ATtiny2313RCLichtsteuerungStandard

2013-09-02

Erzeugt von Doxygen 1.8.7

Sam Jun 14 2014 17:28:47

Inhaltsverzeichnis

Kapitel 1

Ausstehende Aufgaben

Datei attiny2313_rc_lichtsteuerung_standard.c

- Umschalten der Zustände aus der Hauptschleife entfernen und in eine eigene Funktion packen -> diese Funktion wird nur ca. alle 17 ms benötigt (nach Erkennung eines Eingangsimpulses oder nach Wachtdog Überlauf) und muß nicht in der Hauptschleife andauernd überprüft werden -> uC länger im Schlafmodus? -> weniger Stromverbrauch
- Volatile -> ISR Otimierung bei Zugriff auf blktm_oflow in blink_timer.c -> siehe mikrocontroller.net
- Umschalten von Ein- / Aus und umgekehrt erst nach mehreren gleichen Eingangsimpuls längen
- alle nicht benötigten uC Teile abschalten
- Abspeichern der Einstellungen (Schaltpunkte, Blinkfolgen, etc.) im EEPROM -> weniger Speicherverbrauch im Flash, nicht hardcoded -> dann möglich umstellen über Jeti Protokoll
- Ausgabe des Schaltzustandes über das Jeti Protokoll
- Ändern der Schaltzeitpunkte, Blinkfrequenz, etc. über das Jeti Protokoll
- Messen von PWM Tastverhältnisse siehe Atmel App Note

Kapitel 2

Datei-Verzeichnis

2.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

Kapitel 3

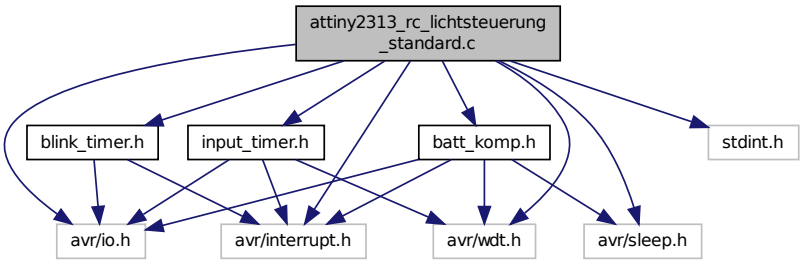
Datei-Dokumentation

3.1 attiny2313_rc_lichtsteuerung_standard.c-Dateireferenz

Hauptfile des Projekts ATtiny2313 RC Lichtsteuerung Standard.

```
#include <avr/io.h>
#include <stdint.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include "input_timer.h"
#include "blink_timer.h"
#include "batt_komp.h"
```

Include-Abhängigkeitsdiagramm für attiny2313_rc_lichtsteuerung_standard.c:



Makrodefinitionen

- #define F_CPU 1000000UL

Funktionen

- int main ()

  Hauptfunktion.
- ISR (WDT_OVERFLOW_vect)

  Interrupt Service Routine für Watchdog Timer.

Variablen

- volatile uint8_t in0_puls_len = 0

  *Zählerstand vom Input Timer (Timer0) = Pulslänge des Eingangs 1 -> Zählerstand * 64us = Pulslänge in ms.*
- volatile uint8_t blktm_oflow = 0

  Anzahl der Überläufe vom Blink Timer (Timer 1) -> Ein Überlauf entspricht 0,065s.
- volatile uint8_t batt_full = 1

  Batteriestatus -> solange = 1 wird die Hauptschleife ausgeführt, wenn = 0 wird alles abgeschalten und der uC in den Schlafmodus versetzt.
- volatile uint8_t no_sig = 0

  Status Empfänger Eingangssignal -> 0 = Signal vorhanden, 1 = kein Signal.

3.1.1 Ausführliche Beschreibung

Hauptfile des Projekts ATtiny2313 RC Lichtsteuerung Standard.

Autor

V. Pippan (webmaster@vpippan.at)

Datum

2013-08-30

Version

20130830

Noch zu erledigen

- Umschalten der Zustände aus der Hauptschleife entfernen und in eine eigene Funktion packen -> diese Funktion wird nur ca. alle 17 ms benötigt (nach Erkennung eines Eingangsimpulses oder nach Wachtdog Überlauf) und muß nicht in der Hauptschleife andauernd überprüft werden -> uC länger im Schlafmodus? -> weniger Stromverbrauch
- Volatile -> ISR Otimierung bei Zugriff auf blktm_oflow in blink_timer.c -> siehe mikrocontroller.net
- Umschalten von Ein- / Aus und umgekehrt erst nach mehreren gleichen Eingangsimpuls längen
- alle nicht benötigten uC Teile abschalten

- Abspeichern der Einstellungen (Schaltpunkte, Blinkfolgen, etc.) im EEPROM -> weniger Speicherverbrauch im Flash, nicht hardcoded -> dann möglich umstellen über Jeti Protokoll
- Ausgabe des Schaltzustandes über das Jeti Protokoll
- Ändern der Schaltzeitpunkte, Blinkfrequenz, etc. über das Jeti Protokoll
- Messen von PWM Tastverhältnisse siehe Atmel App Note

3.1.2 Makro-Dokumentation

3.1.2.1 #define F_CPU 1000000UL

3.1.3 Dokumentation der Funktionen

3.1.3.1 ISR (WDT_OVERFLOW_vect)

Interrupt Service Routine für Watchdog Timer.

Ein Überlauf des Watchdog Timer löst diesen Interrupt aus. Wenn für längere Zeit kein Empfänger Eingangssignal anliegt wird der Watchdog Timer nicht zurückgesetzt. Dies führt zu einem Überlauf des Watchdog Timers und innerhalb dieser Routine wird die Variable no_sig auf 1 gesetzt. Dies führt zum Abschalten der Beleuchtung, falls diese vorher eingeschalten war.

3.1.3.2 int main ( )

Hauptfunktion.

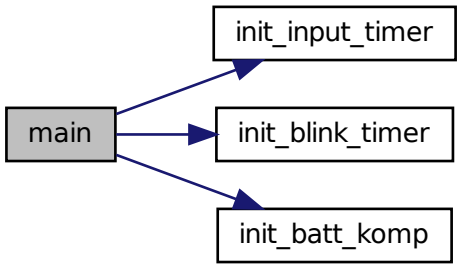
Die Hauptfunktion enthält die Initialisierungen beim Programmstart und die Hauptschleife des Programms. Wird die Hauptschleife abgebrochen (= Batterie leer) werden die Lichter ausgeschalten, alle Interrupts deaktiviert, die Timer gestoppt und der uC in den Power Down Schlafmodus versetzt. Aus diesem Zustand kann der uC nur durch einen Reset (Anstecken eines geladenen Akkus) wieder aktiviert werden! < Zählerstand bei dem Navigationslichter einschalten sollen (bei meinem Servotester -> 12, normal 21)

< Zählerstand bei dem Landelichter einschalten sollen (bei meinem Servotester -> 19, normal 26)

Hauptschleife

Die Hauptschleife wird ausgeführt solange die Variable batt_full = 1 ist. batt_full = 0 bedeutet einen leeren Akku und die Hauptschleife wird beendet.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



3.1.4 Variablen-Dokumentation

3.1.4.1 volatile uint8_t batt_full = 1

Batteriestatus -> solange = 1 wird die Hauptschleife ausgeführt, wenn = 0 wird alles abgeschalten und der uC in den Schlafmodus versetzt.

3.1.4.2 volatile uint8_t blktm_oflow = 0

Anzahl der Überläufe vom Blink Timer (Timer 1) -> Ein Überlauf entspricht 0,065s.

3.1.4.3 volatile uint8_t in0_puls_len = 0

Zählerstand vom Input Timer (Timer0) = Pulslänge des Eingangs 1 -> Zählerstand ∗ 64us = Pulslänge in ms.

3.1.4.4 volatile uint8_t no_sig = 0

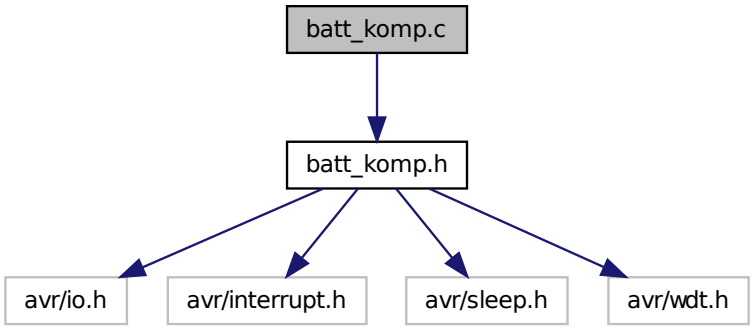
Status Empfänger Eingangssignal -> 0 = Signal vorhanden, 1 = kein Signal.

3.2 batt_komp.c-Dateireferenz

Komparator zur Batterie Spannungsüberwachung.

```
#include "batt_komp.h"
```

Include-Abhängigkeitsdiagramm für batt_komp.c:



Funktionen

- void init_batt_komp ()

  Komparator zur Akkuüberwachung einstellen.

- ISR (ANA_COMP_vect)

  Interrupt Service Routine für den Analog Komparator.

3.2.1 Ausführliche Beschreibung

Komparator zur Batterie Spannungsüberwachung.

Autor

V. Pippan (webmaster@vpippan.at)

Datum

2013-08-30

Version

20130830

Als Vergleichspannung für den Komparator wird die interne Referenzspannung des uC verwendet. Diese beträgt typ. 1,1V und schwankt über die Temperatur.
Die hohe Akkuspannung wird über einen Spannungsteiler so geteilt, daß ein Vergleich mit der Referenzspannung möglich wird. Allerdings ändert sich diese heruntergeteilte Spannung, aufgrund des Teilungsfaktors, nur mehr in einem kleinen Bereich.
Durch eine Berechnung mit Hilfe der Fehlerfortpflanzung (siehe Maxima Datei) habe ich sichergestellt, daß trotz störender Einflüsse (Widerstandstoleranzen, Temperaturänderung) der Akku sicher als leer erkannt wird. Ein Berechnungsbeispiel findet sich im Tabellendokument welches auch die Bauteilliste etc. enthält.

3.2.2 Dokumentation der Funktionen

3.2.2.1 void init_batt_komp ( )

Komparator zur Akkuüberwachung einstellen.

- Digital Input Buffer für Pins AIN0 und AIN1 deaktivieren -> spart Strom
- Interrupt an positiver Komparator Flanke auslösen, interne Referenzspannung an den Komparator Eingang AIN0 schalten, Komparator Interrupt aktivieren

3.2.2.2 ISR (ANA_COMP_vect)

Interrupt Service Routine für den Analog Komparator.

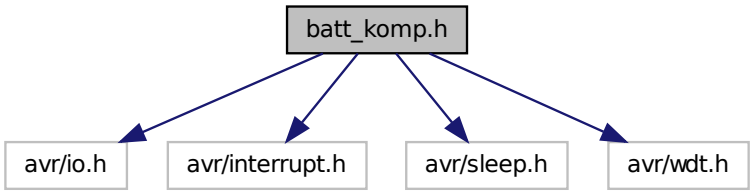
Wenn die Akkuspannung unter einen vorgegebenen Wert fällt, erzeugt der Analoge Komparator eine positive Flanke an seinem Ausgang und dieser Interrupt wird ausgeführt. Innerhalb dieser Routine wird batt_full = 0 gesetzt und somit die Hauptschleife des Programms beendet, alles ausgeschalten und der uC in den Schlafmodus versetzt.

3.3 batt_komp.h-Dateireferenz

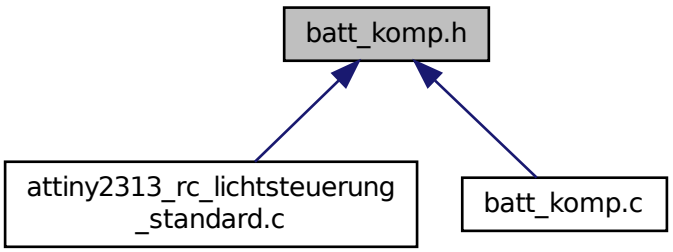
Include File für batt_komp.c.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
```

Include-Abhängigkeitsdiagramm für batt_komp.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Funktionen

- void init_batt_komp ()

  Komparator zur Akkuüberwachung einstellen.

Variablen

- volatile uint8_t batt_full

  Batteriestatus -> solange = 1 wird die Hauptschleife ausgeführt, wenn = 0 wird alles abgeschalten und der uC in den Schlafmodus versetzt.

3.3.1 Ausführliche Beschreibung

Include File für batt_komp.c.

Autor

V. Pippan (webmaster@vpippan.at)

Datum

2013-08-30

Version

20130830

3.3.2 Dokumentation der Funktionen

3.3.2.1 void init_batt_komp ( )

Komparator zur Akkuüberwachung einstellen.

- Digital Input Buffer für Pins AIN0 und AIN1 deaktivieren -> spart Strom
- Interrupt an positiver Komparator Flanke auslösen, interne Referenzspannung an den Komparator Eingang AIN0 schalten, Komparator Interrupt aktivieren

3.3.3 Variablen-Dokumentation

3.3.3.1 volatile uint8_t batt_full

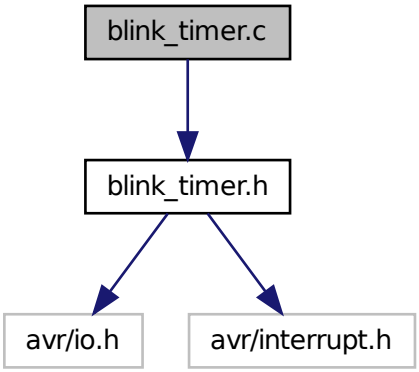
Batteriestatus -> solange = 1 wird die Hauptschleife ausgeführt, wenn = 0 wird alles abgeschalten und der uC in den Schlafmodus versetzt.

3.4 blink_timer.c-Dateireferenz

Timing für das blinken der Lichter.

```
#include "blink_timer.h"
```

Include-Abhängigkeitsdiagramm für blink_timer.c:



Funktionen

- void init_blink_timer ()
- ISR (TIMER1_OVF_vect)

  Interrupt Service Routine für Timer 1 (Blink Timer) Überlauf.

3.4.1 Ausführliche Beschreibung

Timing für das blinken der Lichter.

Autor

V. Pippan (webmaster@vpippan.at)

Datum

2013-09-02

Version

20130902

Der 16 Bit Timer wird zum erzeugen der Blinksignale benutzt. Bei einer CPU Frequenz von 1 MHz und ohne Vorteiler benötigt der Timer 0,065 s ((1/f_cpu) $*$ 2^16) bis zum Überlauf. Beim Timer Überlauf wird ein Interrupt ausgelöst und die Variable blktm_oflow um 1 erhöht. Diese Variable bestimmt somit die verstrichene Zeit (Anzahl Timerüberläufe $*$ 0.065 s). Daraus wird dann der aktuelle Status innerhalb der Blinkfolge abgeleitet und die Lichter entsprechend ein oder ausgeschalten.
Durch das zählen der Timerüberläufe ist die kürzest mögliche Zeitperiode = 0,065 s. Dies bedeutet, eine Pause oder die Zeit vom Ein- bis zum Ausschalten (blitzen) eines Ausganges kann nicht kürzer als 0,065 s sein. Für eine originalgetreue Darstellung eines Positionsblitzes sollte dies aber ausreichend kurz sein.

Erklärung zum erstellen eigener Blinkfolgen:

Am besten zeichnet man sich die gewünschte Blinkfolge auf ein Blatt Papier, beginnend mit der Pause nach Ende der vorherigen Blinkfolge. Die Folge unterteilt man dann in 0,065 s lange Abschnitte. Je mehr Abschnitte, desto länger dauert die Folge (desto länger sind die Lichter eingeschalten). Die Abschnitte werden durchnummeriert und anhand der vorher gezeichneten Folge sieht man, ob beim entsprechenden Abschnitt (= Zustand) ein Licht ein oder auszuschalten ist.
Nun braucht man nur mehr bei den case Anweisungen in der ISR die Nummer des Abschnitts und den Befehl zum schalten des Lichtes eintragen. Beim Ausschalten des letzten Lichtes (letzter Zustand = nullter Zustand) noch darauf achten, die Variable blktm_oflow = 0 zu setzen. Beim nächsten Timerüberlauf sind dann wieder 0,065 Sekunden vergangen, wir befinden uns im Zustand 1 und die Blinkfolge beginnt von vorne. Zur Verdeutlichung ein Bild meiner Blinkfolge:

3.4.2 Dokumentation der Funktionen

3.4.2.1 void init_blink_timer ( )

Überlauf Interrupt für Timer 1 (Blink Timer) einschalten

3.4.2.2 ISR (TIMER1_OVF_vect)

Interrupt Service Routine für Timer 1 (Blink Timer) Überlauf.

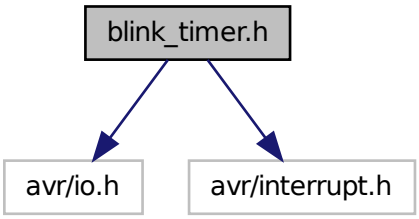
Dieser Interrupt wird bei jedem Überlauf von Timer 1 (Blink Timer) aufgerufen. Innerhalb der Routine wird die Variable blktm_oflow um eins erhöht. Durch die switch Anweisung wird der aktuelle Wert von blktm_oflow überprüft und entsprechend die Lichter ein oder ausgeschalten. Dadurch entsteht die Blinksequenz.

3.5 blink_timer.h-Dateireferenz

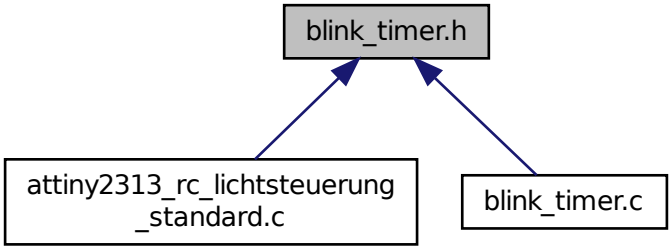
Include File für blink_timer.c.

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

Include-Abhängigkeitsdiagramm für blink_timer.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Funktionen

- void init_blink_timer ()

Variablen

- volatile uint8_t blktm_oflow

  Anzahl der Überläufe vom Blink Timer (Timer 1) -> Ein Überlauf entspricht 0,065s.

3.5.1 Ausführliche Beschreibung

Include File für blink_timer.c.

Autor

V. Pippan (webmaster@vpippan.at)

Datum

2013-09-02

Version

20130902

3.5.2 Dokumentation der Funktionen

3.5.2.1 void init_blink_timer ( )

Überlauf Interrupt für Timer 1 (Blink Timer) einschalten

3.5.3 Variablen-Dokumentation

3.5.3.1 volatile uint8_t blktm_oflow

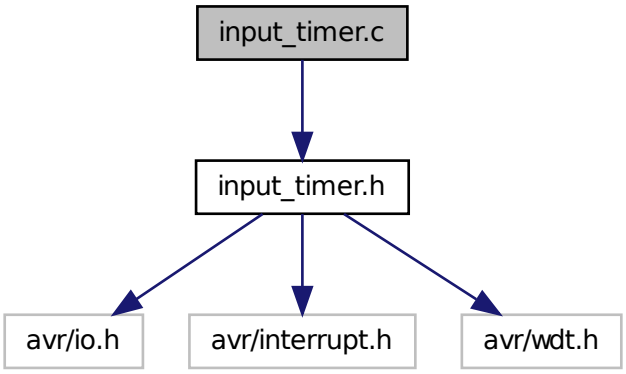
Anzahl der Überläufe vom Blink Timer (Timer 1) -> Ein Überlauf entspricht 0,065s.

3.6 input_timer.c-Dateireferenz

Erfassung der Impulslänge des Empfängersignals mit dem Timer0.

```
#include "input_timer.h"
```

Include-Abhängigkeitsdiagramm für input_timer.c:



Funktionen

- void init_input_timer ()

  Timer0 zur Messung des Empfängersignals einstellen.
- ISR (INT0_vect)

  Interrupt Service Routine für externen Interrupt INT0.

3.6.1 Ausführliche Beschreibung

Erfassung der Impulslänge des Empfängersignals mit dem Timer0.

Autor

V. Pippan (webmaster@vpippan.at)

Datum

2013-09-02

Version

20130902

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with ATtiny2313 RC Lichtsteuerung Standard. If not, see
www.gnu.org/licenses.

Das RC Eingangssignal hat eine Periodendauer von ca. 20ms -> f = 50Hz. Die Länge des positiven Pulses schwankt zwischen minimal 0,75ms und maximal 2,25ms, die Standard Länge liegt zwischen minimal 1ms und maximal 2ms, wobei die Neutralstellung bei 1,5ms liegt.

Der uC läuft mit 1MHz = Periodendauer von 1us. Mit f_CPU / 64 erhält man eine Periodendauer von 64us für den Timer Takt. D.h. eine Erhöhung des Timer Zählerstandes um 1 bedeutet eine Verlängerung des RC-Empfänger Pulses um 64us. Der 8bit Timer zählt bis 2^8 (256) -> Eine Periodendauer von 64us * 256 ergibt 16,38ms bis der Timerüberlauf erreicht wird.
Mit dieser Timergeschwindigkeit kann man ohne Timerüberlauf bis zur max. Pulsdauer des RC-Empfängersignals (2,25ms) zählen. Bei minimaler Pulsdauer ergibt sich damit ein Zählerstand von 12, bei max. Pulsdauer ein Zählerstand von 35 (bei Standard Länge der Pulse min. 16 und max. 31), die Neutralstellung liegt bei einem Zählerstand von 23. Das Eingangssignal kann dabei auf 64us genau, dies entspricht 23 (15 bei Standard Pulslängen) Schritten, aufgelöst werden. Was für diesen Zweck hier genau genug sein sollte. Anmerkung: Durch Verwendung des 16bit Timers kann das RC Eingangssignal wesentlich feiner aufgelöst werden, was für eine einfache Schaltanwendung wie hier aber nicht notwendig ist.

Umrechnung von gegebenem Schaltzeitpukt auf den Zählerstand:

RC-Empfängersignal Schaltzeitpunkt (ms) / (64us * 10^-3) = Zählerstand bei dem geschalten wird

Erkennung eines nicht vorhandenen RC-Empfängersignals mit dem Watchdog Timer:

Der Watchdog Timer wird nach jedem korrekt erkanntem Eingangsimpuls (negative Flanke) zurückgesetzt. Wenn nach einer negativen Flanke kein weiterer Eingangsimpuls (keine positive Flanke) mehr kommt, dann verbleibt das Programm einfach so lange im aktuellen Zustand, bis der Watchdog anspricht und das Programm in den Standard Zustand (gesamte Beleuchtung aus) versetzt.

Der Fall, daß das Eingangssignal nach der positiven Flanke unterbrochen wird (z.B.: durch abziehen des Steckers, ausschalten der Empfängerstromversorgung) spielt hier keine Rolle. Sobald am Optokoppler nichts mehr anliegt, zieht der Widerstand R16 den Pin auf Masse, was eine fallende Flanke erzeugt. Somit entsteht auch in diesem Fall ein korrekt erkannter (wenn auch zu kurzer) Eingangsimpuls.

3.6.2 Dokumentation der Funktionen

3.6.2.1 void init_input_timer ( )

Timer0 zur Messung des Empfängersignals einstellen.

- Einstellen von Pin 6 (PD2) als Quelle für externen Interrupt
- externen Interrupt aktivieren und konfigurieren

3.6.2.2 ISR (INT0_vect)

Interrupt Service Routine für externen Interrupt INT0.

Diese ISR dient der Erfassung der Pulslänge vom RC-Empfängersignal. Sie wird bei jeder Flanke des Eingangssignals aufgerufen. Das umschalten auf welche Flanke reagiert werden soll erfolgt dabei innerhalb dieser Routine.

Nach einer positiven Flanke (Eingang ist HIGH):

- Timer 0 (Input Timer) starten
- Umschalten auf Interrupt Auslösung nach negativer Flanke

Nach einer negativen Flanke (Eingang ist LOW):

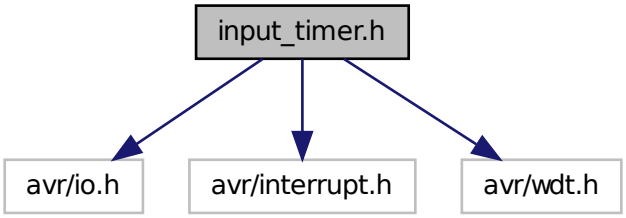
- Timer 0 (Input Timer) stoppen
- Auslesen des Timer Wertes
- Timer Zählerstand auf 0 zurücksetzen
- Umschalten auf Interrupt Auslösung nach positiver Flanke
- Watchdog Timer zurücksetzen

3.7 input_timer.h-Dateireferenz

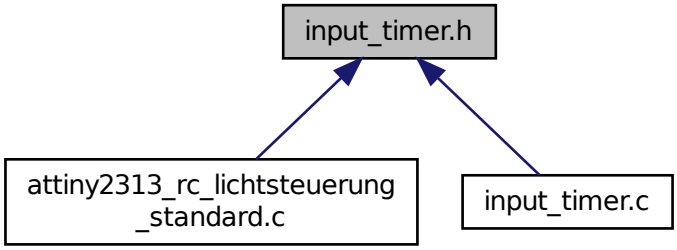
Include File für input_timer.c.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>
```

Include-Abhängigkeitsdiagramm für input_timer.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Funktionen

- void init_input_timer ()

  Timer0 zur Messung des Empfängersignals einstellen.

Variablen

- volatile uint8_t in0_puls_len

  Zählerstand vom Input Timer (Timer0) = Pulslänge des Eingangs 1 -> Zählerstand ∗ 64us = Pulslänge in ms.

3.7.1 Ausführliche Beschreibung

Include File für input_timer.c.

Autor

V. Pippan (webmaster@vpippan.at)

Datum

2013-06-26

Version

20130626

3.7.2 Dokumentation der Funktionen

3.7.2.1 void init_input_timer ( )

Timer0 zur Messung des Empfängersignals einstellen.

- Einstellen von Pin 6 (PD2) als Quelle für externen Interrupt
- externen Interrupt aktivieren und konfigurieren

3.7.3 Variablen-Dokumentation

3.7.3.1 volatile uint8_t in0_puls_len

Zählerstand vom Input Timer (Timer0) = Pulslänge des Eingangs 1 -> Zählerstand ∗ 64us = Pulslänge in ms.