11 I2C Schnittstelle

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

10. I2C Schnittstelle

Ziele

Nach dieser Lektion sollten Sie:

1. wissen wie die Kommunikation zwischen I2C Master und Slave funktioniert

Video

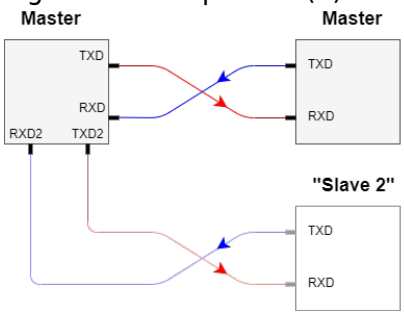
Dokumentation von Atmel

- Application Note: TWI Module as I2C Master
- Application Note: TWI Module as I2C Slave
- alternative Implementierung von Elia Ritterbusch

Übersicht über die am häufigsten verwendeten, seriellen Schnittstellen

USART

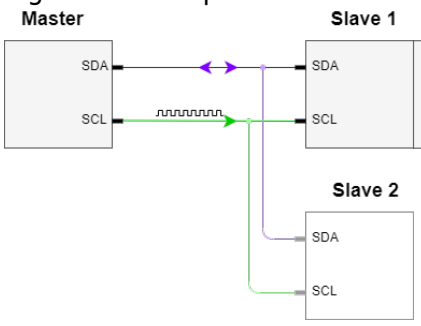
Fig. ##: Konzeptbild U(S)ART



- Keine gibt Takt vor. Es sind gleichberechtigte Kommunikationspartner (siehe figure ##).
- Jeder darf zu jederzeit senden.
- Senden und Empfangen geschieht über zwei

I2C

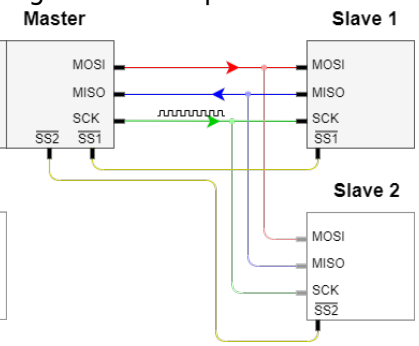
Fig. ##: Konzeptbild I2C



- Master gibt Takt vor (siehe figure ##).
- Slave darf nur zu bestimmten Zeiten senden und nur, wenn der Master dies anfordert.
- Senden und Empfangen geschieht über die

SPI

Fig. ##: Konzeptbild SPI



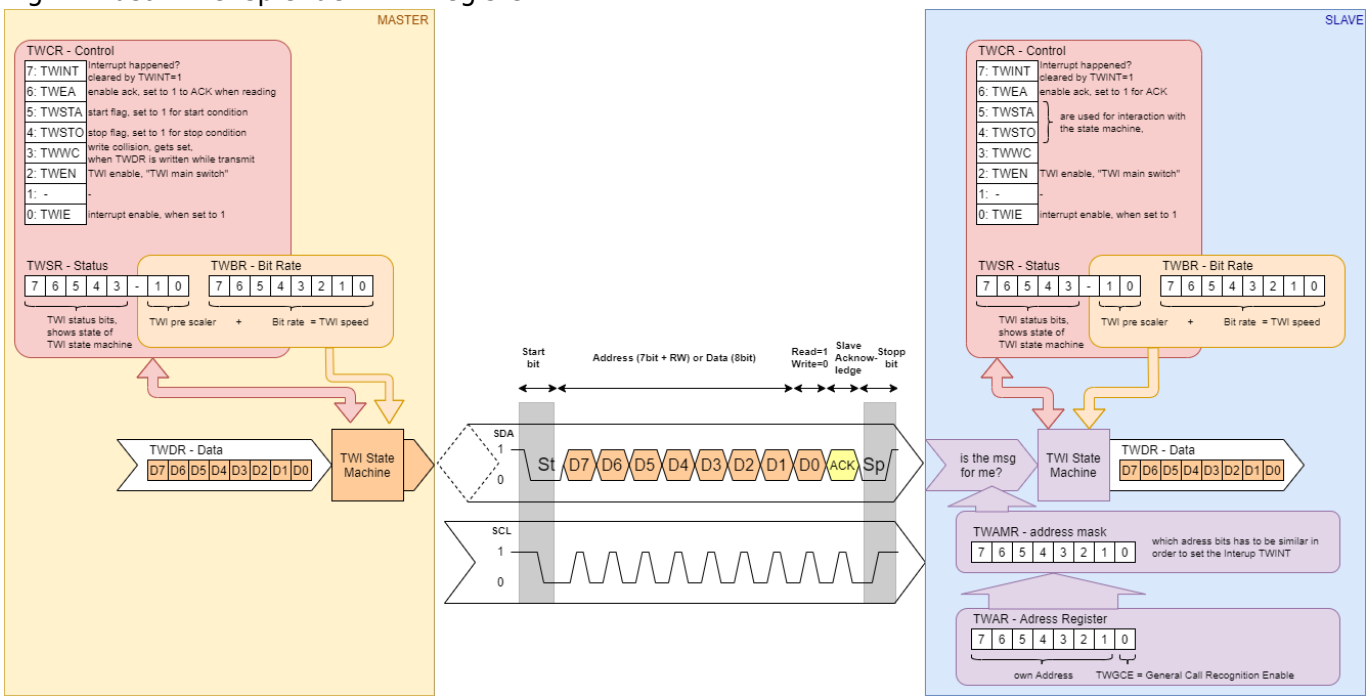
- Master gibt Takt vor (siehe figure ##).
- Slave darf nur zu bestimmten Zeiten senden und nur, wenn der Master dies anfordert.
- Senden und Empfangen

separate Leitungen.
- Kommunikation ist nur zwischen zwei Geräten möglich.
  Ein weiterer Slave würde eine weiteren U(S)ART-Bus benötigen.

gleiche Leitung.
- Alle Slaves hören am gleichen Bus mit und schreiben auf die gleiche Leitung.
- Jeder Slave muss anhand der Signale überprüfen, ob die Daten für ihn gemeint sind.

geschieht über zwei separate Leitungen.
- Alle Slaves hören auf der gleichen Leitung mit und schreiben auf die gleiche Leitung.
- Der gewünschte Slave wird über die Slave Select Leitung ausgewählt.

Fig. 2: Zusammenspiel der TWI-Register

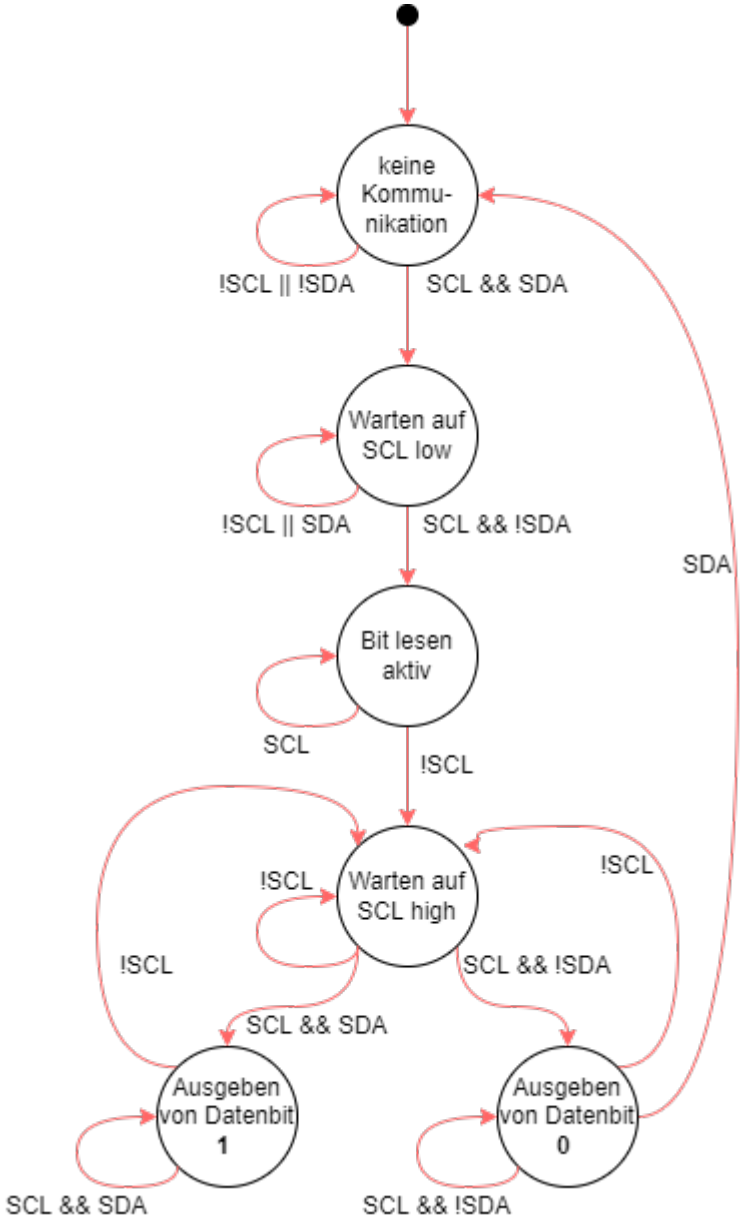


Statemachine für Datenpaket

Statemachine der I2C Kommunikation

Statemachine der I2C Kommunikation

Fig. 1: Statemachine der I2C Kommunikation

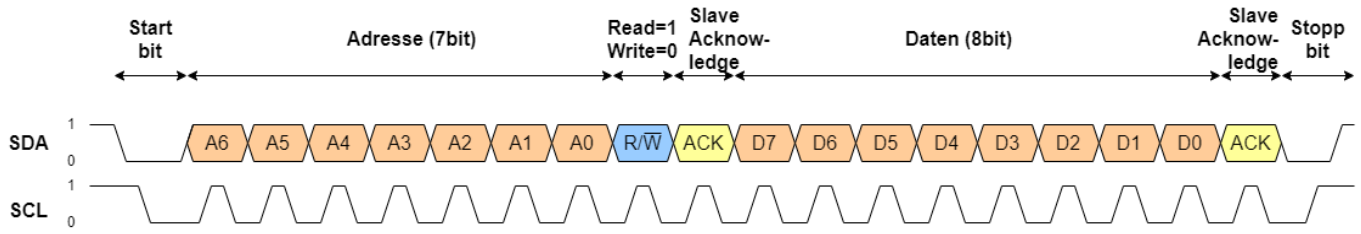


I2C in Kürze und Zeitverlaufsdiagramm

Übertragung

Für die I2C Übertragung “trommelt” der Master-IC auf der Taktleitung (SCL). Bei jedem “Trommelschlag” (SCL=High), darf der Slave die Datenleitung (SDA) lesen.
D.h. während der Datenübertragung bleibt die Datenleitung bei SCL=High konstant.
Eine Flanke (=Signalwechsel) auf der Datenleitung während SCL=High definiert Beginn und Ende der Kommunikation.
Eine fallende Flanke auf SDA bei SCL=High stellt das Startbit dar, eine steigende Flanke das Stoppbit.
Läuft keine Kommunikation sind Daten- und Tankleitung auf High.

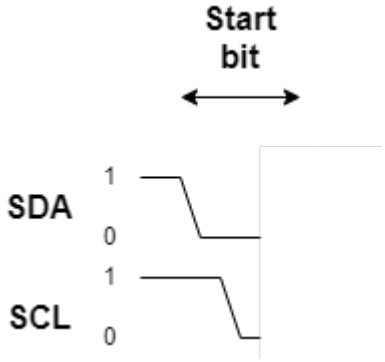
Fig. 3: Zeitverlaufsdiagramm der I2C Kommunikation



Startbedingung

Um die Übertragung zu beginnen muss die Startbedingung eingeleitet werden. Während SCL HIGH ist (a), geht SDA von HIGH auf LOW. Anschließend startet SCL mit LOW (b).

Fig. 4: Startbedingung



Für eine Startbedingung werden die Bits innerhalb des TWCR wie folgt gesetzt:

```
TWCR = (1<<TWINT)|(1<<TWEN);       // Setting
TWINT clears interupt flag
                                   // to set the
following state:
     | (1<<TWIE )                  // Enable TWI
Interrupt.
     | (1<<TWSTA)|(0<<TWSTO);      // Initiate a
START condition.
```

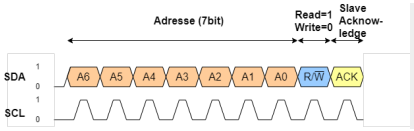
Übertragung

Die entscheidende Voraussetzung für eine erfolgreiche Bitübertragung ist, dass sich der Zustand von SDA nur ändern darf solange SCL auf LOW ist. Allerdings ist der Zustand von SDA erst gültig, wenn SCL auf HIGH ist.

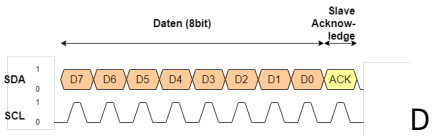
Fig. 5: Übertragung

Für die Übertragung eines Bytes muss TWDR und TWCR wie folgt gesetzt werden.
Zunächst wird die Übertragung der Adresse (SLA_W) betrachtet:

```
TWDR = SLA_W;                      // Load SLA_W
into TWDR
TWCR = (1<<TWINT)|(1<<TWEN);       // Setting
```




```
TWINT clears interupt flag
                                        // to start
transmission of address
```



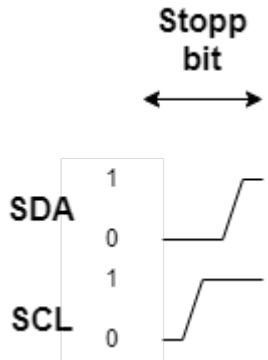
Die Daten (DATA) werden in gleicher Weise übertragen:

```
TWDR = DATA;                            // Load DATA
into TWDR
TWCR = (1<<TWINT)|(1<<TWEN);            // Setting
TWINT clears interupt flag
                                        // to start
transmission of address
```

Stoppbedingung

Die Stoppbedingung beendet die Übertragung. SCL geht auf HIGH (c), anschließend wechselt die SDA-Leitung von LOW nach HIGH (d).

Fig. 6: Stoppbedingung



Für eine Stoppbedingung werden die Bits innerhalb des TWCR wie folgt gesetzt:

```
TWCR = (1<<TWINT)|(1<<TWEN);            // Setting
TWINT clears interupt flag
                                        // to set the
following state:
     | (1<<TWIE )                       // Enable TWI
Interrupt.
     | (0<<TWSTA)|(1<<TWSTO);           // Initiate a
STOP condition.
```

Software

TWI Master

i2c_master.hex

```
/* ------------------------------------------------------------------------
---

 Experiment 10:   I2C Kommunikation
 =============    ==============================

 Dateiname  : I2C_SimpleMaster.c
 Autoren    : Tim Fischer       (Hochschule Heilbronn, Fakultaet T1)
 Datum      : 23.06.2021
 Version    : 1.0
 Hardware:  Simulide 0.5.16-RC5

 Software:  Entwicklungsumgebung: AtmelStudio 7.0
            C-Compiler: AVR/GNU C Compiler 5.4.0

 Funktion : TBD

 Displayanzeige:    TBD

 Tastenfunktion:    keine

 Jumperstellung:    keine

 Fuses im uC:       CKDIV8: Aus (keine generelle Vorteilung des Takts)

 Header-Files:  lcd_lib_de.h (Library zur Ansteuerung LCD-Display Ver.1.3)

// ------------------------------------------------------------------------
---*/

// Deklarationen
// ============================================================

// Festlegung der Quarzfrequenz
#define F_CPU 8000000UL // CPU Frequenz von 8MHz
#define F_SCL 40000L    // Baudrate von 100 kHz

// Include von Header-Dateien
#include <stdio.h>
#include <avr/interrupt.h>
#include <math.h>
#include <util/delay.h>

// Konstanten
#define SET_BIT(PORT, BIT)  ((PORT) |=  (1 << (BIT))) // Port-Bit Zustand
setzen
#define CLR_BIT(PORT, BIT)  ((PORT) &= ~(1 << (BIT))) // Port-Bit Zustand
loeschen
#define TGL_BIT(PORT, BIT)  ((PORT) ^=  (1 << (BIT))) // Port-Bit Zustand
wechseln (toggle)
```

```
//Funktionsprototypen
void I2C_Init();
void I2C_transmitStart();
void I2C_transmitDataOrAddress(char Data);
void I2C_transmitStop();
void I2C_Data();

uint8_t TWI_Address = 0b0001010;
uint8_t TWI_Data    = 0b00110111;


int main(void)
{
    cli ();                         // Interrupt aktivieren
    SET_BIT(DDRD, PORTD1);        // Debug Ausgang ansprechen -> Wechsel auf
low
    TGL_BIT(PORTD, PORTD1);        // Debug Ausgang ansprechen -> Wechsel
auf High
    I2C_Init();                     // Initialisierung von TWI anstoßen


    while (1)
    {
        TGL_BIT(PORTD, PORTD1); // Debug Ausgang ansprechen -> Wechsel auf
High
        I2C_Init();                 // Initialisierung von TWI anstoßen
        TGL_BIT(PORTD, PORTD1); // Debug Ausgang ansprechen -> Wechsel auf
Low
        I2C_transmitStart();    // Startbit schreiben
        TGL_BIT(PORTD, PORTD1); // Debug Ausgang ansprechen -> Wechsel auf
High
        I2C_transmitDataOrAddress((TWI_Address<<1) + 0);// Adresse senden
        TGL_BIT(PORTD, PORTD1); // Debug Ausgang ansprechen -> Wechsel auf
Low
        I2C_transmitDataOrAddress(TWI_Data);// Daten senden
        TGL_BIT(PORTD, PORTD1); // Debug Ausgang ansprechen -> Wechsel auf
High
        I2C_transmitStop();        // Stoppbit schreiben
        TGL_BIT(PORTD, PORTD1); // Debug Ausgang ansprechen -> Wechsel auf
Low
        _delay_us(1);
    }
}

////////////////////////////////////////
// I2C Initialisierung
////////////////////////////////////////
void I2C_Init()
{
    TWSR = CLR_BIT(TWSR, TWPS0);// Es wird kein Prescaler verwendet
    TWSR = CLR_BIT(TWSR, TWPS1);//
```

```
    TWCR = 0;                       //
    TWBR = ((F_CPU/F_SCL)-16)/2;// die Bitrate wird mittels CPU Frequenz und
Serial Clock Frequenz ermittelt
}

/////////////////////////////////////////
// I2C Startbit senden
/////////////////////////////////////////
void I2C_transmitStart()
{
    TWCR = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT); // TWSTA = Startbit aktivieren,
TWEN = TWI starten (ENable), TWINT = Interrupt bit löschen (durch setzen)
    while (!(TWCR & (1<<TWINT))){};            // warten bis Übertragung
erfolgreich, Trigger ist hier das Setzen von TWINT
}


/////////////////////////////////////////
// I2C Adressbyte/Daten senden
/////////////////////////////////////////
void I2C_transmitDataOrAddress(char Data)
{
    TWDR = Data;
    TWCR = (1<<TWINT)|(1<<TWEN);            // TWEN = TWI starten (ENable),
TWINT = Interrupt bit löschen (durch setzen)
    while (!(TWCR & (1<<TWINT)));            // warten bis Übertragung
erfolgreich, Trigger ist hier das Setzen von TWINT
}

/////////////////////////////////////////
// I2C Stoppbit senden
/////////////////////////////////////////
void I2C_transmitStop()
{
    TWCR=(1<<TWSTO)|(1<<TWINT)|(1<<TWEN);    // TWSTO = Stopptbit
aktivieren, TWEN = TWI starten (ENable), TWINT = Interrupt bit löschen
(durch setzen)
    while (!(TWCR & (1<<TWINT)));            // warten bis Übertragung
erfolgreich, Trigger ist hier das Setzen von TWINT
    //    while(TWCR & (1<<TWSTO));                // warten bis Übertragung
erfolgreich, Trigger ist hier das setzen von TWINT
}
```

TWI Slave

```
/* ------------------------------------------------------------------------
---
```

```
 Experiment 10:   I2C Kommunikation
 =============    ==============================

 Dateiname  : I2C_SimpleMaster.c

 Autoren    : Tim Fischer       (Hochschule Heilbronn, Fakultaet T1)

 Datum      : 23.06.2021

 Version    : 1.0
 Hardware:  Simulide 0.5.16-RC5

 Software:  Entwicklungsumgebung: AtmelStudio 7.0
            C-Compiler: AVR/GNU C Compiler 5.4.0

 Funktion:  tbd


// ------------------------------------------------------------------------
---*/

// Deklarationen
// ============================================================

// Festlegung der Quarzfrequenz
#define F_CPU 16000000UL
#define F_SCL 10000L //100 kHz

// Include von Header-Dateien
//#include <stdio.h>
#include <avr/interrupt.h>
//#include <math.h>
//#include <util/delay.h>

// Konstanten
#define SET_BIT(PORT, BIT)  ((PORT) |=  (1 << (BIT))) // Port-Bit Zustand
setzen
#define CLR_BIT(PORT, BIT)  ((PORT) &= ~(1 << (BIT))) // Port-Bit Zustand
loeschen
#define TGL_BIT(PORT, BIT)  ((PORT) ^=  (1 << (BIT))) // Port-Bit Zustand
wechseln (toggle)

/// Port for the I2C
#define I2C_DDR DDRD
#define I2C_PIN PIND
#define I2C_PORT PORTD

// Pins to be used in the bit banging
#define I2C_SCL 0
#define I2C_SDA 1
```

```
#define RISING_EDGE     1
#define FALLING_EDGE    0
#define TRUE            1
#define FALSE           0
#define BITS_IN_BYTE    8

// Used PC interrupts
#define I2C_PCINT_SCL PCINT16
#define I2C_PCINT_SDA PCINT17

//Funktionsprototypen
void I2C_Init();
void I2C_setAddress(char Address);
char I2C_readData();

uint8_t TWI_Address         =  0b0001010;
uint8_t TWI_AddressMask     = 0b11111110;

int main(void)
{
    DDRC= 0xFF;                                 // Auf DDRC die Daten
ausgeben
    I2C_setAddress(TWI_Address);                // eigene Adresse setzen
    while (1)
    {
        PORTC = I2C_readData();                 // Daten an PortC ausgeben
    }
}

////////////////////////////////////////////////
// Setzen der I2C Adresse auf die der Slave hört
////////////////////////////////////////////////
void I2C_setAddress(char Address)
{
    TWAR = (Address<<1);                                // Adresse in das
Pseudoregister schreiben
    TWAMR= TWI_AddressMask;                               // Adressmaske in
das Pseudoregister schreiben
    TWCR = (1<<TWEA)|(1<<TWEN)|(1<<TWIE);               // Enable Ack,
Enable Interupt und Enable TIW

}

////////////////////////////////////////////////
// Auslesen der übermittelten Daten
////////////////////////////////////////////////
char I2C_readData()
{
    while (!(TWCR & (1<<TWINT)));                       // warte solange
bis TWINT gesetzt ist
    TWCR = (1<<TWSTO)|(1<<TWEA)|(1<<TWINT);               // Resetieren der
```

```
State Machine, nicht zwingend notwendig
    return TWDR;                                        // übermittle Daten
}
```

Beispiele

- Simulide: `…\share\simulide\examples\Arduino\sofware_i2c_lcd\i2c_lcd-arduino` (hierbei wird Software I2C eingesetzt)
- Software I2C:
  - Library von Peter Fleury: library, Dokumentation
  - Software I2C Slave