

## 7 Beispiele

Der folgende Programmcode basiert auf einem ATmega8 der Firma Atmel. Das Beispiel setzt die Standardbeschaltung (siehe Kapitel 3.6.2, Standardbeschaltung) voraus, d.h. das am Voice-Emitter II eine Versorgungsspannung anliegt, ein Lautsprecher angeschlossen wurde und das der TWI-Bus mit dem steuernden Mikrocontroller verbunden ist.

Der nachfolgende Programmcode beschreibt, wie der Voice-Emitter II von einem Mikrocontroller angesprochen werden kann.

Zuerst werden jeweils Funktionen für die Ansteuerung des TWI-Busses gezeigt. Diese Funktionen sind als Beispiel zu verstehen. Sie können hinsichtlich Robustheit und Effizienz verbessert werden (z.B. durch Timeout bei Schleifen, Verwendung von Interrupts, etc).

### 7.1 Beispiele in C

Die Kommunikation über den TWI-Bus erfolgt immer nach dem Schema

- Startkondition senden
- Daten senden oder empfangen
- Stopkondition senden

Wenn gelesen werden soll, muss das Bit 0 der TWI-Adresse 1 sein, sonst 0.

Nach dem Lesen eines Bytes muss der Empfänger mit ACK antworten, sofern er ein weiteres Byte erwartet, sonst Antwortet er mit NACK.

Die folgenden vier Funktionen stellen die Basis für den Zugriff auf den TWI-Bus dar.

```
#define F_CPU 1000000L // CPU-Frequenz
#define F_SCL 10000L // TWI-Frequenz

#include <avr\io.h>
#include <util\twi.h>
#include <util\delay.h>
#include <string.h>

typedef uint8_t BYTE;

// initialisiert den Timer für den TWI-Bus
void twiInit()
{
    TWSR = 0; // kein prescaler
    TWBR = (BYTE) ((F_CPU/F_SCL)-16)/2; // TWI-Speed
}

/* -----
Start condition senden
adr: Adresse des Slave

Rückgabe: 1 -> ok
          0 -> Fehler
----- */
BYTE twiStart(BYTE adr)
{
    // Start condition senden
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

    // warten bis µC fertig
    while(!(TWCR & (1<<TWINT)));
}
```

```

// Abbruch bei Fehler
BYTE twiStatus = TW_STATUS & 0xF8;
if ( (twiStatus != TW_START) && (twiStatus != TW_REP_START) )
    return 0;

// Geräteadresse senden
TWDR = adr;
TWCR = (1<<TWINT) | (1<<TWEN);

// warten bis µC fertig und ACK/NACK empfangen wurde
while(!(TWCR & (1<<TWINT)));

// Abbruch bei Fehler
twiStatus = TW_STATUS & 0xF8;
if ( (twiStatus != TW_MT_SLA_ACK) && (twiStatus != TW_MR_SLA_ACK) )
    return 0;

return 1;
}

/* -----
   Stop condition senden
   ----- */
void twiStop()
{
    // Stop condition senden
    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

    // Warten bis Senden abgeschlossen ist
    while(TWCR & (1<<TWSTO));
}

/* -----
   Datenbyte senden
   data: zu sendendes Datenbyte

   Rückgabe: 1 -> ok
              0 -> Fehler
   ----- */
BYTE twiWrite(BYTE data)
{
    // Datenbyte versenden
    TWDR = data;
    TWCR = (1<<TWINT) | (1<<TWEN);

    // Warten, bis µC fertig
    while(!(TWCR & (1<<TWINT)));

    // Status abfragen, Fehler wenn Slave nicht mit ACK geantwortet hat
    if( (TW_STATUS & 0xF8) != TW_MT_DATA_ACK)
        return 0;
    return 1;
}

/* -----
   Datenbyte lesen und mit ACK beantworten

   Rückgabe: empfangenes Datenbyte
   ----- */
BYTE twiReadACK()
{
    // Daten empfangen, anschließend ACK senden
    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    // Warten bis Datenbyte empfangen wurde
    while(!(TWCR & (1<<TWINT)));

    // Datenbyte auslesen
    return TWDR;
}

```

```

/* -----
   Datenbyte lesen und mit NACK beantworten

   Rückgabe: empfangenes Datenbyte
   ----- */
BYTE twiReadNACK()
{
    // Daten empfangen, anschließend NACK senden
    TWCR = (1<<TWINT) | (1<<TWEN);
    // Warten bis Datenbyte empfangen wurde
    while(!(TWCR & (1<<TWINT)));

    // Datenbyte auslesen
    return TWDR;
}

```

Das Senden von Befehlen an den Voice-Emitter II kann mit den TWI-Basis-Funktionen wie folgt durchgeführt werden.

```

/* -----
   Befehl an den VoiceEmitter senden
   bAdr:      TWI-Adresse des VoiceEmitter
   pBuf:      Zeiger auf den zu sendenden Datenbereich
   bSize:     Anzahl zu sendender Bytes

   Rückgabe: 1 -> ok
              0 -> Fehler
   ----- */
BYTE ve_sendCmd(BYTE bAdr, BYTE* pBuf, BYTE bSize)
{
    // start condition senden, schreibender Zugriff,
    // daher Bit 0 der Adresse auf 0 setzen
    if (!twiStart(bAdr & 0xfe)) return 0;

    // Pufferinhalt übertragen
    while (bSize > 0)
    {
        // ein Byte senden
        if (!twiWrite(*pBuf) )
        {
            // Stop condition bei Fehler
            twiStop();
            return 0;
        }

        bSize--;
        pBuf++;
    }

    // Befehl abschließen
    twiStop();

    return 1;
}

```

Analog dazu werden Daten vom Voice-Emitter II mit den TWI-Basis-Funktionen durchgeführt. Die Besonderheit beim Voice-Emitter II ist, das er immer zuerst die Anzahl noch folgender Bytes mitteilt.

```

/* -----
   Daten vom VoiceEmitter abholen
   bAdr:      TWI-Adresse des VoiceEmitter
   pBuf:      Zeiger auf den zu sendenden Datenbereich
               Achtung: Der Puffer muss 17 Bytes groß sein

   Rückgabe: Anzahl gelesener Bytes

```

```

----- */
BYTE ve_getData(BYTE bAdr, BYTE* pBuf)
{
    // start condition senden, lesender Zugriff,
    // daher Bit 0 der Adresse auf 1 setzen
    if (!twiStart(bAdr | 0x01))
    {
        twiStop();
        return 0;
    }

    // 1. Byte = Anzahl noch folgender Bytes
    BYTE bBytes = twiReadACK();

    BYTE bCnt = bBytes;
    while (bCnt > 0)
    {
        if (bCnt > 1)
            // Byte lesen und mit ACK beantworten
            *pBuf = twiReadACK();
        else
            // letztes Byte lesen und mit NACK beantworten
            *pBuf = twiReadNACK();

        bCnt--;
        pBuf++;
    }

    twiStop();

    return bBytes;
}

```

Daten, die der Voice-Emitter II sendet, beginnen immer mit der Anzahl Folgebytes, einem Statusbyte und einer Fehlernummer. Danach folgen, sofern vorhanden, die eigentlichen Nutzdaten wie z.B. Dateiinhalte oder abgefragte Systemparameter.

Wenn der Voice-Emitter II keine Nutzdaten zur Verfügung stellt, bedeutet das nicht, das im Laufe der Befehlsabarbeitung keine weiteren Daten anfallen. Um die Daten vollständig abzuholen, muss das Flag BUSY im Status-Byte abgefragt werden. Erst wenn der Voice-Emitter II nicht mehr BUSY ist und keine Daten mehr vorhanden sind, sind alle Daten übertragen worden.

Die folgende Funktion fragt den Voice-Emitter II nach Daten ab und blendet Status- und Fehlerbyte aus. Sind keine Daten vorhanden, dann wartet sie ab, bis der Voice-Emitter II nicht mehr BUSY ist.

Die Funktion muss solange aufgerufen werden, bis sie keine Daten mehr liefert.

```

/* -----
Nutzdaten vom VoiceEmitter abholen, wartet
falls noch Daten gesendet werden könnten
bAdr:     TWI-Adresse des VoiceEmitter
pBuf:     Zeiger auf den zu sendenden Datenbereich
          Achtung: Der Puffer muss 17 Bytes groß sein

Rückgabe: Anzahl gelesener Bytes
          0: es folgen keine weiteren Daten
          >0: Anzahl Daten im Puffer, weitere Daten
             ggf. vorhanden
----- */
BYTE ve_getPayloadData(BYTE bAdr, BYTE* pBuf)
{
    // BUSY-Flag mit 1 initialisieren, damit while-Schleife
    // mindestens einmal durchlaufen wird
    BYTE bBusyFlag = 1;

    // Abfrage solange wiederholen, bis VE nicht mehr BUSY ist

```

```

while (bBusyFlag)
{
    // Daten vom VoiceEmitter abfragen
    BYTE bSize = ve_getData(bAdr, pBuf);

    // Der VoiceEmitter sendet immer drei Bytes:
    // Anzahl Folgebytes, Status und Fehlernummer
    // Status und Fehlernummer sind im Puffer abgelegt
    // Falls bSize < 2, dann Fehler bei der TWI-Kommunikation
    if (bSize < 2)
        return 0;

    // wenn Nutzdaten vorhanden, dann Puffer umkopieren,
    // d.h. die ersten beiden Bytes überschreiben
    if (bSize > 2)
    {
        for (BYTE bIdx = 0; bIdx < bSize-2; bIdx++)
        {
            *(pBuf+bIdx) = *(pBuf+bIdx+2);
        }

        // Länge der Nutzdaten korrigieren und zurückgeben
        return bSize-2;
    }

    // Status ist erstes Byte im Puffer
    BYTE bStatus = *pBuf;

    // BUSY-Flag ist Bit 0 des Statusbyte
    bBusyFlag = bStatus & 0x01;

    // Wenn BUSY, dann wird die Schleife wiederholt
    // Im Fall der Wiederholung etwas warten um keine unnötig große
    // Last beim VoiceEmitter durch ständige Abfragen zu erzeugen
    if (bBusyFlag)
        _delay_ms(100);
}

// VoiceEmitter ist nicht mehr BUSY, es sind keine Daten vorhanden
// und weitere Daten werden nicht mehr generiert

return 0;
}

```

Die Funktion `ve_getPayloadData` kann dazu verwendet werden, um zu warten, bis der Voice-Emitter II nicht mehr im Status BUSY ist.

```

/* -----
Warten, bis VoiceEmitter nicht mehr BUSY ist
bAdr:      TWI-Adresse des VoiceEmitter
----- */
void ve_waitForReady(BYTE bAdr)
{
    BYTE buffer[17];

    // in ve_getPayloadData werden Nutzdaten abgeholt und
    // so lange gewartet, bis das BUSY-Flag 0 ist
    while (ve_getPayloadData(bAdr, (BYTE*)&buffer) > 0);
}

```

Wenn nur die Fehlernummer des Voice-Emitter II abgefragt werden soll, ohne Nutzdaten zu empfangen, kann die folgende Funktion verwendet werden.

Vorsicht: wenn der interne Puffer des Voice-Emitter II voll ist, dann wartet er auf die Abholung der Daten und unterbricht seine Befehlsabarbeitung.

```

/* -----
Fehlernummer des VoiceEmitter abfragen
bAdr:      TWI-Adresse des VoiceEmitter
----- */

```

```

Rückgabe: Fehlernummer des VoiceEmitter
           0xff bei TWI-Kommunikationsfehler
----- */
BYTE ve_getError(BYTE bAdr)
{
    // start condition senden, lesender Zugriff,
    // daher Bit 0 der Adresse auf 1 setzen
    if (!twiStart(bAdr | 0x01))
    {
        twiStop();
        return 0xff;
    }

    // 1. Byte = Anzahl noch folgender Bytes
    BYTE bBytes = twiReadACK();
    // muss immer >= 2 sein, sonst TWI-Kommunikationsfehler
    if (bBytes < 2)
    {
        twiStop();
        return 0xff;
    }

    // 2. Byte = Status (wird hier nicht weiter verwendet)
    // BYTE bStatus =
    twiReadACK();

    // 3. Byte = Fehlernummer
    BYTE bError = twiReadNACK();

    // ggf. noch folgende Bytes werden hier nicht abgefragt
    // und müssen, sofern vorhanden, separat abgerufen werden

    twiStop();
    return bError;
}

```

Mit den zuvor gezeigten Funktionen kann der Voice-Emitter II gesteuert werden. Das Beispiel ist mit Kommentaren versehen, so dass es ohne weitere Bemerkungen verständlich sein sollte. Die gezeigten Beispiele verwenden die Dateien auf der mitgelieferten MicroSD-Karte.

```

#define VE_TWI_ADRESS 0x50

int main()
{
    /* -----
       TWI-Initialisieren und Start des VoiceEmitters abwarten
       ----- */
    // TWI-Timer initialisieren
    twiInit();

    // Start des VoiceEmitters abwarten
    _delay_ms(100);
    // Warten, bis VoiceEmitter ready
    ve_waitForReady(VE_TWI_ADRESS);

    /* -----
       Verzeichnis einstellen und einen Klang abspielen
       ----- */
    // Verzeichnis /demo einstellen (Befehl C)
    ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"CDEMO", strlen("CDEMO"));

    // Klang READY.WAV abspielen
    ve_waitForReady(VE_TWI_ADRESS);
    ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"PREADY.WAV", strlen("PREADY.WAV"));

    /* -----
       VoiceEmitter bis 4 zählen lassen
    */
}

```

```

----- */
for (BYTE idx = 1; idx <= 4; idx++)
{
    // Befehl P aufbereiten
    // Nutzung der Dateien ZD00?000.WAV
    char szCmd[20];
    strcpy((char*)&szCmd, "PZD001000.WAV");
    szCmd[5] = '0' + idx;

    ve_waitForReady(VE_TWI_ADRESS);
    // Befehl absenden
    ve_sendCmd(VE_TWI_ADRESS, (BYTE*)&szCmd, strlen((char*)&szCmd));
}

/* -----
Fehler provozieren und darauf reagieren
----- */
#define FATERR_FILE_NOT_FOUND 21

// Nicht vorhandene Datei abspielen lassen -> Fehler
ve_waitForReady(VE_TWI_ADRESS);
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"PXYZ.WAV", strlen("PXYZ.WAV"));

// Fehlermeldung abfragen
ve_waitForReady(VE_TWI_ADRESS);
BYTE bErr = ve_getError(VE_TWI_ADRESS);

// Im Fehlerfall einen Klang ausgeben
if (bErr == FATERR_FILE_NOT_FOUND)
    ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"PERR_21.WAV", strlen("PERR_21.WAV"));

/* -----
Klang abspielen, Abspielposition und Lautstärke ändern und
Abspielen abbrechen
----- */
// ASCII-Modus einstellen, damit Parameter als ASCII-Ziffern angegeben
// werden können
ve_waitForReady(VE_TWI_ADRESS);
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"A", 1);

// Klang zum Abspielen öffnen, jedoch noch nicht abspielen
ve_waitForReady(VE_TWI_ADRESS);
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"OTAKE5.WAV", strlen("OTAKE5.WAV"));

// Abspielposition auf 27 Sekunden einstellen
ve_waitForReady(VE_TWI_ADRESS);
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"S27s", sizeof("S27s"));

// Abspielen starten
ve_waitForReady(VE_TWI_ADRESS);
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"p", 1);

// 5 sekunden warten
_delay_ms(5000);

// Lautstärke auf 50% einstellen
// Der Befehl 'V' funktioniert auch im BUSY-Modus, daher muss hier
// nicht auf den VoiceEmitter gewartet werden
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"V50%", sizeof("V50%"));

// 5 sekunden warten
_delay_ms(5000);

// Abspielposition auf 12 Sekunden einstellen
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"S12s", sizeof("S12s"));

// 5 sekunden warten
_delay_ms(5000);

// Lautstärke auf 90% einstellen,
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"V90%", sizeof("V90%"));

// 10 sekunden warten

```

```
_delay_ms(10000);

// Abspielen abbrechen
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"X1", sizeof("X1"));

/* -----
   Die ersten 50 Zeichen aus einer Datei abfragen und, sofern es
   Ziffern sind, vorlesen
   ----- */

// Binär-Modus einstellen
ve_waitForReady(VE_TWI_ADRESS);
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"B", 1);

// Die ersten 50 Zeichen der Datei DIGITS.TXT abfragen
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"DDIGITS.TXT", strlen("DDIGITS.TXT"));

BYTE buffer[50];
BYTE bBufferIdx = 0;
while (bBufferIdx < 50)
{
    BYTE tmpBuffer[17];
    // Daten abfragen
    BYTE bLen = ve_getPayloadData(VE_TWI_ADRESS, (BYTE*)&tmpBuffer);

    if (bLen == 0)
        // Keine weiteren Daten vorhanden
        break;

    // Überlauf des Puffers verhindern
    if (bLen + bBufferIdx > 50)
        bLen = 50-bBufferIdx;

    // tmpBuffer in buffer umkopieren
    for (BYTE n = 0; n < bLen; n++)
    {
        buffer[bBufferIdx+n] = tmpBuffer[n];
    }

    bBufferIdx += bLen;
}

// Dateiausgabe abbrechen
ve_sendCmd(VE_TWI_ADRESS, (BYTE*)"X", strlen("X"));
ve_waitForReady(VE_TWI_ADRESS);

// Zeichen vorlesen, wenn es Ziffern sind
for (BYTE idx = 0; idx < bBufferIdx; idx++)
{
    if ( (buffer[idx] >= '0') && (buffer[idx] <= '9') )
    {
        // Befehl aufbereiten
        char szCmd[20];
        strcpy((char*)&szCmd, "PZD001000.WAV");
        szCmd[5] = buffer[idx];

        ve_waitForReady(VE_TWI_ADRESS);
        // Befehl absenden
        ve_sendCmd(VE_TWI_ADRESS, (BYTE*)&szCmd, strlen((char*)&szCmd));
    }
}

ve_waitForReady(VE_TWI_ADRESS);

while (1);

return 0;
}
```