmann). *The Power of 10: Rules for Developing Safety-Critical Code*. Нижче наведено вільний переклад.

1. Не ускладнюйте програму використанням оператора **goto**, функціями **setjump () / longjump()** і рекурсією.
2. У всіх циклах повинна бути фіксована верхня межа.
3. Уникайте динамічне виділення пам'яті.
4. Код функції повинен вміщатися на одній друкованій сторінці.
5. На кожну функцію має припадати мінімум дві ***assert***-перевірки (вхід, вихід).
6. Всі об'єкти (змінні) повинні бути оголошені з мінімально можливим рівнем видимості.
7. Перевіряйте значення яке повертає функція в місці її виклику, а передані параметри всередині функції.
8. Використовуйте препроцесор тільки для простих макро-визначень.
9. Уникайте вказівників на функції і обмежуйтеся одним знаком розіменування вказівника.
10. Компілюйте з усіма ключами попереджень; виправте всі попередження до релізу програми.