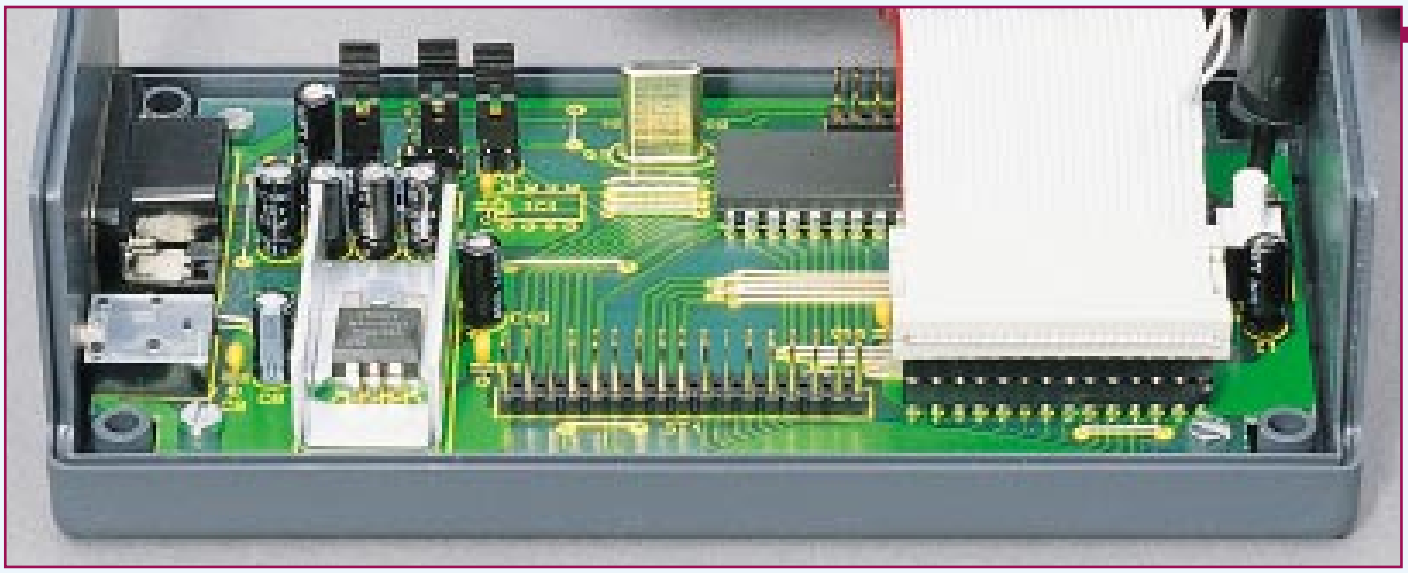

Mikrocontroller Grundlagen

Teil 21

In diesem Artikel beschreiben wir universelle Routinen, die es ermöglichen, beliebige I²C-Bausteine vom PC aus über die Grundschaltung (Kapitel 2) anzusteuern.

```
PRGM54:  ; Einlesen einer Zeile, Interpretation und Ausführung der Befehle
         ; Steuerung des Zugriffes auf verschiedene I2C Bausteine
         LCALL L52INIT              ; Init der seriellen Schnittstelle
                                    ; 9600 Baud, 8 Daten
         MOV DPTR, #TXT_STRT        ; Starttext ausgeben
         LCALL W_ROM                ; Ausgabe des ROM-Strings
PRGM54S: LCALL READLINE             ; Eine Zeile über die serielle
                                    ; Schnittstelle einlesen
         ; Auswertung der empfangenen Zeichen
         MOV  R0, #V24_BUFF         ;
         MOV  A, @R0                ;
         CJNE A, #0, P54_1          ; Springe, wenn nicht Zeile zu Ende
         SJMP PRGM54S               ; Schleife, weil Leerzeile empfangen
         ;
P54_1:   ANL  A, #11011111B         ; Umwandlung in Großschreibung
         CJNE A, #'H', P54_2        ; Springe, wenn nicht Befehl „H“
         SJMP P54_H                 ; Befehl „H“ erkannt
         ;
P54_2:   CJNE A, #'W', P54_3        ; Springe, wenn nicht Befehl „W“
         SJMP P54_W                 ; Befehl „W“ erkannt
         ;
P54_3:   CJNE A, #'R', P54_4        ; Springe, wenn nicht Befehl „R“
         SJMP P54_R                 ; Befehl „R“ erkannt
         ;
P54_4:   ; unbekannter Befehl
         MOV DPTR, #TXT_ERR_UB1     ; Fehlermeldung
         CALL W_ROM                 ; Ausgabe des ROM-Strings
         MOV  A, V24_BUFF           ; erstes Zeichen lesen
         LCALL L52AUSG              ; Ausgabe des in A übergebenen
                                    ; Zeichens
         MOV DPTR, #TXT_ERR_UB2     ; Fehlermeldung
         ACALL W_ROM                ; Ausgabe des ROM-Strings
         JMP  PRGM54G               ;

P54_H:   ; Befehl „H“ erkannt
         MOV DPTR, #TXT_HELP        ; Hilfetext ausgeben
         ACALL W_ROM                ; Ausgabe des ROM-Strings
         JMP  PRGM54G               ;
```

Bild 162: Hauptschleife des Testprogrammes 54

6.10.5 Universeller I²C Zugriff

Abbildung 162 zeigt die Hauptschleife des Testprogramms 54, in der zunächst nach der Ausgabe des Starttextes, mit Hilfe des Unterprogrammes W_ROM, eine komplette Zeile über die serielle Schnittstelle des Mikrocontrollers eingelesen wird.

Nach Abschluß der Zeile durch Empfang des Steuerzeichens Carriage Return erfolgt die Auswertung der empfangenen ASCII-Zeichen, die im RAM des Mikro-

Tabelle 26: Syntax der unterstützten Befehle mit jeweils einem Beispiel

Schreibzugriff: Wxxyy...		
Bsp.:	W40AA	Schreibzugriff auf den 8 Bit Port des PCF 8574
	WA0001122	Schreiben der Daten 11 H und 22 H in die Speicher ab der Adresse des EEPROMs 24C04
Schreib- und Lesezugriff: Wxxyy...Rn		
Bsp.:	WA000R8	Lesen der ersten 8 Speicherstellen des EEPROMs 24C04
Lesezugriff: Rxxn		
Bsp.:	R401	Portinhalt des 8 Bit Parallel Port Bausteins PCF 8574 lesen
Hilfetext: H		
		Hilfetext ausgeben, Parameter werden ignoriert

W: Schreibbefehl
R: Lesebefehl
xx: Slave-Adresse (immer 2stellig)
yy: 8 Bit Daten (immer 2stellig)
n: Anzahl Bytes (0...F, immer 1stellig)

```
P54_W:    ; Befehl „W“ erkannt
          MOV  A, V24_ANZAHL  ;
          CJNE A, #5, $+2     ; Anzahl mind. 5
          JNC  P54W_1         ; Springe, wenn Anzahl Zeichen > 5
          JMP  P54_WE1        ; Anzahl Parameter ist zu gering
          ;
P54W_1:   MOV  R0, #V24_BUFF+1 ; Beginn mit dem ersten Zeichen des Strings
          MOV  R1, #I2C_BUFF  ; Zeiger laden
          MOV  I2C_ANZAHL, #0 ; Anzahl zunächst 0
P54W_1W:  MOV  A, @R0         ; 1 Nibble lesen
          ANL  A, #11011111B  ; Umwandlung in Großschreibung
          CJNE A, #’R’,P54W_2 ; Springe, wenn nicht „R“
          SJMP P54W_R         ; „R“-Befehl
          ;
P54W_2:   MOV  A, @R0         ; 1 Nibble lesen
          JZ   P54W_END       ; Springe, wenn Ende des Strings
          CALL DECODE_2BYTE   ; 2 Zeichen decodieren
          JNC  P54W_3         ; Springe, wenn kein Konvertierungsfehler erkannt
          JMP  P54RW_G        ; ungültige Daten
          ;
P54W_3:   ; 8 bit Date steht in A und ist ok
          MOV  @R1, A         ; Date speichern
          INC  R1             ; Zeiger = Zeiger +1
          INC  I2C_ANZAHL     ; Anzahl = Anzahl +1
          SJMP P54W_1W        ; Schleife
          ;
P54W_END:                     ; Ende des Strings
          MOV  A, I2C_BUFF    ; Slaveadresse lesen
          MOV  R0, #I2C_BUFF +1 ; Pufferbeginn für die I2C-Ausgabe
          MOV  R1, I2C_ANZAHL ; Anzahl der im Puffer gesp. Werte
          DEC  R1             ; 1 abziehen, wg. der Slaveadresse
          CALL I2C_WANZ       ; n-Byte über die I2C-Schnittstelle ausgeben
          ; Rückgabe: C  gesetzt, wenn Fehler erkannt
          JNC  P54W_E1        ; Springe, wenn kein Fehler erkannt
          JMP  P54W_FI        ; Fehler bei der I2C Übertragung
          ;
P54W_E1:  ; kein Fehler bei der Datenübertragung
          MOV DPTR, #TXT_OK   ; Quitierung
          CALL W_ROM          ; Ausgabe des ROM-Strings
          JMP  PRGM54G        ;
```

Bild 163: Bearbeitung des Schreibbefehls

```
P54W_R:   ; „R“-Befehl im „W“-Befehl erkannt
          INC  R0             ;
          MOV  A, @R0         ; 1 Nibble lesen
          CALL ASC_NIB        ; Umwandlung ASCII
                              ; („0“..“F“) -> Byte (0..15)
          ; „C“ gesetzt, wenn Fehler erkannt
          JNC  P54W_R1        ; Springe, wenn kein Fehler
          JMP  P54RW_G        ; Springe, weil Fehler erkannt
P54W_R1:  MOV  R2, A          ; Anzahl zu lesender Bytes
          MOV  R0, #I2C_BUFF +1 ; Beginn der Zeichenfolge
          MOV  R1, I2C_ANZAHL ; n Bytes zum Schreiben
          DEC  R1
          MOV  I2C_ANZAHL, A  ; Anzahl zu lesender Bytes sichern
          MOV  R3, I2C_BUFF   ; Slaveadresse
          SJMP P54R_GEM       ; gemeinsame Weiterverarbeitung
```

Bild 164: Kombination des Schreib- und Lesebefehls

```
P54_R:    ; Befehl „R“ erkannt
          MOV  A, V24_ANZAHL  ;
          CJNE A, #4, P54R_F1 ; Springe, wenn Anzahl <> 4
          MOV  R0, #V24_BUFF+1 ; Beginn mit dem ersten Zeichen des Strings
          MOV  A, @R0         ; 1 Nibble lesen
          CALL DECODE_2BYTE   ; 2 Zeichen decodieren
          JC   P54RW_G        ; Springe, wenn Konvertierungsfehler erkannt
          ; 8 bit Date steht in A und ist ok
          MOV  I2C_BUFF, A    ; Date speichern
          MOV  I2C_ANZAHL, #0 ; Anzahl zunächst 0
          MOV  A, @R0         ; Zeichen aus dem String lesen
          CALL ASC_NIB        ; Umwandlung ASCII („0“..“F“) -> Byte (0..15)
          ; „C“ gesetzt, wenn Fehler erkannt
          JC   P54RW_G        ; Springe, wenn Fehler erkannt
          MOV  I2C_ANZAHL, A  ; Anzahl zu lesender Bytes sichern
          MOV  R0, #I2C_BUFF +1 ; Beginn der Zeichenfolge
          MOV  R1, #0         ; 0 Bytes zum Schreiben
          MOV  R2, A          ; Anzahl zu lesender Bytes
          MOV  R3, I2C_BUFF   ; Slaveadresse
P54R_GEM: CALL I2C_WR         ; n-Byte über die I2C Schnittstelle ausgeben und
                              ; m-Byte über die I2C Schnittstelle zurücklesen
          ; Rückgabe: C  gesetzt, wenn Fehler erkannt
          JNC  P54R_1         ; Springe, wenn kein Fehler erkannt
          SJMP P54W_FI        ; Fehler bei der I2C Übertragung
P54R_1:   ;  kein Fehler bei der Datenübertragung
          MOV  R0, #I2C_BUFF +1 ; Beginn des Lesepuffers
P54R_W:   MOV  A, @R0         ; Date lesen
          SWAP A              ; zuerst MSB
          CALL NIB_ASC        ; Nibble -> ASCII Konvertierung
          LCALL L52AUSG       ; Ausgabe des in A übergebenen Zeichens
          MOV  A, @R0         ; Date lesen
          CALL NIB_ASC        ; Nibble -> ASCII Konvertierung
          LCALL L52AUSG       ; Ausgabe des in A übergebenen Zeichens
          MOV  A, #’ ,        ; Blank als Trennug
          LCALL L52AUSG       ; Ausgabe des in A übergebenen Zeichens
          INC  R0             ; Zeiger = Zeiger +1
          DJNZ I2C_ANZAHL, P54R_W; Schleife, bis alle ausgegeben
          ;
          MOV DPTR, #TXT_OK   ; Quitierung
          ACALL W_ROM         ; Ausgabe des ROM-Strings
          SJMP PRGM54G        ;
```

Bild 165: Bearbeitung des Lesebefehls

controllers ab der Speicherzelle V24_BUFF abgelegt sind. Das Ende der Zeile wird durch das sogenannte Null-Byte gekennzeichnet. Bei Empfang einer Leerzeile würde das Null-Byte gleich am Anfang im Puffer abgelegt werden, woraufhin das Programm die Schleife mit dem erneuten Einlesen einer Zeile wiederholt.

Im weiteren Verlauf des Programmes wird zunächst ab dem Label P54_1 eine Umwandlung des Befehls, der immer zu Beginn des Puffers stehen muß, in Großbuchstaben durch Löschen des Bits 5 vorgenommen. Anschließend erfolgt die Verzweigung in eine der drei Programmteile, die zu den möglichen Befehlen „H”, „W” oder „R” gehören.

Bei Selektierung eines unbekannten Befehls erfolgt ab dem Label P54_4 die Ausgabe der entsprechenden Fehlermeldung, die zusätzlich noch die eingegebene Zeile wiederholt.

Unten in Abbildung 162 erfolgt die Abarbeitung des Befehls „H”, der einen kleinen Hilfetext über die serielle Schnittstelle mit Hilfe des Unterprogrammes W_ROM ausgibt.

6.10.5.1. Befehlssyntax

Tabelle 26 zeigt die Syntax der unterstützten Befehle des Testprogrammes 54 mit jeweils einem Beispiel. Es werden die Befehle „W”, „R” und „H” in Groß- und Kleinschreibung akzeptiert, deren Bedeutung nachfolgend kurz erklärt wird.

Der Befehl „W” nimmt einen reinen Schreibzugriff auf den adressierten I²C-Bus-Teilnehmer vor. Der erste Parameter ist immer die Slave-Adresse, gefolgt von den zu übertragenden Daten, deren Anzahl immer gerade sein muß. Für die Abarbeitung eines I²C-Schreibbefehles sind mindestens 5 ASCII-Zeichen (Befehl, Slave-Adresse und mindestens eine Date) zu übertragen.

Der reine Lesezugriff auf einen I²C-Bus-Teilnehmer erfolgt durch den Befehl „R”, der neben der Angabe der Slave-Adresse noch die Anzahl der über den I²C-Bus zu lesenden Bytes angibt (einstellig, 0...F).

Weiterhin ist eine Kombination aus dem Schreib- und Lesebefehl möglich, wobei zunächst der Schreibbefehl gefolgt von der Slave-Adresse und mindestens einem Parameter anzugeben ist. Der Lesevorgang wird dann durch den Befehl „R” gefolgt von der Anzahl der zu lesenden Bytes eingeleitet, wobei die nochmalige Angabe der Slave-Adresse nicht notwendig ist.

Über den Befehl „H” kann ein kleiner Hilfetext mit einer kurzen Syntaxbeschreibung abgerufen werden.

6.10.5.2 I²C-Schreibzugriff

Abbildung 163 zeigt den Programmteil für die Abarbeitung des Schreibbefehls „W”. Zunächst wird die Anzahl der Parameter überprüft, die einschließlich des eigenen Befehls mindestens 5 betragen muß, anderenfalls erfolgt eine entsprechende Fehlermeldung. Es schließt sich die Überprüfung auf den „R” Befehl an, dessen Funktionsweise im folgenden Kapitel näher beschrieben wird.

Nach der Null-Byte-Überprüfung, die das Ende der Zeichenfolge ankündigt, wird die Dekodierung der nächsten beiden im

```
P54_WE1:  ; Anzahl Parameter ist zu gering
          MOV DPTR, #TXT_ERR_ANZ  ; Fehlermeldung
          ACALL W_ROM             ; Ausgabe des ROM-Strings
          SJMP PRGM54G            ;
          ;
P54R_F1:  ; falsche Anzahl Parameter
          MOV DPTR, #TXT_ERR_UG4  ; Fehlermeldung
          ACALL W_ROM             ; Ausgabe des ROM-Strings
          SJMP PRGM54G            ;
          ;
P54RW_G: ; ungültige Date erkannt
          MOV DPTR,#TXT_ERR_PAR1 ; Fehlermeldung
          ACALL W_ROM             ; Ausgabe des ROM-Strings
          CALL W_RAM_LINE         ; Eine Zeile ausgeben
          MOV DPTR,#TXT_ERR_PAR2 ; Fehlermeldung
          ACALL W_ROM             ; Ausgabe des ROM-Strings
          SJMP PRGM54G            ;
          ;
P54W_FI:  ; Fehler bei der I2C Übertragung
          MOV DPTR, #TXT_ERR_I2C ; Fehlermeldung
          ACALL W_ROM             ; Ausgabe des ROM-Strings
          ;SJMP PRGM54G           ;
PRGM54G: JMP  PRGM54S             ; Endlosschleife
```

Bild 166: Behandlungen der Fehlermeldungen

Puffer abgelegten Zeichen vorgenommen, die wie im vorhergehenden Abschnitt beschrieben in Hexadezimal („0" bis „9" bzw. „A" bis „F") vorliegen. Bei einer Syntax-Verletzung (ASCII-Zeichen nicht im geforderten Bereich) erfolgt eine entsprechende Fehlermeldung.

Eine korrekt konvertierte 8-Bit-Date legt die nachfolgenden Befehle in den I²C-Puffer ab, gefolgt von der Dekodierung des nächsten Parameters.

Nach erfolgreicher Überprüfung der angegebenen Parameter schließt sich die Ausgabe der N-Byte über die I²C-Schnittstelle an, wobei auch hier nach einer Fehlererkennung zu der entsprechenden Behandlungsroutine verzweigt wird. Im Nicht-Fehlerfall erfolgt die Ausgabe der positiven Quittierung über die serielle Schnittstelle.

6.10.5.3 I²C Schreib-/Lesezugriff

Wie bereits im vorhergehenden Kapitel beschrieben, kann nach der Ausführung des Schreibbefehls noch das Lesen von N-Bytes über die I²C-Schnittstelle erfolgen. Dazu wird die in Abbildung 164 beschriebene Befehlsfolge abgearbeitet, die zunächst die Anzahl der zu lesenden Bytes dekodiert, und anschließend im Nicht-Fehlerfall das Lesen der Bytes über den I²C-Bus ab dem Label P54R_GEM vornimmt, welches im nachfolgenden Kapitel beschrieben wird.

6.10.5.4 I²C Lesezugriff

Abbildung 165 zeigt die Befehlsfolge, um den Lesezugriff auf den I²C-Bus vornehmen zu können. Zunächst wird die Anzahl der Parameter überprüft, die genau 4 (Befehl, Slave-Adresse und Anzahl) sein muß. Nach der Dekodierung der Slave-Adresse und der Anzahl der über den I²C-Bus zu lesenden Bytes erfolgt der Zugriff auf den I²C-Bus über die Routine I2C_WR, die bereits im 19. Teil dieser Artikelserie beschrieben wurde.

Nach einem fehlerhaften Zugriff auf den I²C-Bus kommt eine Ausgabe einer entsprechenden Fehlermeldung.

Nach erfolgreichem Bus-Zugriff werden die Bytes jeweils in hexadezimaler Form über die serielle Schnittstelle ausgegeben, bei Trennung durch jeweils ein Leerzeichen.

6.10.5.5 Fehlerbehandlungen

Abbildung 166 zeigt die Behandlung der bei der Konvertierung erkannten syntaktischen Fehler.

Im allgemeinen wird dazu jeweils die Anfangsadresse des Meldetextes, der im ROM abgelegt ist, in den Datenpointer DPTR geladen. Das Unterprogramm W_ROM sorgt für die Ausgabe der im ROM-Bereich abgelegten ASCII-Zeichen, um

```
CR              EQU 0DH           ; Carriasge Return
LF              EQU 0AH           ; Line Feed
TXT_STRT:       DB                ,I2C Master Treiber  Hilfe mit „H"',CR,LF,0
TXT_ERR_UB1:    DB ,Fehler: unbekannter Befehl :"',0
TXT_ERR_UB2:    DB ,,,  Hilfe mit „H"',CR,LF,0
TXT_ERR_ANZ:    DB ,Fehler: zu wenig Parameter',CR,LF,0
TXT_ERR_PAR1:   DB ,Fehler: ungültigen Parameter :"',0
TXT_ERR_PAR2:   DB ,,,,,CR,LF,0
TXT_ERR_UG4:    DB ,Fehler: Anzahl Parameter ist <> 3',CR,LF,0
TXT_ERR_I2C:    DB ,Fehler bei der I2C Datenübertragung',CR,LF,0
TXT_OK:         DB ,ok',CR,LF,0
TXT_HELP:       DB ,Hilfe: Befehle, die unterstützt werden: ,,CR,LF
                DB ,W Slaveadresse Par1 Par2..',CR,LF
                DB ,W Slaveadresse Par1 Par2..R Anzahl',CR,LF
                DB ,R Par1 Anzahl',CR,LF
                DB ,H diese Hilfe',CR,LF
                DB ,W braucht mind. 4 Parameter (Adr. u. mind. 1Byte)',CR,LF
                DB ,R braucht genau 3 Parameter (Adr, Anzahl)',CR,LF
                DB ,Ende der Hilfe',CR,LF,0
                ;
W_ROM:          ; Ausgabe einer Zeichenfolge, die im ROM steht
    ;           DPTR zeigt auf den Anfang des Ausgabestrings
W_ROM_W: MOV  A, #0               ;
                MOVC A, @A+DPTR ; Zeichen lesen
                JZ   W_ROM_E      ; Ende, wenn 0-Byte
                LCALL L52AUSG   ; Ausgabe des in A übergebenen Zeichens
                INC  DPTR         ; ROM-Zeiger um 1 erhöhen
                SJMP W_ROM_W    ; Schleife
W_ROM_E:        RET               ; Ende des Unterprogrammes
```

Bild 167: Ausgabe der erforderlichen Texte über die serielle Schnittstelle

```
ASC_NIB:        ; Umwandlung ASCII („0"..."F") -> Byte (0..15)
                ; „C" gesetzt, wenn Fehler erkannt
                CJNE A, #'0',$+2        ;
                JC   ASC_NERR           ; Springe, wenn Zeichen < als „0"
                CJNE A, #'9'+1,$+2      ;
                JNC  ASC_N1             ; Springe, wenn > „9" ist
                ; Zeichen ist im Bereich „0"..."9"
                ADD  A, #100H -'0'      ; ASCII „0" abziehen
                ; „C" immer gesetzt
                SJMP ASC_NOK            ;
ASC_N1:         ANL  A, #11011111B      ; Umwandlung in Großschreibung
                CJNE A, #'A',$+2        ;
                JC   ASC_NERR           ; Springe, wenn Zeichen < als „A"
                CJNE A, #'F'+1,$+2      ;
                JNC  ASC_NERR           ; Springe, wenn > „F" ist
                ; Zeichen ist im Bereich „A"..."F"
                ADD  A, #100H -'A'+10   ; ASCII „A" abziehen
ASC_NOK: ;kein Fehler
                CLR  C                  ; C = 0
                RET                     ;
ASC_NERR:; Fehler
                SETB C                  ; C = 1
                RET                     ;

DECODE_2BYTE:; 2 Byte decodieren
                ; R0 zeigt auf das nächste Zeichen
                MOV  A, @R0             ; Zeichen aus dem String lesen
                CALL ASC_NIB            ; Umwandlung ASCII
                                        ; („0"..."F") -> Byte (0..15)
                ; „C" gesetzt, wenn Fehler erkannt
                JC   DECO_FG            ; Springe, wenn Fehler erkannt
                SWAP A                  ;
                MOV  R6, A              ; Inhalt sichern
                INC  R0                 ; nächstes Zeichen
                MOV  A, @R0             ; Zeichen aus dem String lesen
                CALL ASC_NIB            ; Umwandlung ASCII
                                        ; („0"..."F") -> Byte (0..15)
                ; „C" gesetzt, wenn Fehler erkannt
                JC   DECO_FG            ; Springe, wenn Fehler erkannt
                 ORL  A, R6             ; oberes Nibble von R6 dazu
                INC  R0                 ; nächstes Zeichen
                ; „C" ist noch 0
                RET                     ;
DECO_FG: SETB C                 ; C = 1
                RET

TAB_ASC:        DB ,0123456789ABCDEF'   ;
NIB_ASC: ; Nibble > ASCII Wandlung
                ANL  A, #00001111B      ; nur das untere Nibble (0..15)
                MOV  DPTR, #TAB_ASC     ; Tabellenanfang laden
                MOVC A, @A+DPTR         ; ASCII-Zeichen aus Tabelle lesen
                RET                     ;
```

Bild 168: Umwandlung der ASCII-Zeichen in 4-Bit-Nibble und Umwandlung von 4-Bit-Nibble in ASCII-Zeichen

dem Anwender eine möglichst detaillierte Fehlerbeschreibung geben zu können.

6.10.5.6 Unterprogramme

Zur Ausführung der oben beschriebenen Funktionen sind einige Unterprogramme erforderlich, die nun kurz beschrieben werden sollen. Abbildung 167 zeigt die im ROM abgelegten ASCII-Zeichen, und das dazugehörige Unterprogramm W_ROM, welches die im ROM abgelegte Zeichenfolge, die durch das Null-Byte beendet wird, über die serielle Schnittstelle ausgibt.

Abbildung 168 zeigt das Unterprogramm ASC_NIB, welches die Umwandlung von dem in A übergebenen ASCII-Zeichen in den zugehörigen 4-Bit-Binär-Code übersetzt. Bei einer Fehlererkennung (ASCII-Zeichen außerhalb des erlaubten Bereiches „0" bis „9" bzw. „A" bis „F") erfolgt das Setzen

```
W_RAM_LINE:; Eine Zeile ausgeben
            MOV  R0, #V24_BUFF     ; Bufferanfang laden
            MOV  R1, V24_ANZAHL    ; Anzahl laden
            ;
W_RAM_ANZ:; R1 Bytes vom RAM Inhalt ab der Adresse R0 ausgeben
W_RAM_W: MOV  A, @R0               ; Zeichen lesen
            JZ   W_RAM_E           ; Ende, wenn 0-Byte
            LCALL L52AUSG          ; Ausgabe des in A
                                   ; übergebenen Zeichens
            INC  R0                ; Zeiger = Zeiger +1
            DJNZ R1, W_RAM_W       ; Schleife
W_RAM_E: RET                       ;

CR_LF:      ; Carriage Returen und Line Feed ausgeben
            MOV  A, #0DH           ; Carriage Return
            LCALL L52AUSG          ; Ausgabe des in A
                                   ; übergebenen Zeichens
            MOV  A, #0AH           ; Linefeed
            LCALL L52AUSG          ; Ausgabe des in A
                                   ; übergebenen Zeichens
            RET                    ; Ende des Unterprogrammes
```

Bild 169: Ausgabe einer Zeichenfolge, die in RAM steht

Universelle I²C-Treiber erlauben die einfache Ansteuerung von beliebigen I²C-Bus-Slaves über eine V24/RS232-Schnittstelle.

des Carry-Flags, welches vom aufrufenden Programmteil entsprechend auszuwerten ist.

In Abbildung 168 ist ebenfalls das Unterprogramm DECODE_2BYTE zu sehen, welches zwei im RAM durch das Register R 0 adressierte ASCII-Zeichen in einem 8Bit-Wert umwandelt. Dazu wird das Unterprogramm ASC_NIB zweimal aufgerufen.

Das Unterprogramm NIB_ASC geht den umgekehrten Weg und wandelt das in A übergehende 4Bit-Nibble (0 bis 15) in das dazugehörige ASCII-Zeichen („0" bis „9" bzw. „A" bis „F") um, wobei das Unterprogramm gleichzeitig die einfache Benutzung von Tabellen demonstriert.

Abbildung 169 zeigt das Unterprogramm W_RAM_LINE, das den Inhalt des V24-Puffers über die serielle Schnittstelle ausgibt. Die Terminierung erfolgt auch hier durch ein Null-Byte.

Das darunter abgebildete Unterprogramm CR_LF gibt die Zeichenfolge Carriage Return und Line Feed über die serielle Schnittstelle aus, um den Abschluß einer Zeichenfolge zu kennzeichnen.

Abbildung 170 zeigt das Unterprogramm READLINE, welches einen kompletten Zeilen-Editor darstellt. Die über die serielle Schnittstelle empfangenen Zeichen werden zunächst überprüft und anschließend in dem V24-Bereich abgelegt. Eine Falscheingabe kann durch die Betätigung der Backspace-Taste korrigiert werden. Die Routine akzeptiert maximal 40 ASCII-Zeichen, so daß ein RAM-Überlauf bzw. eine Kollision mit anderen RAM-Inhalten ausgeschlossen ist.

Der Abschluß einer Zeile erfolgt ausschließlich nach dem Empfang des ASCII-Zeichens Carriage Return, womit gleichzeitig das Null-Byte als Kennzeichnung des String-Endes in den RAM-Bereich geschrieben wird.

Die Ablage des empfangenen ASCII-Zeichens erfolgt ab dem Label READ_3A, gefolgt von der Berechnung der richtigen RAM-Adresse und dem Schreiben des ASCII-Zeichens in den adressierten Speicher. Anschließend kommt das Echo des ASCII-Zeichens über die serielle Schnittstelle, um gleichzeitig eine visuelle Kontrolle für die Bedienung zu ermöglichen.

Im 22. Teil der Mikrocontroller-Grundlagen-Artikelserie folgt eine Übersicht über die unterschiedlichen 8051kompatiblen Mikrocontroller der verschiedenen Hersteller. ELV

```
I2C_ANZAHL EQU 020H                ; Anzahl der gültigen Zeichen im I2C Speicher
I2C_BUFF   EQU 021H                ; Puffer für die I2C Daten (max. 15 Zeichen)
V24_ANZAHL EQU 030H                ; Speicherplatz für den Sende-, bzw. Empfangspuffer
V24_BUFF   EQU 031H                ; Puffer ab 21H max. 40 Zeichen
READLINE:; Eine Zeile über die serielle Schnittstelle einlesen
            MOV  V24_ANZAHL, #0; zunächst ist die Anzahl der Zeichen 0
READL_S: LCALL L52READ             ; Warten, bis ein Zeichen empfangen
            CJNE A, #8, READL_1    ; Springe, wenn Zeichen nicht Back-Space
            ; Zeichen ist Back-Space
            MOV  A, V24_ANZAHL     ; Zeiger lesen
            ;CLR  C                ; C = 0
            ;SUBB A, #V24_BUFF+1   ; Bufferanfang abziehen
            ;JNC  READL_0A         ; Springe, wenn Anzahl > 0 ist
            JNZ  READL_0A          ; Springe, wenn Anzahl > 0 ist
            SJMP READL_S           ; Zeichen ignorieren
READL_0A:DEC  V24_ANZAHL           ; letztes Zeichen damit löschen
            MOV  A, #08H           ; zunächst Cursor um 1 nach links
            LCALL L52AUSG          ; Ausgabe des in A übergebenen Zeichens
             MOV  A, #' ,          ; dann überschreiben
            LCALL L52AUSG          ; Ausgabe des in A übergebenen Zeichens
            MOV  A, #08H           ; und nochmal Cursor nach links
            LCALL L52AUSG          ; Ausgabe des in A übergebenen Zeichens
            SJMP READL_S           ; Schleife
            ;
READL_1:    CJNE A, #0AH, READL_2  ; Springe, wenn Zeichen nicht Line Feed
            ; Zeichen ist Line Feed
            SJMP READL_S           ; Zeichen ignorieren
            ;
READL_2:     CJNE A, #0DH, READL_3 ; Springe, wenn Zeichen nicht Carriage Return
            ; Zeichen ist Carriage Return
            MOV  A, #V24_BUFF      ; Adresse des Bufferanfangs
            ADD  A, V24_ANZAHL     ; Zeiger auf Bufferende
            MOV  R0, A             ; Zeiger nach R0
            MOV  @R0, #0           ; Ende Kennzeichnung schreiben
            ;
            MOV  A, #0DH           ; Carriage Return
            LCALL L52AUSG          ; Ausgabe des in A übergebenen Zeichens
            MOV  A, #0AH           ; Linefeed
            LCALL L52AUSG          ; Ausgabe des in A übergebenen Zeichens
            SJMP READL_E           ;
READL_3:    ; Zeichen kann in Puffer abgelegt werden
            MOV  R0, V24_ANZAHL    ;
            CJNE R0, #40, READL_3A ; Springe, wenn Bufferende noch
                                   ; nicht erreicht
            SJMP READL_S           ; Schleife
READL_3A:PUSH ACC                  ; Zeichen speichern
            MOV  A, V24_ANZAHL     ; Anzahl lesen
            ADD  A, #V24_BUFF      ; Pufferanfang addieren
             MOV  R0, A            ;
            POP  ACC               ; Zeichen wieder zurückholen
            MOV  @R0, A            ; Zeichen in Puffer ablegen
            LCALL L52AUSG          ; Ausgabe des in A übergebenen Zeichens
            INC  V24_ANZAHL        ; Zeiger auf nächstes Zeichen
            SJMP READL_S           ; Schleife
            ;
            READL_E: RET           ; Ende des Unterprogrammes
```

Bild 170: Einlesen einer Zeile über einen komfortablen Zeileneditor