avrctrl-lib Nachschlagewerk

20030206

Erzeugt von Doxygen 1.2.18

Thu Feb 6 21:19:29 2003

Inhaltsverzeichnis

1 AVR-Ctrl Lib

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

Das Paket ”AVR-Ctrl Lib” bietet für die spezielle AVR Hardware ”AVR-Ctrl” eine Sammlung wichtiger Standardfunktionen, um mit den vorhandene Ein- und Ausgabeeinheiten schnell und effektiv zu arbeiten. Die Bibliothek wurde gezielt nur für die Benutzung in Verbindung mit dem GNU C Compiler und der avr-libc entwicklet. Es existieren einige API Adaptionen zum CodeVision C Compiler.

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 folgende Liste von Hardwarekomponenten des AVR-Ctrl werden bis jetzt von dieser Bibliothek unterstützt.

AVR CPU:

- at90s8535
- Prozessortakt 8 MHz (fest)

Eingabeeinheiten:

- Standard Tastaturfeld mit 5 Tasten an Port A

Ausgabeeinheiten:

- LED Balken an Port B [1]
- Relais an Port B (indirekt durch LED Balken)
- alphanumerisches LCD an Port C; bis zu 4x64 Zeichen

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

AVR CPU:

- atmega163
- Prozessortakt konfigurierbar [2]

Eingabeeinheiten:

- Infrarotempfänger an Port D
- Temperatursensor an Port D
- Analog-Digital-Wandler an Port A

Ausgabeeinheiten:
- PWM an Port D

Kommunikationsschnittstellen:
- RS232 (UART) an Port D
- Dallas ”One Wire” Bus an Port D [3]
- I2C an Port D (?)
- STDIN, STDOUT, STDERR an RS232 bzw. LCD (?)

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

Zu beachten:
[1] zusätzlich mit normierter Punkt- bzw. Balkengrafik
[2] wichtig für die primitiven ”delay” Funktionen
[3] zusammen mit Temperatursensor und anderen Bausteinen

2 avrctrl-lib Modul-Verzeichnis

2.1 avrctrl-lib Module

Hier folgt die Aufzählung aller Module:

3 avrctrl-lib Seitenindex

3.1 avrctrl-lib Zusätzliche Informationen

Hier folgt eine Liste mit zusammengehörigen Themengebieten:

4 avrctrl-lib Modul-Dokumentation

4.1 Unterbrechungen / Verzögerungen (ungenau)

4.1.1 Ausführliche Beschreibung

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

Diese Headerdatei deklariert ...

Noch zu erledigen:
macros to allow specifying delays directly in microseconds (with MCU clock frequency defined by the user). With constant delays, all floating point math would be done at compile time.

Deklarationen unabhängig vom Systemtakt

- #define delay_9T()
- #define delay_8T()
- #define delay_7T()
- #define delay_6T()
- #define delay_5T()
- #define delay_4T()
- #define delay_3T()
- #define delay_2T()
- #define delay_1T()

Deklarationen mit Abhängigkeit vom Systemtakt

- void delay_ms (unsigned int msec)
- void delay_us (unsigned int usec)

4.1.2 Makro-Dokumentation

4.1.2.1 #define delay_1T()

Wert:

```
asm volatile(   "nop\n\t"                       \
                ::)
```

Unterbricht die Ausführung für einen Intervall von 1 Systemtakt.

4.1.2.2 #define delay_2T()

Wert:

```
asm volatile(   "nop\n\t"                       \
                        "nop\n\t"                       \
                ::)
```

Unterbricht die Ausführung für einen Intervall von 2 Systemtakten.

4.1.2.3 #define delay_3T()

Wert:

```
asm volatile(   "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                ::)
```

Unterbricht die Ausführung für einen Intervall von 3 Systemtakten.

4.1.2.4 #define delay_4T()

Wert:

```
asm volatile(   "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                ::)
```

Unterbricht die Ausführung für einen Intervall von 4 Systemtakten.

4.1.2.5 #define delay_5T()

Wert:

```
asm volatile(   "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                ::)
```

Unterbricht die Ausführung für einen Intervall von 5 Systemtakten.

4.1.2.6 #define delay_6T()

Wert:

```
asm volatile(   "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                ::)
```

Unterbricht die Ausführung für einen Intervall von 6 Systemtakten.

4.1.2.7 #define delay_7T()

Wert:

```
asm volatile(   "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                ::)
```

Unterbricht die Ausführung für einen Intervall von 7 Systemtakten.

4.1.2.8 #define delay_8T()

Wert:

```
asm volatile(   "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                ::)
```

Unterbricht die Ausführung für einen Intervall von 8 Systemtakten.

4.1.2.9 #define delay_9T()

Wert:

```
asm volatile(   "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                        "nop\n\t"                       \
                ::)
```

Unterbricht die Ausführung für einen Intervall von 9 Systemtakten.

4.1.3 Dokumentation der Funktionen

4.1.3.1 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.

4.1.3.2 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.

4.2 Tastaturfeld Unterstützung

4.2.1 Ausführliche Beschreibung

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

Diese Headerdatei deklariert einen einfachen Low-Level Zugang zu den 5 Tasten unterhalb des LCD. 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:
Parametrisierung über mehrere Ports verstreuter Bits.

Low-Level Zugriff

- #define key_init()
- #define key_scancode() (unsigned char)(AVRCTRL_LIB_KEY_PORT_IN & KEY_SCAN_ALL)

Scancodes

- #define KEY_SCAN_T1 _BV(7)
- #define KEY_SCAN_T2 _BV(6)
- #define KEY_SCAN_T3 _BV(5)
- #define KEY_SCAN_T4 _BV(4)
- #define KEY_SCAN_T5 _BV(3)
- #define KEY_SCAN_ALL

4.2.2 Makro-Dokumentation

4.2.2.1 #define key_init()

Wert:

```
AVRCTRL_LIB_KEY_PORT_DDR =                                  \
                (AVRCTRL_LIB_KEY_PORT_DDR & ~KEY_SCAN_ALL) | 0x00;
```

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

Rückgabe:
Das Makro key_init() besitzt keinen Rückgabewert.

4.2.2.2 #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).

4.2.2.3 #define KEY_SCAN_T1 _BV(7)

Bitwert für Taste T1, erste von links.

4.2.2.4 #define KEY_SCAN_T2 _BV(6)

Bitwert für Taste T2, zweite von links.

4.2.2.5 #define KEY_SCAN_T3 _BV(5)

Bitwert für Taste T3, dritte von links.

4.2.2.6 #define KEY_SCAN_T4 _BV(4)

Bitwert für Taste T4, vierte von links.

4.2.2.7 #define KEY_SCAN_T5 _BV(3)

Bitwert für Taste T5, fünfte von links.

4.2.2.8 #define key_scancode() (unsigned char)(AVRCTRL_LIB_KEY_PORT_IN & KEY_SCAN_ALL)

Ließt den aktuellen Zustand der Tasten ein und liefert einen entsprechenden Scancode.

Rückgabe:

Das Makro key_scancode() gibt den aktuellen Scancode als `unsigned char` zurück.

4.3 LCD Unterstützung

4.3.1 Ausführliche Beschreibung

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

Diese Headerdatei deklariert zunächst einen Low-Level Zugang für das LCD. Es werden dabei 1 bis 4 Zeilen Displays mit bis zu 64 Zeichen je Zeile unterstützt (HITACHI LCD). Es wird immer von einer Nibble Kommunikation mit dem LCD ausgegangen. Die Steuersignale RS, RD und EN sind am selben LCD Port frei definierbar. Das eine unbenutzte Bit ist wie die Steuersignale zum Zeitpunkt der Übersetzung parametrisierbar und wird zur Laufzeit berücksichtigt.

Vor der Benutzung der LCD Funktionen, muß dieser Teil der Bibliothek mit lcd_init() initialisiert werden. Hierbei wird der Typ des LCD verschlüsselt übergeben. Dieser Schlüssel, lcd_matrix, hat folgenden Aufbau:

- Bit 5..0:
  - Anzahl der Zeichen je Zeile
  - Werte zwischen 0 und 63, z.B.:
    - 0x10: 16 Zeichen
- Bit 7..6:
  - Anzahl Zeilen
  - Werte zwischen 0 und 3, z.B.:
    - 0: wie 3 (Kompatibilität)
    - 1: 1 Zeile
    - 2: 2 Zeilen
    - 3: 4 Zeilen

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) ?

Low-Level Zugriff / Brajer Vlado Konformität

- #define lcd_control(D, C, B)
- void lcd_init (unsigned char lcd_matrix)
- void lcd_cls (void)
- void lcd_home (void)
- void lcd_goto (unsigned char addr)
- void lcd_ctrl (unsigned char control)
- void lcd_putch (unsigned char data)
- void lcd_print10 (unsigned long x)
- void lcd_print2 (unsigned char x)
- void lcd_print3 (unsigned int x)
- void lcd_print5 (unsigned int x)
- void lcd_printbin (unsigned char x)
- void lcd_printf (const char *format,...)
- void lcd_printhex (unsigned char x)
- void lcd_putstr (unsigned char *data)

CodeVision Konformität

- #define lcd_clear() lcd_cls()
- #define lcd_putchar(C) lcd_putch((unsigned char)(C))
- #define lcd_puts(S) lcd_putstr((unsigned char *)(S))
- #define _lcd_ready() delay_ms(2)
- void lcd_gotoxy (unsigned char x, unsigned char y)
- void lcd_write_byte (unsigned char addr, unsigned char data)
- void _lcd_write_data (unsigned char data)

4.3.2 Makro-Dokumentation

4.3.2.1 #define _lcd_ready() delay_ms(2)

Wartet, bis das LCD zur Kommunikation bereit ist.

Rückgabe:
Das Makro _lcd_ready() besitzt keinen Rückgabewert.

4.3.2.2 #define lcd_clear() lcd_cls()

...

Rückgabe:
Das Makro lcd_clear() besitzt keinen Rückgabewert.

4.3.2.3 #define lcd_control(D, C, B)

Wert:

```
lcd_ctrl((unsigned char)(       \
                                        ( (D) << 2 )    \
                                |       ( (C) << 1 )    \
                                |         (B)           ))
```

...

Parameter:

D gibt mit ungleich Null an, ob das Display aktiv ist.

C gibt mit ungleich Null an, ob der Cursor aktiv ist.

B gibt mit ungleich Null an, ob der Cursor blinkt.

Rückgabe:
Das Makro lcd_control() besitzt keinen Rückgabewert.

4.3.2.4 #define lcd_putchar(C) lcd_putch((unsigned char)(C))

...

Rückgabe:
Das Makro lcd_putchar() besitzt keinen Rückgabewert.

4.3.2.5 #define lcd_puts(S) lcd_putstr((unsigned char *)(S))

...

Rückgabe:
Das Makro lcd_puts() besitzt keinen Rückgabewert.

4.3.3 Dokumentation der Funktionen

4.3.3.1 void _lcd_write_data (unsigned char *data*)

...

... ...

Parameter:
data ist das auszugebende Zeichen.

Rückgabe:
Die Funktion _lcd_write_data() besitzt keinen Rückgabewert.

4.3.3.2 void lcd_cls (void)

...

... ...

Rückgabe:
Die Funktion lcd_cls() besitzt keinen Rückgabewert.

4.3.3.3 void lcd_ctrl (unsigned char *control*)

...

... ...

Bit 0 : cursor blink on(1) / off(0) Bit 1 : cursor on(1) / off(0) Bit 2 : display on(1) / off(0)

Parameter:
control ist die Maske mit den enthaltenen Anweisungen.

Rückgabe:
Die Funktion lcd_ctrl() besitzt keinen Rückgabewert.

4.3.3.4 void lcd_goto (unsigned char *addr*)

...

... ...

Parameter:
addr gibt die Adresse im Datenspeicher des LCD an.

Rückgabe:
Die Funktion lcd_goto() besitzt keinen Rückgabewert.

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

...

... ...

Parameter:
x gibt die X-Koordinate im Datenspeicher des LCD an.

y gibt die Y-Koordinate im Datenspeicher des LCD an.

Rückgabe:
Die Funktion lcd_gotoxy() besitzt keinen Rückgabewert.

4.3.3.6 void lcd_home (void)

...

... ...

Rückgabe:
Die Funktion lcd_home() besitzt keinen Rückgabewert.

4.3.3.7 void lcd_init (unsigned char *lcd_matrix*)

...

... ...

... ...

Parameter:
lcd_matrix definiert den Typ des LCD. Siehe Gruppenbeschreibung.

Rückgabe:
Die Funktion lcd_init() besitzt keinen Rückgabewert.

4.3.3.8 void lcd_print10 (unsigned long *x*)

...

... ...

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_print10() besitzt keinen Rückgabewert.

4.3.3.9 void lcd_print2 (unsigned char *x*)

...

... ...

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_print2() besitzt keinen Rückgabewert.

4.3.3.10 void lcd_print3 (unsigned int *x*)

...

... ...

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_print3() besitzt keinen Rückgabewert.

4.3.3.11 void lcd_print5 (unsigned int *x*)

...

... ...

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_print5() besitzt keinen Rückgabewert.

4.3.3.12 void lcd_printbin (unsigned char *x*)

...

... ...

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_printbin() besitzt keinen Rückgabewert.

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

...

... ...

Parameter:
format ist der Formatstring.

Rückgabe:
Die Funktion lcd_printf() besitzt keinen Rückgabewert.

4.3.3.14 void lcd_printhex (unsigned char *x*)

...

... ...

Parameter:
x ist die auszugebende Zahl.

Rückgabe:
Die Funktion lcd_printhex() besitzt keinen Rückgabewert.

4.3.3.15 void lcd_putch (unsigned char *data*)

...

... ...

Parameter:
data ist das auszugebende Zeichen.

Rückgabe:
Die Funktion lcd_putch() besitzt keinen Rückgabewert.

4.3.3.16 void lcd_putstr (unsigned char * *data*)

...

... ...

Parameter:
data zeigt auf die auszugebende Zeichenkette.

Rückgabe:
Die Funktion lcd_putstr() besitzt keinen Rückgabewert.

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

...

... ...

Parameter:
addr gibt die Adresse im Datenspeicher des LCD an.
data ist das auszugebende Zeichen.

Rückgabe:
Die Funktion lcd_write_byte() besitzt keinen Rückgabewert.

4.4 LED Balken Unterstützung

4.4.1 Ausführliche Beschreibung

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

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

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

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.

Mit der Funktion led_graph_control() wird ein Optionsfeld übergeben, ssen Bits folgende Bedeutung haben:

- Bit 0:
  - rückgesetzt(0): Punktgrafik
  - gesetzt(1): Balkengrafik
- Bit 3..1:
  - Position des Bezugs- bzw. Nullpunkts
  - Werte zwischen 0 und 7, z.B.:
    - Null(000): unterste LED an Bit 0 des LED Ports
    - Sieben(111): oberste LED an Bit 7 des LED Ports
- Bit 7..4:
  - bedeutungslos, nicht interpretiert

Noch zu erledigen:
Parametrisierung über mehrere Ports verstreuter Bits.

Low-Level Zugriff

- #define led_init()
- #define led_cls()
- #define led_clear() led_cls()
- #define led_put(BITSET)
- #define led_get() (unsigned char)(AVRCTRL_LIB_LED_PORT_IN) \

Balkengrafik

- #define LED_BAR (_BV(0))
- #define LED_POINT (~(_BV(0)))
- #define LED_MARGIN(M) (((M) & 0x7) << 1)
- void led_graph_control (int led_maxlevel, signed char led_options)
- void led_graph_put (int bitset_level)

4.4.2 Makro-Dokumentation

4.4.2.1 #define LED_BAR (_BV(0))

Bytewert zum Aktivieren der LED Balkengrafik.

4.4.2.2 #define led_clear() led_cls()

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

Rückgabe:
Das Makro led_clear() besitzt keinen Rückgabewert.

4.4.2.3 #define led_cls()

Wert:

```
{                                                       \
        AVRCTRL_LIB_LED_PORT = AVRCTRL_LIB_LED_OFF;     \
}
```

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

Rückgabe:
Das Makro led_cls() besitzt keinen Rückgabewert.

4.4.2.4 #define led_get() (unsigned char)(AVRCTRL_LIB_LED_PORT_IN) \

Gibt den Wert der aktuell gesetzten LED Ausgabebits zurück.

Rückgabe:
Mit dem Makro led_get() wird das aktuelle Bitmuster vom LED Port gelesen und als `unsigned char` zurückgegeben.

4.4.2.5 #define led_init()

Wert:

```
{                                                       \
        AVRCTRL_LIB_LED_PORT_DDR = 0xff;                \
        led_cls();                                      \
}
```

Initialisiert den LED Port zur Byteausgabe (alle 8 Bits) und setzt alle Bits auf den inaktiven Zustand (OFF value).

Rückgabe:
Das Makro led_init() besitzt keinen Rückgabewert.

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

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

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

Bytewert zum Aktivieren der LED Punktgrafik.

4.4.2.8 #define led_put(BITSET)

Wert:

```
{                                                   \
        AVRCTRL_LIB_LED_PORT = (unsigned char)(BITSET); \
}
```

Setzt alle Bits auf den angegebenen Wert bitset.

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

Rückgabe:
Das Makro led_put() besitzt keinen Rückgabewert.

4.4.3 Dokumentation der Funktionen

4.4.3.1 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.

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 Gruppenbeschreibung.

Rückgabe:

Die Funktion led_graph_control() besitzt keinen Rückgabewert.

4.4.3.2 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 avrctrl-lib Zusätzliche Informationen

5.1 Acknowledgments

This document tries to tie together the labors of a large group of people. Without these individuals' efforts, we wouldn't have a terrific, **free** set of tools to develop AVR projects. We all owe thanks to:

- The GCC Team, which produced a very capable set of development tools for an amazing number of platforms and processors.
- Denis Chertykov [denisc@overta.ru] for making the AVR-specific changes to the GNU tools.

- Denis Chertykov and Marek Michalkiewicz [marekm@linux.org.pl] for developing the standard libraries and startup code for **AVR-GCC**.
- Theodore A. Roth [troth@verinet.com] for setting up avr-libc's CVS repository, bootstrapping the documentation project using doxygen, and continued maintenance of the project on http://savannah.gnu.org/projects/avr-libc.
- All the people at http://www.mikrocontroller.com who have made the interesting AVR-Ctrl board design.
- Uros Platise for developing the AVR programmer tool, **uisp**.
- Joerg Wunsch [joerg@FreeBSD.ORG] for adding all the AVR development tools to the FreeBSD [http://www.freebsd.org] ports tree.
- Brian Dean [bsd@bsdhome.com] for developing **avrprog** (an alternate to **uisp**).
- All the people who have submitted suggestions, patches and bug reports. (See the AUTHORS files of the various tools.)
- And lastly, all the users who use the software. If nobody used the software, we would probably not be very motivated to continue to develop it. Keep those bug reports coming. ;-)

5.2 Frequently Asked Questions

1. Was bedeutet AVR-Ctrl ?
2. Warum gibt es diese Bibliothek ?

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.

5.3 Liste der zu erledigenden Dinge

Gruppe avrctrl_delay macros to allow specifying delays directly in microseconds (with MCU clock frequency defined by the user). With constant delays, all floating point math would be done at compile time.

Gruppe avrctrl_key Parametrisierung über mehrere Ports verstreuter Bits.

Gruppe avrctrl_lcd Beseitigung der unklaren Nummerierung für die Anzahl Zeichen je Zeile in lcd_matrix, oder anders: Was ist bei Anzahl gleich Null(0) ?

Gruppe avrctrl_led Parametrisierung über mehrere Ports verstreuter Bits.

Index