ECE3411 – Spring 2019

Lecture 14

I2C: Inter Integrated Circuit

Sung Yeul Park
Department of Electrical & Computer Engineering
University of Connecticut
Email: sung_yeul.park@uconn.edu

With the help of:

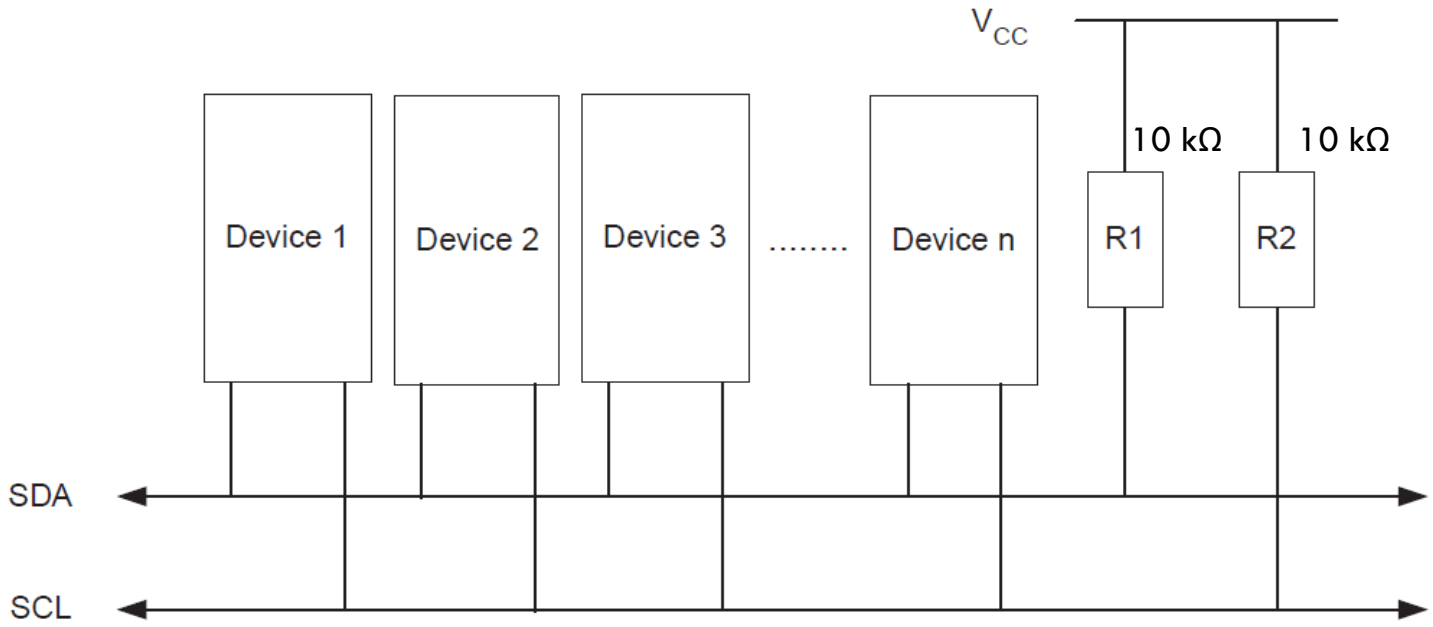
www.wikipedia.org

ATmega328P Datasheet

Copied from ECE3411 – Fall 2015,
by Marten van Dijk and Syed Kamran Haider

I^2C: Inter Integrated Circuit

- Also known as Two Wire Interface (TWI) or I square C
- Allows up to 128 different devices to be connected using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA).
- A pull-up resistor (typically 10 kΩ) is needed for each of the TWI bus lines.
- All devices connected to the bus have individual addresses.



I^2C Terminologies

- I^2C (TWI) protocol allows several devices (up to 128) to be connected.
- Each device is identified by a configurable 7-bit address.
- Each device can communicate with any other device
  - The transmitter addresses the receiver by its 7-bit address.

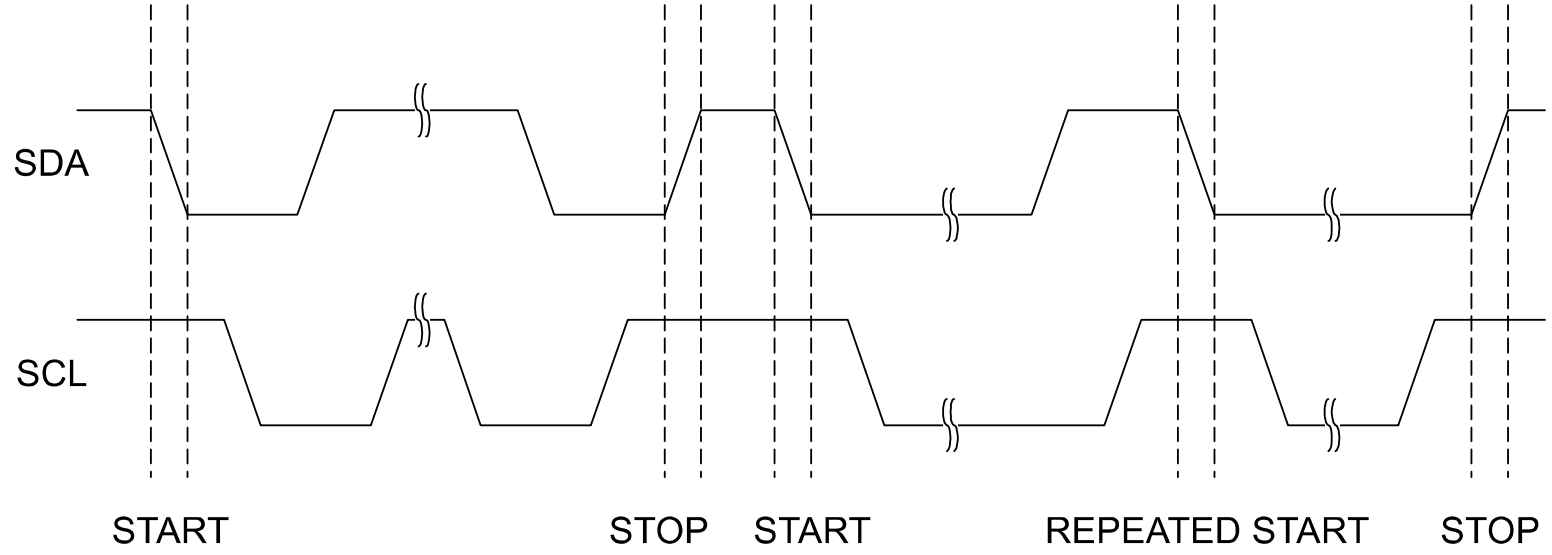
Table 26-1. TWI Terminology

Term	Description
Master	The device that initiates and terminates a transmission. The Master also generates the SCL clock.
Slave	The device addressed by a Master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

I²C START and STOP Conditions

- START and STOP conditions are signaled by changing the level of the SDA line when the SCL line is high.
- When a new START condition is issued between a START and STOP condition, this is referred to as a REPEATED START condition

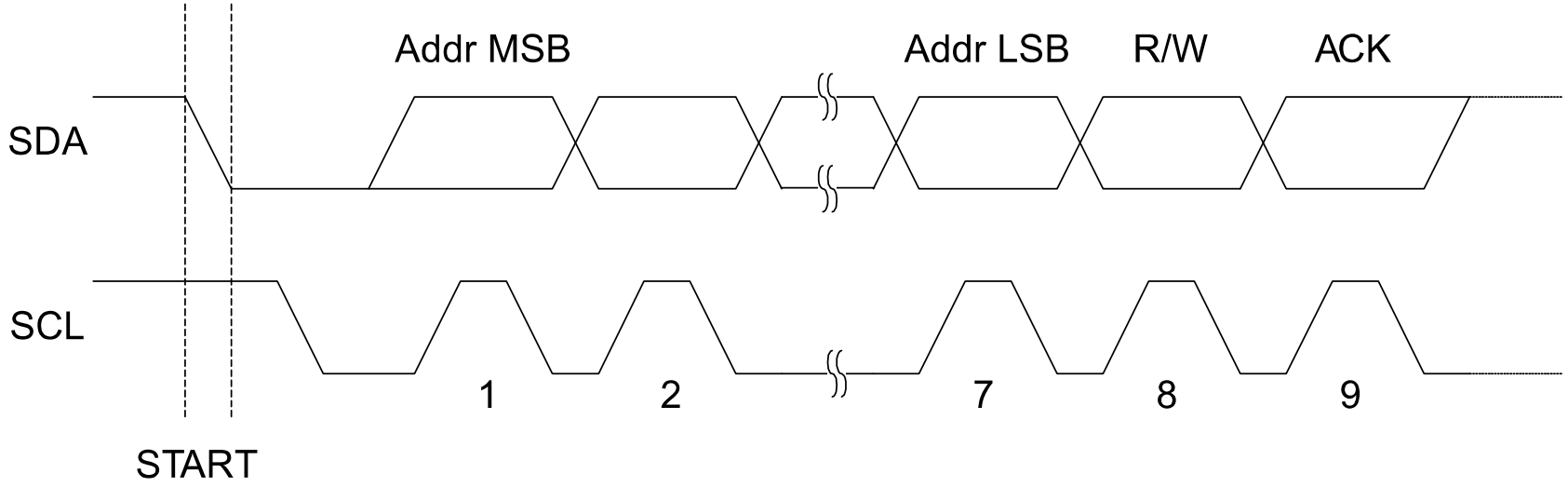
Figure 26-3. START, REPEATED START, and STOP Conditions



I²C Address Packet Format

- All address packets transmitted on the TWI bus are 9 bits long:
  - 7 address bits, one READ/WRITE control bit and an acknowledge bit.
- When a Slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle.
- The Master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission.

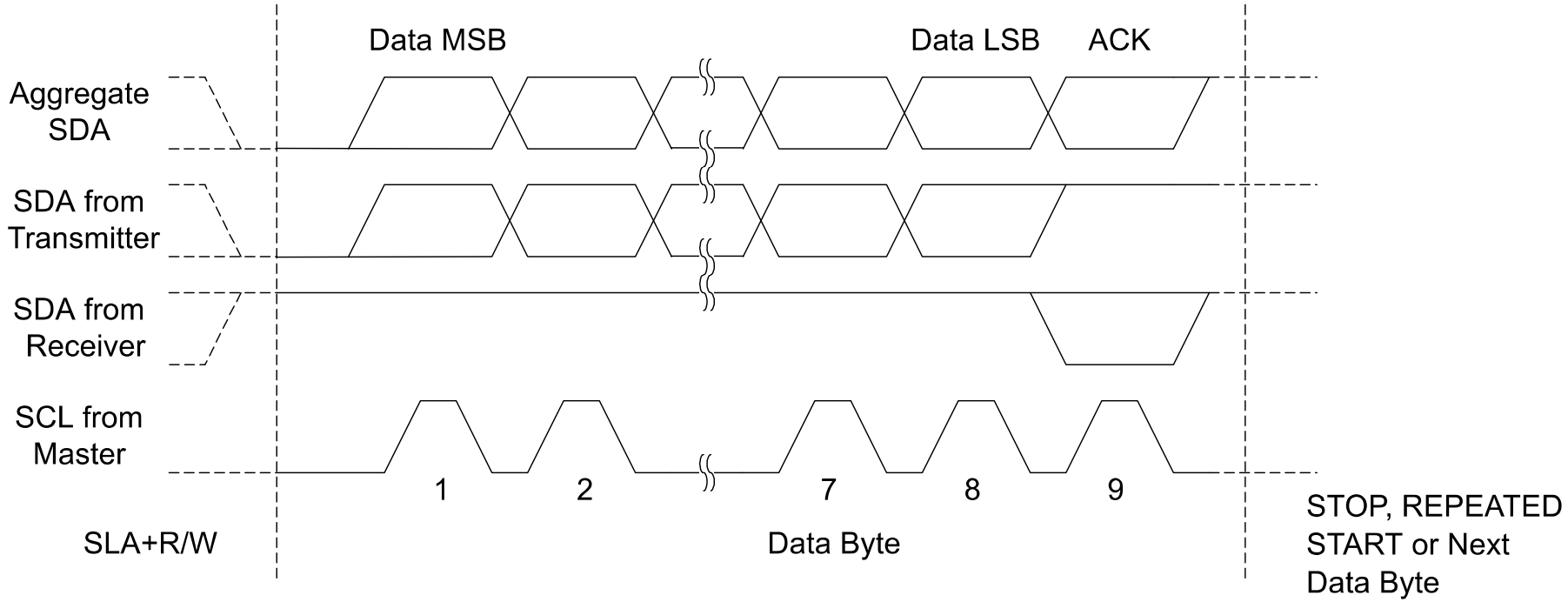
Figure 26-4. Address Packet Format



I²C Data Packet Format

- All data packets transmitted on the TWI bus are 9 bits long:
  - One data byte and one acknowledge bit.
- An Acknowledge (ACK) is signaled by the Receiver pulling the SDA line low during the ninth SCL cycle. If the Receiver leaves the SDA line high, a NACK is signaled.

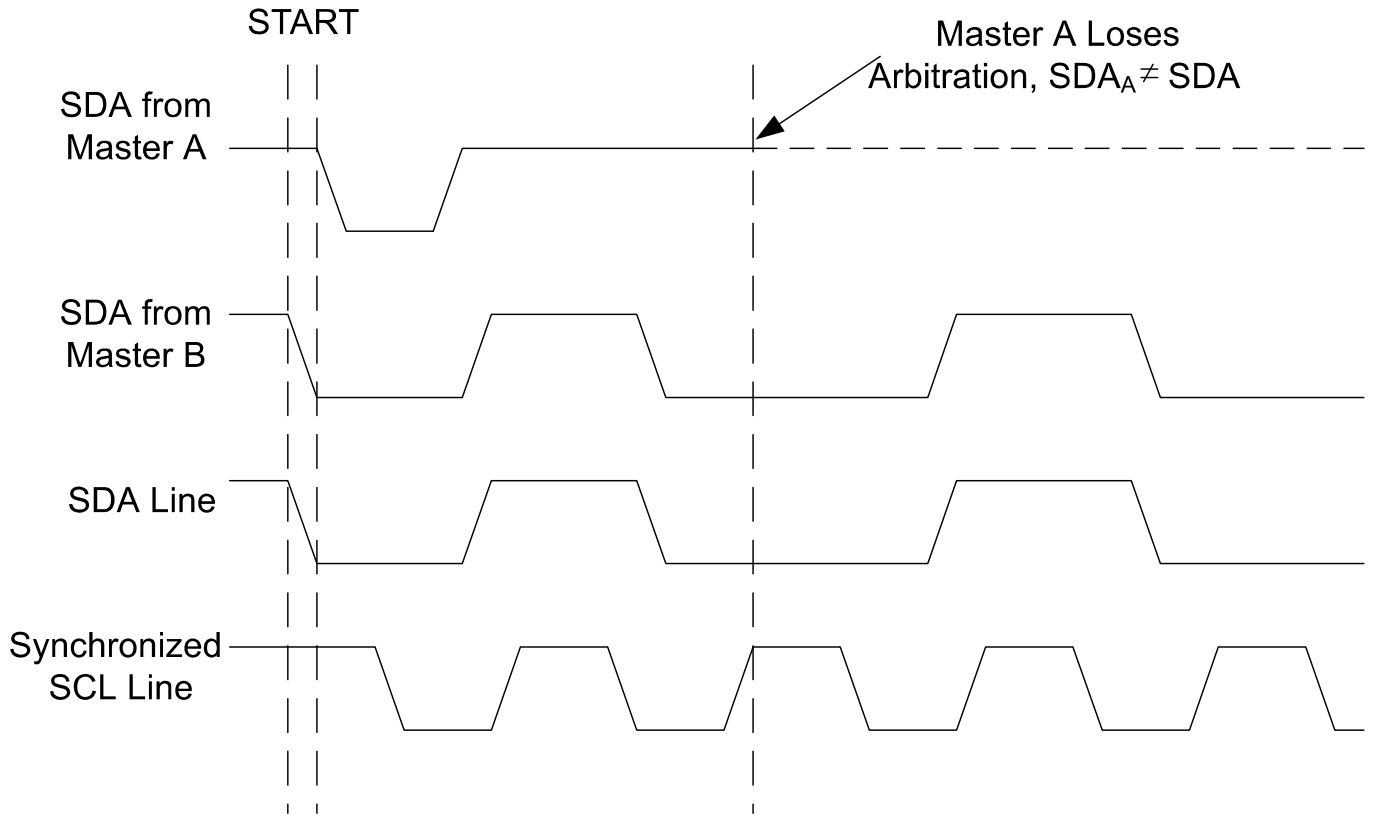
Figure 26-5. Data Packet Format



I²C Bus Arbitration

- Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data.
- If the value read from the SDA line does not match the value the Master had output, it has lost the arbitration.

Figure 26-8. Arbitration Between Two Masters



TWCR

Bit	7	6	5	4	3	2	1	0
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN		TWIE
Access	R/W	R/W	R/W	R/W	R	R/W		R/W
Reset	0	0	0	0	0	0		0

Bit 7 – TWINT TWI Interrupt Flag

Bit 6 – TWEA TWI Enable Acknowledge

Bit 5 – TWSTA TWI START Condition

Bit 4 – TWSTO TWI STOP Condition

Bit 3 – TWWC TWI Write Collision Flag

Bit 2 – TWEN TWI Enable

Bit 0 – TWIE TWI Interrupt Enable

TWSR

Bit	7	6	5	4	3	2	1	0
	TWS7	TWS6	TWS5	TWS4	TWS3		TWPS[1:0]	
Access	R	R	R	R	R		R/W	R/W
Reset	1	1	1	1	1		0	0

Bits 3, 4, 5, 6, 7 – TWS TWI Status Bit
The TWS[7:3] reflect the status of the TWI logic and the two-wire Serial Bus. The different status codes are described in Transmission Modes. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the Status bits. This makes status checking independent of prescaler setting. This approach is used in this data sheet unless otherwise noted.

Bits 1:0 – TWPS[1:0] TWI Prescaler
These bits can be read and written, and control the bit rate prescaler.

Table 26-8. TWI Bit Rate Prescaler

TWPS[1:0]	Prescaler Value
00	1
01	4
10	16
11	64

To calculate bit rates, refer to Bit Rate Generator Unit. The value of TWPS[1:0] is used in the equation.

A typical I^2C Transmission Summary

- When the TWI has finished an operation and expects application response, the TWINT Flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT Flag is set, the user must update all TWI Registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI Register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be set.
- Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

I²C Transmission Example

```
// General TWI Master staus codes
#define TWI_START                  0x08  // START has been transmitted
#define TWI_REP_START              0x10  // Repeated START has been transmitted
#define TWI_ARB_LOST               0x38  // Arbitration lost
// TWI Master Transmitter and Receiver staus codes
#define TWI_MTX_ADR_ACK            0x18  // SLA+W has been transmitted and ACK received
#define TWI_MTX_ADR_NACK           0x20  // SLA+W has been transmitted and NACK received
#define TWI_MTX_DATA_ACK           0x28  // Data byte has been transmitted and ACK received
#define TWI_MTX_DATA_NACK          0x30  // Data byte has been transmitted and NACK received
#define TWI_MRX_ADR_ACK            0x40  // SLA+R has been transmitted and ACK received
#define TWI_MRX_ADR_NACK           0x48  // SLA+R has been transmitted and NACK received
#define TWI_MRX_DATA_ACK           0x50  // Data byte has been received and ACK transmitted
#define TWI_MRX_DATA_NACK          0x58  // Data byte has been received and NACK transmitted
// TWI Slave Transmitter staus codes
#define TWI_STX_ADR_ACK            0xA8  // Own SLA+R has been received; ACK has been returned
#define TWI_STX_ADR_ACK_M_ARB_LOST 0xB0  // Arbitration lost in SLA+R/W as Master;
                                         // own SLA+R has been received; ACK has been returned
#define TWI_STX_DATA_ACK           0xB8  // Data byte in TWDR has been transmitted;
                                         // ACK has been received
#define TWI_STX_DATA_NACK          0xC0  // Data byte in TWDR has been transmitted;
                                         // NOT ACK has been received
#define TWI_STX_DATA_ACK_LAST_BYTE 0xC8  // Last data byte in TWDR has been transmitted
                                         // (TWEA = "0"); ACK has been received
```

I²C Transmission Example

```
// TWI Slave Receiver staus codes
#define TWI_SRX_ADR_ACK              0x60  // Own SLA+W has been received ACK has been returned
#define TWI_SRX_ADR_ACK_M_ARB_LOST 0x68  // Arbitration lost in SLA+R/W as Master; own SLA+W
                                         // has been received; ACK has been returned
#define TWI_SRX_GEN_ACK              0x70  // General call address has been received;
                                         // ACK has been returned
#define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78  // Arbitration lost in SLA+R/W as Master; General
                                         // call address received; ACK has been returned
#define TWI_SRX_ADR_DATA_ACK         0x80  // Previously addressed with own SLA+W; data has been
                                         // received; ACK has been returned
#define TWI_SRX_ADR_DATA_NACK        0x88  // Previously addressed with own SLA+W; data has been
                                         // received; NOT ACK has been returned
#define TWI_SRX_GEN_DATA_ACK         0x90  // Previously addressed with general call; data has
                                         // been received; ACK has been returned
#define TWI_SRX_GEN_DATA_NACK        0x98  // Previously addressed with general call; data has
                                         // been received; NOT ACK has been returned
#define TWI_SRX_STOP_RESTART         0xA0  // A STOP condition or repeated START condition has
                                         // been received while still addressed as Slave
// TWI Miscellaneous status codes
#define TWI_NO_STATE                 0xF8  // No relevant state info available; TWINT = "0"
#define TWI_BUS_ERROR                0x00  // Bus error due to illegal START or STOP condition
```

I²C Transmission Example

```
#define WRITE 0

uint8_t TWI_Master_Transmit(uint8_t Address, uint8_t Data)
{
        TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);      // Send START condition
        while (!(TWCR & (1<<TWINT)));                // Wait for TWINT Flag set.
        if ((TWSR & 0xF8) != TWI_START)              // Check value of TWI Status Register.
                ERROR();

        TWDR = (Address << 1) | (WRITE);             // Load SLA_W (Slave Address & Write) into TWDR.
        TWCR = (1<<TWINT) | (1<<TWEN);               // Clear TWINT to start address transmission.
        while (!(TWCR & (1<<TWINT)));                // Wait for TWINT Flag set.
        if ((TWSR & 0xF8) != TWI_MTX_ADR_ACK)        // Check value of TWI Status Register.
                ERROR();

        TWDR = Data;                                 // Load DATA into TWDR Register.
        TWCR = (1<<TWINT) | (1<<TWEN);               // Clear TWINT bit to start transmission of data.
        while (!(TWCR & (1<<TWINT)));                // Wait for TWINT Flag set.
        if (((TWSR & 0xF8) != TWI_MTX_DATA_ACK)&&    // Check value of TWI Status Register.
            ((TWSR & 0xF8) != TWI_MTX_DATA_NACK))
                ERROR();
        TWCR = (1<<TWINT)|(1<<TWEN)| (1<<TWSTO);     // Transmit STOP condition.
}
```

Note: The code above assumes that several definitions have been made, for example by using include-files.

I²C Reception Example

```
void TWI_Slave_Initialize(uint8_t Address)
{
        TWAR = (Address << 1)|(1);     // Load Slave Address into TWAR Register.
        TWCR = (1<<TWEA)|(1<<TWEN);    // Enable TWI & Acknowledgements.
}
```

```
uint8_t TWI_Slave_Receive(void)
{
        TWCR = (1<<TWEA)|(1<<TWEN);              // Enable TWI & Acknowledgements.
        while (!(TWCR & (1<<TWINT)));            // Wait for TWINT set (once slave is addressed)
        if ((TWSR & 0xF8) != TWI_SRX_ADR_ACK) // Check value of TWI Status Register.
                ERROR();
        TWCR = (1<<TWINT) | (1<<TWEN);           // Clear TWINT bit start reception of data.
        while (!(TWCR & (1<<TWINT))); // Wait for TWINT Flag set.
        if (((TWSR & 0xF8) != TWI_SRX_ADR_DATA_ACK) &&// Check if Data rcvd & ACK returned
            ((TWSR & 0xF8) != TWI_SRX_ADR_DATA_NACK))
                ERROR();
        TWCR = (1<<TWINT) | (1<<TWEN);           // Clear TWINT bit.
        return TWDR;                             // Read TWDR Register.
}
```

Note: The code above assumes that several definitions have been made, for example by using include-files.

I2C based temperature sensor