Änderungen 06.02.2012:

Die TIMER1-ISR-Routinen sind in ASM geschrieben.
Die Timer1_CompA_ISR wird in weniger als 40 Takten ausgeführt.
In der Timer1_Capt_ISR war leider noch eine "Handbremse" erforderlich:
Der Timer1 wird während der Ausführung dieser ISR angehalten!

Die Datei "interrupt.S" muss beim Compilieren eingebunden werden (bei AVR-GCC ins makefile eintragen, bei Code:Blocks und WINAVR einfach die Datei ins Projekt einbinden).

Der Prescale-Wert des Timer1 wurde von 64 auf 8 reduziert.
Da aber der CompA-Interrupt auch in ASM immer noch ca. 37 Takte zur Ausführung benötigt, sind alle CompareA-Werte mit dem Faktor 5 multipliziert.
Damit hat die ISR 8 * 5 = 40 Takte Zeit zur Ausführung.

Die PWM-Auflösung konnte auf ca. 12.5 Bit erhöht werden (zählt nun bis 6150), es ergibt sich eine PWM-Frequenz von 81Hz bei 20MHz CPU-Takt
(Gegenrechnung: 40 * 6150 * 81 = 19.926.000Hz.)

Die Anzahl der dimmbaren LED's kann bis zu 8 betragen.
Voreingestellt sind 3 LED's.
Wird eine andere Anzahl benötigt, dann muss ihre Anzahl in der global.h verändert werden.
(Achtung, der Wert 1 als Anzahl ist nicht vorgesehen, weil mit einer Eingabe von genau zwei Werten Einstellungen vorgenommen werden).

Die LED's werden beginnend an den Pins PD7, PD6 ... angeschlossen.
Wenn 8 LED's verwendet werden sollen, dann ist RxD (=PD0) nicht mehr verfügbar.

Ist die Serielle Schnittstelle aktiviert und RxD nicht angeschlossen, dann muss der Pin mit einem Pullup auf VCC gezogen werden (sonst fallen Störungen über die serielle Schnittstelle ein).

Wenn die Steuerung über die serielle Schnittstelle erfolgt, dann dürfen die Kommandos nicht mit einem RETURN abgeschlossen werden.

Sollten die (Power-)LED's trotz der Wiederholfrequenz von 81Hz gelegentlich flackern, dann hilft es möglicherweise, wenn das Netzteil mit einem kräftigen Elko gestützt wird.

Es sind 4 Testmodi eingebaut. Sie werden über den Parameter 1 + (251..254) aufgerufen. Beendet werden sie mit jeder beliebigen Eingabe.
Stellt diese Eingabe eine gültige Eingabe dar, dann wird sie im Anschluss ausgeführt.

Es folgt die ergänzte Beschreibung von 2009:

Der Dimmer ist für LED's gedacht, deren Helligkeit logarithmisch (also der menschlichen Wahrnehmung folgend) gedimmt werden soll.

Dazu werden die gewüschten Helligkeitswerte als 1-Byte je LED an den Dimmer geschickt, der sucht sich aus einer Tabelle einen Compare-Wert aus dem 12.5-Bit Raum (0..6150) und füttert damit den Timer.

Die Daten werden alternativ via TWI oder RS232 entgegengenommen (TWI ist die bessere Wahl, weil hier die richtige Anzahl der übergebenen Parameter geprüft werden kann ! Über die serielle Schnittstelle werden die Daten binär erwartet, ein RETURN kann daher nicht von einem Helligkeitswert 13 unterschieden werden.)

Es können bis zu 8 LEDs bedient werden (wenn die serielle Schnittstelle verwendet werden soll, dann entfällt eine LED am Pin RxD).

Der Controller läuft mit 20MHz, bei einem TOP-Compare-Wert von 6150 und einem Prescale von (8 * 5) ergibt sich eine Wiederholungsrate von 81 Hz
Als Controller kann ein mega48 oder mega8 verwendet werden, der Speicherbedarf

liegt bei 2800 Byte incl. der Demos.

Das Dimmen erfolgt mit Hilfe des Timer1 und des CompareA-IRQ.

Damit das Dimmen für 8 PortPins funktioniert, muss einiges an Vorarbeit geleistet werden:

Die Schaltvorgänge werden in zeitlicher Abfolge der Schaltvorgänge sortieren und der jeweils erforderliche LED-Portstatus wird berechnet.

Anschließend werden die vorverdauten Daten im Array pwm_buffer[] abgelegt.

Das Array besteht gedanklich aus zwei Hälften, der unteren und der oberen. Während die aktuellen Werte z.B. in der unteren Hälfte stehen und permanent von der CompareA_ISR ausgelesen und in die Register OCR1AH, OCR1AL und den Port D geschrieben werden, legt das Programm die nächsten Werte in der oberen Hälfte ab und setzt das Flag pwm_update.
Die Timer1_Capt_ISR, die am Ende des PWM-Taktes aufgerufen wird, erkennt das gesetzte Flag pwm_update (löscht es) und liest von nun an die Daten aus der oberen Hälfte des Arrays.
Das Programm wird anschließend die nächsten Daten im unteren Bereich von pwm_buffer[] bereitstellen ...

Die Interrupt-Routine muss knapp gehalten werden, da sie innerhalb von 40 Takten einen neuen Compare-Wert und das Portbyte schreiben muss.
Reicht die Zeit nicht aus, wird ein Schaltvorgang, der direkt beim nächsten PWM-Wert ansteht, verloren gehen - und dann flackern die LED's heftig oder gehen auf Dauerlicht.

Neben dem direkten Setzen von Helligkeitswerten ist ein fading eingebaut:

zwischen den zuletzt verwendeten Helligkeitswerten und den neu angegebenen Werten wird ein linearer Übergang erzeugt.

Die Anzahl der Zwischenschritte kann von 2 bis 255 gewählt werden.

Es werden 81 Übergänge pro Sekunde erzeugt, bei einer Schrittzahl von 255 dauert der Übergang also ca. 3 Sekunden.

Soll der Übergang noch geschmeidiger werden, dann kann noch eine zusätzliche Verzögerungen zwischen den Werteänderungen eingebaut werden (es wird dann nur nach jedem x.ten PWM-Neustart ein neuer Helligkeitswert zugewiesen).

Um Werte an den Dimmer zu senden, ist folgende Sequenz zu übergeben (unter der Annahme, dass 3 LEDs definiert sind):

```
TWI:        TWI_ADR, Parameter, LED0, LED1, LED2
            2, 0, 255, 128, 192

            Die Daten werden ohne Komma in binärer Form erwartet.
```

```
Seriell:    Parameter, LED0, LED1, LED2
            0, 255, 128, 192

            Auch hier werden die Daten ohne Komma in binärer Form erwartet.
            Es darf kein abschließendes RETURN gesendet werden, die '13' würde
            als Helligkeitswert interpretiert werden.
            Ausserdem würde die Anzahl der empfangenen Bytes um 1 erhöht,
            das Programm erkennt den Eingang eines Datensatzes aber an der
            Anzahl der empfangenen Bytes.
            Die Verwendung der TWI-Schnittstelle ist die deutlich bessere Wahl.
```

Der vorangestellte Parameter hat folgende Bedeutung:

0 die Helligkeitsdaten werden direkt übernommen (kein fading)

1 Ist der Parameter 1, dann hat das nachfolgende, 2. Byte abhängig von seinem Wert unterschiedliche Funktion:

0 .. 250

definiert eine Verzögerung beim fading und beim der Einstellung neuer Werte (Standard ist 0),
der Wert 1 legt einen, der Wert 2 legt zwei ... zusätzliche PWM-Takte als Wartezeit nach jeder Änderung ein.

251 ruft Demo1 auf
252 ruft Demo2 auf
253 ruft Demo3 auf
254 ruft Demo4 auf

255 beendet eine Demos ohne sonstige Nebenwirkungen

2-255 gibt die Anzahl der Zwischenschritte an beim Übergang vom letzten Helligkeitswert zum neu angegebenen Wert an

Während ein Kommando ausgeführt wird (beim fading kann das ja einige Sekunden dauern) kann bereits der nächste Datensatz gesendet werden.

Während eines Farbüberganges kann der Fortschritt über die TWI-Schnittstelle verfolgt werdem: beim Auslesen wird die Anzahl der noch anstehenden Übergänge zurückgeliefert.

Die Rückgabe 0 bedeutet, dass der Übergang abgeschlossen ist.

Zum Thema Soft-PWM und LED-Fading gibt es in der Artikelsammlung einige Beiträge, die u.a. Grundlage für diese Programm waren.

Die Berechnung der CompareA-Werte-Tabelle wurde nach den Angaben aus dem Beitrag "LED-Fading" durchgeführt.

Michael S.