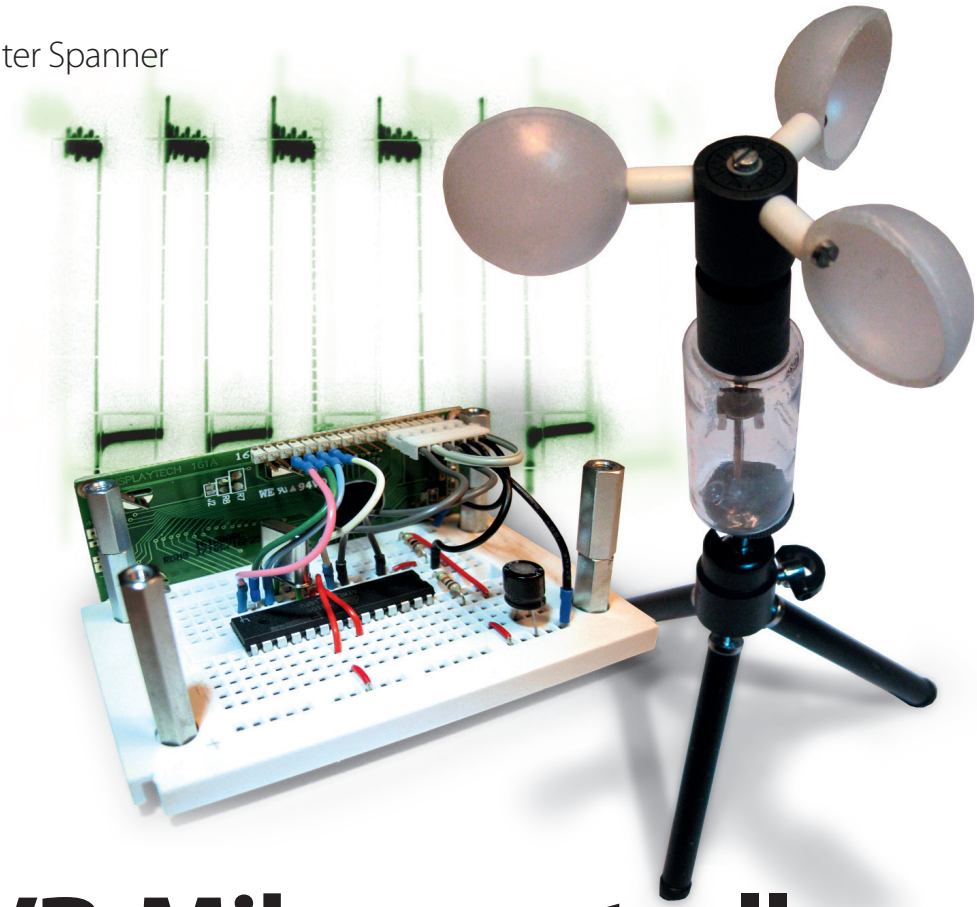
FRANZIS
PC & ELEKTRONIK

Dr. Günter Spanner



AVR-Mikrocontroller in C programmieren

Über 30 Selbstbauprojekte mit ATtiny13, ATmega8 und ATmega32

Vorwort

Im Vergleich zum Einstieg in die Elektronik ist bei der Mikrocontrollertechnik eine deutlich höhere Hürde zu überwinden. In der Elektronik kann man wirklich ganz elementar beginnen. Eine LED, ein Transistor, zwei Widerstände und eine Batterie genügen für einfache Experimente. Zwar ist die „Hardware" für einen Mikrocontrollereinstieg ebenfalls sehr einfach und kann nur aus dem Controller selbst, einem Widerstand und einer LED bestehen, allerdings sind zur Programmierung des Controllers mindestens noch ein PC und ein Programmierkabel erforderlich. Spätestens ab hier werden Sie eine gute Einführung zu schätzen wissen.

In der einführenden Literatur zum Thema wird meist zunächst der Mikrocontroller selbst eingehend dargelegt. Register, Speicher und Interrupts werden diskutiert. Es folgt eine formale Einführung in eine Programmiersprache wie Assembler, Bascom oder auch C. Erst in einem letzten, meist nicht sehr umfangreichen Teil werden, wenn überhaupt, nur sehr simpel gehaltene und wenig praxisrelevante „Anwendungen" gezeigt. Die Erfahrung lehrt, dass der Leser oft schon weit vor den Praxiskapiteln den Überblick verloren hat und aufgibt.

In diesem Buch sollen deshalb hochinteressante und zugleich lehrreiche Mikrocontrolleranwendungen im Vordergrund stehen. Die vollständig beschriebenen Projekte sind alle für den praktischen Einsatz in Labor und Alltag geeignet, neben dem Lerneffekt steht hier also insbesondere auch die Freude am Zusammenbau kompletter und nützlicher Geräte im Fokus. Dabei kann der Aufbau der Hardware durchgehend auf Laborsteckboards erfolgen. Dies ermöglicht die Herstellung aller Gräte mit sehr geringem Aufwand. Das Erproben der Anwendungen wird damit zum lehrreichen Vergnügen.

Darüber hinaus ist dieses Buch hervorragend als Handbuch geeignet. Ein umfassendes Inhalts- und Stichwortverzeichnis erleichtert das schnelle Auffinden einzelner Themen. Damit bleibt dieses Werk, selbst wenn längst schon viele der Projekte aufgebaut und erprobt wurden, als wertvolles Kompendium von größtem Nutzen.

Neben den eher ernsthafteren Applikationen wie Thermometern, Timern und Drehzahlmessern kommen auch Fun-Projekte wie ein vollautomatisierter Laser-Schießstand mit echtem Halbleiterlaser oder ein elektronisches Roulette mit Auslaufeffekt nicht zu kurz.

Mit anspruchsvolleren Geräten z. B. aus dem Bereich der Bioelektronik, wie einem elektronischen EKG mit Signalauswertung eines Brustgurts oder einem Fitness-Monitor mit USB-Anschluss, werden auch für Fortgeschrittene hochinteressante Projekte angeboten.

Als Programmiersprache wird durchgehend C verwendet. Dies ermöglicht zum einen den Einsatz von kostenlosen C-Compilern. Zum anderen wird dadurch die Wiederverwendung von Unterprogrammbibliotheken optimal unterstützt. Der klar strukturierte und modulare Aufbau der Software hält die Anwendungsprogramme kurz und übersichtlich. Im Rahmen der einzelnen Applikationen wird auf alle relevanten Elemente der Programmiersprache C und die zugehörigen Erweiterungsbibliotheken ausführlich eingegangen. In einem Kompendium (s. Kapitel 15) werden die wichtigsten speziell für die Mikrocontroller-Programmierung erforderlichen Sprechelemente erläutert, sodass sich auch Einsteiger tiefer in die Materie einarbeiten können.

Die vorgestellten einfachen und kostengünstigen Fertiggehäuse oder Eigenbauten aus Acrylglas und Aluminium können von jedermann problemlos aufgebaut werden. Die daraus entstehenden Geräte bestechen nicht nur durch ihre praxisgerechte Funktion und technische Perfektion, sondern auch durch ihr individuelles und gefälliges Design. Tipps & Tricks zu jedem Projekt runden die detaillierten Aufbaubeschreibungen ab.

Das Buch soll „Learning by Doing“ bestmöglich unterstützen. Es setzt auf deduktives Lernen anhand der dargestellten Projekte. Daher ist das Werk auch hervorragend als Praxis-Buch für

- Schulen
- Berufsschulen und auch
- Fachhochschulen und Universitäten

einsetzbar. Durch viele Anwendungen wie Geschwindigkeitsmessung, Digital-Voltmeter oder Rechteckgenerator ist das Buch als Anregung für Praktika oder Fach- und Studienarbeiten in den Naturwissenschaften bzw. im Natur- und Technikunterricht bestens geeignet.

Dr. Günter Spanner

München, Mai 2010

Korrekturen, Ergänzungen oder Anregungen werden gerne unter der Adresse elo@franzis.de angenommen.

Warnhinweise

1 Die Versuche sind ausschließlich für Batteriespeisung vorgesehen. Falls aufgebaute Geräte dauerhaft betrieben werden, dürfen nur geprüfte doppelt-isolierte Sicherheitsnetzgeräte verwendet werden, da bei einem Isolationsfehler eines einfachen Netzteils lebensgefährliche Spannungen an nicht-isolierten Bauelementen anliegen können.

2 Autor und Verlag übernehmen keinerlei Haftung für Schäden, die durch den Aufbau der beschriebenen Projekte entstehen.

3 Elektronische Schaltungen können elektromagnetische Störstrahlung aussenden. Verlag und Autor haben keinen Einfluss auf die technische Ausführung der in diesem Buch vorgestellten Schaltungen. Der Anwender der Schaltung ist daher selbst für die Einhaltung der relevanten Emissionsgrenzwerte verantwortlich.

Inhaltsverzeichnis

3 Programmier-Interfaces

Dem einfachen Aufbau von µC-Schaltungen steht der Aufwand für die Programmierung des Bausteins gegenüber. Hierfür ist wiederum eine eigene Hardware erforderlich. Dieser sogenannte Programmer muss aber nur einmal angeschafft und kann dann immer wieder eingesetzt werden. Die Klassiker unter den Programmern sind die Kits von ATMEL selbst, die in verschiedenen Versionen angeboten werden. Da sie aber oft keine reinen Programmer sind, sondern komplette Evaluationsboards, sind sie entsprechend teuer.

3.1 Hardware-Voraussetzungen

Deutlich günstiger sind reine Programmieradapter. Geräte mit USB-Schnittstelle sind als Bausatz bereits für einstellige Euro-Beträge erhältlich. Für die Projekte in diesem Buch sind diese Programmer vollständig ausreichend.

Daneben besteht noch die Möglichkeit, Programmer mit sehr einfachen Mitteln komplett selbst zu bauen [9]. Diese „Einfachstprogrammer“ haben jedoch zwei Nachteile: Zum einen können sie nur an seriellen oder parallelen PC-Ports betrieben werden. Diese sind an modernen PCs oder Laptops meist nicht mehr vorhanden. Zum anderen ist die Betriebssicherheit dieser Lösungen nicht sehr hoch. Weniger erfahrene Anwender sollten daher besser auf fertige Programmer zurückgreifen.

Eine weitere Möglichkeit für Einsteiger bieten die Lernpakete des Franzis-Verlags [7, 8]. Sie enthalten neben einem allgemein verständlichen Handbuch auch gleich den passenden Programmer. Allerdings ist dieser Programmer oft nur für die ebenfalls beiliegenden Mikrocontrollertypen (z. B. Tiny13) direkt einsetzbar. Will man andere Controllertypen programmieren, so lassen sich aber mit den erworbenen Kenntnissen aus den Lernpaketen leicht auch universell nutzbare Programmer aufbauen.

Die folgende Tabelle fasst die verschiedenen Programmertypen zusammen und bietet Hilfestellung bei der Auswahl einer geeigneten Variante.

Die zu den Programmern gehörenden Treiber können im Allgemeinen kostenlos von den Internetseiten der Hersteller heruntergeladen werden.

Tab. 3.1: USB-Programmer-Varianten

Programmer für µC	Schnittstelle zum PC	Löten erforderlich?	Kosten ca. (in €)	Notwendige Vorkenntnisse
Lernpaket Mikrocontroller	RS232 erforderlich	JA	50.-	Keine, da ausführliche Beschreibung im Handbuch
Lernpaket MSR mit dem PC	USB	NEIN	70.-	Keine, da ausführliche Beschreibung im Handbuch
„Einfachst-Programmer"	RS232 erforderlich	JA	5.-	Grundlegende Elektronikerfahrung empfehlenswert
USB-Programmer (Bausätze)	USB	JA, SMD!	5.-	Grundlegende Elektronikerfahrung empfehlenswert
ATMEL Evaluations-boards z. B. STK 500	Verschiedene Schnittstellen	NEIN	80.-	Keine, da ausführliche Beschreibung im Handbuch

Abb. 3.1 zeigt zwei Programmertypen. Der Programmer links wurde aus einem Bausatz gefertigt und verfügt über einen eigenen Mikrocontroller. Er kann direkt an einen USB-Port angeschlossen werden und ist damit sehr universell und zuverlässig einsetzbar. Der rechte Programmer stammt aus dem Lernpaket Mikrocontroller [7] und benötigt eine serielle Schnittstelle am PC. Dafür ist er sehr leicht aufzubauen und gestattet so mit einfachen Mitteln die Programmierung eines ATtiny13.

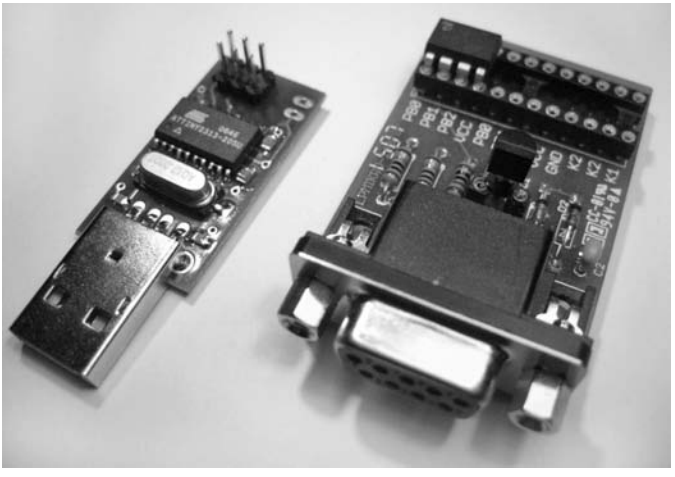


Abb. 3.1: Programmer

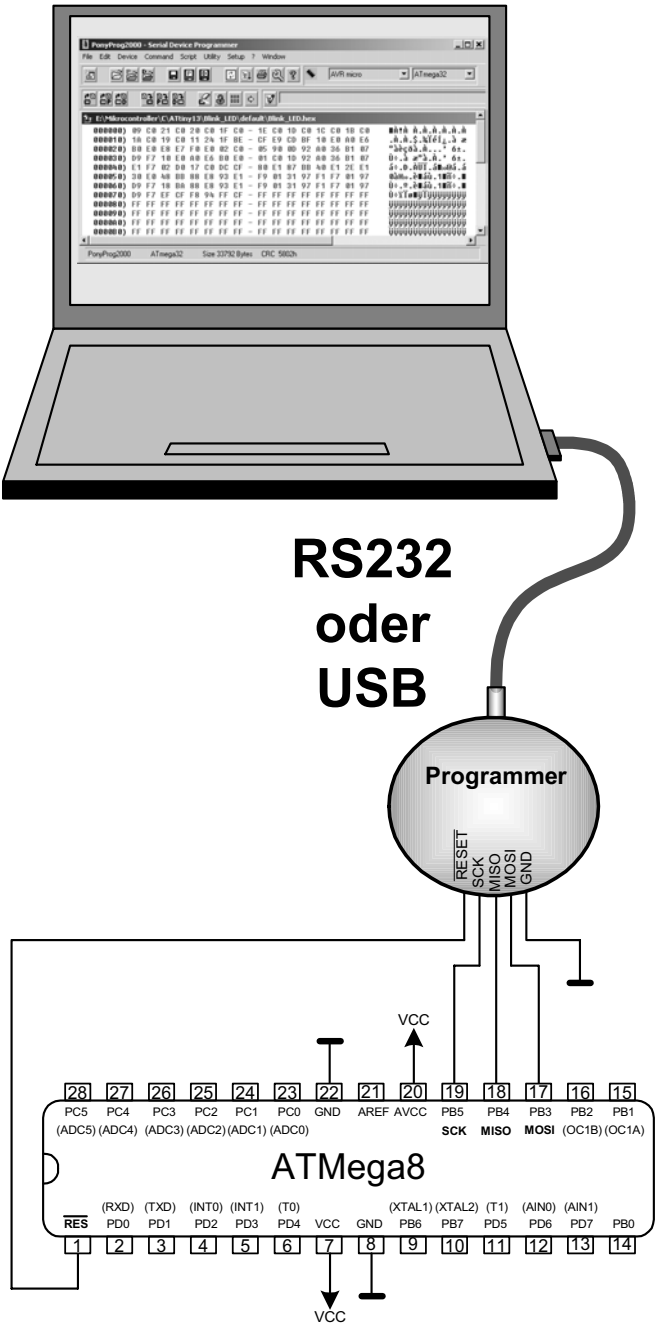


Abb. 3.2: Anschluss eines Programmers

Abschließend zeigt *Abb. 3.2* noch den Anschluss eines Programmers an den ATMega8. Bei anderen Controllertypen liegen die entsprechenden Signale natürlich an anderen Pins (s. hierzu jeweiliges Datenblatt), die Bezeichnung SCL (**S**erial **Cl**ock), MISO (**M**aster **i**n **S**lave **o**ut) und MOSI (**M**aster **o**ut **S**lave **i**n) ist aber bei allen in diesem Buch eingesetzten Typen gleich.

Die Kontaktbelegung der Programmer kann den zugehörigen Datenblättern entnommen werden.

3.2 Software-Tools

Neben der Programmer-Hardware selbst wird natürlich auch eine Software benötigt, mit der das fertige µC-Programm in den Chip geladen werden kann. Hierfür stehen verschiedenste Programme zur Verfügung. Eines der bekanntesten ist PonyProg [9]. Dieses sehr brauchbare Tool kann kostenlos aus dem Internet geladen werden und ist für die hier vorgestellten Projekte vollkommen ausreichend.

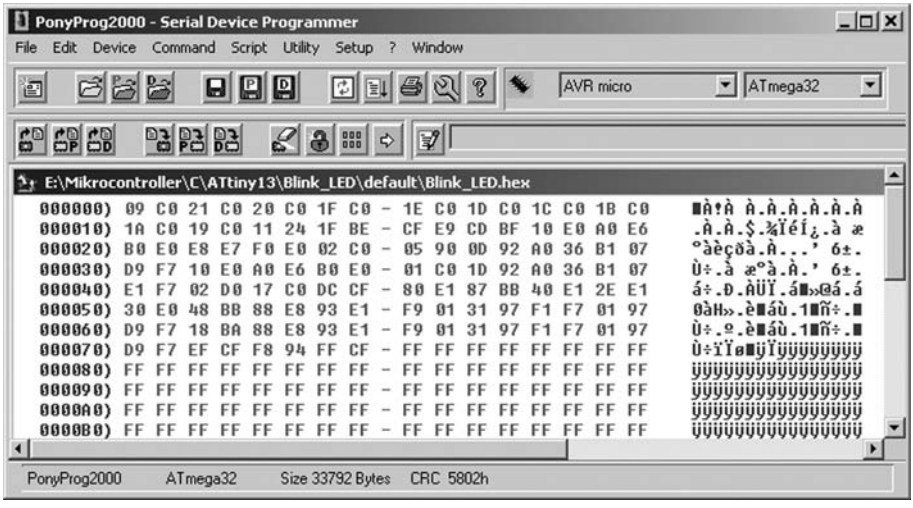


Abb. 3.3: PonyProg

3.3 C-Compiler

Die Mikrocontroller der ATMEL-AVR-Serie wurden auf eine Programmierung mit C optimiert. Natürlich lassen sich die Controller auch in Assembler programmieren. Die

Programmierung in einer Hochsprache kann aber naturgemäß deutlich strukturierter und übersichtlicher erfolgen.

Im professionellen Bereich hat sich die Verwendung von C als Programmiersprache praktisch ausnahmslos durchgesetzt. Aber auch bei nicht professionellen Anwendern nimmt C bei der Programmierung von ATMEL-Controllern einen deutlichen Spitzenplatz ein.

Insbesondere auch aufgrund seiner kostenlosen Verfügbarkeit ist der AVR-GCC-Compiler [10] weit verbreitet. Deshalb wurde er auch für sämtliche in diesem Buch vorgestellten Projekte eingesetzt.

Die CD zu diesem Buch enthält für alle Projekte einen Ordner, in dem jeweils der vollständige Quellcode und die zugehörige *.hex-Datei enthalten sind. Damit wird es möglich, sowohl die *.hex-Datei direkt auf den Ziel-Controller zu programmieren als auch das Projekt mit der *.aps Datei zu öffnen und den C-Code zu bearbeiten.

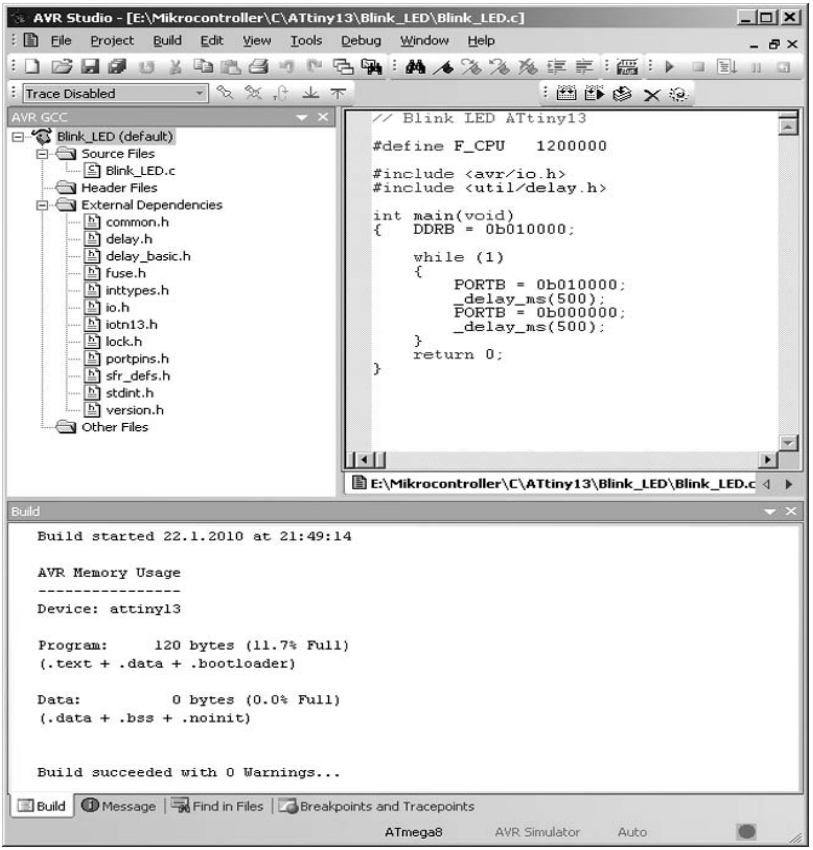


Abb. 3.4: AVR-GCC C-Compiler

3.4 Fuses

Die sogenannten Fuses sind einstellbare Konfigurationsparameter eines Mikrocontrollers. Sie können mit den Programmiertools verändert werden. Allerdings ist hier höchste Sorgfalt geboten. Wird ein Fuse-Bit irrtümlicherweise falsch gesetzt, so kann der gesamte Mikrocontroller eventuell unbrauchbar werden.

Für die meisten Projekte in diesem Buch muss nur ein Fuse-Bit verändert werden. Da der ATmega8 mit der Einstellung „interner Oszillator, 1 MHz“ ausgeliefert wird, muss er gegebenenfalls auf 8-MHz-Quarzbetrieb umgestellt werden. Dafür muss die voreingestellte Fuse-Konfiguration nach *Abb. 3.5* in die Konfiguration nach *Abb. 3.6* umgestellt werden, d .h, die drei Häkchen bei CKSEL1, CKSEL2 und CKSEL3 müssen entfernt und dafür das Häkchen bei CKSEL0 gesetzt werden. Entsprechendes gilt für die Taktumschaltung im Tiny13 und Mega32.

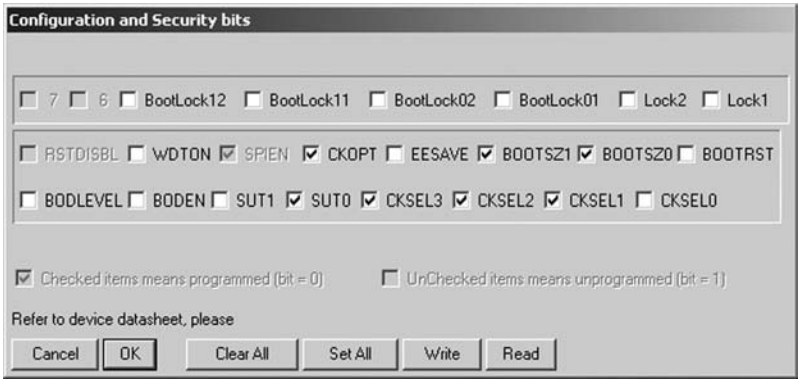


Abb. 3.5: Default Fuse-Einstellung des ATmega8

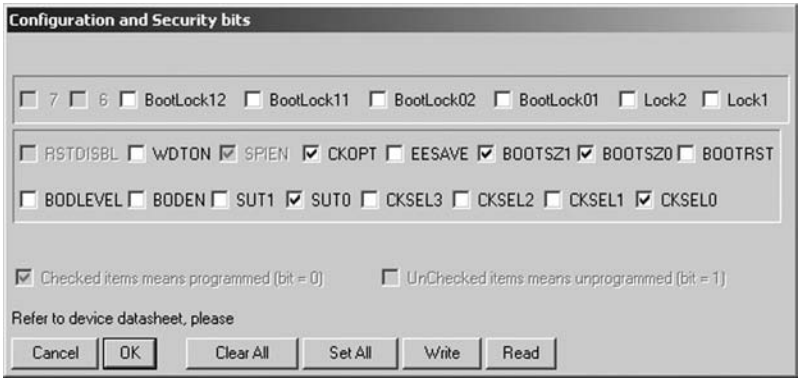


Abb. 3.6: Fuse-Einstellung des ATmega8 für Betrieb mit externem Quarz

3.5 Von der Idee zum Gerät

Abb. 3.7 zeigt den Gesamtablauf für die Entwicklung eines µC-basierten Geräts. Meist nimmt dabei die Programmerstellung (Software „SW“) mit Test den weitaus größten Zeitraum in Anspruch. Die Hardware („HW“) dagegen besteht oft nur noch aus dem Mikrocontroller selbst und einigen wenigen peripheren Bauteilen, die häufig in kurzer Zeit aufgebaut und geprüft werden können.

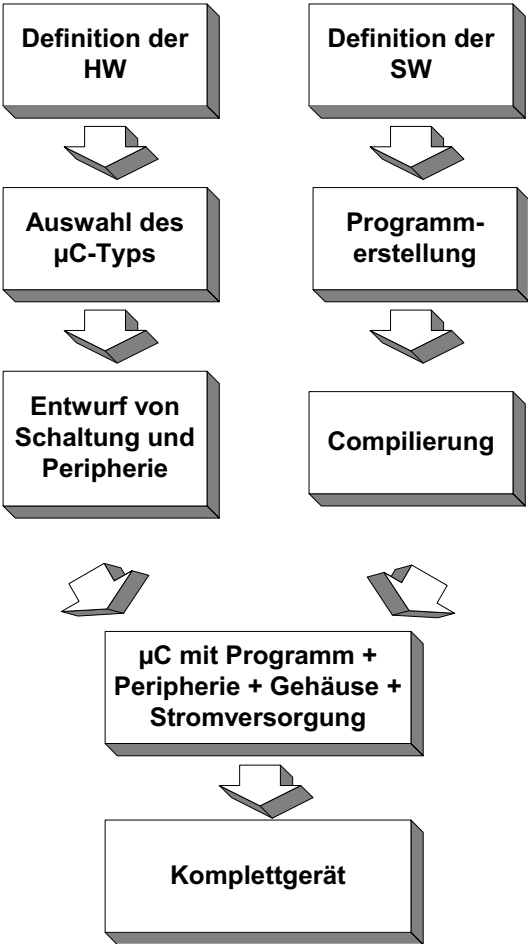


Abb. 3.7: Von der Idee zum Gerät

14 Die Include-Bibliotheken

In diesem Kapitel werden die in den einzelnen Projekten verwendeten Include-Dateien näher beschrieben. Prinzipiell kann man die Unterprogramme aus diesen Bibliotheken natürlich auch in eigenen Programmen nutzen, ohne ihre Funktionsweise genauer zu verstehen. Bei fortgeschrittenen Anwendungen ist es aber meist sehr hilfreich, wenn man zumindest eine ungefähre Vorstellung vom Aufbau dieser Funktionen hat.

14.1 Bit Utilities: BitUtilities.h

Die einfachste Include-Datei ist die BitUtilities.h. Diese stellt aber drei sehr wichtige und häufig genutzte Funktionen zur Verfügung:

- das Setzen eines Bits in einem Port
- das Löschen eines Bits in einem Port
- das sogenannte Toggeln, d. h. Invertieren eines Port-Bits

Das Listing zeigt die Funktionen im Einzelnen. Weitere Details dazu können dem Abschnitt 15.1 entnommen werden.

```
// General bit utilities
#define sbi(PORT, bit)    (PORT|=(1<<bit))  // set bit in PORT

#define cbi(PORT, bit)    (PORT&=~(1<<bit)) // clear bit in PORT
#define tgl(PORT, bit)    (PORT^=(1<<bit))  // toggle bit in PORT
```

Include-Datei BitUtilities.h

14.2 Pulsweitenmodulation: Initialize_PWM.h

Eine weitere einfache Include-Datei ist Initialize_PWM.h. Die enthaltene Funktion dient zur Initialisierung des fast PWM mode. Weitere Details finden sich wieder in Abschnitt 15.5.

```
// Initialize fast PWM mode
void init_PWM (unsigned int top)   // configure fast PWM
{ICR1 = top;     // set top value for PWM
 OCR1A = 0;      // set compare A
 OCR1B = 0;      // set compare B
```

```
  TCCR1A |= (1 << COM1A1); // non-inverting mode for Pin OC1A
 TCCR1A |= (1 << COM1B1);  // non-inverting mode for Pin OC1B
 TCCR1A |= (1 << WGM11);   // fast PWM Mode

 TCCR1B |= (1 << WGM13) | (1 << WGM12);     // fast PWM Mode
 TCCR1B |= (1 << CS11);
}
```

Include-Datei Initialize_PWM.h

14.3 Ansteuerung eines LCD-Displays: LCD_display.h

Etwas umfangreicher ist die Include-Datei für die Ansteuerung eines LCD-Displays. Hier werden zunächst die verschiedenen Codes für die Displaysteuerung festgelegt.

Es folgt die Definition der PORT-Pins. Standardbelegung ist PORTD mit den PINS 0 bis 5. Bei verschiedenen Projekten muss diese Pinbelegung geändert werden, deshalb existieren von dieser Include-Datei mehrere Varianten auf der beiliegenden CD.

In der Routine

```
void lcd_send(unsigned char type, unsigned char c)
```

wird jeweils ein Byte, aufgeteilt in Upper und Lower Nibble, zum LCD gesendet, dabei wird zwischen COMMAND und DATA unterschieden. Im ersten Fall wird das Byte als Display-Kommando interpretiert, z. B. als „Cursor Home“. Im zweiten Fall wird das gesendete Zeichen direkt auf der LCD-Matrix dargestellt.

Die nächste Routine

```
void lcd_set_cursor(uint8_t x, uint8_t y)
```

gestattet es, den Cursor an eine beliebige Stelle des Displays zu platzieren.

Mit

```
void lcd_write(char *data)
```

kann ein kompletter String an das Display gesendet werden. Die Übertragung erfolgt dabei zeichenweise mittels der Routine void lcd_send.

Schließlich kann mit

```
void lcd_init()
```

das Display initialisiert werden. Ohne vorherigen Aufruf dieser Routine ist das Display nicht adressierbar.

```
// LCD Output for HD44780 compatible displays
// Target: ATMEGA8
#define LCD_CLEAR    0x01  // Clear display: 0b 0000 0001

#define LCD_HOME   0x02 // Cursor home:      0b 0000 0010
#define LCD_ON     0x0C // Cursor invisible: 0b 0000 1100
#define LCD_OFF    0x08 // Display off:       0b 0000 1000
#define POS_01     0x80 // Line 1 - Column 0 0b 1000 0000
#define POS_02     0xC0 // Line 2 - Column 0 0b 1100 0000
// definition of port pins - data 4-7 -> LCDPORT4-7
#define LCDPORT    PORTD
#define LCDDDR     DDRD
#define LCD_PIN_RS   2
#define LCD_PIN_E    3
#define LCD_PIN_D4   4
#define LCD_PIN_D5   5
#define COMMAND    0
#define DATA  1
#include „BitUtilities.h“
// Function: Toggle enable pin
void toggle_enable_pin(void)
{sbi(LCDPORT, LCD_PIN_E); cbi(LCDPORT, LCD_PIN_E);

}
// Send Byte to LCD Controller
void lcd_send(unsigned char type, unsigned char c)
{unsigned char sic_c;      // Backup for c

 // send high nibble
 sic_c = c;       // save original c
 sic_c &= ~0x0f;       // set bit 0-3 == 0
 if (type==DATA)
   sic_c |= (1<<LCD_PIN_RS); // Data: RS = 1
 LCDPORT = sic_c;       // send high nibble
 toggle_enable_pin();
 // send low nibble
 sic_c = c;       // save original c
 sic_c = sic_c<<4;        // exchange nibbles
 sic_c &= ~0x0f;       // set bit 0-3 == 0
 if (type==DATA)
   sic_c |= (1<<LCD_PIN_RS); // Data: RS = 1
 LCDPORT = sic_c;       // send low nibble
 toggle_enable_pin();
 _delay_ms(5);          // Wait for LCD controller
}
// set cursor to line x and column y
void lcd_set_cursor(uint8_t x, uint8_t y)
{uint8_t i;
 switch (x)
   {  case 1: i=0x80+0x00+y; break;         // 1. line
     case 2: i=0x80+0x40+y; break;          // 2. line
     default: return;                        // invalid line
   }
   lcd_send(COMMAND, i);
```

```
}
// write string to LCD
void lcd_write(char *data)
{ while(*data) {lcd_send(DATA, *data); data++;}
}
// initialize LCD Controller
void lcd_init()
{// Set Port to Output
 LCDPORT = 0x00; LCDDDR = 0xFF;
 _delay_ms(50); // Wait for LCD
 // 4-bit Modus config
 sbi(LCDPORT, LCD_PIN_D5); cbi(LCDPORT, LCD_PIN_D4);
 // 4-Bit Modus start
 sbi(PORTD, LCD_PIN_E); cbi(PORTD, LCD_PIN_E);
 _delay_ms(5);
 // 2 Lines, 4-Bit Mode
 lcd_send(COMMAND, 0x28);
 lcd_send(COMMAND, LCD_OFF); lcd_send(COMMAND, LCD_CLEAR);
 lcd_send(COMMAND, 0x06);    lcd_send(COMMAND, LCD_ON);
}
```

Include-Datei LCD_display.h

14.4 IR-Fernbedienungsempfang: RemoteControl.h

Dieses Include-File gestattet den Empfang von IR-Fernbedienungssignalen. Wie in Abschnitt 10.1 dargelegt, besteht es aus einer Sequenz von Datenpulsen.

Die Erkennung einer Pulsflanke erfolgt durch die Detektion eines Signalpegelwechsels. Die geeignete logische Verknüpfung hierfür ist die EXOR-Funktion. Sie liefert eine logische Eins, wenn zwei zeitlich nacheinander liegende Eingangswerte unterschiedliche Pegel haben, d.h. nach jeder Datenpuls-Flanke.

Wenn das Signal im richtigen zeitlichen Abstand weiter abgetastet wird, kann man so sequenziell alle Bits erfassen. Beim RC5-Code hat man nach 14 Bitzeiten alle Bits aufgezeichnet.

```
// Remote control receiver on PB0
volatile unsigned int  RemCo_data;
unsigned char b, t;  // current bit and time
unsigned int  r; // aux register
ISR (TIMER0_OVF_vect)
{ TCNT0 = 254;
 if((b^PINB) & 1)         // edge detection @ PB0
    {b = ~b;
         if(!r || t>22 )
    {r <<= 1;
     if(!b & 1) r |= 1;   // add current bit
```

```
          t = 0;
     }
     }
   if( ++t > 33 )
 { // check for length & external transfer
   if(((1<<13) & r  && !((1<<14) & r))) RemCo_data = r;
   r = 0;
   }
}
```

Include-Datei RemoteControl.h

14.5 Serielles Interface: serial.h

Diese Include-Datei dient zur Übertragung von Informationen über die serielle Schnittstelle des Mikrocontrollers. In der Routine

```
void setup_uart()
```

erfolgt zunächst die Initialisierung der Schnittstelle. Dabei werden die Baudrate und die Bit-Konfiguration (Start-, Stoppbitlänge sowie Parität) festgelegt.

Die Routinen `void uart_putchar(char c)` und `void uart_write(char *str)` dienen zur Ausgabe von Zeichen bzw. ganzen Strings auf die Schnittstelle.

```
// Serial Communication
// ATmega8
#include <avr/io.h>
#include <util/delay.h>
#define UART_BAUDRATE  1200
// calculate configuration parameter
#define UART_SETTING ((F_CPU/16L/UART_BAUDRATE)-1)
void setup_uart()
{// set baud rate
 UBRRH = (char) (UART_SETTING >> 8);
 UBRRL = (char) (UART_SETTING);
 // activate Rx and Tx
 UCSRB = (1<<RXEN) | (1<<TXEN);
 // select async 8N1
 UCSRC = (1<<URSEL) | (3<<UCSZ0);
}
void uart_putchar(char c)
{while (!(UCSRA & (1 << UDRE))); // wait for buffer ready
 UDR = c;
}
void uart_write(char *str)
{while(*str) {uart_putchar(*str); str++;}
}
```

Include-Datei serial.h

14.6 Ansteuerung der 4x7-Segment-LED-Anzeige: LED_display.h

Für die Ansteuerung einer 4x7-Segmentanzeige kann die folgende Include-Datei verwendet werden. Die Kathoden der Display-LEDs müssen dabei an Port D (Pins 0-7) angeschlossen werden:

Tab. 14.1: Anschluss der LED-Anzeige an der LED-Anzeige an den µC

PORTD	7	6	5	4	3	2	1	0
Display Pin	a	f	b	g	c	DP	d	e
Zustand z. B. für "0"	0	0	0	1	0	1	0	0
°-Zeichen	0	0	0	0	1	1	1	1
Bit-Wertigkeit	128	64	32	16	8	4	2	1

In der Initialisierungsroutine werden diese Ports entsprechend als Ausgänge deklariert:

```
DDRD = 0b11111111;
```

Es folgen die Definitionen für die Interruptsteuerung. Abschließend werden Interrupts allgemein zugelassen (`sei();`).

Die Zahlendarstellung erfolgt durch Zerlegung der übergebenen Zahl in einzelne Ziffern in der Routine `void number_output (uint16_t number)`. Mit `void digit (uint8_t value, uint8_t pin)` werden die Ziffern an die korrekte Position im Display geschrieben. Dabei werden die Ziffern im Array

```
const int8_t numbers [10]
```

wie folgt codiert: Soll z. B. die Ziffer "0" dargestellt werden, so müssen alle Segmente bis auf das g-Segment leuchten. Da eine Anzeige mit gemeinsamer Anode gewählt wurde, bedeutet dies, dass alle Ausgänge, bis auf den am g-Segment angeschlossenen Pin, auf Low liegen müssen. Nach Tab. 14.1 und Tab. 14.2 bedeutet dies ein Binärmuster für die „null" von:

```
0b00010100
```

oder den Wert

0*128+**0***64+**0***32+**1***16+**0***8+**1***4+**0***2+**0***1 = 16 + 4 = 20

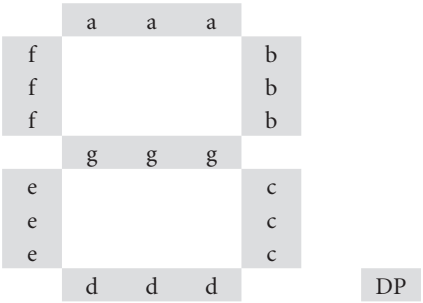
in dezimaler Schreibweise. Analog können so auch alle anderen Ziffern codiert werden. Auch andere Zeichen sind auf diese Weise darstellbar. So kann das °-Zeichen als binär

```
0b00001111
```

oder dezimal 15 in die Anzeige geschrieben werden.

Die Interrupt-Service-Routine sorgt schließlich dafür, dass alle Ziffern zeitlich nacheinander, d. h. im Zeitmultiplex dargestellt werden.

Tab. 14.2: Codierung der Segmente



Ziffer	Binär	a	f	b	g	c	DP	d	e		Dezimal
0	0b00010100	0	0	0	1	0	1	0	0	=	20
1	0b11010111	1	1	0	1	0	1	1	1	=	215
2	0b01001100	0	1	0	0	1	1	0	0	=	76
3	0b01000101	0	1	0	0	0	1	0	1	=	69
4	0b10000111	1	0	0	0	0	1	1	1	=	135
5	0b00100101	0	0	1	0	0	1	0	1	=	37
6	0b00100100	0	0	1	0	0	1	0	0	=	36
7	0b01010111	0	1	0	1	0	1	1	1	=	87
8	0b00000100	0	0	0	0	0	1	0	0	=	4
9	0b00000101	0	0	0	0	0	1	0	1	=	5
°	0b00001111	0	0	0	0	1	1	1	1	=	15

```
// 4 x 7 digit LED display control
// seven segment codes
const int8_t numbers [10] = {20, 215, 76, 69, 135, 37, 36, 87, 4, 5};
volatile uint8_t Digit_X000, Digit_0X00, Digit_00X0, Digit_000X;
// store single digits
volatile uint8_t active_digit = 0;          // active digit
void initialize_LED_display()
{DDRD = 0b11111111;
 DDRC = 0b00001111;
 OCR1A = 8;      // load compare value for 1 ms repetition time
 TIMSK |= (1 << OCIE1A);   // interrupt if compare match
 // div 1024 & clear counter on compare match
 TCCR1B = (1<<CS12) | (1<<CS10) | (1<<WGM12);

 sei();
}
// Write number at indicated position to display
```

```
void digit (uint8_t value, uint8_t pin)
{ PORTD = numbers[value];  // send number to output
  PORTC &= ~(1 << pin); // PinLow => current digit on
}
// distribute number to digits
void number_output (uint16_t number)
{ Digit_X000 = number/1000;  number %= 1000;
  Digit_0X00 = number/100;   number %= 100;
  Digit_00X0 = number/10;  number %= 10;
  Digit_000X = number;
}
ISR (SIG_OUTPUT_COMPARE1A)
{ PORTC = 0b00001111;   // Digits off
  if(active_digit == 0) digit (Digit_000X, PC0);
  if(active_digit == 1) digit (Digit_00X0, PC1);
  if(active_digit == 2) digit (Digit_0X00, PC2);
  if(active_digit == 3) digit (Digit_X000, PC3);

  active_digit ++; if (active_digit == 4) active_digit = 0;

}
```

Include-Datei LED_display.h

21 Stichwortverzeichnis

U

V

W

Z

FRANZIS
PC & ELEKTRONIK

Dr. Günter Spanner

AVR-Mikrocontroller in C programmieren

Im Vergleich zur Elektronik, bei der man ganz elementar beginnen kann, ist bei der Mikrocontrollertechnik eine deutlich höhere Hürde zu überwinden Zwar ist die „Hardware" für einen Mikrocontrollereinstieg sehr einfach und kann nur aus dem Controller selbst, einem Widerstand und einer LED bestehen, allerdings sind zur Programmierung des Controllers mindestens noch ein PC und ein Programmierkabel erforderlich. Spätestens ab hier werden Sie ein gutes Fachbuch zu schätzen wissen.

Alle Projekte in diesem Buch sind vollständig beschrieben und für den praktischen Einsatz in Labor und Alltag geeignet. Neben dem eigentlichen Lerneffekt steht insbesondere die Freude am Zusammenbau kompletter und nützlicher Geräte im Fokus. Dabei kann der Aufbau der Hardware durchgehend auf Laborsteckboards erfolgen. Dies ermöglicht die Herstellung aller Geräte mit sehr geringem Aufwand. Das Erproben und Testen der Anwendungen wird zum lehrreichen Vergnügen.

Als Programmiersprache wird durchgehend C verwendet, was einen klar strukturierten und modularen Aufbau der Software ermöglicht. Der Autor geht im Rahmen der einzelnen Applikationen auf alle relevanten Elemente der Programmiersprache C und die zugehörigen Erweiterungsbibliotheken ausführlich ein.

Das Buch unterstützt das „Learning by Doing". Es setzt auf deduktives Lernen anhand der dargestellten Projekte. Durch viele Anwendungen wie Geschwindigkeitsmessung, Digitalvoltmeter oder Rechteckgenerator ist das Buch als Anregung für Schule, Studium, Ausbildung, Beruf und Hobby bestens geeignet.

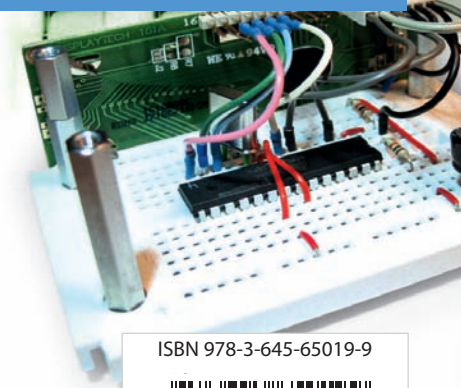
Darüber hinaus kann das Werk aber auch hervorragend als Handbuch dienen. Ein umfassendes Inhalts- und Stichwortverzeichnis erleichtert das schnelle Auffinden einzelner Themen. Damit ist dieses Buch ein wertvolles Kompendium und umfassendes Nachschlagewerk.

Praktische Experimente:

- LED-Lichtspiele
- Präzise Zeitmessung
- Messwerterfassung
- IR-Fernbedienungsempfänger
- Mikrocontroller und USB-Schnittstelle
- Bioelektronik
- und viele mehr

Auf CD-ROM:

- Alle Projekte mit vollständig dokumentiertem C-Programmcode
- Programmbibliothek und umfassendes C-Kompendium



ISBN 978-3-645-65019-9

39,95 EUR [D]