

# software\_i2c\_slave

## Student Group

First Name	Surname	Matrikel Nr.

## Table of Contents

```

/* -----
---

Experiment 10b:   I2C Kommunikation - Software I2C für slave
=====

Dateiname: softTWI_slave.c
Autoren   : Tim Fischer      (Hochschule Heilbronn, Fakultät T1)
Datum     : 20.06.2021
Version   : 0.1
Hardware  : Simulide 0.5.16-RC5

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

Funktion:  Dient als Emulation eines TWI Slaves.

Features:
- Nachbildung des Daten-, Adress- und Adressmaskierungsregisters
- Im Kontrollregister sind nur TWEA, TWEN und TWINT umgesetzt
- ACK wird bei Wunsch gesendet
- bis 40kHz bei 8MHz Takt in Simulide getestet (jedoch ohne weitere
Funktionen)

To Do:
- bit counting über "bitPos" besser umsetzen
- Variable IO-Pins umsetzen
- TWSR-umsetzen
- header-file erstellen
- für längere Datenübertragung überprüfen

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

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

// 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 Address          = 0b0001010;
uint8_t AddressMask     = 0b11111110;

// Interne Variablen für Soft I2C
uint8_t bitPos          = BITS_IN_BYTE + 1; // Startbit wird auch als bit
erkannt, deswegen +1
uint8_t receivedData    = 0;                // mit leerer TWDR-Variable
beginnen
uint8_t receivedSda     = TRUE;             // SDA wird anfangs auf high
sein
uint8_t receivedSda_old = FALSE;           // Vorgängerwert zu SDA
uint8_t receivedScl     = TRUE;            // SCL wird anfangsauf high
sein
uint8_t receivedScl_old = TRUE;            // Vorgängerwert zu SCL
uint8_t isAddressByte   = TRUE;           // erstes gelesenes Byte ist
Adressbyte
uint8_t isCorrectAddress= FALSE;          // anfangs ist nicht kla, ob die
an den Slave mit Adresse TWAR gesendet wurde
uint8_t isAcknowleging  = FALSE;          // SDA zunächst nicht auf Low
ziehen
```

```

uint8_t TWCR_soft      = 0;          // Variable als Kontrollregister
Ersatz
uint8_t TWDR_soft      = 0;          // Variable als Datenregister
Ersatz
uint8_t TWAR_soft      = 0;          // Variable als Adressregister
Ersatz
uint8_t TWAMR_soft     = 0;          // Variable als
Adressmaskenregister Ersatz

int main(void)
{
    DDRC= 0xFF;                // Auf DDRC die Daten
ausgeben
    I2C_Init();                // soft TWI initialisieren

    I2C_setAddress(Address);    // eigene Adresse setzen
    while (1)
    {
        PORTC = I2C_readData();  // Daten an PortC ausgeben
    }
}

////////////////////////////////////
// I2C Initialisierung
////////////////////////////////////
void I2C_Init()
{
    SET_BIT(PCICR , PCIE2);      // Interrupt, wenn PC an Port
D wie über PCMSK2 gegeben
    SET_BIT(PCMSK2, I2C_PCINT_SCL); // Interrupt, wenn PC an SCL
Pin
    SET_BIT(PCMSK2, I2C_PCINT_SDA); // Interrupt, wenn PC an SDA
Pin
    sei();
}

////////////////////////////////////
// Pin Change Interupt Handler
////////////////////////////////////
ISR(PCINT2_vect)
{
    if(!(TWCR_soft && TWEN)) {return;}; // Falls
nicht Enabled, dann abbrechen
    receivedScl = (!(PIND & (1<<PIND0))==0); // SCL und
SDA als Bitwert einlesen
    receivedSda = (!(PIND & (1<<PIND1))==0);

    if (receivedScl == receivedScl_old) // wenn
eine Änderung in SDA auftritt
    {
        if((receivedScl==TRUE)) // und

```

```

SCL auf High liegt, dann ist an dieser Position ein Start oder Stopbit
    {
        receivedData = 0; // Daten
zurücksetzen
        isAddressByte = TRUE; // nächstes
übertragenes Byte ist Addressbyte
        TGL_BIT(PORTB, PORTB3); // DEBUG
        bitPos = BITS_IN_BYTE+1; // Stopp ->
Start nicht als Bit zählen (Workaround, die Anzahl der Bits bis zum ACK
korrekt zu ermitteln)
    };

    } else if(receivedSda == receivedSda_old) { // wenn
SDA sich über SCL=high nicht geändert hat
        if (receivedScl == FALLING_EDGE) { // und eine
Steigende Flanke auf SLC anliegt
            if (isAcknowleging){ // wenn noch
ACK anliegt, dann
                CLR_BIT(I2C_DDR , I2C_SDA); //
ACK extern wieder aufheben (war einen Takt angelegen)
                isAcknowleging = FALSE; // ACK
auc intern aufheben
                bitPos = BITS_IN_BYTE+1; //
Workaround, die Anzahl der Bits bis zum nächsten ACK korrekt zu ermitteln
            }
            receivedData = (receivedData <<1) + receivedSda_old;// neues
Datenbit ans Datenbyte anreihen
            bitPos--;
        };
    };
    if (bitPos ==0) // wenn
8bits eingelesen
    {
        bitPos = BITS_IN_BYTE; // Anzahl
zurücksetzen
        TWDR_soft = receivedData; //
empfangene Daten ins Pseudoregister

        if (TWCR_soft & (1<<TWEA)) // wenn
Acknowledge Enable
        {
            SET_BIT(I2C_DDR, I2C_SDA); // für
ACK SDA auf low ziehen
            isAcknowleging = TRUE; //
internes Flag setzen
        };
        if (isAddressByte) // falls
es sich um das Adressbyte gehandelt hat
        {
            isCorrectAddress = ((TWDR_soft & TWAMR_soft)>>1) == ((TWAR_soft
& TWAMR_soft) >>1); // Flag für Übereinstimmung von ermittelten und

```

```

gegebener Adresse in den maskierten Bits berechnen
    isAddressByte = FALSE; //
Flag für Adressbyte zurücksetzen
    }
    else if (isCorrectAddress) // falls
es sich um die korrekte Adresse handelt
    {
        isCorrectAddress= FALSE; // Flag
für korrekte Adresse zurücksetzen
        TWCR_soft = TWCR_soft |(1<<TWINT); //
Interuptbit in Pseudoregister setzen
    };
};
receivedSda_old = receivedSda; // alte Werte
für das nächste mal merken
receivedScl_old = receivedScl;
}

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

// sei();
}

////////////////////////////////////
// Auslesen der übermittelten Daten
////////////////////////////////////
char I2C_readData()
{
    while (!(TWCR_soft & (1<<TWINT))); // warte
solange bis TWINT gesetzt ist
    CLR_BIT(TWCR_soft, TWINT); // lösche
TWINT (nur im Soft TWI notwendig)
    return TWDR_soft; // übermittle
Daten
}

```

From:  
<https://first.mexle.te.hs-heilbronn.de/> - **MEXLE Wiki**

Permanent link:  
[https://first.mexle.te.hs-heilbronn.de/microcontrollertechnik/software\\_i2c\\_slave?rev=1624225822](https://first.mexle.te.hs-heilbronn.de/microcontrollertechnik/software_i2c_slave?rev=1624225822)

Last update: **2021/06/20 23:50**

