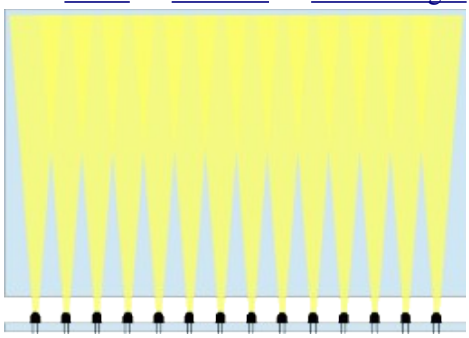


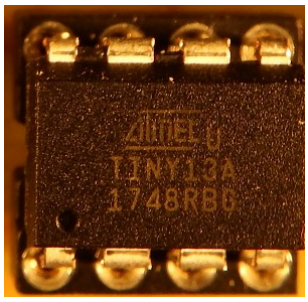
AVR-Anwendungen

Puppenhaus-Beleuchtung mit ATtiny13

Hardware, Aufbau und Software für die Puppenhaus-Beleuchtung

Puppenhaus-Beleuchtung ATtiny13

1 Die Einfachvariante ohne Mikrocontroller



Man kann es sich auch einfach machen: nur LEDs, ohne Schnörkel und Getue. Hier gibt es einen solchen Einfachregler ohne Prozessor.

Viel schöner ist es allerdings, die Regelung mit einem Mikrocontroller zu realisieren, wie die nachfolgenden Kapitel zeigen.

2 Eigenschaften der Puppenhaus-Beleuchtung mit ATtiny13

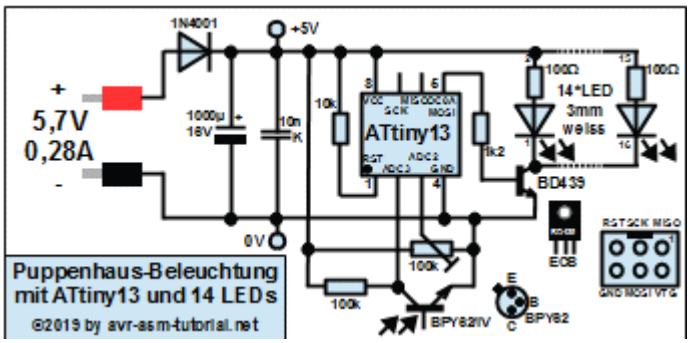
Die Hardware-Eigenschaften der beschriebenen Beleuchtung:

- Steuert 14 Leuchtdioden mit 256 stufigem Hellingkeitswert für die LEDs an.
- Einstellbare Helligkeit mit (Trimm-)Potentiometer.
- Misst die Hintergrundhelligkeit und passt die Lichtstärke entsprechend an.
- Betrieb aus einem vorhandenen 5,7 V-Netzteil oder aus einem eigenen Trafonetzteil.
- Gut dokumentierter freier Assembler-Quellcode, leicht modifizierbar für andere Taktfrequenzen und anpassbar an andere Prozessortypen.

Seitenanfang	Eigenschaften	Hardware	Aufbau	Software

3 Hardware

3.1 Prozessorteil



Dies ist der Prozessorteil des Schaltbilds. Es zeigt den Controller ATtiny13 im 8-poligen PDIP-Gehäuse.

Die folgenden Ports des ATtiny13 erfüllen folgende Aufgaben:

1. Der OC0A-Pin steuert über einen NPN-Transistortreiber die LEDs mit einem pulsweiten-modulierten Signal mit einer Frequenz von

etwa 73 Hz an.
2. Am Pin 2 (ADC3) erfolgt die Messung der Hintergrundbeleuchtung mittels AD-Wandler. Ein Fototransistor nimmt die Lichtstärke auf und wandelt sie in eine Spannung um, die vom AD-Wandler laufend gemessen wird.
3. Am Pin 3 (ADC2) ist ein Trimmer oder ein Potentiometer mit 100k angeschlossen. Damit erfolgt die manuelle Helligkeitsregelung.
4. Die Stromversorgung des ATtiny13 erfolgt an Pin 8 (+5V) und und 4 (0V) und ist mit einem 10 nF-Keramikkondensator gegen Stromspitzen im Prozessor geblockt.
5. Der Reset-Pin liegt mit 10 k an +5V.

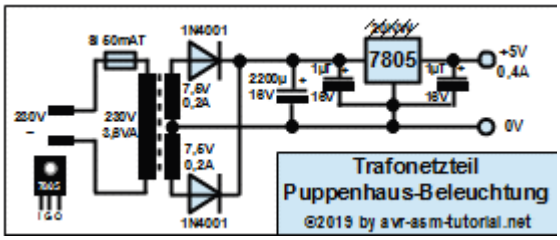
In der Schaltung ist noch ein ISP6-Wannenstecker eingearbeitet, mit dem man den Prozessor in der Schaltung programmieren kann.

Es werden 14 LEDs mit dem PWM-Signal ausgesteuert. Die LEDs sind auf einen Strom von 20 mA im eingeschalteten Zustand ausgelegt, wofür die 14 Widerstände mit 100Ω dienen. Sollen andere LEDs verwendet werden, müssen diese Widerstände entsprechend angepasst werden.

Sollen die LEDs mit ihren nominell zulässigen 50 mA betrieben werden, müssen Widerstände mit 33Ω verwendet werden. Das Netzteil muss dann maximal 14*50 = 700 mA liefern, der Trafo muss auf mindestens 5,25 VA ausgelegt sein.

3.2 Netzteil

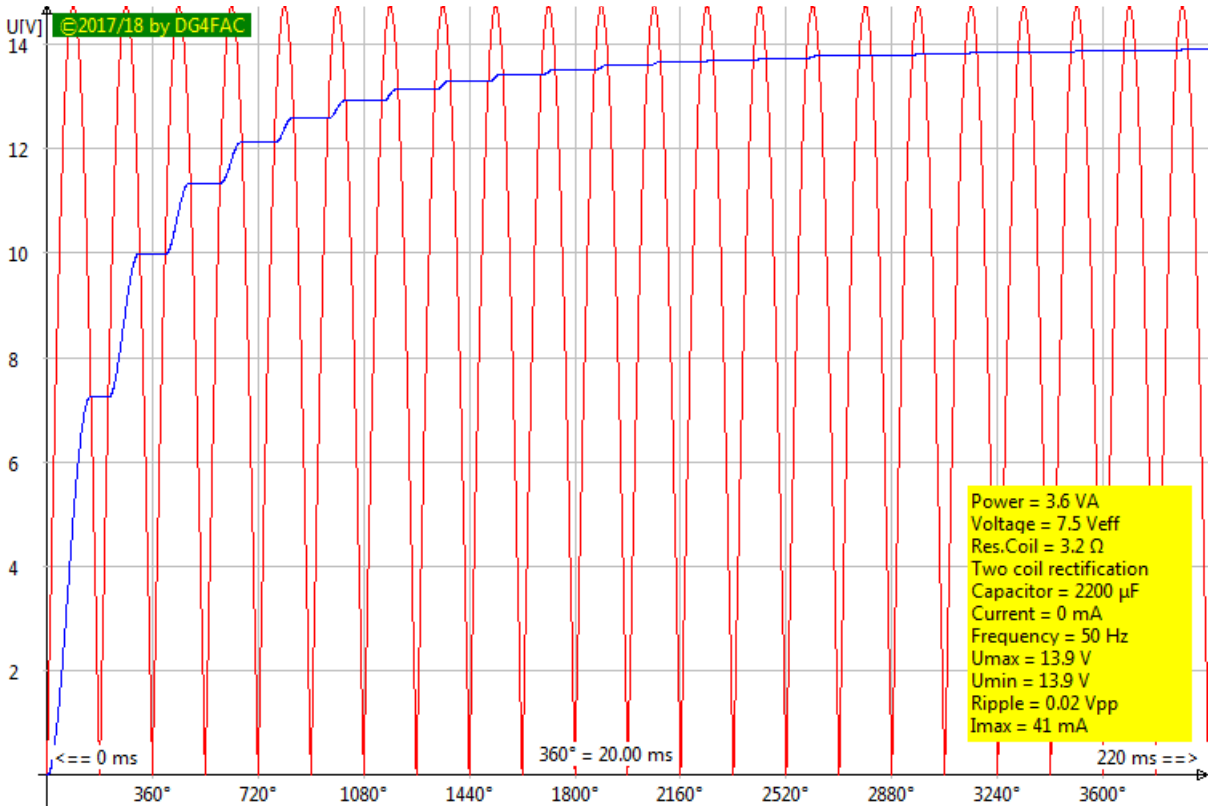
Soll die Schaltung aus einem vorhandenen 5,7V-Netzteil gespeist werden, kann dies über eine Siliziumdiode erfolgen, die die Speisespannung auf 5,0V herabsetzt. Ist kein solches Netzteil vorhanden, kann ein Trafonetzteil wie im Schaltbild die Speisung übernehmen. Die 1N4001-Diode wird dann überbrückt, der Elko von 1.000µF entfällt.



Das Netzteil muss folgende Eigenschaften besitzen:

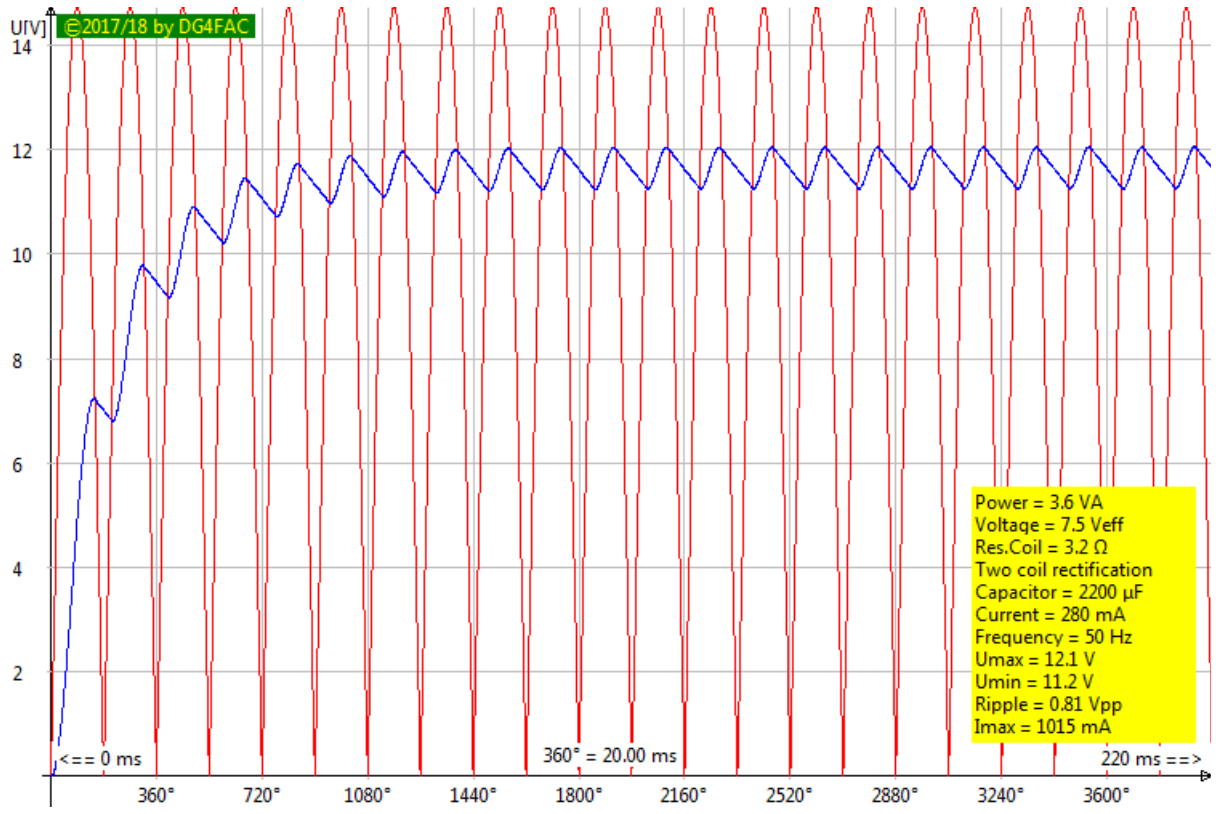
1. Der Spannungsregler 7805 benötigt mindestes 7V, um korrekt zu arbeiten. Es kommt daher ein Trafo mit 7,5 oder 9V infrage.
2. Der Strom durch die LEDs beträgt maximal 14 * 20 = 280 mA. Es muss daher ein Trafo mit mindestens 7,5V * 0,28 = 2,1 VA bzw. 9V *0,28A = 2,5VA verwendet werden.
3. Sollen die LEDs mit ihrer maximalen Leistung betrieben werden (50 mA), müsste es eher ein 10VA-Typ sein.

Im Schaltbild ist so ein Standard-Netzteil mit einem 2*7,5V/3,6VA-Trafo mit seiner Dimensionierung zu sehen.



Das hier simuliert das dargestellte Netzteil ohne Last (siehe die Power-Supply-Software hier). Die Spannung am Elko bleibt ohne Last bei deutlich unter 16 V, deshalb kommen die Elkos mit einer Spannungsbelastbarkeit von 16 V aus. Wird ein 9V-Trafo verwendet, werden 16V überschritten und der Elko muss auf 25V ausgelegt werden.

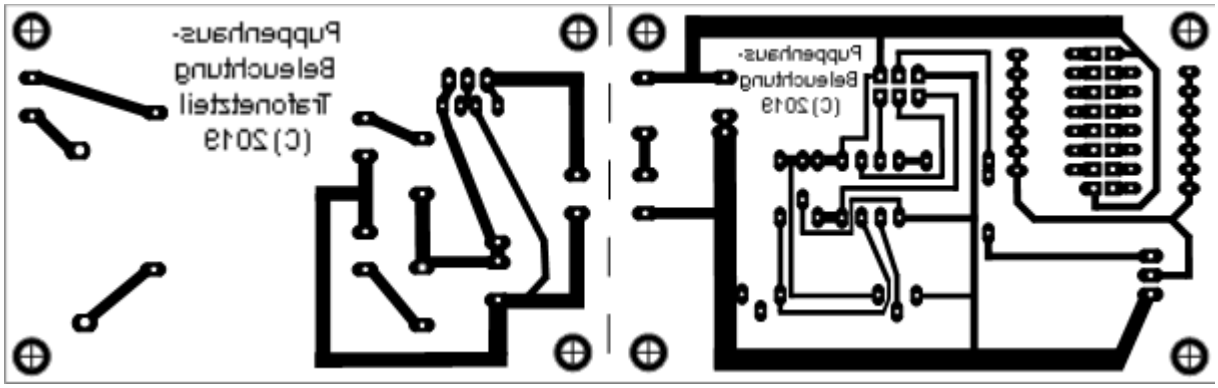
Das hier ist die Netzteilspannung bei 280 mA Last. Der Elko von 2200µF ist ausreichend, zur Not reicht auch ein 1000µF aus.



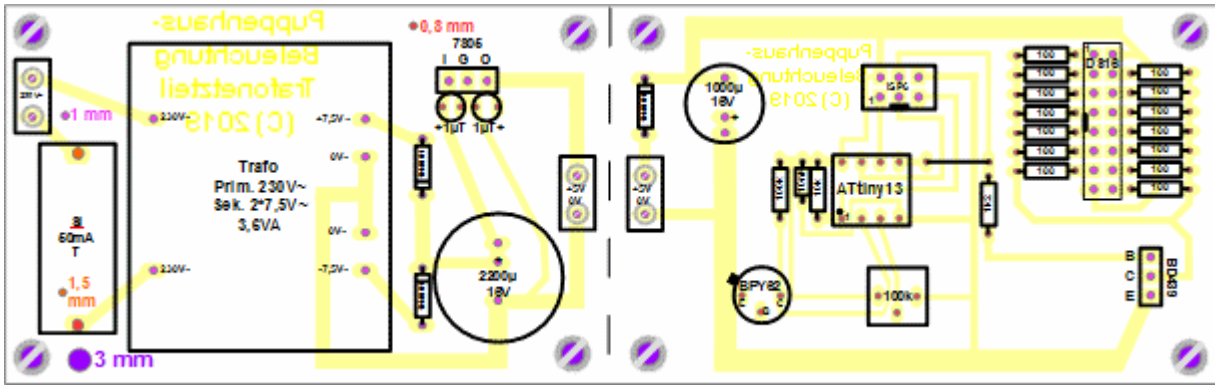
4 Aufbau

4.1 Gedruckte Platine

Das Layout für die gedruckte Platine wurde diesmal mit OpenOffice-Draw gezeichnet (was ziemlich mühsam vor sich geht). Hier gibt es das entsprechende OpenOffice-Dokument.



Die linke Hälfte der 160 x 50-mm-Platine ist für das Netzteil, die rechte Hälfte für den Prozessor und die LED-Ansteuerung vorgesehen. Wer das Netzteil nicht braucht, belichtet eine Viertel-Europlatine 80x50mm mit dem rechten Teil des Layouts.



Das ist die Platzierung aller Komponenten auf der Platine. Die 230V-Anschlüsse sind an einer Schraubklemme verfügbar. Die Sicherung passt in einen 5x20-mm Sicherungshalter mit aufsetzbarer Plastikkappe. Der 3,6VA-Trafo passt in die entsprechenden Bohrlöcher. Wer das Netzteil separat verwenden will, kann die 5V an eine weitere Schraubklemme führen.

Die LEDs sind an eine 16-polige Steckerleiste angeschlossen und können mit einer Buchse verpolungssicher mit Flachbandkabel an die LEDs verdrahtet werden.

4.2 Stückliste Prozessorteil

Das hier sind alle verwendeten Bauteile in einer Stückliste. Mit ca. 17 € sind Sie voll dabei. Bitte beachten, dass der ATtiny13 programmiert werden muss, um seine Funktion erfüllen zu können (siehe Software-Kapitel).

Stückliste Puppenhaus-Beleuchtung

Anzahl	Bauteil	Reichelt-#	Einzelpreis	Gesamtpreis
1	Mikrocontroller ATtiny13	ATTINY 13A-PU	0,68	0,68
1	Transistor BD439	BD 439	0,31	0,31
1	Fototransistor BPY62/IV	BPY 62/IV	1,80	1,80
1	Leistungsdiode 1N4001	1N 4001	0,02	0,02
14	Leuchtdioden 3mm weiß	SLOAN L3-W32N-BV	0,64	8,96
14	Widerstand 100 Ohm	1/4W 100	0,10	1,44
1	Widerstand 1k2	1/4W 1k2	0,10	0,10
1	Widerstand 10k	1/4W 10k	0,10	0,10
1	Widerstand 100k	1/4W 100k	0,10	0,10
1	Elko 1000µF/16V	RND 150EHR92035	0,14	0,14
1	Keramikkondensator 10nF	X7R-2,5 10N	0,07	0,07
1	Trimmer 100k liegend	76-10 100K	0,30	0,30
1	Stiftleiste 2x8	SL 2X10G 2,54	0,10	0,10
1	Wannenstecker 6-polig gerade	WSL 6G	0,16	0,16
1	Pfostenverbinder 2x8	BKL 10120113	0,52	0,52
0,33	Flachbandkabel 16-polig 3m	AWG 28-16F 3M	3,70	1,22
1	Zwergstecker 2,6mm rot	ZS 26 RT	0,32	0,32
1	Zwergstecker 2,6mm schwarz	ZS 26 SW	0,30	0,30
1	IC-Fassung 8-polig	MPE 001-1-008-3	0,26	0,26
Gesamt				16,91

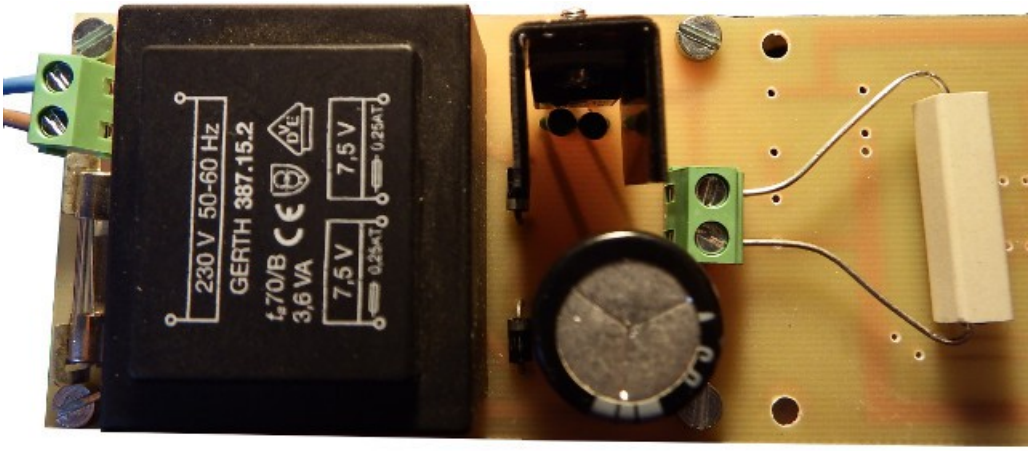
4.3 Stückliste Netzteil

Wer das Netzteil bauen will braucht noch mal 6,5 Euronen dafür.

Stückliste Trafonetzteil Puppenhaus-Beleuchtung

Anzahl	Bauteil	Reichelt-#	Einzelpreis	Gesamtpreis
1	Spannungsregler 7805	TS 7805 CZ	0,35	0,35
1	Trafo 2*7,5V 3,6VA	387.15-2	3,15	3,15
1	Sicherung 50mA träge 5x20	RND 170-00042	1,52	1,52
1	Sicherungshalter 5x20	PL 112000	0,19	0,19
1	Sicherungshalterkappe	RND 170-00007	0,08	0,08
1	Elko 2200µF 16V	RAD 2.200/16	0,21	0,21
2	Tantalelko 1µF 16V	TANTAL 1,0/35	0,22	0,44
2	Anschlussklemme 2-polig	AKL 057-02	0,31	0,62
Gesamt				6,56

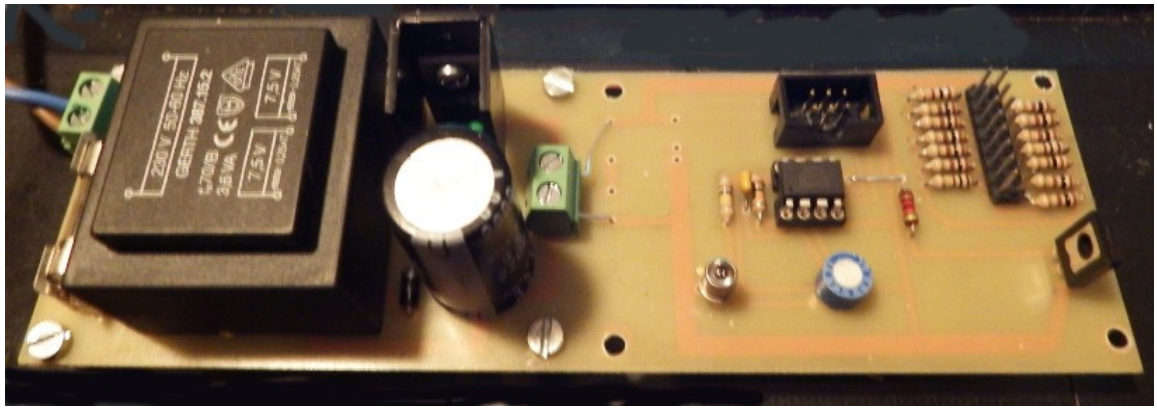
4.4 Bestückung und Test des Netzteils



Das ist das fertig aufgebaute Netzteil. Für den Dauerlast-Test ist ein 18Ω/5W-Widerstand angeschlossen, der dauerhaft 0,28A zieht. Alle Temperaturen bleiben dabei im vertretbaren Rahmen.

4.5 Bestückung der Gesamtplatine

Das ist die Gesamtplatine mit Netzteil und Prozessor-Elektronik. Es ist noch viel Platz auf der Platine.



Seitenanfang	Eigenschaften	Hardware	Aufbau	Software

5 Software

Die Software ist in Assembler geschrieben, damit sie mühelos in den kleinen Flashspeicher des ATtiny13 passt. Der ist mit gerade mal 8% seiner Kapazität ausgelastet.

5.1 Download

Die Software im asm-Format kann hier heruntergeladen oder im Anhang angesehen werden.

In der jetzigen Version ist die Hintergrundbeleuchtung abgeschaltet, weil ich keinen funktionierenden BPY62 mehr habe. Die Regelung mit dem Trimmer funktioniert aber einwandfrei.

5.2 Assemblieren

Zum Assemblieren des Quellcodes ist ein Assembler vonnöten, der .IF-, .ERROR- und .MESSAGE-Direktiven versteht. Der mit dem Studio 4 und höher ausgelieferte Assembler 2 kann das. Wer nicht mit Windows arbeitet, kann sich meinen Assembler gavrasm entweder als ausführbare Linux- oder Windows-64-Bit Version herunterladen oder für andere Betriebssysteme den Quellcode dafür und diesen mit FPC kompilieren.

Wie das Assemblieren mit gavrasm geht ist hier näher beschrieben. Die vom gavrasm ausgegebene Warnung kann man ohne Weiteres ignorieren.

5.3 Flashen

Die vom Assembler produzierte Hex-Datei ist in den Controller zu brennen. Dafür, dass das in der fertigen Schaltung geht, gibt es den ISP6-Stecker. Die werksseitig eingestellten Fuses des ATtiny13 müssen dabei nicht verstellt werden, der Prozessor arbeitet mit 1,2 MHz Takt und verbringt mehr als 99% der Zeit mit Schlafen und dem Warten auf AD-Wandler-Interrupts.

5.4 Software- und Hardware-Debugging

Als Testhilfen sind in der Software die nachfolgend beschriebenen Schalter eingebaut. Alle Schalter sind auf Eins ("Yes") zu setzen um sie zu aktivieren und auf Null ("No") um sie

abzuschalten. Es sollte immer nur einer der Debugschalter eingeschaltet sein. In der Betriebsversion müssen alle Schalter auf Null gesetzt werden, damit das Programm normal abläuft.

1. Debug-Schalter zur direkten Anzeige der AD-Wandler-Ergebnisse vom Potentiometer/Trimmer: das zeigt die Messwerte am Eingang ADC2 als PWM-Wert an. Dazu ist an den DB16-Stecker mindestens eine LED anzubringen. Bei voll aufgedrehtem Trimmer sollte(n) die LED(s) voll leuchten, bei niedrigeren Einstellungen entsprechend dunkler werden. Im Quellcode der Software ist dazu in Zeile 23 der Schalter **Debug_trim** auf "Yes" zu stellen, zu assemblieren und die erzeugte Hexdatei ins Flash zu brennen. Die Steuerung durch den Fototransistor ist dabei abgeschaltet.
2. Debug-Schalter zur direkten Anzeige der AD-Wandler-Ergebnisse vom Fototransistor: das zeigt die Messwerte am Eingang ADC3 als PWM-Wert an. Dazu ist an den DB16-Stecker mindestens eine LED anzubringen. Bei voll beleuchtetem Transistor sollte(n) die LED(s) nahezu ausgehen, bei voller Dunkelheit voll angehen. Im Quellcode der Software ist dazu in Zeile 24 der Schalter **Debug_opto** auf "Yes" zu stellen, zu assemblieren und die erzeugte Hexdatei ins Flash zu brennen. Die Steuerung durch den Trimmer ist dabei abgeschaltet.

Nicht vergessen, die Schalter wieder auf "No" zu setzen, um die betriebsfähige Endversion zu erzeugen.

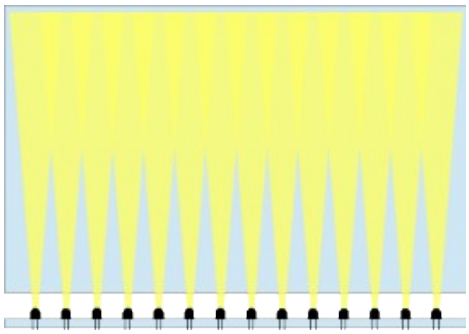
Zu Beginn des Programmablaufes erfolgt eine Testmultiplikation, um das Programm einfacher simulieren zu kömnnen (z. B. mit avr_sim). Zwei Parameter können simuliert werden:

1. **cTrim1** kann auf die ADC-Summe aus 64 Einzelmessungen des Potentiometers gesetzt werden (maximal 64*1023).
2. **cOpto1** stellt die erste ADC-Summe aus 64 Einzelmessungen des Fototransistors dar (maximal 64*1023).

Mit Vorliegen des ersten gemessenen Wertes wird dieser Wert überschrieben.

Seitenanfang	Eigenschaften	Hardware	Aufbau	Software	RGB-Designer

Lob, Tadel, Fehlermeldungen, Genöle und Geschimpfe oder Spam bitte über das Kommentarformular an mich.



Puppenhausbeleuchtung mit ATtiny13 **Assembler Quellcode Puppenhausleuchte**

Assembler-Quellcode für die Puppenhausleuchte

Die Originalsoftware im asm-Format gibt es hier.

```
;
; ********************************
; * Puppenhausleuchte ATtiny13    *
; * Version 1 Januar 2019         *
; * (C)2019 avr-asm-tutorial.net  *
; ********************************
;
.nolist
.include "tn13adef.inc" ; Define device ATtiny13A
.list
;
; ********************************
;         D E B U G
; ********************************
;
.equ Yes = 1
.equ No = 0
;
; Do not alter PWM value
.equ Debug_pwm = No
;
; Only trim and opto ADC debugging
.equ Debug_trim = Yes
.equ Debug_opto = No
;
; Simulate the first value on start-up
.equ cTrim1 = 64*1023 ; Sum value trimmer
.equ cOpto1 = 64*512 ; Sum value fototransistor
;
; ********************************
;        H A R D W A R E
; ********************************
;
; Device: ATtiny13A, Package: 8-pin-PDIP_SOIC
;
;
;         1 /         |8
;  Res o--|RESET  VCC|--o +5V
; Foto o--|PB3    PB2|--o
;  Pot o--|PB4    PB1|--o
```

```
;   0V o--|GND    PB0|--o LEDs
;       4 |__________|5
;
;
; *********************************
;  P O R T S   A N D   P I N S
; *********************************
;
.equ pLedO = PORTB ; Led output port
.equ pLedD = DDRB ; Led direction port
.equ bLedO = PORTB0 ; Led output portpin
.equ bLedD = DDB0 ; Led direction portpin
;
; *********************************
;   A D J U S T A B L E   C O N S T
; *********************************
;
.equ clock=1200000 ; Define clock frequency
.equ cLedMin = 10 ; Minimum light intensity
;
; *********************************
;  F I X  &  D E R I V.  C O N S T
; *********************************
;
.equ cTc0Clk = clock / 256 ; Define from clock
;
;
; *********************************
;           T I M I N G
; *********************************
;
; LED-PWM:
;  Clock frequency  = 1,200,000 Hz
;  Prescaler        =        64
;  8-Bit PWM        =       256
;  PWM-Frequency    =        73.2 Hz
;  PWM cycle        =        13.65 ms
;
; AD conversion:
;  AD clock prescaler   = 128
;  AD conversion cycles =  13
;  AD summation         =  64
;  AD channels          =   2
;  Complete measurement =   5.63 Hz
;  Complete cycle       = 177 ms
;  Number of PWM cycles =  13.0
;
; Sleep share       = 99.1%
;
; *********************************
;       R E G I S T E R S
; *********************************
;
; free: R0 to R11
.def rAdcA = R12 ; Channel A MSB
.def rAdcL = R13 ; AD summation, LSB
.def rAdcH = R14 ; dto., MSB
.def rSreg = R15 ; Save/Restore status port
.def rmp = R16 ; Define multipurpose register
.def rimp = R17 ; Multipurpose inside ints
.def rFlag = R18 ; Flag register
  .equ bAdcCc = 0 ; ADC cycle complete
  .equ bAdcCh = 1 ; ADC cycle channel
.def rAdcCtr = R19 ; ADC measurement counter
```

```
; free: R20 to R29
; used: Z = R31:R30 for multiplication
;
; *********************************
;           S R A M
; *********************************
;
.dseg
.org SRAM_START
sLabel1:
.byte 16 ; Reserve 16 bytes
;
; *********************************
;         C O D E
; *********************************
;
.cseg
.org 000000
;
; *********************************
; R E S E T  &  I N T - V E C T O R S
; *********************************
        rjmp Main ; Reset vector
        reti ; INT0
        reti ; PCI0
        reti ; OVF0
        reti ; ERDY
        reti ; ACI
        reti ; OC0A
        reti ; OC0B
        reti ; WDT
        rjmp AdcIsr ; ADCC
;
; *********************************
;  I N T - S E R V I C E  R O U T .
; *********************************
;
; AD conversion complete interrupt
AdcIsr:
  in rSreg,SREG ; Save SREG
  in rimp,ADCL ; Read LSB conversion result
  add rAdcL,rimp ; Add LSB to sum
  in rimp,ADCH ; dto., MSB
  adc rAdcH,rimp ; Add MSB to sum
  dec rAdcCtr ; Decrease counter
  brne AdcIsr1 ; Not zero
  ; ADC measurement counter at zero
  sbr rFlag,1<<bAdcCc ; Set conversion complete flag
  out SREG,rSreg ; Restore SREG
  reti
AdcIsr1:
  ldi rimp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0) ;
Restart ADC
  out ADCSRA,rimp ; in control port A
  out SREG,rSreg ; Restore SREG
  reti
;
; *********************************
;  M A I N  P R O G R A M  I N I T
; *********************************
;
Main:
        ldi rmp,Low(RAMEND)
        out SPL,rmp ; Init LSB stack pointer
```

```
; Init the LED output port
  cbi pLedO,bLedO ; LEDs off
  sbi pLedD,bLedD ; LED pin = output
; Init Timer TC0 as PWM
  ldi rmp,0xff ; Half PWM
  out OCR0A,rmp
  ldi rmp,(1<<COM0A1)|(1<<WGM01)|(1<<WGM00) ; Fast PWM
  out TCCR0A,rmp ; To TC0 control port A
  ldi rmp,(1<<CS01)|(1<<CS00) ; Prescaler = 64
  out TCCR0B,rmp ; To TC0 control port B
; Start Adc
  ldi rmp,High(cTrim1) ; Simulate trimmer value
  mov rAdcA,rmp
  ldi rmp,High(cOpto1) ; Simulate fototransistor value
  mov rAdcH,rmp
  sbr rFlag,1<<bAdcCh ; Force multiplication
  rcall AdcCc ; Start ADC cycle
; Init Sleep and interrupts
  ldi rmp,1<<SE ; Sleep enable
  out MCUCR,rmp ; in Master control port
        sei ; Enable interrupts
;
; *********************************
;    P R O G R A M   L O O P
; *********************************
;
Loop:
  sleep ; Go to sleep
  nop ; Wake up by ADC int
  sbrc rFlag,bAdcCc ; Conversion complete flag?
  rcall AdcCc ; Call conversion complete
  rjmp Loop ; and return to sleep
;
; ADC cycle complete
AdcCc:
  cbr rFlag,1<<bAdcCc
  sbrc rFlag,bAdcCh ; Next channel?
  rjmp AdcCcCalc ; Channels complete, calculate
  sbr rFlag,1<<bAdcCh ; Set next channel
  .if Debug_trim == Yes
    out OCR0A,rAdcH ; Write result to PWM
        .endif
  mov rAdcA,rAdcH ; Copy MSB result
  ldi rmp,(1<<MUX1)|(1<<MUX0) ; Measure channel ADC3
  out ADMUX,rmp
  rjmp AdcCcRestart
AdcCcCalc:
  ; End of second cycle, calculate new PWM value
  .if Debug_opto == Yes
    out OCR0A,rAdcH
        .endif
  mov rmp,rAdcH ; Copy MSB
  neg rmp ; Fototransistor input, difference to 256
  cpi rmp,cLedMin ; Less than minimum light?
  brcc AdcCcCalc1 ; No
  ldi rmp,cLedMin ; Set to minimum light
AdcCcCalc1:
  clr ZL ; Clear multiplication result, LSB
  clr ZH ; dto., MSB
  mov rAdcL,rAdcH ; MSB to LSB
  clr rAdcH ; MSB = 0
AdcCcCalc2:
  tst rmp ; End of multiplication?
  breq AdcCcCalc4
```

```
  lsr rmp ; Next bit
  brcc AdcCcCalc3 ; 0, do not add
  add ZL,rAdcL ; Add LSB
  adc ZH,rAdcH ; and MSB
AdcCcCalc3:
  lsl rAdcL ; Multiply by 2, LSB
  rol rAdcH ; dto., MSB
  rjmp AdcCcCalc2 ; Continue multiplication
AdcCcCalc4:
  .if (Debug_pwm != Yes) && (Debug_trim != Yes) && (Debug_opto != Yes)
    out OCR0A,ZH ; Set new PWM value to MSB result
        .endif
  cbr rFlag,1<<bAdcCh
  ldi rmp,(1<<MUX1) ; Measure channel ADC2
  out ADMUX,rmp
AdcCcRestart:
  clr rAdcL ; Clear sum, LSB
  clr rAdcH ; dto., MSB
  ldi rAdcCtr,64 ; Start next 64 measurements
  ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0) ;
Restart ADC
  out ADCSRA,rmp ; in control port A
  ret
;
; End of source code
;
```