Portable Programmierung für Mikrocontroller

Franz Fiala, N, TGM LIT-76, DSK-407: PAR.C, GL.*, PRO.EXE

Programmentwicklung für Mikrocontroller wird bereits selbstverständlich mit Hochsprachen, zumeist in C durchgeführt. Die dazu erhältlichen Compiler erzeugen sehr kurzen und effektiven Kode, vergleicht man etwa mit den ersten Versuchen in diese Richtung PC·NEWS-24/Seite 16ff. Es besteht kaum Bedarf für Assemblerroutinen. Lediglich der Preis für ein vollständiges Entwicklungspaket bestehend aus C-Compiler, Assembler, Linker, Debugger, Echtzeitkern und den erforderlichen Bibliotheken und Utilities wird manchen vom Einsatz abschrecken. Glücklicherweise sind die Preise für Mehrfachinstallationen für den Unterrichtsbetrieb angemessen, sodaß einem Einsatz im Labor eigentlich nichts mehr im Wege stehen sollte.

Die zur Verfügung stehenden Werkzeuge Compiler, Assembler und Debugger liefern zwar ausgezeichneten Kode, die sonstige Umgebung kann sich aber nicht mit dem Komfort herkömmlicher **PC-Compiler** messen.

Die folgenden Programme wurden für den PC mit dem BORLAND-C-Compiler Version 3.1 und für den 8051 mit dem KEIL-Compiler C51 geschrieben. Sinngemäß können die Programme aber auch für andere Compiler angewendet werden.

Die Sprache C ist für praktisch alle Prozessortypen, für jeden Rechner und in vielen Varianten erhältlich und eignet sich dank des ANSI-Standards sehr gut für portable Programme. Portabel heißt also, Programme so zu schreiben, daß sie sowohl mit Compiler A als auch mit Compiler B, sowohl auf der CPU X als auch CPU Y ablaufen.

Die ANSI-Sprachdefinition enthält natürlich keinerlei Bezug zu einer bestimmten Hardware oder zu einer bestimmten CPU. Jede Implementierung eines ANSI-Compilers für eine bestimmte CPU, für ein bestimmtes Betriebssystem bedingt daher eine Reihe ungenormter Schlüsselwörter oder Funktionen, die von der Hardware abhängig sind.

Der Schlüssel zur portablen Compilierung (derselbe Code läuft mehreren Prozessoren) sind die Präprozessoranweisungen `#ifdef..#endif` und andere, mit denen nicht portierbare Schlüsselwörter und Funktionen von Fall zu Fall neu definiert oder umdefiniert werden. Jeder Compiler definiert eigene Makros, die zu seiner Identifikation mit #ifdef abgefragt werden können, z.B. bedeutet

```
#ifdef __C51__                  /* KEIL      */
#ifdef __TURBOC__               /* BORLAND   */
#ifdef _MSC_VER                 /* MICROSOFT */
```

, daß die folgenden Zeilen nur dann kompiliert werden, wenn das Makro `__C51__` oder `__TURBOC__` oder `_MSC_VER` definiert ist, und das ist nur dann der Fall, wenn der KEIL-, BORLAND- oder MICROSOFT-Compiler den Text bearbeitet.

Ein portierbar geschriebenes Programm besteht, (wenn man nicht von vornherein beim Entwurf auf diese Eigenschaften Rücksicht genommen hätte) aus einer relativ zerhackten Folge von `#ifdef..#endif` -Anweisungen, die den Kode sehr schwer lesbar machen. Die im folgenden beschriebene Vorgangsweise zeigt, wie man die Lesbarkeit weitgehend erhalten kann und mit einem Minimum an bedingter Kompilierung auskommt.

Programmstruktur

Im Prinzip werden Programme für Mikrocontroller nicht in einem einzigen Hauptprogramm ausgeführt, sieht man von kleinen Experimenten ab. Komplexere Probleme bestehen aus Funktionen, die **hardwarenahe Kommunikationsaufgaben** zu erfüllen haben (Interaktion mit den IO-Strukturen der Schaltung, wie Ports, Timer usw.) und aus Funktionen, die **anwendungsnahe Aufgaben** übernehmen und die die eigentliche Aufgabe der gesamten Konstruktion auszuführen haben.

Nur bei sehr einfachen Problemen tritt eine Anwendung unmittelbar mit der Hardware in Kontakt.

In diesem Sinne unterscheidet sich ein Mikrocontroller-Programm kaum von einem Betriebssystems eines PC. Die hardwarenahen Programmteile entsprechen dem BIOS und dessen Erweiterungen, anwendungsnahen Programmteile entsprechen dem Anwenderprogramm. Der Unterschied zum PC ist, daß man im Mikrocontroller auch das BIOS selbst schreiben muß.

Der kommunikative Funktionenblock besteht allgemein aus niederwertigen Funktionen (zur Kommunikation mit den Ports, Interruptroutinen...) und aus höherwertigen Funktionen, die auf die niederwertigen Funktionen zugreifen. Im PC wäre das die Trennung in BIOS und Betriebssystem. Das Betriebssystem benützt im allgemeinen BIOS-Funktionen, reichert sie aber um wichtige Eigenschaften an. Der Aufbau gleicht also durchaus dem eines Personalcomputers, wenn auch in stark vereinfachter Form.

Solange man sich mit der Programmierung der Ports und Register beschäftigt, ist auch eine bessere Benutzerführung nicht erforderlich, da hilft der Debugger weit mehr. Wenn aber diese low-level-Routinen „stehen“ und man sich mehr dem eigentlichen Problem zuwendet, hat man immer weniger mit den Strukturen des Mikrocontrollers zu tun, denn die bereits geschriebenen Routinen übernehmen die Kommunikation mit der Hardware. Das eigentliche Problem, z.B. eine Ampelsteürung kann aber unter TURBO-C oder VISUAL-C genauso gut ausformuliert werden wie unter einem Mikrocontroller-C. Darüberhinaus können alle kleinen und großen Hilfen zum Debuggen angewendet werden und die Logik des Programms auf Herz und Nieren getestet werden, bevor man es mit dem Mikrocontroller-C zur weiteren Bearbeitung übergibt.

Dasselbe Programm muß also sowohl mit dem Mikrocontroller-C als auch mit dem komfortablen PC-C kompilierbar sein. Dabei tritt eine kleine Schwierigkeit auf: Um alle Eigenheiten des Mikrocontrollers durch die Sprache C ausdrücken zu können, benötigt es einige Erweiterungen. (Das ist übrigens auch beim C für den PC der Fall, denn dort gibt es z.B. Erweiterungen wie `near, far, huge` usw., die nicht urprünglich in C enthalten sind.) Im C51 von KEIL sind es folgende neue Schlüsselwörter:

```
bit, sbit, sfr, sfr16, data, idata, pdata, bdata, xdata, code, reentrant, interrupt
```

Es gibt drei Möglichkeiten, mit diesen, einem „gewöhnlichen“ C-Compiler unbekannten, Begriffen umzugehen.

- ignorieren
- ersetzen
- nichts tun

Im ersten Fall wird einfach das neue Schlüsselwort entfernt, im zweiten Fall durch ein anderes, gültiges ersetzt, im dritten Fall kollidiert dieses Schlüsselwort mit einem anderen, bereits definierten, sodaß eine individülle Behandlung nötig ist. Damit diese Überlegungen nicht bei jeder einzelnen Programmzeile neu angestellt werden müssen, werden sie in einer eigenen Headerdatei `pc.h` zentral erledigt:

```
/* pc.h */
#define bit   unsigned char
#define sbit  unsigned char
#define sfr   unsigned char
#define sfr16 unsigned int
#define data
#define idata
#define pdata
#define bdata
#define xdata
#define code
#define reentrant
```

Wie man sieht, wurden die C51-Typen `bit, sbit, sfr` und `sfr16` durch `unsigned char` und `unsigned int` ersetzt, die anderen Begriffe werden in einem PC-Compiler ignoriert, Code und Daten jeder Art erscheinen in demselben Speicher.

Egal ob Compiler für den PC oder für den Mikrocontroller, beide kennen nicht die vielen besonderen Bezeichnungen der Mikrocontroller-Register. Besondere Headerdateien informieren sie darüber. Für jede Mikrocontroller-Type gibt es beim KEIL-Compiler im allgemeinen eine andere Header-Datei. Nachfolgend ist am Beispiel des Kontrollers 8051 die veränderte Datei `REG51.H` abgedruckt.

```
#ifndef __REG51_H
#define __REG51_H

#ifndef __C51__
#include <pc.h>
#endif

#ifndef __ABSACC_H
#include <absacc.h>
#endif

/* (c) Copyright KEIL ELEKTRONIK GmbH.  1989,
       All rights reserved. */
/*  Register Declarations for 8051 Processor */

sfr P0   = 0x80; /*  BYTE Register  */
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0  = 0x8A;
sfr TL1  = 0x8B;
sfr TH0  = 0x8C;
sfr TH1  = 0x8D;
sfr IE   = 0xA8;
sfr IP   = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;

sbit CY   = 0xD7; /*  PSW   */ /*  BIT Register  */
sbit AC   = 0xD6;
sbit F0   = 0xD5;
sbit RS1  = 0xD4;
sbit RS0  = 0xD3;
sbit OV   = 0xD2;
sbit P    = 0xD0;

sbit TF1  = 0x8F; /* TCON */
sbit TR1  = 0x8E;
sbit TF0  = 0x8D;
sbit TR0  = 0x8C;
sbit IE1  = 0x8B;
sbit IT1  = 0x8A;
sbit IE0  = 0x89;
sbit IT0  = 0x88;

sbit EA   = 0xAF; /*  IE   */
sbit ES   = 0xAC;
sbit ET1  = 0xAB;
sbit EX1  = 0xAA;
sbit ET0  = 0xA9;
sbit EX0  = 0xA8;

sbit PS   = 0xBC; /*  IP   */
sbit PT1  = 0xBB;
sbit PX1  = 0xBA;
sbit PT0  = 0xB9;
sbit PX0  = 0xB8;

sbit RD   = 0xB7; /*  P3  */
sbit WR   = 0xB6;
sbit T1   = 0xB5;
sbit T0   = 0xB4;
sbit INT1 = 0xB3;
sbit INT0 = 0xB2;
sbit TXD  = 0xB1;
sbit RXD  = 0xB0;

sbit SM0  = 0x9F; /*  SCON  */
sbit SM1  = 0x9E;
sbit SM2  = 0x9D;
sbit REN  = 0x9C;
sbit TB8  = 0x9B;
sbit RB8  = 0x9A;
sbit TI   = 0x99;
sbit RI   = 0x98;
#endif
```

Am Beginn dieser Header-Datei kann man einige Zeilen erkennen, die nur für das Kompilieren mit dem PC-Compiler eingefügt wurden und die im Originalzustand nicht enthalten sind:

```
#ifndef __REG51_H          verhindert mehrfache Inklusion
#define __REG51_H

#ifndef __C51__            nur inkludieren,
                           wenn mit PC-Compiler gearbeitet wird
#include <pc.h>
#endif
```

Drei Beispiele veranschaulichen die portable Programmierung.

Die Beispiele wurden so gewählt, daß sie vollständig am PC simulierbar sind, was aber nicht mit allen Aufgabenstellungen in dieser Form funktionieren muß.

1. CENTRONICS-Schnittstelle (PAR.C)

Am PC ist die CENTRONICS-Schnittstelle durch die parallelen Schnittstellenkarten implementiert. Programme im PC greifen auf den Drucker üblicherweise über höherwertige Betriebssystemfunktionen zu. In diesem Beispiel müssen aber die einzelnen Bits individüll gesteürt werden. Der Einfachheit halber wird kein Handshake implementiert. Eine Zeitverzögerung sorgt dafür, daß der Drucker die Zeichen verarbeiten kann.

	µC	PC
Daten	P1	0x3f8
Strobe	P3.2	0x3fa, Bit 0

Das Programm wird zur Gänze am PC vorbereitet und auch getestet. Dabei muß man am PC eigentlich keinerlei Harwareveränderungen vornehmen, der angeschlossene Drucker ist ausreichend.

Im PC wird eine in der Bibliothek vorhandene Routine `delay()` vorausgesetzt. Diese Routine wird im Mikrocontroller-C als eine Softwareverzögerung nachempfunden. Der Zählerwert in der `for`-Schleife im Mikrocontroller wird durch Experiment bestimmt.

Man sieht an diesem Beispiel sehr schön, daß lediglich die Kommunikationsroutinen für PC und UC verschieden sind, hier die Funktion `print_char().` Die Textausgabe mit `print_text()` ist bereits frei von bedingter Kompilierung, ihre Richtigkeit kann bereits vom TURBO-Compiler allein gestetet werden.

Hält man sich daran, alle niederwertigen Funktionen die Port- und Registerkommunikation durchführen zu lassen, steht einem komfortablen und zuverlässigen Test durch den PC-Compiler nichts mehr im Wege.

Beim Testen des Programms mit einem Probetext muß man darauf achten, ob man einen Zeilendrucker oder einen Seitendrucker ansteuert. Je nachdem muß das letzte Zeichen des Testtextes 13 (CR) oder 12 (FF) sein.

Testen des Programms:

Bevor das Programm mit dem Drucker getestet wird, kann man das auch in TURBO-C und mit dem dScope ausführen. In TURBO-C werden einfach `printf()`-Anweisungen in die innerste Funktion `print_char(9)` eingefügt, die den auszugebenden Text am Bildschirm abbilden. Im dScope verwenden wir eine SIGNAL-Funktion, die das Geschehen mitverfolgt und jeweils bei einer Flanke am Strobe-Bit das Ergebnis in das EXE-Fester schreibt, also genau den auszugebenden Text:

```
Textausgabe mit Zeilendrucker
Textausgabe mit Seitendrucker
```

Listing

```
/*
 * PAR.C
 * =====
 * Vereinfachte Ausgabe von Zeichen an eine parallele
 * CENTRONICS-Schnittstelle
 */

#define TEST_PAR

#include <reg51.h>

#ifdef __C51__

#define CENTDATA P1      /* Daten zum Drucker */
#define CENTCONT P3      /* Steuerport */
#define CENTSTRB 2       /* Bitnummer des Strobebit */

void delay_ms1(void)
{
#define DELAY_MS1 57
  int ms1;
  for (ms1=0; ms1<DELAY_MS1; ms1++) ;
}

void delay(unsigned long ms)
{
  int t;
  for (t=0; t<ms; t++ )
    delay_ms1();
}

#else /* PC ist am Werk */
```

```
#include <dos.h>
#include <stdio.h>

#define CENTDATA 0x378   /* Daten zum Drucker */
#define CENTCONT 0x37A   /* Steuerport */
#define CENTSTRB 0       /* Bitnummer des Strobebit */

#endif

void print_char(char c);
void print_text(char *t);
void print_nl(void);
void print_np(void);
void print_textnl(char *t);
void print_textnp(char *t);

void print_char(char c)
{
#ifdef __C51__
  unsigned char control;
  CENTDATA=c;                   /* Daten ausgeben */
  control=CENTCONT;             /* Strobebit setzen */
  control |= 1<<CENTSTRB;
  CENTCONT=control;
#else
  unsigned char control;
  outportb(CENTDATA,c);         /* Daten ausgeben */
  control=inportb(CENTCONT);    /* Strobebit setzen */
  control |= 1<<CENTSTRB;
  outportb(CENTCONT,control);
  printf("%c",c);               /* Kontrolle */
#endif
  delay(1);                     /* warten */
#ifdef __C51__
  control=CENTCONT;             /* Strobebit löschen */
  control &= ~(1<<CENTSTRB);
  CENTCONT=control;
#else
  control=inportb(CENTCONT);    /* Strobebit löschen */
  control &= ~(1<<CENTSTRB);
  outportb(CENTCONT,control);
#endif
  delay(100);
}

void print_text(char *t)
{
  while (*t)
  {
    print_char(*t);
        t++;
  }
}

void print_nl(void)
{
  print_text("\x0d\x0a");
}

void print_np(void)
{
  print_text("\x0c");
}

void print_textnl(char *t)
{
  print_text(t);
  print_nl();
}

void print_textnp(char *t)
{
  print_text(t);
  print_np();
}

#ifdef TEST_PAR
void main(void)
{
  print_nl();
  print_textnl("Textausgabe mit Zeilendrucker");
  print_textnp("Textausgabe mit Seitendrucker");
}
#endif
```

2. Pulsdauermodulation (GL.C)

Ein etwas aufwendigeres Beispiel zeigt, wie man den Timer im PC auch den Timer des Mikrocontrollers simulieren lassen kann.

Das zu lösende Problem ist die Herstellung einer Pulsdauermodulation mit variablem Tastverhältnis mit einem 8051 aus einer gegebenen Gleichspannung. Dieses Problem trat bei der Konstruktion eines Schaltnetzteiles auf. In der endgültigen Schaltung wurde die Gleichspannung aus einem ADC gewonnen, in diesem Beispiel wird angenommen, daß die Spannung an Port 0 als Digitalwert anliegt. Zur Simulation am PC wird die Gleichspannung durch einen Tastendruck an einer der Zifferntasten vorgegeben, die pulsdauermodulierte Spannung erscheint an einem Portpin der parallelen Schnittstelle und kann dort oszillografiert werden.

Der Timer generiert 500 Interrupts pro Sekunde und bestimmt den Pulsabstand. Bei jedem Interrupt wird der Impuls gestartet. Zur präzisen Bestimmung der Pulslänge, sollte man einen zweiten Timer starten, der die Impulslänge bestimmt und bei dessen Ablauf der Impuls beendet wird. Hier wird zur Vereinfachung der Zählerstand des Pulsabstandstimers abgefragt und daraus die Impulslänge bestimmt.

Um diese Vorgänge am PC nachbilden und testen zu können, muß der Timer 0 verwendet werden. Dieser generiert in der Grundeinstellung Interrupts im Abstand von 52 ms. In dieser Anwendung soll er es aber alle 2 ms tun. Daher wird der Timer für die Dauer des Tests umprogrammiert, und die interne Uhr bleibt stehen (oder läuft viel schneller, jenachdem, wie man die Sache angeht).

Das dieses Beispiel den hier vorgegebenen Rahmen sprengt, wurde das Programm ebenso wie das folgende gemeinsam mit der dazugehörigen Beschreibung auf die Begleitdiskette PCN-DSK-407 ausgelagert.

3. Fernwirken (PRO.EXE, LIT-76)

Im Rahmen eines Abschlußjahrgangs der Abteilung für Elektronik für Berufstätige am TGM wurde in einer Klassenarbeit eine Fernwirkanlage realisiert, die über die serielle Schnittstelle kommuniziert. Das Problem bestand im Aufbau des seriellen Übertragungsprotokolls. In diesem Beispiel bietet sich ein Schichtenmodell mit folgenden Regeln an:

Eine Schicht kann (soll) nur mit der jeweils darunterliegenden Schicht kommunizieren aber sie kann nicht durch diese Schicht hindurchgreifen. Die Kommunikation kann über Variable oder über Funktionen erfolgen. Besser ist die modernere Kommunikationsform über Funktionen, d.h. die Variablen einer Schicht sind für die darüberliebenden Schichten nicht zugänglich. In C wird das durch die Bezeichnung `static` erleichtert. `static` verhindert, daß Funktionen oder Variablen über das Modul hinaus gültig sind.

Nur die oberste Schicht MES enthält ein Hauptprogramm. Die darunterliegenden Schichten sind nur Funktionssammlungen, die allein kein ablauffähiges Programm ergeben, sondern erst durch das Aufrufen durch die darüberliegnede Schicht einen Sinn erhalten.

Die Schichtung wird in C auf einfache Weise unterstützt. Um den Durchgriff von MES auf INT zu verhindern, genügt es, die Headerdatei INT.H nicht zu inkludieren.

```
Modul
MES : Messung,                                  MES.C
      Polling der seriellen Schnittstelle
      Polling des Protokollablaufs
      Hauptprogramm
-----------------------------------------------
PRO : Paketierung und Depaketierung             PRO.C  PRO.H
      Protokoll
-----------------------------------------------
INT : Zeichen senden, Zeichen empfangen         INT.C  INT.H
      ------------------------------------
      Interrupt-Service-Routinen
-----------------------------------------------
```

Kommunikation zwischen den Modulen