# Mikrocontroller-Grundlagen Teil 19

***Anhand mehrerer Beispiele beschreiben wir ausführlich die Ansteuerung eines EEPROMS mit I²C-Schnittstelle.***

## 6.9 EEPROM am I²C-Bus

Der wohl am häufigsten eingesetzte Baustein mit I²C-Schnittstelle ist das EEPROM. Durch den relativ einfachen Schreib-/Lesezugriff und den Datenerhalt nach Abschalten der Versorgungsspannung ist es der ideale Datenspeicher für Einstellungen und Abgleichparameter, die damit dauerhaft zur Verfügung stehen. Die Standard-I²C-EEPROMs sind in einem 8poligen DIP-Gehäuse untergebracht und werden über eine 5V-Betriebsspannung versorgt. Die Erzeugung der Programmierspannung erfolgt intern und erfordert somit keine zusätzliche Spannungsversorgung.

Der Speicher der EEPROMs ist in 8-Bit-Worten organisiert, die in Bänken zu jeweils 256 Byte angeordnet sind. Eine Kurzübersicht über die marktgängigen EEPROMs zeigt Tabelle 25 („ELVjournal" 4/96, Seite 73). Für eine detaillierte Beschreibung der EEPROM-Funktionen sei auf den Artikel „I²C-EEPROM-Board" in dieser Ausgabe verwiesen.

## 6.10 Treiber für die I²C-Schnittstelle

Im Rahmen der Mikrocontroller-Grundlagen-Serie sind mehrere in Assembler erstellte Treiber für die I²C-Schnittstelle entwickelt worden, deren hierarchische Anordnung Abbildung 144 zeigt. Das gemeinsame Unterprogramm DELAY5 nimmt eine definierte Verzögerung von mindestens 5 ms vor. Darüber angeordnet sind die eigentlichen I²C-Zugriffsroutinen I2C_STRT, I2C_STOP, I2C_WBIT und I2C_RBIT, die die Start- und Stopp-Bedingungen sowie das Schreiben und Lesen eines Bits über den I²C-Bus steuern.

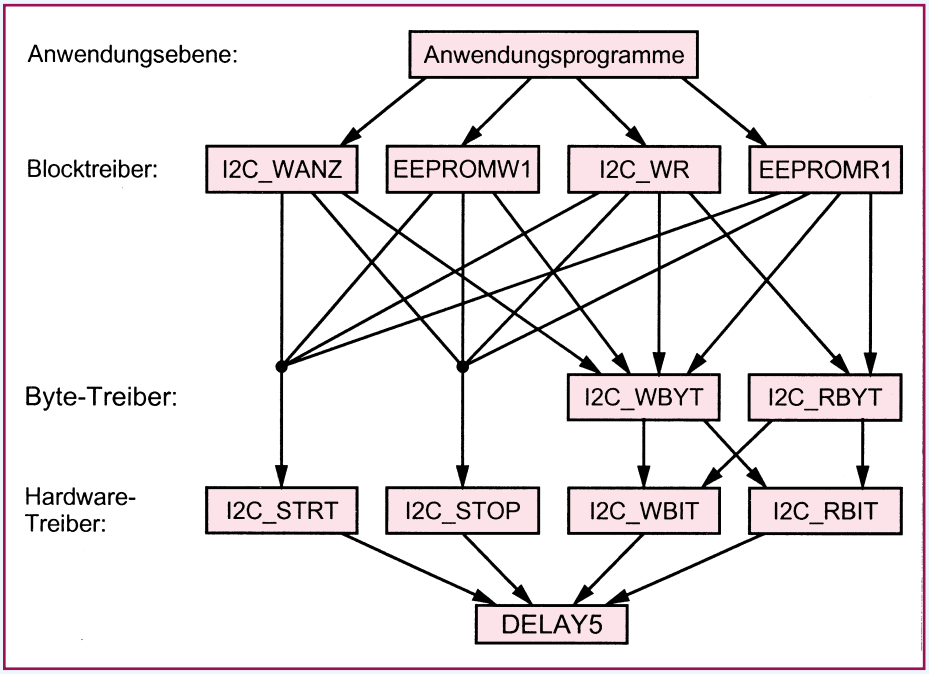


**Bild 144: Hierarchische Anordnung der I²C-Treiber**

Die in der nächst höheren Ebene angeordneten Routinen I2C_WBYT und I2C_RBYT schreiben bzw. lesen ein komplettes Byte vom I²C-Bus, wobei die darunter angeordneten Routinen entsprechend genutzt werden.

In der zweithöchsten Ebene befinden sich die Block-Zugriffroutinen I2C_WANZ, EEPROMW1, I2C_WR und EEPROMR1, die jeweils eine komplette I²C-Datenübertragung, beginnend mit der Generierung der Start-Bedingung und endend mit der Stopp-Bedingung, steuern. Eine detallierte Beschreibung der Routinen erfolgt in den folgenden Kapiteln.

Oben in Abbildung 144 sind die Anwendungsprogramme dargestellt, die widerum - je nach Bedarf - auf die darunter angeordneten Block-Treiber zugreifen.

Nachfolgend beschreiben wir die einzelnen Unterprogramme detailliert, die notwendig sind, um die Funktion der anschließend beschriebenen Anwendungsprogramme zu gewährleisten.

### 6.10.1 Hardware-Treiber

Abbildung 145 zeigt die allgemein notwendigen Definitionen und die Verzögerungsroutine DELAY5. Die oben dargestellten Definitionen legen die Port-Anschlußpins für die I²C-Daten- und Steuerleitung SDA und SCL fest. Die anschließende Zuordnung definiert die EEPROM-Basisadresse für den Zugriff auf ein am I²C-Bus angeschlossenes EEPROM.

Das Unterprogramm DELAY5 gewähr-

```
;******* Unterprogramme *********************************************
00B2          SDA          EQU   P3.2       ; SDA Anschluß
00B3          SCL          EQU   P3.3       ; SCL Anschluß

00A0          ADR_EEPR     EQU   10100000B; EEPROM Bank 0 Adresse Bit 0..3 müssen 0 sein
002E                       DELAY5:          ; Verzögerung um mind. 5 µsec für Q=11,059 MHz
                                            ; ACALL(LCALL) und RET benötigen bereits
                                            ; 2 +2 = 4 Maschinenzyklen (4,3 µsec)
002E 00                    NOP              ; Verzögerung ca. 1 µsec
002F 22                    RET              ; Rücksprung
```

**Bild 145: Allgemeine Definitionen und die Verzögerungsroutine DELAY5**

```
0030                 I2C_STRT:  ; Start-Bedingung des I2C Busses generieren
                                ; Zerstört:          * R7
                                ; Rückgabe:          * C gesetzt, wenn Fehler detektiert
0030 D2B3                       SETB   SCL      ; zunächst definierte Start-Bedingung erzeugen
0032 112E                       ACALL  DELAY5   ; Delay 5 µsec
0034 D2B2                       SETB   SDA      ; falls noch andere Pegel auf dem Bus
0036 112E                       ACALL  DELAY5   ; Delay 5 µsec
0038 7F0A                       MOV    R7, #10  ; * 10 Zyklen prüfen, ob der Bus frei ist
003A D3                         SETB   C        ; * zunächst Fehlerbedingung setzen
003B 30B30C          I2C_STW:   JNB    SCL, I2C_STE;* Springe, wenn SCL nicht high
003E 30B209                     JNB    SDA, I2C_STE;* Springe, wenn SDA nicht high
0041 DFF8                       DJNZ   R7, I2C_STW;* Schleife 10 mal durchlaufen
0043 C2B2                       CLR    SDA      ; Start-Bedingung zuerst SDA auf L-Pegel
0045 112E                       ACALL  DELAY5   ; Delay 5 µsec
0047 C2B3                       CLR    SCL      ; Start-Bedingung danach SCL auf L-Pegel
0049 C3                         CLR    C        ; kein Fehler
004A 22              I2C_STE:   RET              ; Rücksprung
```

**Bild 146: I2C_STRT-Unterprogramm**

```
004B          I2C_STOP:   ; Stopp-Bedingung des I2C-Busses generieren
                          ; Zerstört:          -
                          ; Rückgabe:          -
004B C2B2                 CLR    SDA      ; Vorbereitung
004D 112E                 ACALL  DELAY5   ; Delay 5 µsec
004F D2B3                 SETB   SCL      ; Stopp-Bedingung zuerst SCL auf H-Pegel
0051 112E                 ACALL  DELAY5   ; Delay 5 µsec
0053 D2B2                 SETB   SDA      ; Stopp-Bedingung danach SDA auf H-Pegel
0055 22                   RET             ; Rücksprung
```

**Bild 147: I2C_STOP-Unterprogramm**

```
0056          I2C_WBIT:   ; Ausgabe eines Bits und Taktgenerierung für den I2C-Bus
                          ; Eingang :          C Polarität des Datenbits
                          ; Zerstört:          * R7
                          ; Rückgabe:          * C gesetzt, wenn Fehler festgestellt
0056  92B2                MOV    SDA, C       ; Ausgabe des Datenbits
0058  112E                ACALL  DELAY5       ; Delay 5 µsec
005A  500C                JNC    I2C_W1       ; * Springe, wenn Bit low ist
005C  7F0A                MOV    R7, #10      ; * 10 Zyklen prüfen, ob der Bus frei ist
005E  20B207  I2C_WW1: JB SDA,   I2C_W1       ; * Springe, wenn SDA high
0061  DFFB                DJNZ   R7, I2C_WW1  ; * Schleife 10 mal durchlaufen
0063  D2B3                SETB   SCL          ; * Bus freigeben
0065  D3                  SETB   C            ; * Fehlerbedingung setzen
0066  8013                SJMP   I2C_WEND     ; * Sprung zum Ende
0068  D2B3    I2C_W1:     SETB   SCL          ; Taktleitung setzen
006A  112E                ACALL  DELAY5       ; Delay 5 µsec
006C  7F0A                MOV    R7, #10      ; * 10 Zyklen prüfen, ob der Bus frei ist
006E  20B307  I2C_WW2:    JB     SCL, I2C_W2  ; * Springe, wenn SCL high
0071  DFFB                DJNZ   R7, I2C_WW2  ; * Schleife 10 mal durchlaufen
0073  D2B2                SETB   SDA          ; * Datenbus freigeben
0075  D3                  SETB    C           ; * Fehlerbedingung setzen
0076  8003                SJMP   I2C_WEND     ; * Ende, da Bus nicht frei
0078  C2B3    I2C_W2:     CLR    SCL          ; Taktleitung zurücksetzen
007A  C3                  CLR    C            ; kein Fehler erkannt
007B  22      I2C_WEND:   RET                 ; Rücksprung
```

**Bild 148: Unterprogramm I2C_WBIT**

```
007C          I2C_RBIT:   ; Taktgenerierung und Lesen eines Bits vom I2C-Bus
                          ; Eingang :          -
                          ; Zerstört:          * R7
                          ; Rückgabe:          * C gesetzt, wenn Fehler festgestellt
                          ; F0 Inhalt des zurückgelesenen Datenbits
007C  D2B2                SETB   SDA          ; SDA setzen
007E  112E                ACALL  DELAY5       ; Delay 5 µsec
0080  D2B3                SETB   SCL          ; Taktleitung setzen
0082  112E                ACALL  DELAY5       ; Delay 5 µsec
0084  7F0A                MOV    R7, #10      ; * 10 Zyklen prüfen, ob der Bus frei ist
0086  20B305  I2C_RW:     JB     SCL, I2C_R1  ; * Springe, wenn SCL high
0089  DFFB                DJNZ   R7, I2C_RW   ; * Schleife 10 mal durchlaufen
008B  D3                  SETB   C            ; * Fehlerbedingung setzen
008C  8007                SJMP   I2C_REND     ; * Ende, da Bus nicht frei
008E  A2B2    I2C_R1:     MOV    C, SDA       ; Inhalt von SDA lesen
0090  92D5                MOV    F0, C        ; Inhalt speichern
0092  C2B3                CLR    SCL          ; Taktleitung zurücksetzen
0094  C3                  CLR    C            ; kein Fehler erkannt
0095  22      I2C_REND:   RET                 ; Rücksprung
```

**Bild 149: Unterprogramm I2C_RBIT**

leistet eine Verzögerung von mindestens 5 ms, um die Timing-Anforderung des I²C-Busses einhalten zu können. Da bereits der Aufruf dieser Routine durch ACALL bzw. LCALL und der abschließende RET-Befehl jeweils 2 Maschinenzyklen benötigen, ist hier nur ein NOP-Befehl erforderlich.

In Abbildung 146 ist das Unterprogramm I²C_STRT dargestellt, welches die Start-Bedingung für den I²C-Bus generiert. Zunächst wird durch das Setzen der SCL- und SDA-Steuerleitungen eine definierte Ausgangsbedingung erzeugt. Danach prüft die Routine über 10 Schleifendurchläufe, ob die beiden Steuerleitungen auch gesetzt bleiben. Diese Funktion ist für den Multimaster-Betrieb wichtig, da hier auch mehrere Master gleichzeitig auf den Bus zugreifen können, und somit überprüft werden muß, ob nicht gerade ein anderer Master den Bus benutzt. Für den Single-Master-Betrieb können die mit „*“ gekennzeichneten Zeilen entfallen. Nach der Überprüfung des Bus-Zustandes werden in einem Abstand von 5 ms die Steuerleitung SDA und SCL auf Low-Pegel gesetzt (Start-Bedingung).

Bei fehlerfreiem Durchlauf wird die Routine mit gelöschtem „C“-Flag verlassen, während im Fehlerfall das „C“-Flag gesetzt bleibt.

Abbildung 147 zeigt das Unterprogramm I2C_STOP, welches die Stopp-Bedingung für den I²C-Bus generiert. Vorbereitend wird zunächst die SDA-Datenleitung auf Low-Pegel gelegt und nach jeweils einer Verzögerung von 5 ms zunächst die SCL-Steuerleitung und anschließend die SDA-Leitung nacheinander auf High-Pegel gelegt (Stopp-Bedingung), womit auch gleichzeitig der Bus für weitere Aktivitäten freigegeben ist.

Abbildung 148 zeigt das Unterprogramm I2C_WBIT, welches 1 Bit über den I²C-Bus zum Slave überträgt. Zu Beginn des Unterprogramms wird der gewünschte Pegel des Data-Bit auf die SDA-Steuerleitung des I²C-Busses gegeben. Bei der Ausgabe eines High-Pegels wird in den folgenden Zeilen geprüft, ob die Datenleitung auch tatsächlich High-Pegel annimmt.

Nach dem anschließenden Setzen der Taktleitung SCL erfolgt ebenfalls eine Überprüfung, ob diese Steuerleitung tatsächlich High-Pegel angenommen hat. Die Überprüfungen der High-Pegel an der SDA- und SCL-Leitung sind nur im Multimaster-Betrieb erforderlich. Im Singlemaster-Betrieb können die mit „*“ gekennzeichneten Programmzeilen ersatzlos entfallen.

Zum Abschluß der Routine wird der Pegel der Taktleitung SCL wieder auf Low zurückgesetzt, womit die Ausgabe eines Bit abgeschlossen ist.

Das Einlesen eines Datenbit erfolgt über die in Abbildung 149 dargestellte Routine I2C_RBIT. Zunächst erfolgt das Setzen der SDA-Datenleitung auf High-Pegel, damit der Slave seine Information auf den Datenbus legen kann. Nach einer 5 ms langen Pause und dem Setzen der Taktleitung SCL erfolgt, wie bei dem Unterprogramm I2C_WBIT beschrieben, die Überprüfung des High-Pegels der Taktleitung, die im Singlemaster-Betrieb auch entfallen kann.

Danach wird der Pegel der Datenleitung SDA eingelesen und die Abspeicherung der Information in dem universell verwendbaren Flag F0 vorgenommen. Es folgt das

**Bild 150 : Ausgabe eines Bytes (8 Bit) über den I²C-Bus**

```
0096          I2C_WBYT: ; 1 Byte über die I2C Schnittstelle ausgeben
                        ; Eingang :     Das zu übertragene Zeichen (Byte) steht in A
                        ; Zerstört:     R6, R7, F0
                        ; Rückgabe:     C gesetzt, wenn Fehler erkannt wurde
0096  7E08              MOV    R6, #08      ; Schleife 8 mal durchlaufen
0098  33      I2C_WBY1: RLC    A            ; Bit 7 jeweils in C schieben
0099  1156              ACALL  I2C_WBIT     ; Ausgabe eines Bits auf den I2C-Bus
009B  400C              JC     I2C_WBYE     ; Springe zum Ende, wenn Fehler erkannt
009D  DEF9              DJNZ   R6, I2C_WBY1 ; Schleife 8 mal durchlaufen
009F  117C              ACALL  I2C_RBIT     ; Lesen eines Bits vom I2C-Bus
                                            ; In F0 steht der Inhalt des ACK-Bits
00A1  4006              JC     I2C_WBYE     ; Springe zum Ende, wenn Fehler
00A3  A2D5              MOV    C, F0        ; ACK-Bit lesen
00A5  5002              JNC    I2C_WBYE     ; Springe, wenn ACK-Bit ok ist
00A7  114B              ACALL  I2C_STOP     ; Stopp-Bedingung des I2C-Busses generieren
00A9  22      I2C_WBYE: RET                 ; Rücksprung
```

```
00AA          I2C_RBYT:  ; 1 Byte über die I2C-Schnittstelle lesen
                         ; Eingang :     C  gelöscht, wenn ACK im 9.Bit aktiviert werden soll
                         ; Zerstört:     R5, R6, R7, F0, A
                         ; Rückgabe:     C  gesetzt, wenn Fehler erkannt wurde
                         ;               A  Inhalt des 8 Bit Datenwortes
00AA 33                  RLC    A          ; Wert in Bit 0 von A schieben
00AB FD                  MOV    R5, A      ; Inhalt speichern
00AC 7E08                MOV    R6, #08    ; Schleife 8 mal durchlaufen
00AE 117C  I2C_RB1:      ACALL  I2C_RBIT   ; Lesen eines Bits vom I2C-Bus
00B0 400C                JC     I2C_RBYE   ; Springe zum Ende, wenn Fehler erkannt
00B2 A2D5                MOV    C, F0      ; Datenbit nach C kopieren
00B4 33                  RLC    A          ; C jeweils in Bit 0 schieben
00B5 DEF7                DJNZ   R6, I2C_RB1; Schleife 8 mal durchlaufen
00B7 CD                  XCH    A, R5      ; Inhalt von A und R5 tauschen
00B8 13                  RRC    A          ; Bit 0 nach C schieben
00B9 CD                  XCH    A, R5      ; Inhalt von A und R5 zurück tauschen
00BA 1156                ACALL  I2C_WBIT   ; Ausgabe eines Bits auf den I2C-Bus
00BC D2B2                SETB   SDA        ; Bus auf jeden Fall wieder freigeben
                                           ; In A steht das 8 Bit Datenwort
00BE 22    I2C_RBYE:     RET               ; Rücksprung
```

**Bild 151: Einlesen eines 8Bit-Wortes über die I²C-Schnittstelle**

```
00BF          I2C_WANZ:  ; n-Byte über die I2C-Schnittstelle ausgeben
                         ; Eingang :     A : Slaveadresse
                         ; R0:           RAM-Zeiger auf den Beginn der Zeichenfolge
                         ; R1:           Anzahl der zu schreibenden Bytes (0..n)
                         ; Zerstört:     R5, R6, R7, F0, A
                         ; Rückgabe:     C  gesetzt, wenn Fehler erkannt
00BF 1130                ACALL  I2C_STRT   ; Start-Bedingung des I2C-Busses generieren
00C1 4013                JC     I2C_WANE   ; Springe, wenn Fehler erkannt
00C3 C2E0                CLR    ACC.0      ; Bit 0(R/W) für Schreib-Zugriff löschen
00C5 1196                ACALL  I2C_WBYT   ; Slaveadresse über I2C ausgeben
00C7 400D                JC     I2C_WANE   ; Springe, wenn ACK-Fehler erkannt
00C9 E9                  MOV    A, R1      ; Anzahl lesen
00CA 6008                JZ     I2C_WE1    ; Springe, wenn die Anzahl = 0 ist
00CC E6    I2C_WA1:      MOV    A, @R0     ; nächstes Byte lesen
00CD 1196                ACALL  I2C_WBYT   ; 1 Byte über die I2C-Schnittstelle ausgeben
00CF 4005                JC     I2C_WANE   ; Springe, wenn Fehler erkannt
00D1 08                  INC    R0         ; Zeiger um 1 erhöhen
00D2 D9F8                DJNZ   R1, I2C_WA1; Schleife R 1 mal wiederholen
00D4 114B  I2C_WE1:      ACALL  I2C_STOP   ; Stopp-Bedingung des I2C-Busses generieren
00D6 22    I2C_WANE:     RET               ; Rücksprung
```

**Bild 152: Schreiben von n-Byte über den I²C-Bus**

Zurücksetzen der SCL-Steuerleitung auf Low-Pegel.

Auf den 4 vorstehend aufgeführten grundlegenden Routinen bauen die im nächsten Kapitel beschriebenen Programmteile auf.

### 6.10.2 Byte-Treiber

Abbildung 150 zeigt das Unterprogramm I2C_WBYT, welches ein 8Bit-Datenwort über den I²C-Bus ausgibt. Das in A (Akkumulator) übergebene Datenwort wird bitweise über das Unterprogramm I2C_WBIT ausgegeben. Anschließend erfolgt das Einlesen der ACK-Information, die die Empfangsbestätigung des angesprochenen Slave-Bausteins darstellt.

Bei einer nicht korrekt empfangenen ACK-Bestätigung wird von dem Unterprogramm automatisch die I²C-Stopp-Bedingung generiert zum Abbruch der bestehenden Verbindung.

Abbildung 151 zeigt das Unterprogramm I2C_RBYT, welches ein 8Bit-Datenwort vom angesprochenen Slave einliest und über das 9. Bit den gewünschten ACK-Pegel überträgt.

### 6.10.3 Block-Treiber

Abbildung 152 zeigt das Unterprogramm I2C_WANZ, welches die über R 1 übergebene Anzahl von Zeichen über die I²C-Schnittstelle ausgibt, wobei das Register R 0 auf den Beginn der Zeichenfolge im RAM des Mikrocontrollers zeigt. Nach der Generierung der Start-Bedingung erfolgt zunächst die Übertragung der in A übergebenen Slave-Adresse, deren Bit 0 für den Schreibzugriff gelöscht wird. Danach schreibt das Unterprogramm die Anzahl der in R 1 übergebenen 8Bit-Daten zum I²C-Bus. Den Abschluß des Unterprogrammes bildet die Generierung der I²C-Stopp-Sequenz, die die Freigabe des Busses vornimmt.

Abbildung 154 zeigt das Unterprogramm I2C_WR, welches n-Byte über die I²C-Schnittstelle schreibt und anschließend m-Bytes vom angeschlossenen Slave zurückliest. Bei den meisten Bausteinen mit I²C-Schnittstelle ist es sinnvoll bzw. erforderlich, bevor Daten gelesen werden, zunächst bestimmte Setup- oder Initialisierungsdaten zu schreiben. Bei den EEPROMs mit I²C-Schnittstelle sollte vor einem Lesezugriff immer die gewünschte Adresse geschrieben werden.

Nach dem Beginn der I²C-Übertragung durch die Generierung der Start-Bedingung erfolgt zunächst die Übertragung der zu schreibenden 8Bit-Daten zum Slave, wobei jeweils das Setzen des ACK-Flags durch den Slave kontrolliert wird. Anschließend werden nach der erneuten Generierung der Start-Bedingung die zu lesenden Bytes vom I²C-Bus gelesen, wobei das Unterprogramm seinerseits durch das Setzen des ACK-Bits jedes 8Bit-Wort bestätigt.

Das letzte zu lesende Byte wird, um den I²C-Bedingungen zu entsprechen, mit nicht aktiviertem ACK-Bit quittiert, wodurch der Slave die Information bekommt, daß die Datenübertragung beendet ist. Zum Abschluß generiert das Unterprogramm die Stopp-Bedingung und gibt damit den Bus wieder frei.

```
0118          EEPROMW1:  ; 1 Byte über die I2C-Schnittstelle zum  EEPROM übertragen
                         ; Eingang :  R0: Adresse im EEPROM
                         ;            R1: Bank im EEPROM
                         ;            R2: Date, die übertragen werden soll
                         ; Zerstört:  R6, R7, F0, A
                         ; Rückgabe: C  gesetzt, wenn Fehler erkannt
0118 1130                ACALL  I2C_STRT    ; Start-Bedingung des I2C-Busses generieren
011A 4018                JC     EEPROMWE    ; Springe, wenn Fehler erkannt
011C E9                  MOV    A, R1       ; Bank-Nummer laden
011D 5407                ANL    A, #00000111B; nur die unteren 3 Bits sind gültig
011F 23                  RL     A           ; Adresse an die richtige Stelle schieben
0120 44A0                ORL    A, #ADR_EEPR; EEPROM Adresse mit der Bank
                                            ; Nummer verknüpfen
0122 C2E0                CLR    ACC.0       ; Bit 0(R/W) löschen, für Schreib-Zugriff
0124 1196                ACALL  I2C_WBYT    ; Slaveadresse über I2C ausgeben
0126 400C                JC     EEPROMWE    ; Springe, wenn Fehler erkannt
0128 E8                  MOV    A, R0       ; Adresse im EEPROM lesen
0129 1196                ACALL  I2C_WBYT    ; Adresse, die angesprochen werden soll
012B 4007                JC     EEPROMWE    ; Springe, wenn Fehler erkannt
012D EA                  MOV    A, R2       ; Date lesen
012E 1196                ACALL  I2C_WBYT    ; Date, die geschrieben werden soll
0130 4002                JC     EEPROMWE    ; Springe, wenn Fehler erkannt
0132 114B                ACALL  I2C_STOP    ; Stopp-Bedingung des I2C-Busses generieren
0134 22    EEPROMWE:     RET                ; Rücksprung
```

**Bild 153: Schreiben eines Bytes zum EEPROM**

**Bild 154: Schreiben von n-Byte und anschließend Lesen von m-Byte über den I²C-Bus**

Mit den beiden bisher beschriebenen universellen Routinen lassen sich zwar die unterschiedlichsten I²C-Bausteine ansprechen, wobei vor und nach dem Aufruf ein entsprechend großer Aufwand besteht, um die Daten weiter zu verarbeiten. Die beiden folgenden Unterprogramme EEPROMW1 und EEPROMR1 ermöglichen das direkte Schreiben bzw. Lesen von einem adressierbaren Speicher des EEPROMs.

Bei dem ersten Unterprogramm EEPROMW1 (Abbildung 153) ist dazu in dem Register R 0 die Adresse im EEPROM, in R 1 die Bank des EEPROMs und in R 2 die Daten, die geschrieben werden sollen, zu übergeben.

Nach der Generierung der Start-Bedingung erfolgt die Verknüpfung der EEPROM-Adresse mit der gewünschten Bank, woraus die endgültige Slave-Adresse resultiert. Nach deren Ausgabe erfolgt die Übertragung der in R 0 übergebenen EEPROM-Adresse und der Daten, die geschrieben werden sollen. Den Abschluß bildet auch bei diesem Unterprogramm die Generierung der Stopp-Bedingung.

Abbildung 155 zeigt das Unterprogramm EEPROMR1, welches 1 Byte von dem angeschlossenen EEPROM liest. Dazu ist in dem Register R 0 die Adresse und in R1 die Bank im EEPROM zu übergeben. Nach der Generierung der Start-Bedingung erfolgt, wie bei EEPROMW1, die Verknüpfung der Bank mit der EEPROM-Adresse und der anschließenden Ausgabe über den I²C-Bus.

Nach der erfolgreichen Übertragung erfolgt noch die Ausgabe der Adresse, die im EEPROM selektiert werden soll. Um das Lesen eines Byte zu ermöglichen, ist zunächst nach der Generierung der Repeated-Start-Bedingung die erneute Ausgabe der EEPROM-Adresse mit gesetztem Bit 0 erforderlich, die dem Slave (EEPROM) mitteilt, daß die folgenden Daten vom Slave zum Master zu übertragen sind.

Da nur 1 Byte vom EEPROM gelesen werden soll, und dieses somit gleichzeitig das letzte Byte darstellt, erfolgt keine ACK-Bestätigung seitens des Masters, der zum Abschluß der Übertragung die Stopp-Bedingung generiert.

Nach der Erläuterung der I²C-Treiber beschreiben wir im 20. Teil der Mikrocontroller-Grundlagen-Serie die Anwendungsprogramme zum Auslesen bzw. Beschreiben von EEPROMs. ELV

**Bild 155: Lesen eines Bytes vom EEPROM**

```
00D7          I2C_WR:   ; n-Byte über die I2C Schnittstelle ausgeben  und
                        ; m-Byte über die I1C Schnittstelle  zurücklesen
                        ; Eingang :   R0: RAM-Zeiger auf den Beginn der Zeichenfolge
                        ;             R1: Anzahl der zu schreibenden Bytes (0..n)
                        ;             R2: Anzahl der zu lesenden Bytes (0..m)
                        ;             R3: Slaveadresse
                        ; Zerstört:   R4, R5, R6, R7, F0, A
                        ; Rückgabe: C gesetzt, wenn Fehler erkannt
00D7  E8                MOV     A, R0           ; Anfangsadresse für das RAM lesen
00D8  FC                MOV     R4, A           ; und sichern
00D9  E9                MOV     A, R1           ; Anzahl Bytes zum Schreiben
00DA  6013              JZ      I2C_WR2         ; Springe, wenn die Anzahl = 0 ist
00DC  1130              ACALL   I2C_STRT        ; Start-Bedingung des I2C-Busses generieren
00DE  4037              JC      I2C_WRE         ; Springe, wenn Fehler erkannt
00E0  EB                MOV     A, R3           ; Slaveadresse lesen
00E1  C2E0              CLR     ACC.0           ; Bit 0(R/W) für Schreib-Zugriff löschen
00E3  1196              ACALL   I2C_WBYT        ; Slaveadresse über I2C ausgeben
00E5  4030              JC       I2C_WRE        ; Springe, wenn Fehler erkannt
00E7  E6      I2C_WR1:  MOV     A, @R0          ; nächstes Byte lesen
00E8  1196              ACALL   I2C_WBYT        ; 1 Byte über die I2C-Schnittstelle ausgeben
00EA  402B              JC      I2C_WRE         ; Springe, wenn Fehler erkannt
00EC  08                INC     R0              ; Zeiger um 1 erhöhen
00ED  D9F8              DJNZ    R1, I2C_WR1     ; Schleife R 1 mal wiederholen
00EF  1130    I2C_WR2:  ACALL   I2C_STRT        ; Repeated Start-Bedingung generieren
00F1  4024              JC      I2C_WRE         ; Springe, wenn Fehler erkannt
00F3  EC                MOV     A, R4           ; Anfangsadresse lesen
00F4  F8                MOV     R0, A           ; und Zeiger setzen
00F5  EA                MOV     A, R2           ; Anzahl Bytes zum Lesen
00F6  601A              JZ      I2C_WR3         ; Springe, wenn die Anzahl = 0 ist
00F8  EB                MOV     A, R3           ; Slaveadresse lesen
00F9  D2E0              SETB    ACC.0           ; Bit 0(R/W) für Lese-Zugriff setzen
00FB  1196              ACALL   I2C_WBYT        ; Slaveadresse über I2C ausgeben
00FD  4018              JC      I2C_WRE         ; Springe, wenn Fehler erkannt
00FF  1A                DEC     R2              ; Anzahl um 1 heruntersetzen, da das letzte
0100  EA                MOV     A, R2           ; Byte mit NACK abgeschlossen wird
0101  6009              JZ      I2C_WR4         ; Springe, wenn die Anzahl = 0 ist
0103  C3      I2C_WR5:  CLR     C               ; C löschen, damit ACK generiert wird
0104  11AA              ACALL   I2C_RBYT        ; 1 Byte über die I2C-Schnittstelle lesen
0106  400F              JC      I2C_WRE         ; Springe, wenn Fehler erkannt
0108  F6                MOV     @R0, A          ; nächstes Byte speichern
0109  08                INC     R0              ; Zeiger um 1 erhöhen
010A  DAF7              DJNZ    R2, I2C_WR5     ; Schleife R 2 mal wiederholen
010C  D3      I2C_WR4:  SETB    C               ; C setzen, da das letzte Byte gelesen
010D  11AA              ACALL   I2C_RBYT        ; letztes Byte über die I2C-Schnittstelle lesen
010F  4006              JC      I2C_WRE         ; Springe, wenn Fehler erkannt
0111  F6                MOV     @R0, A          ; letztes Byte speichern
0112  C3      I2C_WR3:  CLR     C               ; kein Fehler erkannt
0113  EC                MOV     A, R4           ; Zeiger lesen und wieder
0114  F8                MOV     R0, A           ; auf den Anfang des Speichers zurücksetzen
0115  114B              ACALL   I2C_STOP        ; Stopp-Bedingung des I2C-Busses generieren
0117  22      I2C_WRE:  RET                     ; Rücksprung
```

```
0135          EEPROMR1: ; 1 Byte über die I2C-Schnittstelle vom EEPROM lesen
                        ; Eingang:   R0: Adresse im EEPROM
                        ;            R1: Bank im EEPROM
                        ; Zerstört:  R5, R6, R7, F0
                        ; Rückgabe:  C : gesetzt, wenn Fehler erkannt
                        ;            A : gelesenes Datenbyte
0135  1130              ACALL   I2C_STRT        ; Start-Bedingung des I2C-Busses generieren
0137  4024              JC      EEPROMRE        ; Springe, wenn Fehler erkannt
0139  E9                MOV     A, R1           ; Bank Nummer laden
013A  5407              ANL     A, #00000111B; nur die unteren 3 Bits sind gültig
013C  23                RL      A               ; Adresse an die richtige Stelle schieben
013D  44A0              ORL     A, #ADR_EEPR; EEPROM Adresse mit der Bank
                                                ; Nummer verknüpfen
013F  C2E0              CLR     ACC.0           ; Bit 0(R/W) löschen, für Schreib-Zugriff
0141  F9                MOV     R1, A           ; EEPROM Adresse sichern
0142  1196              ACALL   I2C_WBYT        ; Slaveadresse über I2C ausgeben
0144  4017              JC      EEPROMRE        ; Springe, wenn Fehler erkannt
0146  E8                MOV     A, R0           ; Adresse im EEPROM lesen
0147  1196              ACALL   I2C_WBYT        ; Adresse, die angesprochen werden soll
0149  4012              JC      EEPROMRE        ; Springe, wenn Fehler erkannt
014B  1130              ACALL   I2C_STRT        ; Repeated Start-Bedingung generieren
014D  400E              JC      EEPROMRE        ; Springe, wenn Fehler erkannt
014F  E9                MOV     A, R1           ; EEPROM-Adresse lesen
0150  D2E0              SETB    ACC.0           ; Bit 0(R/W) setzen, für Lese-Zugriff
0152  1196              ACALL   I2C_WBYT        ; Slaveadresse über I2C ausgeben
0154  4007              JC      EEPROMRE        ; Springe, wenn Fehler erkannt
0156  D3                SETB    C               ; Bit setzen, da hier das letzte Bit
0157  11AA              ACALL   I2C_RBYT        ; letztes Byte über die I2C-Schnittstelle lesen
0159  4002              JC      EEPROMRE        ; Springe, wenn Fehler erkannt
015B  114B              ACALL   I2C_STOP        ; Stopp-Bedingung des I2C-Busses generieren
015D  22      EEPROMRE: RET                     ; Rücksprung
```