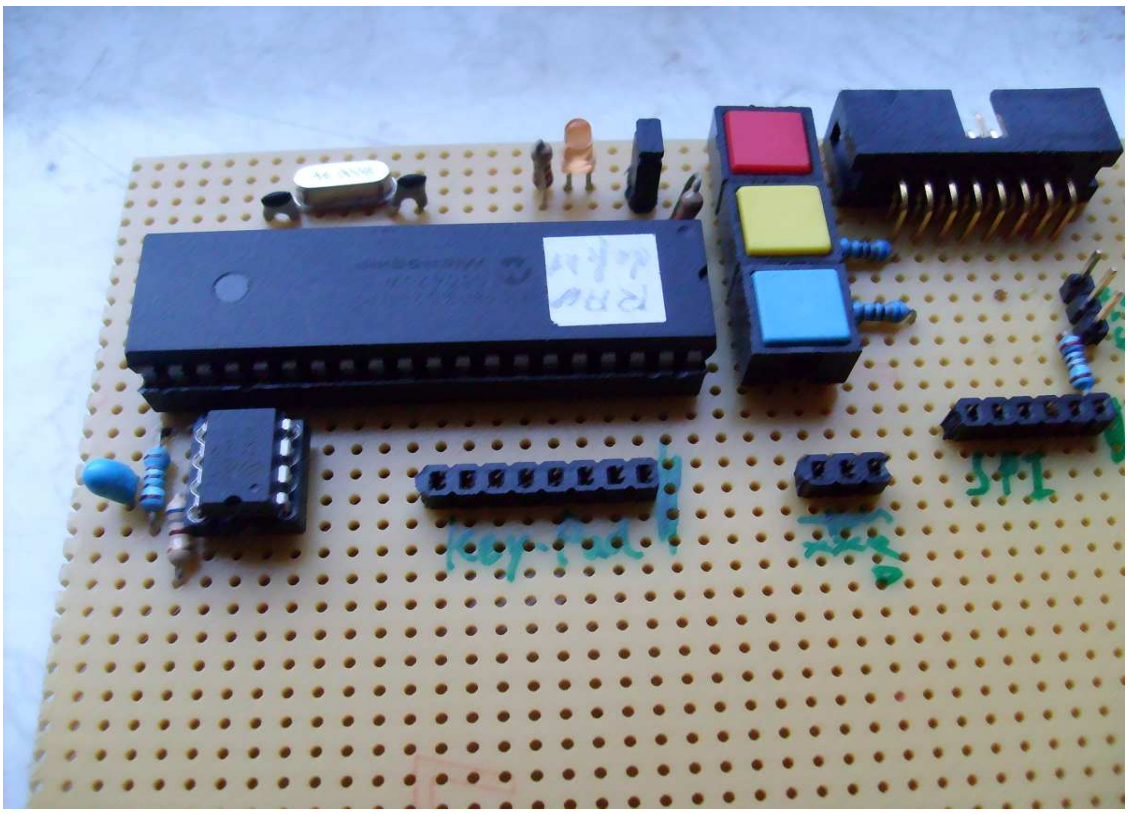

Selbstgebauter PIC-gestützter manipulationsgeschützter Datenlogger mit Bediengerät

Wettbewerb "Jugend Forscht" 2009

Lucas Jürgens (13 Jahre)

Arbeitsgemeinschaft "Jugend Forscht"

des Christian-Gymnasiums Hermannsburg

Leitung: StD Thomas Biedermann

Inhaltsverzeichnis

1 Ziel

Mein Ziel ist es, einen Datenlogger zu bauen, der möglichst günstig, klein und manipulationssicher ist.
Man soll ihn nur mit einem Bedienteil steuern können, damit man nicht versehentlich die Einstellung ändert.
Er soll Daten von verschiedenen Sensoren aufnehmen und auch speichern können.

2 Umsetzung

Um die Elektronik zu steuern, verwende ich sowohl im Bedienteil als auch im Datenlogger selbst einen PIC – Mikrocontroller. Beide werden vorher mit einem von mir programmierten Programm „gebrannt“.
Um den Datenlogger einzustellen, muss man ihn an das Bedienteil anschließen. So kann man auch die gespeicherten Messwerte wieder auslesen und auf den PC übertragen.

3 PIC-Mikrocontroller

Ein PIC ist ein kleiner Mikrocontroller, den man über einen PC programmieren kann. Sie werden oft in z.B. Digitaluhren, Handys usw. verwendet. Sie können Tastendrücke und vieles mehr verarbeiten, so kann man z.B. bei Digitaluhren die Uhrzeit einstellen. Für Mikrocontroller gibt es viele Anbieter, ich verwende in meinem Projekt Mikrocontroller von der Firma Microchip.

Alle PICs dieser Serie verfügen neben dem Prozessor über 2 verschiedene Speicher (Flash und EEPROM), Digital-Ports sowie eine Programmierschnittstelle (SPI). Je nach Ausführung können weitere Komponenten vorhanden sein, z.B. AD-Wandler, Timer usw. Für mein Projekt benutze ich einen PIC mit der Bezeichnung PIC18F452.
Er verfügt über:

- 8 A/D Wandler mit 10 Bit Auflösung
- 4 Timer

- 32 KByte Flashspeicher
- 256 Byte EEPROM
- USART-Schnittstelle

Die A/D-Wandler (Analog-Digital-Wandler) wandeln die analogen Werte (die Signale von den Sensoren) in binär speicherbare digitale Werte um. Da mir 8 A/DWandler zu Verfügung stehen, kann ich mehrere Messdaten gleichzeitig aufnehmen. Deren Auflösung beträgt 10 Bit.
Die integrierte USART-Schnittstelle erlaubt die serielle Datenübertragung vom und zum Bedienteil sowie zum PC.

3.1 Die Programmierung eines PICs

Der PIC arbeitet jeden Befehl, der im Programmcode steht, einzeln und nacheinander ab. Mit geeigneten Befehlssequenzen kann ich bewirken, dass z.B. bei einem Tastendruck eine LED blinkt oder ein Messwert aufgenommen und gespeichert wird.
Zuerst schreibt man den Programmcode in der Programmiersprache C. Wenn der C-Code fertig ist, wird er compiliert, das heißt, dass er in eine für den PIC spezielle Maschinensprache umgewandelt wird.
Für den Compiliervorgang benutze ich die kostenlose Studentenversion Mplab von Microchip [2], die auch die entsprechenden C-Bibliotheken bereitstellt.
Wenn der Compiler keine Fehler ausgibt, wird eine HEX-Datei erstellt, die mit einer speziellen Software unter Verwendung eines einfachen Programmierinterfaces auf den PIC geladen werden kann.
Dazu benutze ich das Freeware-Programm IC-Prog [4]. Der PIC wird dazu über die SPI-Anschlüsse mit dem Programmierinterface verbunden, welches wiederum über den LPT-Port mit dem Computer verbunden ist. Beim Programmieren löscht das Programm den Inhalt des Programmspeichers, überträgt anschließend die Daten an den PIC und überprüft sie, als Ergebnis ist nach weniger als einer Minute der PIC „gebrannt“. Mit der selben Software kann auch der Inhalt des PIC-Ramspeichers und EEPROM ausgelesen werden.

3.2 Der Aufbau eines C-Programmcode eines PICs

Zunächst müssen die Header-Files angegeben werden, auf die der Compiler zugreifen muss und die die meisten Befehle enthalten, die für den PIC speziell benötigt werden. Eine entsprechende Zeile lautet z.B.:

```
#inlude <p18f452.h>
```

Diese enthält vor allem für diesen PIC die speziellen vordefinierten Bezeichner für die einzelnen Ports sowie die Adressen dieser Ports und besonderer Register.
Anschließend werden die Einstellungen für den Chip vorgenommen, dazu gehören z.B. die Herkunft des Taktsignals, das Verhalten bei zu niedriger Versorgungsspannung usw. Dies geschieht in einem Abschnitt, der mit

```
#pragma romdata CONFIG
```

beginnt und mit der Zeile

```
#pragma romdata
```

abgeschlossen wird. Anschließend werden Funktionen deklariert, die in anderen Modulen definiert sind, wie z.B.

```
extern void SleepMS(int MilliSeconds);
```

bzw. die im weiteren Verlauf des Codes benutzt und definiert werden und einen anderen Rückgabewert als die Voreinstellung int haben, z.B.:

```
void BeekSpkr(int freq, int duration);
```

Als nächstes werden die Portregister festgelegt, also ob sie Ein- oder Ausgänge darstellen sollen, z.B.:

```
TRISA = 0b00000000;
```

zudem wird das Verhalten der einzelnen Interruptquellen eingestellt, z.B.:

```
RCONbits.IPEN = 0;
```

Um eine LED blinken zu lassen, sieht der Code dann folgendermaßen aus:

```
while(1)                    Endlosschleife
{                           Beginn der Schleife
   PORTA |= 0xff;           Alle LED’s an PortA anschalten.
   SleepMS(1000);           Warte 1 sek
   PORTA = 0x00;            Alle LED’s wieder ausschalten
   SleepMS(1000);           Wartet 1 sek
}                           Ende der Schleife
```

Hinter jeden Befehl kommt ein Semikolon, dann weiß der Compiler, dass der Befehl zu Ende ist. So könnte man die vier Befehle auch in zwei Zeilen schreiben:

```
PORTA |= 0xff; SleepMS(1000);
PORTA |= 0x00; SleepMS(1000);
```

4 Die Elektronik

Damit der PIC läuft, benötigt er mindestens folgende unverzichtbare Komponenten:

- Ein Quarz für den Takt (ich verwende in beiden Geräten ein 4Mhz-Quarz)
- Eine stabilisierte 5V Spannungsversorgung

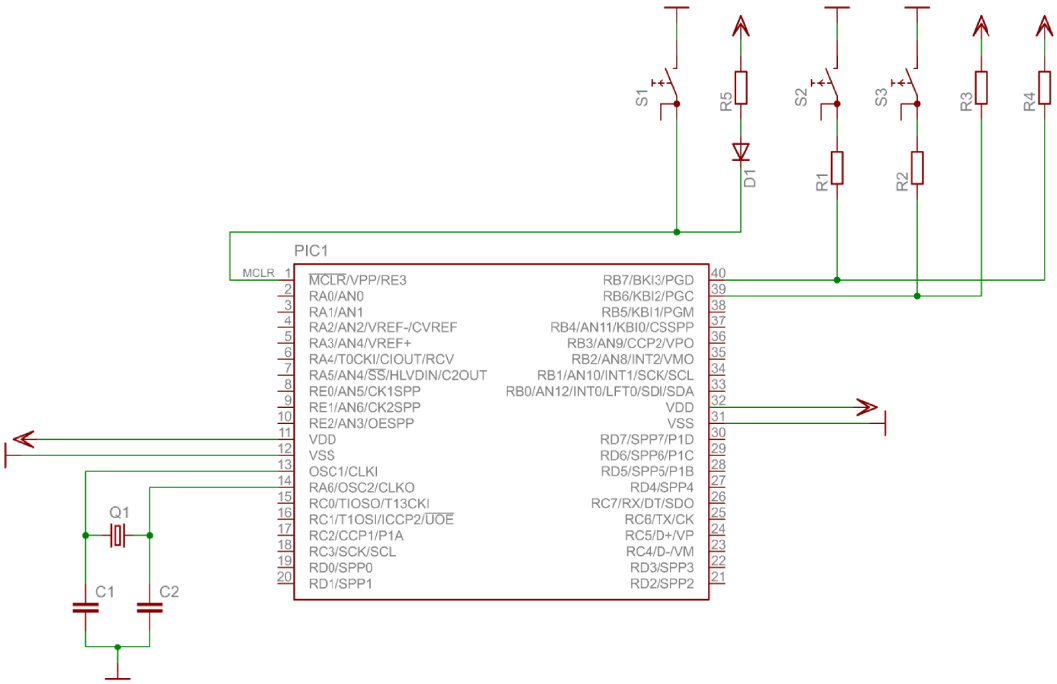

Abb. 1: Minimalkonfiguration für den von mir verwendeten PIC

4.1 Der Datenlogger

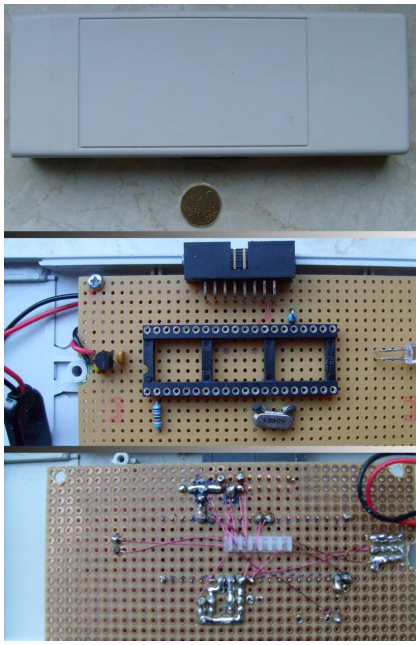

Abb. 2: Datenlogger von oben / unten, darüber das Gehäuse

Der Datenlogger hat neben der Standardkonfiguration noch einen externen Anschluss für einen Sensor (SV1) und einen I2C-Speicherchip (IC1) zur Erweiterung des Speicherplatzes von intern 256 Byte um weitere 128 Kbit (16 KB).

Der Datenlogger wird mit einem 9V-Akku betrieben. Da der PIC 5 V braucht, habe ich einen Spannungsregler eingebaut, der die Spannung herunter regelt. Das Gerät befindet sich in einem kleinen Gehäuse, damit man es leicht transportieren kann. Es gibt keine von außen zugänglichen Bedienelemente. Alle Funktionen (Sensorbetrieb, Datenkommunikation, Programmieren des PIC, Laden des Akkus) werden über verschiedene Anschlussverbinder gesteuert. Den vollständigen Schaltplan zeigt Abb. 3.

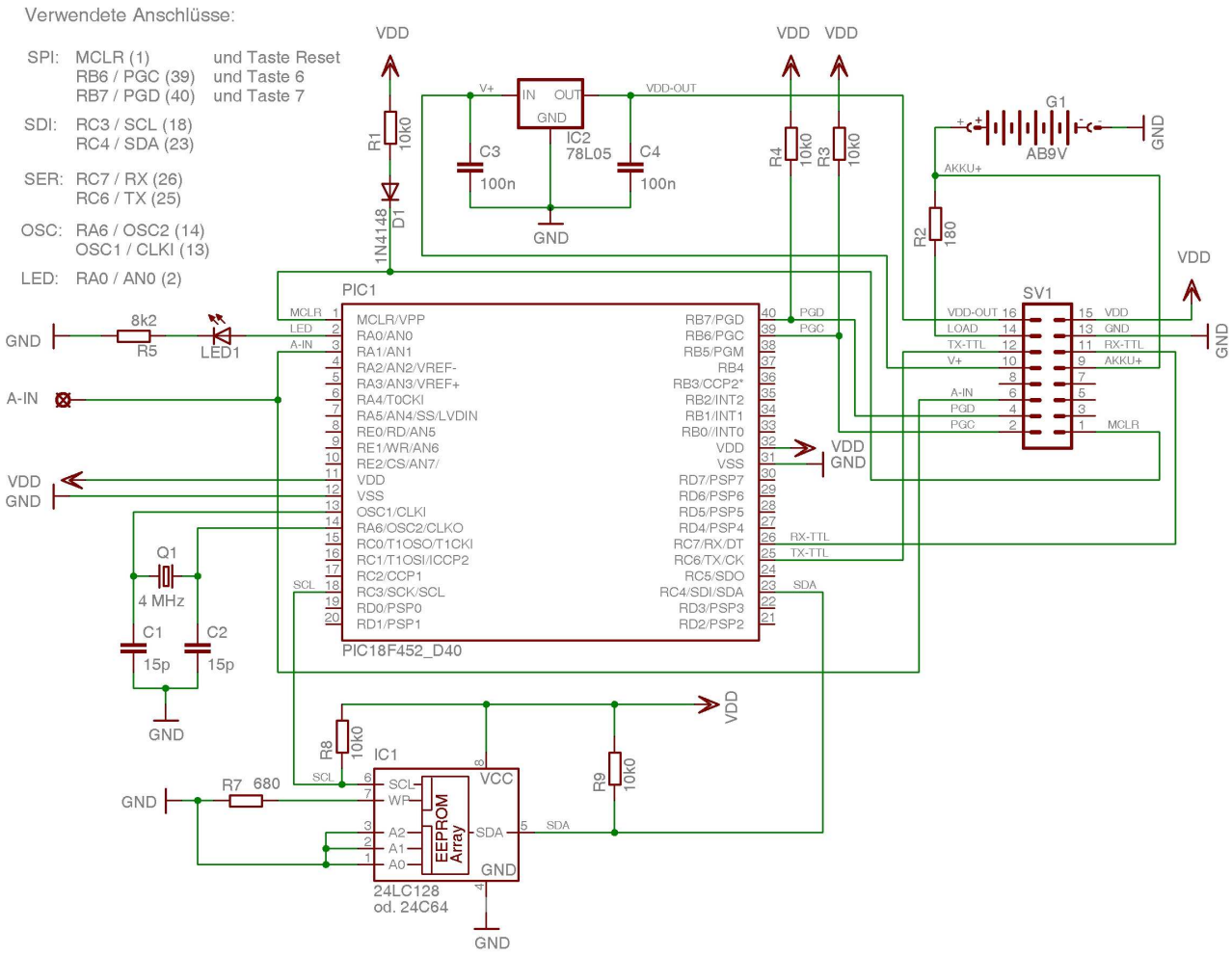

Abb. 3: Vollständiger Schaltplan des Datenloggers

Wenn ein Sensor am externen Sensor-Eingang angeschlossen wird, schaltet sich der Datenlogger automatisch über zwei Brücken zwischen den Pins 9 und 10 bzw. 15 und 16 ein. Das Signal wird über Pin 6 an einen der Analog-Inputs des PICs geführt.
Um eine Verbindung mit dem Bedienteil aufzubauen, ist am Sensorsteckplatz eine USART-Schnittstelle (Schnittstelle zur seriellen Datenübertragung, Pins 11 und 12) vorgesehen. In diesem Fall erfolgt die Stromversorgung vom Bediengerät über die Pins 13 (GND) und 15 (VDD). Wenn der Datenlogger die Kommunikation mit dem Bedienteil hergestellt hat, kann man ihn nun anschließend damit konfigurieren.
Zur Programmierung des PICs im Innern des Gerätes kann das LPT-Interface angeschlossen werden, dieses steuert die SPI-Schnittstelle (Pins 1 – 3) und stellt ebenfalls die benötigten Versorgungsspannungen bereit.
Zum Laden des Akkus kann eine beliebige Spannungsquelle (12 V) an die Pins 13 und 14 angeschlossen werden.

4.2 Das Bedienteil

Das Bedienteil ist im Prinzip genauso gebaut wie der Datenlogger, verfügt aber über einige zusätzliche Erweiterungen.
Statt eines Akkus verfügt das Gerät über ein eigenes Netzteil, mit dem auch der Akku des Datenloggers geladen werden kann.

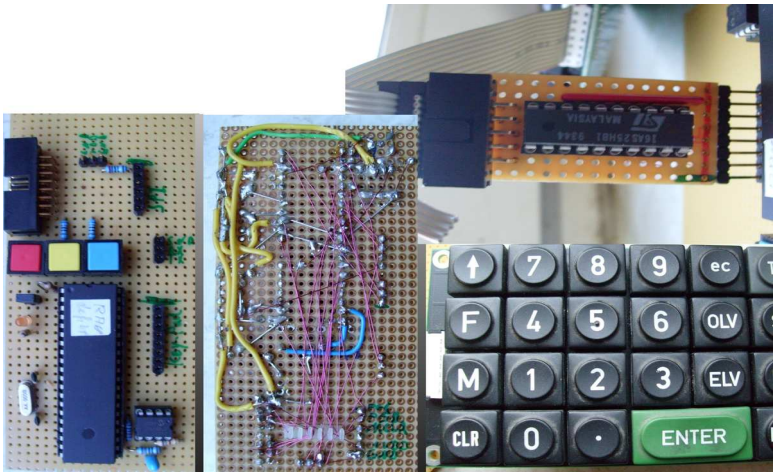

Abb. 4: Bedienteil des Datenloggers

Zur Benutzerführung enthält es außerdem ein zweizeiliges LC-Display, das die alphanumerische Anzeige von bis zu 16 Zeichen pro Zeile erlaubt. Die Bedienung erfolgt über ein Tastenfeld, das über 23 Tasten verfügt.
Den vollständigen Schaltplan findet man in der nachfolgenden Abb. 5.

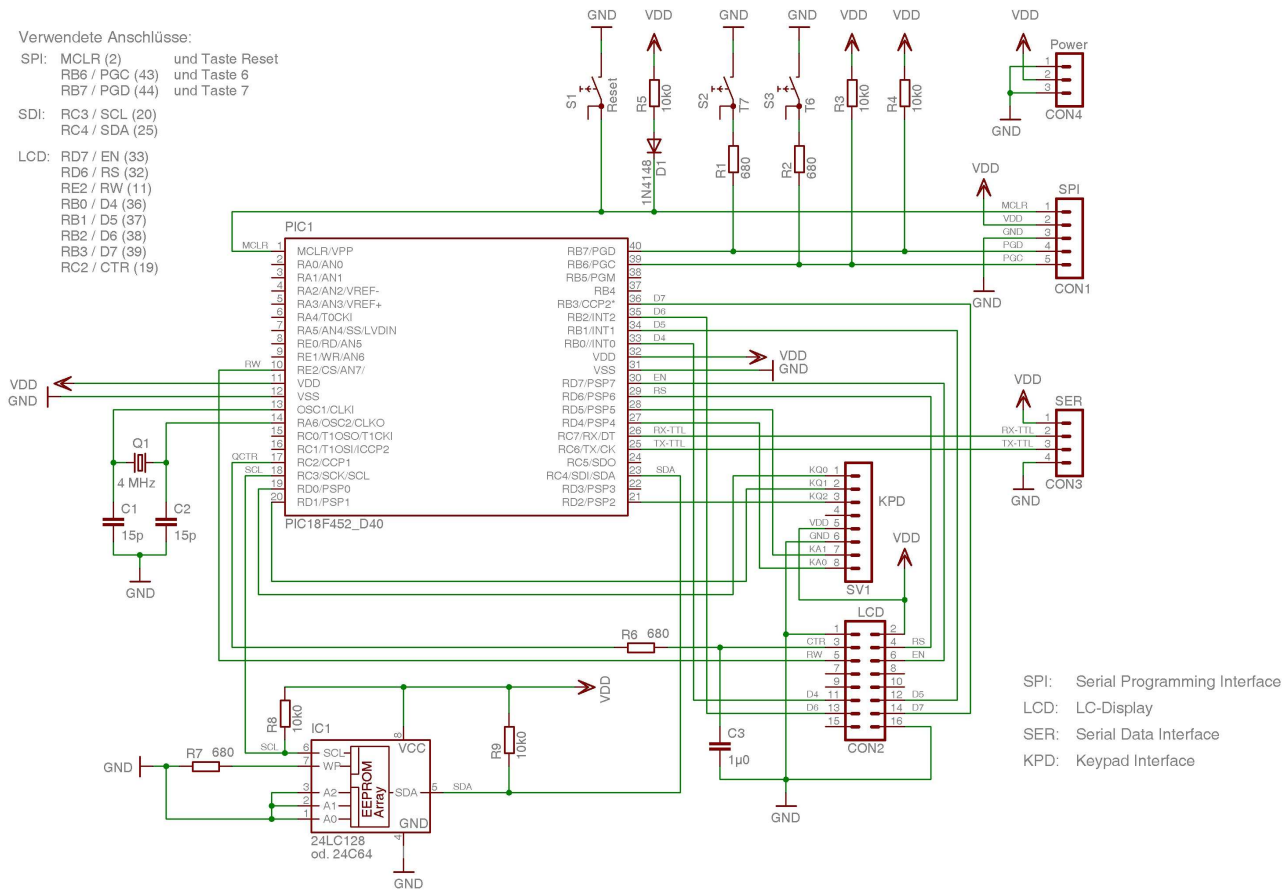

Abb. 5: Schaltplan des Bedienteiles

4.2.1 Das Keypad

Um das Keypad mit dem PIC zu abzufragen, wird ein GAL (ein programmierbarer Logikbaustein) verwendet. Es wandelt die Signale, die es vom Keypad bekommt, in 3-Bit kodierte digitale Signale um, die den Wert einer Taste in einer Reihe wiedergeben. Der PIC fragt binär kodiert (KA0, KA1) jede der vier Reihen einzeln ab und bekommt dann ein binäres Signal (KQ0 ... KQ2) zurück. Ist dieses 0b000, ist keine Taste gedrückt, sonst enthält der Wert die Tastennummer. Die fünf Bits (KA0, KA1, KQ0, KQ1 und KQ2) enthalten somit für jede Taste einen eindeutigen Binärcode.

Um das Keypad mit dem PIC abzufragen, verwende ich im C-Programm eine Interruptroutine, die auf dem Timer 1 des PIC aufsetzt:

```
void InterruptHandlerHigh ()
{
      if (INTCONbits.TMR0IF) {
         if (key.rdy == 0) {
            zeile++;                          Zeilenzähler um 1 erhöhen.
            if (zeile == 1) {                 Wenn der Zeilenzähler 1 ist:
               PORTDbits.RD4 = 0;             Signal an das GAL, um Zeile 1 abzufragen.
```

```
            PORTDbits.RD5 = 0;              Signal an das GAL, um Zeile 1 abzufragen.
         }
         if (zeile == 2) {                  Wenn der Zeilenzähler 2 ist:
            PORTDbits.RD4 = 1;              Signal an das GAL, um Zeile 2 abzufragen.
            PORTDbits.RD5 = 0;              Signal an das GAL, um Zeile 2 abzufragen.
         }
         if (zeile == 3) {                  Wenn der Zeilenzähler 3 ist:
            PORTDbits.RD4 = 0;              Signal an das GAL, um Zeile 3 abzufragen.
            PORTDbits.RD5 = 1;              Signal an das GAL, um Zeile 3 abzufragen..
         }
         if (zeile == 4) {                  Wenn der Zeilenzähler 4 ist:
            PORTDbits.RD4 = 1;              Signal an das GAL, um Zeile 4 abzufragen.
            PORTDbits.RD5 = 1;              Signal an das GAL, um Zeile 4 abzufragen.
         }
         if ((PORTD & 0x07) != 0) {         Wenn das GAL ein Signal zurückgibt,
            key.code = PORTD & 0b00000111;  ist key.code gleich der Tastenspalte auf dem Keypad
            key.wcnt = 0;
         }
         if (key.code != -1) {
            if (++key.wcnt >= key.wait) key.rdy = 1;
         }
      }
      if (zeile > 4) zeile = 1;             Wenn Zeilenzähler Zeile 4 erreicht hat zurücksetzten.
      WriteTimer0(TimerHL);                 Den Interupttimer zurücksetzten.
      INTCONbits.TMR0IF = 0;                Timer-Interruptflag zurücksetzten.
   }
}
```

Ein weiteres Codesegment im Hauptprogramm führt nun je nach gedrückter Taste ein bestimmtes Ereignis aus:

```
   while (1) {                              Endlosschleife
      ...                                   (weitere Codeabschnitte, z.B. USART-Abfrage)
      if (KeyPressed()) {                   Wenn eine Taste auf dem Keypad gedrückt wurde:
         keypadkeycode = GetKey()+(zeile*10);  Keypadtastencode  bilden (Taste 1 Zeile 2 = 21)
         switch (keypadkeycode) {           Mehrfach-Auswahl
         case 12:                           z.B. wenn der Tastecode 12 ist:
            LCD_strOutROM (1, 1, "Taste 12");
            LCD_strOutROM (2, 1, "wurde betätigt.");
            break;
         }
      }
   }
```

4.2.2 Das Display

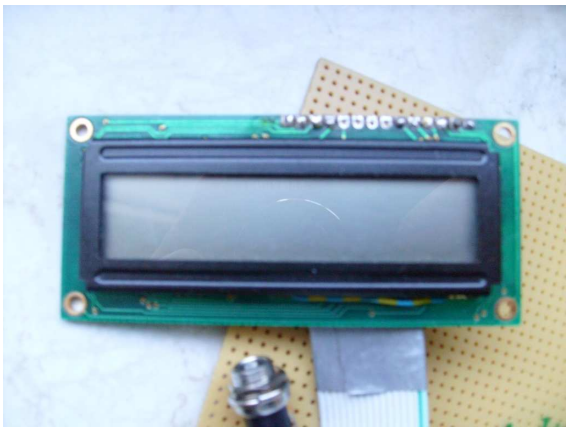

Abb. 6: LC-Display des Bediengerätes

Das Display ist direkt mit dem PIC verbunden. Es wird im 4-Bit-Modus angesprochen und verwendet einen Industriestandard (HD44780) für die Ansteuerung.
Um den Kontrast für das Display einstellen zu können, muss dazu ein Timer konfiguriert werden, der über den damit verbundenen Pulsweitenmodulator die gewünschte Spannung liefert.:

```
OpenTimer2(TIMER_INT_OFF & T2_PS_1_4 & T2_POST_1_8); Timer für Kontrasteinstellung.
OpenPWM1(175);                                       Pulsweitenmodulator einstellen.
SetDCPWM1(15);                                       Kontrast einstellen.
LCD_initialise ();                                   LCD initialisieren.
```

Dann kann man z.B. mit folgendem Befehl leicht Informationen auf dem Display ausgeben:

```
LCD_strOutROM (1, 1, "Hello World!");        Gibt „Hello World“ auf Zeile 1 ab Spalte 1 aus.
```

5 Die Kommunikation zwischen Bedienteil und Datenlogger

Die Kommunikation basiert auf dem Client – Server Prinzip. Dabei übernimmt das Bediengerät sinnvoller Weise die Rolle des Servers und der Datenlogger die des Clients.
Zur Kommunikation verwende ich die serielle Schnittstelle des PICs (USART). Weil die PICs direkt miteinander verbunden sind, kann ich auf die sonst üblichen Pegelwandler verzichten.
Damit man die USART-Schnittstelle benutzen kann, muss man sie vorher in beiden Geräten einstellen und konfigurieren:

```
OpenUSART(USART_TX_INT_OFF &          Interruptroutine Senden ausschalten.
          USART_RX_INT_OFF  &         Interruptroutine Empfangen ausschalten.
          USART_ASYNCH_MODE &         Asynchroner USART-Modus.
          USART_EIGHT_BIT   &         8-bit Modus.
          USART_CONT_RX     &
          USART_BRGH_HIGH,
          25                          Taktteiler für 2400 baud Geschwindigkeit.
    );
```

Damit die beiden Geräte wissen, ob sie Daten bekommen haben, gibt es ein Flag,

```
PIR1bits.RCIF.
```

Wenn der PIC neue Daten empfangen hat, steht dieses Flag auf „1“.

Wenn man diese Information abgerufen hat, muss man es manuell wieder auf „0“ zurücksetzen, damit ein weiteres Zeichen empfangen werden kann.

Durch das Flag kann wieder eine Art Interruptsteuerung programmiert werden. Im Datenlogger als auch im Bediengerät könnte diese so aussehen:

```
while(1){                              Endlosschleife.
   ...                                 (weiterer Code, z.B. Keypad oder Messungen)
   if(PIR1bits.RCIF == 1){             Wenn neue Daten empfangen wurden:
      usartd = ReadUSART();            Variable „usartd“ erhält den Wert der USART-Daten.
      PIR1bits.RCIF = 0;               Flag zurücksetzten.
      ...                              (weiterer Code: Auswertung des empfangenen Zeichens)
   }
}
```

Damit nun das Bedienteil eine Verbindung zum Datenlogger aufbauen kann, muss der Datenlogger antworten.

Das Bedienteil sendet zum Aufbau der Verbindung die Ziffer “7“ (hier wäre aber auch jedes andere Zeichen möglich). Ist der Datenlogger angeschlossen, dann antwortet er, indem er eine „8“ zurück sendet.

Datenlogger-Code zum Aufbau der Kommunikation:

```
if (usartd == 7){                      Wenn die Ziffer 7 empfangen wurde:
   usartd=0;                           Daten zurücksetzten.
   WriteUSART (8);                     Sende eine 8 zurück.
}
```

Wenn das Bedienteil dann diese „8“ empfangen hat, weiß es, dass eine Verbindung möglich und aufgebaut ist. Entsprechend kann auch für jedes andere empfangene Zeichen ein entsprechender Code vorgesehen werden.

6 Anwendung – Erfahrungen

Der Datenlogger ist multifunktional, denn man kann ihn in vielen Bereichen anwenden, z.B. in der Medizin, bei naturwissenschaftlichen Beobachtungen usw. Eine Anwendung hat er bereits im Bereich CO_2 gefunden, indem eine Mitschülerin und ich einen CO_2-Datenlogger gebaut haben. Mit diesem CO_2-Datenlogger kann man mehrere Stunden Messwerte in regelmäßigen Abständen (z.B. alle fünf Minuten) speichern und später mit dem PC wieder auslesen. Dann kann man die ausgelesene Datei in Excel importieren und dann z.B. den Tagesverlauf des CO_2-Gehaltes in Klassenräumen bestimmen und vergleichen.

In meinem letzten Projekt hatte ich bereits einen einfachen Datenlogger gebaut, dieser hatte jedoch einen sehr kleinen Speicher und war nicht manipulationssicher. Außerdem konnte er nicht transportiert werden, da er nicht, wie der neue, mit Akku betrieben werden konnte.

7 Quellen

[1] Datenblatt der Firma Microchip (PIC18F452.pdf)

[2] MPLAB, Download von Microchip.com (C18 Student-Edition)

[3] MPLAB_C18_Libraries.pdf, Download von Microchip.com (C18 Student-Edition)

[4] IC-Prog, Download von IC-Prog.com (Version 1.05 C)

Schaltpläne erstellt mit Eagle 4.16 r2 Light