avrhal-lib Nachschlagewerk

20030717

Erzeugt von Doxygen 1.2.18

Thu Jul 17 20:10:27 2003

Inhaltsverzeichnis

1 AVR HAL Bibliothek

Die AVR HAL Bibliothek stellt eine C API für den Zugang zu standardisierten Hardwarekomponenten, wie LED, LCD, Tasten und Sensoren, bereit. Mit Hilfe der Bibliothek können allgemeingültige Anwendungen für spezifische Boards auf Basis des AVR Mikrocontrollers programmiert werden. Die Bibliothek ist aus der Entwicklung von "AVR-Ctrl Lib", einer speziellen Hardwareabstraktion des AVR-Ctrl Boards, entstanden. Die Bibliothek wurde gezielt nur für die Benutzung in Verbindung mit dem GNU C Compiler und der avr-libc entwickelt. Es existieren einige API Adaptionen zum CodeVision C Compiler.

Die aktuellste Version dieses Dokumments ist immer verfügbar unter:
http://www.li-pro.net/projects/avr

Diese Dokumentation wurde erstellt in Zusammenhang mit der Boardkonfiguration:
avrctrl-8535-8mhz

Zu beachten:
Dieses Dokument unterliegt häufigen Veränderung. Daher kann es falsche oder nicht korrekte Informationen enthalten. Solltest Du solche Stellen finden, dann schicke eine Email mit der Beschreibung und wenn möglich der Berichtigung des Fehlers an linz@li-pro.net. Ebenso sind Emails mit Anregungen und Ergänzungen zum Inhalt willkommen.

1.0.1 Unterstützte Hardware

Die folgenden Hardwarekomponenten werden bis jetzt von dieser Bibliothek unterstützt.

Board:
- AVR-Ctrl von http://www.mikrocontroller.com

AVR CPU:
- AT90S8535
- ATMega163
- Prozessortakt 1, 2, 3, 4, 5, 6, 7, und 8 MHz

Eingabeeinheiten:
- Tastaturfeld mit 5 Tasten (direkt kontaktiert)
- Dallas Semiconductors *One Wire* Bus
- One Wire Temperatursensor DS1820/DS1822

Ausgabeeinheiten:
- LED Balken mit 8 LED (direkt kontaktiert)
- LED Punkt- und Balkengrafik mit Normierung
- alphanumerisches LCD; bis zu 4x64 Zeichen

Für die folgenden Komponenten ist die Unterstützung in Zukunft geplant oder bereits in Arbeit.

Eingabeeinheiten:
- Infrarotempfänger
- Analog-Digital-Wandler

Ausgabeeinheiten:
- Puls-Weiten-Modulator

Kommunikationsschnittstellen:
- serielle Schnittstelle RS232 (UART)
- Inter-IC-Connection I2C (Bit Bang Treiber)
- STDIN, STDOUT, STDERR an RS232 bzw. LCD

sonstiges:
- Sondierung, Auswahl und Integration eines Schedules für Multiprozeßarbeit

1.0.2 Konzept und Anwendung

Die Bibliothek wird zum Übersetzungszeitpunkt für eine spezifische AVR Zielhardware (Board) konfiguriert und übersetzt. Hierfür wird die `configure` Option `--enable-board=BOARD` benutzt, wobei `BOARD` einer der folgenden Bezeichner sein kann:

- avrctrl-8535-8mhz
- avrctrl-8535-8mhz-small
- avrctrl-m163-8mhz
- avrctrl-m163-8mhz-small

Dieser Bezeichner hat den Aufbau BOARD-MCU-CLOCK-EXT und kann je Übersetzungslauf nur einmal angegeben werden. Im Ergebnis entsteht eine Bibliothek mit dem Bezeichner im Dateinamen, nämlich `libavrhal-BOARD-MCU-CLOCK-EXT.a`, und ein zugehöriger Konfigurationsheader namens `BOARD-MCU-CLOCK-EXT.h` in der Interfacedefinition der Bibliothek. Es besteht die Möglichkeit, für jede Boardkonfiguration die dazugehörige AVR HAL Bibliothek hintereinader zu erzeugen und zu installieren. Für das AVR-Ctrl Board hätte das folgendes Aussehen:

1. `./configure --enable-board=avrctrl-8535-8mhz && make install`

   ergibt: `$(PREFIX)/avr/include/avrhal/avrctrl-8535-8mhz.h`

   und `$(PREFIX)/avr/lib/libavrhal-avrctrl-8535-8mhz.a`

2. `./configure --enable-board=avrctrl-8535-8mhz-small && make install`

   ergibt: `$(PREFIX)/avr/include/avrhal/avrctrl-8535-8mhz-small.h`

   und `$(PREFIX)/avr/lib/libavrhal-avrctrl-8535-8mhz-small.a`

3. `./configure --enable-board=avrctrl-m163-8mhz & make install`
   ergibt: `$(PREFIX)/avr/include/avrhal/avrctrl-m163-8mhz.h`
   und `$(PREFIX)/avr/lib/libavrhal-avrctrl-m163-8mhz.a`
4. `./configure --enable-board=avrctrl-m163-8mhz-small && make install`
   ergibt: `$(PREFIX)/avr/include/avrhal/avrctrl-m163-8mhz-small.h`
   und `$(PREFIX)/avr/lib/libavrhal-avrctrl-m163-8mhz-small.a`

Die so entstandene Installation der Bibliothek kann nun in eigenen AVR Projekten benutzt werden. Dazu werden die jeweils notwendigen Headerdateien wie bei jeder gewöhnlichen Bibliothek in die C Quellen aufgenommen, z.B. für das LCD und die Tastatur:

```
#include <avrhal/key.h>
#include <avrhal/lcd.h>
```

Hiermit wird der Präprozessor automatisch den *Configuration Wrapper* mit einbinden (`avrhal/configure.h`). Dieser muß über ein Predefine konfiguriert werden. Nur so kann dieser die richtige Boardkonfiguration bereitstellen (z.B. für `--enable-board=avrctrl-8535-8mhz` der Inhalt von `avrhal/avrctrl-8535-8mhz.h`). Das Predefine, in Form eines CPP Makros, muß im eigenen AVR Projekt erfolgen. Im Regelfall wird diese Makrodefinition über die `gcc` Option `-D` vorgenommen. Folgende Definitionen sind möglich und stehen in direktem Zusammenhang zu obigen Board Bezeichnern:

- `gcc` Option: `-DAVRHAL_CONFIG_avrctrl_8535_8mhz`
- `gcc` Option: `-DAVRHAL_CONFIG_avrctrl_8535_8mhz_small`
- `gcc` Option: `-DAVRHAL_CONFIG_avrctrl_m163_8mhz`
- `gcc` Option: `-DAVRHAL_CONFIG_avrctrl_m163_8mhz_small`

Eine somit übersetzte C Quelle muß dann mit dem richtigen Bibliotheksarchiv verlinkt werden:

- `ld` Option: -lavrhal-avrctrl-8535-8mhz
- `ld` Option: -lavrhal-avrctrl-8535-8mhz-small
- `ld` Option: -lavrhal-avrctrl-m163-8mhz
- `ld` Option: -lavrhal-avrctrl-m163-8mhz-small

Einige Beispiel dazu können dieser Dokumentation entnommen werden.

2 avrhal-lib Modul-Verzeichnis

2.1 avrhal-lib Module

Hier folgt die Aufzählung aller Module:

3 avrhal-lib Datenstruktur-Verzeichnis

3.1 avrhal-lib Datenstrukturen

Hier folgt die Aufzählung aller Datenstrukturen mit einer Kurzbeschreibung:

4 avrhal-lib Seitenindex

4.1 avrhal-lib Zusätzliche Informationen

Hier folgt eine Liste mit zusammengehörigen Themengebieten:

5 avrhal-lib Modul-Dokumentation

5.1 CRC Berechnungen

Dallas One Wire

- unsigned char crc8_ow (unsigned char *buf, size_t buf_size)
  8 Bit CRC Summenberechnung

5.1.1 Ausführliche Beschreibung

```
#include <avrhal/crc.h>
```

Diese Headerdatei deklariert eine Sammlung von Funktionen für die Berechnung spezieller CRC Summen. Im Regelfall führen die Funktionen keinerlei Tests auf Speicherüberläufe oder NULL Zeiger Referenzierung durch.

Für die Erstellung dieses Codes wurden folgende Referenzen benutzt: [1].

Zu beachten:
[1] Dallas Semiconductor Application Note 27 ”Understanding and Using CRC”

Warnung:
Unter Umständen wird dieser Teil der Bibliothek in Zukunft in die Sandard C Bibliothek für AVR Controller, *avrlibc*, übergehen.

5.1.2 Dokumentation der Funktionen

5.1.2.1 unsigned char crc8_ow (unsigned char * *buf*, size_t *buf_size*)

8 Bit CRC Summenberechnung

Mit dieser Funktion wird die im One Wire Bus benutzte 8 Bit CRC Summe über den mit *data* und *length* referenzierten Puffer berechnet. Die Berechnung erfolgt nach den Anweisungen aus [1].

Parameter:
buf Zeiger auf den Anfang des Puffers.

buf_size Anzahl der Bytes im Puffer (Größe).

Rückgabe:
Die Funktion crc8_ow() gibt die berechnete CRC Summe zurück.

5.2 Verzögerungen

Deklarationen mit Abhängigkeit vom Systemtakt

- void delay_ms (unsigned int msec)
  Unterbricht die Ausführung für einen Intervall von Millisekunden.
- void delay_us (unsigned int usec)
  Unterbricht die Ausführung für einen Intervall von Mikrosekunden.

Deklarationen unabhängig vom Systemtakt

- void delay_9T (void)
  Unterbricht die Ausführung für einen Intervall von 9 Systemtakten.
- void delay_8T (void)
  Unterbricht die Ausführung für einen Intervall von 8 Systemtakten.
- void delay_7T (void)
  Unterbricht die Ausführung für einen Intervall von 7 Systemtakten.
- void delay_6T (void)
  Unterbricht die Ausführung für einen Intervall von 6 Systemtakten.
- void delay_5T (void)
  Unterbricht die Ausführung für einen Intervall von 5 Systemtakten.
- void delay_4T (void)
  Unterbricht die Ausführung für einen Intervall von 4 Systemtakten.
- void delay_3T (void)
  Unterbricht die Ausführung für einen Intervall von 3 Systemtakten.
- void delay_2T (void)
  Unterbricht die Ausführung für einen Intervall von 2 Systemtakten.
- void delay_1T (void)
  Unterbricht die Ausführung für einen Intervall von 1 Systemtakt.

5.2.1 Ausführliche Beschreibung

```
#include <avrhal/delay.h>
```

Diese Headerdatei deklariert eine Sammlung von Funktionen für die **ungenaue** Unterbrechung bzw. Verzögerung linear ablaufender Anweisungen. Diese Funktionen sind unabhängig von einer festen Zeitbasis, wie z.B. ein Timer. Ihre Implementation ist **immer** von der zugrunde liegenden Taktfrequenz der CPU abhängig. Der Ablauf der Funktionen, also demnach auch die Verzögerung, kann durch unvorhergesehene Ereignisse, wie die Behandlung eines Interrupt, verlängert werden.

Noch zu erledigen:
Ausbau der Unterstützung verschiedener CPU Taktfrequenzen, speziell für delay_us().

5.2.2 Dokumentation der Funktionen

5.2.2.1 void delay_ms (unsigned int *msec*)

Unterbricht die Ausführung für einen Intervall von Millisekunden.

Die Funktion delay_ms() unterbricht die Ausführung der aufrufenden Instanz für *msec* Millisekunden. Die Unterbrechung kann durch Systemaktivitäten, z.B. Interrupts, oder durch die Zeit, die zum Bearbeiten des Aufrufs verwendet wird, verlängert werden.

Der reguläre Jitter beträgt etwa 0,29 % der Gesamtverzögerung. Die Funktion delay_ms() benutzt den WDR Aufruf, um ein eventuelles Reset durch den Watchdog zu vermeiden.

Rückgabe:
Die Funktion delay_ms() besitzt keinen Rückgabewert.

5.2.2.2 void delay_us (unsigned int *usec*)

Unterbricht die Ausführung für einen Intervall von Mikrosekunden.

Die Funktion delay_us() unterbricht die Ausführung der aufrufenden Instanz für *usec* Mikrosekunden. Die Unterbrechung kann durch Systemaktivitäten, z.B. Interrupts, oder durch die Zeit, die zum Bearbeiten des Aufrufs verwendet wird, verlängert werden.

Rückgabe:
Die Funktion delay_us() besitzt keinen Rückgabewert.

5.3 Temperatursensor DS1820/DS1822

Konvertierung und Abfrage

- #define DS1820_FAMILY 0x10
  One Wire Familien Code von DS1820/DS1822.
- unsigned char ds1820_convert (unsigned char device)
  Konvertierung der Temperatur auslösen.
- signed int ds1820_temp10_C (unsigned char device)
  Temperatur auslesen.

CodeVision Konformität

- int ds1820_temperature_10 (unsigned char *addr)

5.3.1 Ausführliche Beschreibung

```
#include <avrhal/ds1820.h>
```

Diese Headerdatei deklariert einen einfachen Low-Level Zugang zu den Temperatursensoren DS1820/DS1822 von Dallas Semiconductors über den One Wire Bus. Um die hier aufgeführten Funktionen benutzen zu können, muß der One Wire Bus initialisiert und betriebsbereit sein. Es ist keine weitere Initialisierung dieses Teils der Bibliothek notwendig.

Siehe auch:
One Wire Buszugriff

Noch zu erledigen:
Integration der *ALARM* Ereignisbehandlung. Anlenung an CodeVision's unsigned char ds1820_set_alarm(unsigned char *addr, signed char temp_low, signed char temp_high).

5.3.2 Dokumentation der Funktionen

5.3.2.1 unsigned char ds1820_convert (unsigned char *device*)

Konvertierung der Temperatur auslösen.

Mit dieser Funktion wird der über *device* adressierte Temperatursensor zu einer neuen Temperaturkonvertierung aufgefordert. Die Funktion wartet mit Hilfe von ow_ready() auf das Ende dieser Konvertierung.

Die Geräteauswahl *device* antspricht dem Index in dem von ow_rom_search() gefüllten Feld mit erkannten Geräteadressen.

Parameter:

device Geräteindex der internen Geräteverwaltung.

Rückgabe:

Die Funktion ds1820_convert() gibt eine Null(0) zurück, wenn ein Fehler aufgetreten ist. Ansosnsten wird ein Wert ungleich Null(!=0) für eine erfolgreiche Konvertierung zurückgegeben.

5.3.2.2 int ds1820_temp10_C (unsigned char *device*)

Temperatur auslesen.

Mit dieser Funktion wird der über *device* adressierte Temperatursensor mit Hilfe von ds1820_convert() zu einer neuen Temperaturkonvertierung aufgefordert und das Ergebnis davon eingelesen. Die Temperatur wird in eine dezimale Grad-Celsius Zahl überführt und zurückgegeben.

Die Geräteauswahl *device* antspricht dem Index in dem von ow_rom_search() gefüllten Feld mit erkannten Geräteadressen.

Parameter:

device Geräteindex der internen Geräteverwaltung.

Rückgabe:

Die Funktion ds1820_temp10_C() gibt die aktuelle Temperatur in Grad-Celsius oder -9999 bei Auftreten eines Fehlers zurück.

5.3.2.3 int ds1820_temperature_10 (unsigned char * *addr*)

CodeVision API

Aliasfunktion für ds1820_temp10_C() wobei bei dieser Funktion direkt über die 8 Byte lange One Wire Geräteadresse auf den Temperatursensor zugegriffen wird. Die Geräteadresse wird dabei automatisch in die Gerätenummer überführt.

Parameter:

addr One Wire Geräteadresse.

Rückgabe:

Die Funktion ds1820_temperature_10() besitzt den Rückgabewert von ds1820_temp10_C().

5.4 Tastaturfeld

Scancodes

- #define KEY_SCAN_T1 _BV(7)

  Bitwert für Taste T1, erste Taste von links.

- #define KEY_SCAN_T2 _BV(6)

  Bitwert für Taste T2, zweite Taste von links.

- #define KEY_SCAN_T3 _BV(5)

  Bitwert für Taste T3, dritte Taste von links.

- #define KEY_SCAN_T4 _BV(4)

  Bitwert für Taste T4, vierte Taste von links.

- #define KEY_SCAN_T5 _BV(3)

  Bitwert für Taste T5, fünfte Taste von links.

- #define KEY_SCAN_ALL

  Bitwert für alle Tasten T1 bis T5 (Tastenmaske).

Low-Level Zugriff

- void key_init (void)

  (Re-)Initialisierung des Tastaturfeldes.

- unsigned char key_scancode (void)

  Tastatur Sancode einlesen.

5.4.1 Ausführliche Beschreibung

```
#include <avrhal/key.h>
```

Diese Headerdatei deklariert einen einfachen Low-Level Zugang zu exakt 5 Tasten an einem konfigurierbaren Port. Es werden ausschließlich nur die 5 Bits beachtet, an denen auch Taster angeschlossen sind (Bit 3..7). Die restlichen Bit 0..2 des KEY Port bleiben unberührt.

Vor der Benutzung der KEY Funktionen, muß dieser Teil der Bibliothek mit key_init() initialisiert werden. Die Bibliothek stellt für die Ermittlung gedrückter Tasten die Funktion key_scancode() bereit. Jeder Taste ist im Scancode ein Bit zugeordnet. Für die einfache Auswertung sind diese Codes Bestandteil dieser Deklaration.

Noch zu erledigen:

Überführung der Initialisierung nach Sektion `.init1`
Parametrisierung über mehrere Ports verstreuter Bits.
Unterstützung für Tastaturmatrix.
Wenn machbar, dann eine Ereigniskontrolle (call-back) einführen.

5.4.2 Makro-Dokumentation

5.4.2.1 #define KEY_SCAN_ALL

Wert:

```
(       KEY_SCAN_T1                                             \
|       KEY_SCAN_T2                                             \
|       KEY_SCAN_T3                                             \
|       KEY_SCAN_T4                                             \
|       KEY_SCAN_T5     )
```

Bitwert für alle Tasten T1 bis T5 (Tastenmaske).

5.4.3 Dokumentation der Funktionen

5.4.3.1 void key_init (void) `[inline, static]`

(Re-)Initialisierung des Tastaturfeldes.

Initialisiert den KEY Port zur Biteingabe (alle 5 Bits). Dabei bleiben die unteren Bits unbehandelt.

Rückgabe:

Die Funktion key_init() besitzt keinen Rückgabewert.

5.4.3.2 unsigned char key_scancode (void) `[inline, static]`

Tastatur Sancode einlesen.

Ließt den aktuellen Zustand der Tasten ein und liefert einen entsprechenden Scancode. Die folgende Abbildung zeigt die Tasten T1 bis T5 und die zugehörigen Bitwerte als Teil des Scancodes, wenn T2, T3 und T5 gedrückt sind.

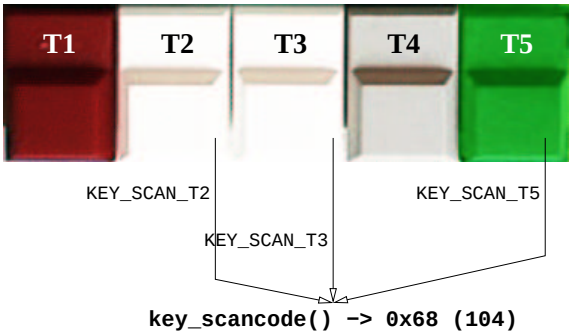


Abbildung 1: Tastenfeld mit Scancode

Es wird immer ein Byte als Scancode zurückgegeben. Dieses setzt sich wie folgt zusammen:

```
AVR-Ctrl key numbers:
 ___________________________________
|                                   |
|      LCD ...                      |
|___________________________________|
  _____ _____ _____ _____ _____
|  T1 |  T2 |  T3 |  T4 |  T5 |
|_____|_____|_____|_____|_____|
|\___/|\___/|\___/|\___/|\___/|
||___|||___|||___|||___|||___||
|/___\|/___\|/___\|/___\|/___\|

 ___ ___ ___ ___ ___ ___ ___ ___
|   |   |   |   |   |   |   |   |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | - scancode (Byte)
|___|___|___|___|___|___|___|___|
  |   |   |   |   |   |_______|
  |   |   |   |   |       |_________ unused
  |   |   |   |   |
  |   |   |   |   |___________________ KEY_SCAN_T5
  |   |   |   |   |
  |   |   |   |___|___________________ KEY_SCAN_T4
  |   |   |   |
  |   |   |___|_______________________ KEY_SCAN_T3
  |   |   |
  |   |___|___________________________ KEY_SCAN_T2
  |   |
  |___|___________________________________ KEY_SCAN_T1
```

Rückgabe:

Die Funktion key_scancode() gibt den aktuellen Scancode zurück.

5.5 Alphanumerisches LCD

Low-Level Zugriff / Brajer Vlado Konformität

- void lcd_init (unsigned char lcd_matrix)
  (Re-)Initialisirung des LCD Moduls.
- void lcd_cls (void)
  LCD Anzeige löschen.
- void lcd_home (void)
  Cursor auf erste beschreibbare Position zurück.
- void lcd_goto (unsigned char addr)
  Cursor auf eine beliebige Position setzen.
- void lcd_ctrl (unsigned char lcd_control)
  Kontrolle der LCD Arbeitsweise.
- void lcd_cleol (void)
  LCD Zeile ab Cursor löschen.
- void lcd_putch (unsigned char data)
  Zeichenausgabe ab Cursor.
- void lcd_print10 (unsigned long x)
  Ausgabe einer 10-stelligen Dezimalzahl.
- void lcd_print2 (unsigned char x)
  Ausgabe einer 2-stelligen Dezimalzahl.
- void lcd_print3 (unsigned int x)
  Ausgabe einer 3-stelligen Dezimalzahl.
- void lcd_print5 (unsigned int x)
  Ausgabe einer 5-stelligen Dezimalzahl.
- void lcd_printbin (unsigned char x)
  Ausgabe einer 8-stelligen Binärzahl.
- void lcd_printf (const char *format,...)
  Primitiver Nachbau der Std-Lib-C printf() Funktion.
- void lcd_printhex (unsigned char x)
  Ausgabe einer 2-stelligen Hexadezimalzahl.

- void lcd_putstr (unsigned char *data)
  Zeichenkettenausgabe ab Cursor.
- void lcd_control (unsigned char d, unsigned char c, unsigned char b)
  Vereinfachter Aufruf von lcd_ctrl().

CodeVision Konformität

- void lcd_gotoxy (unsigned char x, unsigned char y)
  Cursor auf eine beliebeige X-Y-Position setzen.
- void lcd_write_byte (unsigned char addr, unsigned char data)
  Datenbyte ausgeben.
- void _lcd_write_data (unsigned char data)
  Kommandobyte ausgeben.
- void lcd_clear (void)
  LCD Anzeige löschen.
- void lcd_putchar (char c)
  Zeichenausgabe ab Cursor.
- void lcd_puts (char *str)
  Zeichenkettenausgabe ab Cursor.
- void _lcd_ready (void)
  LCD Bereitschaftstest.

5.5.1 Ausführliche Beschreibung

```
#include <avrhal/led.h>
```

Diese Headerdatei deklariert zunächst einen Low-Level Zugang für ein alphanumerisches LCD. Es werden dabei 1 bis 4 Zeilen Displays mit bis zu 64 Zeichen je Zeile unterstützt. Diese LCD besitzen in aller Regel einen eigenen Controller, der auf dem Typ HD44780 von Hitachi basiert. Hier eine Liste von Vergleichstypen. Displays mit einem dieser Chips sollten mit dem hier bereitgestellten Code zusammenarbeiten:

- Hitachi **HD44780** / **HD44780S** / **HD44780U** / **LM018L**
- Samsung **KS0066U**
- Seiko-Epson **SED1278**

Es wird immer von einer Nibble Kommunikation (4 Bit) mit dem LCD ausgegangen. Der LCD Port ist auf `PORTC` festgelegt. Die Steuersignale RS, RD und EN sind am selben LCD Port zum Übersetzungszeitpunkt frei definierbar. Das eine unbenutzte Bit wird zur Laufzeit auch als solches berücksichtigt. Damit ergibt sich eine konfigurierte Bitverteilung von:

- `PORTC` Bit 7..4: Daten Nibble (DB7..4)
- `PORTC` Bit 3: unbenutzt
- `PORTC` Bit 2: Signal EN
- `PORTC` Bit 1: Signal RD
- `PORTC` Bit 0: Signal RS

Vor der Benutzung der LCD Funktionen, muß dieser Teil der Bibliothek mit lcd_init() initialisiert werden. Hierbei wird der Typ des LCD Moduls verschlüsselt übergeben. Für die weitere Benutzung des LCD existieren abstrahierte aktionsbezogene Funktionen zum Positionieren des Cursors und Ausgeben von Zahlen und Zeichenketten. Nach Datenbuch des Controllers werden folgende Funktionen des LCD unterstützt (LCD instruction set):

- **Clear Display**
- **Return Home**
- **Display ON/OFF Control**
- **Set CGRAM Address**
- **Set DDRAM Address**
- **Read Busy Flag & Address** (wenn gewünscht)

Der letzte Punkt wird nur unterstützt, wenn zum Übersetzungszeitpunkt das Polling aktiviert wurde. Bei deaktiviertem Polling wird mit einem Timeout Verfahren gearbeitet. Dieses sollte mit allen LCD Typen funktionieren. Es kann aber vorkommen, daß manche Controller den gegebenen Zeitrahmen nicht einhalten. Das Timeout Verfahren sollte dann zum Einsatz kommen, wenn man kleineren Code benötigt und weniger auf Ablaufzeit achtet.

Um zwischen Polling und Timeout Verfahren zu entscheiden, muß man ... . Das **Polling** Verfahren wurde aktiviert.

Für die Erstellung dieses Codes wurden folgende Referenzen benutzt:

- http://bray.velenje.cx/avr/lcd/lcd.html
- http://www.doc.ic.ac.uk/~ih/doc/lcd/index.html
- http://www.repairfaq.org/filipg/LINK/F_LCD_HD44780.html
- http://home.iae.nl/users/pouweha/lcd/lcd.shtml

Noch zu erledigen:

Überführung von Teilen der Initialisierung nach Sektion `.init1`
Unterstützung eigener frei definierter Grafikzeichen.
Balken- und Punktgrafik ähnlich den LED Funktionen.
Wenn möglich mehr Hardwarebeschleunigung.

5.5.2 Dokumentation der Funktionen

5.5.2.1 void _lcd_ready (void) [inline, static]

LCD Bereitschaftstest.

CodeVision API

Wartet, bis das LCD zur Kommunikation bereit ist. Diese Funktion muß immer vor dem Schreiben von Daten an das LCD mit _lcd_write_data() aufgerufen werden.

Rückgabe:

Die Funktion _lcd_ready() besitzt keinen Rückgabewert.

5.5.2.2 void _lcd_write_data (unsigned char *data*) [inline, static]

Kommandobyte ausgeben.

CodeVision API

Schreibt das Byte *data* in das LCD Anweisungsregister (instruction register). Diese Funktion kann zum Modifizieren der LCD Konfiguration benutzt werden.

Beispiel:

```
/* enables the displaying of the cursor */
_lcd_ready();
_lcd_write_data(0xe);
```

Parameter:

data ist das auszugebende Zeichen.

Rückgabe:

Die Funktion _lcd_write_data() besitzt keinen Rückgabewert.

5.5.2.3 void lcd_clear (void) [inline, static]

LCD Anzeige löschen.

CodeVision API

Aliasfunktion für lcd_cls(). Löscht die Anzeige des LCD und setzt den Cursor auf die linke obere Ecke.

Rückgabe:

Die Funktion lcd_clear() besitzt keinen Rückgabewert.

5.5.2.4 void lcd_cleol (void)

LCD Zeile ab Cursor löschen.

Mit dieser Funktion wird der Inhalt der aktuellen Zeile ab aktueller Cursorposition bis zum Ende gelöscht (mit Leerzeichen beschrieben).

Rückgabe:
Die Funktion lcd_cleol() besitzt keinen Rückgabewert.

5.5.2.5 void lcd_cls (void)

LCD Anzeige löschen.

Mit dieser Funktion wird der aktuell dargestellte Inhalt des Datenspeichers im LCD gelöscht (mit Leerzeichen beschrieben).

LCD instruction set:
Es kommt die LCD Funktion **Clear Display** zur Anwendung:

```
 RS   RD   DB7  DB6  DB5  DB4  DB3  DB2  DB1  DB0
[ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 1 ]
```

Rückgabe:
Die Funktion lcd_cls() besitzt keinen Rückgabewert.

5.5.2.6 void lcd_control (unsigned char *d*, unsigned char *c*, unsigned char *b*) [inline, static]

Vereinfachter Aufruf von lcd_ctrl().

Warnung:
Diese Funktion existiert aus Gründen der Abwärtskompatibilität und wird in einer der nächsten Veröffentlichungen der Bibliothek stillschweigend entfallen.

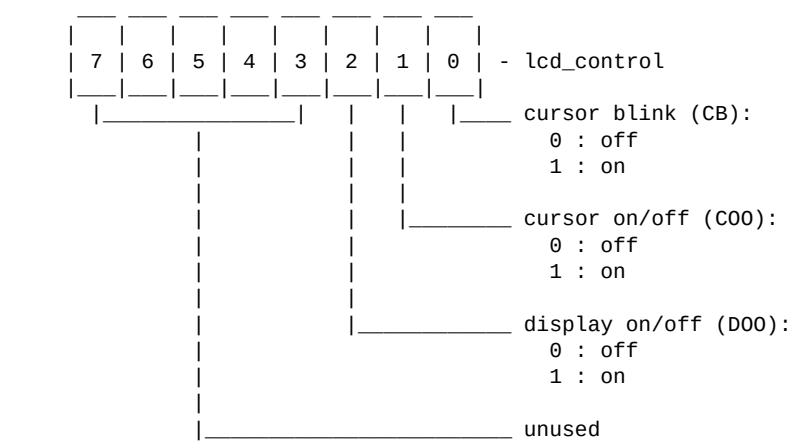
Parameter:
- ***d*** gibt mit Eins(1) an, ob das Display aktiv ist.
- ***c*** gibt mit Eins(1) an, ob der Cursor aktiv ist.
- ***b*** gibt mit Eins(1) an, ob der Cursor blinkt.

Rückgabe:
Die Funktion lcd_control() besitzt keinen Rückgabewert.

5.5.2.7 void lcd_ctrl (unsigned char *lcd_control*)

Kontrolle der LCD Arbeitsweise.

Mit dieser Funktion wird die Arbeitsweise des LCD verändert. Hierzu wird das übergebene Kontrollbyte *lcd_control* wie folgt interpretiert.



LCD instruction set:

Es kommt die LCD Funktion **Display ON/OFF Control** zur Anwendung:

```
 RS   RD   DB7  DB6  DB5  DB4  DB3  DB2  DB1  DB0
[ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 1 ][ D ][ C ][ B ]

   D equ. DOO
   C equ. COO
   B equ. CB
```

Parameter:

lcd_control ist das Kontrollbyte mit der neuen LCD Arbeitsweise.

Rückgabe:

Die Funktion lcd_ctrl() besitzt keinen Rückgabewert.

Noch zu erledigen:

Es fehlen noch die Kontrolle über die Bewegungsrichtung des Cursors und das Verschieben des Datenspeichers im LCD. Dabei soll die LCD Funktion **Entry Mode Set** verwendet werden.

5.5.2.8 void lcd_goto (unsigned char *addr*)

Cursor auf eine beliebige Position setzen.

Mit dieser Funktion wird der Cursor von der aktuellen Position auf eine mit *addr* übergebene neue gesetzt. Diese Position entspricht exakt einer Adresse im Datenspeicher

des LCD und sollte den Wert `0x7f` nicht überschreiten. Alle weieren Zeichenausgaben am LCD erfolgen ab dieser neuen Position.

LCD instruction set:

Es kommt die LCD Funktion **Set DDRAM Address** zur Anwendung:

```
 RS   RD   DB7  DB6  DB5  DB4  DB3  DB2  DB1  DB0
[ 0 ][ 0 ][ 1 ][AC6][AC5][AC4][AC3][AC2][AC1][AC0]

   AC[6..0] equ. (addr & 0x7f)
```

Parameter:

addr gibt die neue Adresse im Datenspeicher des LCD an.

Rückgabe:

Die Funktion lcd_goto() besitzt keinen Rückgabewert.

5.5.2.9 void lcd_gotoxy (unsigned char *x*, unsigned char *y*)

Cursor auf eine beliebeige X-Y-Position setzen.

CodeVision API

Mit dieser Funktion wird der Cursor von der aktuellen Position auf eine mit *x* und *y* neu definierte gesetzt. Diese Position setzt sich aus einer X-Koordinate und einer Y-Koordinate zusammen, wobei X das Zeichen in einer Zeile und Y die Zeile selbst ist. Die Zählweise beginnt bei beiden Koordinaten mit 0 ab der linken oberen Ecke. Es gilg also immer: lcd_gotoxy(0,0); entspricht lcd_goto(0); entspricht lcd_home();

Diese Funktion rechnet den kartesischen Koordinatenwert in Abhängigkeit des mit lcd_init() konfigurierten LCD Types in eine absolute Adresse im Datenspeicher des LCD um und benutzt dann die Funktion lcd_goto(), um den Cursor neu zu positionieren. Bei dieser Berechnung erfolgt kein Test auf gültigen Wertebereich!

Parameter:

x gibt die neue X-Koordinate im Anzeigebereich des LCD an.

y gibt die neue Y-Koordinate im Anzeigebereich des LCD an.

Rückgabe:

Die Funktion lcd_gotoxy() besitzt keinen Rückgabewert.

5.5.2.10 void lcd_home (void)

Cursor auf erste beschreibbare Position zurück.

Mit dieser Funktion wird der Cursor von der aktuellen Position auf die aller erste beschreibbare zurückgesetzt. Diese Position befindet sich in der linken oberen Ecke im Anzeigebereich des LCD und entspricht der Adresse `0x00` im Datenspeicher des LCD. Alle weieren Zeichenausgaben am LCD erfolgen ab dieser neuen Position.

LCD instruction set:

Es kommt die LCD Funktion **Return Home** zur Anwendung:

```
  RS   RD   DB7  DB6  DB5  DB4  DB3  DB2  DB1  DB0
[ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 0 ][ 1 ][ 0 ]
```

Rückgabe:

Die Funktion lcd_home() besitzt keinen Rückgabewert.

5.5.2.11 void lcd_init (unsigned char *lcd_matrix*)

(Re-)Initialisirung des LCD Moduls.

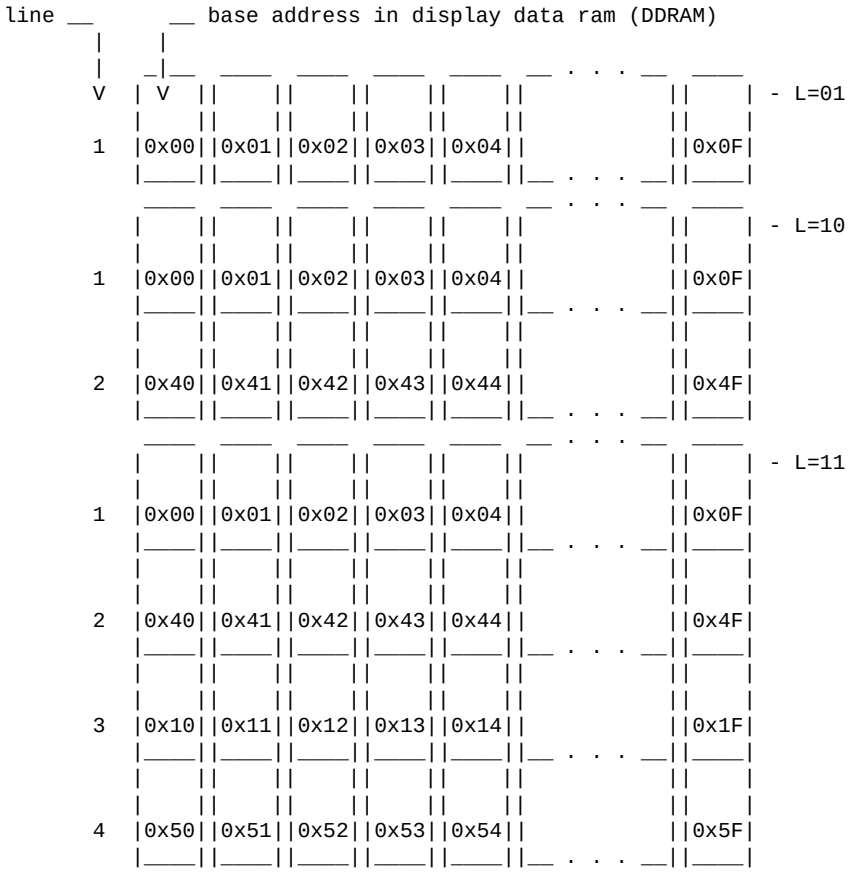
Mit Aufruf dieser Funktion wird das am LCD Port `PORTC` angeschlossene LCD Modul neu (re-)initialisiert. Dabei wird die 4 Bit Datenübertrag zum Modul fest eingestellt. Der Cursor wird inkremental eingestellt.

Neben den Einstellungen des Ports selbst und der Konfiguration des LCD Moduls werden wichtige interne Parameter gesetzt, um die Koordination der Datenandressen in Abhängigkeit des angeschlossenen LCD Moduls zu realisieren. Hierzu wird der übergebene Typenschlüssel des LCD Moduls *lcd_matrix* benutzt. Die Bedeutung der Bits im Typenschlüssel zeigt die folgende Abbildung.

```
 ___ ___ ___ ___ ___ ___ ___ ___
|   |   |   |   |   |   |   |   |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | - lcd_matrix
|___|___|___|___|___|___|___|___|
  |___|   |___________________|
    |               |_____________ chars per line (CPL):
    |                              0x00 : n/d [1]
    |                                 . . .
    |                              0x10 : 16 characters
    |                                 . . .
    |                              0x14 : 20 characters
    |                                 . . .
    |                              0x3f : 63 characters [1]
    |
    |_____________________________ number of lines (L):
                                      00 : like 11 [2]
                                      01 : 1 line
[1] see ToDo list                     10 : 2 lines
[2] compatibility to CodeVision       11 : 4 lines
```

Der Typenschlüssel wird intern dazu benutzt, die Basisadresse (Anfang) einer jeden Zeile im Daten RAM des LCD zu berechnen. Somit kann eine X/Y Positionierung in

die reale Adresse umgerechnet werden. Auch andere positionsabhängige Operationen sind erst damit möglich. Die folgende Abbildung zeigt den Vergleich von drei verschiedenen 16 Zeichen LCD mit 1, 2 und 4 Zeilen.



CodeVision API

Da bei CodeVision zwei Headerdateien für verschiedene LCD Typen existieren, ist die hier deklarierte API nicht voll kompatibel. Bei einer Codeüberführung von CodeVision zu dieser Bibliothek muß lcd_init() gegebenenfalls angepaßt werden. Vorrangig wird immer ein 4 Zeilen LCD angenommen.

Parameter:

lcd_matrix definiert den Typ des LCD Moduls. Siehe oben.

Rückgabe:

Die Funktion lcd_init() besitzt keinen Rückgabewert.

Noch zu erledigen:

Beseitigung der unklaren Nummerierung für die Anzahl Zeichen je Zeile in *lcd_matrix*, oder anders: Was ist bei Anzahl gleich Null(0) ?
Differenzierung in real existierende LCD Typen und somit wegfall der Berechnung von Anfangsadressen im Datenspeicher. Die Adressen werden zum Übersetzungszeitpunkt ermittelt und fest einprogrammiert.

5.5.2.12 void lcd_print10 (unsigned long *x*)

Ausgabe einer 10-stelligen Dezimalzahl.

Mit dieser Funktion wird die mit *x* übergebene 32 Bit positive Ganzzahl unter Zuhilfenahme von lcd_putch() ab der aktuellen Anzeigeposition am LCD mit exakt 10 Stellen als Dezimalzahl auf dem LCD ausgegeben. Führende Nullen werden immer mit ausgegeben! Es erfolgt **keine** Kontrolle des Wertebereiches!

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_print10() besitzt keinen Rückgabewert.

5.5.2.13 void lcd_print2 (unsigned char *x*)

Ausgabe einer 2-stelligen Dezimalzahl.

Mit dieser Funktion wird die mit *x* übergebene 8 Bit positive Ganzzahl unter Zuhilfenahme von lcd_putch() ab der aktuellen Anzeigeposition am LCD mit exakt 2 Stellen als Dezimalzahl auf dem LCD ausgegeben. Führende Nullen werden immer mit ausgegeben! Es erfolgt eine Kontrolle des Wertebereiches, wobei sich *x* zwischen 0 und 99 bewegen darf.

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_print2() besitzt keinen Rückgabewert.

5.5.2.14 void lcd_print3 (unsigned int *x*)

Ausgabe einer 3-stelligen Dezimalzahl.

Mit dieser Funktion wird die mit *x* übergebene 16 Bit positive Ganzzahl unter Zuhilfenahme von lcd_putch() ab der aktuellen Anzeigeposition am LCD mit exakt 3 Stellen als Dezimalzahl auf dem LCD ausgegeben. Führende Nullen werden immer mit ausgegeben! Es erfolgt eine Kontrolle des Wertebereiches, wobei sich *x* zwischen 0 und 999 bewegen darf.

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_print3() besitzt keinen Rückgabewert.

5.5.2.15 void lcd_print5 (unsigned int *x*)

Ausgabe einer 5-stelligen Dezimalzahl.

Mit dieser Funktion wird die mit *x* übergebene 16 Bit positive Ganzzahl unter Zuhilfenahme von lcd_putch() ab der aktuellen Anzeigeposition am LCD mit exakt 5 Stellen als Dezimalzahl auf dem LCD ausgegeben. Führende Nullen werden immer mit ausgegeben! Es erfolgt **keine** Kontrolle des Wertebereiches!

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_print5() besitzt keinen Rückgabewert.

5.5.2.16 void lcd_printbin (unsigned char *x*)

Ausgabe einer 8-stelligen Binärzahl.

Mit dieser Funktion wird die mit *x* übergebene 8 Bit positive Ganzzahl unter Zuhilfenahme von lcd_putch() ab der aktuellen Anzeigeposition am LCD mit exakt 8 Stellen als Binärzahl auf dem LCD ausgegeben. Dabei wird mit dem MSB begonnen.

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_printbin() besitzt keinen Rückgabewert.

5.5.2.17 void lcd_printf (const char * *format*, ...)

Primitiver Nachbau der Std-Lib-C printf() Funktion.

Mit dieser sehr einfach gehaltenen Nachbildung der aus der Standard C Bibliothek bekannten *printf()* Funktion kann man ASCII Zeichenketten mit einfachen Formatanweisungen auf dem LCD ab der aktuellen Anzeigeposition am LCD ausgeben. Hierbei wird ausschließlich nur die Funktion *lcd_putch()* benutzt. Bekannt sind folgende Formatierer ohne Formaterweiterungen wie führende Nullen oder Stellenbegrenzer:

- `c` : gibt ein ASCII Zeichen aus
- `u` und `d` : gibt eine Zahl vom Typ `int` als 5-stellige Dezimalzahl mit führenden Nullen aus

- `x` : gibt eine Zahl vom Typ `int` als 2-stellige Hexadezimalzahl mit führenden Nullen aus

Warnung:
Diese Funktion wird in späteren Versionen der Bibliothek eventuell durch den generischen *printf()* Mechanismus der Standard C Bibliothek für AVR Controller, *avrlibc*, stillschweigend ersetzt.

Parameter:
format ist der Formatstring.

Rückgabe:
Die Funktion lcd_printf() besitzt keinen Rückgabewert.

5.5.2.18 void lcd_printhex (unsigned char *x*)

Ausgabe einer 2-stelligen Hexadezimalzahl.

Mit dieser Funktion wird die mit *x* übergebene 8 Bit positive Ganzzahl unter Zuhilfenahme von lcd_putch() ab der aktuellen Anzeigeposition am LCD mit exakt 2 Stellen als Hexadezimalzahl auf dem LCD ausgegeben. Führende Nullen werden immer mit ausgegeben!

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_printhex() besitzt keinen Rückgabewert.

5.5.2.19 void lcd_putch (unsigned char *data*)

Zeichenausgabe ab Cursor.

Mir dieser Funktion wird das mit *data* übergebene ASCII Zeichen auf dem LCD direkt und ohne Filterung an der aktuellen Anzeigeposition am LCD ausgegeben, also in den Daten RAM des LCD geschrieben. Dabei wird die interne Cursorsteuerung entsprechend korrigiert. Der Cursor wird um ein Zeichen nach rechts verschoben. Am Zeilenende erfolgt der automatische Umbruch. Am Displayende erfolgt der automatische Sprung zur ersten beschreibbaren Position wie unter Verwendung von *lcd_home()*.

Parameter:
data ist das auszugebende Zeichen.

Rückgabe:
Die Funktion lcd_putch() besitzt keinen Rückgabewert.

5.5.2.20 void lcd_putchar (char *c*) [inline, static]

Zeichenausgabe ab Cursor.

CodeVision API
Aliasfunktion für lcd_putch(). Zeigt das Zeichen *c* an der aktuellen Anzeigeposition am LCD an.

Parameter:
c ist das auszugebende Zeichen.

Rückgabe:
Die Funktion lcd_putchar() besitzt keinen Rückgabewert.

5.5.2.21 void lcd_puts (char * *str*) [inline, static]

Zeichenkettenausgabe ab Cursor.

CodeVision API
Aliasfunktion für lcd_putstr(). Zeigt die Zeichenkette, auf die *str* zeigt, ab der aktuellen Anzeigeposition am LCD an.

Parameter:
str zeigt auf die auszugebende Zeichenkette.

Rückgabe:
Die Funktion lcd_puts() besitzt keinen Rückgabewert.

5.5.2.22 void lcd_putstr (unsigned char * *data*)

Zeichenkettenausgabe ab Cursor.

Mir dieser Funktion wird die über *data* referenzierte ASCII Zeichenkette unter Zuhilfenahme von lcd_putch() auf dem LCD direkt und ohne Filterung an der aktuellen Anzeigeposition am LCD ausgegeben.

Parameter:
data zeigt auf die auszugebende Zeichenkette.

Rückgabe:
Die Funktion lcd_putstr() besitzt keinen Rückgabewert.

5.5.2.23 void lcd_write_byte (unsigned char *addr*, unsigned char *data*)

Datenbyte ausgeben.

CodeVision API

Schreibt das Byte *data* an die Adresse *addr* im LCD Zeichengenerator oder Datenspeicher.

Beispiel:

```
#include <stddef.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <inttypes.h>

#include <avrhal/delay.h>
#include <avrhal/lcd.h>

#define b0000000        0x00
#define b0001111        0x0f
#define b0000011        0x03
#define b0000101        0x05
#define b0001001        0x09
#define b0010000        0x10
#define b0100000        0x20
#define b1000000        0x40

/* table for the LCD user defined character:
 * arrow that points to the top right corner */
unsigned char arrow[8] = {
        b0000000,
        b0001111,
        b0000011,
        b0000101,
        b0001001,
        b0010000,
        b0100000,
        b1000000
};

/* function used to define user characters */
void define_char(unsigned char *pc, unsigned char char_code)
{
        unsigned char i, a;

        a = ( char_code << 3 ) | 0x40;
        for (i = 0; i < 8; i++) lcd_write_byte(a++, *pc++);
}


void main(void)
{
        /* initialize the LCD for 2 lines and 16 colums */
        lcd_init(16);
```

```
        /* define first user character with arrow */
        define_char(arrow, 0);

        /* switch to writing in Display RAM */
        lcd_gotoxy(0, 0);
        lcd_puts("User char 0: ");

        /* display first user character */
        lcd_putchar(0);

        while (1); /* loop for ever */
}
```

Parameter:

addr gibt die Adresse im Datenspeicher des LCD an.

data ist das auszugebende Zeichen.

Rückgabe:

Die Funktion lcd_write_byte() besitzt keinen Rückgabewert.

5.6 LED Balken

Balkengrafik

- #define LED_BAR (_BV(0))

  Bytewert zum Aktivieren der LED Balkengrafik.

- #define LED_POINT (~(_BV(0)))

  Bytewert zum Aktivieren der LED Punktgrafik.

- #define LED_MARGIN(M) (((M) & 0x7) << 1)

  Bytewert zum Festlegen des Null- bzw. Bezubgspunktes für die LED Grafik.

- void led_graph_control (int led_maxlevel, signed char led_options)

  Verändert die interne Arbeitsweise der LED Grafik.

- void led_graph_put (int bitset_level)

  Gibt einen Wert als LED Grafik aus.

Low-Level Zugriff

- void led_init (void)

  LED Port initialisieren.

- void led_cls (void)

LED Balken löschen.

- void led_put (unsigned char bitset)
  LED Balken beschreiben.
- unsigned char led_get (void)
  LED Balken zurücklesen.
- void led_clear (void)
  LED Balken löschen.

5.6.1 Ausführliche Beschreibung

```
#include <avrhal/led.h>
```

Diese Headerdatei deklariert zunächst einen Low-Level Zugang für einen LED Balken. Dabei werden immer exakt 8 Bits (LEDs) des LED Ports benutzt. Der LED Port ist auf `PORTB` festgelegt.

Darüber hinaus existiert eine zur Laufzeit parametrisierbare LED Grafik mit Normierung. Hierbei kann man den Nullpunkt zwischen der untersten und der obersten LED frei definieren und wenn nötig nachträglich verschieben. Somit ist die Anzeige von positiven und negativen Werten möglich. Diese können dann je nach Einstellung als Punkt (eine LED) oder als Balken in Bezug zum Nullpunkt dargestellt werden. Mehr dazu bei der Beschreibung der Funktion led_graph_control().

Die Arbeitsweise der LED Grafik wird durch zwei interne Parameter bestimmt, welche mit led_graph_control() verändert werden können. Alle Ausgaben müssen dann mit led_graph_put() erfolgen.

Vor der Benutzung der LED Funktionen, muß dieser Teil der Bibliothek mit led_init() initialisiert werden.

Noch zu erledigen:
Überführung der Initialisierung nach Sektion `.init1`
Parametrisierung über mehrere Ports verstreuter Bits.

5.6.2 Makro-Dokumentation

5.6.2.1 #define LED_MARGIN(M) (((M) & 0x7) << 1)

Bytewert zum Festlegen des Null- bzw. Bezubgspunktes für die LED Grafik.

Parameter:
M ist dabei das (LED-)Bit für den Nullpunkt.

5.6.3 Dokumentation der Funktionen

5.6.3.1 void led_clear (void) [inline, static]

LED Balken löschen.

Macht das Gleiche wie led_cls(). Setzt alle Bits auf den inaktiven Zustand (OFF(0) value).

Rückgabe:
Die Funktion led_clear() besitzt keinen Rückgabewert.

5.6.3.2 void led_cls (void) [inline, static]

LED Balken löschen.

Setzt alle Bits auf den inaktiven Zustand (OFF(0) value).

Rückgabe:
Die Funktion led_cls() besitzt keinen Rückgabewert.

5.6.3.3 unsigned char led_get (void) [inline, static]

LED Balken zurücklesen.

Gibt den Wert der aktuell gesetzten LED Ausgabebits zurück.

Rückgabe:
Mit der Funktion led_get() wird das aktuelle Bitmuster vom LED Port zurückgelesen.

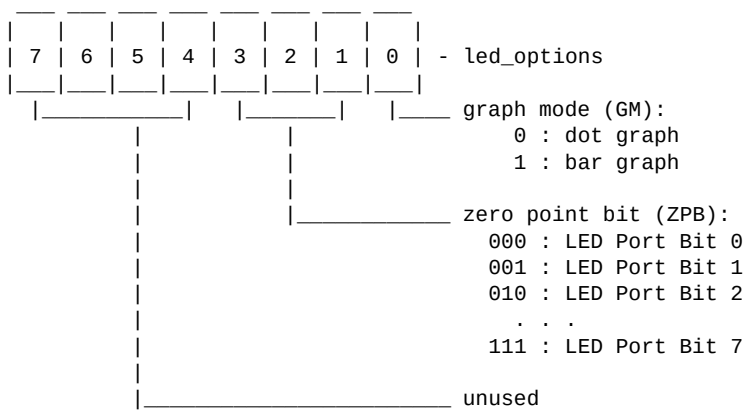
5.6.3.4 void led_graph_control (int *led_maxlevel*, signed char *led_options*)

Verändert die interne Arbeitsweise der LED Grafik.

Als Basisinitialisierung der LED Grafik ist die unterste LED als Nullpunkt eingestellt (Bit 0) und es gibt keine Normierung. Die Funkltion led_graph_put() verhält sich somit wie led_put().

Mit der Funktion led_graph_control() kann diese Arbeitsweise verändert werden. Hierzu wird der maximal darstellbarer Realwert *led_maxlevel* für die Normierung und das Optionsfeld *led_options* übergeben. Durch das Optionsfeld wird zwischen Punkt- oder Balkengrafik entschieden (GM) und es wird das Bit für den Bezugs- bzw. Nullpunkt

der Grafik festgelegt (ZPB). Die Bedeutung der Bits im Optionsfeld zeigt die folgende Abbildung.



Die folgende Abbildung zeigt ein kurzes Beispiel. Dabei wird die Balkengrafik (GM=1) der zugehörigen Punktgrafik (GM=0) gegenüber gestellt. Der Nullpunkt ist mit ZBP=010 auf das LED Port Bit 2 festgelegt. Der mit der Funktion led_graph_put() übergebene Anzeigewert `bitset_level` wird mit dem Wert `led_maxlevel` normiert an LED Port Bit 6 ausgegeben. Die LED Port Bits 0 bis 1 sind der negative Anzeigebereich.

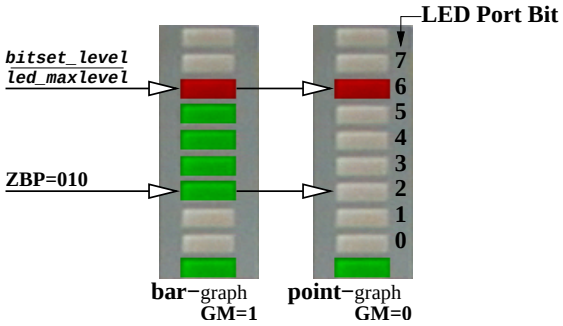


Abbildung 2: LED Grafik Beispiel

Parameter:

led_maxlevel ist der maximal darstellbarer Realwert und wird für die Normierung benutzt. Der Wert muß ungleich Null und größer 8 oder kleiner -8 sein. Erfüllt der Wert diese Regel nicht, wird die Normierung deaktiviert. Der Wert kann auch negativ sein. Dann sollte der Bezugspunkt (Nullpunkt) auf einen Wert größer Null gesetzt werden.

led_options definiert die Arbeitsweise der LED Grafik. Siehe oben.

Rückgabe:

Die Funktion led_graph_control() besitzt keinen Rückgabewert.

5.6.3.5 void led_graph_put (int *bitset_level*)

Gibt einen Wert als LED Grafik aus.

Je nach Parametrisierung durch led_graph_control() wird mit dieser Funktion der übergebene Wert zunächst normiert und dann als Grafik ausgegeben.

Zu beachten:
Ist keine Normierung eingestellt, so wird der angegebene Wert mit led_put() direkt als Bitmuster ausgegeben!

Parameter:
bitset_level ist eine positive oder negative ganze Zahl, die je nach Konfiguration der LED Garfik angezeigt wird.

Rückgabe:
Die Funktion led_graph_put() besitzt keinen Rückgabewert.

5.6.3.6 void led_init (void) [inline, static]

LED Port initialisieren.

Initialisiert den LED Port zur Byteausgabe (alle 8 Bits).

Rückgabe:
Die Funktion led_init() besitzt keinen Rückgabewert.

5.6.3.7 void led_put (unsigned char *bitset*) [inline, static]

LED Balken beschreiben.

Setzt alle Bits auf den angegebenen Wert bitset.

Parameter:
bitset ist vom Typ `unsigned char` und wird als auszugebendes Bitmuster interpretiert.

Rückgabe:
Die Funktion led_put() besitzt keinen Rückgabewert.

5.7 One Wire Buszugriff

Module

- Temperatursensor DS1820/DS1822

Datenstrukturen

- struct ow_dev_flags_s
  One Wire Geräteflags.
- struct ow_device_s
  One Wire Geräteeintrag.
- struct ow_rom_code_s
  One Wire ROM Code.

Low-Level Zugriff und Geräteverwaltung

- typedef ow_rom_code_s ow_rom_code_st
  Typvereinbarung für einen One Wire ROM Code.
- typedef ow_device_s ow_device_st
  Typvereinbarung für einen One Wire Geräteeintrag.
- unsigned char ow_init (void)
  (Re-)Initialisirung des One Wire Moduls.
- unsigned char ow_rom_search (ow_device_st *ow_dev)
  One Wire ROM Code Ermittlung.
- unsigned char ow_byte_read (void)
  Lesen eines Bytes vom One Wire Bus.
- unsigned char ow_byte_write (unsigned char byte)
  Schreiben eines Bytes zum One Wire Bus.

Spezialfunktionen

- unsigned char ow_ready (void)
  Primitiver Bereitschaftstest.
- unsigned char ow_buf_read (unsigned char *buf, size_t buf_size, unsigned char crc_included)
  Bytes vom One Wire Bus in einen Puffer lesen.
- unsigned char ow_buf_write (unsigned char *buf, size_t buf_size, unsigned char crc_included)
  Bytes aus einem Puffer zum One Wire Bus schreiben.

- unsigned char ow_function (unsigned char code, unsigned char device)

  Bytes aus einem Puffer zum One Wire Bus schreiben.

CodeVision Konformität

- unsigned char w1_init (void)
- unsigned char w1_read (void)
- unsigned char w1_write (unsigned char data)
- unsigned char w1_crc8 (void *p, unsigned char n)

5.7.1 Ausführliche Beschreibung

```
#include <avrhal/ow.h>
```

Diese Headerdatei deklariert einen einfachen Low-Level Zugang zum Eindrahtkommunikationssystem *One Wire* von Dallas. Mit dessen Hilfe können daran angeschlossene Sensoren, Speicher oder sonstige I/O-Bausteine angesprochen werden. Der One Wire Bus ist an `PORTD` Bit 4 angeschlossen. Zur Zeit sind folgende One Wire Geräte über ein eigenes API ansprechbar:

- Temperatursensor DS1820/DS1822

Dieser Teil der Bibliothek muß zunächst mit ow_init() initialisiert werden, um in der internen Geräteverwaltung alle bekannten Geräte zu vergessen. Der Initialisierungsdurchlauf kann also mehrfach erfolgen. Nach einer (Re-)Initialiserung müssen mit ow_rom_search() alle am One Wire Bus befindlichen Geräte gesucht und deren *ROM Code* in eine Liste hinterlegt werden. Hierzu muß der Benutzer der Bibliothek ausreichend Speicherplatz als Feld vom Typ `ow_device_st` bereitstellen, da alle Funktionen dieser One Wire Implementierung keinerlei Tests auf Speicherüberläufe durchfüren. Das Feld muß also für jedes physisch am Bus befindliche One Wire Gerät ein Feldeintrag vorhalten, in dem dann, nach erfolgreichem Suchen, der zugehörige ROM Code abgelegt ist.

Für die Erstellung dieses Codes wurden folgende Referenzen benutzt: [1], [2], [3],

Zu beachten:

[1] Dallas Semiconductor Application Note 159 ”Ultra-Reliable 1-Wire Communications”

[2] Dallas Semiconductor Application Note 187 ”1-Wire Search Algorithm”

[3] Dallas Semiconductor Application Note 126 ”1-Wire Communications Through Software”

Noch zu erledigen:

Überführung von Teilen der Initialisierung nach Sektion `.init1`

Integration einer One Wire *ALARM* Ereignisbehandlung.
Code-Optimierung bei Eingerätebenutzung (Wegfall der Umfangreichen ROM Code Suche, Wrapper Macros).

5.7.2 Dokumentation der benutzerdefinerten Typen

5.7.2.1 ow_device_st

Typvereinbarung für einen One Wire Geräteeintrag.

Siehe auch:
ow_device_s

5.7.2.2 ow_rom_code_st

Typvereinbarung für einen One Wire ROM Code.

Siehe auch:
ow_rom_code_s

5.7.3 Dokumentation der Funktionen

5.7.3.1 unsigned char ow_buf_read (unsigned char * *buf*, size_t *buf_size*, unsigned char *crc_included*)

Bytes vom One Wire Bus in einen Puffer lesen.

Mit dieser Funktion können unter Zuhilfenahme von ow_byte_read() so viele Bytes hintereinander vom One Wire Bus in den über *buf* referenzierten Puffer gelesen werden, wie über *buf_size* angegeben sind. Auf Wunsch erfolgt zusätzlich mit crc8_ow() ein Test der im letzten Byte enthaltenen CRC Summe, wenn *crc_included* ungleich Null ist. Das jeweilige One Wire Gerät, von dem gelesen werden soll, muß zuvor durch andere Funktionen entsprechend vorbereitet werden (z.B. mit ow_function()).

Parameter:
buf Zeiger auf den Anfang des Puffers.
buf_size Anzahl der Bytes im Puffer (Größe).
crc_included Anzeige über enthaltene CRC Summe.

Rückgabe:
Die Funktion ow_buf_read() gibt bei fehlerhafter Übertragung eine Null(0) Zurück. Ansonsten wird eine Eins(1) zurückgegeben.

5.7.3.2 unsigned char ow_buf_write (unsigned char * *buf*, size_t *buf_size*, unsigned char *crc_included*)

Bytes aus einem Puffer zum One Wire Bus schreiben.

Mit dieser Funktion können unter Zuhilfenahme von ow_byte_write() so viele Bytes hintereinander aus den über *buf* referenzierten Puffer zum One Wire Bus geschrieben werden, wie über *buf_size* angegeben sind. Auf Wunsch erfolgt zusätzlich mit crc8_ow() die Berechnung einer CRC Summe im letzten Byte des Puffers, wenn *crc_included* ungleich Null ist. Das jeweilige One Wire Gerät, in das geschrieben werden soll, muß zuvor durch andere Funktionen entsprechend vorbereitet werden (z.B. mit ow_function()).

Warnung:

Soll die Berechnung der CRC Summe innerhalb dieser Funktion erfolgen, dann darf das letzte Byte im Puffer keine relevanten Nutzdaten enthalten. Das letzte Byte `buf[(buf_size-1)]` wird mit der CRC Summe über die vorgelagerten Bytes `buf[0..(buf_size-2)]` überschrieben!

Parameter:

buf Zeiger auf den Anfang des Puffers.

buf_size Anzahl der Bytes im Puffer (Größe).

crc_included Anzeige über enthaltene CRC Summe.

Rückgabe:

Die Funktion ow_buf_read() gibt bei fehlerhafter Übertragung eine Null(0) zurück. Ansonsten wird eine Eins(1) zurückgegeben.

5.7.3.3 unsigned char ow_byte_read (void)

Lesen eines Bytes vom One Wire Bus.

Mit dieser Funktion wird die Signalfolge zum Lesen von genau einem Byte vom One Wire Bus ausgelöst. Das jeweilige One Wire Gerät, von dem gelesen werden soll, muß zuvor durch andere Funktionen entsprechend vorbereitet werden (z.B. mit ow_function()).

Rückgabe:

Die Funktion ow_byte_read() gibt das vom Bus gelesene Byte zurück.

5.7.3.4 unsigned char ow_byte_write (unsigned char *byte*)

Schreiben eines Bytes zum One Wire Bus.

Mit dieser Funktion wird die Signalfolge zum Schreiben von genau einem Byte am One Wire Bus ausgelöst. Das zu schreibende Byte wird mit *byte* übergeben. Das jeweilige One Wire Gerät, in das geschrieben werden soll, muß zuvor durch andere Funktionen entsprechend vorbereitet werden (z.B. mit ow_function()).

Parameter:

byte zu schreibendes Byte.

Rückgabe:

Die Funktion ow_byte_write() gibt immer eine Eins(1) zurück, da der Erfolg oder Mißerfolg einer Byteübertragung am One Wire Bus nicht direkt kontrolliert werden kann.

5.7.3.5 unsigned char ow_function (unsigned char *code*, unsigned char *device*)

Bytes aus einem Puffer zum One Wire Bus schreiben.

Mit dieser Funktion wird unter Zuhilfenahme von ow_byte_write() und ow_buf_write() ein neues Gerätekommando *code* an das über *device* referenzierte One Wire Gerät initiert. Hierbei stellt *device* einen Index in dem der Funktion ow_rom_search() übergebenen Feld vom Typ `ow_device_st` dar und zeigt so ein bestimmtes Gerät am Bus an.

Parameter:

code One Wire Gerätekommando

device Referenz auf das zu benutzende One Wire Gerät

Rückgabe:

Die Funktion ow_function() gibt bei fehlerhafter Übertragung eine Null(0) zurück. Ansonsten wird eine Eins(1) zurückgegeben.

5.7.3.6 unsigned char ow_init (void)

(Re-)Initialisirung des One Wire Moduls.

Mit Aufruf dieser Funktion wird der One Wire Bus durch ein entsprechendes Busreset initialisiert und die interne Geräteverwaltung zurückgesetzt.

Rückgabe:

Die Funktion ow_init() gibt eine Null(0) zurück, wenn keines der angeschlossenen One Wire Geräte das Busreset beantwortet. Ansonsten wird eine Eins(1) zurückgegeben.

5.7.3.7 unsigned char ow_ready (void)

Primitiver Bereitschaftstest.

Mit dieser Funktion kann die Bereitschaft eines One Wire Gerätes nach Ausführung eines vorher abgesetzten Kommandos durch einen einfachen Bitlesezyklus getestet werden. Viele One Wire Geräte zeigen so die Beendigung einer Aktion an, so z.B. die einmalige Temperaturmessung im Sensor DS1820.

Zum Schutz gegen endloses Warten aufgrund fehlerhafter One Wire Geräte oder Störungen beim Lesen eines Bits vom Bus, ist der Testzyklus mit einer Zeitschleife von 1 Sekunde versehen. Nach erfolglosem Test über diese Zeit hinaus wird automatisch abgebrochen!

Rückgabe:

Die Funktion ow_ready() gibt eine Null(0) zurück, wenn vom One Wire Gerät innerhalb der Zeitschleife von 1 Sekunde keine Bereitschaft angezeigt wurde. Ansonsten wird ein Wert ungleich Null(!=0) für einen erfolgreichen Bereitschaftstest zurückgegeben.

5.7.3.8 unsigned char ow_rom_search (ow_device_st * *ow_dev*)

One Wire ROM Code Ermittlung.

Mit dieser Funktion wird der One Wire Bus so wie in [2] beschrieben nach Geräten durchsucht, wobei deren ROM Codes ermittelt und in einem Feld vom Typ `ow_device_st` hinterlegt werden. Der für dieses Feld notwendige Speicher wird **nicht** von der Bibliothek bereitgestellt, sondern muß vom Benutzer über den Zeiger *ow_dev* referenziert werden.

Warnung:

Für die interne Geräteverwaltung wird das über *ow_dev* referenziert Feld durch die Bibliothek weiter benutzt; auch über die Arbeit von ow_rom_search() hinaus. Sollte sich also die Größe dieses Feldes, welche immer in direktem Bezug zu der Anzahl physisch vorhandener One Wire Geräte steht, verändern, so muß eine Reinitialisierung mittels ow_init() und ow_rom_search() erfolgen.

Parameter:

ow_dev Zeiger auf ein Feld für die interne Geräteverwaltung.

Rückgabe:

Die Funktion ow_rom_search() gibt eine Null(0) zurück, wenn kein One Wire Gerät gefunden wurde. Ansonsten wird die Anzahl der gefundenen One Wire Geräte zurückgegeben. Diese **muß** der Größe des durch *ow_dev* referenzierten Feldes entsprechen.

5.7.3.9 unsigned char w1_crc8 (void * *p*, unsigned char *n*) `[inline, static]`

CodeVision API

Aliasfunktion für crc8_ow(). Ermittelt die im Dallas One Wire Bus verwendete 8 Bit CRC Summe für den über den Zeiger referenzierten Block der angegebenen Länge.

Parameter:

p zeigt auf den Block.

n Größe des Blocks in Byte.

Rückgabe:

Die Funktion w1_crc8() besitzt den Rückgabewert von crc8_ow().

5.7.3.10 unsigned char w1_init (void) [inline, static]

CodeVision API

Aliasfunktion für ow_init(). Initialisiert die One Wire Geräte am Bus.

Rückgabe:

Die Funktion w1_init() besitzt den Rückgabewert von ow_init().

5.7.3.11 unsigned char w1_read (void) [inline, static]

CodeVision API

Aliasfunktion für ow_byte_read(). Liest ein Byte vom One Wire Bus.

Rückgabe:

Die Funktion w1_read() besitzt den Rückgabewert von ow_byte_read().

5.7.3.12 unsigned char w1_write (unsigned char *data*) [inline, static]

CodeVision API

Aliasfunktion für ow_byte_write(). Schreibt ein Byte zum One Wire Bus.

Parameter:

data zu schreibendes Byte.

Rückgabe:

Die Funktion w1_write() besitzt den Rückgabewert von ow_byte_write().

6 avrhal-lib Datenstruktur-Dokumentation

6.1 ow_dev_flags_s Strukturreferenz

One Wire Geräteflags.

`#include "avrhal/ow.h"`

Datenfelder

- unsigned alarm:1
- unsigned parpower:1

6.1.1 Ausführliche Beschreibung

One Wire Geräteflags.

6.1.2 Dokumentation der Datenelemente

6.1.2.1 unsigned alarm

Alarm

6.1.2.2 unsigned parpower

parasitäre Energieversorgung

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- ow.h

6.2 ow_device_s Strukturreferenz

One Wire Geräteeintrag.

`#include "avrhal/ow.h"`

Datenfelder

- ow_rom_code_st rom_code
- unsigned char status
- ow_dev_flags_s flags

6.2.1 Ausführliche Beschreibung

One Wire Geräteeintrag.

Jedes One Wire Gerät besitzt neben dem allgemeingültigen ROM Code noch weitere funktionsbezogene Status- und Kontrollinformationen. Diese werden in diesem strukturierten Typ zusammengefaßt. Die Wahl zwischen der Benutzung des Statusbytes oder der differenzierten Flags hängt von der jeweiligen Geräteklasse ab. Mehr dazu findet man im Handbuch des zum ROM Code zugehörigen One Wire Gerätes.

6.2.2 Dokumentation der Datenelemente

6.2.2.1 struct ow_dev_flags_s flags

Geräteflags

6.2.2.2 ow_rom_code_st rom_code

One Wire ROM Code

6.2.2.3 unsigned char status

Statusbyte (z.B. DS2405)

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- ow.h

6.3 ow_rom_code_s Strukturreferenz

One Wire ROM Code.

`#include "avrhal/ow.h"`

Datenfelder

- unsigned char family
- unsigned char serial [6]
- unsigned char crc

6.3.1 Ausführliche Beschreibung

One Wire ROM Code.

Dieser strukturierte Typ stellt ausreichend Felder für einen One Wire ROM Code bereit. Dieser ROM Code ist nach [2] 64 Bit breit und besteht aus 1 Byte für die Geräteklasse, 6 Byte für eine Seriennummer und 1 Byte CRC Summe über die vorangegangenen 7 Byte.

6.3.2 Dokumentation der Datenelemente

6.3.2.1 unsigned char crc

8 Bit CRC Summe

6.3.2.2 unsigned char family

Geräteklasse

6.3.2.3 unsigned char serial[6]

Seriennummer

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- ow.h

7 avrhal-lib Zusätzliche Informationen

7.1 Anerkennungen

Mein erster Dank ist an Linus Torvalds und die vielen Mitarbeiter um ihn für den Aufbau und die rege Weiterentwicklung des freien Betriebssystemkerns ”Linux”, sowie allen GNU Entwicklern gerichtet. Nur mit der erfolgreichen Symbiose von GNU/Linux war mir diese Arbeit ohne Lizenzkosten möglich.

Diese Bibliothek versucht die Mühe und Arbeit einer großen Gruppe von Leuten zu vereinen. Ohne diese individuellen Anstrengungen würden wir keine **freie** API Implementierung für die Benutzung grundlegender Hardwarekomponenten in eigenen AVR Projekten haben. Wir alle sind zum Dank verpflichtet:

- Das GCC und BIN-Utilities Team, ohne deren Arbeit wir keinen freien C/C++ Compiler und Assembler für die Entwicklung von ausgesprochen optimalem AVR Code hätten.

- Theodore A. Roth <troth@verinet.com> und weitere Kernentwickler um ihn für die ausgereifte und umfangreiche Standard C Bibliothek für AVR Mikrocontroller, die gute Zusammenfassung vieler Dokumente und Anleitungen im Umgang mit den freien GNU Entwicklungswerkzeugen sowie die Fortführung und Pflege dieses Projektes unter: http://savannah.gnu.org/projects/avr-libc.

- Brajer Vlado <vlado.brajer@kks.s-net.net> für die Veröffentlichung seiner Quellen zur Ansteuerung von alphanumerischen LCD unter http://bray.velenje.cx/avr.

- Dallas Semiconductors für die Bereitstellung von Codebeispielen und Development Kits für ihren 1-Wire Bus, alias One Wire, unter http://www.ibutton.com/software/1wire/wirekit.html

- Alle Leute von http://www.mikrocontroller.com für den Entwurf und den günstigen Vertrieb des interessanten AVR-Ctrl Boards.

- Alle Leute mit Anregungen, Berichtigungen oder sonstigen Hilfen zum Fortbestand dieser Bibliothek (siehe AUTHORS).

- Und schließlich alle Leute, die diese Bibliothek in ihren Projekten benutzen und somit weiter empfehlen.

Für die Erstellung der Dokumentation wurde die Hilfe verschiedener freier Projekte und Programme in Anspruch genommen. Es liegt im Interesse deren Entwickler, sie namendlich zu nennen. Ich möchte mich somit bedanken bei:

- Editor **VIM** (Vi IMproved) von Bram Moolenaar <bram@vim.org>: http://www.vim.org

- Dokumentationssystem **Doxygen** von Dimitri van Heesch <dimitri@stack.nl>: http://www.doxygen.org

- Textsatzsystem TeX/LaTeX in der Distribution **TeTeX** von Thomas Esser <te@dbs.uni-hannover.de>: http://www.tug.org/teTeX

- Vektorgrafikeditor **Xfig** von Supoj Sutanthavibul <supoj@nwg.nectec.or.th>, Brian V. Smith <BVSmith@lbl.gov> und Paul King <king@dstc.edu.au>: http://www.xfig.org

- Vektorgrafikkonverter **pstoedit** von Wolfgang Glunz <wglunz@pstoedit.net>: http://www.pstoedit.net

- Postscript Interpreter **Ghostscript** von verschiedenen Entwicklern: http://www.ghostscript.com

7.2 Frequently Asked Questions

AVR HAL steht für AVR Hardware Abstraction Layer und soll spezifische Hardwaregegebenheiten verschiedener Boards auf Basis eines AVR Microcontrollers in der Form einer API verbergen. Hierzu werden von der gleichnamigen Bibliothek standardisierte Funktionsaufrufe bereitgestellt, welche in Abhängigkeit zur konfigurierten Hardware die entsprechenden Portzugriffe auf unterster Ebene kapselt. AVR HAL ist aus einer kleinen Programmierbibiliothek für das AVR-Ctrl Board entstanden.

Zurück zu FAQ Index.

AVR-Ctrl ist der Name für ein spezielles AVR Board Design. Es stammt von der deutschen AVR Portalseite http://www.mikrocontroller.com und kann von dort als Bausatz oder unbestückte Platine bezogen werden.

Zurück zu FAQ Index.

Im Frühjahr 2002 begann ich mich in meiner Freizeit intensiv mit der Programmierung von AVR Controllern zu beschäftigen. Hierzu benötigte ich eine kleine, preiswerte Hardware mit einem AVR mittlerer Leistungsklasse. Die Wahl fiel schnell auf AVR-Ctrl. Ferner war es mein Ziel, ausschließlich mit ”freien” Entwicklungswerkzeugen, in erster Linie dem GNU C Compiler, unter dem ”freien” Betriebssystem Linux zu programmieren. Die meißten Anwendungen für AVR-Ctrl wurden und werden mit dem komerziellen C Compiler ”CodeVision” erstellt. Dieser Compiler ist aber nicht ”frei” und nur für M$ Windows erhältlich.

Mit dieser Bibliothek soll die Möglichkeit geschaffen werden, mit dem GNU C Compiler ebenso effizient und schnell wie unter CodeVision Anwendungen für AVR-Ctrl zu erstellen. Zunächst liegt der Schwerpunkt bei der breiten Unterstützung der vorhandenen Hardwarekomponenten. Parallel dazu soll aber auch die API Konformität zu CodeVision gewahrt bleiben.

Zurück zu FAQ Index.

7.3 Beispielprojekt: led

Originalboard:
AVR-Ctrl

Der Name dieses Beispieles ist ein wenig irreführend. Das Beispiel soll zeigen, wie mit Hilfe der LED API der hierfür konfigurierte 8 Bit Ausgabeport für einfache Bitausgaben benutzt werden kann (siehe Bild). Hinter dem Treiber `IC2` können über die Anschlüsse LAST[7..0] externe Verbraucher, wie Lampen oder Motorwicklungen angeschlossen werden. Je Bit von Port B kann eine separate Last geschaltet werden. Diese sind aufgrund des offenen Kollektors von `IC2` bei Ausgabe einer 1 eingeschlatet und bei 0 ausgeschaltet. Auf dem Herkunftsboard *AVR-Ctrl* sind parallel zu diesen Ausgängen LEDs angeschlossen. Somit kann man auf diesem Board die Zustände der geschalteten Lasten kontrollieren (siehe auch Beispielprojekt: ledband).

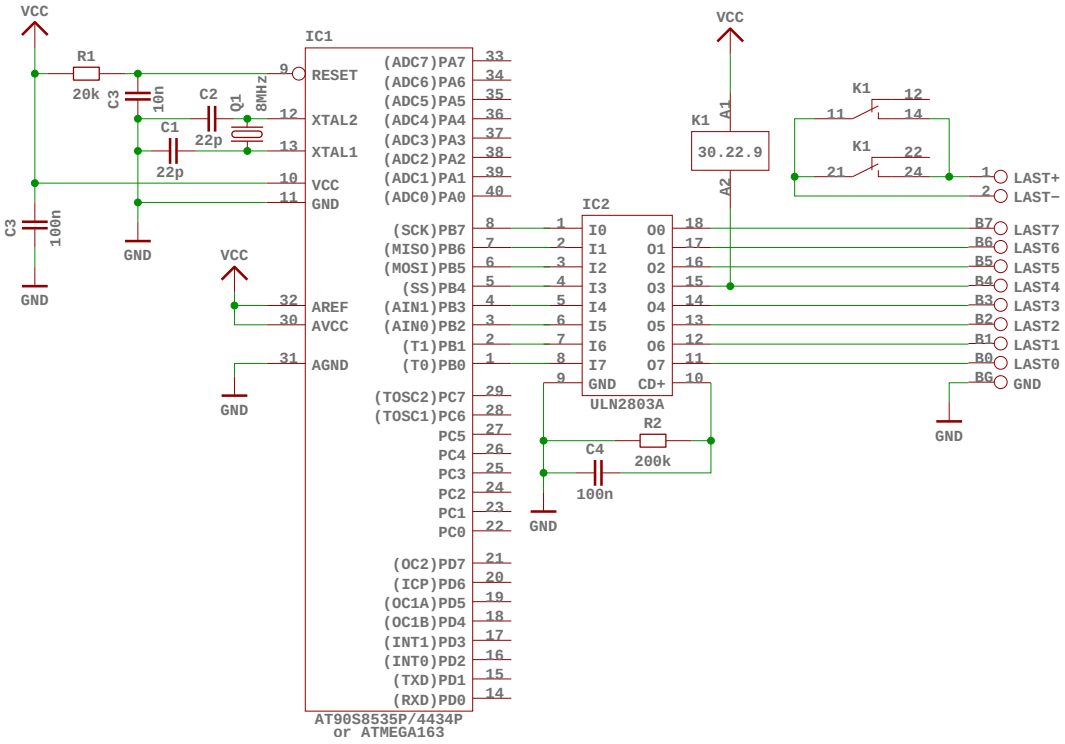


Abbildung 3: Schaltplan angeschlossener Lasten

7.3.1 Konfiguration der Bibliothek

Die Bibliothek muß intern mit den folgenden Parametern übersetzt werden. Dies kann durch Benutzung der originalen Boardkonfiguration über `--enable-board=avrctrl-8535-8mhz` erreicht werden:

- AVRHAL_LIB_LED_PORT="PORTB"
- AVRHAL_LIB_LED_PORT_DDR="DDRB"
- AVRHAL_LIB_LED_PORT_IN="PINB"
- AVRHAL_LIB_LED_OFF="0"

7.3.2 Verwendung der Bibliothek

7.3.2.1 Der Quellcode

main.c

Zunächst werden alle notwendigen Headerdateien der AVR HAL Bibliothek eingebunden:

```
#include <avrhal/delay.h>

#include <avrhal/led.h>
```

Danach werden notwendige globale Definitionen und Deklarationen vorgenommen:

```
enum {
        UP,
        DOWN
} direction;
```

In der Mainroutine wird vor Eintritt in die Endlosschleife das LED Port initialisiert. Das muß mindestens einmal nach einem Reset und noch vor dem ersten Portzugriff erfolgen:

```
int main(void)
{
        uint8_t led = 0x01;
        led_init();
```

Nach der erfolgreichen Basisinitialisierung kann die Hauptfunktion in der Endlosschleife ablaufen. In unserem Beispiel wird dazu demonstrativ eine 1 am LED Port alle 100ms hin und her geschoben. Alle angeschlossenen Lasten sollten nacheinander ein- und ausgeschaltet werden:

```
        while (1)
        {
                led_put(led);
```

```
                delay_ms(100);
                /* direction state machine */
                switch (direction)
                {
                        case UP:
                                /* count up means
                                 * shift left */
                                led <<= 1;
                                if (!led)
                                {
                                        /* change direction */
                                        led = 0x80;
                                        direction = DOWN;
                                }
                                break;
                        case DOWN:
                                /* count down means
                                 * shift right */
                                led >>= 1;
                                if (!led)
                                {
                                        /* change direction */
                                        led = 0x01;
                                        direction = UP;
                                }
                                break;
                        default:
                                break;
                }
        }
} /* main() */
```

Der vollständige Quellcode kann der AVR HAL Bibliothek unter `doc/examples/led` entnommen werden.

7.3.2.2 Übersetzung

C Quelle zu Objekt compilieren

```
[avr@host] > avr-gcc -g -O2 -Wall -Wstrict-prototypes -Wa \
                     -mmcu=at90s8535 -DAVRHAL_CONFIG_avrctrl_8535_8mhz \
                     -c -o main.o main.c
```

Objekt zu ELF Datei linken

```
[avr@host] > avr-gcc -Wl,-Map=led.map,--cref -mmcu=at90s8535 \
                     -o led.out main.o -lavrhal-avrctrl-8535-8mhz
```

ELF Datei in Binärdatei, Intel HEX und Motorola S-Record Format überführen

```
[avr@host] > avr-objcopy -O binary -R .eeprom led.out led.out-rom.bin
[avr@host] > avr-objcopy -O ihex -R .eeprom led.out led.out-rom.hex
[avr@host] > avr-objcopy -O srec -R .eeprom led.out led.out-rom.s19
```

Zum Beispiel Motorolas S-Record:

```
S00E00006C65642D726F6D2E73313936
S113000010C02AC029C028C027C026C025C024C0CB
S113001023C022C021C020C01FC01EC01DC01CC0E0
S11300201BC011241FBECFE5D2E0DEBFCDBF10E060
S1130030A0E6B0E0ECEBF0E003C0C89531960D9279
S1130040A036B107D1F710E0A0E6B0E001C01D92E0
S1130050A236B107E1F701C0D3CFCFE5D2E0DEBFCE
S1130060CDBFC1E08FEF87BB01E010E0C8BB84E6E1
S113007090E018D0809160009091610000 9751F059
S11300800197A1F7C69591F7C1E010926100109213
S11300906000ECCFCC0F51F7C0E8109361000093DF
S11300A06000E4CFE82FF92F309639F0A0EDB7E0E7
S10F00B01197F1F7A8953197C9F708954E
S9030000FC
```

Optionale Informationen:

```
[avr@host] > avr-objdump -x led.out > led.inf
[avr@host] > avr-size -d led.out > led.siz
[avr@host] > avr-size -x led.out >> led.siz
```

Zum Beispiel Sekmentgrößen:

```
   text        data     bss     dec     hex filename
    188       0       2     190      be led.out

   text    data     bss     dec     hex filename
   0xbc     0x0     0x2     190      be led.out
```

7.4 Beispielprojekt: ledband

Originalboard:
AVR-Ctrl

Im Gegensatz zum Beispielprojekt: led werden in diesem alle 8 Bit des hierfür konfigurierten LED Port für die Ansteuerung eines LED Balkens bestehend aus 10 LED benutzt (siehe Bild). Die unterste LED zwischen Anschluß 20 und 1 zeigt einfach nur die Betriebsspannung an. Die oberste LED zwischen Anschluß 11 und 10 wird nicht benutzt. Die restlichen LED werden über die LED Grafik API angesteuert. Das erlaubt die normiert skalierte Anzeige beliebig großer Werte als Punkt- oder Balkengrafik. In unserem Beispiel als Balkengrafik.

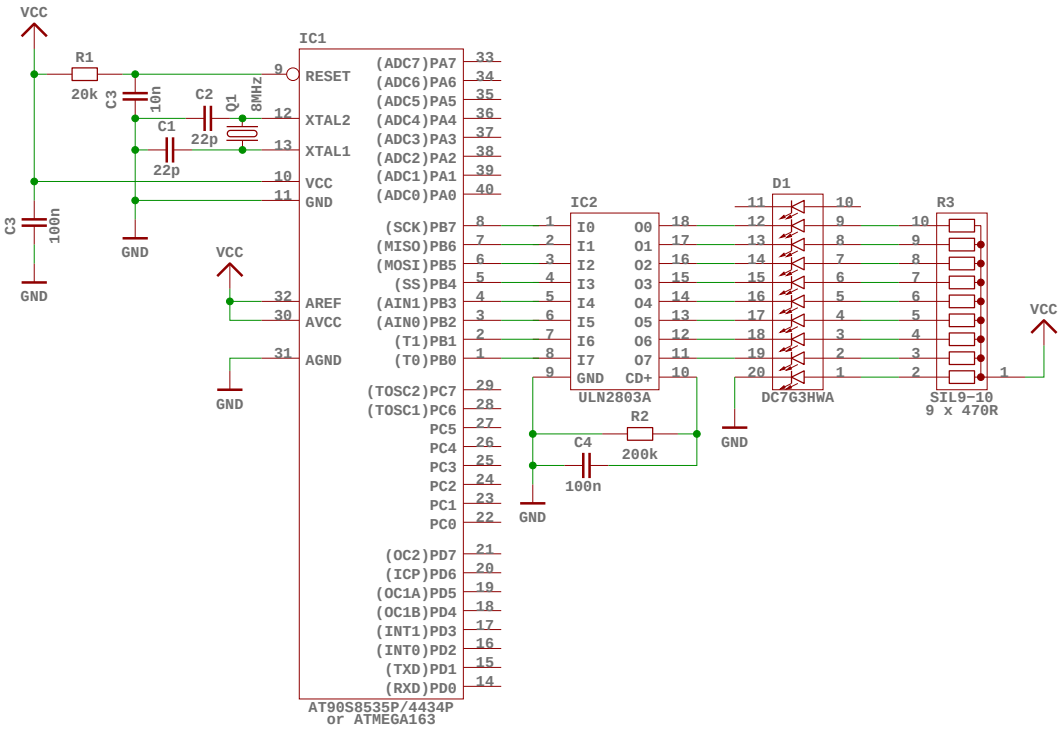


Abbildung 4: Schaltplan des angeschlossenen LED Balkens

7.4.1 Konfiguration der Bibliothek

Die Bibliothek muß intern mit den folgenden Parametern übersetzt werden. Dies kann durch Benutzung der originalen Boardkonfiguration über `--enable-board=avrctrl-8535-8mhz` erreicht werden:

- AVRHAL_LIB_LED_PORT="PORTB”
- AVRHAL_LIB_LED_PORT_DDR="DDRB”
- AVRHAL_LIB_LED_PORT_IN="PINB”
- AVRHAL_LIB_LED_OFF="0”

7.4.2 Verwendung der Bibliothek

7.4.2.1 Der Quellcode

main.c

Zunächst werden alle notwendigen Headerdateien der AVR HAL Bibliothek eingebunden:

```
#include <avrhal/delay.h>

#include <avrhal/led.h>
```

Danach werden notwendige globale Definitionen und Deklarationen vorgenommen:

```
#define MAX_LEVEL 32
enum {
        UP,
        DOWN
} direction;
```

Der hier definierte Wert *MAX_LEVEL* wird später für die Normierung herangezogen. In der Mainroutine wird vor Eintritt in die Endlosschleife das LED Port initialisiert. Das muß mindestens einmal nach einem Reset und noch vor dem ersten Portzugriff erfolgen. Jetzt kann der Grafikteil der LED API parametrisiert werden. Hierzu wird mit *2*MAX_LEVEL* der größte Anzeigbare Wert für die interne Normierung definiert und mit *LED_BAR* und *LED_MARGIN(4)* die Balkengrafik mit Nullpunkt bei Bit 4 aktiviert:

```
int main(void)
{
        int8_t led = 0;
        led_init();
        led_graph_control(2*MAX_LEVEL, LED_BAR | LED_MARGIN(/* Bit */ 4 /* is margin */));
```

Nach der erfolgreichen Basisinitialisierung kann die Hauptfunktion in der Endlosschleife ablaufen. In unserem Beispiel wird dazu demonstrativ der Wertebereich von +/- MAX_LEVEL mit einer Zeitverzögerung je Wert von 10ms durchlaufen und am LED Port über die LED Grafikfunktion ausgegeben:

```
        while (1)
        {
                led_graph_put((int16_t)led);
                delay_ms(10);
                /* direction state machine */
                switch (direction)
                {
                        case UP:
                                /* count up means
                                 * shift left */
                                led++;
                                if (led == MAX_LEVEL)
                                {
                                        /* change direction */
                                        direction = DOWN;
                                }
                                break;
                        case DOWN:
                                /* count down means
                                 * shift right */
                                led--;
                                if (led == -MAX_LEVEL)
```

```
                                {
                                        /* change direction */
                                        direction = UP;
                                }
                                break;
                        default:
                                break;
                }
        }
    } /* main() */
```

Der vollständige Quellcode kann der AVR HAL Bibliothek unter `doc/examples/ledband` entnommen werden.

7.4.2.2 Übersetzung

C Quelle zu Objekt compilieren

```
[avr@host] > avr-gcc -g -O2 -Wall -Wstrict-prototypes -Wa \
                     -mmcu=at90s8535 -DAVRHAL_CONFIG_avrctrl_8535_8mhz \
                     -c -o main.o main.c
```

Objekt zu ELF Datei linken

```
[avr@host] > avr-gcc -Wl,-Map=ledband.map,--cref -mmcu=at90s8535 \
                     -o ledband.out main.o -lavrhal-avrctrl-8535-8mhz
```

ELF Datei in Binärdatei, Intel HEX und Motorola S-Record Format überführen

```
[avr@host] > avr-objcopy -O binary -R .eeprom ledband.out ledband.out-rom.bin
[avr@host] > avr-objcopy -O ihex -R .eeprom ledband.out ledband.out-rom.hex
[avr@host] > avr-objcopy -O srec -R .eeprom ledband.out ledband.out-rom.s19
```

Zum Beispiel Motorolas S-Record:

```
S01200006C656462616E642D726F6D2E7331399D
S113000010C02AC029C028C027C026C025C024C0CB
S113001023C022C021C020C01FC01EC01DC01CC0E0
S11300201BC011241FBECFE5D2E0DEBFCDBF10E060
S1130030A0E6B0E0E2E3F2E003C0C89531960D9289
S1130040A236B107D1F710E0A2E6B0E001C01D92DC
S1130050A636B107E1F701C0D3CFCFE5D2E0DEBFCA
S1130060CDBF10E08FEF87BB69E080E490E0C4D09F
S1130070C0E0D0E08C2F9D2FB0D08AE090E023D058
S113008080916400909165000097 89F0019791F741
S11300901150103E29F0C12FDD27C7FDD095EACFBE
S11300A0109265001092640 0C0EEDFEFE3CF1F5F93
S11300B0103289F781E090E09093650080936400AA
S11300C0C0E2D0E0D7CFE82FF92F309639F0A0ED79
S11300D0B7E01197F1F7A8953197C9F708958930DA
S11300E0910584F0282F392F97FF02C0295F3F4FD5
S11300F043E0359527954A95E1F7309363002093C3
S1130100620008952FEF883F92077CF497FD0796CD
S113011023E0959587952A95E1F7909581959F4FD2
S113012090936300809362000895109263001092 8C
```

```
S113013062000895089598 2F80916000282F2695D5
S11301402770290F80FF3CC0622F772767FD7095C9
S1130150892F992787FD9095F72FE62FE81BF90B38
S1130160E617F707A4F421E030E0932F822F02C0B2
S1130170880F991F6A95E2F7482F480F4150932F33
S1130180822F02C0880F991FEA95E2F715C06E17F7
S11301907F07B4F421E030E0932F822F02C0880F50
S11301A0991FEA95E2F7482F4150932F822F02C0FE
S11301B0880F991F6A95E2F78195482348BB0895F3
S11301C027FF02C018BA089581E090E002C0880FAA
S11301D0991F2A95E2F788BB089560916200709197
S11301E063006115710529F00ED0972F862FA3DFC8
S11301F0089588BB089508951F93162F70DF1093F8
S1130200600001F910895AA1BBB1B51E107C0AA1FE0
S1130210BB1FA617B70710F0A61BB70B881F991FA3
S11302205A95A9F780959095682F792F8A2F9B2F3F
S105023008952B
S10502320100C5
S9030000FC
```

Optionale Informationen:

```
[avr@host] > avr-objdump -x ledband.out > ledband.inf
[avr@host] > avr-size -d ledband.out > ledband.siz
[avr@host] > avr-size -x ledband.out >> ledband.siz
```

Zum Beispiel Sekmentgrößen:

```
   text        data     bss     dec     hex filename
    562       2       4     568     238 ledband.out

   text    data     bss     dec     hex filename
  0x232     0x2     0x4     568     238 ledband.out
```

7.5 Beispielprojekt: display

Originalboard:

AVR-Ctrl

Mit diesem Beispiel soll ein Teil der LCD API vorgestellt werden. Das alphanumerische LCD wird über einen 4 Bit Datenkanal an den vorkonfigurierten LCD Port zusammen mit den Steuersignalen angeschlossen (siehe Bild). Ein Portanschluß bleibt dabei unbenutzt.

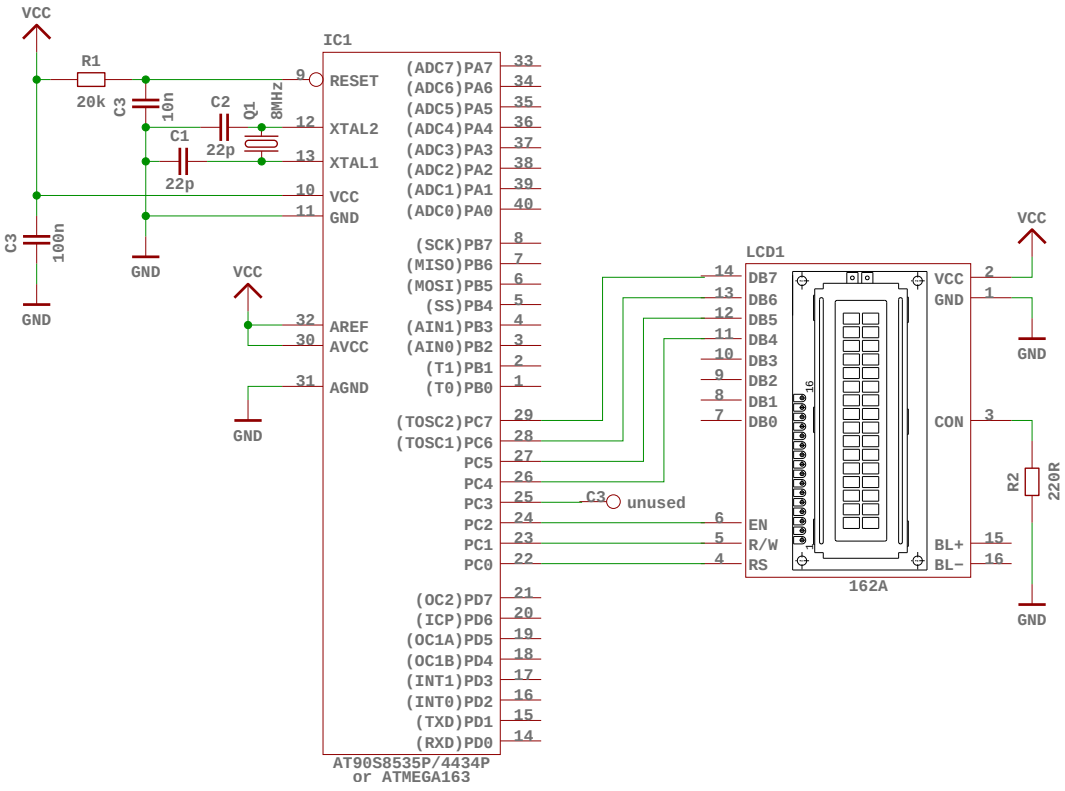


Abbildung 5: Schaltplan des angeschlossenen LCD

7.5.1 Konfiguration der Bibliothek

Die Bibliothek muß intern mit den folgenden Parametern übersetzt werden. Dies kann durch Benutzung der originalen Boardkonfiguration über `--enable-board=avrctrl-8535-8mhz` oder `--enable-board=avrctrl-8535-8mhz-small` erreicht werden:

- AVRHAL_LIB_LCD_PORT="PORTC”
- AVRHAL_LIB_LCD_PORT_DDR="DDRC”
- AVRHAL_LIB_LCD_PORT_IN="PINC”
- AVRHAL_LIB_LCD_RS="0”
- AVRHAL_LIB_LCD_RD="1”
- AVRHAL_LIB_LCD_EN="2”
- AVRHAL_LIB_LCD_UNUSED_BIT="3”
- bei avrctrl-8535-8mhz-small: AVRHAL_LIB_LCD_COM=LCD_DELAY
- bei avrctrl-8535-8mhz: AVRHAL_LIB_LCD_COM=LCD_POLLING

In unserem Beispiel benutzen wir dei `small` Boardkonfiguration, um kleineren Code zu erhalten. Es ist also `AVRHAL_LIB_LCD_COM=LCD_DELAY` vorkonfiguriert.

7.5.2 Verwendung der Bibliothek

7.5.2.1 Der Quellcode

main.c

Zunächst werden alle notwendigen Headerdateien der AVR HAL Bibliothek eingebunden:

```
#include <avrhal/delay.h>

#include <avrhal/lcd.h>
```

Danach werden notwendige globale Definitionen und Deklarationen vorgenommen, wobei für den Überlauf des AVR Zählers ein Signalhändler aktiviert wird:

```
uint16_t cnt = 0;
SIGNAL(SIG_OVERFLOW0)
{
        cnt++;
}
```

In der Mainroutine wird vor Eintritt in die Endlosschleife der AVR Zähler entsprechend unseren Bedürfnissen eingerichtet, das LCD Port initialisiert und die Meldung *"–= AVR-Ctrl =–"* in der ersten Zeile des LCD ausgegeben. Der LCD API wird bei der Initialisierung die Größe des LCD von 16 Zeichen in 2 Zeilen übergeben:

```
int main(void)
{
        outp(BV(CS02)|BV(CS00), TCCR0); // tmr0 @ 7,8125 kHz
        timer_enable_int(BV(TOIE0));
        sei();
        lcd_init((2 /* rows */ << 6) + 16 /* columns */);
        lcd_putstr("--= AVR-Ctrl =--");
        lcd_putstr("Counter = ");
        lcd_control(1, 0, 0);                   // cursor block off
```

Nach der erfolgreichen Basisinitialisierung wird in der Endlosschleife kontinuierlich der Zählerstand eines 16 Bit Wertes ausgegeben, welcher immer bei Überlauf des AVR Zählers inkrementiert wird:

```
        while (1)
        {
                lcd_gotoxy(10,1);
                lcd_print5(cnt);
        }
} /* main() */
```

Der vollständige Quellcode kann der AVR HAL Bibliothek unter `doc/examples/display` entnommen werden.

7.5.2.2 Übersetzung

C Quelle zu Objekt compilieren

```
[avr@host] > avr-gcc -g -O2 -Wall -Wstrict-prototypes -Wa \
                     -mmcu=at90s8535 -DAVRHAL_CONFIG_avrctrl_8535_8mhz_small \
                     -c -o main.o main.c
```

Objekt zu ELF Datei linken

```
[avr@host] > avr-gcc -Wl,-Map=display.map,--cref -mmcu=at90s8535 \
                     -o display.out main.o -lavrhal-avrctrl-8535-8mhz-small
```

ELF Datei in Binärdatei, Intel HEX und Motorola S-Record Format überführen

```
[avr@host] > avr-objcopy -O binary -R .eeprom display.out display.out-rom.bin
[avr@host] > avr-objcopy -O ihex -R .eeprom display.out display.out-rom.hex
[avr@host] > avr-objcopy -O srec -R .eeprom display.out display.out-rom.s19
```

Zum Beispiel Motorolas S-Record:

```
S0120000646973706C61792D726F6D2E73313971
S113000010C02AC029C028C027C026C025C024C0CB
S113001023C023C021C020C01FC01EC01DC01CC0DF
S11300201BC011241FBECFE5D2E0DEBFCDBF10E060
S1130030A0E6B0E0E0ECF3E003C0C89531960D9281
S1130040A038B107D1F710E0A0E8B0E001C01D92DC
S1130050A638B107E1F718C0D3CF1F920F920FB69D
S11300600F9211248F939F93809180009091810002F
S1130070019690938100809380009F918F910F90BF
S11300800FBE0F901F901895CFE5D2E0DEBFCDBF15
S113009085E083BF81E089BF789480E936D080E62B
S11300A090E0C6D081E790E0C3D084E094D061E0D2
S11300B08AE0A7D08091800090918100C8D0F7CFCA
S11300C0282F85B38870922F907F892B85BBAA9A9D
S11300D0AA9885B388702295207F822B85BBAA9A23
S11300E0AA980895282F95B39870807F8160982BE3
S11300F095BBAA9AAA9885B388702295207F21601F
S1130100822B85BBAA9AAA980895982F8F738093FF
S1130110820080937E00805C80937F00892F82958B
S11301208695869583 7011F0833021F483E0809363
S1130130840005C0991F9927991F9093840084B364
S1130140876F84BB80E29EE4E7D085B388708063C8
S113015085BBAA9AAA9888E893E1DED085B3887013
S1130160806385BBAA9AAA988EE690E0D5D085B321
S11301708870806385BBAA9AAA9882E390E0CCD069
S113018085B38870806285BBAA9AAA9882E390E0BE
S1130190C3D085B38870806285BBAA9AAA9885B3B8
S11301A08870806885BBAA9AAA9882E390E0B4D04C
S11301B085B3887085BBAA9AAA9885B3887080662F
S11301C085BBAA9AAA9882E390E0A6D0B0D02AD0A0
S11301D087E001D00895992782FF02C02CE001C076
```

```
S11301E028E081FF02C02A6001C0286080FF02C0AD
S11301F0296001C02860822F63DF82E390E08CD005
S11302000895282F962F8091840098239093850039
S1130210ECE7F0E0E90FF11D209383008081820F69
S11302208CD0089582E04CDF80ED97E075D008957E
S1130230CF93DF93D92FC82F8881882329F0899100
S113024082D088818823D9F7DF91CF910895CF9206
S1130250DF92EF92FF920F931F93E82EF92E50E155
S1130260C52E57E2D52E7D2D6C2D94D0062F172F39
S1130270862F805D68D0912F802F7D2D6C2D78D0B6
S1130280E81AF90A48EEC42E43E0D42E9F2D8E2D91
S11302907D2D6C2D7FD0062F172F862F805D53D098
S11302A0912F802F7D2D6C2D63D0E81AF90A34E646
S11302B0C32ED12C9F2D8E2D7D2D6C2D6BD0062F12
S11302C0172F862F805D3FD0912F802F7D2D6C2D91
S11302D04FD0E81AF90A9F2D8E2D6AE070E05AD0AB
S11302E0062F172F862F805D2ED0912F802F23E08D
S11302F0880F991F2A95E1F7800F911F800F911F96
S1130300E81AF90A8E2D805D1ED01F910F91FF907F
S1130310EF90DF90CF900895E82FF92F309631F0C9
S11303200000000000000000003197D1F7089581E03B
S1130330C7DE80ED97E0F0DF08958068C1DE80EDD0
S113034097E0EADF0895CEDE82E390E0E5DF809176
S113035083008F5F8093830090918200891731F42A
S1130360809185008F5F682F80E04BDF08950895AA
S11303705527002480FF02C0060E571F660F771F03
S11303806115710521F096958795009799F7952F3A
S1130390802D0895AA1BBB1B51E107C0AA1FBB1FD8
S11303A0A617B70710F0A61BB70B881F991F5A95FD
S11303B0A9F780959095682F792F8A2F9B2F089500
S11303C02D2D3D204156522D4374726C203D2D2D10
S11303D000436F756E746572203D2000004000007C
S9030000FC
```

Optionale Informationen:

```
[avr@host] > avr-objdump -x display.out > display.inf
[avr@host] > avr-size -d display.out > display.siz
[avr@host] > avr-size -x display.out >> display.siz
```

Zum Beispiel Sekmentgrößen:

```
   text        data     bss     dec     hex filename
    960      32       6     998     3e6 display.out

   text    data     bss     dec     hex filename
  0x3c0    0x20     0x6     998     3e6 display.out
```

7.6 Beispielprojekt: keys

Originalboard:

AVR-Ctrl

Dieses Beispiel soll die Benutzung der KEY API aufzeigen. Am vorkonfigurierten KEY Port sind dazu 5 Tasten direkt angeschlossen. (siehe Bild). Die Eingänge dieses Ports sind im Ruhezustand über R2 auf 0 kurzgeschlossen. Bei einer gedrückten Taste wird der zugehörige Eingang auf 1 gezogen. Drei der 8 Bits am KEY Port werden nicht von der KEY API interpretiert. Sie stehen der Anwendung frei zur Verfügung. Auf dem Herkunftsboard *AVR-Ctrl* könnten dort analoge Signale angeschlossen werden.

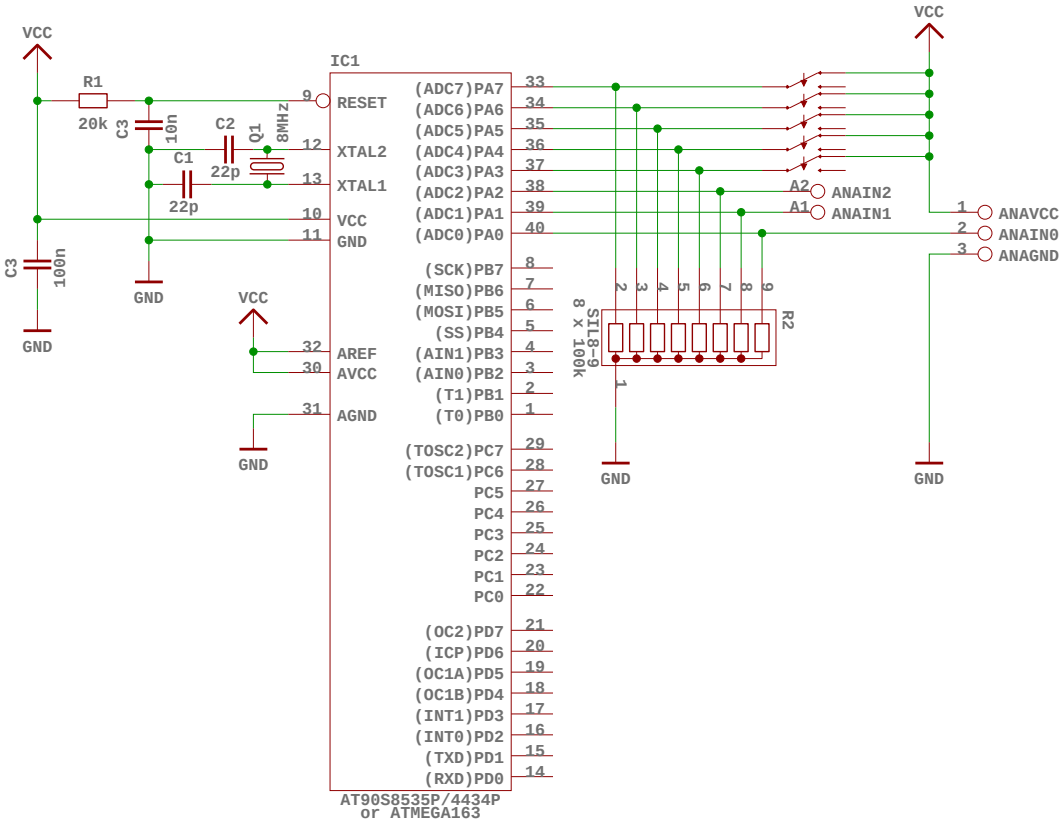


Abbildung 6: Schaltplan angeschlossener Tasten

Nicht im Schaltplan enthalten ist der in diesem Beispiel benutzte Anschluß des LCD als Anzeige. Hierzu wird auf das Beispielprojekt: display verwiesen.

7.6.1 Konfiguration der Bibliothek

Die Bibliothek muß intern mit den folgenden Parametern übersetzt werden. Dies kann durch Benutzung der originalen Boardkonfiguration über `--enable-board=avrctrl-8535-8mhz-small` erreicht werden:

- AVRHAL_LIB_KEY_PORT="PORTA”
- AVRHAL_LIB_KEY_PORT_DDR="DDRA”
- AVRHAL_LIB_KEY_PORT_IN="PINA”

7.6.2 Verwendung der Bibliothek

7.6.2.1 Der Quellcode

main.c

Zunächst werden alle notwendigen Headerdateien der AVR HAL Bibliothek eingebunden:

```
#include <avrhal/delay.h>

#include <avrhal/key.h>

#include <avrhal/lcd.h>
```

Eine globale Definition oder Deklaration ist bei diesem Beispiel nicht notwendig. In der Mainroutine wird vor Eintritt in die Endlosschleife das KEY und das LCD Port initialisiert:

```
int main(void)
{
        uint8_t scancode, scroll_pos = 0;
        uint8_t scroll[] = " press a key.. ";
        key_init();
        lcd_init((2 /* rows */ << 6) + 16 /* columns */);
        lcd_control(1, 0, 0);                   // cursor block off
```

Nach der erfolgreichen Basisinitialisierung wird in der Endlosschleife kontinuierlich alle 25ms der KEY Scancode ausgelesen und als dezimale und hexadezimale Zahl am LCD ausgegeben. Nebenbei wird ein Aufforderungstext zum Drücken einer Taste als Laufschrift in der obersten Zeile des LCD ausgegeben:

```
        while (1)
        {
                scancode = key_scancode();
                lcd_clear();
                lcd_gotoxy(4,1);
                lcd_printhex(scancode);
                lcd_gotoxy(10,1);
                lcd_print3(scancode);
                lcd_gotoxy(0,0);
                lcd_putstr(&scroll[strlen(scroll) - 1 - (scroll_pos++ >> 3)]);
                if (scroll_pos == 8 * strlen(scroll)) scroll_pos = 0;
                delay_us(25000);
        }
} /* main() */
```

Der vollständige Quellcode kann der AVR HAL Bibliothek unter `doc/examples/keys` entnommen werden.

7.6.2.2 Übersetzung

C Quelle zu Objekt compilieren

```
[avr@host] > avr-gcc -g -O2 -Wall -Wstrict-prototypes -Wa \
                     -mmcu=at90s8535 -DAVRHAL_CONFIG_avrctrl_8535_8mhz_small \
                     -c -o main.o main.c
```

Objekt zu ELF Datei linken

```
[avr@host] > avr-gcc -Wl,-Map=keys.map,--cref -mmcu=at90s8535 \
                     -o keys.out main.o -lavrhal-avrctrl-8535-8mhz-small
```

ELF Datei in Binärdatei, Intel HEX und Motorola S-Record Format überführen

```
[avr@host] > avr-objcopy -O binary -R .eeprom keys.out keys.out-rom.bin
[avr@host] > avr-objcopy -O ihex -R .eeprom keys.out keys.out-rom.hex
[avr@host] > avr-objcopy -O srec -R .eeprom keys.out keys.out-rom.s19
```

Zum Beispiel Motorolas S-Record:

```
S00F00006B6579732D726F6D2E733139AE
S113000010C02AC029C028C027C026C025C024C0CB
S113001023C022C021C020C01FC01EC01DC01CC0E0
S11300201BC011241FBECFE5D2E0DEBFCDBF10E060
S1130030A0E6B0E0E2EEF3E003C0C89531960D927D
S1130040A837B107D1F710E0A8E7B0E001C01D92CE
S1130050AC37B107E1F701C0D3CFCFE4D2E0DEBFC4
S1130060CDBFEE2480E1CC2EDD2E0894C11CD11C22
S1130070BD2DAC2DE0E6F0E001900D928A95E1F7FC
S11300808AB387708ABB80E976D084E0DAD089B3FA
S1130090082F1127087F107043D061E084E0E7D077
S11300A0802F49D161E08AE0E2D0802F992705D1E1
S11300B060E0862FDCD0FD2DEC2D01900020E9F7C7
S11300C031978E2D869586958695E81BF109319793
S11300D0E3948E2F9F2FE2D08E2D9927FD2DEC2DAA
S11300E001900020E9F73197EC19FD09EE0FFF1F8D
S11300F0EE0FFF1FEE0FFF1F8E179F0721F088EAF8
S113010091E603D0C4CFEE24FACFE82FF92F30962E
S113011031F0000000000000000003197D1F708958D
S113012081E004D080ED97E0F0DF0895282F85B3B7
S11301308870922F907F892B85BBAA9AAA9885B341
S113014088702295207F822B85BBAA9AAA9808954D
S1130150282F95B39870807F8160982B95BBAA9ABD
S1130160AA9885B388702295207F2160822B85BB55
S1130170AA9AAA980895982F8F738093780080093F1
S11301807200805C80937300892F82958695869592
S1130190837011F0833021F483E080937A0005C0EA
S11301A0991F9927991F90937A0084B3876F84BB12
S11301B080E29EE4AADF85B38870806385BBAA9A37
S11301C0AA9888E893E1A1DF85B38870806385BB32
S11301D0AA9AAA988EE690E098DF85B38870806327
S11301E085BBAA9AAA9882E390E08FDF85B38870D2
S11301F0806285BBAA9AAA9882E390E086DF85B3E1
S11302008870806285BBAA9AAA9885B38870806832
S113021085BBAA9AAA9882E390E077DF85B38870B9
S113022085BBAA9AAA9885B38870806685BBAA9A6A
S1130230AA9882E390E069DF73DF2AD087E001D0D7
S11302400895992782FF02C02CE001C028E081FFB5
```

```
S113025002C02A6001C0286080FF02C0296001C07A
S11302602860822F63DF82E390E04FDF0895282F18
S1130270962F80917A00982390937B00E0E7F0E03A
S1130280E90FF11D209379008081820F67D00895D2
S113029082E04CDF80ED97E038DF0895CF93DF9361
S11302A0D92FC82F8881882329F089915DD088812E
S11302B08823D9F7DF91CF9108950F931F93CF939C
S11302C0DF93D92FC82F83E0C83ED80760F58C2F61
S11302D09D2F64E670E06FD0062F172F862F805D68
S11302E043D0912F802F64E670E053D0C81BD90B04
S11302F08C2F9D2F6AE070E05ED0062F172F862F7B
S1130300805D32D0912F802F23E0880F991F2A958A
S1130310E1F7800F911F800F911FC81BD90B8C2F01
S1130320805D22D003C084E790E0B8DFDF91CF91F5
S11303301F910F910895CF93982F82958F70805DB0
S11303408A3308F0895F9F70C92FC05DCA3308F0F3
S1130350C95F0AD08C2F08D0CF9108958068E6DE5B
S113036080ED97E0D2DE0895F3DE82E390E0CDDE07
S11303708091790008F5F8093790090917800891730C
S113038031F480917B008F5F682F80E070DF0895E7
S11303900895552700248 0FF02C0060E571F660FDC
S11303A0771F6115710521F096958795009799F748
S11303B0952F802D0895AA1BBB1B51E107C0AA1FCE
S11303C0BB1FA617B70710F0A61BB70B881F991FF2
S11303D05A95A9F780959095682F792F8A2F9B2F8E
S10503E008957A
S11303E2207072657373206 1206B65792E2E200054
S10B03F2004000004572720096
S9030000FC
```

Optionale Informationen:

```
[avr@host] > avr-objdump -x keys.out > keys.inf
[avr@host] > avr-size -d keys.out > keys.siz
[avr@host] > avr-size -x keys.out >> keys.siz
```

Zum Beispiel Sekmentgrößen:

```
   text        data     bss     dec     hex filename
    994      24       4    1022     3fe keys.out

   text    data     bss     dec     hex filename
  0x3e2    0x18     0x4    1022     3fe keys.out
```

7.7 Beispielprojekt: ow

Originalboard:

AVR-Ctrl

Dieses Beispiel soll die Benutzung der *One Wire* OW API aufzeigen. Am vorkonfigurierten OW DQ Bit ist mindestens ein One Wire Gerät angeschlossen, in unserem Beispiel der Temperatursensor DS1820 von Dallas (siehe Bild). Wahlweise können

weitere OW Geräte angeschlossen werden. Als Anzeigeeinheit wird ein LCD benutzt, das wie im Beispielprojekt: display angeschlossen und benutzt wird.

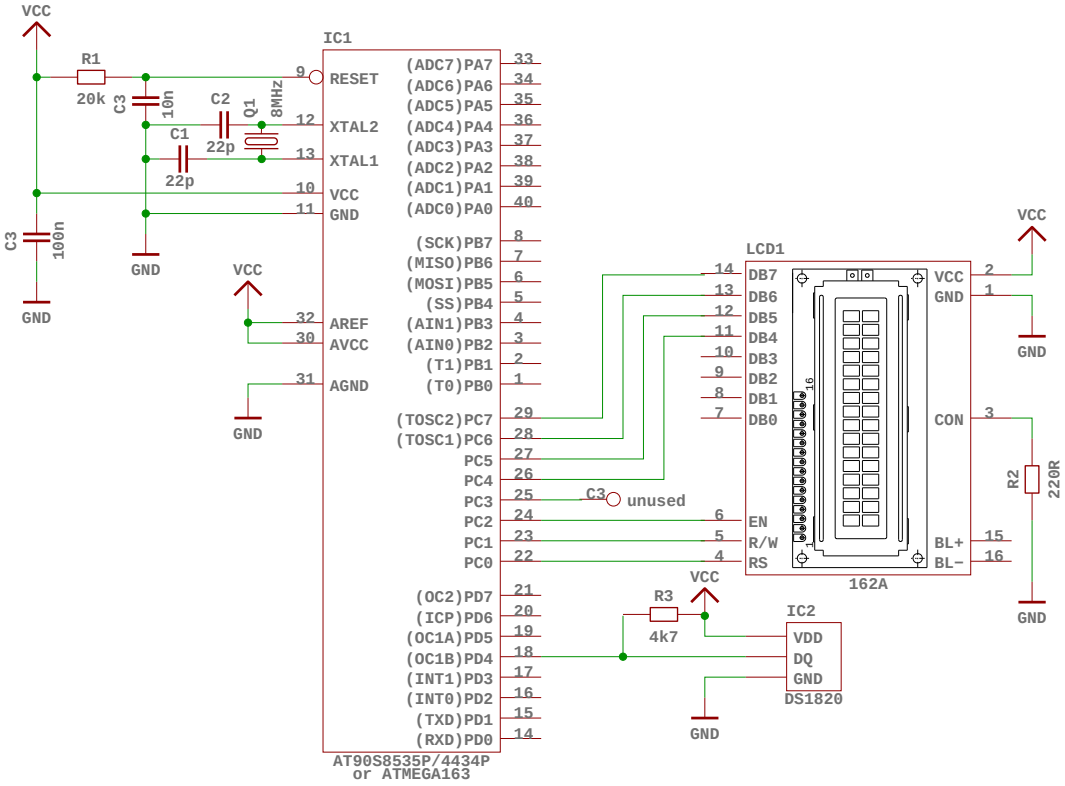


Abbildung 7: Schaltplan angeschlossener OW Geräte

7.7.1 Konfiguration der Bibliothek

Die Bibliothek muß intern mit den folgenden Parametern übersetzt werden. Dies kann durch Benutzung der originalen Boardkonfiguration über `--enable-board=avrctrl-8535-8mhz` erreicht werden:

- AVRHAL_LIB_OW_PORT="PORTD”
- AVRHAL_LIB_OW_PORT_DDR="DDRD”
- AVRHAL_LIB_OW_PORT_IN="PIND”
- AVRHAL_LIB_OW_DQBIT="4”

Wahlweise kann auch die Bibliotheksvariante `avrctrl-8535-8mhz-small` benutzt werden. Dabei wird aber die Anzahl der benutzbaren Geräte automatisch auf 1 beschränkt. Das muß auch bei der Beschaltung beachtet werden!

7.7.2 Verwendung der Bibliothek

7.7.2.1 Der Quellcode

main.c

Zunächst werden alle notwendigen Headerdateien der AVR HAL Bibliothek eingebunden:

```
#include <avrhal/delay.h>

#include <avrhal/lcd.h>

#include <avrhal/ow.h>
```

Eine globale Definition oder Deklaration ist bei diesem Beispiel nicht notwendig. In der Mainroutine wird vor Eintritt in die Endlosschleife das LCD Port und der OW Kommunikationsstack initialisiert und vorhandene OW Geräte gesucht. Schlägt das fehl, wird eine entsprechende Fehlermeldung ausgegeben:

```
int main(void)
{
        ow_device_st devarray[2];
        unsigned char devices;
        unsigned int  waiting = 5000;
        lcd_init((2 /* rows */ << 6) + 16 /* columns */);
        lcd_putstr("--= AVR-Ctrl =--");
        lcd_control(1, 0, 0);                   // cursor block off
        if (!ow_init()) {
                lcd_putstr("ERROR: ow_init()");
                while (1);
        }
        devices = ow_rom_search(devarray);
```

Nach der erfolgreichen Basisinitialisierung wird in der Endlosschleife kontinuierlich die Anzahl und der ROM Code der gefundenen OW Geräte ausgegeben. Zwischen dem Anzeigewechsel wird immer 5 Sekunden gewartet. Über das Define __AVRHAL_LIB_SMALL wird die jeweils benutzte Variante der Bibliothek identifiziert und der Code entsprechend anders gestaltet:

```
        while (1)
        {
                unsigned char dev;
                lcd_gotoxy(0,1); lcd_cleol();
#ifdef __AVRHAL_LIB_SMALL
                lcd_putstr("OW_DEVICES: ");
                lcd_print2(devices);
#else
                lcd_printf("OW_DEVICES: %d", devices);
#endif
                delay_ms(waiting);
                for (dev = 0; dev < devices; dev++) {
```

```
                        unsigned char snidx;
#ifdef __AVRHAL_LIB_SMALL
                        unsigned char *cp = (unsigned char *)&(devarray[dev].rom_code);
#endif
                        lcd_gotoxy(0,1); lcd_cleol();
#ifdef __AVRHAL_LIB_SMALL
                        for (snidx = 0; snidx < sizeof(ow_rom_code_st); snidx++) {
                                lcd_printhex(cp[snidx]);
                        }
#else
                        lcd_printf("%x", devarray[dev].rom_code.family);
                        for (snidx = 0; snidx < 6; snidx++) {
                                lcd_printf("%x", devarray[dev].rom_code.serial[snidx]);
                        }
                        lcd_printf("%x", devarray[dev].rom_code.crc);
#endif
                        delay_ms(waiting);
                }
        }
} /* main() */
```

Der vollständige Quellcode kann der AVR HAL Bibliothek unter `doc/examples/ow` entnommen werden.

Warnung:

Die Anzahl der angeschlossenen Geräte muß mit der Feldgröße von `devarray[]` übereinstimmen. In unserem Beispiel:

```
ow_device_st devarray[2];
```

C Quelle zu Objekt compilieren

```
[avr@host] > avr-gcc -g -O2 -Wall -Wstrict-prototypes -Wa \
                    -mmcu=at90s8535 -DAVRHAL_CONFIG_avrctrl_8535_8mhz \
                    -c -o main.o main.c
```

Objekt zu ELF Datei linken

```
[avr@host] > avr-gcc -Wl,-Map=ow.map,--cref -mmcu=at90s8535 \
                    -o ow.out main.o -lavrhal-avrctrl-8535-8mhz
```

ELF Datei in Binärdatei, Intel HEX und Motorola S-Record Format überführen

```
[avr@host] > avr-objcopy -O binary -R .eeprom ow.out ow.out-rom.bin
[avr@host] > avr-objcopy -O ihex -R .eeprom ow.out ow.out-rom.hex
[avr@host] > avr-objcopy -O srec -R .eeprom ow.out ow.out-rom.s19
```

Zum Beispiel Motorolas S-Record:

```
S00D00006F772D726F6D2E73313986
S113000010C02AC029C028C027C026C025C024C0CB
S113001023C022C021C020C01FC01EC01DC01CC0E0
S11300201BC011241FBECFE5D2E0DEBFCDBF10E060
S1130030A0E6B0E0E6EAF8E003C0C89531960D9278
S1130040AA3AB107D1F710E0AAEAB0E001C01D92C4
S1130050A43BB107E1F701C0D3CFCDE4D2E0DEBFCA
S1130060CDBF80E9D5D080E690E079D184E036D167
S113007021D2882321F481E790E071D1FFCF8C2F26
S11300809D2F0196CAD2982E482E552442E8242E3C
S113009040E0342E6C2E7D2E0894611C711C31E9D5
S11300A0A32E30E0B32E61E080E02ED167D05F92C2
S11300B04F923F922F92D6D288E893E153D0EE2408
S11300C00F900F900F900F90E91468F761E080E0B3
S11300D01BD154D08E2D9927082F192F000F111FD3
S11300E0000F111F000F111F080F191F060D171DF8
S11300F0F12FE02F808199279F938F93BF92AF9226
S1130100B1D2FF240F900F900F900F90C02ED12EDC
S11301100F5F1F4FF12FE02F81910E2F1F2F992773
S11301209F938F93BF92AF929DD20F900F900F9099
S11301300F90F394F5E0FF1568F7FD2DEC2D878102
S1130140992 79F938F93BF92AF928CD288E893E1C3
S113015009D00F900F900F900F90E394E91408F4D6
S1130160B5CFA1CFE82FF92F309639F0A0EDB7E045
S11301701197F1F7A8953197C9F70895CF93C091D6
S1130180B10080E2D6D08091B1008823D1F780916C
S1130190B3008150682F8C2FB7D0CF91089584B3CA
S11301A08870876084BBA99AA89881E090E016D3F0
S11301B0AA9A81E090E012D39F99F7CFAA9884B3CA
S11301C0876F84BB0895282F85B38870922F907F02
S11301D0892B85BBAA9AAA9885B388702295207F1B
S11301E0822B85BBAA9AAA980895282F95B3987054
S11301F0807F8160982B95BBAA9AAA9885B3887052
S11302002295207F2160822B85BBAA9AAA98089503
S1130210982F8F738093B00080939600805C8093B6
S11302209700892F829586958695837011F0833087
S113023021F483E08093B20005C0991F9927991F88
S11302409093B20084B3876F84BB80E29EE4C6D2ED
S113025085B38870806385BBAA9AAA9888E893E1DD
S1130260BDD285B38870806385BBAA9AAA988EE6AE
S113027090E0B4D285B38870806385BBAA9AAA98AB
S113028082E390E0ABD285B38870806285BBAA9A82
S1130290AA9882E390E0A2D285B38870806285BB7D
S11302A0AA9AAA9885B38870806885BBAA9AAA98E6
S11302B082E390E093D285B3887085BBAA9AAA980A
S11302C085B38870806685BBAA9AAA9882E390E079
S11302D085D28FD22AD087E001D00895CF93992771
S11302E082FF02C0CCE001C0C8E081FF02C0CA6046
S11302F001C0C86080FF02C0C96001C0C8604FDF90
S11303008C2F61DFCF910895282F962F8091B20012
S1130310982 39093B300E4E9F0E0E90FF11D2093F2
S1130320B1008081820F69D2089539DF82E04BDF0A
S113033008951F93182F33DF812F57DF8091B10069
S11303408F5F8093B1009091B000891731F4809150
S1130350B3008F5F682F80E0D7DF1F910895CF939C
S1130360DF93D92FC82F8881882329F08991E1DF71
S1130370888 18823D9F7DF91CF910895949880B329
S1130380992 7282F392F2071307084FD03C0932FB3
```

```
S1130390822F08958C9A80EE91E020D28C988CE381
S11303A090E01CD2849906C084EA91E017D281E0DF
S11303B090E0089584EA91E011D280E090E00895FD
S11303C008959498882341F08C9A86E090E006D2B0
S11303D08C9880E490E007C08C9A8CE390E0FED186
S11303E08C988AE090E0FAD108950895CF93DF9332
S11303F094988C9A86E090E0F1D18C9889E090E012
S1130400EDD180B39927D92FC82FC071D07084FD46
S113041006C087E390E0E2D18C2F9D2F05C087E3CF
S113042090E0DCD181E090E0DF91CF910895FF92DC
S11304300F931F93CF93DF93F82EC0E0D0E001E039
S113044010E0912F802F0C2E02C0880F991F0A9460
S1130450E2F78F21B6DFC00FD11FC830D1057CF37E
S1130460912F802FE5E0CDB7DEB70EC20F931F9317
S1130470CF93DF9380E000E010E0C82FDD27B6DFE4
S1130480882359F081E090E0002E02C0880F991F64
S11304900A94E2F78C2B9D2B02C08C2F9D2F0F5FAB
S11304A01F4F083011054CF39927DF91CF911F910D
S11304B00F91089594988C981092AC005FDF882374
S11304C019F480E090E0089581E090E008950895A3
S11304D0AF92BF92CF92DF92EF92FF920F931F934E
S11304E0CF93082F192F21E0E22ECC24AC2CFE2C24
S11304F0DC2C8091AF00882309F069C03FDF88239A
S113050049F4C092AE00C092AF00C092AD0080E04A
S113051090E06DC080EF8BDF69DFC82F67DFC130EB
S113052019F4813009F440C0C81711F09C2F18C089
S11305308091AD00E81640F4F12FE02FEA0DF11D93
S113054090819F2129F003C090E0E81609F491E01E
S1130550992331F4CE2C88E08E1510F0E092AE0091
S11305602A2D3327CE2DCF5FBF2CBB0C913039F40D
S1130570F12FE02FE20FF31F80818F2907C0F12FA5
S1130580E02FE20FF31FF09480818F218083892F65
S113059018DFEC2EFB2CBB2019F4A39491E0F92E68
S11305A0E7E0EA1508F0B8CFF0E4FE1580F468E05F
S11305B070E0912F802FEDD0882349F4C092AD00D4
S11305C0CC2019F481E08093AF0081E0D82EDD20A7
S11305D029F0F12FE02F8081882339F4DD24D09293
S11305E0AE00D092AF00D092AD008D2D9927CF915F
S11305F01F910F91FF90EF90DF90CF90BF90AF903D
S1130600008951092AE001092AF001092AD0060DF1A
S1130610992708955DDF992708959093AB008093FF
S1130620AA001092AC00EDDF8823C1F08091AC00E9
S11306308F5F8093AC009927282F392F43E0220F36
S1130640331F4A95E1F7280F391F8091AA00909132
S1130650AB00820F931FDEDF882341F78091AC004B
S1130660099270895A0E0B0E0E7E3F3E0EAC04FE0A3
S1130670A42EB12CAC0EBD1EFB2DEA2DC190D19041
S1130680AE2EBF2EFD2DEC2D8191CE2EDF2E853288
S113069059F0882309F47BC04CDEFD2DEC2D8191AB
S11306A0CE2EDF2E8532A9F7FD2DEC2D8191CE2E95
S11306B0DF2E282F332727FD309524363105B9F056
S11306C02536310524F42336310541F00EC0253793
S11306D0310569F02837310591F007C0FB2DEA2D6B
S11306E022E030E0A20EB31E808123DECBCF3AE0BD
S11306F0832E912C20E1E22E27E2F22E05C090E118
S1130700E92EF12C8E2C9F2CFB2DEA2D22E030E0DB
S1130710A20EB31E008111818837ECF4843639F4BB
S113072017FF05C0109501951F4F8DE202DE32E0E0
S1130730E316F10480F00E151F0568F49F2D8E2D2D
S11307406AE070E062D0E62EF72E6230710518F090
```

```
S11307500617170798F3912F802F7F2D6E2D55D0F4
S1130760F72FE62FE856FF4F8081E3DD912F802F8E
S11307707F2D6E2D4AD0082F192F9F2D8E2D792D68
S1130780682D43D0E62EF72E672B29F77BCFECE0BC
S113079074C0F92FE82F40E0615070408FEF6F3F35
S11307A07807C9F05191A8E0252F3327842F992782
S11307B08227932780FF05C088E148274695406833
S11307C001C046955695A15079F7615070408FEF5E
S11307D06F3F780739F7842F99270895E82FF92F69
S11307E0309631F000000000000000003197D1F78E
S11307F00895D5DC81E0E7DC08951F93182FCFDC42
S11308001068812FE0DC1F910895AA1BBB1B51E1E6
S113081007C0AA1FBB1FA617B70710F0A61BB70B6C
S1130820881F991F5A95A9F780959095682F792F5D
S11308308A2F9B2F08952F923F924F925F926F922F
S11308407F928F929F92AF92BF92CF92DF92EF925C
S1130850FF920F931F93CF93DF93CDB7DEB7CA1BDD
S1130860DB0B0FB6F894DEBF0FBECDBF09942A8808
S1130870398848885F846E847D848C849B84AA84B0
S1130880B984C884DF80EE80FD800C811B81AA813D
S1130890B981CE0FD11D0FB6F894DEBF0FBECDBF08
S10908A0CA2FDB2F0895AE
S11308A62D2D3D204156522D4374726C203D2D2D25
S11308B6004552524F523A206F775F696E69742829
S11308C629004F575F444556494345533A2025640A
S11308D60025780000400000303132333435363795
S10D08E6383941424344445460000FE
S9030000FC
```

Optionale Informationen:

```
[avr@host] > avr-objdump -x ow.out > ow.inf
[avr@host] > avr-size -d ow.out > ow.siz
[avr@host] > avr-size -x ow.out >> ow.siz
```

Zum Beispiel Sekmentgrößen:

```
   text        data     bss     dec     hex filename
   2214      74      10    2298     8fa ow.out

   text    data     bss     dec     hex filename
  0x8a6    0x4a     0xa    2298     8fa ow.out
```

7.8 Beispielprojekt: ds1820

Originalboard:
AVR-Ctrl

Dieses Beispiel soll die Benutzung der *Code Vision* Konformität für den Temperatorsensor DS1820 von Dallas Semiconductors aufzeigen. Am vorkonfigurierten OW DQ

Bit ist mindestens ein DS1820 angeschlossen, in unserem Beispiel zwei (siehe Bild). Als Anzeigeeinheit wird ein LCD benutzt, das wie im Beispielprojekt: display angeschlossen und benutzt wird.

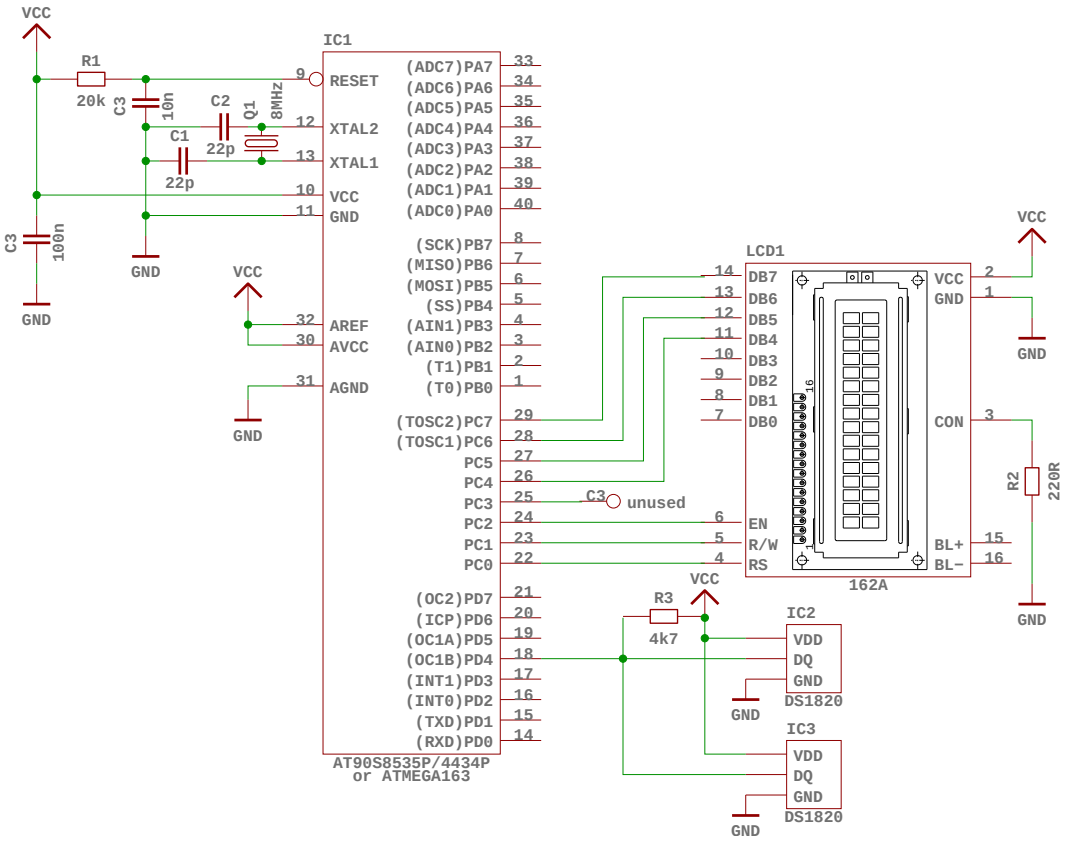


Abbildung 8: Schaltplan angeschlossener Temperatursensoren

7.8.1 Konfiguration der Bibliothek

Die Bibliothek muß intern mit den folgenden Parametern übersetzt werden. Dies kann durch Benutzung der originalen Boardkonfiguration über `--enable-board=avrctrl-8535-8mhz` erreicht werden:

- AVRHAL_LIB_OW_PORT="PORTD”
- AVRHAL_LIB_OW_PORT_DDR="DDRD”
- AVRHAL_LIB_OW_PORT_IN="PIND”
- AVRHAL_LIB_OW_DQBIT="4”

7.8.2 Verwendung der Bibliothek

7.8.2.1 Der Quellcode

main.c

Zunächst werden alle notwendigen Headerdateien der AVR HAL Bibliothek eingebunden:

```
#include <avrhal/delay.h>

#include <avrhal/lcd.h>

#include <avrhal/ow.h>

#include <avrhal/ds1820.h>
```

Eine globale Definition oder Deklaration ist bei diesem Beispiel nicht notwendig. In der Mainroutine wird vor Eintritt in die Endlosschleife das LCD Port sowie der OW Kommunikationsstack initialisiert und vorhandene OW Geräte gesucht. Schlägt das fehl, wird eine entsprechende Fehlermeldung ausgegeben. Im Gegensatz zum Beispielprojekt: ow wird Bei allen Aktionen die Code Vision API benutzt. Daher wird mit `ow_devices[]` auch ein genügend großes Feld vom Typ `unsigned char` und nicht `ow_device_st` angelegt sowie `w1_init()` und `w1_search()` benutzt:

```
int main(void)
{
        unsigned char ow_devices[2 * 9];
        unsigned char devices;
        lcd_init((2 /* rows */ << 6) + 16 /* columns */);
        lcd_putstr("--= AVR-Ctrl =--");
        lcd_putstr("T");
        lcd_control(1, 0, 0);                   // cursor block off
        w1_init();                                                          // NO ERROR HA
        devices = w1_search(0xf0, ow_devices);  // NO ERROR HANDLING !!!
        while (1)
        {
                uint8_t dev;
                for (dev = 0; dev < devices; dev++) {
                        int temp = ds1820_temperature_10(&(ow_devices[dev * 9]));
                        lcd_gotoxy(1,1);
                        lcd_printf("%d = %d.%c %cC", dev + 1, temp / 10,
                                        temp % 10 + 0x30, 0xdf);
                        delay_ms(2000);
                }
        }
} /* main() */
/***********************************EndOfFile*************************
$Id: $
vim:tw=78:ts=4:sw=4:ai:
*/
```

Nach der erfolgreichen Basisinitialisierung wird in der Endlosschleife kontinuierlich die Temperatur aus den gefundenen OW Temperatursensoren ausgelesen und am LCD angezeigt. Zwischen dem Anzeigewechsel wird immer 2 Sekunden gewartet:

Der vollständige Quellcode kann der AVR HAL Bibliothek unter `doc/examples/ds1820` entnommen werden.

Warnung:

Die Anzahl der angeschlossenen Geräte muß mit dem Neuntel der Feldgröße von `ow_devices[]` übereinstimmen. In unserem Beispiel:

```
unsigned char ow_devices[2 * 9];
```

C Quelle zu Objekt compilieren

```
[avr@host] > avr-gcc -g -O2 -Wall -Wstrict-prototypes -Wa \
                     -mmcu=at90s8535 -DAVRHAL_CONFIG_avrctrl_8535_8mhz \
                     -c -o main.o main.c
```

Objekt zu ELF Datei linken

```
[avr@host] > avr-gcc -Wl,-Map=ds1820.map,--cref -mmcu=at90s8535 \
                     -o ds1820.out main.o -lavrhal-avrctrl-8535-8mhz
```

ELF Datei in Binärdatei, Intel HEX und Motorola S-Record Format überführen

```
[avr@host] > avr-objcopy -O binary -R .eeprom ds1820.out ds1820.out-rom.bin
[avr@host] > avr-objcopy -O ihex -R .eeprom ds1820.out ds1820.out-rom.hex
[avr@host] > avr-objcopy -O srec -R .eeprom ds1820.out ds1820.out-rom.s19
```

Zum Beispiel Motorolas S-Record:

```
S011000064733138323 02D726F6D2E733139C6
S113000010C02AC029C028C027C026C025C024C0CB
S113001023C022C021C020C01FC01EC01DC01CC0E0
S11300201BC011241FBECFE5D2E0DEBFCDBF10E060
S1130030A0E6B0E0E8EDFAE003C0C89531960D9271
S1130040A839B107D1F710E0A8E9B0E001C01D92CA
S1130050A23AB107E1F701C0D3CFCDE4D2E0DEBFCD
S1130060CDBF80E9CDD080E690E05BD181E790E020
S113007058D184E02BD100D28C2E9D2E0894811C63
S1130080911C992D882DABD2C82E33E7A32E30E0D6
S1130090B32EDD24DC14E8F70D2D1127912F802FCA
S11300A0880F991F880F991F880F991F800F911F20
S11300B0880D991D3AD0E82EF92E61E0862F1CD1C7
S11300C08FED90E09F938F939F2D8E2D6AE070E0CB
S11300D0B8D4C0969F938F939F2D8E2D6AE070E0C5
S11300E0B0D47F936F930F5F1F4F1F930F93BF92F3
```

```
S11300F0AF929AD280ED97E00CD08DB79EB70A9656
S11301000FB6F8949EBF0FBE8DBFD394DC1420F2BB
S1130110C0CFE82FF92F309639F0A0EDB7E0119752
S1130120F1F7A8953197C9F70895FF920F931F939C
S1130130CF93082F192FC0E0F0909A00CF1500F547
S11301402C2F3327932F822F43E0880F991F4A9532
S1130150E1F7820F931F2091980030919900280FA6
S1130160391F48E050E0732F622F912F802F44D421
S1130170892B19F48C2F1FD305C0CF5FCF1500F343
S113018081EF98EDCF911F910F91FF90089584B363
S11301908870876084BBA99AA89881E090E000D316
S11301A0AA9A81E090E0FCD29F99F7CFAA9884B3F1
S11301B0876F84BB0895282F85B38870922F907F12
S11301C0892B85BBAA9AAA9885B388702295207F2B
S11301D0822B85BBAA9AAA980895282F95B3987064
S11301E0807F8160982B95BBAA9AAA9885B3887062
S11301F02295207F2160822B85BBAA9AAA98089514
S1130200982F8F7380939E0080938400805C8093EA
S11302108500892F829586958695837011F08330A9
S113022021F483E08093A00005C0991F9927991FAA
S11302309093A00084B3876F84BB80E29EE4B0D225
S113024085B38870806385BBAA9AAA9888E893E1ED
S1130250A7D285B38870806385BBAA9AAA988EE6D4
S113026090E09ED285B38870806385BBAA9AAA98D1
S113027082E390E095D285B38870806285BBAA9AA8
S1130280AA9882E390E08CD285B38870806285BBA3
S1130290AA9AAA9885B38870806885BBAA9AAA98F6
S11302A082E390E07DD285B3887085BBAA9AAA9830
S11302B085B38870806685BBAA9AAA9882E390E089
S11302C06FD2C0D22AD087E001D00895CF93992766
S11302D082FF02C0CCE001C0C8E081FF02C0CA6056
S11302E001C0C86080FF02C0C96001C0C8604FDFA0
S11302F08C2F61DFCF910895282F962F8091A00035
S1130300982239093A100E2E8F0E0E90FF11D209317
S11303109F008081820F9AD2089539DF82E04BDFFB
S11303200895CF93DF93D92FC82F8881882329F08C
S1130330899194D288818823D9F7DF91CF91089548
S1130340949880B39927282F392F2071307084FD19
S113035003C0932F822F08958C9A80EE91E020D2CF
S11303608C988CE390E01CD2849906C084EA91E0D6
S113037017D281E090E0089584EA91E011D280E000
S113038090E00895089594988823 41F08C9A86E02B
S113039090E006D28C9880E490E007C08C9A8CE3BD
S11303A090E0FED18C988AE090E0FAD10895089507
S11303B0CF93DF9394988C9A86E090E0F1D18C9857
S11303C089E090E0EDD180B39927D92FC82FC0716F
S11303D0D07084FD06C087E390E0E2D18C2F9D2F7E
S11303E005C087E390E0DCD181E090E0DF91CF911C
S11303F00895FF920F931F93CF93DF93F82EC0E0DD
S1130400D0E001E010E0912F802F0C2E02C0880F65
S1130410991F0A94E2F78F21B6DFC00FD11FC830AD
S1130420D1057CF3912F802FE5E0CDB7DEB745C32E
S11304300F931F93CF93DF9380E000E010E0C82F69
S1130440DD27B6DF882359F081E090E0002E02C05A
S1130450880F991F0A94E2F78C2B9D2B02C08C2FD6
S11304609D2F0F5F1F4F083011054CF39927DF9123
S1130470CF911F910F91089594988C9810929A009F
S11304805FDF882319F480E090E0089581E090E034
S1130490008950895AF92BF92CF92DF92EF92FF92A8
S11304A00F931F93CF93082F192F21E0E22ECC2412
```

```
S11304B0AC2CFE2CDC2C80919D00882309F069C0B3
S11304C03FDF882349F4C0929C00C0929D00C092F3
S11304D09B0080E090E06DC080EF8BDF69DFC82F68
S11304E067DFC13019F4813009F440C0C81711F036
S11304F09C2F18C080919B00E81640F4F12FE02F48
S1130500EA0DF11D90819F2129F003C090E0E816C7
S113051009F491E0992331F4CE2C88E08E1510F083
S1130520E0929C002A2D3327CE2DCF5FBF2CBB0C2D
S1130530913039F4F12FE02FE20FF31F80818F29DE
S113054007C0F12FE02FE20FF31FF09480818F2179
S11305508083892F18DFEC2EFB2CBB2019F4A39485
S113056091E0F92EE7E0EA1508F0B8CFF0E4FE15C3
S113057080F468E070E0912F802FEDD0882349F457
S1130580C0929B00CC2019F481E080939D0081E00F
S1130590D82EDD2029F0F12FE02F8081882339F433
S11305A0DD24D0929C00D0929D00D0929B008D2D92
S11305B09927CF911F910F91FF90EF90DF90CF90EB
S11305C0BF90AF90089510929C0010929D001092DD
S11305D09B0060DF992708955DDF99270895909324
S11305E099008093980010929A00EDDF8823C1F05F
S11305F080919A008F5F80939A009927282F392F32
S113060043E0220F331F4A95E1F7280F391F8091E9
S11306109800909199008 20F931FDEDF882341F7A1
S113062080919A0099270895A0E0B0E0E9E1F3E011
S113063021C24FE0A42EB12CAC0EBD1EFB2DEA2D21
S1130640C190D190AE2EBF2EFD2DEC2D8191CE2EDA
S1130650DF2E853259F0882309F47BC0FFD0FD2DAD
S1130660EC2D8191CE2EDF2E8532A9F7FD2DEC2DB8
S11306708191CE2EDF2E282F332727FD3095243667
S11306803105B9F02536310524F42336310541F01E
S11306900EC02537310569F02837310591F007C0C0
S11306A0FB2DEA2D22E030E0A20EB31E8081D6D0CD
S11306B0CBCF3AE0832E912C20E1E22E27E2F22EDA
S11306C005C090E1E92EF12C8E2C9F2CFB2DEA2DF8
S11306D022E030E0A20EB31E008111818837ECF4D1
S11306E0843639F417FF05C0109501951F4F8DE22C
S11306F0B5D032E0E316F10480F00E151F0568F45E
S11307009F2D8E2D6AE070E086D1E62EF72E6230A2
S1130710710518F00617170798F3912F802F7F2D76
S11307206E2D79D1F72FE62FEA57FF4F808196D0AF
S1130730912F802F7F2D6E2D6ED1082F192F9F2D75
S11307408E2D792D682D67D1E62EF72E672B29F78C
S11307507BCFECE0ABC1F92FE82F40E06150704053
S11307608FEF6F3F7807C9F05191A8E0252F332709
S1130770842F99278227932780FF05C088E1482783
S113078046954068 01C046955695A15079F7615049
S113079070408FEF6F3F780739F7842F99270895BA
S11307A0E82FF92F309631F0000000000000000001F
S11307B03197D1F70895A9E0B0E0E0EEF3E063C12A
S11307C0182FBFD0882371F0612F8EEB8CD0882333
S11307D049F041E069E070E08C2F9D2F019654D0E0
S11307E0882319F481EF98ED2AC019868A81992704
S11307F0382F22278981482F5527240F351F37FF8B
S113080005C081E08987309521953F4F359527951F
S1130810932F822F53E0880F991F5A95E1F7820F87
S1130820931F280F391F40FF02C02B5F3F4F89855C
S1130830882319F0309521953F4F932F822FE3E0C1
S113084029963DC1A4DC81E0B6DC08951F93182FDE
S11308509EDC1068812FAFDC1F9108951F93182F21
S113086096DC812FBADC80919F008F5F80939F007C
```

```
S1130870909 19E00891731F48091A1008F5F682FB9
S113088080E03ADD1F910895EF92FF920F931F933A
S1130890CF93DF93E82EF92ED72FC62F042F10E025
S11308A0672B61F0C5DDFF2DEE2DE10FF11D808377
S11308B01F5F812F99278C179D07A0F341DD8823A3
S11308C049F0002351F06C2F7D2F9F2D8E2D43DF97
S11308D0882319F080E090E002C081E090E0E6E037
S11308E0CDB7DEB7E9C01F93CF93182FC62F8091E1
S11308F09A00882329F0681718F422DD882319F454
S113090080E090E01BC085E574DD8C2F9927282FAB
S1130910392F43E0220F331F4A95E1F7280F391F7F
S113092080919800909199008 20F931F40E068E0B5
S113093070E01DD0812F5DDD81E090E0CF911F91AB
S11309400895CF93DF93682F84E4CDDF882319F4CF
S113095080E090E009C0C0E0D0E03CD0882311F4EE
S1130960C1E0D0E08C2F9D2FDF91CF910895FF92AD
S11309700F931F93CF93DF93082F192FD72FC62FD1
S1130980442371F0972F862F0197682F792F912F89
S1130990802FE1DEF12FE02FEC0FFD1F31978083D4
S11309A0FF24209789F0F12FE02FEF0DF11D8081B6
S11309B020DD882319F480E090E008C0F3948F2DA3
S11309C099278C179D0778F381E090E0E5E0CDB797
S11309D0DEB773C0CF93C4E603C080E197E2E0DEE4
S11309E0E7DC8823D1F38C2FC1508823B1F7CF5F84
S11309F08C2F9927CF910895E62FF72FA82FB92F81
S1130A0004C08D910190801921F441505040C8F7E1
S1130A10881B990B0895AA1BBB1B51E107C0AA1F91
S1130A20BB1FA617B70710F0A61BB70B881F991F8B
S1130A305A95A9F780959095682F792F8A2F9B2F27
S1130A40089597FB092E07260AD077FD04D0E3DF2B
S1130A5006D000201AF4709561957F4F0895F6F73B
S1130A60909581959F4F08952F923F924F925F9258
S1130A706F927F928F929F92AF92BF92CF92DF92AA
S1130A80EF92FF920F931F93CF93DF93CDB7DEB70F
S1130A90CA1BDB0B0FB6F894DEBF0FBECDBF0994A3
S1130AA02A88398848885F846E847D848C849B84FA
S1130AB0AA84B984C884DF80EE80FD800C811B8108
S1130AC0AA81B981CE0FD11D0FB6F894DEBF0FBE37
S10B0AD0CDBFCA2FDB2F0895EE
S1130AD82D2D3D204156522D4374726C203D2D2DF1
S1130AE80054002564203D2025642E2563202563B9
S1130AF84300004000003031323334353637383 95A
S10B0B08414243444546000 04C
S9030000FC
```

Optionale Informationen:

```
[avr@host] > avr-objdump -x ds1820.out > ds1820.inf
[avr@host] > avr-size -d ds1820.out > ds1820.siz
[avr@host] > avr-size -x ds1820.out >> ds1820.siz
```

Zum Beispiel Sekmentgrößen:

```
   text        data     bss     dec     hex filename
   2776      56      10    2842     b1a ds1820.out
```

```
 text    data     bss     dec     hex filename
0xad8    0x38     0xa    2842     b1a ds1820.out
```

7.9 Liste der zu erledigenden Dinge

Gruppe avrhal_delay Ausbau der Unterstützung verschiedener CPU Taktfrequenzen, speziell für delay_us().

Gruppe avrhal_ow_ds1820 Integration der *ALARM* Ereignisbehandlung. Anlenung an CodeVision's unsigned char ds1820_set_alarm(unsigned char *addr, signed char temp_low, signed char temp_high).

Gruppe avrhal_key Überführung der Initialisierung nach Sektion `.init1`

Parametrisierung über mehrere Ports verstreuter Bits.

Unterstützung für Tastaturmatrix.

Wenn machbar, dann eine Ereigniskontrolle (call-back) einführen.

Gruppe avrhal_lcd Überführung von Teilen der Initialisierung nach Sektion `.init1`

Unterstützung eigener frei definierter Grafikzeichen.

Balken- und Punktgrafik ähnlich den LED Funktionen.

Wenn möglich mehr Hardwarebeschleunigung.

Global lcd_init(unsigned char lcd_matrix) Beseitigung der unklaren Nummerierung für die Anzahl Zeichen je Zeile in *lcd_matrix*, oder anders: Was ist bei Anzahl gleich Null(0) ?

Differenzierung in real existierende LCD Typen und somit wegfall der Berechnung von Anfangsadressen im Datenspeicher. Die Adressen werden zum Übersetzungszeitpunkt ermittelt und fest einprogrammiert.

Global lcd_ctrl(unsigned char lcd_control) Es fehlen noch die Kontrolle über die Bewegungsrichtung des Cursors und das Verschieben des Datenspeichers im LCD. Dabei soll die LCD Funktion **Entry Mode Set** verwendet werden.

Gruppe avrhal_led Überführung der Initialisierung nach Sektion .init1

Parametrisierung über mehrere Ports verstreuter Bits.

Gruppe avrhal_ow Überführung von Teilen der Initialisierung nach Sektion .init1

Integration einer One Wire *ALARM* Ereignisbehandlung.

Code-Optimierung bei Eingerätebenutzung (Wegfall der Umfangreichen ROM Code Suche, Wrapper Macros).

Index