Microcontrollers

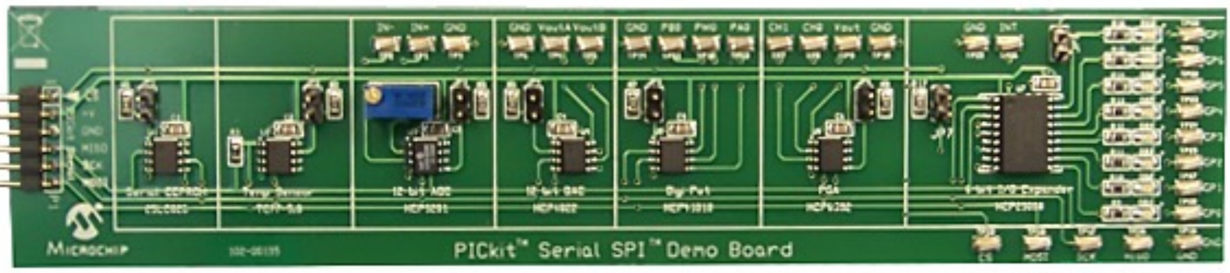
SPI – I2C

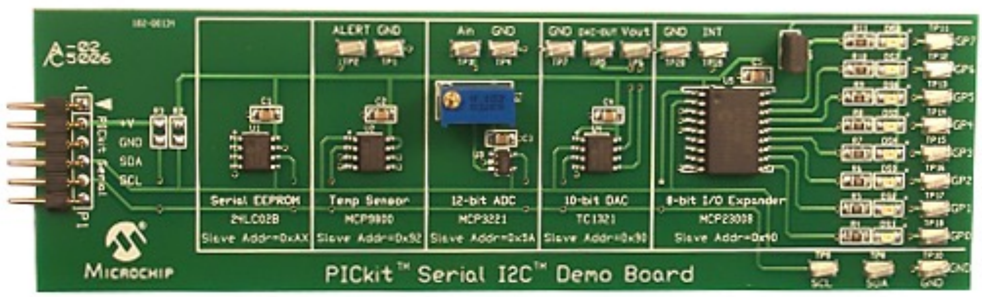
Christiaen Slot – c.g.m.slot@saxion.nl
Hans Stokkink – j.s.d.stokkink@saxion.nl

Content

- PICkit boards
- SPI – Serial Peripheral Interface
- SPI on Atmega328
- IO-expander example SPI
- I^{2}C – TWI – Two Wire Interface
- TWI on Atmega328
- IO-expander example TWI

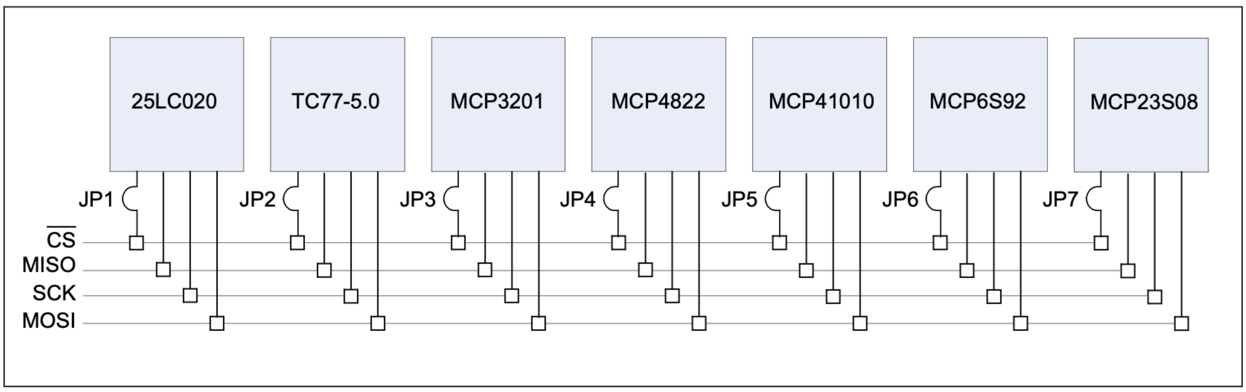
PICkit demo board SPI/I2C





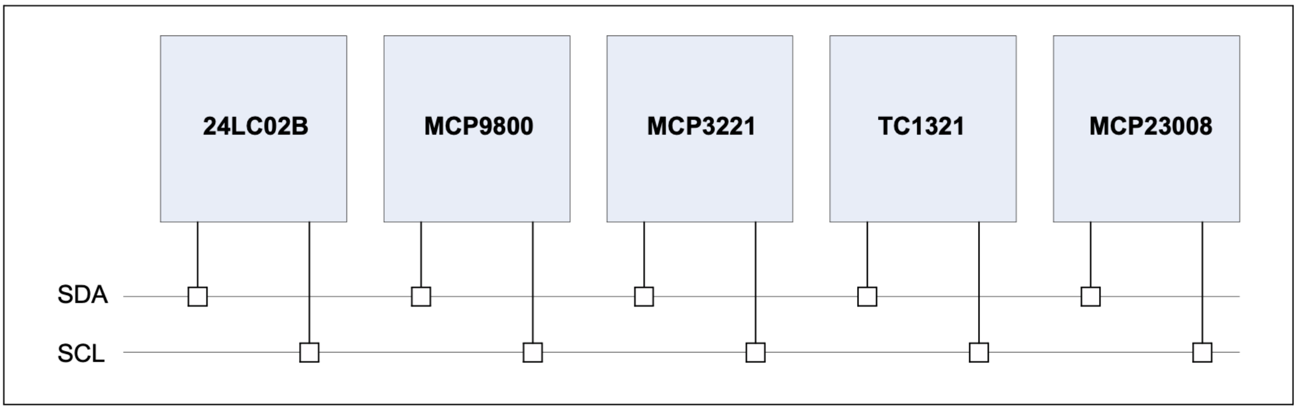
PICkit demo board SPI

Note:	Only one jumper should be inserted into JP1 though JP7 at a time. Incorrect device operation will occur.



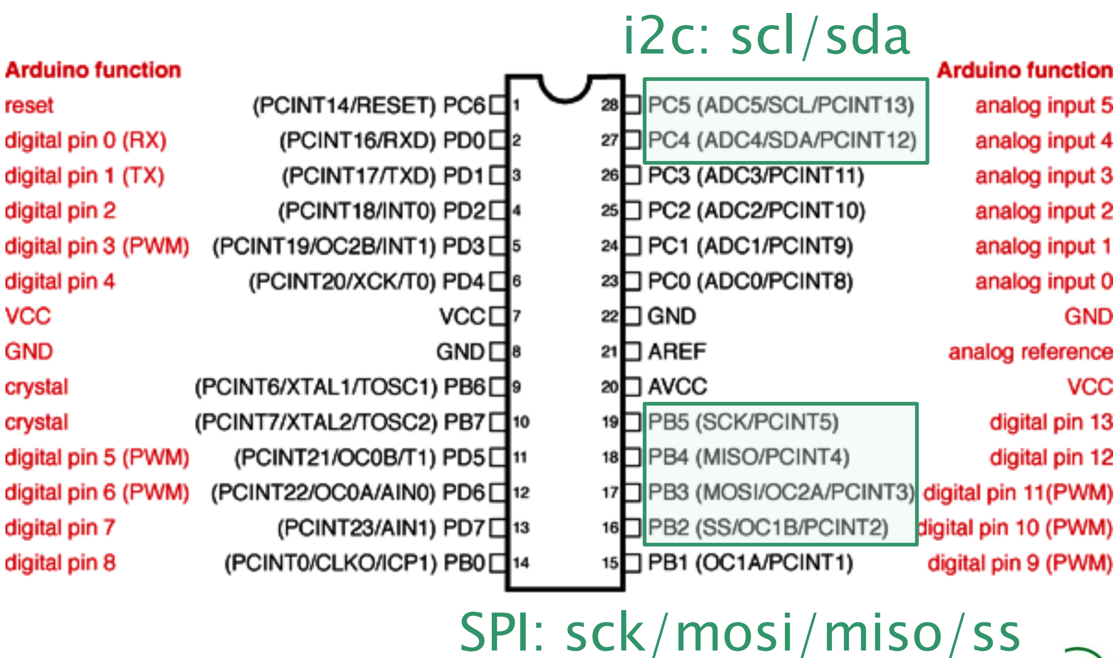
EEPROM | Thermal sensor | ADC (12-bit) | DAC | digital potmeter | PGA | IO expander

PICkit demo board I2C



EEPROM — Thermal sensor — ADC (12-bit) — DAC (10-bit) — IO expander

Atmega328 pin mapping



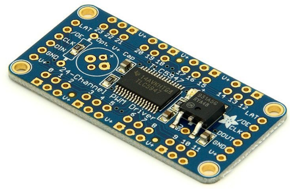
Content

- PICkit boards
- **SPI – Serial Peripheral Interface**
- SPI on Atmega328
- IO-expander example SPI
- I^{2}C – TWI – Two Wire Interface
- TWI on Atmega328
- IO-expander example TWI

Serial Peripheral Interface (SPI)

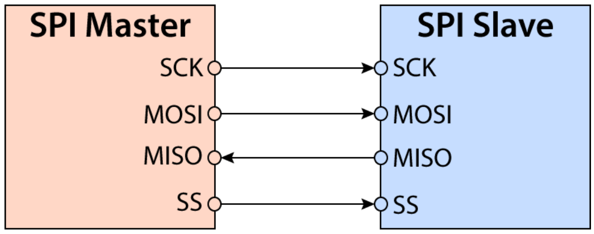
- The SPI is a Synchronous Protocol developed by Motorola.
- It is found on many devices and like I2C can be used to chain many devices together.

- send and receive data at the same time
- one device always Master, other(s) slave(s)
- can simply be implemented in software



SPI: 24-channel PWM driver

SPI – connections



Lines:

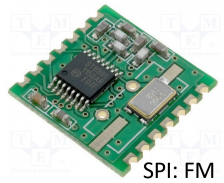
- Clock
- Slave select
- Master in – Slave out
- Slave in – Master out

SPI: 32 Mbit Flash memory

SPI – advantages

- Full duplex communication
- Higher throughput than I²C or SMBus
- Complete protocol flexibility for the bits transferred
- Not limited to 8-bit words
- Arbitrary choice of message size, content, and purpose
- Extremely simple hardware interfacing
- Typically lower power requirements than I²C or SMBus due to less circuitry (including pullups)
- No arbitration or associated failure modes
- Slaves use the master's clock, and don't need precision oscillators
- Slaves don't need a unique address -- unlike I²C or SCSI
- Transceivers are not needed

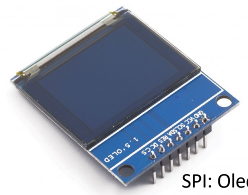
SPI – disadvantages



SPI: FM tranceiver

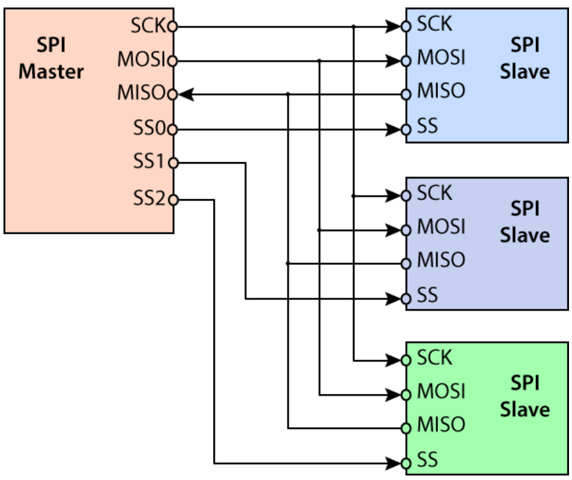
- Requires more pins on IC packages than I²C
- No in-band addressing; out-of-band chip select signals are required on shared buses
- No hardware flow control by the slave (but the master can delay the next clock edge to slow the transfer rate)
- No hardware slave acknowledgment (the master could be "talking" to nothing and not know it)
- Supports only one master device
- No error-checking protocol is defined
- Generally prone to noise spikes causing faulty communication
- Only handles short distances compared to RS-232, RS-485, or CAN-bus

SPI – multiple slaves

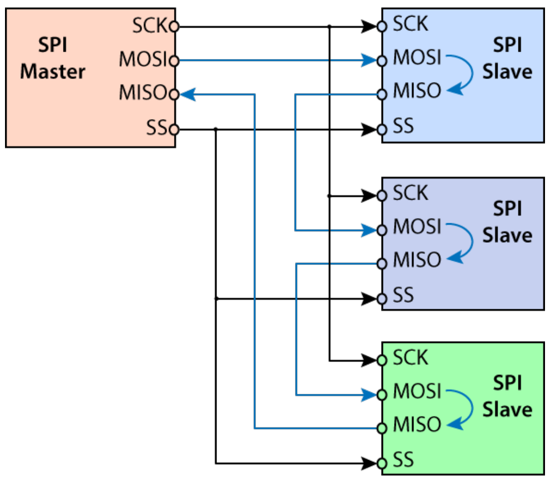


SPI: Oled

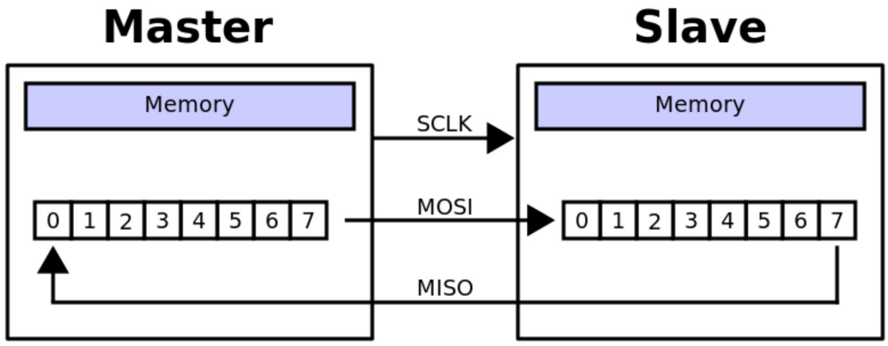
multiple “select” lines

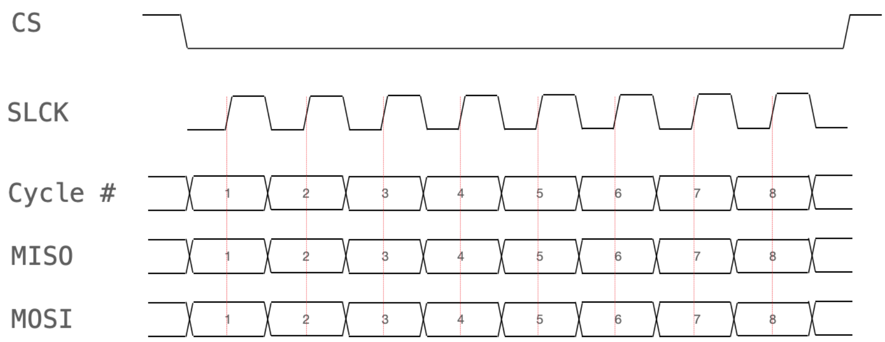


Daisy Chaining



SPI – “exchanging bytes”

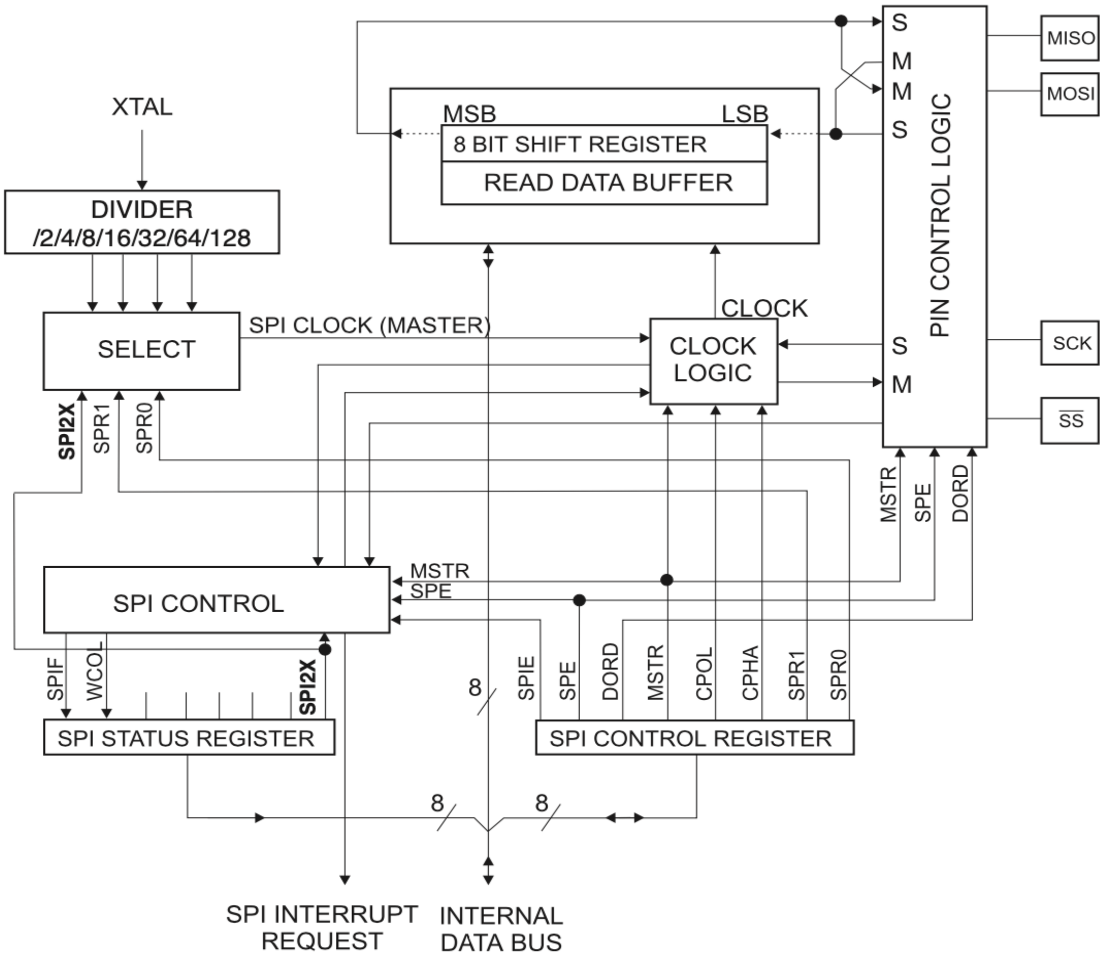




Content

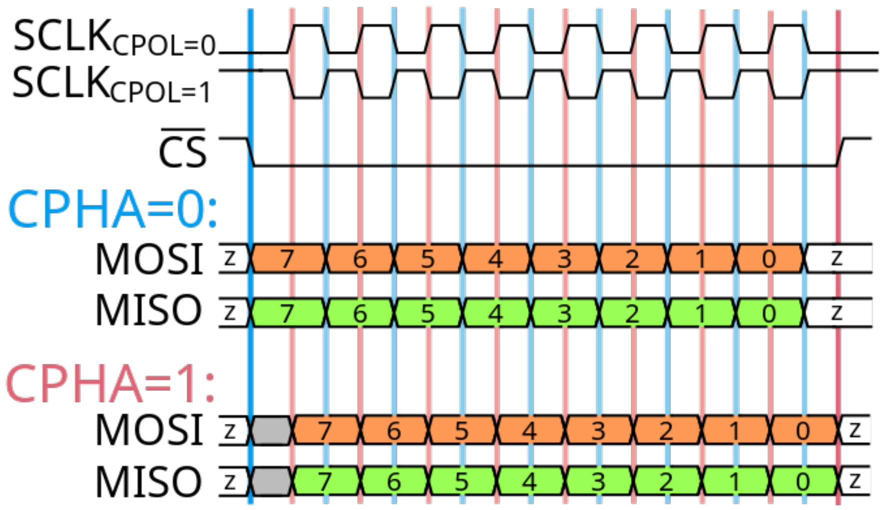
- PICkit boards
- SPI – Serial Peripheral Interface
- **SPI on Atmega328**
- IO-expander example SPI
- I^2C – TWI – Two Wire Interface
- TWI on Atmega328
- IO-expander example TWI

SPI – Atmega328



SPI – different modes

- Clock Polarity → when is clock idle
- Clock Phase → when to sample data



SPI – setup

Table 18-5. Relationship Between SCK and the Oscillator Frequency

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	$f_{osc}/64$

SPI Control Register

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	

```
// Set SS, MOSI and SCK output, all others input
DDRB = (1<<SS) | (1<<MOSI) | (1<<SCK);

// Enable SPI, Master, set clock rate fosc/16
// standard clock polarity and phase, no interrupt
SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0);

/* Set the slave select pin (Active low) */
DISABLE_SS;
```


SPI – transmitter/receiver

SPI Status Register

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	

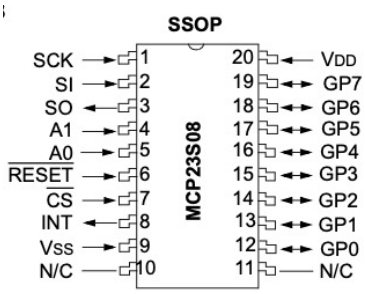
```
/* spi data buffer load and send */
uint8_t spi_tranceiver (uint8_t data)
{
    // Load data into the buffer
    SPDR = data;

    //Wait until transmission complete
    while(!(SPSR & (1<<SPIF) ));

    // Return received data
    return(SPDR);
}
```

Content

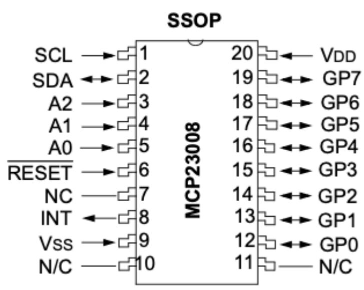
- PICkit boards
- SPI – Serial Peripheral Interface
- SPI on Atmega328
- **IO-expander example SPI**
- I^{2}C – TWI – Two Wire Interface
- TWI on Atmega328
- IO-expander example TWI

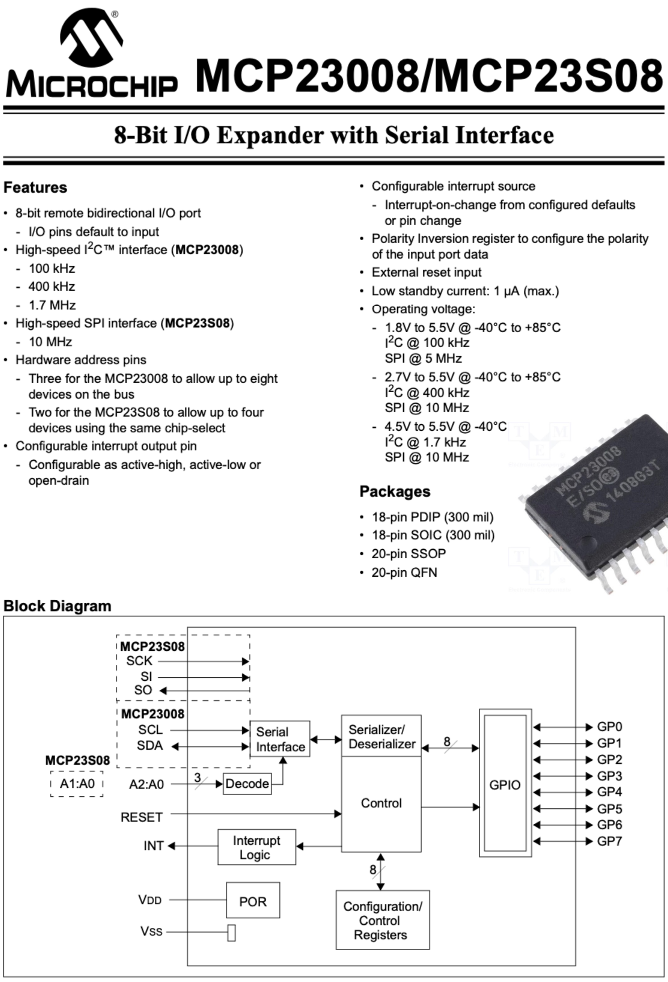
example: IO-expander

SPI



I2C





MICROCHIP **MCP23008/MCP23S08**

8-Bit I/O Expander with Serial Interface

Features

- 8-bit remote bidirectional I/O port
  - I/O pins default to input
- High-speed I^{2}C™ interface (**MCP23008**)
  - 100 kHz
  - 400 kHz
  - 1.7 MHz
- High-speed SPI interface (**MCP23S08**)
  - 10 MHz
- Hardware address pins
  - Three for the MCP23008 to allow up to eight devices on the bus
  - Two for the MCP23S08 to allow up to four devices using the same chip-select
- Configurable interrupt output pin
  - Configurable as active-high, active-low or open-drain
- Configurable interrupt source
  - Interrupt-on-change from configured defaults or pin change
- Polarity Inversion register to configure the polarity of the input port data
- External reset input
- Low standby current: 1 µA (max.)
- Operating voltage:
  - 1.8V to 5.5V @ -40°C to +85°C
    I^{2}C @ 100 kHz
    SPI @ 5 MHz
  - 2.7V to 5.5V @ -40°C to +85°C
    I^{2}C @ 400 kHz
    SPI @ 10 MHz
  - 4.5V to 5.5V @ -40°C
    I^{2}C @ 1.7 kHz
    SPI @ 10 MHz

Packages

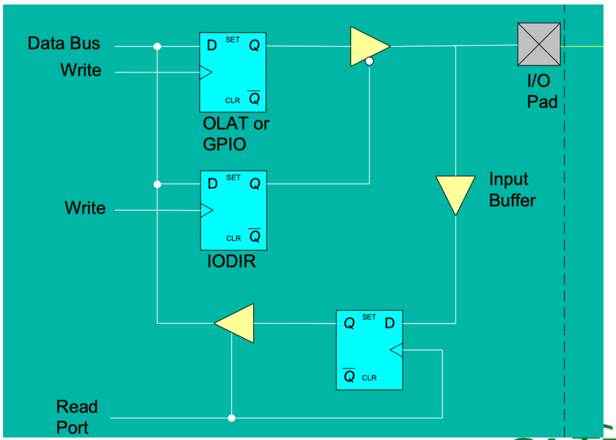
- 18-pin PDIP (300 mil)
- 18-pin SOIC (300 mil)
- 20-pin SSOP
- 20-pin QFN

Block Diagram

IO-expander

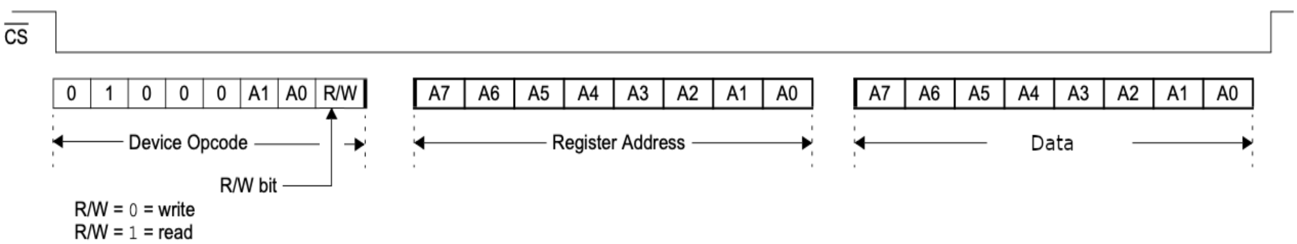
- address bits for multiple IO-expanders
- configure as "input" or "output"
- GPIO & output-latch register

- control registers
  - data direction
  - interrupt control
  - pull-up resisters
  - etc.



using IO-expander with SPI

1. "drop" CS line to start
2. send "control" byte (with address and R/W)
3. send "name" of the register to read/write
4. receive or send actual data byte
5. "raise" CS line to end



using IO-expander with SPI

```
/* IO expander data direction set */
void IOEXP_IODIR(uint8_t data)
{
    uint8_t r_data = 0;
    // Make slave select go low
    ENABLE_SS;
    /* Send device address + r/w */
    spi_tranceiver((1<<6)|WRITE);
    /* Send command */
    spi_tranceiver(IODIR);
    spi_tranceiver(data);
    // Make slave select go high
    DISABLE_SS;
}
```

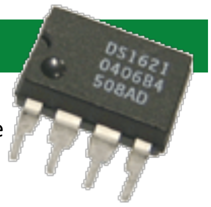
```
/* IO expander latch  set */
void IOEXP_datalatch(uint8_t data)
{
    // Make slave select go low
    ENABLE_SS;
    /* Send device address + r/w */
    spi_tranceiver((1<<6)|WRITE);
    /* Send command */
    spi_tranceiver(OUTP_LATCH);
    spi_tranceiver(data);
    // Make slave select go high
    DISABLE_SS;

}
```

```
#define READ 1
#define WRITE 0

#define IODIR 0
#define OUTP_LATCH 10
```


Content

- PICkit boards
- SPI – Serial Peripheral Interface
- SPI on Atmega328
- IO-expander example SPI
- **I²C – TWI – Two Wire Interface**
- TWI on Atmega328
- IO-expander example TWI

TWI – I^{2}C – overview



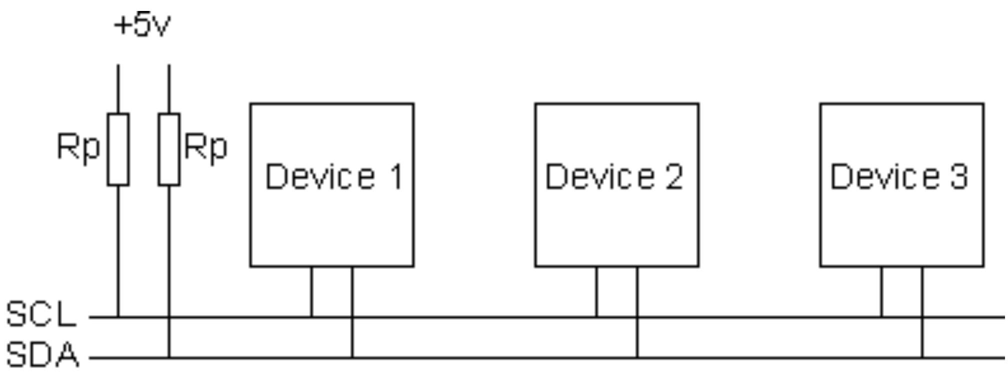
i2c: ds1612 temperature sensor

- I^{2}C (Philips 1982), Two Wire Interface (others)
- Multiple masters & slaves
- open drain Clock line & data line
- +5V or 3.3V
- 7 bits address (of which 16 reserved) = 112 nodes possible
- 10/100/400 kbits/s / 3.4 Mbits/s
- complex ordering of protocol bits/bytes

TWI – I^{2}C – connection

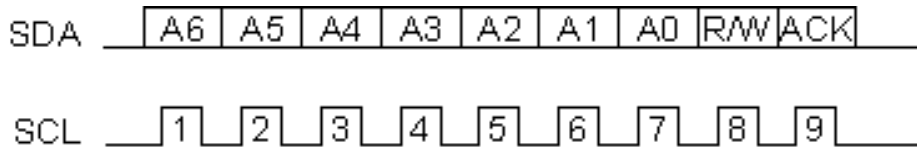
i2c: IR distance sensor

- connect pull ups to Vcc
- “Pull line low to activate”

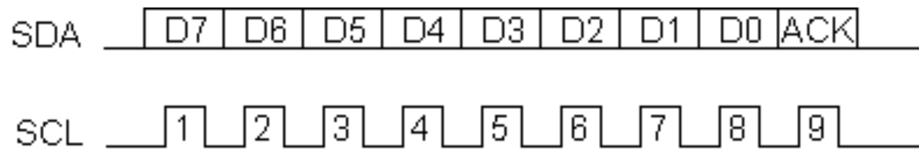


TWI – I²C – protocol (simplified)

- For each databit, clock line goes high-low
- Command: 7 bit adres + R/W + slave ACK

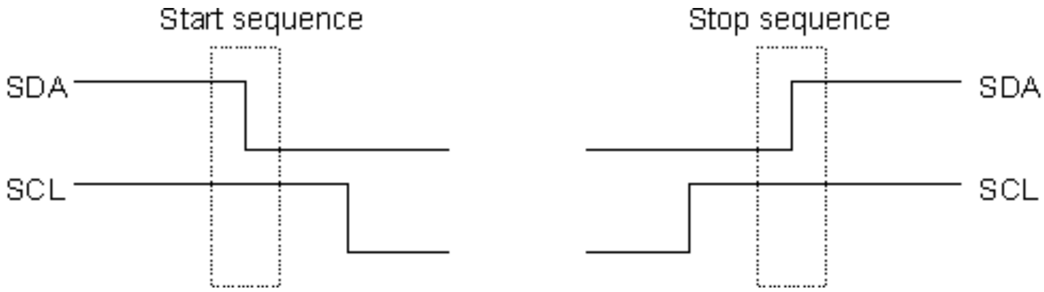


- Then receive/transmit mode … data + ACK



TWI – I²C – protocol (simplified)

- Start: pull SDA low, then SCL
- Stop: let SCL go high, let SDA go high



- Repeated start: Set SDA high, let SCL go high, and pull SDA low again.

Content

- PICkit boards
- SPI – Serial Peripheral Interface
- SPI on Atmega328
- IO-expander example SPI
- I^{2}C – TWI – Two Wire Interface
- **TWI on Atmega328**
- IO-expander example TWI

TWI – Atmega328

Figure 21-9. Overview of the TWI Module

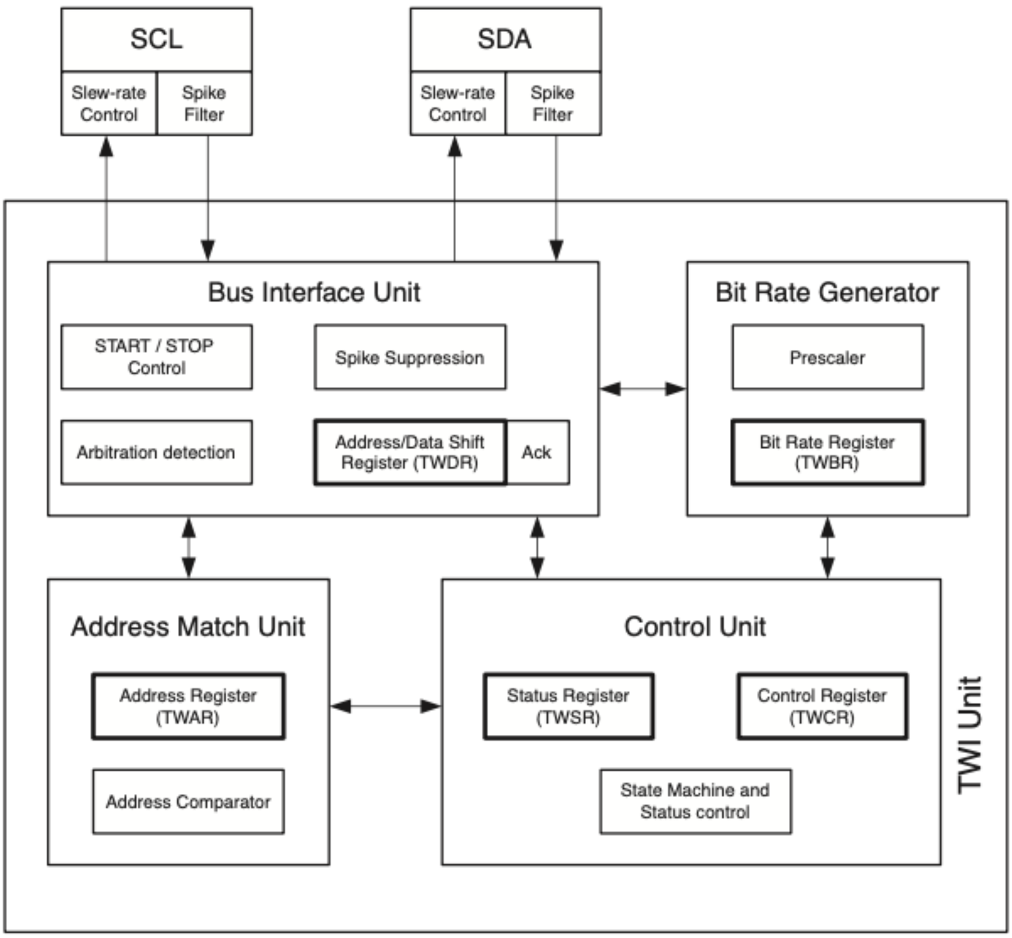


Figure 21-12. Formats and States in the Master Transmitter Mode

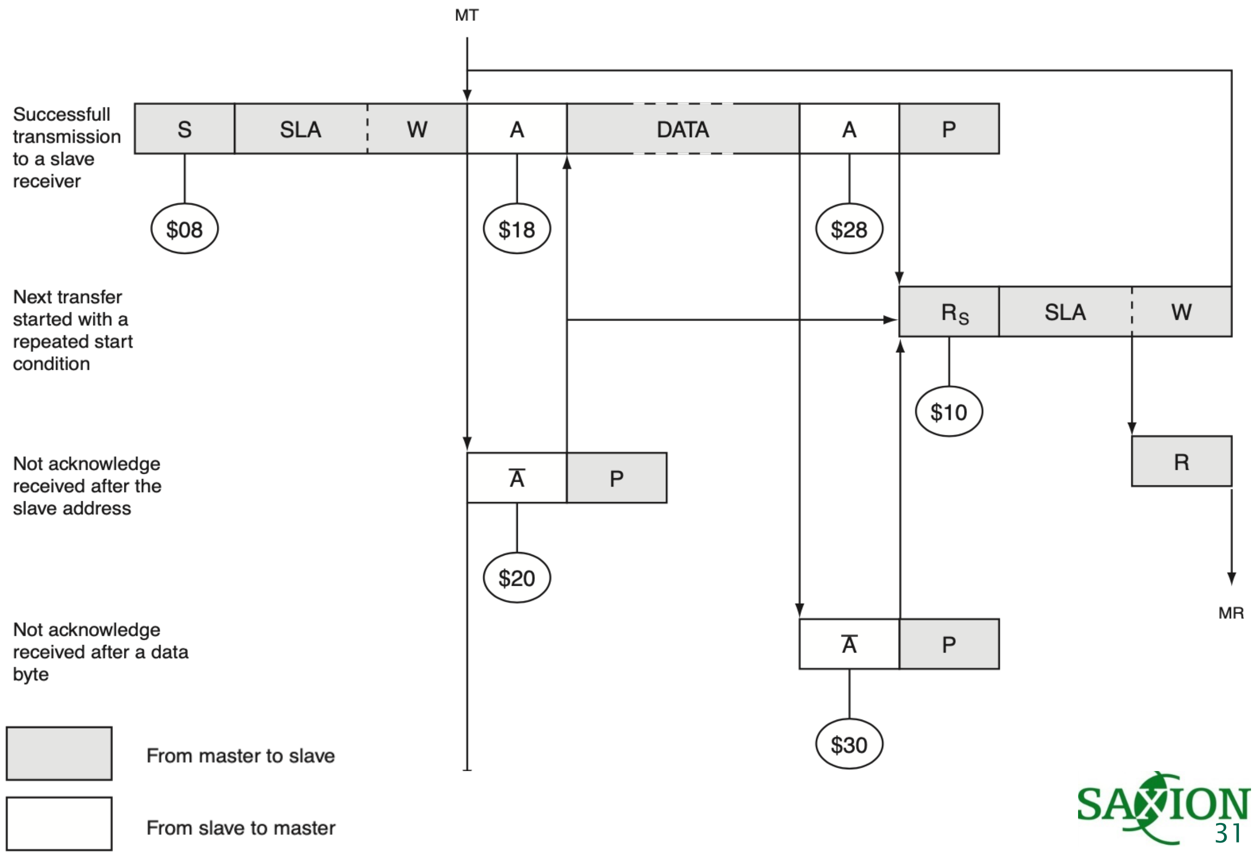
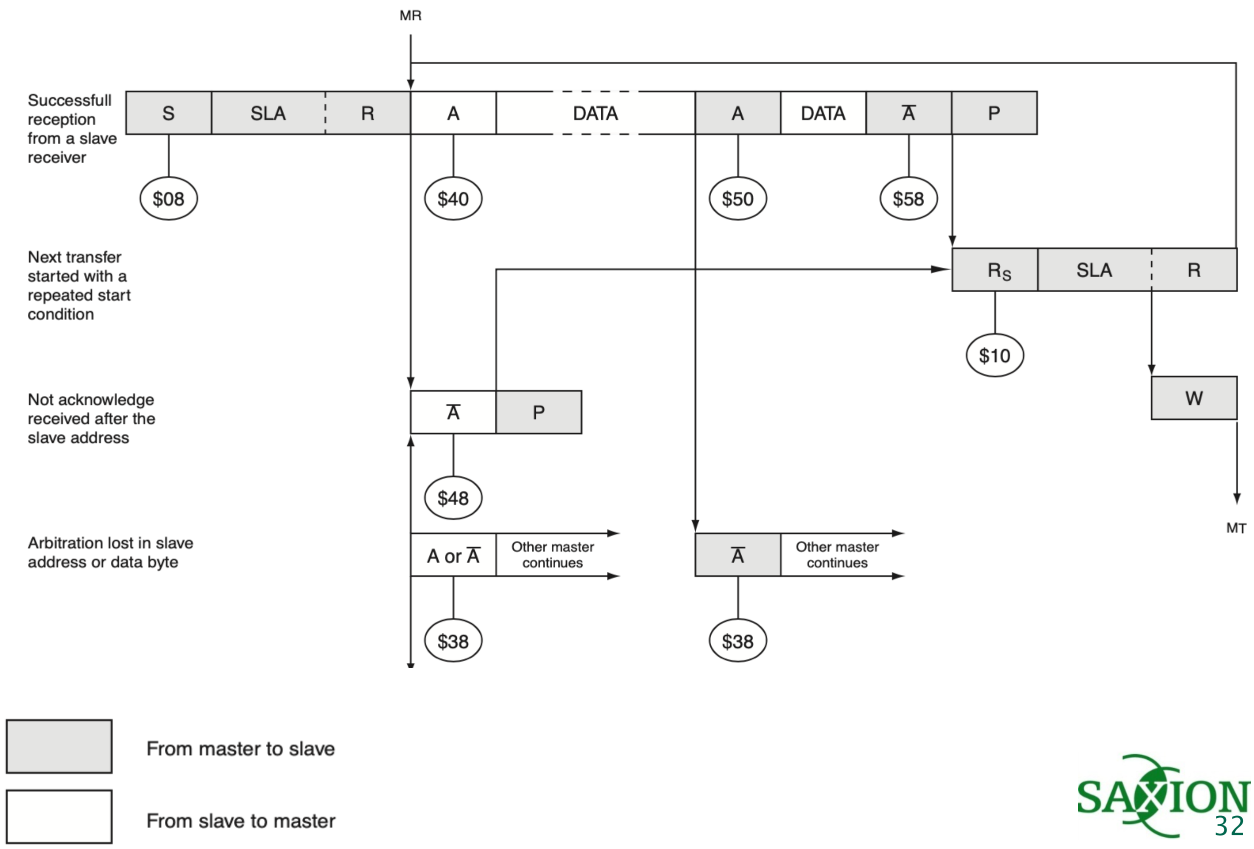


Figure 21-14. Formats and States in the Master Receiver Mode



Registers

TWI Control Register

Bit	7	6	5	4	3	2	1	0	
(0xBC)	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TWI Status Register

Bit	7	6	5	4	3	2	1	0	
(0xB9)	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

TWI Data Register

Bit	7	6	5	4	3	2	1	0	
(0xBB)	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

TWI (Slave) Address Register

Bit	7	6	5	4	3	2	1	0	
(0xBA)	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

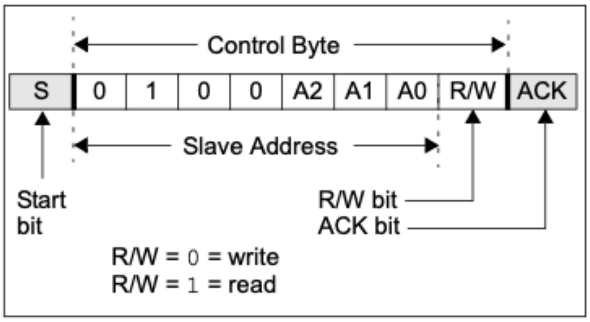
Content

- PICkit boards
- SPI – Serial Peripheral Interface
- SPI on Atmega328
- IO-expander example SPI
- I²C – TWI – Two Wire Interface
- TWI on Atmega328
- **IO-expander example TWI**

using IO-expander with TWI

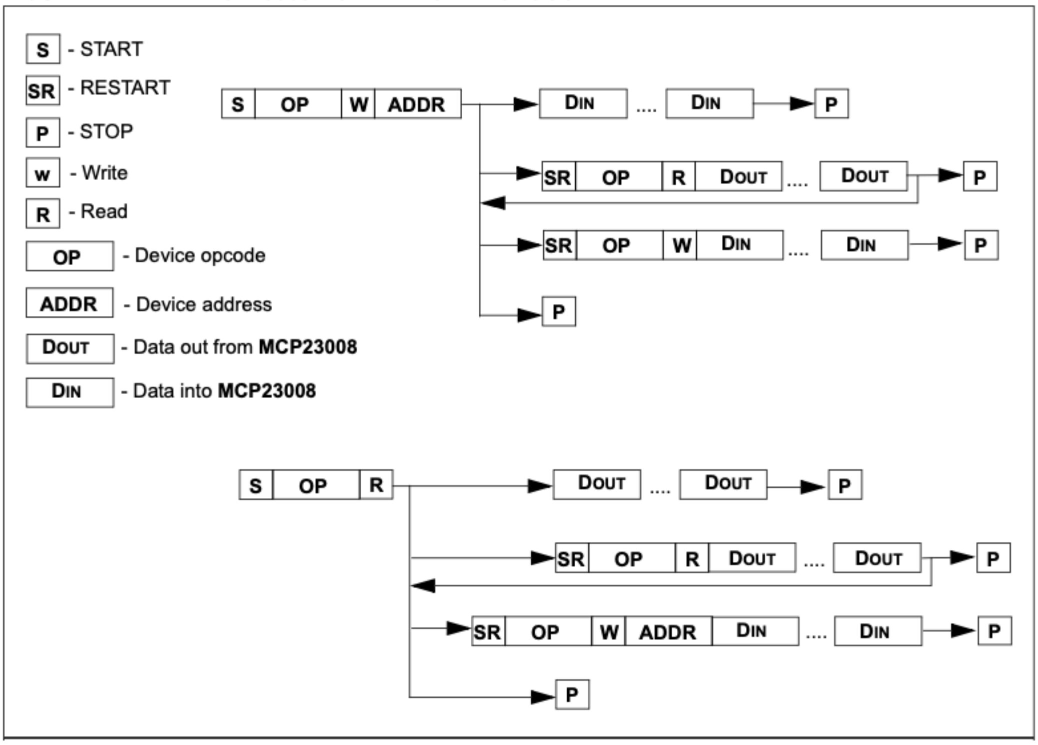
1. ‘start bit’
2. send ”control” byte (with address and R/W)
3. wait for ‘ack’ (or ‘nack’)
4. send/receive bytes followed by ‘ack’s’
5. ‘stop bit’

FIGURE 1-2: I^2C^{TM} CONTROL BYTE FORMAT



using IO-expander with TWI

FIGURE 1-1: MCP23008 I^2C^{TM} DEVICE PROTOCOL



```c
/* START I2C Routine */
unsigned char i2c_transmit(unsigned char type) {
    switch(type) {
        case I2C_START:    // Send Start Condition
        TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
        break;
        case I2C_DATA:     // Send Data with No-Acknowledge
        TWCR = (1 << TWINT) | (1 << TWEN);
        break;
        case I2C_DATA_ACK: // Send Data with Acknowledge
        TWCR = (1 << TWEA) | (1 << TWINT) | (1 << TWEN);
        break;
        case I2C_STOP:     // Send Stop Condition
        TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
        return 0;
    }
    // Wait for TWINT flag set on Register TWCR
    while (!(TWCR & (1 << TWINT)));
    // Return TWI Status Register, mask the prescaler bits (TWPS1,TWPS0)
    return (TWSR & 0xF8);
}

void Write_MCP23008(unsigned char reg_addr,unsigned char data)
{
    // Start the I2C Write Transmission
    i2c_start(MCP23008_ID,MCP23008_ADDR,TW_WRITE);
    // Sending the Register Address
    i2c_write(reg_addr);
    // Write data to MCP23008 Register
    i2c_write(data);
    // Stop I2C Transmission
    i2c_stop();
}
```