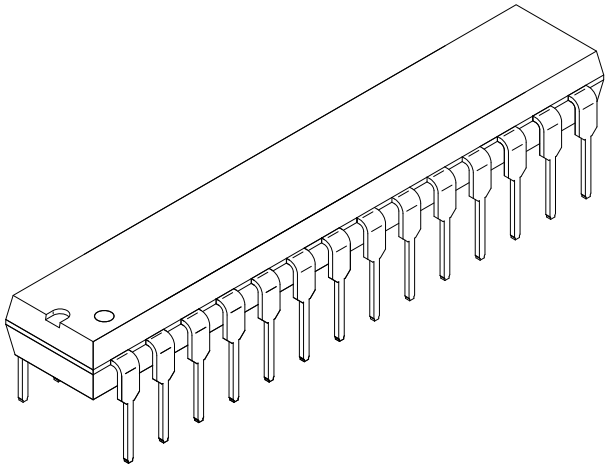
Mikrocontroller-Seminar für Einsteiger 2008

Seminarleitfaden



Nicolas DL1DOW Sänger

18. September 2008

Das vorliegende Script dient als Begleitmaterial zum gleichnamigen Seminar.
Dieses Dokument darf frei kopiert, ausgedruckt oder verändert werden, solange die jeweiligen Quellen genannt werden.
Für Fehler, unsinnige Behauptungen oder nur langweiligen Inhalt und daraus resultieren Sach-, Personen- oder Vermögensschäden übernimmt der Autor keine Haftung.

`http://www.dl1dow.de`

1 Inhaltsübersicht

Die ganze Veranstaltung ist als absolute Anfänger-Einführungsveranstaltung gedacht, um der Welt der Mikrocontroller das Geheimnisvolle zu nehmen. „Alte Hasen“ sind natürlich als Hilfestellung jederzeit auch willkommen. Vorraussetzung sollten Basis-Programmierkenntnisse sein („Was sind Schleifen?“, „Wozu dienen Variablen?“...). Benutzt wird der ATMEGA8 und der ATMEGA168 in einer Entwicklungsumgebung („AVR-Studio“ von ATMEL), Programmiersprache ist C (Basiskenntnisse hilfreich, aber nicht erforderlich). Theorie, Software und Hardware sollen dabei immer abwechselnd und ineinandergreifend behandelt werden, um mit ersten Resultaten und Übungen später die direkte Anwendung in eigenen Projekten zu ermöglichen.

Termin/Ort

Freitag der 3. Oktober und Samstag der 4. Oktober,
Freitag von 10:30 bis ca. 17:30, Samstag von 10:00 bis ca. 17:30
Marieneheim in Bielefeld-Schildesche Engersche Str. (in der Nähe vom Mediamarkt)

Inhalt

- Grundschaltung und Schnittstellen
- Übersicht Entwicklungsumgebungen und Compiler
- Grundlagen μC (In- und Outputfunktionen, Ports)
- Programmierung in C (keine Vorkenntnisse erforderlich)
  - Grundsätzlicher Programmablauf
  - Schleifen und Bedingungen
  - Datentypen und Operationen
- Flashen eines μCs (ISP/High-Voltage-Programming, Bootloader, Fuse-Bytes)
- Peripherie des Controllers (AD-Wandler, USART, PWM)
- Ansteuerung von LCDs
- Kommunikation zu einem PC über RS232
- Timer und Interrupts
- Entprellen von Tastern und Schaltern

Die auf den Kurs abgestimmte Testplatine, die benötigten Elektronikkomponenten und ein USB-ISP-Programmer sind im Teilnahmebeitrag enthalten.

Inhaltsverzeichnis

1 Inhaltsübersicht **1**

2 Vorbereitung **4**
2.1 PC und Software 4
2.2 Mitbringliste 6

3 Hardware **7**
3.1 Testplatine 7
3.2 Programmieradapter 10

4 Programmierung **16**
4.1 „Hallo Welt“ 16
4.2 LCDs 17
4.3 A/D-Wandler 20
4.4 Die „Arduino“-Entwicklungsumgebung 21

5 Schaltungen und Flashen **25**
5.1 Externer Programmer 25
5.2 In-System-Programming 25
5.3 Bootloader 26
5.4 Fuse- und Lockbits 27

6 Kleines C-Lexikon **29**
6.1 Operatoren 29
6.2 Library-Funktionen 30
6.3 Datentypen und Funktionen 31
6.4 Dies und das 32

7 Weiterführende Literatur **34**

A Literaturverzeichnis **36**

B Quelltexte **37**

2 Vorbereitung

Die folgenden Vorbereitungen müssen auf jeden Fall vor dem Beginn des Seminars abgeschlossen sein. Bei Schwierigkeiten stehe ich gerne mit Rat zur Seite.

2.1 PC und Software

Im Rahmen des Seminars sollte ein PC mit Windows-Betriebssystem[1] (ab Windows 2000) und mindestens einem freien USB-Anschluß mitgebracht werden. An Rechenleistung reicht problemlos alles ab Pentium III. Am praktischsten ist natürlich ein Laptop, aber nicht unbedingt zwingend.

Nach Absprache kann ein Rechner (ohne Monitor) für die Dauer des Seminars zur Verfügung gestellt werden.

Die folgende Software sollte schon vor dem Seminar selbst heruntergeladen und installiert werden:

- AVR Studio 4.13 mit Service Pack 2[2] (kostenlos, vor dem Download ist eine Registrierung bei ATMEL erforderlich.) Mittlerweile ist auch AVRStudio 4.14 unter der gleichen URL erhältlich. Beide Versionen sind für unsere Zwecke gleichwertig.
- WinAVR[3] (Freeware, enthält den GCC und AVRdude)
- USB AVR-ISP Tool[4] (mindestens Version 0.8). Eine Linux-Version ist verfügbar. Das Programm enthält die USB-Treiber für den Programmieradapter und das Eval-Board.

Es sollte eine aktuelle Version der jeweiligen Software installiert werden, da sich ältere Versionen teilweise deutlich anders verhalten. *Die Software sollte in einen Pfad installiert werden, der keine Leerzeichen enthält, sonst könnten später Probleme auftreten, wenn die Kommandozeile verwendet werden muß.* (Gut: `C:\AVRTools\WinAVR` , schlecht: `C:\AVR Tools\WinAVR`).

Windows 2000

Bei Windows 2000 ist das Service Pack 4 erforderlich. Falls Schwierigkeiten bei der Installation der verwendeten Software auftreten, kann diese notfalls auch noch kurzfristig zu Beginn des

[1] Die Entwicklungsumgebung ist auf virtuellen Maschine (getestet unter VMWare) unter Linux lauffähig.
[2] `http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725`
[3] `http://winavr.sourceforge.net/download.html`
[4] `http://www.ullihome.de/index.php/USBAVR-ISP-Download/de`

Seminars installiert werden. Bitte dazu etwas früher kommen. Zumindest die Grundversion von AVRStudio sollte jedoch schon im Voraus heruntergeladen werden, da diese nicht frei kopiert werden darf.

Windows XP 32 Bit

Bei Windows XP ist mindestens das Service Pack 2 erforderlich. Auch hier gilt: Falls Schwierigkeiten bei der Installation der verwendeten Software auftreten, kann diese notfalls auch noch kurzfristig zu Beginn des Seminars installiert werden. Bitte dazu etwas früher kommen. Zumindest die Grundversion von AVRStudio sollte jedoch schon im Voraus heruntergeladen werden, da diese nicht frei kopiert werden darf.

Windows Vista, Windows XP 64 Bit

Da bislang auf meiner Seite keine Erfahrungen mit der verwendeten Software unter Windows Vista oder einem 64-Bit-Betriebssystem vorliegen, muß in diesem Fall im Vornherein schon alles getestet werden.

AVRStudio und WinAVR müssen installiert sein. Es muß sichergestellt sein, daß beide Zusammenarbeiten, indem ein Testbeispiel (z.B. das „Hallo Welt“) kompiliert und die Hex-Datei erzeugt wird.

Der Programmer wird unter Windows Vista nicht aus der AVR-Studio-Entwicklungsumgebung heraus benutzbar, sondern mit der Kommandozeile zu bedienen sein. Daher sollte man sich unter Vista mit der Kommandozeile vertraut machen.

Linux

Der GCC ist natürlich für Linux erhältlich und normalerweise im APT oder RPM direkt verfügbar. Außerdem werden benötigt:

- AVR-LIBC (Libraries für AVR)
- Programmiersoftware: AVRdude gibt es auch für Linux. Der im Seminar verwendete Programmieradapter wird dazu mit der USBASP-Firmware versehen.
- Entwicklungsumgebung: Für Linux gibt es CDK4AVR; AVRStudio funktioniert auch mit den Wine-Bibliotheken, nur das direkte flashen des Controllers aus AVRStudio heraus funktioniert nicht.

Auf einer virtuellen Maschine mit Windows XP (32 Bit) funktioniert alles genau wie unter einem nativen Windows. Alles in allem sollte mit Linux alles vorher installiert und getestet werden, da dazu während des Seminars kein Support gegeben werden kann.

2.2 Mitbringliste

- PC (siehe oben)
- USB-Kabel
- Dreifach-Steckdosenleiste
- Schreibutensilien
- Multimeter (wenn möglich)

Während des Seminars wird ein Manuskript ausgeteilt, das zum Teil ausführlicher als das auf der Website herunterladbare ist. Das Skript von der Website sollte deshalb nicht vorher ausgedruckt werden.

3 Hardware

3.1 Testplatine

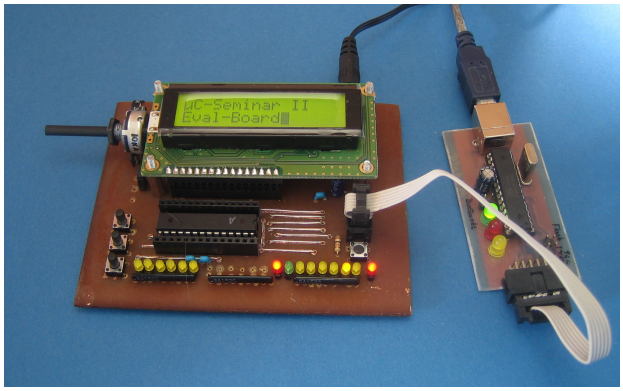


Abbildung 3.1: Fertig aufgebautes Testboard (links) mit Programmieradapter (rechts)

Die übliche äußere Beschaltung des ATmega8/ATmega168 gestaltet sich sehr einfach und beschränkt sich auf eine 5V-Spannungsversorgung, einen Quarz (1-16 MHz), einen Pull-Up-Widerstand für die Resetleitung und eine handvoll (also fünf) Kondensatoren. Die im Rahmen dieses Seminars verwendete Testplatine verfügt darüberhinaus über folgende Zusätze:

- LEDs an jedem Port erlauben einen direkten Überblick über den Status jeder Leitung.
- Ein USB-Seriell-Adapter auf Basis eines ATtiny 2313 für Experimente erlaubt die einfache Verbindung mit einem PC und die Verwendung eines Bootloaders. Der Adapter meldet sich als virtueller COM-Port (Windows) an. Die Geschwindigkeit 9600 Baud 8N1 ist fest vorgegeben.
- Ein LCD (2×16 Zeichen) mit HD47780-kompatiblem Controller
- Ein Poti kann für die Simulation analoger Sensorsignale verwendet werden.
- Drei Taster dienen als digitale Eingabe.
- Alle Pins des Controllers sind auf eine Steckerleiste geführt, was flexibel weitere Testschaltungen ermöglicht.

Das Layout im Eagle-Format ist im Internet verfügbar[1].

[1] `http://www.dl1dow.de/inhalt/selbstbau/mikrocontrollerseminar2/seminar2008.htm`

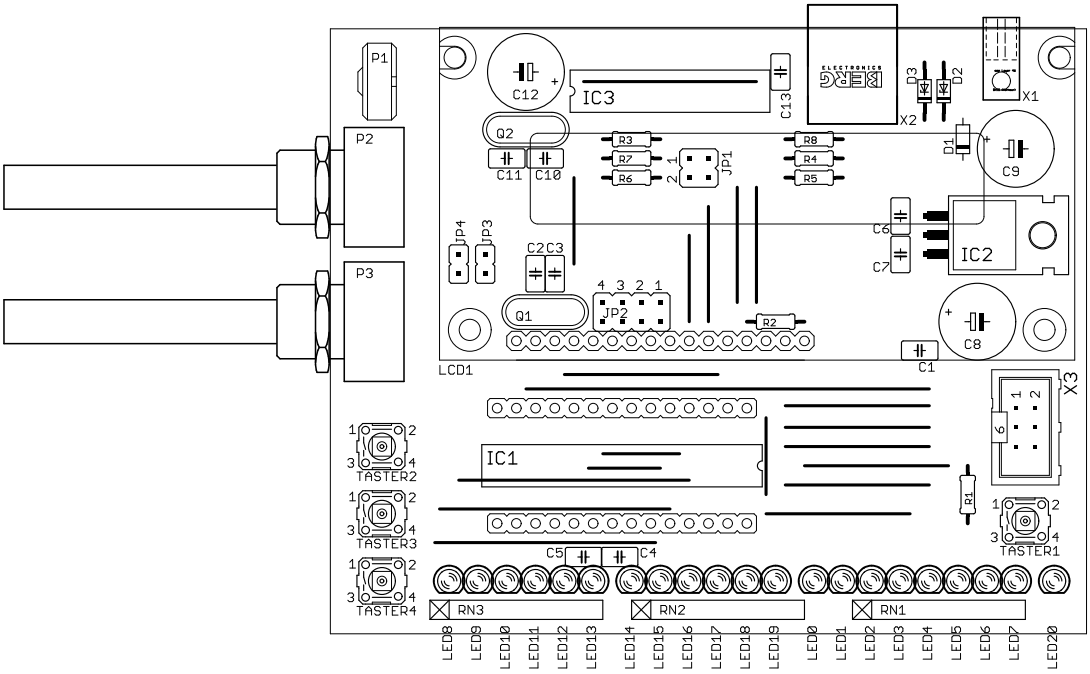


Abbildung 3.2: Bestückungsplan (Maßstab 1:1) der Testplatine

Bestückung

Die Bestückung kann sich an dem Bestückungsplan in Abbildung 3.2 und der Stückliste in Tabelle 3.1 auf der nächsten Seite orientieren:

- Bei der Bestückung ist bei den insgesamt zwanzig Lötbrücken zu beginnen.
- Die sechs LEDs für PortC (LEDs 14-19) sind nicht zu bestücken. Mehr dazu gibt es im Abschnitt „interne und externe Pull-Up-Widerstände“ auf Seite 10.
- Die Sockelleisten lassen sich am besten zerteilen, indem erst ein Pin herausgezogen und dann mit einem scharfen Messer geteilt wird.
- Die Bohrungen des Displays müssen vor der Montage auf 3,5mm aufgebohrt werden, um ein Verschrauben mit M3 zu ermöglichen. Auf die Hauptplatine und unter das Displaymodul werden 4,4mm hohe Sockelleisten eingelötet. Die Verbindung zwischen Haupt- und Displayplatine erfolgt dann mittels einer eingesteckten Stiftleiste. Die Abstandsbolzen werden von unten mit Schrauben M3×6mm angeschraubt, so daß das Gewinde nach oben zeigt. So läßt sich das Display nachher bequem an- und abstecken.
- Der USB-Seriell-Adapter kann auch weggelassen werden, ohne das andere Funktionen beeintächtigt werden. Hierzu sind einfach die in der Stückliste markierten Bauteile nicht zu bestücken.

Bezeichnung	Wert	Gehäuse	Reichelt-Best.-Nr.
C1	100n		nicht bestückt
C2,C3,C10[2] ,C11[2]	22p	2,5mm	KERKO 22P
C4-C7,C13[2]	100n	2,5mm	X7R-2,5 100N
C8,C9,C12[2]	100μF	5,0mm	RAD 100/63
D1	1 N 4148	7,5mm	1N 4148
D2[2] ,D3[2]	Z-Diode 3,9V	7,62mm	ZF 3,9
IC1(1)	ATmega168	DIL	ATMEGA 168-20DIP
IC1(2)	IC-Fassung 28pol		GS 28P-S
IC1(3)	2× Sockelleisten 14pol		BL 1X20G8 2,54 [1]
IC2(1)	LM 7805	TO-220	μA 7805
IC2(2)	Schraube M3×6mm		SKL M3X6 [1]
IC2(3)	Fächerscheibe 3,2mm		SKZ 3,2-100 [1]
IC2(4)	Mutter M3		SK M3 [1]
IC3(1)[2]	ATtiny2313	DIL	ATTINY 2313 DIP
IC3(2)[2]	IC-Fassung 20pol		GS 20P
JP1(1)[2]	Stiftleiste 2×2pol	2,54mm	SL2X40G 2,54 [1]
JP1(2)[2]	Jumper (2×)	2,54mm	JUMPER 2,54 BL
JP2(1)	Stiftleiste 4×2pol	2,54mm	SL2X40G 2,54 [1]
JP2(2)	Jumper (4×)	2,54mm	JUMPER 2,54 BL
JP3,JP4(1)	Stiftleiste 1×2pol	2,54mm	SL2X40G 2,54 [1]
JP3,JP4(2)	Jumper (1×)	2,54mm	JUMPER 2,54 BL
LCD1(1)	LCD-Modul POWERTIP PC1602LRS-FSO-B-Y6		Pollin Best.-Nr.: 120 476
LCD1(2)	Sockelleiste 1×16pol (2×)	2,54mm	BL 1X20G 2,54 [1]
LCD1(3)	Stiftleister 1×16pol	2,54mm	SL 1X36G 2,54 [1]
LCD1(4)	Abstandsbolzen 20mm (4×)		DA 20MM
LCD1(5)	Mutter M3 (4×)		SK M3 [1]
LCD1(6)	Schraube M3×6mm (4×)		SKL M3X6 [1]
Taster1	–	–	TASTER 3301
LED0,LED20	Low-Current-LED 3mm rot	2,54mm	LED 3MM 2MA RT
LED1	Low-Current-LED 3mm grün	2,54mm	LED 3MM 2MA GN
LED2-LED13	Low-Current-LED 3mm gelb	2,54mm	LED 3MM 2MA GE
LED14-LED19			nicht bestückt
P1	Trimmer 50kΩ	2,54mm	PT 10-S 50K
P2	Potentiometer linear 47kΩ	5,08mm	P4M-LIN 47K
P3	Potentiometer linear 47kΩ		nicht bestückt
Q1	Quarz 16MHz		16,0000-HC49U-S
Q2[2]	Quarz 12MHz		12,0000-HC49U-S
R1,R3[2]	10kΩ	–	1/4W 10K
R2	10Ω	–	1/4W 10
(R4-R7)[2]	68Ω	–	1/4W 68
R8[2]	2,2kΩ	–	1/4W 2,2K
RN1-RN3	Widerstandsnetzwerk 8 Wid. 1,5kΩ	2,54mm	SIL 9-8 1,5K
TASTER2- 4	Taster 1 Schließer		TASTER 9303
X1	Hohlbuchse		HEBW 21
X2[2]	USB-Buchse B gewinkelt		USB BW
X3	Wannenstecker 6pol	2,54mm	z.B. CSD-Electronics Best.Nr.: 015-WG06

[1] Reicht für mehrere Anschlüsse/Leiterplatten
[2] Nur bestücken wenn auf der Platine ein USB-TTL-Seriell-Wandler vorgesehen ist!

Tabelle 3.1: Stückliste der Testplatine

Portpin	DIL-pin	Arduino	Com	Sonst.	Anschluß	
PB0	14	D8			Display EN	
PB1	15	D9		PWM		
PB2	16	D10	SS	PWM		
PB3	17	D11	MOSI	PWM		
PB4	18	D12	MISO		Display RW	
PB5	19	D13	SCK		Display RS	
PC0	23	A0		ADC	Poti1	
PC1	24	A1		ADC	Poti2	
PC2	25	A2		ADC	Taster1	
PC3	26	A3		ADC	Taster2	
PC4	27	A4		ADC	Taster3	
PC5	28	A5		ADC		
PD0	2	D0	RXD		RS232	Display 8 Bit
PD1	3	D1	TXD		RS232	Display 8 Bit
PD2	4	D2		PWM[1]		Display 8 Bit
PD3	5	D3				Display 8 Bit
PD4	6	D4			Display 4 Bit	Display 8 Bit
PD5	11	D5		PWM[1]	Display 4 Bit	Display 8 Bit
PD6	12	D6		PWM[1]	Display 4 Bit	Display 8 Bit
PD7	13	D7			Display 4 Bit	Display 8 Bit

[1] Nicht ATmega8

Tabelle 3.2: Pinout des ATmega8/ATmega168 auf der Testplatine

Interne und externe Pull-up-Widerstände

Wie aus dem Schaltplan der Testplatine in Abbildung 3.2 auf Seite 15 hervorgeht, schließen die Taster gegen Masse. Um einen definierten Zustand der Portpins PC0 bis PC2 zu erreichen sind Pull-up-Widerstände erforderlich. Diese sind im ATmega8/ATmega168 vorhanden und können softwareseitig aktiviert werden.

Werden die LEDs LED14, LED15 und LED16 bestückt wirken sie diesen Pull-up-Widerständen als Pull-down entgegen, so daß zum einen die Pull-ups unnötig belastet werden und zu anderen die Taster keinen definierten Zustand mehr aufweisen.

Werden die Potentiometer P2 und P3 bestückt oder die Eingänge PC0 bis PC1 als analoge Eingänge verwendet, sind die entsprechenden LEDs ebenfalls nicht zu bestücken, da sie als nichtlineare Pull-down-Widerstände wirken. Die mit einem Taster bestückten Pins PC0, PC1 und PC2 können als analoge Eingänge verwendet werden, wenn die internen Pull-up-Widerstände deaktiviert und die Taster nicht betätigt werden.

3.2 Programmieradapter

Der hier verwendete ISP-Programmieradapter wird „USB AVR-Lab“[11] genannt und ist mit der entsprechenden Firmware voll kompatibel zum STK500 von ATMEL. Er meldet unter

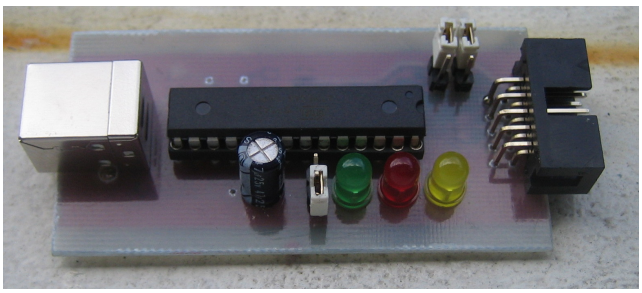


Abbildung 3.3: ISP-Programmieradapter

Windows einen virtuellen Com-Port an, auch für Linux sind Treiber verfügbar. Damit kann er mit sämtlichen Entwicklungsumgebungen und Programmierprogrammen, die das STK500 unterstützen, direkt verwendet werden. Dies sind (Aufzählung garantiert unvollständig):

- AVRStudio
- AVRdude
- WinAVR (baut auf AVRdude auf)
- Arduino (baut auf AVRdude auf)
- Bascom
- ...

Damit handelt es sich trotz des geringen Aufwandes und Preises um einen vollwertigen USB-AVR-Programmieradapter, der sich nicht hinter kommerziellen Produkten zu verstecken braucht.

(An die Unbelehrbaren: Über den integrierten Bootloader läßt sich auch die „USBASP“-Firmware aufspielen.)

Da der Programmieradapter einen virtuellen COM-Port emuliert, wird er im Rahmen dieses Seminars auch als USB-Seriell-Wandler (RS232 mit TTL-Pegeln) verwendet.

Zu den weiteren Fähigkeiten und Einstellungen des Programmieradapters kann [11] konsultiert werden.

Aufbau

Der Aufbau sollte auch für einen Lötanfänger keine Schwierigkeiten bereiten. Es handelt sich um eine Mischbestückung aus bedrahteten und großen SMD-Kompontenten, was eine sehr handliche Platine ergibt. Für die Bestückung ist die Stückliste in Tabelle 3.3 und der Bestückungsplan in Abbildung 3.4 auf der nächsten Seite zu konsultieren. Es sollte mit den vier Drahtbrücken begonnen werden, ansonsten ist die Bestückungsreihenfolge egal.

Für das Aufspielen der Firmware wird der Programmer einfach mit einem zehnpoligen 1-zu-1-Kabel mit einem funktionierenden Programmer verbunden. Jumper JP1 und JP3 sind

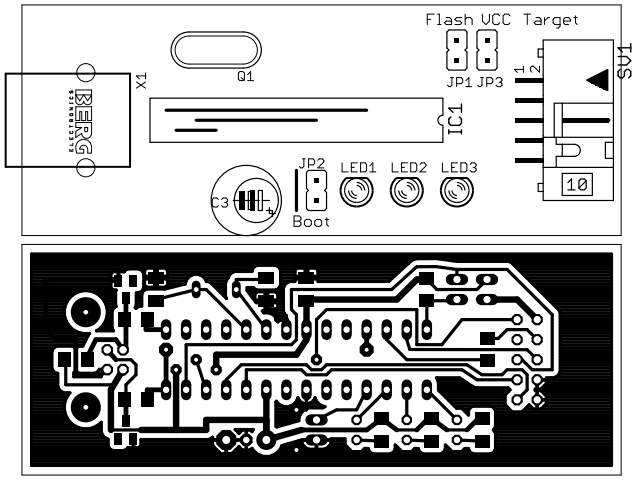


Abbildung 3.4: Bestückung und Ätzvorlage des ISP-Programmieradapters

gesteckt. Es wird der Bootloader des USB-AVR-Lab[2] aufgespielt und die Fuses H:`0xC8` und L: `0xBF` gesetzt. Ab jetzt ist der Programmieradapter unter der „USB AVR-ISP Tool“-Software verfügbar und die STK500-kompatible Firmware kann aufgespielt werden.

Programmierkabel

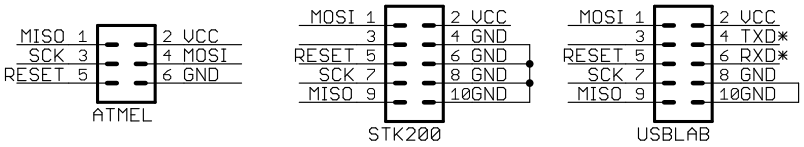


Abbildung 3.5: ISP-Schnittstelle: Verschiedene Standards

Für die ISP-Schnittstelle bei AVRs sind verschiedene Belegungen üblich. Von ATMEL selbst ist eine sechspolige Schnittstelle vorgesehen, wie sie in Abbildung 3.5 (links) vorgesehen ist.

Auf der Entwicklungsplatine „STK200“, die von Kanda entwickelt wurde, ist eine zehnpolige Schnittstelle vorhanden. Obwohl nicht von ATMEL vorgesehen, erfreut sich diese Anschlußbelegung breiter Beliebtheit, so daß sogar die Entwicklungsplatine „STK500“ von ATMEL beide Anschlüsse aufweist. Viele ISP-Programmieradapter verfügen über diese Schnittstelle (siehe Abbildung 3.5 (mitte)), die oft als „STK200-Standard“ bezeichnet wird.

Die Schnittstelle des „USB AVR-Lab“ ist eine leichte Modifikation des „STK200“-Steckers. Zwei Massepins wurden umfunktioniert, um –je nach Firmware– entweder als RS232-kompatible Schnittstelle mit TTL-Pegeln oder als I^2C-Schnittstelle zur Verfügung zu stehen. *Wird das USB AVR-Lab mit einer Leiterplatte verbunden, bei der diese Pins hart auf Masse liegen, müssen diese Leitungen im Programmierkabel unterbrochen werden!*

[2] `http://www.ullihome.de/index.php/USBAVR-ISP-Download/de`

Bezeichnung	Wert	Gehäuse	Reichelt-Best.-Nr.
C1	22pF	1206 o. 0805	NPO-G1206 22P
C2	22pF	1206 o. 0805	NPO-G1206 22P
C3	10μF	2,5mm	RAD 10/35
C4	100nF	1206 o. 0805	X7R-G1206 100N
D1	Z-Diode 3,9V	SOT23	SMD ZD 3,9
D2	Z-Diode 3,9V	SOT23	SMD ZD 3,9
IC1(1)	ATmega 8	DIL-28	ATMEGA 8-16 DIP
IC1(2)	IC-Sockel	DIL-28	GS 28P-S
JP1-JP3(1)	Stiftleiste 2pol	2,5mm	SL 2X25G 2,54
JP1-JP3(2)	Steckbrücke	2,5mm	JUMPER 2,54 GN
LED1	LED grün	2,5mm	LED 3MM 2MA GN
LED2	LED rot	2,5mm	LED 3MM 2MA RT
LED3	LED gelb	2,5mm	LED 3MM 2MA GE
Q1	Quarz 12MHz Grundton	–	12,0000-HC49U-S
R1,R2	68Ω	1206 o. 0805	SMD 1/4W 68
R3	2,2kΩ	1206 o. 0805	SMD 1/4W 2,2K
R4,R5,R10	1,5kΩ	1206 o. 0805	SMD 1/4W 1,5K
R6	10kΩ	1206 o. 0805	SMD 1/4W 10K
R7	1kΩ	1206 o. 0805	SMD 1/4W 1,0K
SV1	Wannenstecker 10pol	–	WSL 10W
X1	USB-Buchse B gewinkelt	–	USB BW

Tabelle 3.3: Stückliste des Programmieradapters

Verbindung mit der Testplatine

Für die Verbindung mit der Testplatine dürfen **keine** Jumper am Programmer stecken! Es ist das Adapterkabel aus Abbildung 3.6 auf der nächsten Seite zu verwenden.

Jumper-Einstellungen

- **JP 1** Programmieradapter flashen: Dieser Jumper wird nur gesteckt, wenn auf den Programmieradapter eine Firmware oder ein Bootloader mittels eines anderen ISP-Programmieradapters aufgespielt wird, was normalerweise nur vor der allerersten Inbetriebnahme erfolgt. Später werden andere Firmwares über den integrierten Bootloader per USB aufgespielt. Der Jumper ist also normalerweise immer *offen*. Ist dieser Jumper versehentlich beim normalen Betrieb gesteckt, startet der Programmer ständig neu und der virtuelle COM-Port wird entweder immer wieder neu oder nie erkannt.
- **JP 2** Bootloader-Modus: Ist dieser Jumper gesteckt geht der Programmieradapter in den sogenannten „Bootloader-Modus“, wobei er darauf wartet, eine neue Firmware per USB aufgespielt zu bekommen. Im Normalbetrieb ist der Jumper also *offen*.
- **JP 3** 5V-Versorgung der Zielschaltung: Dieser Jumper wird gesteckt, wenn die Zielschaltung über den Programmieradapter und damit mit dem USB-Anschluß des PCs mit 5V versorgt werden soll. Diese Option ist nur mit sehr viel Vorsicht zu genießen, da bei vielen Desktop-PCs die 5V-Versorgung des USB-Ports „hart“ an die 5V-Schiene des Netzteils angeschlossen ist. *Ein Kurzschluß in der Zielschaltung oder eine anderweitige*

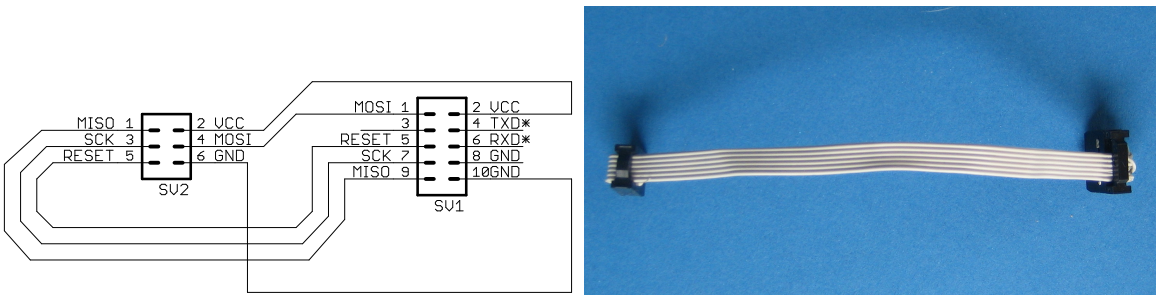
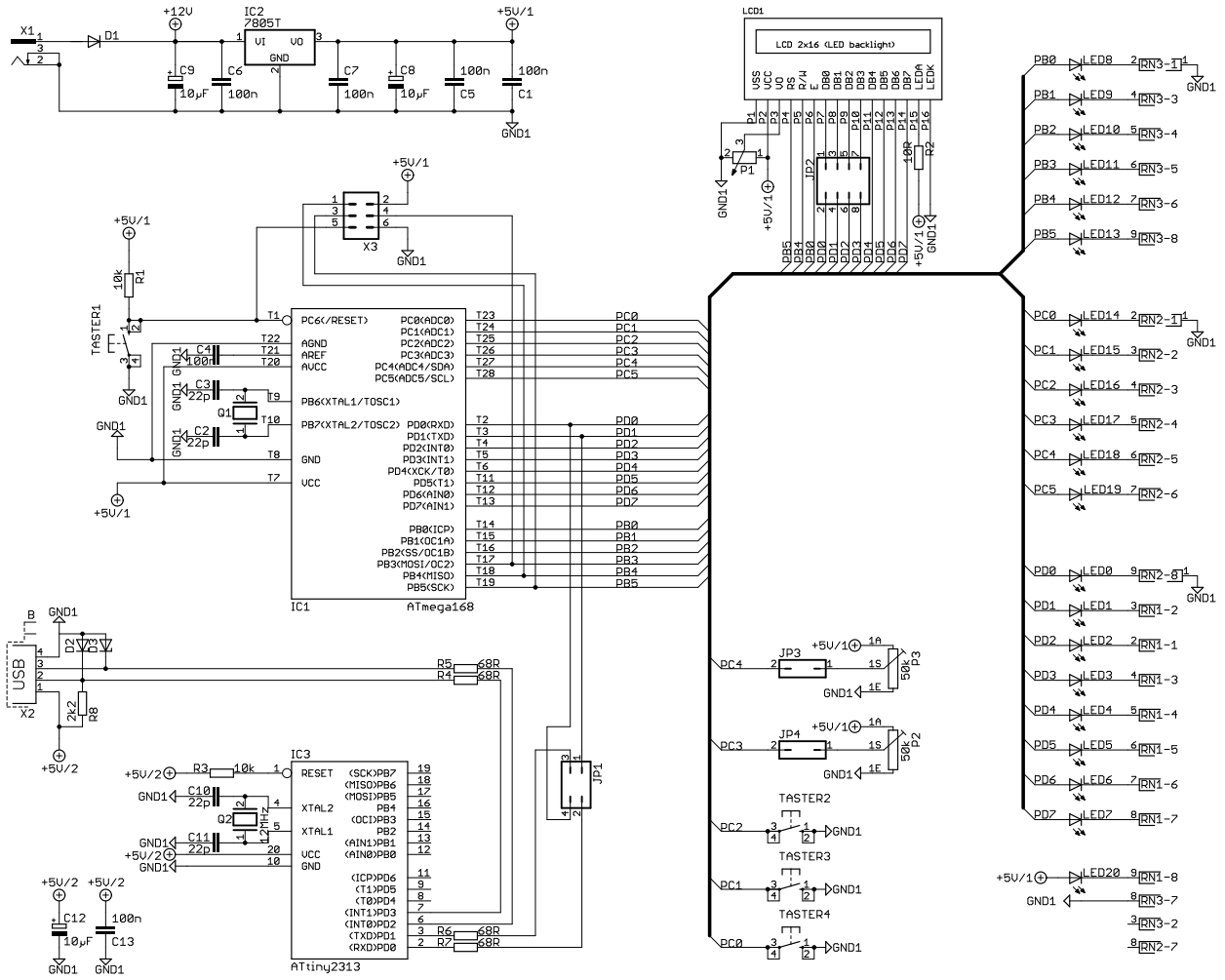


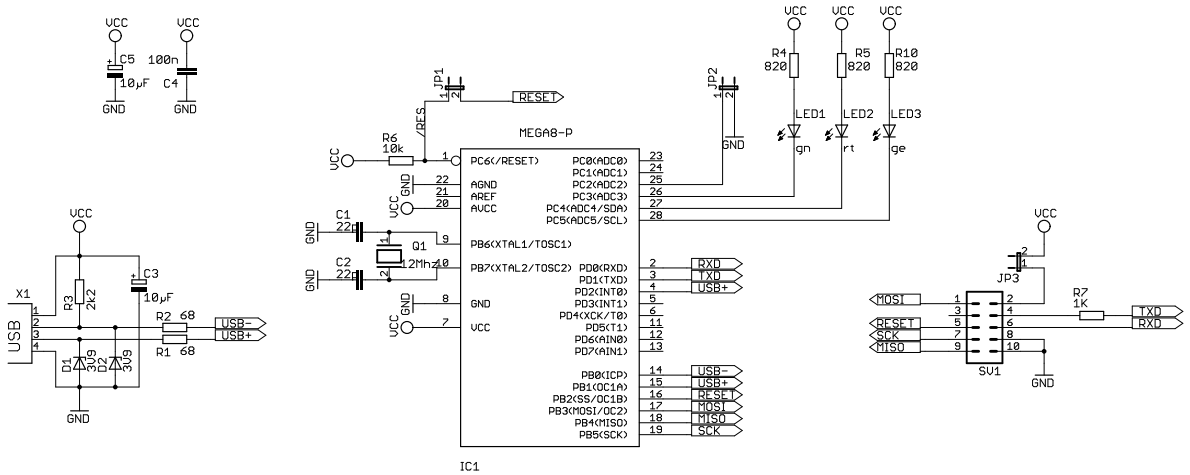
Abbildung 3.6: Programmierkabel: Belegung (links) und praktische Ausführung (rechts)

Versorgungsspannung kann die Zerstörung des PC-Netzteils zur Folge haben! Deshalb ist der Jumper im Normalbetrieb ebenfalls *offen*.

Schaltplan der Testplatine



Schaltplan ISP-Programmieradapter



4 Programmierung

4.1 „Hallo Welt“

```c
// "Hallo Welt": Blinken von LED PC3;
//
// GCC/AVR-Studio kompatibles C
#include <util/delay.h>
#include <avr/io.h>

int main(void)
{
  // Datenrichtungsregister: PC3 ist Ausgang, alle anderen Eingang
  DDRC= 0b00001000;

  // Endlosschleife
  while(1)
  {
    // PC3 ist High, alle anderen Low
    PORTC = 0b00001000;

    // Warteschleife 500ms
    for (int i=0; i<50; i++)
    {
      _delay_ms(10);
    }

    // Alle Pins in Port C sind Low
    PORTC = 0b00000000;

    // Warteschleife 500ms
    for (int i=0; i<50; i++)
    {
      _delay_ms(10);
    }
  }
  return 1;
}
```

Oben ist das „Hallo Welt“, bestehend aus einer blinkenden LED an Pin PC3 in der Programmiersprache C/C++ für Mikrocontroller, wie es der GCC (Gnu Cross Compiler) verwendet, der auch in AVR-Studio eingebunden ist.

4.2 LCDs

Umgang mit Text-LCDs

Text-LCDs stellen eine einfache und elegante Möglichkeit dar, die Ein- und Ausgabemöglichkeiten eines mikrocontrollergesteuerten Geräts zu erweitern. Normalerweise ist ein LCD zwar ein reines Ausgabegerätz, trotzdem werden auch die Eingabemöglichkeiten erweitert (man denke nur an Softkeys). Außerdem lassen sich auch Meßwerte so deutlich eleganter und auch menschenlesbarer als z.B. mit einfachen LED-Anzeigen realisieren.

Mit dem ATmega 168 lassen sich auch einfache Grafikdisplays ansteuern (zum Vergleich: Display 64 × 128 Pixel s/w ≙ 1kb). Viele Textdisplays sind mehr oder weniger „HD44780“-kompatibel. Diese lassen sich besonders einfach ansteuern, da keine besonderen Anforderungen an das Timing gestellt werden; zudem ist die Schnittstelle auch außerordentlich gut dokumentiert.

Im Großen und Ganzen ist die Pinbelegung immer recht ähnlich, für die exakte Pinbelegung ist in jedem Fall das Datenblatt zu konsultieren. Insbesondere die Beschaltung für die Kontrastspannung und für die Hintergrund-LEDs variiert.

Ein solches Display liegt auch dem Bausatz bei. *Achtung: Die Pinbelegung ist zwar standardgemäß, die Reihenfolge ist jedoch ungewöhnlich! Deshalb sollten die Pins mit einem Filzstift markiert werden!*

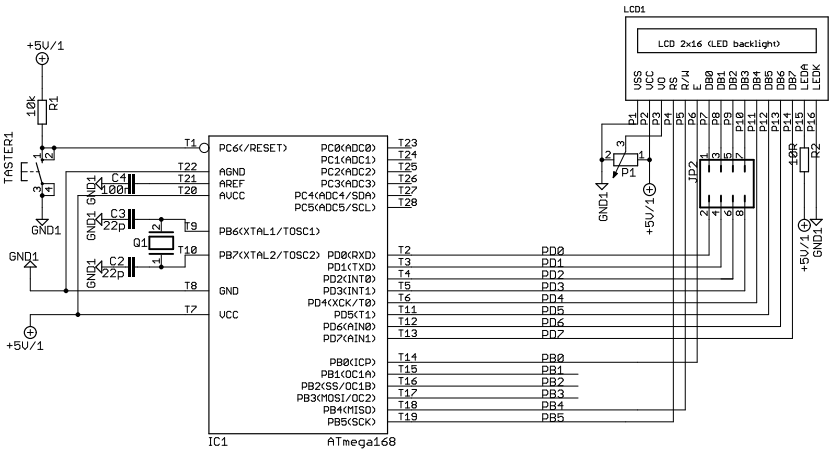


Abbildung 4.1: Anschlußschema des LCDs

Ansteuerung per Microcontroller

Die Ansteuerung eines HD44780-basierten Displays gestaltet sich sowohl was den Hardware- als auch den Softwareaufwand angeht extrem einfach und eignet sich somit sehr gut als Übungsprojekt für Anfänger.

Das HD44780-Interface besteht aus 8 Datenleitungen(D0-D7) sowie den 3 Steuerleitungen RS (Register Select), R/W (Read/Write) und E (Enable). Die Displays werden mit 5V Betriebsspannung versorgt, die Kontrastspannung V0 bekommt man, indem man ein 10k-Poti zwischen VCC und GND anschließt und den mittleren Anschluß als Kontrastspannung verwendet. Bei ca. 4,5 Volt werden dann die Pixel sichtbar. Optional haben viele Displays noch eine LED-Beleuchtung eingebaut, diese benötigt oft einen Vorwiderstand, um an 5V angeschlossen zu werden. Möchte man nur einen 8-Bit-Port zur Ansteuerung verwenden, so ist dies möglich, indem man lediglich die Pins D4-D7 sowie die 3 Steuerleitungen an den Controller anschließt, das Display muß dann im 4-Bit-Modus betrieben werden.

Auf der Testplatine sind alle Leitungen für den 8-Bit-Modus angeschlossen (siehe Abbildung 4.1). Werden die vier Jumper von JP2 abgezogen lassen sich die Portleitungen PD0 – PD3 anderweitig verwenden (z.B. für serielle Datenübertragung).

Grundlage Datenübertragung: 8-bit-LCDs

- Byte liegt an Pins an (8 bit)
- Enable-Impuls veranlaßt den LCD-Controller, das Byte einzulesen
- R/S-Pin zeigt an, ob es sich bei dem Byte um einen Befehl oder um ein Datum handelt.
- R/W-Pin wird nur gebraucht, wenn auch Daten vom Displaycontroller eingelesen werden.
- Das Display benötigt also 11 Datenleitungen

Grundlage Datenübertragung: 4-bit-LCDs

- Halbbyteweise Datenübertragung
- Höherwertiges Halbbyte liegt an den Pins an
- R/S-Pin zeigt an, ob es sich bei dem Byte um einen Befehl oder um ein Datum handelt.
- Enable-Impuls veranlaßt den LCD-Controller, das halbe Byte einzulesen
- Niederwertiges Halbbyte liegt an den Pins an
- R/S-Pin muß unverändert bleiben
- Enable-Impuls veranlaßt den LCD-Controller, das zweite halbe Byte einzulesen
- Das Display braucht also 7 Datenleitungen

4.2.1 Initialisierung

Initialisierung im 8 Bit Modus

Im 8 Bit Modus kann die Initialisierungssequenz in Form kompletter Bytes übertragen werden:

- Nach dem Anlegen der Betriebsspannung muß eine Zeit von mindestens ca. 15ms gewartet werden, um dem LCD-Kontroller Zeit für seine eigene Initialisierung zu geben
- `0x30` ins Steuerregister schreiben (RS = 0)
- mindestens 4.1ms warten
- `0x30` ins Steuerregister schreiben (RS = 0)
- mindestens 100μs warten
- 0x30 ins Steuerregister schreiben (RS = 0)
- Mit dem Konfigurier-Befehl `0x30` das Display konfigurieren (8-Bit, 1 oder 2 Zeilen, 5x7 Format) mit den restlichen Konfigurierbefehlen die Konfiguration vervollständigen: Display ein/aus, Cursor ein/aus, etc.

Initialisierung im 4 Bit Modus

Achtung: Im folgenden sind alle Bytes aus Sicht des LCD-Kontrollers angegeben! LCD-seitig werden die Leitungen DB4 – DB7 verwendet, angeschlossen an PORT D an PD4 – PD7. Deshalb muß das Halbbyte (=Nibble) um vier Bits nach links geschoben werden.

Die Sequenz, aus Sicht des Kontrollers, sieht so aus:

- Nach dem Anlegen der Betriebsspannung muß eine Zeit von mindestens ca. 15ms gewartet werden, um dem LCD-Kontroller Zeit für seine eigene Initialisierung zu geben
- `0x3` ins Steuerregister schreiben (RS = 0)
- mindestens 4.1ms warten
- `0x3` ins Steuerregister schreiben (RS = 0)
- mindestens 100μs warten
- `0x3` ins Steuerregister schreiben (RS = 0)
- `0x2` ins Steuerregister schreiben (RS = 0), dadurch wird auf 4 Bit Daten umstellt

Ab jetzt können Befehle als ganze Bytes übertragen werden, indem jeweils das höherwertige Nibble und dann das niederwertige Nibble übertragen werden. Mit dem Konfigurier-Befehl `0x20` das Display konfigurieren (4-Bit, 1 oder 2 Zeilen, 5x7 Format) mit den restlichen Konfigurierbefehlen die Konfiguration vervollständigen: Display ein/aus, Cursor ein/aus, etc.

4.3 A/D-Wandler

Wie die meisten ATmegas verfügt der ATmega168 über einen Analog-Digital-Wandler, der an die Pins des Port C angeschlossen ist. Bei der Version im DIL-Gehäuse sind für den A/D-Wandler sechs Kanäle an den Pins 23 bis 28 verfügbar, beim TQPF- und beim MLF-Gehäuse noch zusätzlich an den Pins 19 und 22, dort stehen also acht Kanäle zur Verfügung. Die Auflösung beträgt 10 Bit. (Zum Vergleich: Bei einer Referenzspannung von 5V sind das $\approx$ 5mV!)

Eine Referenzspannung V_{ref} kann entweder extern eingespeist werden oder vom ATmega selbst zur Verfügung gestellt werden. Eine externe Referenzspannung darf nicht höher als die Versorgungsspannung V_{CC} sein. Die interne Referenzspannung entspricht entweder der Versorgungsspannung oder 1,1V (ATmega168). In diesem Fall sollte an dem Pin $AREF$ ein 100nF-Kondensator gegen Masse angebracht werden.

Die gemessene Spannung muß zwischen 0V und V_{ref} liegen. Da bei der Testplatine am Poti zwischen 0V und 5V abgegriffen werden kann, wird hier die Referenzspannung mit $V_{\text{ref}} = V_{\text{CC}}$ gewählt.

Der A/D-Wandler verfügt über seinen eigenen Takt, der einem ganzzahligen Bruchteil der Systemtaktes entsprechen muß. Er wird über einen Vorteilungfaktor („Prescaler“) gegenüber dem Systemtakt eingestellt. Der eigene Taktgeber ermöglicht es, während des Wandlungsvorganges den CPU-Takt und Teile der Peripherie anzuhalten, um Beeinflussungen des Meßvorgangs durch den Controller selbst zu minimieren (siehe Datenblatt, Stichwort „ADC Noise Reduction Mode“). Bei 10 Bit Auflösung muß der ADC-Clock zwischen 50kHz und 200kHz liegen. Bei weniger als 10 Bit Auflösung kann der Clock etwas höher gewählt werden.

Gestartet werden kann die A/D-Wandlung entweder über Software-Befehle oder Signalflanken. Soll immer der gleiche Kanal abgefragt werden, bietet sich der sogenannte „free running mode“ an. Dabei wird die anliegende Spannung zyklisch mit der höchstmöglichen Frequenz gewandelt und das ADC-Datenregister immer direkt upgedatet.

Wird noch während eines Wandlungsvorgang ein anderer Kanal gewählt, wird die gestartete Wandlung auf dem alten Kanal noch fertiggestellt, bevor gewechselt wird; im Datenregister ist also das Ergebnis vom alten Kanal!

Ein normaler *Sample*-Vorgang braucht 13 ACD-Clock-Zyklen. Bei der ersten Konvertierung, nachdem der ADC eingeschaltet wurde, werden 25 Zyklen benötigt. Nach $2\frac{1}{2}$ Taktzyklen wird das Signal in eine „sample and hold“-Stufe eingelesen und anschließend gewandelt. Damit ist gewährleistet, daß sich das Signal während des Wandelvorgangs nicht mehr ändert.

Das A/D-Wandler wird im Datenblatt auf den Seiten 244 – 259 ausführlich behandelt.

```
1  /* Initialisierung ADC */
2
3  char hilf = 0;
4  // Multiplexer-Kanal ADC3:
5  hilf  |= (0 << MUX3) | (0 << MUX2) | (1 << MUX1) | (1 << MUX0);
```

```
6
7   hilf |=  (1 << REFS0);  // Referenzspannung: AVcc mit externem
8   hilf |=  (0 << REFS1);  // Kondensator an ARef (Datenblatt S. 256)
9
10  hilf |=  (0 << ADLAR);  // Ergebnis linksbündig ausrichten
11
12  ADMUX = hilf;
13
14  hilf   = 0;             // Für ADCSRB
15  hilf  |= (0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0); // Free running mode.
16  ADCSRB = hilf;
17
18  hilf   = 0;             // Für ADCSRA
19  // 16 MHz / 0,2 MHz = 80 -> Teilungsfaktor 128 (Datenblatt S. 258)
20  // 16 MHz / 128 = 125 kHz ADC-Clock
21  hilf  |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
22  hilf  |= (1 << ADATE);  // Auto-Trigger nach ADTSx in ADCSRB
23  hilf  |= (1 << ADEN);   // ADC einschalten
24  hilf  |= (1 << ADSC);   // Wandlung starten
25
26  ADCSRA = hilf;
27
28  DIDR0 |= (1 << ADC3D);  // Digitaler Input ADC4 aus
29
30  ...
31
32  while(1)
33  {
34   ...
35   int ergebnis = ADC;
36   ...
37  }
```

4.4 Die „Arduino“-Entwicklungsumgebung

Die Arduino-Entwicklungsumgebung mit Bootloader verwendet, genau wie AVR-Studio den GCC, bietet aber jede Menge High-Level-Befehle, die den Umgang für mit Elektronik oder Programmierung nicht so sehr Bewanderte erleichtern. Das Projekt kommt aus Spanien und hat mitlerweile im Internet eine recht große Fangemeinde. Die Entwicklergemeinde ist auch recht aktiv, so wurde die Entwicklungsumgebung von Version 7 im April mitlerweile auf Version 10 überarbeitet. Einziger Wermutstropfen: Sie basiert auf Java. Java hat uns gezeigt, wofür man Gigahertz-Prozessoren wirklich braucht: Um die Dinge, die auf einem 286er mit Turbopascal funktionierten mit der gleichen Geschwindigkeit unter Java zu realisieren. Als Vorteil

läßt sich jedoch anführen, daß sie für alle größeren Betriebssysteme (Windows, Linux, MacOS, FreeBSD) verfügbar ist und notfalls auch selbst kompilierbar, da frei und OpenSource.

Installation und Einstellungen

Die Entwicklungsumgebung bedarf keiner Installation; es reicht das Kopieren in ein beschreibbares Verzeichnis. Da der USB-seriell-Wandler nur die Baudrate 9600 mit (8N1, kein Handshake) unterstützt, die gängigen Arduino-Boards jedoch mit 19200 Baud betrieben werden, wird eine weitere Hardware hinzugefügt. Dazu werden einfach in der Datei `./hardware/boards.txt` die folgenden Zeilen hinzugefügt (Arduino-Entwicklungsumgebung vorher schließen):

```
##############################################################

eval.name=Eval-Board ATmega168 Bootloader

eval.upload.protocol=stk500
eval.upload.maximum_size=14336
eval.upload.speed=9600

eval.bootloader.low_fuses=0xff
eval.bootloader.high_fuses=0xdd
eval.bootloader.extended_fuses=0x00
eval.bootloader.path=atmega168
eval.bootloader.file=ATmegaBOOT_168_ng.hex
eval.bootloader.unlock_bits=0x3F
eval.bootloader.lock_bits=0x0F

eval.build.mcu=atmega168
eval.build.f_cpu=16000000L
eval.build.core=arduino

##############################################################
```

- „Preferences“: Pfad der „Sketche“, also der Quelltexte nach Belieben einstellen
- „Microcontroller (MCU)“ unter „Tools“ auf ATMEGA 168
- „Serial Port“ unter „Tools“

Das Hauptfenster ist ein Texteditor mit Syntax-Highlighting, automatischer Einrückung des Quelltextes und den üblichen Copy-Paste- und Speicherfunktionen. Außerdem sind ein Terminalprogramm und Schaltflächen für das direkte Hochladen des Codes in den μC vorhanden.

„Hallo Welt“

Das „Hallo Welt“-Programm (blinkende LED) in der „Arduino programming language“ sieht deutlich einfacher als das in C aus. Insbesondere müssen keine zusätzlichen Bibliotheken für die Wartefunktion eingeladen oder direkte Bitmanipulationen durchgeführt werden. Die „Arduino Programming Language“ ist eigentlich normales C für Mikrocontroller; es wird auch der normale GCC-Compiler verwendet, der auch in AVR-Studio verwendet wird, aber eben mit einigen selbstredenden Befehlen, die dem Anfänger das Vorstellungsvermögen fast abnehmen:

```
1  // "Hallo Welt": Blinken von LED PB5 = PIN13
2  //
3  // GCC/Arduino
4
5  void setup()
6  {
7    // Datenrichtungsregister: Pin 13 ist Ausgang
8    pinMode(13,OUTPUT);
9  }
10
11 void loop()
12 {
13  // Pin 13 ist High, alle anderen Low
14  digitalWrite(13, HIGH);
15
16  // Warteschleife 500ms
17  delay(500);
18
19  // Pin 13 wieder auf Low
20  digitalWrite(13, LOW);
21
22   // Warteschleife 500ms
23  delay(500);
24  }
```

Das Ganze sollte unter einem aussagekräftigen Namen abgespeichert werden. Die linke Schaltfläche (Pfeil) startet den Compiler. Im idealen Fall läuft dieser mit der Meldung: `Binary sketch size: 1104 bytes (of a 14336 byte maximum)` ohne Fehler und Warnungen durch. Wenn als Erfolgsmeldung „`854 bytes (of a 7168)`“ erscheint, ist noch ein ATMEGA 8 eingestellt. Sollte der Compiler in einer Endlosschleife gefangen werden, läßt sich der Compilerlauf mit dem „Stop“-Knopf anhalten.

Hochladen des Programmes in den μC

Wenn der Quelltext erfolgreich kompiliert wurde, kann er auf den Mikrocontroller hochgeladen werden. Dazu ist der zweite Knopf von oben rechts, „Upload to I/O Board“ wichtig. Die Funktion des Hochladens des Quelltextes ist eng mit dem Bootloader verknüpft.

Nach einem Reset wartet der Controller für mehrere hundert Millisekunden, ob Daten an der seriellen Schnittstelle anliegen. Liegen keine Daten an, startet der Mikrocontroller sein internes Programm. Liegen Daten an, wartet der Controller, ob ein Schlüsselwort zur Programmierung vorhanden ist. Ist dieser Schlüssel vorhanden, überschreibt der Bootloader das interne Programm mit dem, das über die serielle Schnittstelle übertragen wird.

Theoretisch wäre also folgender Ablauf zum Hochladen eines neuen Programmes notwendig: Reset des Mikrocontrollers (Reset-Knopf, Strom aus oder interner Reset) und Hochladen des neuen Programmes. Da allerdings bei Betätigung der „Upload to I/O Board“-Schaltfläche der Compiler gestartet und der Quelltext noch einmal kompiliert wird, muß das Timing (Reset–Hochladen) ausprobiert werden. (Bei meinem Rechner betätige ich erst die Schaltfläche und danach gemütlich den Reset-Knopf.)

Ergebnis sollte jetzt eine mit ca. 5Hz blinkende rote LED sein. Das neue Programm startet übrigens ohne weiteren Reset.

Die Arduino-Sketches lassen sich sowohl wie oben beschrieben über den Bootloader, als auch normal mit dem Programmieradapter hochladen.

Bemerkungen

Der Vorteil der „Arduino“-Entwicklungsumgebung liegt im Vorhandensein vieler High-Level-Befehle, die vor allem dem Anfänger die Arbeit sehr erleichtern und die Fehlerhäufigkeit herabsetzen. Außerdem ist die Dokumentation (in Englisch) sehr ausführlich und gut geschrieben.

Eine immer noch wachsende Community stellt teils sehr anspruchsvolle Projekte im sogenannten „Playground“ vor, so daß diese meist mühelos nachvollzogen werden können.

Wenn man nicht gerade mit dem Englischen auf Kriegsfuß steht, kann der Einstieg in Mikrocontroller über die „Arduinos“ nur empfohlen werden.

5 Schaltungen und Flashen

Dieses Kapitel befaßt sich mit der Frage: „Wie bekomme ich die fertige `*.hex`-Datei in den Controller?“ Für die meisten modernen μCs gibt es mehrere verschiedene Wege.

5.1 Externer Programmer

Der „klassische“ Weg der Programmierung eines μCs besteht in der Benutzung eines externen Programmiergerätes, meist in Anschluß an einen PC, es gibt aber auch „Stand-Alone“-Geräte. Der Controller wird in das Programmiergerät eingesetzt, die Firmware wird aufgespielt und der Controller anschließend in seine Zielschaltung eingesetzt.

Dieser Weg ist sehr praktikabel, wenn es darum geht, fertige Software in viele μCs einzuspielen und wurde z.B. angewandt, um die ATtiny2313 auf der Testplatine mit der USB-Firmware zu flashen. Einige Programmiermodi, z.B. „High-Voltage-Programming“, mit dem man auch μCs, bei denen Taktgeber oder die anderen Programmierschnittstellen abgeschaltet hat („verfust“) wieder neu bespielen kann, funktionieren nur in einem externen Programmiergerät.

Soll die Software für eine Zielschaltung jedoch erst entwickelt werden, müssen die Controller für jeden Entwicklungsschritt aus der Schaltung herausgenommen, ins Programmiergerät eingesetzt, neu bespielt und wieder in die Zielschaltung eingesetzt werden.

5.2 In-System-Programming

Der oben beschriebene Weg, neue Firmware auf einen μC aufzuspielen ist nicht nur manchmal recht umständlich, sondern im Fall von fest verlöteten μCs sogar unmöglich.

AVRs von ATMEL lassen sich in der Zielschaltung programmieren. Dieser Vorgang wird „*In-system programming*“ (ISP) genannt. Dafür wird ein Programmieradapter, „*In-system programmer*“ (ISP), gelegentlich auch „*In-circuit programmer*“ (ICP) genannt, benötigt. Bei den ATMEL-Mikrocontrollern erfolgt die Übertragung des Programms in den Mikrokontroller in der Zielschaltung seriell, so daß diese Geräte auch als „*In-circuit serial programmer*“ (ICSP) bezeichnet werden. Um die Verwirrung komplett zu machen, wird auch die Schnittstelle in der Zielschaltung, über die die Programmierung erfolgt, abwechselnd als „ISP“ und „ICSP“ bezeichnet.

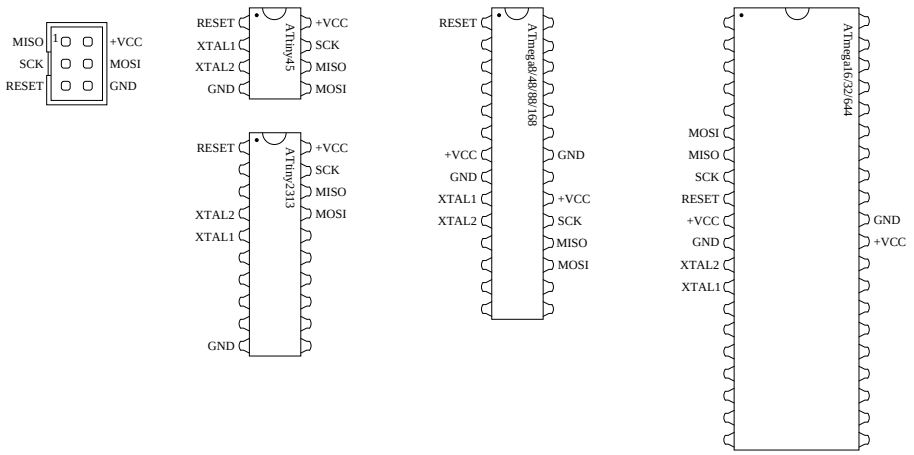


Abbildung 5.1: Beschaltung zum Flashen einiger AVRs

Die vier Pins des μC, die für die ISP-Schnittstelle gebraucht werden, können in der Schaltung auch anderweitig verwendet werden, dabei ist jedoch sicherzustellen, daß während des Programmiervorgangs keine Beeinflussung durch die angeschlossene Schaltung erfolgt. Bei einer Störung des Programmiervorgangs kann man sich durchaus aus dem Controller selbst aussperren, so daß man nur noch über „High-Voltage-Programming“ neu programmieren kann. (Letzteres geht allerdings immer, solange der μC überhaupt noch funktionstüchtig ist).

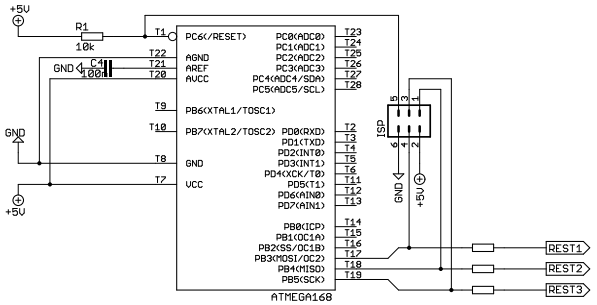


Abbildung 5.2: Entkopplung des ISPs vom Rest der Schaltung

ATMEL empfiehlt, bei den ISP-Pins den Rest der Schaltung über Vorwiderstände anzuschließen, um eine Beeinflussung des Programmiervorgangs auszuschließen (siehe Abbildung 5.2).

5.3 Bootloader

Alle neueren AVR Mikrocontroller der ATmega-Serie besitzen die Möglichkeit, einen sogenannten Bootloader zu installieren. Dies ist ein Programm, welches in einem separaten Bereich des Flash-Speichers abgelegt ist und als erstes nach einem Reset gestartet wird. Es ermöglicht die benutzerfreundliche Installation einer neuen Software ohne spezielle Programmierkabel und

ohne Öffnen des Gerätes. Hier wird einfach eine oft ohnehin vorhandene serielle Schnittstelle verwendet; der Bootloader sorgt dafür, daß die Software sich selbst überschreibt. Der Bootloader selbst muß natürlich einmal durch herkömmliche Programmieradapter/Geräte auf den ATmega gespielt werden.

Ein Bootloader bietet sich an, wenn ohnehin eine serielle oder USB-Schnittstelle vorhanden ist.

Wird der Arduino-Bootloader[1] auf der Testplatine verwendet, muß dieser neu kompiliert werden, um Sketches mit 9600 Baud heraufladen zu können. Dazu wird die folgende Definition angepaßt:

```
#define BAUD_RATE   9600
```

Außerdem muß in der „Arduino“-Entwicklungsumgebung die Geschwindigkeit auf 9600 Baud angepaßt werden (siehe Kapitel 4.4 auf Seite 22).

5.4 Fuse- und Lockbits

Die AVR-Mikrocontroller besitzen verschiedene Speicher„arten“:

- Flash
- EEPROM
- SRAM
- Fuse- und Lockbytes

Der Flash-Speicher enthält den (fast) unveränderlichen Maschinencode der Programme, das EEPROM dient dazu, Daten zu speichern, daß sie auch nach Entfernen der Versorgungsspannung noch erhalten bleiben und das SRAM enthält den Programmspeicher zur Laufzeit.

Die Fusebytes sind Speicherstellen, mit denen sich das globale Verhalten des μCs einstellen läßt, z.B. Oszillator, Watchdog, An- und Abschalten einzelnder Programmierschnittstellen ....

Durch Verstellen der Fusebits eines AVRs ist es unmöglich, diesen zu zerstören. Es ist allerdings möglich, den Controller so einzustellen, daß man mit den normalen Werkzeugen nicht mehr darauf zugreifen kann, ein „verfuster“ Controller kann also mit dem richtigen Werkzeug grundsätzlich wieder repariert werden. Ob dies jedoch wirtschaftlich sinnvoll ist, ist allerdings eine andere Frage.

Manche Programmiersoftware, z.B. AVRdude oder Ponyprog, erlaubt es ohne weitere Nachfrage, die Fuse-Bits so zu setzen, daß kein Zugriff mehr möglich ist („verfust“). Vor dem Verändern der Fusebits sollte man also besser vorher wissen, was damit eingestellt wird. Auch

[1] `http://svn.berlios.de/viewcvs/arduino/trunk/hardware/bootloaders/atmega168/`

sollten die Fuse-Bytes direkt komplett richtig gesetzt werden, nicht nur einzelne Bits verändert, da möglicherweise unangenehme Zwischenzustände auftreten können.

Für den Anfang bietet es sich also an, einen „Fuse-Calculator“ zu benutzen. Ein solches Werkzeug ist in AVR Studio seit Version 4.13 Service Pack 2 enthalten. Ansonsten gibt es auch Online-Berechnungswerkzeuge[2].

Die Fusebits haben eine „invertierte Logik“. Ein programmiertes Fusebit ist nicht wie man annehmen könnte auf "1"gesetzt sondern auf „0“. Eine unprogrammierte Fuse ist „1“. Manche Programme stellen programmierte Fusebits mit einem gesetzten Häkchen dar (z.B. PonyProg), welches dann als eine „0“ (=programmed) interpretiert wird. Möchte man das Fusebit „setzen“ oder „programmieren“, also auf „0“ setzen, muß man das Häkchen setzen. Meine persönliche Meinung ist, daß AVRs und PonyProg am besten zusammenpassen, indem man es nicht benutzt.

[2] `http://palmavr.sourceforge.net/cgi-bin/fc.cgi`

6 Kleines C-Lexikon

6.1 Operatoren

Zuweisungoperatoren

- `=` Zuweisungsoperator; Verwendung: `a = b; a = b = c;`
- `+=` Addition mit Zuweisung: `a += b;` entspricht `a = a + b;`
- `-=` Subtraktion mit Zuweisung: `a -= b;` entspricht `a = a - b;`
- `*=` Multiplikation mit Zuweisung: `a *= b;`
- `/=` Division mit Zuweisung: `a /= b;`
- `%=` Modulo-Operation mit Zuweisung: `a %= b;`
- `<<=` Zuweisung nach bitweisem Linksschieben
- `>>=` Zuweisung nach bitweisem Rechtsschieben
- `&=` Zuweisung nach bitweisem AND
- `|=` Zuweisung nach bitweisem OR
- ∧`=` Zuweisung nach bitweisem XOR (exklusivem OR)
- `++` Inkrement-Operator: `i++;` entspricht `i += 1;`
- `-` Dekrement-Operator: `i--;`

Relationale Operatoren

- `<` kleiner als
- `<=` kleiner oder gleich
- `>` größer als
- `>=` größer oder gleich
- `==` gleich
- `!=` ungleich

Relationale Operatoren werden nur in logischen Ausdrücken verwendet! Beispiel: `a = b == c;`, nicht zu verwechseln mit `a = b = c;`!

Rechenoperatoren

- `+` Addition
- `-` Subtraktion
- `*` Multiplikation
- `/` Division
- `%` Modulo/Divisionsrest

`+`, `-`, `*` und `/` sind auf ganzzahlige und reelle Operanden anwendbar, `%` ist nur für ganzzahlige Operanden einsetzbar. Es ist auf die Datentypen zu achen, die Division zweier Ganzzahlen (Integer) ergibt immer eine ganze Zahl!

6.2 Library-Funktionen

`_delay_ms` und `_delay_us` aus util/delay.h

Das Warten durch „verbraten“ von Prozessorzyklen kann mit großer Genauigkeit anhand der avr-libc Bibliotheksfunktionen `_delay_ms()` und `_delay_us()` erledigt werden. Die Bibliotheksfunktionen sind einfachen Zählschleifen (Warteschleifen) vorzuziehen, da leere Zählschleifen ohne besondere Vorkehrungen sonst bei eingeschalteter Optimierung vom avr-gcc-Kompiler wegoptimiert werden. Die Bibliotheksfunktionen verwenden die in `F_CPU` definierte Taktfrequenz.

Diese Bibliotheksfunktionen funktionieren nur dann korrekt, wenn sie mit zur Übersetzungszeit (beim Kompilieren) bekannten konstanten Werten aufgerufen werden. Die Wartezeit der Funktion `_delay_ms()` ist auf 262,14ms/`F_CPU` (in MHz) begrenzt, d.h. bei 16 MHz kann man nur max. 16,3ms warten. Die Wartezeit der Funktion `_delay_us()` ist auf 768μs/`F_CPU` (in MHz) begrenzt, d.h. bei 16 MHz kann man nur max. 48μs warten. Längere Wartezeiten müssen dann über einen mehrfachen Aufruf in einer Schleife gelöst werden. Der Quellcode muß mit eingeschalteter Optimierung übersetzt werden, sonst wird sehr viel Maschinencode erzeugt und die Wartezeiten stimmen nicht mehr mit dem Parameter überein.

`printf` aus stdio.h

Die Bilbliothek „`stdio.h`“ stellt unter anderem die `printf`-Funktion zur Verfügung. Die Formatierung bei `printf` wird mittels Formatierungszeichen realisiert.

Häufig benutzte Zeichen sind:

%d, %i	Decimal signed integer
%o	Octal integer
%x ,%X	Hex integer
%u	Unsigned integer
%c	Character
%s	String (siehe unten)
%f	double
%e ,%E	double
%g , %G	double
%p	Zeiger
%n	Number of characters written by this printf
	No argument expected
%%	% . No argument expected

Als Parameter muß immer ein Wert, kein Zeiger übergeben werden. (Ausnahme: `%`; bei einem String wird erwartet, daß ein Zeiger übergeben wird.

Analog zu `printf` funktioniert `snprintf`. Die Syntax lautet:

```
snprintf(%s,%n,Argumente)
```

Im Gegensatz zu `sprintf()` wird nur eine endliche Anzahl `%n` Zeichen in den String `%s` geschrieben.

Beispiele:

```
snprintf(s,40,"Der HEX-Code lautet: %#X", zahl);
snprintf(s,40,"Zahl 1 ist %d, Zahl 2 ist %d, zahl1, zahl2);
```

Zusätzliche Formatierungen (Auszug):

-	linksbündig
0	Felder mit 0 ausfüllen (an Stelle von Leerzeichen)
+	Vorzeichen einer Zahl immer ausgeben
blank	positive Zahlen mit Leerzeichen beginnen
#	verschiedene Bedeutung: %#o (Oktal): 0 Präfix wird eingefügt %#x (Hex): 0x Präfix wird bei Werten $\neq$ Null eingefügt %#X (Hex): 0X Präfix wird bei Werten $\neq$ Null eingefügt %#e : Dezimalpunkt immer anzeigen %#E : Dezimalpunkt immer anzeigen %#f : Dezimalpunkt immer anzeigen

6.3 Datentypen und Funktionen

`volatile` volatile wirkt zusammen mit Typdeklarationen als Modifizierer. Beispiel:

```
volatile int a;
```

Das Schlüsselwort `volatile` teilt dem Compiler mit, daß die mit `a` bezeichnete Variable mit dem Datentyp `int` durch Ereignisse außerhalb der Kontrolle des Programms verändert werden kann. Der Wert der Variablen muß deshalb vor jedem Zugriff neu aus dem Hauptspeicher eingelesen werden, d.h. er darf nicht in einem Register des Prozessors zwischengespeichert werden.

Der Compiler arbeitet bei mit volatile deklarierten Variablen ohne jede Optimierung, d.h. läßt die entsprechenden Werte bei jedem Zugriff neu aus dem Hauptspeicher laden und sorgt bei Veränderungen dafür, daß die neuen Werte ohne Verzögerung dort sofort abgelegt werden.

inline Bei der Verwendung von Funktionen entsteht ein gewisser Aufwand an Laufzeit für den Funktionsaufruf, die Parameterübermittlung und die Wertrückgabe. Bei sehr kurzen Funktionen kann dieser „Overhead“ wesentlich werden im Vergleich zum Aufwand für die Abarbeitung der eigentlichen Funktion. Andererseits sorgt das häufige Kopieren und Einfügen von gleichem Code an verschiedene Stellen des Quelltextes für Unübersichtlichkeit. Zur Lösung dieses Konflikts bieten sich folgende Kompromisse an:

- Verwendung von Makros:
- Inline-Funktionen

Der Nachteil von Makros ist, daß keine Typüberprüfung bei der Benutzung solcher „Funktionen“ stattfindet.

Inline-Funktionen sind normalerweise nur von C++, nicht jedoch von C vorgesehen. Nichtsdestotrotz werden sie vom GCC auch unter normalem C unterstützt, nicht jedoch von der Arduino-Umgebung. Die Inline-Funktionen werden vom Compiler - unter Einschluß der Typüberprüfung - an allen Aufrufstellen textlich ersetzt. Ab einer gewissen Komplexität kann ein C++-Compiler die Inline-Aufforderung zurückweisen: Die Funktion wird wie eine normale Funktion behandelt. Dies ist zum Beispiel generell bei rekursiven Aufrufen der Fall.

6.4 Dies und das

Einzelne Bits setzen und löschen

- `x |= (1 << Bitnummer);` Bit setzen
- `x &= (1 << Bitnummer);` Bit löschen

Will man, so wie in Assembler, nur einzelne Bits setzen oder löschen bieten sich die folgenden Makros an:

```
#define CBI(x,y)      x&=~(1<<y)    // Bit loeschen
#define SBI(x,y)      x|= (1<<y)    // Bit setzen

SBI(PORTB,3);                       // Bit 3 in PORTB setzen
CBI(PORTB,4);                       // Bit 4 in PORTB loeschen
```

Reservierte Wörter

Die folgenden Wörter sind in C/C++ reserviert und dürfen deshalb nicht als Funkions- oder Variablennamen verwendet werden:

- asm
- auto
- bool
- break
- case
- catch
- char
- class
- const
- const_cast
- continue
- default
- delete
- do
- double
- dynamic_cast
- else
- enum
- explicit
- extern
- false
- float
- for
- friend
- goto
- if
- inline
- int
- long
- mutable
- namespace
- new
- operator
- private
- protected
- public
- register
- reinterpret_cast
- return
- short
- signed
- sizeof
- static
- static_cast
- struct
- switch
- template
- this
- throw
- true
- try
- typedef
- typename
- union
- unsigned
- using
- virtual
- void
- volatile
- wchar_t
- while

7 Weiterführende Literatur

Da das μC-Seminar nur eine Einführung in die Welt der Mikrocontroller bieten kann, folgt hier ein Überblick über bislang gesichtete Literatur zu diesem Thema.

„Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie“, *Günter Schmitt*, [5]

Für die Einführung in die μCs mit der Programmiersprache C halte ich das Buch [5] „Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie“ für das Beste, was ich in dieser Richtung gelesen habe. Dieses Buch besteht aus einem Assembler- und einem C-Teil, wobei man den Assembler-Teil getrost weglassen kann, ohne daß beim C-Teil Verständnisschwierigkeiten auftreten. Durch die enorme Dicke und Informationsdichte benötigt es allerdings einiges an Disziplin, um das dicke Buch durchzuarbeiten und alle Beispiele nachzuprogrammieren.

Trotzdem sollen einige Nachteile nicht verschwiegen werden:

- Teilweise wird auf veraltete Libraries vom GCC zurückgegriffen.
- Die Quelltexte sind nicht 100% konsistent, an einer Stelle muß man auf einen Download beim Oldenbourg-Verlag zurückgreifen, weil ein Quelltext fehlt, an einer anderen werden ohne Verweis Programme, die erst später im Buch auftauchen vorausgesetzt.
- Die Schaltungen, die den Quelltexten zugrundeliegen, sind teilweise nicht enthalten.

Ansonsten ist dieses Buch, wenn man die Programmiersprache C schon beherrscht, eine wirklich gute Einführung, die ihr Geld wert ist.

„Programmiertechniken für AVR-Mikrocontroller: Darstellung und ausführliche Implementierung“, *Manfred Schwabl-Schmidt*, [6]

Hier muß im nochmal betonen, daß es sich um eine persönliche Meinung handelt, denn mein Urteil über dieses Buch kann nur vernichtend sein. Es soll sich um eine Einführung in AVR-Assembler handeln, allerdings verliert sich der Autor in Formalismen, die noch dazu sehr unübersichtlich dargestellt werden. Sollte man sich wirklich Assembler für AVR antun wollen, halte ich [5] für die deutlich bessere Einführung.

„C Programmieren von Anfang an“, *Helmut Erlenkötter*, [1]

Dieses Buch bietet eine gute und handliche Einführung in die Programmiersprache C, allerdings nicht in Bezug auf Mikrocontroler, sondern auf PC-Basis. Trotzdem ist dieses Taschenbuch seinen geringen Preis absolut wert und bietet sich als Vorgeplänkel an, wenn man [5] durcharbeiten will, C aber noch nicht beherrscht.

„AVR-Tutorial“ von mikrocontroller.net, [2]

Hier handelt es sich um ein sehr gutes Online-Tutorial für AVR-Mikrocontroller, allerdings setzt das Ganze auf Assembler auf. Falls einem das Lesen am Bildschirm nichts ausmacht, findet man hier genau das Richtige für den Einstieg.

„AVR-GCC-Tutorial“ von mikrocontroller.net, [3]

Dieses Online-Tutorial auf dem gleichen Portal kommt an sein Vorbild mit Assembler noch nicht ganz heran, wird aber immer besser. Auch hier: Falls einem das Lesen am Bildschirm nichts ausmacht und man C schon etwas beherrscht, ist das vielleicht genau das Richtige für den Einstieg. Die noch vorhandenen Lücken lassen sich dann problemlos durch das Tutorial [2] schließen.

Programmiersprache C/C++ (Online-Script), *Dr. P. Böhme*, [4]

Ist ein wohl für immer unvollständiges, online verfügbares Script für ein C-Seminar. Das Ganze ist ein gutes Online-Nachschlagewerk für C, da die vorhandenen Lücken für die AVR-Programmierung nicht ins Gewicht fallen.

Datenblatt ATmega ATmega48, ATmega88 und ATmega168 von ATMEL, [8]

Es ist zwar trotz der Dicke kein Buch sondern nur das Datenblatt, aber: Man kann es lesen und es ist gut geschrieben! Wenn die englische Sprache kein Problem ist, kann man durchaus auch dann, wenn es nicht unbedingt sein muß, mal einen Blick hineinwerfen.

Anhang A

Literaturverzeichnis

[1] *Helmut Erlenkötter*, C Programmieren von Anfang an, ISBN 9783499600746

[2] AVR-Tutorial, `http://www.mikrocontroller.net/articles/AVR-Tutorial` Online-Ressource, Abruf: 11.11.2007

[3] AVR-GCC-Tutorial `http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial` Online-Ressource, Abruf: 05.09.2008

[4] *Dr. P. Böhme*, Programmiersprache C/C++, Skript Martin-Luther-Universität Halle-Wittenberg Fachbereich Mathematik und Informatik Institut für Informatik, `http://www2.informatik.uni-halle.de/lehre/c/` Online-Ressource, Abruf: 15.08.2008

[5] *Günter Schmitt*, Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie, Oldenbourg-Verlag, 3. Auflage

[6] *Manfred Schwabl-Schmidt*, Programmiertechniken für AVR-Mikrocontroller: Darstellung und ausführliche Implementierung, Elektor-Verlag Juni 2007

[7] *ATMEL*, Datenblatt ATmega8(L), Version 08/07

[8] *ATMEL*, Datenblatt ATmega48/88/168, Version 04/07

[9] *Electronic Assembly*, Datenblatt DIPS162, Version 07/2003

[10] *Julian Ilett*, How to use intelligent L.C.D.s, Everyday Practical Elektronics, February 1997

[11] *Christian Ulrich*, `http://www.ullihome.de/` Online-Ressource, Abruf 22.07.2008

Anhang B

Quelltexte

Lauflicht 1 (C)

```
1 #include <util/delay.h>
2 #include <avr/io.h>
3
4 void warte10ms(int faktor)
5 {
6  for (int i=faktor; i>0; i--)
7  {
8   _delay_ms(10);
9  }
10 }
11
12 int main()
13 {
14   DDRD=0xFF;
15   DDRB=0x3F;
16
17   while(1) // Hauptschleife
18   {
19     for (int i=0; i<=13; i++)
20     {
21       PORTB |= (1 << i);
22       //PORTD |= (1 << (i-6)); // klappt nicht!
23       PORTD |= (1 << i) >> 6; // so klappt das
24       warte10ms(3);
25     }
26
27     warte10ms(5);
28
29     for(int i=0; i<=13; i++)
30     {
31       PORTB &= ~(1 << i);
32       PORTD &= ~(1 << i) >> 6;
```

```
33        warte10ms(3);
34      }
35      warte10ms(50);
36    } // Ende Hauptschleife
37    return 0;
38  }
```

LCD 8 Bit

Hauptdatei

03_LCD_8_Bit.c

```
1  /* Uebungsprogramm Mikrocontrollerseminar:
2   * Ansteuerung eines HD44780-kompatiblen LCDs
3   * Anschluss des LCDs wird in lcd8.c eingestellt.
4   */
5
6  #ifndef F_CPU
7    #define F_CPU 16000000UL
8  #endif
9
10 #include <avr/io.h>
11 #include <util/delay.h>
12 #include <stdio.h>
13 #include "lcd8.h"
14
15
16 inline void warte10ms(int faktor)
17 {
18  for (int i=faktor; i>0; i--)
19  {
20   _delay_ms(10);
21  }
22 }
23
24
25 int main(void)
26 {
27   // Entsprechende Pins auf Ausgang und LCD initialisieren
28   ...
29
30   // LCD loeschen
31   ...
32
33   // Zeichen ausgeben: "LCD OK"
34   ...
```

```
35    ...
36    ...
37    ...
38    ...
39    ...
40
41    warte10ms(50);
42
43    // In zweite Zeile erstes Zeichen
44    ...
45
46    // Zeichenkette schreiben
47    ...
48
49    while(1);
50
51    return 1;
52 }
```

LCD-Hauptfunktion

lcd8.c

```
1 /* Funktionen fuer die Ansteuerung von HD44780-kompatiblen
2  * LCDs im 8-Bit-Modus
3  */
4
5 #include <avr/io.h>
6 #include <util/delay.h>
7 #include "lcd8.h"
8
9 // Enable-Puls
10 inline void lcd_enablepulse()
11 {
12    ...
13    ...
14    ...
15    ...
16 }
17
18
19 // Kommandos an LCD uebergeben
20 void lcd_putcom(char kommando)
21 {
22    // Byte uebertragen
23    ...
24
```

```
25    ...
26
27    ...
28 }
29
30
31 // Zeichen an LCD ausgeben
32 void lcd_putchar(char zeichen)
33 {
34    ...
35    ...
36    ...
37 }
38
39
40 // LCD initialisieren 8 Bit
41 void lcd_init()
42 {
43   // Pins auf Ausgang
44    ...
45    ...
46
47   // Pins auf Low
48    ...
49    ...
50
51   // Initialisierung starten
52    ...
53    ...
54    ...
55    ...
56    ...
57    ...
58
59    ...
60    ...
61    ...
62    ...
63    ...
64    ...
65    ...
66
67  // Noch ein wenig warten
68   _delay_ms(5);
69 }
```