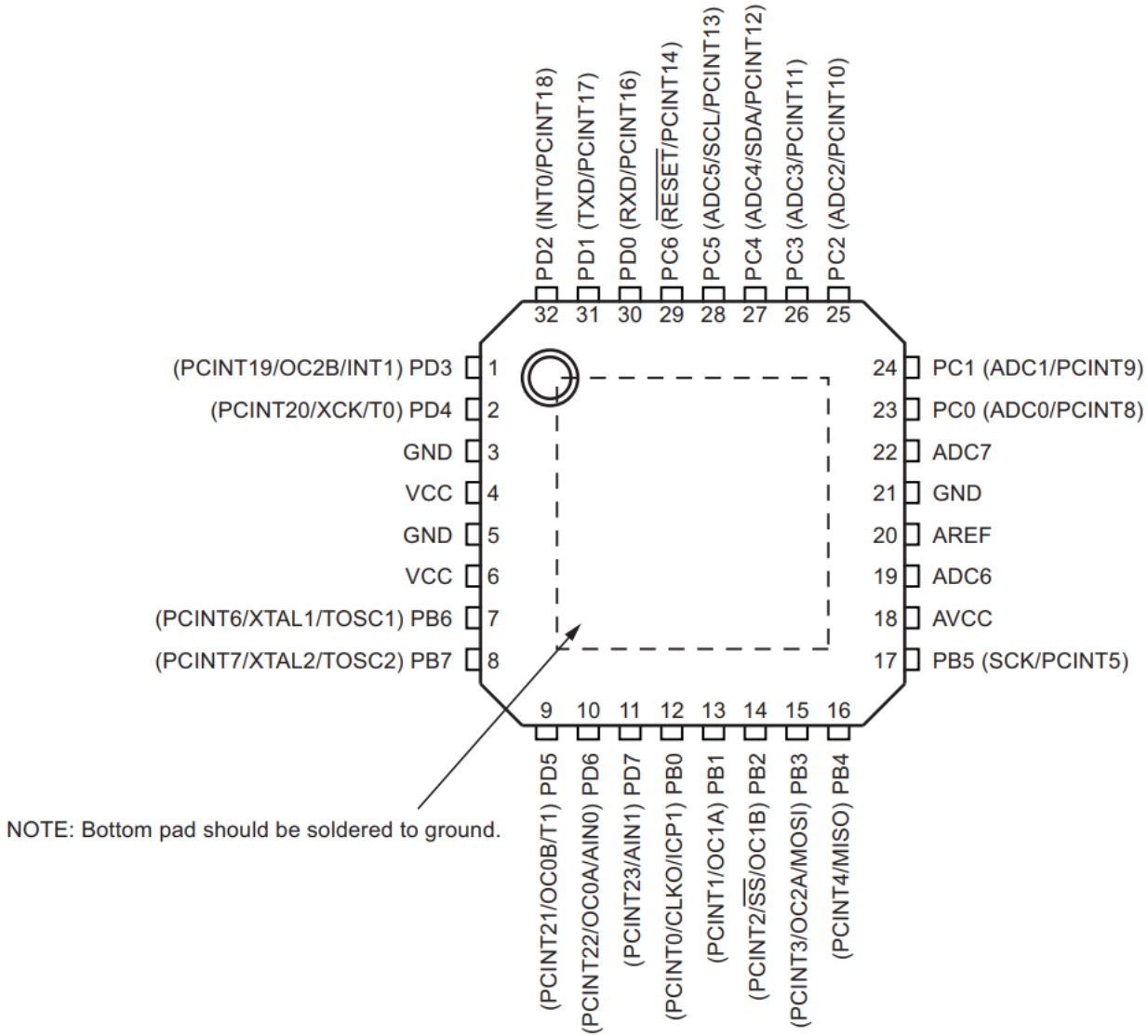
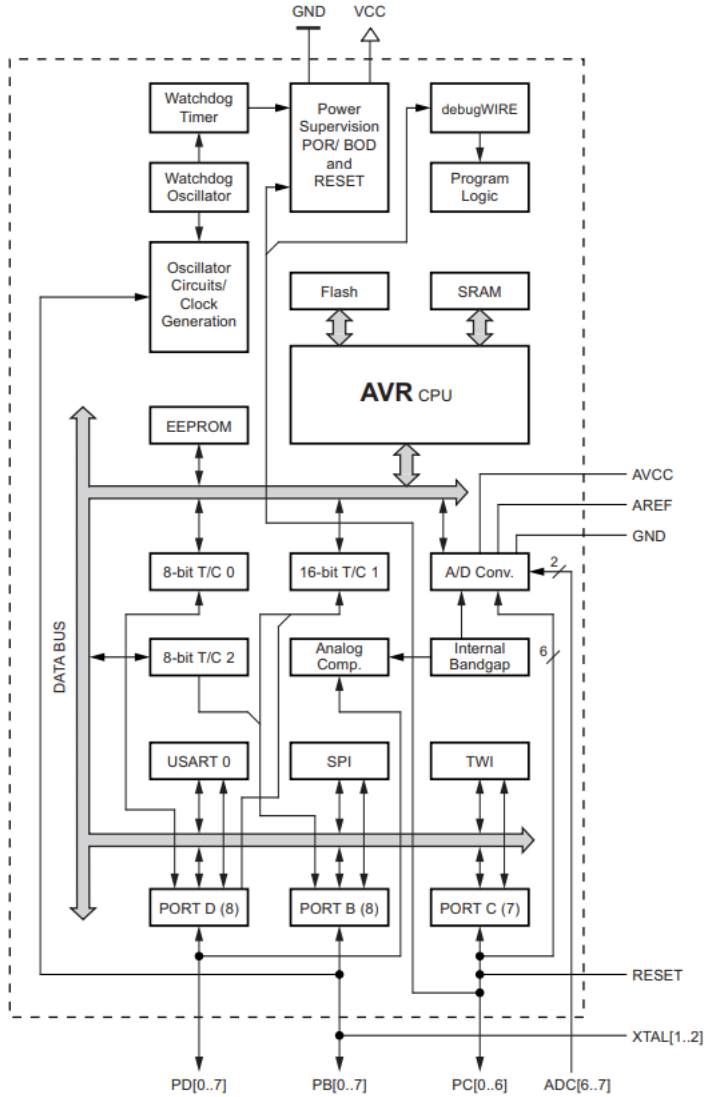
Вбудовані системи

Робота з портами вводу/виводу

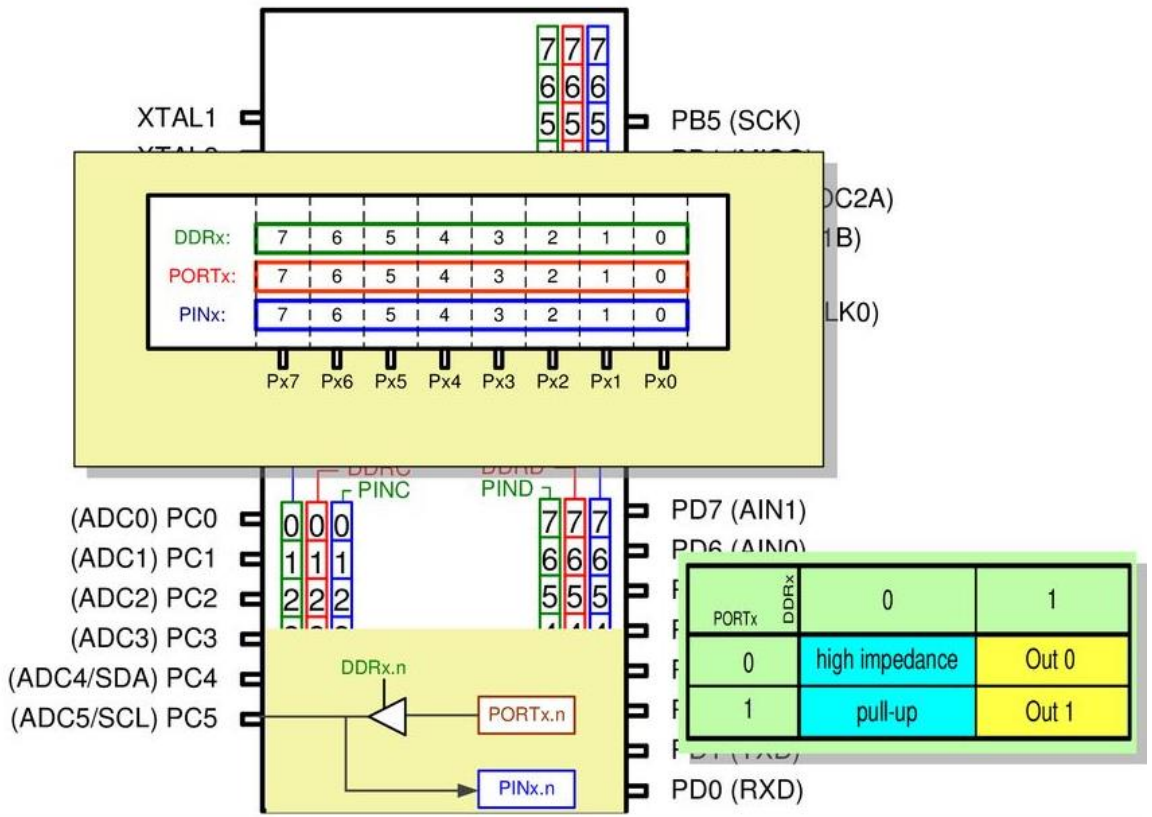
Альтернативні функції портів вводу-виводу МК ATmega328





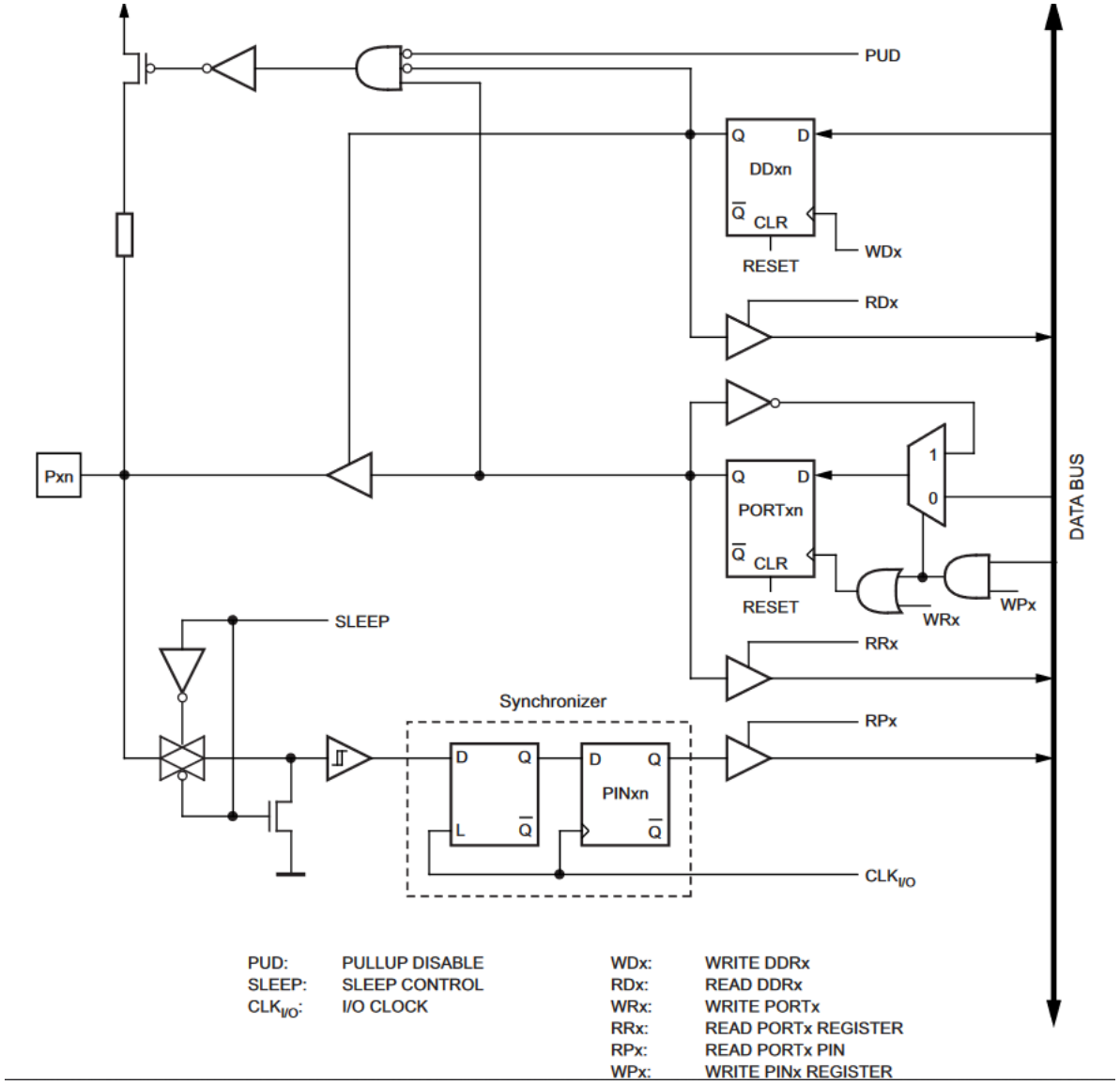
Порти вводу виводу у мікроконтролерах сімейства AVR

Спрощена схема I/O портів



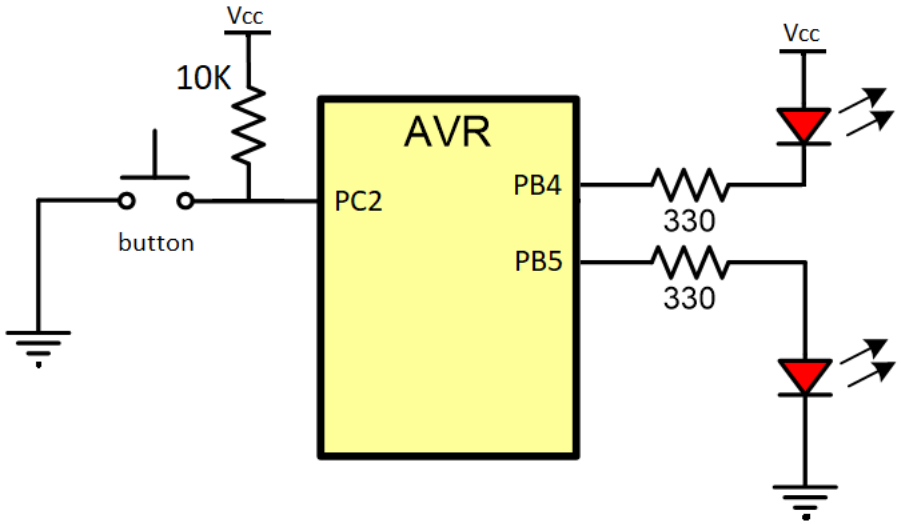
PORTx – регістр запису даних у порт
PINx – регістр для читання даних з порта
DDRx – регістр конфігурації напрямку порта

Логічна схема одного «піна» порта мікроконтролера ATmega328

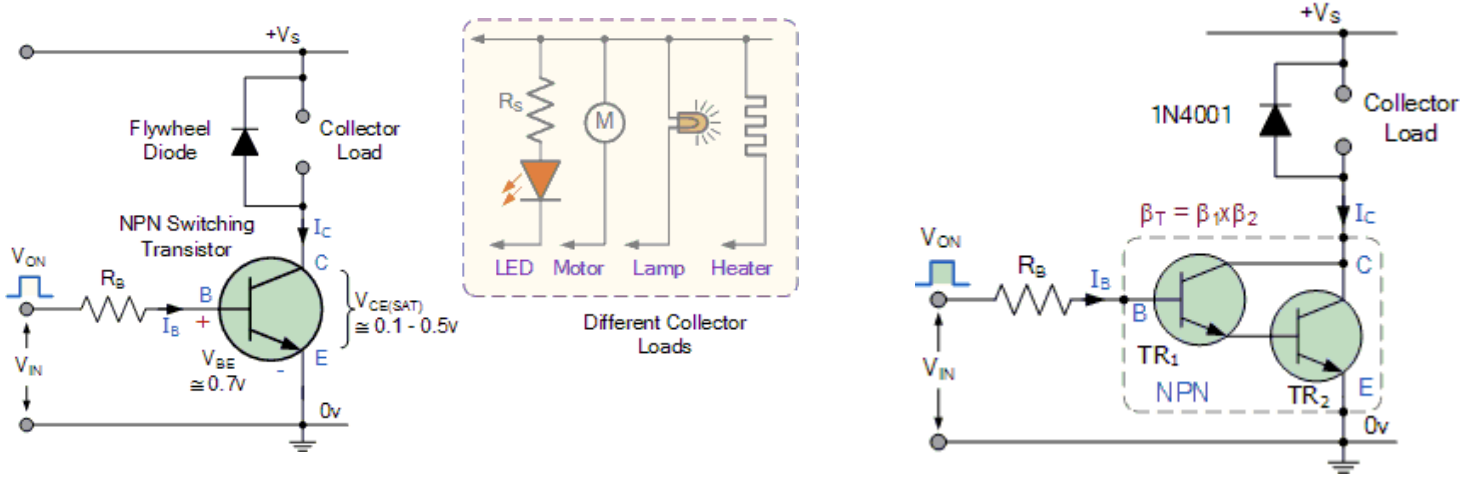


Підключення зовнішніх пристроїв до мікроконтролера

Схема підключення пристроїв вводу-виводу до мікроконтролера



Схемb підключення нагрузки з великим струмом споживання



Розрахунок обмежувального резистора

Rs = (3.3V – 1.6V)/0.005A = 340 Ohm

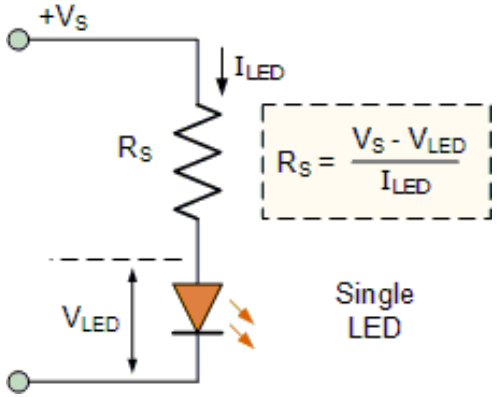
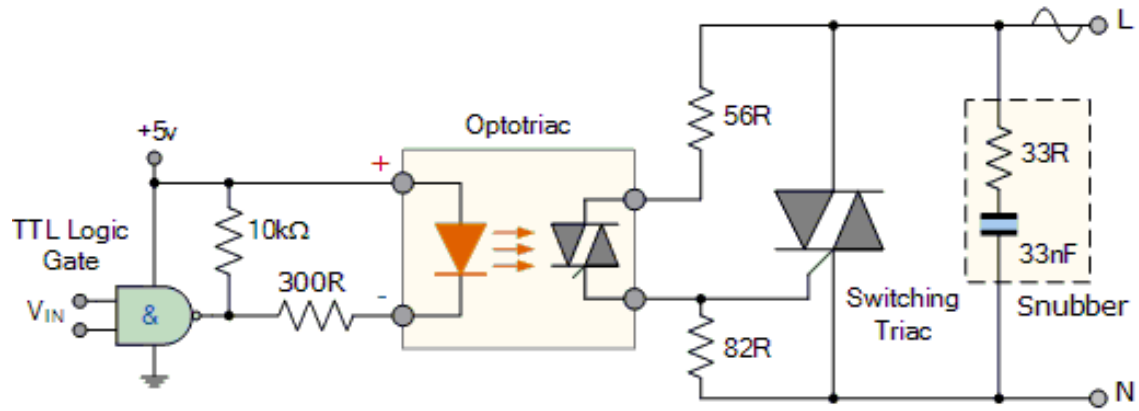


Схема з використанням гальванічної розв’язки та семистора

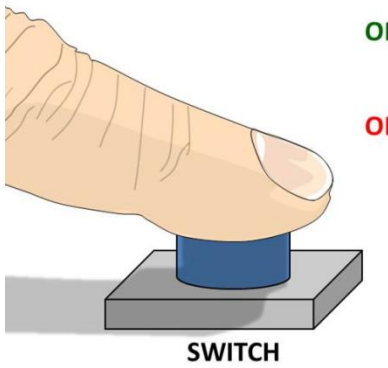


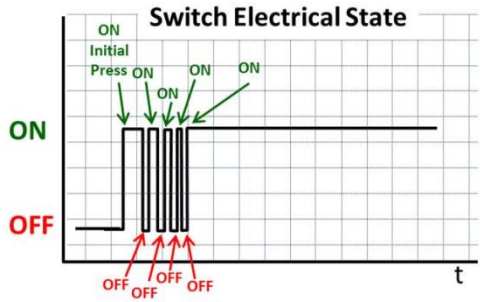
Робота з портами вводу-виводу

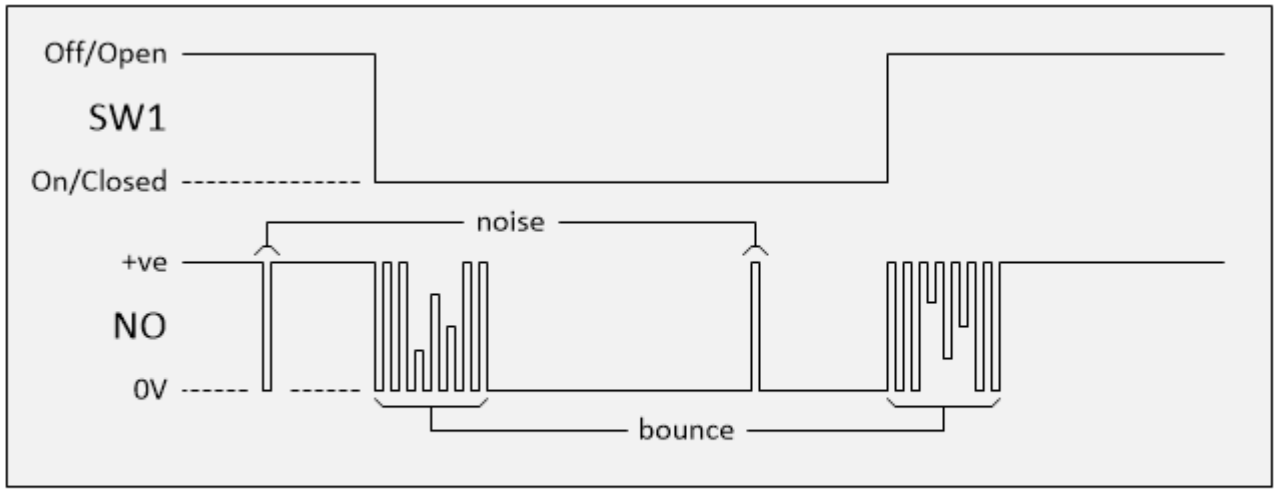
```
//код демонстрації роботи з портами

// налаштування порта B
DDRB |= (1<<PB4)|(1<<PB5);

// запис даних в порт B
PORTB |= (1<<PB4)|(1<<PB5);
_delay_ms(1000);
PORTB &= ~((1<<PB4)|(1<<PB5));

// налаштування порта C
DDRC &= ~(1<<PC2);
//читання даних з порта C
uint8_t  in = PINC & (1<<PC2);
```

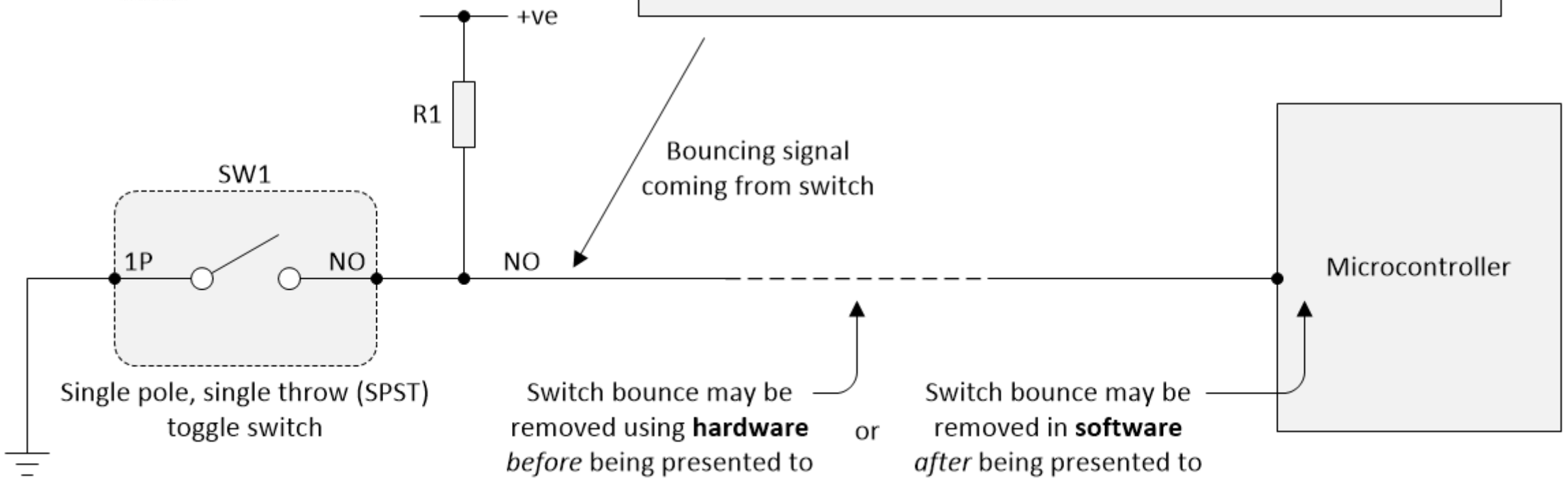
```
; налаштування порта
in r16, DDRD
ori r16, (1<<PB4)|(1<<PB5);
out DDRD, r16
; запис даних в порт B
in r16, PORTD
ori r16, (1<<PB4)|(1<<PB5);
out PORTD, r16
call _delay_ms
in r16, PORTD
andi r16, ~((1<<PB4)|(1<<PB5));
out PORTD, r16

; налаштування порта
in r16, DDRC
andi r16, ~(1<<PC2);
out DDRC, r16
; читання даних з порта C
in r17, PINC
andi r17, ~(1<<PC2)
```

Підключення пристроїв вводу до МК

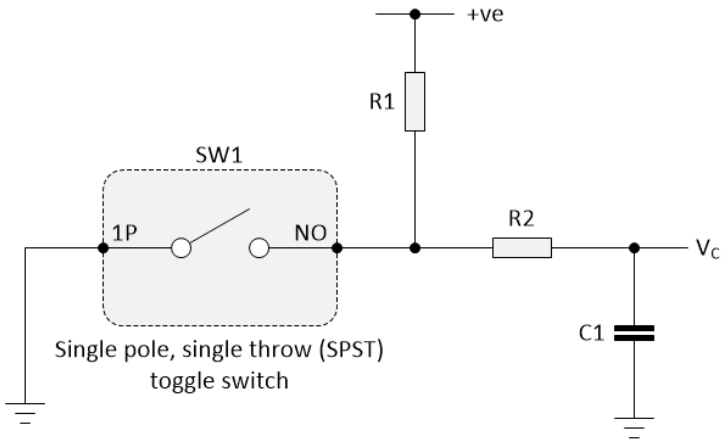


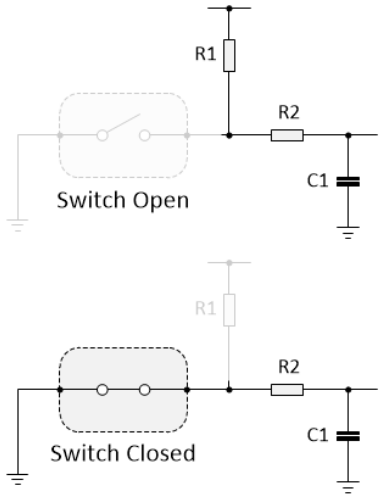


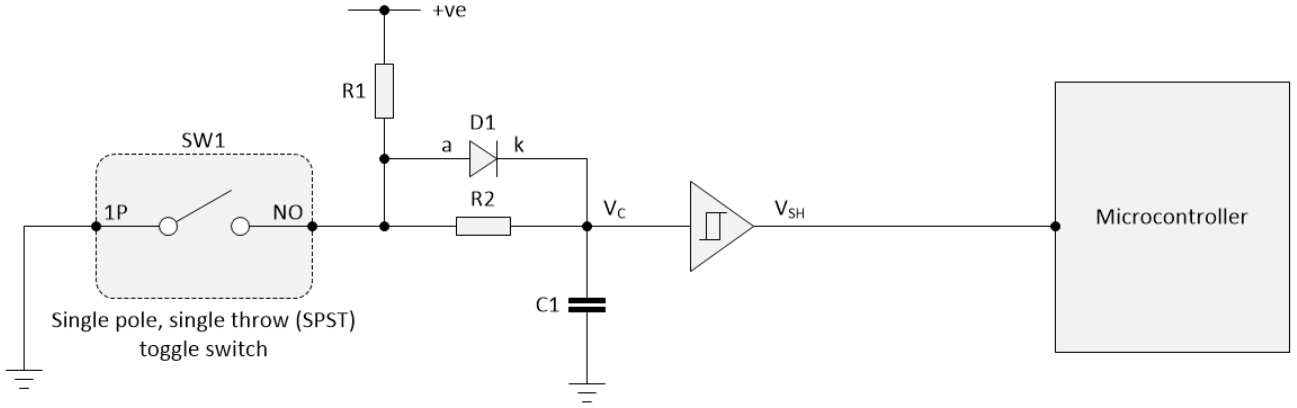


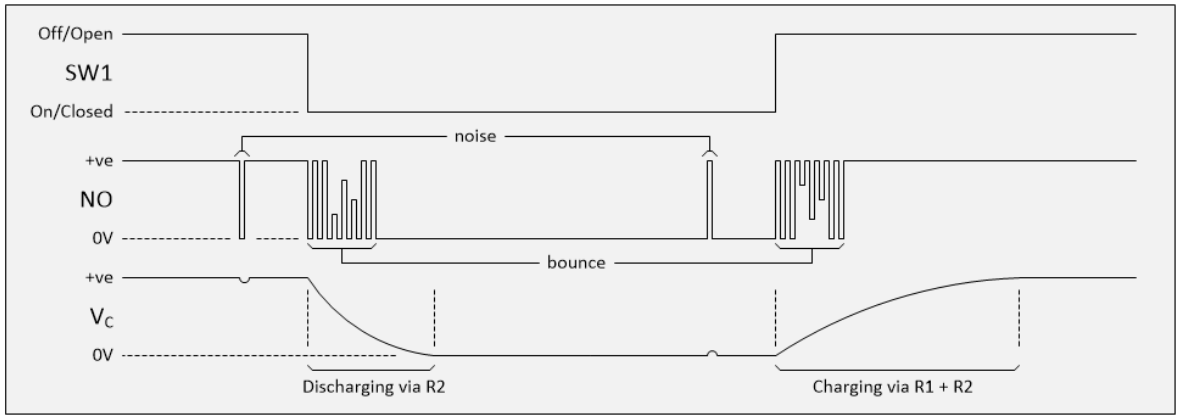


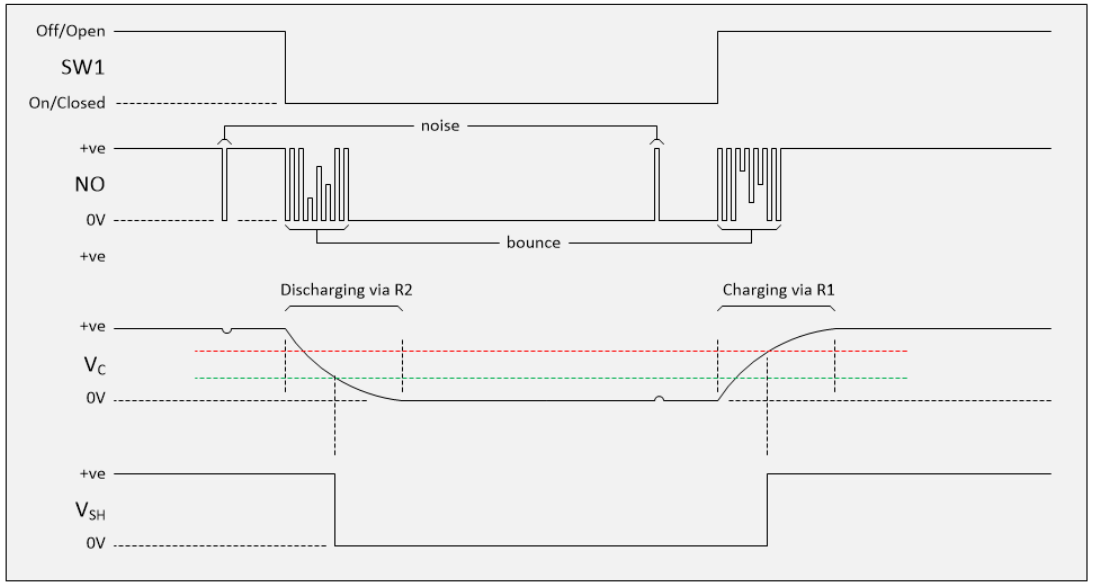
Схеми подавлення «дребезгу» контактів





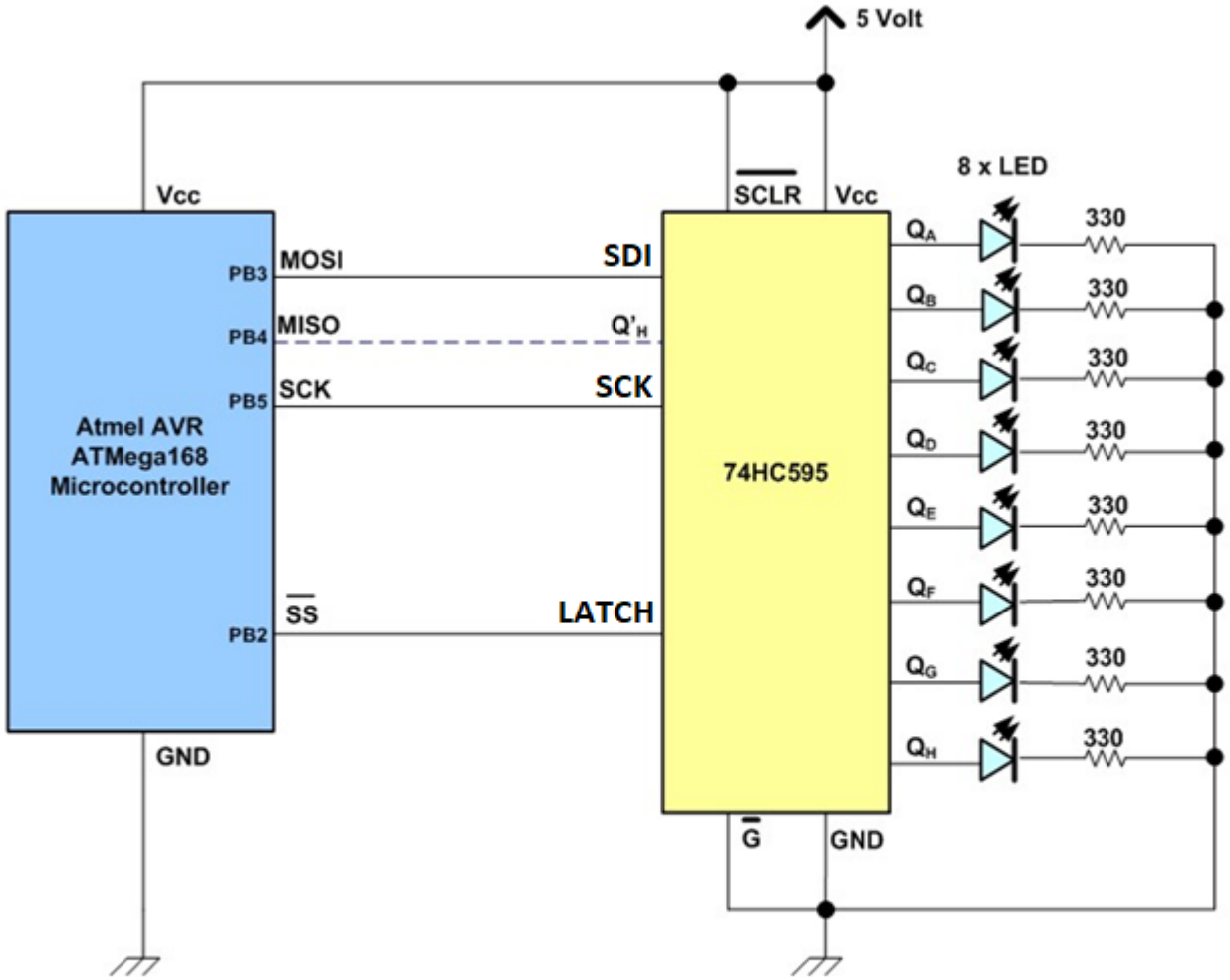




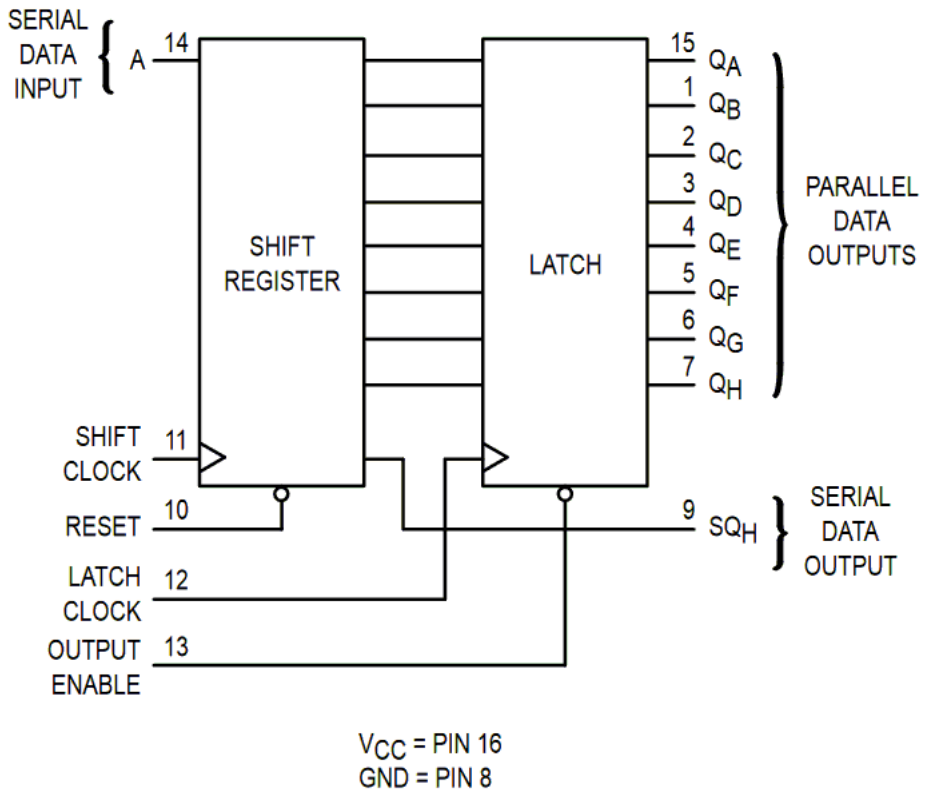


Розширення портів вводу-виводу

Схема підключення регістра зсуву до МК

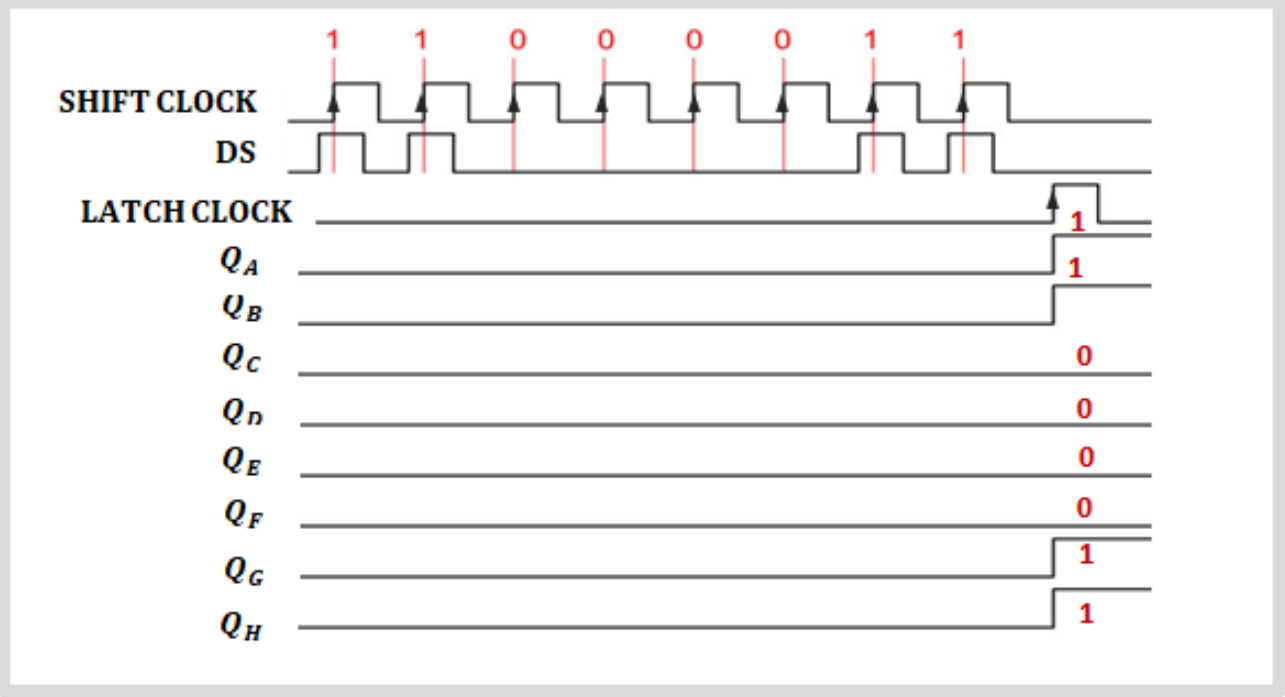


Логічна діаграма 74HC595

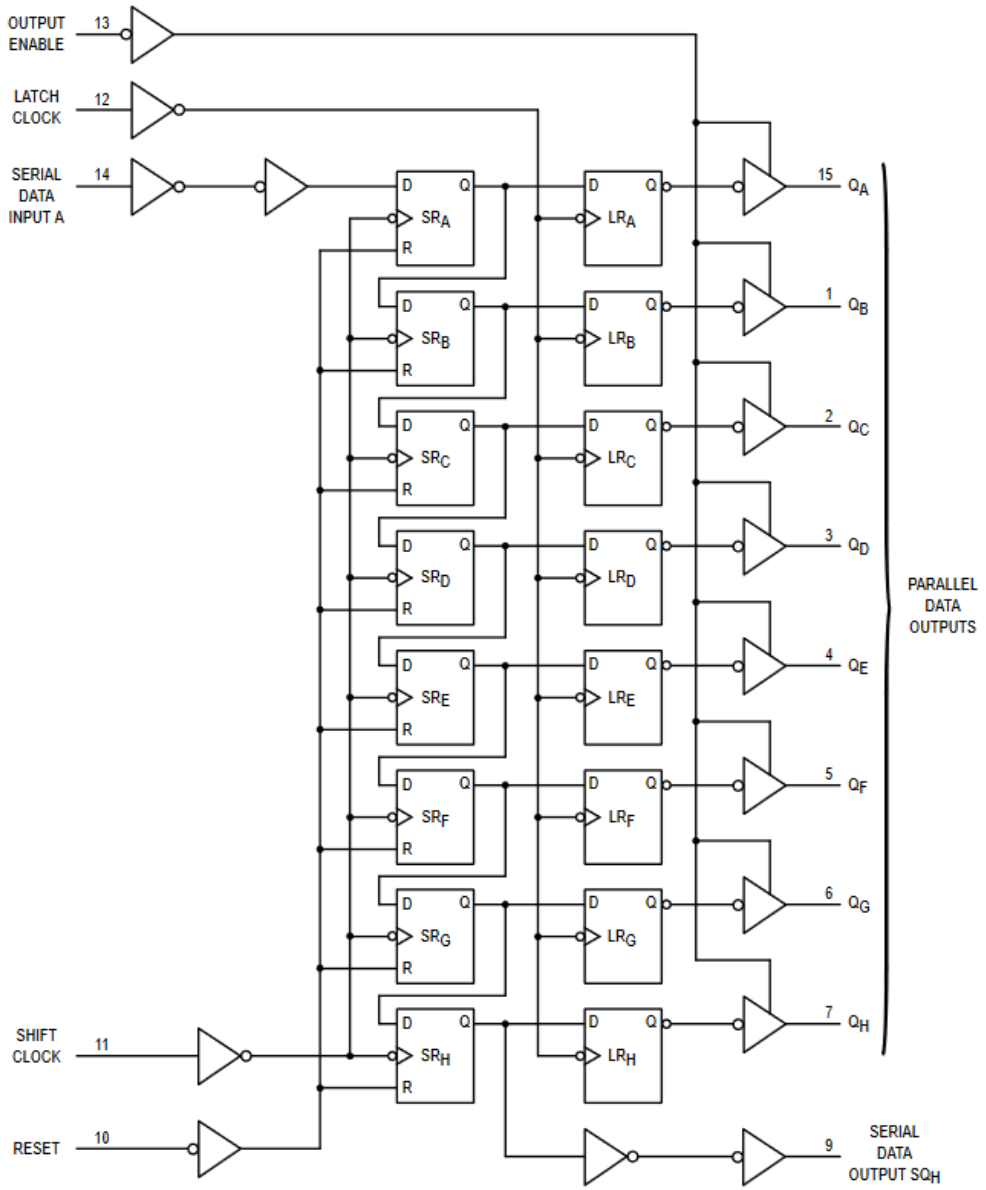


Робота з регістром зсуву 74HC595

Часові діаграми роботи регістра зсуву 74HC595



Логічна схема 74HC595



Запис даних в регістр зсуву

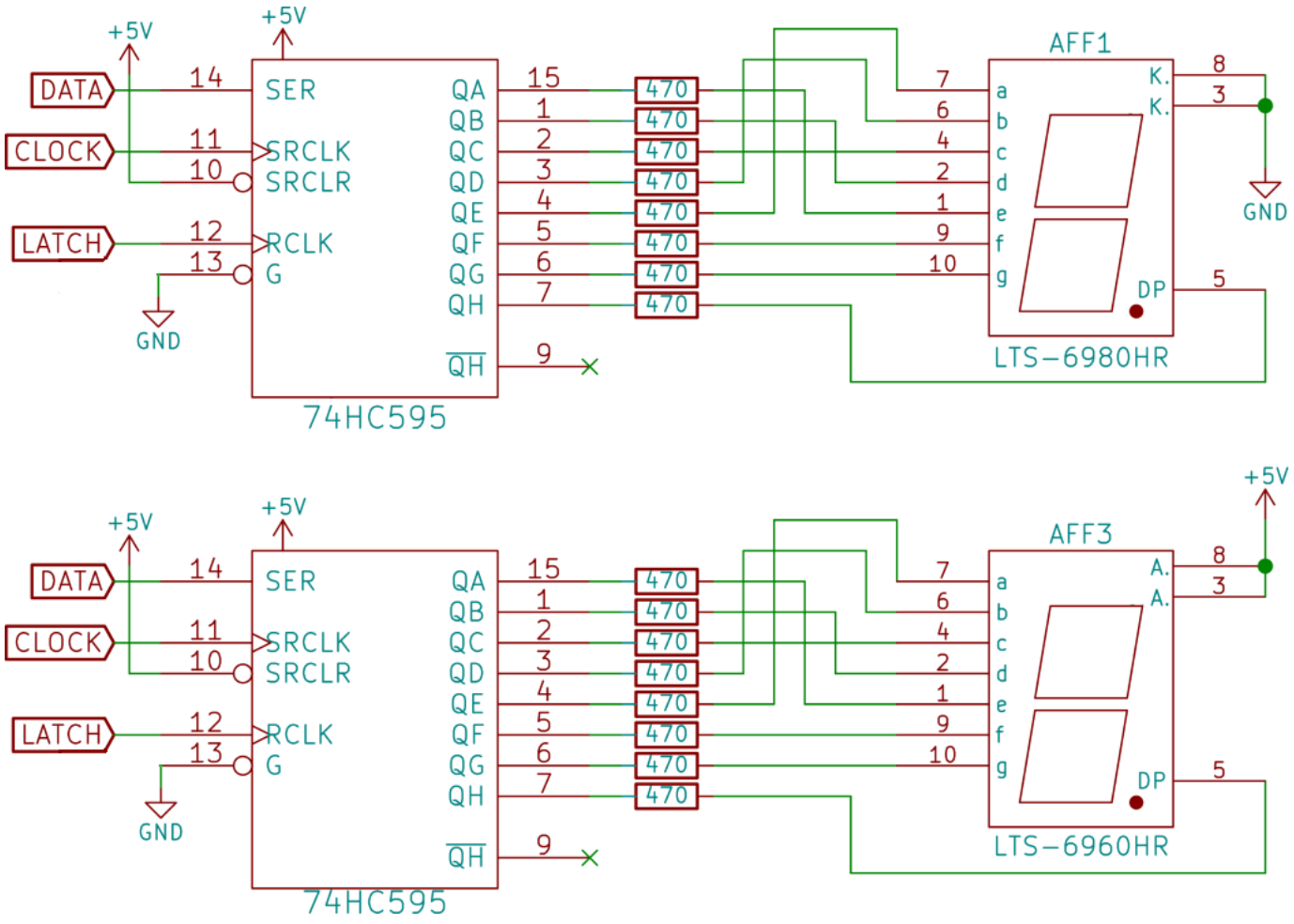
Функція запису даних в регістр зсуву

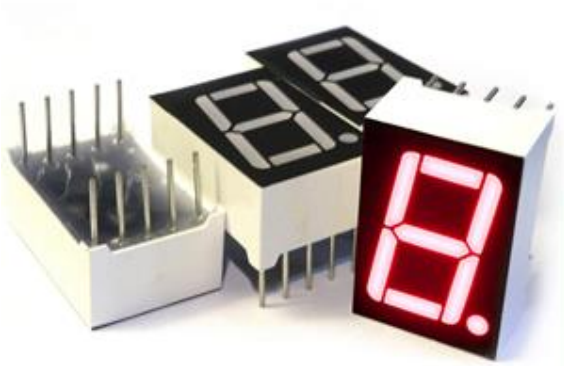
```
#define SCK PB5
#define SDI PB3
#define LATCH PB2

void ShiftRegWrite(uint8_t data)
{
  uint8_t i;

    for(i = 0; i < 8; i++)
    {
        if(data & 0x80) {
            PORTB |= 1 << SDI;
        } else {
           PORTB &= ~(1 << SDI);
        }
        data <<= 1;
        PORTB |= 1 << SCK;
        _delay_us(1);
        PORTB &= ~(1 << SCK);
        _delay_us(1);
    }
    PORTB |= 1 << LATCH;
    _delay_us(1);
    PORTB &= ~(1 << LATCH );
}
```

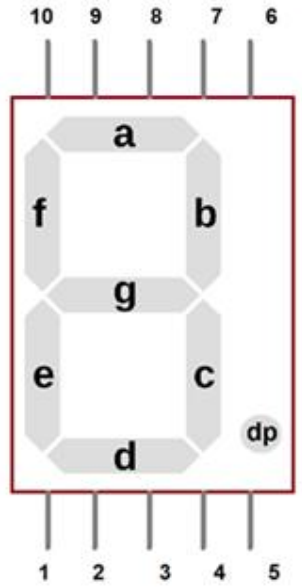
Демонстраційна програма

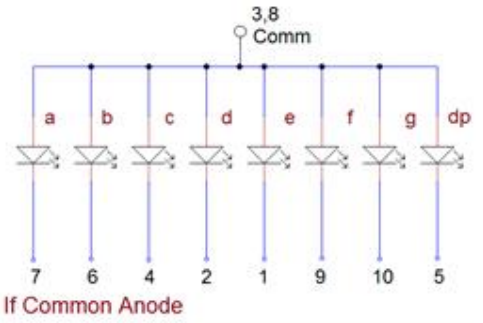
```
#include <avr/io.h>
#include <util/delay.h>

void ShiftRegWrite(uint8_t data)
{
    //…
}

int main(void)
{
  uint8_t d = 1;

    PORTB |= (1<<SCK)|(1<<SDI)|(1<<LATCH);

    for(;;)
    {
        ShiftRegWrite(d);
        d <<= 1;
        if(d == 0) d = 1;
        _delay_ms(250);
    }
  return 0;
}
```

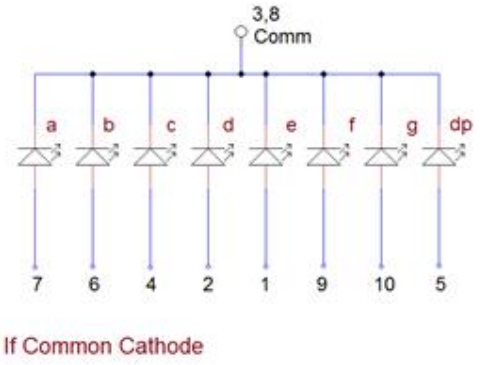
Підключення 7-ми сегментного індикатора



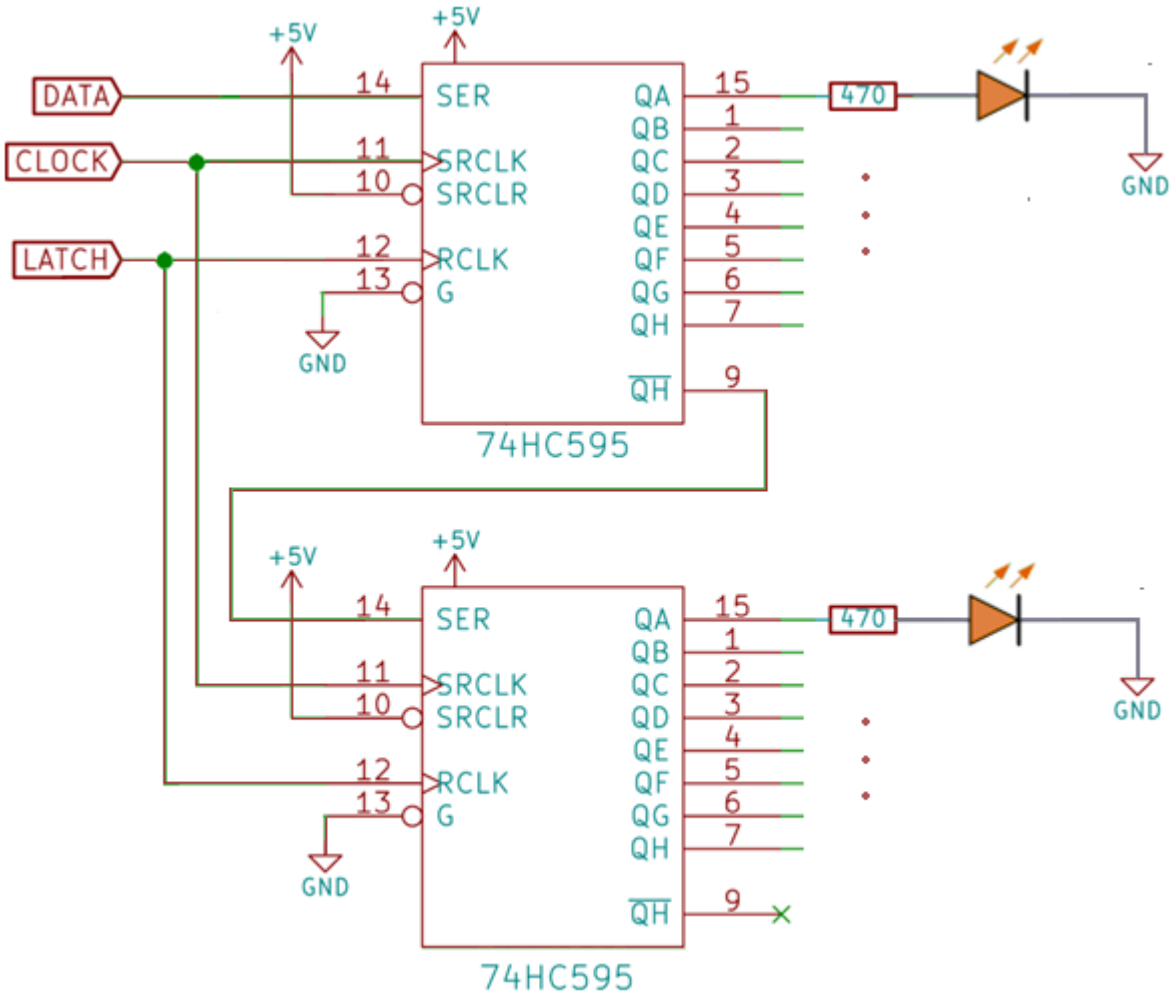








Послідовне з’єднання регістрів зсуву

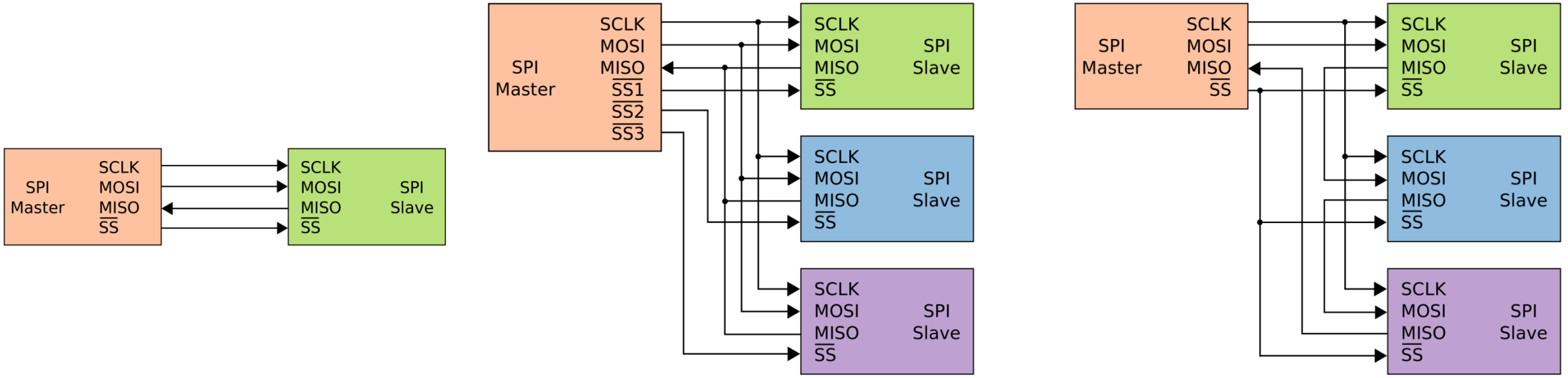


SPI - інтерфейс

SPI (**S**erial **P**eripheral **I**nterface, SPI bus — послідовний периферійний інтерфейс, шина SPI) - Послідовний синхронний повнодуплексний стандарт передачі даних, розроблений фірмою ***Motorola*** для забезпечення простого сполучення мікроконтролерів та периферії. **SPI** також називають чотирьох-провідним інтерфейсом.

Для передачі даних по **SPI** використовуються чотири сигнали:
MOSI або SI — (*Master Out Slave In*). Служить для передачі даних від ведучого пристрою до веденого.
MISO або SO —(*Master In Slave Out*). Служить для передачі даних від веденого пристрою до ведучого.
SCLK або SCK — послідовний тактовий сигнал (*Serial Clock*). Служить для синхронізації ведених пристроїв.
CS або SS — (*Chip Select, Slave Select*) сигнал початку/завершення сеансу зв'язку (вибір веденого пристрою для передачі/читання даних).

Способи під'єднання пристроїв до шини SPI



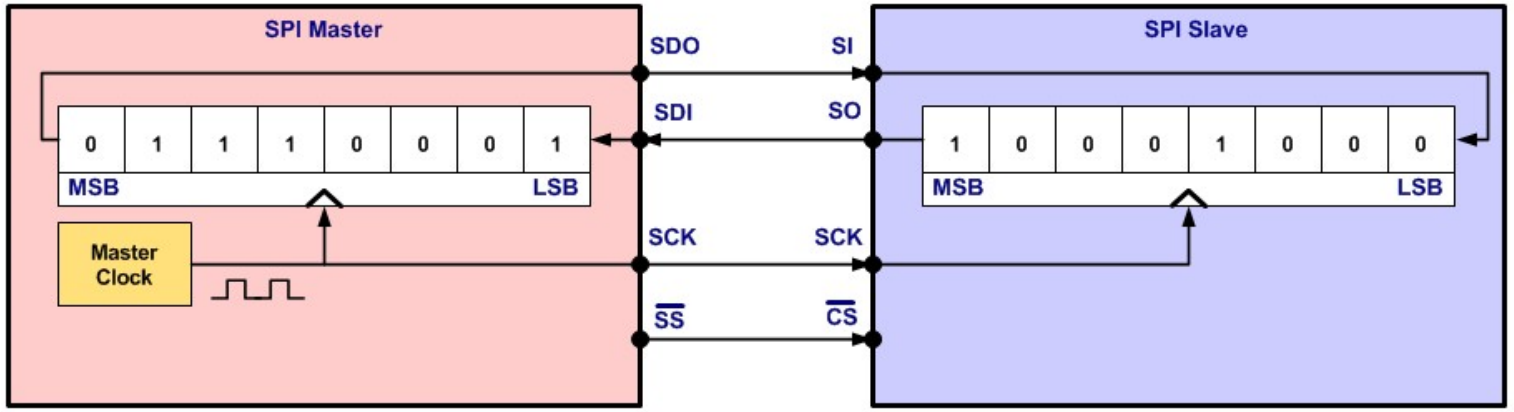
1. Один ведучий та один ведений.

2. Один ведучий, три незалежні (паралельні) ведені.

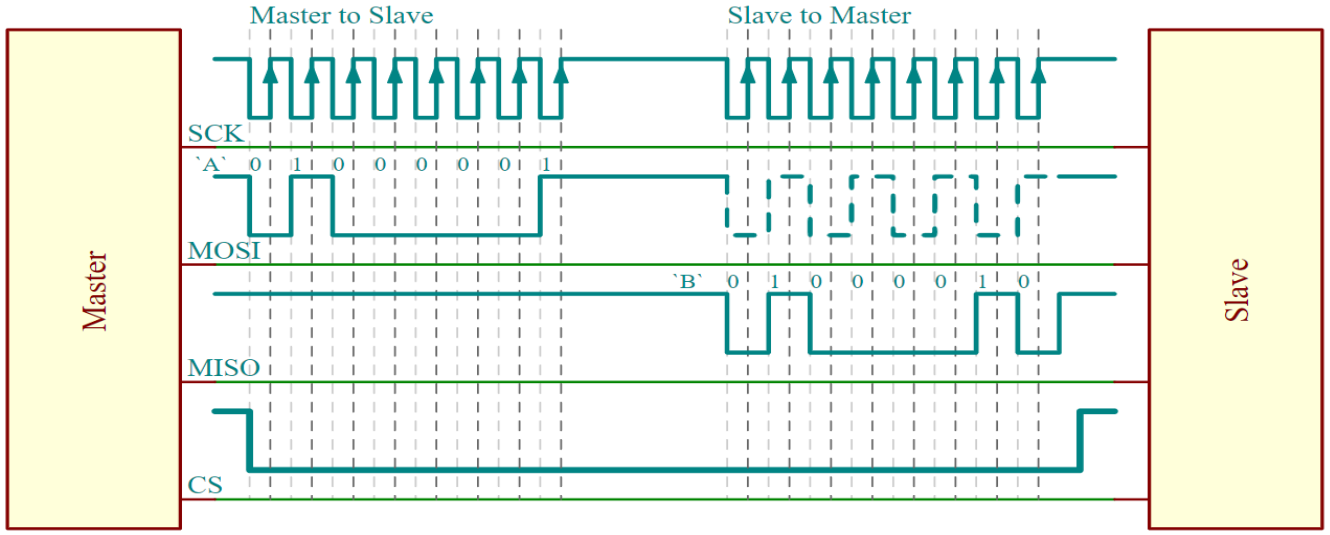
3. Один ведучий, три залежні (послідовні) ведені.

SPI – інтерфейс (продовження)

Спрощена структурна схема spi – інтерфейсу.



Демонстрація передачі даних по spi.

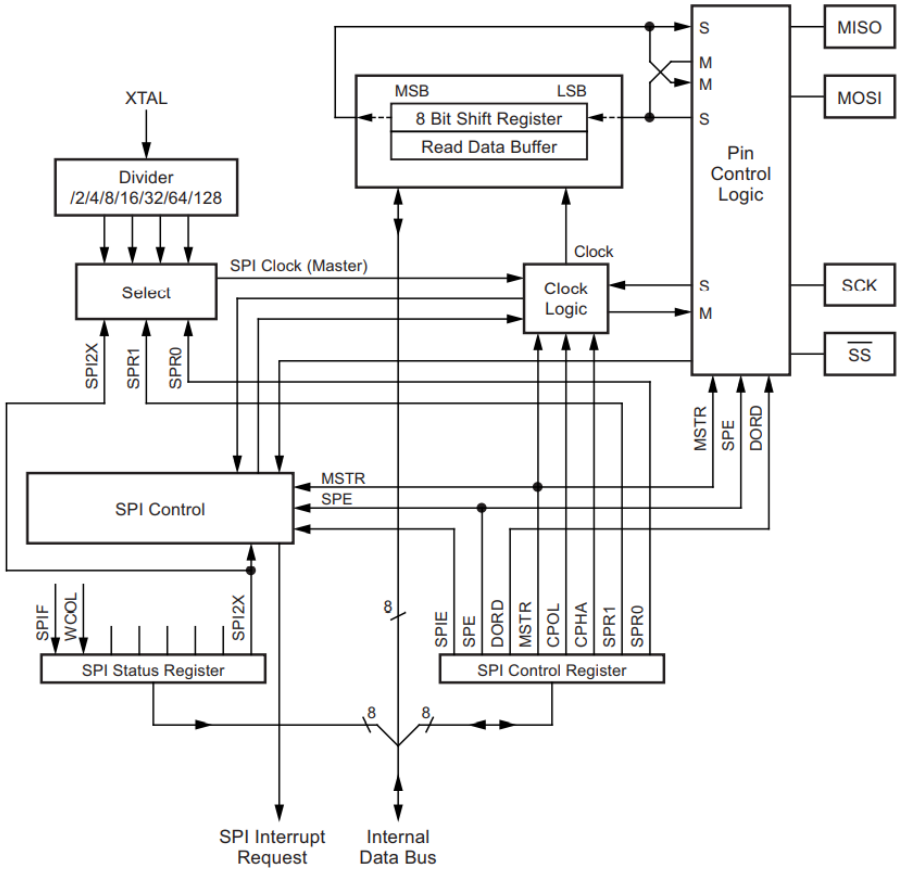


Програмна реалізація
прийому/передачі даних по SPI

```
uint8_t spi_transfer(uint8_t data)
{
  uint8_t i, rcv_data = 0;

    SPI_CLK = 0;
    for(i= 0; i< 8; i++)
    {
        if(data & 0x80)
           SPI_MOSI = 1;
        else
           SPI_MOSI = 0;

        rcv_data <<= 1;
        SPI_CK = 1;
        data <<= 1;
        rcv_data |= (SPI_MISO == 0)? 0: 1;
        SPI_CK = 0;
    }
  return rcv_data;
}
```

SPI інтерфейс мікроконтролера ATmega328

Блок-схема інтерфейсу SPI

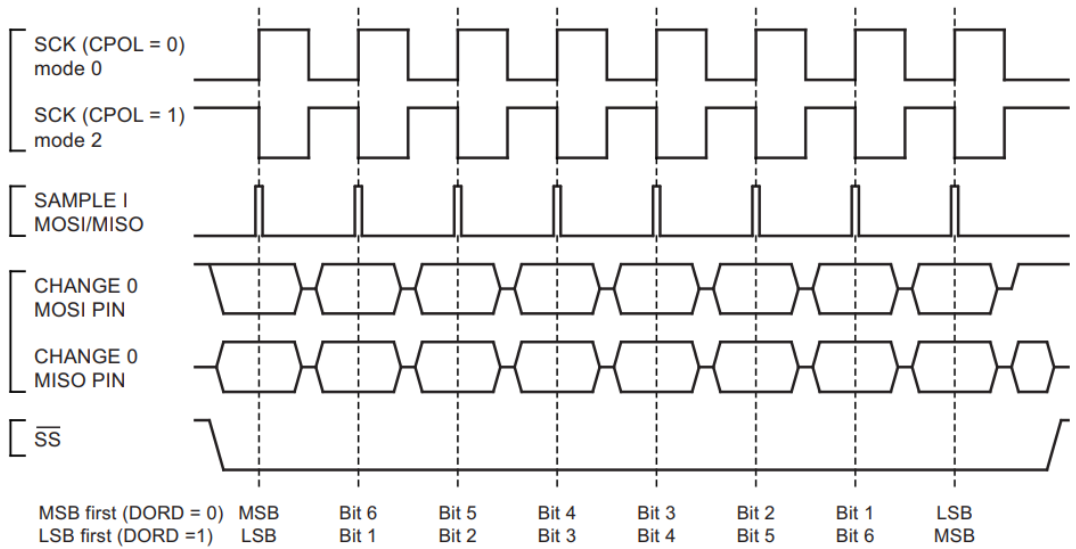


Режими роботи інтерфейсу spi

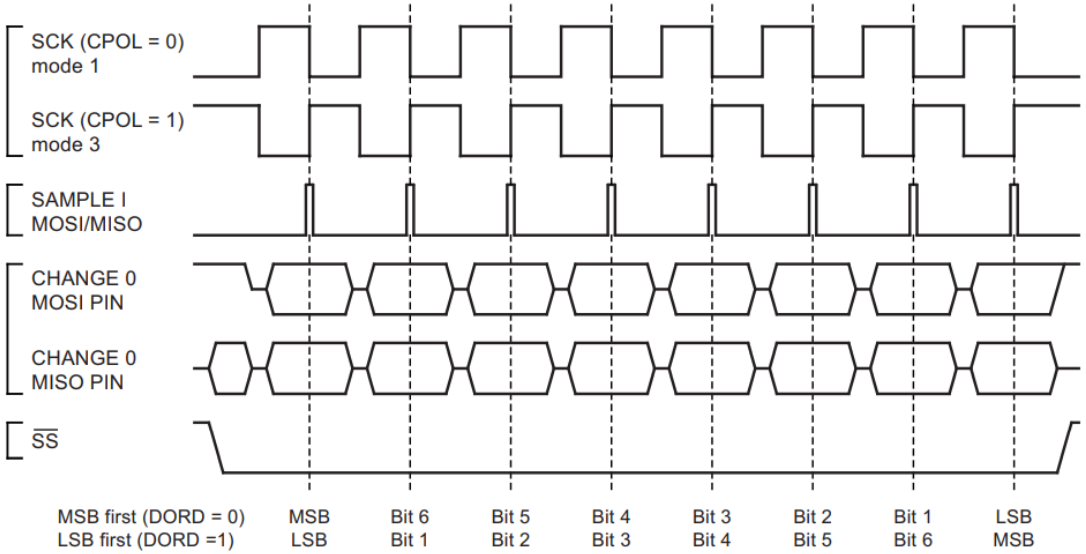
SPI Mode	Conditions	Leading Edge	Trailing eDge
0	CPOL=0, CPHA=0	Sample (rising)	Setup (falling)
1	CPOL=0, CPHA=1	Setup (rising)	Sample (falling)
2	CPOL=1, CPHA=0	Sample (falling)	Setup (rising)
3	CPOL=1, CPHA=1	Setup (falling)	Sample (rising)

Формат передачі даних

SPI Transfer Format with CPHA=0



SPI Transfer Format with CPHA=1



Регістри керування SPI модуля мікроконтролера ATMega328

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Регістр контролю **SPCR**.

Біт 7 - **SPIE**: Дозвіл переривання SPI. Цей біт дозволяє генерацію переривання SPI, якщо встановлено біт SPIF у регістрі SPSR.

Біт 6 - **SPE**: Вмикає-вимикає SPI інтерфейс.

Біт 5 - **DORD**: Встановлює порядок відправки біт, Якщо він встановлений в 1, то першим відправляється молодший біт, якщо в 0 - старший.

Біт 4 - **MSTR**: Назначає пристрій ведучим або веденим. При встановленні біта в 1 пристрій буде ведучим.

Біт 3 - **CPOL**: Полярність синхронізації, визначає, при якому фронті синхронізуючого імпульсу буде ініціюватися режим очікування.

Біт 2 - **CPHA**: Фаза тактування, тобто по якому саме фронту буде здійснюватися передача біта.

Біти 1, 0 - **SPR1, SPR0**: Вибір тактової частоти SPI. Ці біти контролюють швидкість (частоту на SCK) пристрою в режимі ведучого.

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Регістр статусу **SPSR**.

Біт 7 - **SPIF**: Прапор переривання. Встановлюється в 1 після прийому/передачі байта. Даний біт скидається апаратно після обробки відповідного вектора переривання. Крім того, біт SPIF очищається після зчитуванням регістра стану SPI із встановленим SPIF, та наступним доступом до регістра даних SPI (SPDR).

Біт 6 - **WCOL**: Прапор колізії, встановиться в 1 якщо під час передачі даних виконається спроба запису в регістр SPDR. Скидається аналогічно до SPIF.

Біт 0 - **SPI2X**: Біт подвоєння швидкості передачі.

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

Регістр даних SPI **SPDR** - використовується для передачі даних між регістровим файлом та регістром зсуву SPI. Запис даних в регістр ініціює передачу даних. Читання регістра спричиняє зчитування буфера прийому регістру зсуву SPI.

Таблиця швидкостей передачі даних по spi.

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Робота з SPI-інтерфейсом

```
void SPI_MasterInit(void)
{
    // Налаштувати піни MOSI (PB3) і SCK (PB5) як виходи
    DDRB |= (1<<PB3)|(1<<PB5);
    // Включити SPI, задати режим Master,
    // встановити швидкість fck/16
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(uint8_t cData)
{
    // Початок передачі
    SPDR = cData;
    // Чекати завершення передачі
    while(!(SPSR & (1<<SPIF)));
}
```

```
void SPI_SlaveInit(void)
{
    // Налаштувати пін MISO (PB4) як вихід
    // Решта пінів SPI будуть входами
    DDRB = 1<<PD4;
    DDRB &= ~((1<<PB3)|(1<<PB5));
    // Включити SPI
    SPCR = (1<<SPE);
}

uint8_t SPI_SlaveReceive(void)
{
    // Чекати завершення прийому даних
    while(!(SPSR & (1<<SPIF)));
    // повернути результат
    return SPDR;
}
```

Приклад використання апаратного SPI інтерфейсу, для виводу даних в регістр зсуву

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

void SPI_MasterInit(void)
{
    DDRB |= (1<<PB3)|(1<<PB5);
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(uint8_t cData)
{
    SPDR = cData;
    while(!(SPSR & (1<<SPIF)));
}

int main(void)
{
  uint8_t d = 1;
    SPI_MasterInit();
    DDRB |= 1<<PB2; // setup CS

    for(;;)
    {
        SPI_MasterTransmit(d);
        PORTB |= 1 << PB2;
        _delay_us(1);
        PORTB &= ~(1 << PB2);

        d <<= 1;
        if(d == 0) d = 1;
        _delay_ms(250);
    }
  return 0;
}
```