Handbuch

zu Einführung und Bedienung
von

PicAlic

Handbuch für PicAlic Version 1.0

Über diese PIC**ALIC Version 1.0:**

Das hier vorliegende Programm implementiert einen Lichtcontroller nach ALIC-Standard auf Microchip PIC-Microcontrollern. Es werden bis zu acht Ausgangskanäle mit PWM-Modulation zur Einstellung der Helligkeit zur Verfügung gestellt.
Zur Steuerung über eine RC-Anlage können bis zu zwei Signaleingänge genutzt werden, die jeweils einen 4-fach Memory Switch, sowie die Umsetzung des RC-Signals in einen der Geberstellung proportionalen Zahlenwert bieten.
Die Steuerung kann aber auch durch einfache Logikpegel an den Porteingängen des Microcontrollers erfolgen.

Der Quellcode kann zur Erzeugung von Hex-Codes unverändert für die folgenden Controllertypen verwendet werden:

PIC12F629, PIC12F675, PIC16F630, PIC16F636, PIC16F627A, PIC16F628A, PIC16F648A.

Die aktuelle Version des Programms kann als ZIP-Datei heruntergeladen werden:

http://picalic.de/downloads/picalic.zip

Das Programm wurde nach bestem Wissen und mit großer Sorgfalt erstellt und getestet. Dennoch kann nicht ausgeschlossen werden, daß sich evtl. noch der ein- oder andere "Bug" im Programm versteckt. Bugreports ggf. bitte an mich per Email. (siehe "Kontakt") - danke!

Links:

RC Line Forum: Diskussionen um Modellbeleuchtung im Allgemeinen und ALIC im Speziellen.

Was uns noch fehlt:

Leute die mit Lust, Zeit und Know-How, um PC-Tools zu schreiben, z.B. eine grafische Eingabemöglichkeit der Lichtmuster oder ein Simulationsprogramm.

Kontakt:

Thomas Elger
Bgm.-Ametsbichler-Ring 2
D-85586 Poing

Email: thomas(ät)picalic.de

Was ist das?

Der "Advanced Light Controller Instruction Code" **(ALIC)** ist ein standardisierter Prozessor-Befehlscode, der zur Steuerung von Beleuchtungseinrichtungen im Modellbau-Bereich entwickelt wurde und weiter entwickelt wird. Damit sind mit geringem Programmieraufwand komplexere Lichtabläufe möglich, aber auch Standard-Aufgaben, wie z.B. die ACL-Blitzer bei Flugzeugmodellen, lassen sich damit leicht und kostengünstig als Selbstbaulösung realisieren. Der Code ist nicht an eine bestimmte Controller-Hardware gebunden, er kann also auf verschiedenen Hardware-Plattformen verwendet werden.

Wozu soll das gut sein?

Modellbauer verschiedener Sparten, wie RC-Flugmodelle, Modellbahnen, Schiffs-Modelle, Funktionsmodelle, Truck-Modelle usw., stehen immer wieder mal vor der Aufgabe, verschiedene Lichter (meist LEDs) ihrer Modelle ferngesteuert zu schalten oder bestimmte Blink- oder Lichtmuster erzeugen zu können. Für gewisse Standard-Anwendungen, wie z.B. ACL-Blitzer bei Flugmodellen, gibt es fertige Controller zu kaufen. Diese sind aber - wenn überhaupt - nur sehr eingeschränkt programmierbar und können daher nicht immer optimal auf die eigenen Vorstellungen angepasst werden. Für speziellere Aufgaben muß dann schließlich doch ein Beleuchtungs-Controller im Selbstbau erstellt werden.

Herzstück eines solchen Lichtcontrollers ist ein handelsüblicher Microcontroller. Dieser kann kleine LED-Ströme, wie etwa die LEDs einer Ampelanlage auf einer H0-Modellbahnanlage, oft auch direkt ansteuern, so werden außer dem Microcontroller und einer geeigneten Spannungsquelle lediglich simple Vorwiderstände für die LEDs benötigt. Größere Leistungen können problemlos über sehr einfache Treiber-Schaltungen erreicht werden. Die Hardware ist daher billig und kann von den meisten Modellbauern auch ohne tiefer gehende Elektronik-Kenntnisse leicht zusammengelötet werden!

Komplizierter wird es jedoch bei der Programmierung des Microcontrollers. Dazu ist zunächst einmal ein Programmiergerät erforderlich, das den fertigen Programmcode in den Microcontroller "brennen" kann. Schaut man sich den Preis für einen fertig gekauften Licht-Controller an (der ja oft nicht mal das kann, was man eigentlich haben will...), und dagegen den Preis für ein günstiges Programmiergerät, relativiert sich die notwendige Investition für das Programmiergerät.

Die größte Hürde bestand aber bisher darin, das Programm für den Microcontroller zu erstellen! Egal, ob in Assembler oder einer höheren Sprache (z.B. Basic oder C) programmiert wird: der "Selbstbauer" mußte die entsprechende Programmiersprache beherrschen, sich mit der Architektur des verwendeten Microcontrollers auskennen und selbst für einfache Lichtmuster verhältnismäßig aufwändige Programme schreiben. Das dürfte für die Meisten das K.O. für entsprechende Projekte bedeutet haben, denn wer will schon einen mehrwöchigen Programmierkurs absolvieren, nur um ein paar LEDs nach eigenen Vorstellungen blinken zu lassen?

Hier kommt nun **ALIC** ins Spiel! Zwar wird auch hier ein Programm erstellt - denn irgendwie muß man dem Controller ja sagen, wie die LEDs blinken sollen - aber der Befehlssatz dieser "Sprache" besteht aus nur wenigen Elementen und ist direkt auf die Anwendung zur Erzeugung von Lichtmustern zugeschnitten. Die Lichtmuster werden durch einfache Befehle definiert, in der Art etwa: "Licht volle Stärke für 500ms, dann Licht auf halbe Stärke für 300ms, dann Licht aus für 2 Sekunden, dann wiederhole das Ganze von vorn".

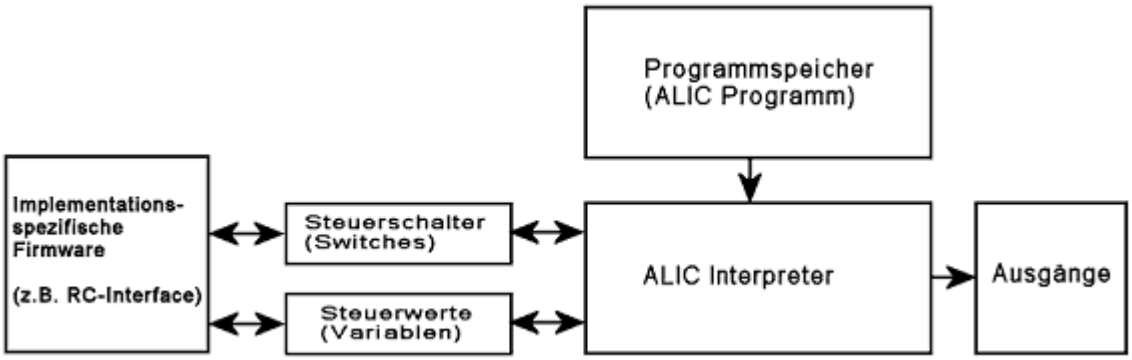
Um eine solche individuelle Lichtsteuer-Lösung implementieren zu können, braucht man also derzeit:

- Programmier-Werkzeuge, um Code zu erzeugen und diesen in den verwendeten Microcontroller zu brennen,
- ein fertiges Programm, das für den verwendeten Microcontroller geschrieben wurde und den ALIC-Code verarbeitet ("Interpreter")
- **keine** näheren Kenntnisse über die Prozessor Architektur und Programmierung des verwendeten Microcontrollers, **dafür** aber gewisse ALIC-Kenntnisse,

Durch die Standardisierung des Codes können von verschiedenen Entwicklern ALIC-Interpreter für diverse Microcontroller geschrieben werden. Auch entsprechende Tools zur Erzeugung von ALIC-Codes mit grafischer Benutzer-Oberfläche oder Simulatoren können unabhängig von einer bestimmten Controller-Hardware entwickelt werden.
Natürlich können auf dieser Basis auch fertige Controller-Bausteine entwickelt werden, die dann auch ohne spezielle Programmier-Hardware über eine gängige Schnittstelle (z.B. USB) vom PC aus mit dem Licht-Steuercode programmiert werden können.

Das Funktionsprinzip:

Der ALIC Controller arbeitet im Prinzip wie ein "gewöhnlicher" Microcontroller, d.h. er holt sich einen Befehl nach dem anderen aus dem Programmspeicher, decodiert ihn und führt die entsprechende Operation aus. Allerdings hat jeder der vorhandenen Ausgänge virtuell seine eigene Ausführungs-Einheit, so daß die Programme für die verschiedenen Ausgänge unabhängig voneinander im Multitasking-Betrieb ablaufen. Der Programmspeicher ist aber nicht nach Ausgangskanälen aufgeteilt, es kann also für verschiedene Ausgänge auch derselbe Programmcode verarbeitet werden.



Zur Beeinflussung des Programmablaufs von außen stehen eine gewisse Anzahl von logischen Schaltern ("Switches") zur Verfügung, sowie ggf. auch ein Satz von Variablen. Die Steuerschalter können lediglich den Zustand "*ein*" oder "*aus*" annehmen, die Variablen jeweils einen Wert zwischen *0* und max. *255*. Die Schalter- und Variablenwerte können über Befehle im ALIC-Programm abgefragt werden und dadurch der Programmlauf beeinflusst werden. Eine direkte Verbindung zwischen der Implementations-spezifischen Firmware und dem ALIC-Interpreter gibt es nicht. Die Kommunikation zwischen beiden findet ausschließlich über das Setzen/Löschen von Switches oder die Übergabe von Steuerwerten statt.

In der Praxis, z.B. bei einem RC-Interface mit zwei Eingängen, werden dann die Stellungen der entsprechenden Schieber am Sender von der RC-Interface Firmware als Zahlenwert in zwei verschiedenen Steuer-Variablen abgelegt. Vom ALIC-Programm aus können diese dann mit bedingten Sprungbefehlen (Ausführung wenn Wert größer bzw. kleiner, als...) ausgewertet werden und entsprechend in den gewünschten Programmteil verzweigt werden, z.B. Blitzer "aus" bei Werten unterhalb 30% Schieberwert, Einfach-Blitzer bei Werten bis 70%, darüber ein Doppel-Blitz.

Ebenso könnte die RC-Interface Firmware auch eine Multiswitch-Auswertung beinhalten und 8 Steuerschalter entsprechend ein- oder ausschalten.

Das PIC**ALIC** Kochrezept

So wird's gemacht: Schritt für Schritt zum eigenen Lichtsteuer-Controller

Zutaten:

1 x PIC-Microcontroller, z.B. PIC16F636
1 x geeignetes Programmiergerät incl. Brenner-Software dafür, oder man kennt einen, der eins hat...
1 x MPLAB IDE (Entwicklungsumgebung), gibt's kostenlos von der Microchip Webseite.
1 x PIC**ALIC** Programmpaket (zu dem auch diese Dokumentation gehört)

Vorbereitung:

Man nehme die MPLAB IDE und installiere sie auf dem eigenen PC. Sofern noch nicht geschehen, entpacke man auch das ZIP-Archiv des PIC**ALIC** Programmpakets auf die Festplatte des PCs, so daß sich dann z.B. der Ordner "C:\PICALIC\" auf der Festplatte befindet, mit den Dateien und Ordnern des PICALIC-Pakets darin.

Zubereitung:

1. Die MPLAB Entwicklungsumgebung wird durch Doppelklick auf das entsprechende Symbol gestartet.

2. Unter dem Menüpunkt *Configure/Select device...* suche man den verwendeten Microcontroller-Typen aus.

3. Im PICALIC-Ordner, Unterordner "ALIC_source" wird nun die ASM-Datei mit den ALIC-Befehlen für die gewünschte Anwendung erstellt und unter einem sinnreichen Namen dort abgelegt, z.B. **blitzer.asm**.

4. Tipp: statt eine solche Datei von Grund auf völlig neu zu erstellen, ist es oft einfacher, eine vorhandene Datei für eine ähnliche Anwendung mit "*File/Open...*" zu laden, abzuändern und mit "*File/Save as...*" unter neuem Namen abzuspeichern.

5. Mit *File/Open...* öffnet man nun die Datei **picalic.asm** aus dem PICALIC-Ordner und nimmt ggf. Änderungen vor, um die Konfiguration des Programmes auf die gewünschte Anwendung anzupassen (z.B. Anzahl der Beleuchtungs- und RC-Kanäle).
   Vor Allem muß die Zeile "#define ALICFILE..." entsprechend auf die zu verwendende ALIC-Quelldatei angepasst werden, z.B.

6. **#define ALICFILE ALIC_source/blitzer.asm**

7. Das Fenster mit der Datei "picalic.asm" ggf. durch Mausklick darauf in den Vordergrund holen und im Menü "*Project/Quickbuild picalic.asm*" die Übersetzung starten.

8. Im Fenster "Output" nachschauen, ob keine "Warnings" oder gar "Errors" erschienen sind. Diesen Meldungen ggf. nachgehen und den Code ausbessern.

9. Wenn keine Fehlermeldungen aufgetaucht sind,

10. Datei "**picalic.hex**" mit dem PIC-Programmiergerät in den Controller brennen.

Guten Appetit!

Mnemonic	Opcode	Parameter	Beschreibung
SETBRIGHT	0x00 -0xBF	<delay>	**Set Brightness value for time <delay>** Assembler-Syntax: SETBRIGHT <value>,<delay> Stellt die Helligkeit auf den Wert <value> ein. Der nachfolgende Befehl wird nach der Zeit <delay> (in "Ticks") ausgeführt. Der Helligkeitswert wird hierbei im Opcode codiert! Wertebereich:value: 0..191 (128 entspricht 100%)delay: 0..255
LIGHTOFF	0x00	<delay>	**Turn Light Off for time <delay>** Andere Form von (SETBRIGHT 0,<delay>)
LIGHTON	0x80	<delay>	**Turn Light On for time <delay>** Andere Form von (SETBRIGHT 128,<delay>)
SMOOTHOFF	0xC0		**Turn Smooth Mode Off** Helligkeitsänderungen nachfolgender SETBRIGHT-Anweisungen erfolgen direkt auf den angegebenen neuen Wert, ohne Zwischenwerte. (Standardeinstellung)
SMOOTHON	0xC1		**Turn Smooth Mode On** Änderungen zwischen den Helligkeitswerten nachfolgender SETBRIGHT-Anweisungen erfolgen kontinuierlich durch lineare Interpolation. Implementation: jedes Steuerkommando mit Opcode >= 0xC0 unterbricht ggf. den Smooth-Modus an dieser Programmstelle. SMOOTH bleibt aktiviert, aber es entsteht ggf. ein Helligkeits-Sprung.
HALTSWOFF	0xC2		**Halt while Channel's Switch is Off** Hält bei ausgeschaltetem Steuerschalter die Programmausführung an dieser Stelle an, bis der Schalter eingeschaltet wird.
HALTSWON	0xC3		**Halt while Channel's Switch is On** Hält bei eingeschaltetem Steuerschalter die Programmausführung an dieser Stelle an, bis der Schalter ausgeschaltet wird.
JSWOFF	0xC4	<addr>	**Jump if Channel's Switch is Off** Wenn der dem eigenen Kanal zugeordnete Steuerschalter ausgeschaltet ist, wird das Programm an Speicheradresse <addr> fortgesetzt.
JSWON	0xC5	<addr>	**Jump if Channel's Switch is On** Wenn der dem eigenen Kanal zugeordnete Steuerschalter eingeschaltet ist, wird das Programm an Speicheradresse <addr> fortgesetzt.
CLRSW	0xC6		**Clear Channel's Switch** Schaltet den dem eigenen Kanal zugeordneten Steuerschalter auf "aus".
SETSW	0xC7		**Set Channel's Switch** Schaltet den dem eigenen Kanal zugeordneten Steuerschalter auf "ein".
JUMP	0xC8	<addr>	**Jump** Das Programm wird an Speicheradresse <addr> fortgesetzt.
RNDDLY	0xC9	<maxticks>	**Random Delay** Das Programm wird nach einer zufällig erzeugten Wartezeit zwischen 0 und <maxticks> fortgesetzt.
RNDBRT	0xCA	<min>,<max>,<delay>	**Set Random Brightness** Der Helligkeitswert wird zufällig im Bereich <min> bis <max> eingestellt und liegt für die Dauer <delay> dort an. funktioniert nicht im Smooth-Modus
RNDJMP	0xCB	<chance>,<addr>	**Random Jump** Das Programm wird mit der Wahrscheinlichkeit <chance> an Addresse <addr> fortgesetzt. <chance> = 0..128 für nie bis immer.
JSWXOFF	0xCC	<x>,<addr>	**Jump if Switch X is Off** Wenn der Steuerschalter <x> ausgeschaltet ist, wird das Programm an Speicheradresse <addr> fortgesetzt.
JSWXON	0xCD	<x>,<addr>	**Jump if Switch X is On** Wenn der Steuerschalter <x> eingeschaltet ist, wird das Programm an Speicheradresse <addr> fortgesetzt.
CLRSWX	0xCE	<x>	**Clear Switch X** Der Steuerschalter <x> wird ausgeschaltet
SETSWX	0xCF	<x>	**Set Switch X** Der Steuerschalter <x> wird eingeschaltet
DEFTICKLEN	0xD0	<time>	**Define Length of Tick** (Header-Befehl)*

			Definiert die Länge eines Ticks in Millisekunden. Standard-Wert: 10 ms funktioniert nur, wenn das Programm mit "#define COMMTICK 0" assembliert wird!
DEFBRIGHT	0xD1	<value>	**Define Brightness Multiplier** (Header-Befehl)* Definiert einen Helligkeits-Multiplikator für den Ausgangs-Kanal. Bereich: 0..255, Standardwert = 128 = *1.0
DEFSWITCH	0xD2	<x>	**Define Channel Switch** (Header-Befehl)* Definiert einen anderen Schalter, der dem Ausgangs-Kanal zugeordnet ist (= channel switch) Standard: x = Kanalnummer
DEFVAR	0xD3	<x>	**Define Control Variable** (Header-Befehl)* Definiert eine andere Steuerwert-Variable, die dem Kanal zugeordnet ist (= channel variable) Standard: x = Kanalnummer
JVARB	0xD8	<val>,<addr>	**Jump if Channel's Variable is Below <val>** Wenn der inhalt der dem Kanal zugeordneten Variablen kleiner ist als <val>, wird das Programm bei <addr> fortgesetzt.
JVARAE	0xD9	<val>,<addr>	**Jump if Channel's Variable is Above or Equal <val>** Wenn der inhalt der dem Kanal zugeordneten Variablen größer oder gleich <val> ist, wird das Programm bei <addr> fortgesetzt.
JVARXB	0xDA	<x>,<val>,<addr>	**Jump if Variable X is Below <val>** Wenn der inhalt Variablen <x> kleiner ist als <val>, wird das Programm bei <addr> fortgesetzt.
JVARXAE	0xDB	<x>,<val>,<addr>	**Jump if Variable X is Above or Equal <val>** Wenn der inhalt der Variablen <x> größer oder gleich <val> ist, wird das Programm bei <addr> fortgesetzt.
DELAY	0xDC	<delay>	**Delay** Fügt eine Wartezeit von <delay> Ticks ein. Der vor diesem Befehl eingestellte Helligkeitswert wird nicht verändert.
P_RESET	0xFF		Program Reset Setzt für den aktuellen Kanal den Programmzähler auf die Einsprungadresse zurück und aktiviert die folgenden Werte: • Ausgangs-Helligkeit: 0 (Licht aus) • Smooth-Modus: aus • Ticklänge: Default (10ms) Schalterstellungen, Variablen und deren Zuordnungen bleiben unverändert.
*Header Befehle: diese Anweisungen müssen am Programmanfang stehen, d.h. unmittelbar nach Einsprungadresse und vor allen anderen Befehlen. Jeder andere Befehl, als vom Typ "Header", macht Header-Befehle im nachfolgenden Programmablauf unwirksam.			

Datenblatt

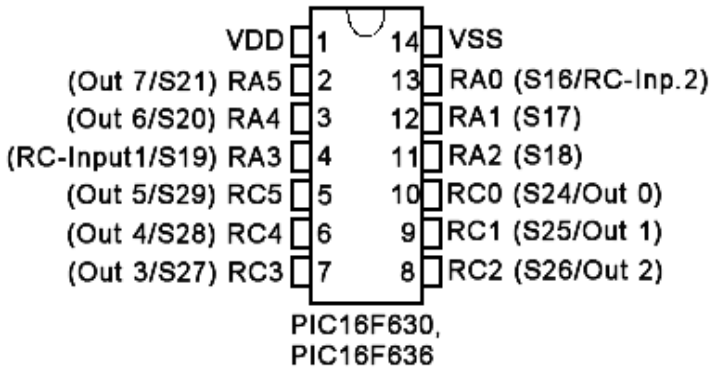
Features:

- bis zu 8 Lichtsteuerkanäle
- Helligkeitssteuerung durch PWM
- PWM bis 255 Schritte Auflösung bei 100Hz (PIC16F636, 8MHz Takt)
- optionales RC-Interface mit bis zu zwei Eingängen
- Auswertung d. RC-Impulse als 4-fach Memory-Switch und proportional
- Ausgänge wahlweise High- oder Low-Aktiv
- Minimale Anzahl externer Komponenten
- Freie Ports als Steuereingänge nutzbar
- Freie Ports auch als Ausgänge ansprechbar (über SETSWX/CLRSWX-Befehle)

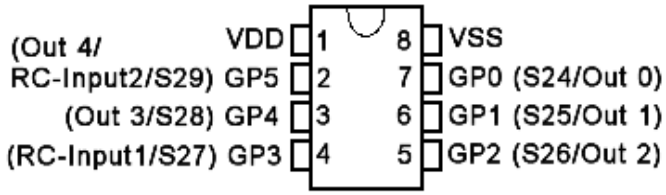
MicroChip PIC-Typen:

Assembler-Einstellung:	**Hex-Code und Pin-kompatibel:**	**max. Anzahl Ausgangskanäle:**	**max. Anzahl RC-Kanäle:**	**EEPROM-Größe (Bytes)** (=max. ALIC Programm-Speicherplatz)
PIC12F675	PIC12F629	5 (4)	1 (2)	128
PIC16F630		8	2	128
PIC16F636		8	2	256
PIC16F648A	PIC16F627A, PIC16F628A	8	1	PIC16F648: 256 PIC16F627A: 128 PIC16F628A: 128

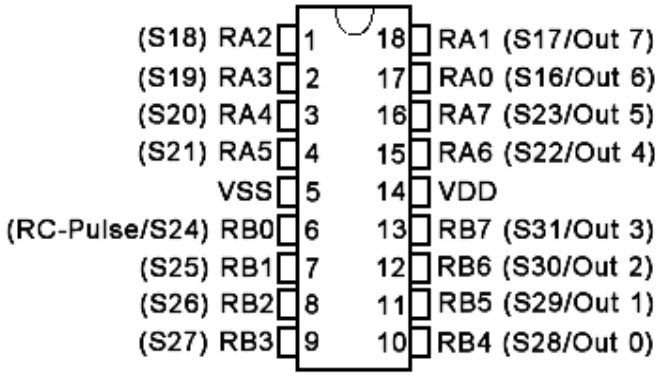
Pinbelegung:



PIC16F630,
PIC16F636



PIC12F675



PIC16F648A

VDD = +5V
VSS = 0V (Masse)
Out x = Steuer-Ausgang des jeweiligen Licht-Kanals (0..7)
RC-Input 1 = RC-Impuls vom Empfänger (erster Kanal)
RC-Input 2 = RC-Impuls vom Empfänger (zweiter Kanal)
Sxx = Schalternummer, unter der der jeweilige Port angesprochen werden kann.

ALIC Programmspeicher:

Der ALIC-Programmcode wird im Daten-EEPROM des Controllers abgelegt. In den ersten NCHAN Bytes (NCHAN=Anz.der Ausgangskanäle,1..8) sind die Einsprungadressen der Programmsegmente für die Ausgangskanäle abgelegt. Diese Daten werden im Main-Sourcecode erzeugt.

Im ALIC Quelltext muß ein entsprechendes Label für jeden existierenden Ausgangskanal vorhanden sein, das die Einsprungadresse des jeweiligen Programms kennzeichet.

Dieses Label lautet: **entry_chX**, wobei X die Kanalnummer (0..7) ist.

Steuerschalter (switches):

Switch Nummer:	Funktion:
0..7	per Standardeinstellung den Ausgangskanälen (0..7) zugeordnet. Sind weniger als 8 Kanäle vorhanden, können die übrigen Steuerschalter frei verwendet werden.
8,9	RC Memory Switches (RC-Kanal 1), Umschaltung bei kurzer Betätigung des Gebers aus der Mittelstellung.
10,11	RC Memory Switches (RC-Kanal 1), Umschaltung bei langer Betätigung des Gebers aus der Mittelstellung.
12,13	RC Memory Switches (RC-Kanal 2), Umschaltung bei kurzer Betätigung des Gebers aus der Mittelstellung.
14,15	RC Memory Switches (RC-Kanal 2), Umschaltung bei langer Betätigung des Gebers aus der Mittelstellung.
16..23	Adressiert die I/O-Ports RA0..RA7 High-Level entspricht "Switch On", unabhängig von "#define ACTIVELOW x"!
24..31	Adressiert die I/O-Ports RC0..RC5 bzw. GP0..GP5 High-Level entspricht "Switch On", unabhängig von "#define ACTIVELOW x"!
32..47	Für RC Multiswitch Interface (Beta-Version) oder Ähnliches.
48	RC inaktiv: ist "on" wenn an den RC-Eingängen keine gültigen Impulse anliegen, sonst "off".

Steuervariablen:

Variable Nummer:	Funktion:
0	Geberstellung des ersten RC-Kanals. Eine Impulslänge von 1,5ms (=ca. Geber-Mittelstellung) erzeugt einen Variablen-Wert von 64, bei ca. 1ms liegt der V.-Wert bei 0 und bei 2ms bei 128. (Standardeinstellung: Ausgangskanal 0 zugeordnet)
1	Geberstellung des zweiten RC-Kanals. (Standardeinstellung: Ausgangskanal 1 zugeordnet)

RC-Interface:

Optional können ein oder zwei Eingänge für die Steuerung über die RC-Anlage genutzt werden. Die Stellungen der entsprechenden Geber werden als Zahlenwert in den Variablen Nr. 0 und 1 abgelegt. Die RC-Signale werden auf evtl. Störimpulse geprüft und diese ggf. ausgefiltert, um Fehlauslösungen durch Empfangsstörungen zu vermeiden. Darüber hinaus ist in die Auswertung eine Hysterese eingebaut, um das Hin- und Herspringen der davon gesteuerten Funktion zu vermeiden, wenn der Geber sehr dicht am Schwellwert steht und die Impulse durch Rauschen auf dem Übertragungsweg evtl. etwas "zittern". Durch die Hysterese wird ein einmal erreichter Stellwert erst wieder durchschritten, wenn das Eingangssignal min. um den Betrag der Hysterese in die andere Richtung wandert.

Zusätzlich hat jeder RC-Kanal eine 4-fach Memory Switch Funktion. Diese arbeitet i.d.R. mit einem 3-stufigen Schalter am Sender, am Besten mit Rückstellfeder auf Mittelstellung. Wird der Schalter für einen kurzen Moment aus der Mittelstellung in eine der Endstellungen gebracht und wieder zurück (bitte nicht zu kurz, da das sonst als Störung ausgefiltert werden könnte), wird die erste bzw. zweite Memory-Funktion umgeschaltet. Bleibt der Schalter etwas länger in der Endstellung (z.B. >1s), wird stattdessen die dritte bzw. vierte Memory-Funktion umgeschaltet.

Das evtl. Fehlen von RC-Eingangsimpulsen kann durch Auswerten des Schalters 48 erkannt werden. So kann z.B. die Beleuchtung eines Modells bei ausgeschalteter RC-Anlage automatisch in einen speziellen Ausstellungs-Modus geschaltet werden.

Definitionen und Standard-Vorgaben:

Zeiteinheit:

Die Delay-Werte im Programmcode werden nicht in einer festen Zeiteinheit, wie z.B. Sekunden oder Millisekunden, angegeben, sondern in sogenannten *"Ticks".* Wenn nichts Anderes definiert ist, gilt als

Standardwert: 1 Tick = 10 Millisekunden.

Mit dem DEFTICKLEN-Befehl kann aber ggf. auch eine andere Tick-Länge eingestellt werden.

Steuerung der Helligkeit (SETBRIGHT):

Bei der Steuerung der Lichtstärke (z.B. über PWM-Modulation) gilt im Allgemeinen für den Parameter-Wert:
128 = **volle Intensität (Y=1)**.
Kleinere Werte verringern die Helligkeit linear, d.h. z.B. 32 entspricht Y=0,25.
Mit dem Befehl DEFBRIGHT kann ggf. noch ein Faktor **X** definiert werden, mit dem die Intensitätswerte für den jeweiligen Ausgang multipliziert werden. Auch hier gilt die Zahl **128** als Faktor **X=1,0**.
Für die Intensität am Ausgang gilt:
YAusg.= X * Y

Wenn die Ausgangshelligkeit nicht steuerbar ist, gilt das höchstwertige Bit des Helligkeits-Parameters als Ein-/Ausschalter (>= 0x80: Licht an, <= 0x7F: Licht aus).

Programmspeicher:

Die ersten Speicherzellen ab Adresse 0 bilden eine Tabelle mit den Einsprungadressen für die vorhandenen Ausgangskanäle. Abhängig von der Implementation (8 oder 16 Bit Adressen), besteht ein Eintrag aus einem oder zwei Bytes.

Die ALIC-Programmteile für die einzelnen Ausgangskanäle können beliebige Adressen des Programmspeichers belegen und können auch in beliebiger Reihenfolge angeordnet sein.

Steuerschalter (Switches):

Es können maximal 256 logische Steuerschalter vorhanden sein. Jeder Schalter kann die Zustände "on" und "off" annehmen. Die ersten Switches (ab Nummer 0) sind standardmäßig jeweils den Ausgangskanälen mit der gleichen Nummer zugeordnet.

Steuerschalter mit Nummern über der Anzahl der Ausgangskanäle sind optional und können Implementations-abhängig folgende Funktionen haben:

- frei verfügbar: als Zustands-Speicherelemente und zur Kommunikation zwischen den Steuerprogrammen verschiedener Ausgänge
- zur Kommunikation zwischen Interface-Firmware (z.B. RC-Multiswitch) und ALIC-Programmen
- zur Abfrage oder Ausgabe von Logikpegeln an den Prozessor-Pins (Ports). Hier gilt "on" als log. high-Pegel, "off" als low-Pegel.

Die Belegung der optionalen Steuerschalter wird dem Anwender durch den Programmierer der Firmware zur Verfügung gestellt.

Steuervariablen:

Die Steuervariablen sind optional und können Werte zwischen 0 und 255 annehmen. Es können bis zu 256 Variablen vorhanden sein, deren Belegung dem Anwender durch den Programmierer der Firmware zur Verfügung gestellt wird. Standardmäßig sind die ersten Variablen ab Adresse 0 jeweils den Ausgangskanälen mit der gleichen Nummer zugeordnet.

Regeln zum Aufbau des ALIC Assembler Quelltextes

Der ALIC Quelltext besteht aus nur wenigen Elementen, die im Folgenden beschrieben werden:

Zahlenwerte

Zahlenwerte können als Dezimalzahlen oder als Hexadezimalzahlen angegeben werden, auch gemischt innerhalb eines Ausdrucks. Hexadezimalzahlen werden durch ein vorangestelltes **0x** gekennzeichnet.Die folgenden Ausdrücke repräsentieren z.B. alle den gleichen Wert (=128):

0x80
128
64 + 0x40

ALIC Befehle

Die Befehlszeile besteht aus dem Befehlskürzel ("Mnemonic") und ggf. für den Befehl notwendigen Parametern. Das Mnemonic muß in Großbuchstaben geschrieben sein und muß mindestens um ein Leerzeichen oder eine Tab-Position ab Anfang der Zeile eingerückt sein. Wenn Parameter vorhanden sind, werden diese durch Kommata getrennt nach dem Befehlskürzel aufgelistet.

Beispiel:

```
        SMOOTHOFF
        SETBRIGHT 128,50
```

Kommentare

Kommentare werden durch ein vorangestelltes Semikolon gekennzeichnet. Alles nach dem Semikolon bis zum Ende der Zeile wird als Kommentar behandelt und vom Assembler nicht weiter verarbeitet. Kommentare können in jeder beliebigen Spalte beginnen.

Beispiel:

```
;************** Dies ist ein Beispiel:
        SMOOTHOFF ;Smooth Modus ausschalten
        SETBRIGHT 128,50 ;jetzt wird's plötzlich ganz hell
```

Labels

Mit Labels können Programmstellen mit einem Namen bezeichnet werden, um diesen dann z.B. in Sprungbefehlen verwenden zu können. Dem Label wird die Speicheradresse des unmittelbar folgenden Befehls zugewiesen. Labels müssen in der ersten Spalte stehen und dürfen Groß- und Kleinbuchstaben, sowie Ziffern und Unterstrich-Zeichen enthalten. Labels können in einer separaten Zeile vor dem Programmbefehl stehen oder gemeinsam mit dem Befehl in einer Zeile.

Es können für die gleiche Programmstelle auch verschiedene Namen vergeben werden, indem mehrere Zeilen mit Labels vor die Programmstelle gesetzt werden.

Beispiel:

```
entry_ch1 ;Programm für Channel 1 beginnt hier,
entry_ch2 ;für Channel 2 ebenfalls
Blinker LIGHTON 50 ;Licht einschalten, 0,5s lang
        LIGHTOFF 50 ;Licht ausschalten, auch 0,5s lang
        JUMP Blinker ;zurück zum Anfang der Sequenz
```

 # defines

Damit werden konstante Werte definiert, die im Programmtext unter dem angegebenen Namen aufgerufen werden können. Die Syntax lautet:

#define <Zeichenkette1> <Zeichenkette2>
Im weiteren Programmtext wird bei der Verarbeitung durch den Assembler dann Zeichenkette1 durch Zeichenkette2 ersetzt.

Beispiel:

```
#define vier 2+2
#define Blinkzeit 50 ;Zeit = 50/100 Sekunden
Blinker LIGHTON Blinkzeit ;Licht einschalten, Dauer = Blinkzeit
 LIGHTOFF Blinkzeit+vier ;Licht ausschalten, ein wenig länger
 JUMP Blinker ;zurück zum Anfang der Sequenz
```

Anpassung der Firmware:

Über die "#define ..."-Zeilen in der Datei **picalic.asm** wird die Übersetzung des Codes gesteuert und so kann der Controller auf die jeweiligen Erfordernisse angepasst werden.

Die Bedeutung der einzelnen Parameter:

#define ALICFILE <File>

Angabe der Datei, die den ALIC-Quelltext für den Lichtsteuercode enthält.

#define CLOCK <4|8>

CPU Taktfrequenz (4 or 8 MHz).
8 MHz ist ohne externe Takterzeugung nur beim PIC16F636 möglich.
Eine höhere Taktfrequenz ermöglicht eine höhere PWM Auflösung und/oder höhere PWM-Frequenz.

#define NCHAN <1..8>

Anzahl der Ausgangskanäle (1..8)
Beim PIC12F675 sind maximal 5 Kanäle möglich.

#define ACTIVELOW <0|1>

LED-Ausgänge sind invertiert wenn dieser Parameter = 1, d.h. der Controller-Ausgang nimmt Low-Pegel (ca. 0V) an, um die LED einzuschalten. Verwendung hauptsächlich für Direktansteuerung der LEDs durch den Controller, da der Ausgang bei "Low" mehr Strom liefern kann.

#define AUXOUTA <Ausgänge>

definiert freie RA-Ports als Ausgang, wenn das entsprechende Bit (0..7) auf "1" gesetzt wird. Diese können dann durch die ALIC-Befehle"SETSWX" und "CLRSWX" zur Ausgabe genutzt werden.
Auf Ports, die bereits durch Ausgangs-Kanäle oder für RC-Eingangsimpulse belegt sind, hat dieser Parameter keinen Einfluss.
Port RA3 kann bei PIC16F630 und PIC16F636 nicht als Ausgang verwendet werden.
Port RA5 kann bei PIC16F627, PIC16F628 und PIC16F648 nicht als Ausgang verwendet werden.

#define AUXOUTB <Ausgänge>

definiert freie RB-Ports als Ausgang, wenn das entsprechende Bit (0..7) auf "1" gesetzt wird. Diese können dann durch die ALIC-Befehle"SETSWX" und "CLRSWX" zur Ausgabe genutzt werden.
Auf Ports, die bereits durch Ausgangs-Kanäle oder für den RC-Eingang belegt sind, hat dieser Parameter keinen Einfluss.

#define AUXOUTC <Ausgänge>

definiert freie RC-Ports (bzw. GP beim PIC12F675) als Ausgang, wenn das entsprechende Bit (0..5) auf "1" gesetzt wird. Diese können dann durch die ALIC-Befehle "SETSWX" und "CLRSWX" zur Ausgabe genutzt werden.

Auf Ports, die bereits durch Ausgangs-Kanäle oder für RC-Eingangsimpulse belegt sind, hat dieser Parameter keinen Einfluss.

PIC12F675: Port GP3 kann nicht als Ausgang geschaltet werden. Anmerkung: "RC" bei der Bezeichnung der Prozessor-Ports hat nichts mit "Radio Control" zu tun!

#define COMMTICK <0|1>

Wenn dieser Parameter auf "1" steht, ist die Zeiteinheit (Tick) für alle Ausgangskanäle fest auf 10 ms eingestellt. "DEFTICKLEN" wird dann ignoriert (spart RAM im Controller).

#define PWMTU <Zeit>

definiert die Länge der PWM-Zeiteinheit in Microsekunden. In Verbindung mit dem Parameter PWMCYC ergibt sich daraus die Länge eines PWM-Zyklus (T) und daraus die PWM-Frequenz (1/T). Empfohlener Wert: 100 bei 4MHz Takt, 50 bei 8 MHz.

#define PWMCYC <Schritte>

gibt die Anzahl der Schritte (* PWMTU) in einem vollen PWM-Zyklus, und damit auch die Helligkeits-Auflösung, sowie zusammen mit PWMTU die PWM-Frequenz.

#define RCCHAN <0|1|2>

definiert die Anzahl der vorhandenen RadioControl-Eingänge (0, 1 oder 2).

#define RCHYST <Wert>

Hysterese für RC-Prozessing (* 8 µs).

#define STARTSW <bits>

Setzt einen Start-Status für die RC-Memory Schalter (Nummer 15..8).

#define RCPW_HIGH <Schwelle>

RC-Impuls Schwellwert (90: ca. 1,7ms) für Betätigung des Memory-Schalters
64 = 1,5ms, 1 Schritt = 8 µs

#define RCPW_LOW <Schwelle>

RC-Impuls Schwellwert (39: ca. 1,3ms) für die andere Richtung des Schalters.

#define MS_LONG <Zeit>

min. Haltezeit des Knüppels in der Endstellung für Betätigung der Memory-Switch Funktionen 3 und 4 (Zeit in 1/10 s, max.15 = 1,5s möglich)

Beispiele und Anwendungen

Einfache Beispiele:

```
entry_ch0
        SETBRIGHT       0,100           ;Helligkeit = 0 für eine Sekunde
        SETBRIGHT       40,50           ;Helligkeit ca.1/3, 0,5 Sekunden lang
        SETBRIGHT       80,50           ;Helligkeit ca.2/3, 0,5 Sekunden lang
        SETBRIGHT       128,100         ;volle Helligkeit, eine Sekunde lang
        P_RESET                         ;dann Wiederholung...
```

```
entry_ch0
        LIGHTOFF        150             ;Licht aus für 1,5 Sekunden
        LIGHTON         10              ;Licht an für 0,1 Sekunden
        JUMP            entry_ch0       ;dann Wiederholung...
```

```
entry_ch0
        SMOOTHON                        ;Helligkeit kontinuierlich ändern
        SETBRIGHT       0,100           ;Start bei H. = 0, 1 Sek. bis nächster Wert
        SETBRIGHT       128,0           ;= maximale Helligkeit, dann sofort
        SMOOTHOFF                       ;"Smooth" ausschalten
        SETBRIGHT       0,100           ;Sprung auf Helligkeit 0, 1 Sekunde
        P_RESET                         ;dann Wiederholung...
```

Licht mit Auf- und Abglimm-Effekt, gesteuert über Schalter:

```
entry_ch0
        SMOOTHON                        ;Helligkeit kontinuierlich ändern
        HALTSWOFF                       ;warten auch Einschalten
        SETBRIGHT       0,30            ;LED aus, in 0,3 Sek. bis zur
        SETBRIGHT       128,0           ;maximalen Helligkeit
        HALTSWON                        ;warten auf Ausschalten
        SETBRIGHT       128,10          ;in 100ms von "voll" auf
        SETBRIGHT       85,20           ;2/3, dann in 200ms auf
        SETBRIGHT       43,40           ;1/3, dann in 400ms auf
        SETBRIGHT       0,0             ;0
        P_RESET                         ;dann Wiederholung...
```

Abhängig von der Schalterstellung wird ein Einfach- bzw. **Doppelblitz** ausgegeben:

```
entry_ch0
        LIGHTOFF        150             ;Licht aus für 1,5 Sekunden
        LIGHTON         10              ;Licht an für 0,1 Sekunden
        JSWOFF          entry_ch0       ;wenn Schalter aus: nur einmal blitzen
        LIGHTOFF        20              ;2.Blitz: kurze Pause
        LIGHTON         10              ;Licht an für 0,1 Sekunden
        P_RESET                         ;dann Wiederholung...
```

Im Programm für einen Kanal werden über Steuerschalter andere Kanäle aktiviert und dadurch Vorgänge synchronisiert:

```
;Erster Kanal ("Master"): blinkt 1x auf, schaltet dann nacheinander die "Slaves" ein:
entry_ch0
        LIGHTOFF        120             ;Licht aus für 1,2 Sekunden
        LIGHTON         10              ;Licht an für 0,1 Sekunden
        SETSWX          1               ;dann zweiten Kanal aktivieren
        LIGHTOFF        10              ;0,1 s warten
        SETSWX          2               ;dritten Kanal aktivieren
        LIGHTOFF        10              ;wieder 0,1s warten
        SETSWX          3               ;vierten Kanal aktivieren
        P_RESET                         ;dann Wiederholung...
;Programm für die "Slaves": nur 1x aufblinken, sobald der Schalter
;(durch "Master") eingeschaltet wurde:
entry_ch1                               ;2. bis 4.Kanal = "Slaves"
entry_ch2                               ;(alle verarbeiten dasselbe Programm)
entry_ch3
        HALTSWOFF                       ;auf Einschalten warten
        CLRSW                           ;den eigenen Schalter wieder ausschalten!
        LIGHTON         10              ;0,1s an
        P_RESET                         ;Reset und wieder von vorn...
```

Jump Befehl, Unendliche Schleife:

```
entry_ch0
        LIGHTOFF        150             ;Licht aus für 1,5 Sekunden
        LIGHTON         10              ;Licht an für 0,1 Sekunden
        JUMP            entry_ch0       ;dann Wiederholung...
```

Simulation von startenden **Leuchtstoffröhren** mit Zufallswerten:

```
entry_ch0                       ;alle Lampen benutzen denselben Code!
entry_ch1
entry_ch2
entry_ch3
wait4on                         ;warten auf "Schalter ein":
        HALTSWOFF               ;bleibt "aus" solange switch auf "off"
        SETBRIGHT 0,30          ;aus lassen für min. 300ms
flicker
        RNDDLY  150             ;delay 0 bis 1,5s
        JSWOFF  wait4on         ;falls Schalter inzwischen wieder "aus": aus!
        RNDJMP  64,turn_on      ;etwa 50% Chance für jetzt Anspringen!
        RNDBRT  10,40,2         ;min. 20 ms Aufblitzen bei niedriger Helligkeit
        RNDDLY  20              ;delay bis 200 ms
        SETBRIGHT 0,0           ;aus
        JUMP    flicker         ;neuer Startversuch
turn_on
        SETBRIGHT 128,0         ;"Ein" -> volle Helligkeit
        HALTSWON                ;bleibt an, solange switch = "on"
        P_RESET
```

Ein- und Ausschalten des **Blitzers** über Schalter 8,
Abhängig von der Schalterstellung von Schalter 9 wird ein Einfach- bzw. Doppelblitz ausgegeben:

```
entry_ch0
        DEFSWITCH       8               ;Schalter 8 = Standard-Schalter für diesen Kanal
schleife
        LIGHTOFF        150             ;Licht aus für 1,5 Sekunden
        HALTSWOFF                       ;Halt, wenn Schalter 8 auf "aus" steht
        LIGHTON         10              ;Licht an für 0,1 Sekunden
        JSWXOFF         9,schleife      ;wenn Schalter 9 aus: nur einmal blitzen
        LIGHTOFF        20              ;2.Blitz: kurze Pause
        LIGHTON         10              ;Licht an für 0,1 Sekunden
        JUMP            schleife        ;dann Wiederholung...
```

Der Helligkeits-Level für den Kanal wird hier mit etwa 1,28 multipliziert, dadurch wird mit dem Wert 100 als SETBRIGHT-Parameter bereits die volle Helligkeit erreicht, und Werte <100 entsprechen der Helligkeitsangabe in Prozent.

```
entry_ch0
        DEFBRIGHT       0xA4            ;Helligkeits-Faktor = ca. * 1,28
        SETBRIGHT       0,200           ;Helligkeit = 0 für zwei Sekunden
        SETBRIGHT       50,200          ;50% Helligkeit, 2 Sekunden lang
        SETBRIGHT       100,200         ;volle Helligkeit, 2 Sekunden lang
        P_RESET
```

Licht mit Auf- und Abglimm-Effekt, gesteuert über Schalter:

```
entry_ch0
        DEFSWITCH       8               ;Definiert Schalter 8 für diesen Kanal
        SMOOTHON                        ;Helligkeit kontinuierlich ändern
        HALTSWOFF                       ;warten auch Einschalten
        SETBRIGHT       0,30            ;LED aus, in 0,3 Sek. bis zur
        SETBRIGHT       128,0           ;maximalen Helligkeit
        HALTSWON                        ;warten auf Ausschalten
        SETBRIGHT       128,10          ;in 100ms von "voll" auf
        SETBRIGHT       85,20           ;2/3, dann in 200ms auf
        SETBRIGHT       43,40           ;1/3, dann in 400ms auf
        SETBRIGHT       0,0             ;0
        P_RESET                         ;dann Wiederholung...
```

Steuerung durch Variablen-Wert. Es wird angenommen, daß der Schieberwert zwischen 0 und 128 liegt und in der Steuervariablen Nummer 1 zur Verfügung gestellt wird.

```
;Programm für erste LED:

entry_ch3
        DEFVAR  1                       ;Schieber(1) diesem Kanal zuordnen
loop_ch3
        JVARAE  40,ch3_on               ;Licht an, wenn Wert >= 40
        P_RESET                         ;sonst Licht aus, zurück auf Anfang ch3_on
        LIGHTON 0                       ;Licht an
        JUMP    loop_ch3                ;Variable erneut abfragen

;Programm für zweite LED:
;macht dasselbe, wie für ch3, nur mit anderem Variablenwert (80).
;Zu Demozwecken auf etwas andere Art programmiert:

entry_ch4
        JVARXB  1,80,ch4_off            ;Schieber(1) < 80: Licht aus
        LIGHTON 0                       ;sonst Licht an
        JUMP    entry_ch4               ;mit JUMP zurück (P_RESET würde Licht ausschalten!)
ch4_off
        P_RESET                         ;Licht aus und von vorn...
```

Afterburner Effekt für Turbine Flieger, 3 Kanäle mit Random Flieckern, ein Drehlicht Effekt und Blinker (Konstante Time):

```
#define TIME 25                  ; Zeitvariabel
#define IMPULS_BURNER 2          ; Zeitwert der Afterburner Effekt
#define MIN_BURNER 2             ; Minimaler Wert der Beleuchtung des RANDOM Befehl

entry_ch0
       DEFVAR 0                                    ; RC Eingang 0 als Steuerung
       JVARB 63,light_off_all_entry0               ; Unter der Helfter des Knuppel kein Effekt
       JVARB 74, entry0_step1                      ; 4 Effektstuffen
       JVARB 94, entry0_step2
       JVARB 115, entry0_step3
       RNDBRT MIN_BURNER,128,IMPULS_BURNER         ; Schalte die LED's an mit Fliekern Effekt
       JUMP entry_ch0
entry0_step1
       RNDBRT MIN_BURNER,64,IMPULS_BURNER          ; Je Wert andere Helligkeit der LED's
       JUMP entry_ch0
entry0_step2
       RNDBRT MIN_BURNER,94,IMPULS_BURNER
       JUMP entry_ch0
entry0_step3
       RNDBRT MIN_BURNER,115,IMPULS_BURNER
       JUMP entry_ch0
light_off_all_entry0
       P_RESET

entry_ch1                                          ; Ab circa 75% zweite LED Stuffen an
       DEFVAR 0                                    ; Gleicher Befehlgeber wie Kanal 0
       JVARB 93,light_off_all_entry1
       JVARB 100, entry1_step1
       JVARB 115, entry1_step2
       RNDBRT MIN_BURNER,128,IMPULS_BURNER         ; LED sind niemals aus, Stuffe 3
       JUMP entry_ch1                              ; mit dem Wert MIN_BURNER (hier 30ms)
entry1_step1
       RNDBRT MIN_BURNER,94,IMPULS_BURNER          ; Stuffe 1 der zweite Ringfarbe
       JUMP entry_ch1
entry1_step2
       RNDBRT MIN_BURNER,115,IMPULS_BURNER         ; Stuffe 2
       JUMP entry_ch1
light_off_all_entry1
       P_RESET

entry_ch2                                          ; Ab 85% tritte Stuffe an (zB Blaue LED's)
       DEFVAR 0
       JVARB 110,light_off_all_entry2
       JVARB 118, entry2_step1
       RNDBRT MIN_BURNER,128,IMPULS_BURNER         ; Stuffe 2
       JUMP entry_ch2
entry2_step1
       RNDBRT MIN_BURNER,110,IMPULS_BURNER         ; Stuffe 1
       JUMP entry_ch2
light_off_all_entry2
       P_RESET

entry_ch3                                          ; Drehlicht Simulation
       SMOOTHON
       SETBRIGHT 0,16
       SETBRIGHT 16,3
       SETBRIGHT 128,2
       SETBRIGHT 128,3
       SETBRIGHT 16,16
       SETBRIGHT 0,50
       P_RESET

entry_ch4                                          ; Doppelter Impulsblinker
       LIGHTOFF TIME+TIME+TIME+TIME                ; Beispiel Benutzung Konstanten
       LIGHTON 3
       LIGHTOFF TIME
       LIGHTON 3
       P_RESET
```

Beispiel für **Drehlicht**. Der Kanal 0 (entry_ch0) ist der Master für alle andere Slave-Kanäle:

```
#define TIME 100              ;Die Sequenz arbeitet mit 1 Sekunde pro Kanal

entry_ch0
  SMOOTHON                    ;Helligkeit kontinuierlich ändern
  LIGHTOFF TIME               ;Licht aus mit LaufZeit des Time Wert (1 Sekunde)
  LIGHTON 0
  SETSW 1                     ;Aktiviere Kanal 1
  LIGHTON TIME
  LIGHTOFF 0
  SETSW 2                     ;Aktiviere Kanal 2
  DELAY TIME                  ;Warte 1000ms (Time = 100)
  SETSW 3                     ;Aktiviere Kanal 3
  DELAY TIME
  SETSW 4                     ;Aktiviere Kanal 4
  DELAY TIME
  JUMP entry_ch0              ;Starte am Anfang an

entry_ch1                     ;Alle 4 Ausgänge als Slave von Kanal 0
entry_ch2
entry_ch3
entry_ch4
  HALTSWOFF
  CLRSW
  SMOOTHON
  LIGHTOFF TIME
  LIGHTON TIME
  LIGHTOFF 0
  P_RESET
```

Yachtbeleuchtung mit Glühlampen-Simulation:

```
#define pause 50            ;Verzögerung
#define light_up 40         ;Zeit für's Aufleuchten (Wert muss kleiner sein, als "pause"!)
#define light_down 15       ;Zeitkonstante für's Ausglimmen

entry_ch4
        DELAY pause         ;Jeder Kanal schaltet sich in der Reihenfolge an
entry_ch3
        DELAY pause
entry_ch2
        DELAY pause
entry_ch1
        DELAY pause
entry_ch0
        SMOOTHON
        LIGHTOFF    light_up        ;Aufleuchten der LED
        LIGHTON 0
        SETSW                       ;Switch zeigt an, dass diese LED nun an ist
wait_ch4
        JSWXOFF 4,wait_ch4          ;wartet, bis die letzte LED (= ch 4) an ist
        DELAY pause
        SETBRIGHT 128,light_down    ;Ausglimm-Sequenz...
        SETBRIGHT 32,light_down
        SETBRIGHT 8,light_down
        SETBRIGHT 2, 2 * light_down
        SETBRIGHT 0,0
        CLRSW                       ;Switch zeigt an: diese LED ist nun aus!
        P_RESET
```

Pyro-Steuerung, die Ausgänge werden in der Reihenfolge per Taste aktiviert (RC Memory Switch) :

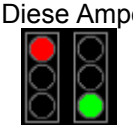
```
#define PULSLEN 5                   ;Impulslänge 5 Ticks = 50 ms

; Es mag sein das Sie die MS_LONG für Schalter 11 reduzieren können auf den Wert 3

entry_ch0
    DEFSWITCH 11                    ;Memory-Switch für die Steuerung
    HALTSWOFF                       ;warten auf Betätigung (off->on)
    LIGHTON PULSLEN                 ;ersten Kanal abfeuern
    LIGHTOFF 0
    HALTSWON                        ;warten auf nächste Betätigung (on->off)
    SETSWX 1                        ;nächsten Kanal abfeuern
    HALTSWOFF                       ;warten auf nächste Betätigung (off->on)
    SETSWX 2                        ;nächsten Kanal abfeuern
    HALTSWON                        ;warten auf nächste Betätigung (on->off)
    SETSWX 3                        ;nächsten Kanal abfeuern
    HALTSWOFF                       ;warten auf nächste Betätigung (off->on)
    SETSWX 4                        ;nächsten Kanal abfeuern
    HALTSWON                        ;warten auf nächste Betätigung (on->off)
    SETSWX 5                        ;nächsten Kanal abfeuern
    HALTSWOFF                       ;warten auf nächste Betätigung (off->on)
    SETSWX 6                        ;nächsten Kanal abfeuern
    HALTSWON                        ;warten auf nächste Betätigung (on->off)
    SETSWX 7                        ;nächsten Kanal abfeuern
    P_RESET                         ;ggf. das Ganze von vorn...

entry_ch1
entry_ch2
entry_ch3
entry_ch4
entry_ch5
entry_ch6
entry_ch7
   HALTSWOFF                        ;warten auf Aktivierung
   CLRSW
   LIGHTON PULSLEN                  ;Pyro abfeuern
   P_RESET
```

ALIC Anwendungsbeispiel: Einfache Ampelanlage



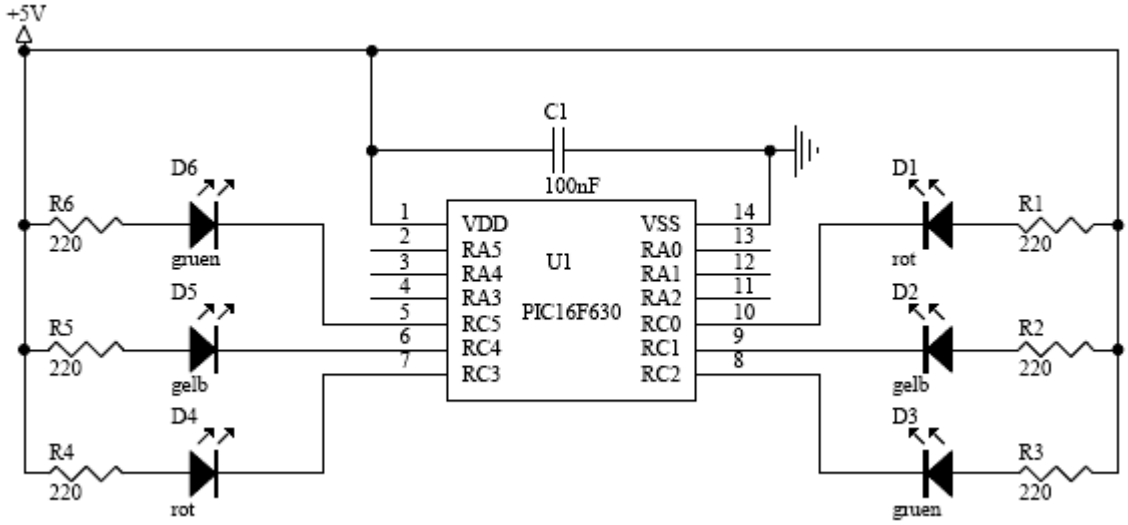
Diese Ampelanlage besteht aus zwei Ampeln mit jeweils einer roten, einer gelben und einer grünen LED, die vom Controller automatisch in der Abfolge der Ampelphasen angesteuert werden. Das Steuerprogramm für das Rotlicht der ersten Ampel (Ausgang 0) bildet dabei die zentrale Steuersequenz für alle Lampen. Die anderen Lampen werden durch SETSWX- und CLRSWX-Befehle über ihre jeweils zugeordneten logischen Schalter zu den passenden Zeiten ein- und ausgeschaltet. Die Steuerprogramme für alle anderen LEDs, als "rot" der ersten Ampel, beschränken sich daher auf die einfache Ein-/Ausschalt-Funktion und sind für alle diese LEDs gleich, daher kann für die Ausgänge 1 bis 5 ein gemeinsames Steuerprogramm verwendet werden.

ALIC-Steuerprogramm für die Ampelanlage:

```
;Schalter/Kanalnummern der Ampel-Lampen:
#define rot_1   0
#define gelb_1  1
#define gruen_1 2
#define rot_2   3
#define gelb_2  4
#define gruen_2 5
;Kanal 0 = Rotlicht der ersten Ampel, steuert die anderen Lichter:
entry_ch0
        DEFTICKLEN      100             ;Zeiteinheit = 100ms
;
ampelschleife
        SETSWX          gruen_2         ;Ampel 2 auf grün
        LIGHTON         100             ;Ampel1 rot/Ampel 2 grün = 10 Sekunden
        CLRSWX          gruen_2         ;Ambel2 grün aus,
        SETSWX          gelb_2          ;gelb an
        DELAY           40              ;Gelbphase(A.2) dauert 4 Sekunden
        CLRSWX          gelb_2
        SETSWX          rot_2           ;Ampel2 auf rot
        DELAY           40              ;beide Ampeln = rot, 4 Sekunden
        SETSWX          gelb_1          ;Ampel1 = rot+gelb
        DELAY           20              ;rot/gelb-Phase =2 Sekunden
        CLRSWX          gelb_1          ;gelb aus
        SETSWX          gruen_1         ;grün an
        LIGHTOFF        100             ;Ampel 1 Grünphase = 10 Sekunden
        CLRSWX          gruen_1
        SETSWX          gelb_1          ;Ampel 1 Gelbphase
        DELAY           40              ;dauert 4 Sekunden
        CLRSWX          gelb_1          ;gelb aus,
        LIGHTON         40              ;beide Ampeln rot: 4 Sekunden
        SETSWX          gelb_2          ;Ampel 2: rot+gelb
        DELAY           20              ;2 Sekunden lang
        CLRSWX          rot_2
        CLRSWX          gelb_2
        JUMP            ampelschleife   ;dann Wiederholung... (Grünphase v.Ampel2)
;
entry_ch1
entry_ch2
entry_ch3
entry_ch4
entry_ch5
        HALTSWOFF                       ;alle Slave-Kanäle = simple Ein/Aus-Schalter
        LIGHTON         0
        HALTSWON
        P_RESET
```

Ampelanlage Hardware:

Solche Signallichter, z.B. auf Modellbahn-Anlagen im H0-Maßstab, brauchen keine großen Leistungen, so daß die LEDs über Vorwiderstände direkt an den Microcontroller angeschlossen werden können. Die entsprechende Schaltung könnte z.B. mit einem PIC16F630 dann so aussehen:



ALIC Anwendungsbeispiel: Beleuchtung für ein Flugmodell

Funktionen:

- **Landescheinwerfer** mit Auf- und Abglimm-Effekt beim Ein-/Ausschalten, um Halogenlampen zu simulieren.
- **Positionslampen** - einfache Ein-/Ausschalt-Funktion.
- **ACL-Blitzer** über zwei Ausgänge: Doppelblitz für Blitzer an den Tragflächen, dazu synchron ein Einfachblitz für das Heck auf dem anderen Kanal.
- **Beacons** - zwei simulierte Drehlichter mit leicht unterschiedlicher "Drehzahl". Diese bestehen aus jeweils nur einer (High-Power-)LED, die durch entsprechende Helligkeitssteuerung den Eindruck einer Lampe mit rotierendem Reflektor macht, wenn man es aus einiger Entfernung betrachtet.

Die einzelnen Funktionen können über die Fernsteuerung ein- und ausgeschaltet werden. Je nach Implementierung des Interfaces im Controller kann das über ein Multiswitch-Modul oder eine Memory-Switch Funktion ausgeführt werden.

```
;*************************************************************
;ALIC Source Code:
;Beleuchtung für Flugmodell
;*************************************************************
;Belegung der Augangskanäle:
;Ch.0 = ACL (Flaechenspitzen, Doppelblitz)
;Ch.1 = ACL (Rumpfende, Einfachblitz)
;Ch.2 = Landescheinwerfer
;Ch.3 = Positionslichter
;Ch.4 = Beacon (Rumpfboden)
;Ch.5 = Beacon (auf Leitwerk)
;Zuordnung der Schalter (z.B. RC-Memoryswitch oder Multiswitch-Modul):
#define sw_LandingLight 8       ;Schalter für Landescheinwerfer
#define sw_ACL          10      ;Schalter für ACL
#define sw_PosLights    11      ;Schalter für Positionslampen
#define sw_Beacons      10      ;Schalter für Beacons

;Code für ACL: Doppelblitz, löst synchron auch Blitz am Heck aus

entry_ch0
        DEFSWITCH       sw_ACL  ;Schalter zuordnen
        HALTSWOFF
        LIGHTON         3       ;Blitz: 30 ms
        SETSWX          1       ;löst Blitz am Heck aus
        LIGHTOFF        13      ;Pause 130 ms
        LIGHTON         3       ;2. Blitz
        LIGHTOFF        130     ;1,3 s Pause
        P_RESET
;Code für Einfach-Blitz:
entry_ch1
        HALTSWOFF               ;warten auf Auslösung
        CLRSW                   ;Schalter wieder zurücksetzen
        LIGHTON         3       ;Blitz!
        P_RESET
;Lande-Scheinwerfer: schalten mit Glühlampen-Simulation
entry_ch2
        DEFSWITCH       sw_LandingLights
        HALTSWOFF
        SMOOTHON
        LIGHTOFF        40      ;Aufglimmen = 0,4s
        LIGHTON         0
        HALTSWON                ;warten, bis Ausschalten
        SETBRIGHT       128,14  ;Ausglimm-Sequenz...
        SETBRIGHT       32,14
        SETBRIGHT       8,14
        SETBRIGHT       2,20
```

```
        SETBRIGHT       0,0
        P_RESET

;Positionslichter: einfacher Ein-/Ausschalter
entry_ch3
        DEFSWITCH       sw_PosLights
        HALTSWOFF
        LIGHTON         0
        HALTSWON
        P_RESET
;Simuliert Beacon/Drehlicht
entry_ch4
        DEFSWITCH       sw_Beacons
        SMOOTHON
        HALTSWOFF
        SETBRIGHT       0,16    ;in 160 ms auf 16
        SETBRIGHT       16,3    ;in 30 ms auf volle Helligkeit
        SETBRIGHT       128,2   ;volle H. 20 ms halten
        SETBRIGHT       128,3   ;dann in 30 ms zurück auf 16
        SETBRIGHT       16,16
        SETBRIGHT       0,45    ;450 ms Ausphase
        P_RESET
;wie Ch.4, aber etwas andere Drehfrequenz
entry_ch5
        DEFSWITCH       sw_Beacons
        SMOOTHON
        HALTSWOFF
        SETBRIGHT       0,16    ;in 160 ms auf 16
        SETBRIGHT       16,3    ;in 30 ms auf volle Helligkeit
        SETBRIGHT       128,2   ;volle H. 20 ms halten
        SETBRIGHT       128,3   ;dann in 30 ms zurück auf 16
        SETBRIGHT       16,16
        SETBRIGHT       0,50    ;500 ms Ausphase
        P_RESET
```

Hardware: Ansteuerung der LEDs

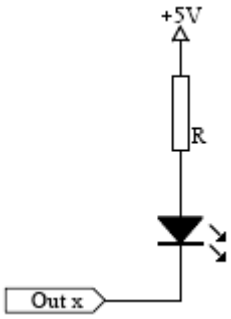
In diesem Abschnitt werden einige Möglichkeiten zur Ansteuerung der LEDs durch die Ausgänge des Microcontrollers beschrieben. Die LEDs können über geeignete Treiberschaltungen durch eine höhere Spannung gespeist werden, als der Microcontroller. Diese Betriebsspannung ist mit "+U" bezeichnet. Das Erdungs-Symbol ⏚ bedeutet "Masse" (0V) und ist im Allgemeinen mit dem Minus-Pol der Spannungsquelle verbunden. Die Masse-Anschlüsse im gesamten System müssen verbunden sein, außer bei Schaltungen mit galvanischer Trennung zwischen Last- und Steuerkreis (dazu später...). Die in den Prinzip-Schaltungen gezeigten LED-Symbole können als Einzel-LEDs, aber auch als Serienschaltung aus mehreren LEDs, sowie als Parallelschaltung mehrerer LED-Zweige interpretiert werden. Bei Parallelschaltung bitte beachten:

LEDs sollten nicht direkt parallel geschaltet werden**, sondern mit jeweils einem eigenen Vorwiderstand für jeden Zweig.**

Aufbau:
Da die Schaltung nur aus wenigen Bauteilen besteht, bietet sich für "Einzelstücke" der Aufbau auf Lochraster-Platine mit Lötpunkten an. Selbst MOSFETs im SO-8 SMD-Gehäuse lassen sich darauf montieren und mit dünnen Drähtchen kontaktieren, wenn man einen Lötkolben mit dünner Spitze besitzt und nicht gerade zu den absoluten "Grobmotorikern" gehört. Wer es "schön" machen will, kann aber selbstverständlich auch eine Platine entwerfen.

Direktanschluss

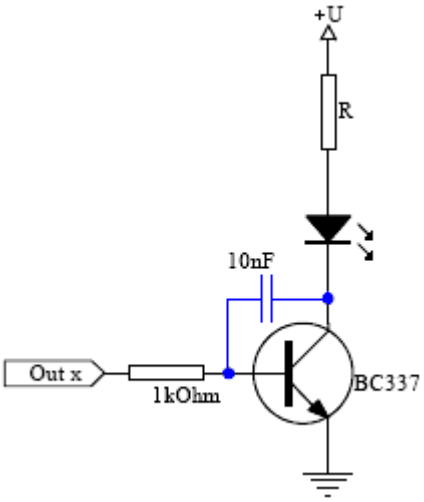
Dies ist die einfachste Art, eine LED an den Controller anzuschließen. Der Ausgang ist in diesem Fall Low-Aktiv, d.h. die LED leuchtet, wenn der Controller 0-Pegel ausgibt. Da die Ausgänge des Microcontrollers nur relativ kleine Ströme liefern können, eignet sich diese Schaltung nur für die Ansteuerung von LEDs mit kleiner Leistung. Die Ausgänge der hier verwendeten PICs sollten nicht mit mehr als 10-15 mA belastet werden, also ergeben sich für den Vorwiderstand (R) Werte von 220 Ohm oder höher. Wegen der geringen Betriebsspannung kommt eine LED Serienschaltung hier in der Regel nicht in Betracht. Eine Parallelschaltung von mehreren LEDs, die jeweils mit nur wenig Strom betrieben werden, ist jedoch möglich (eigener Vorwiderstand für jede LED!).



Bipolarer Transistor als Schalter

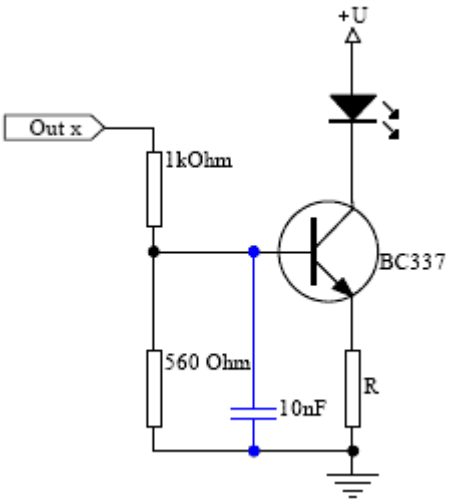
Bei LED-Strömen bis zu einigen hundert Milliampere, oder wenn +U größer ist, als 5V, kann ein Transistor eingesetzt werden. Durch Serienschaltung der LEDs und/oder Parallelschaltung mehrerer LED-Zweige können hier auch viele LEDs auf einmal geschaltet werden. Der hier verwendete Transistor BC337 kann Ströme bis 0,8A und Spannungen bis 45V schalten.

Der optionale Kondensator dient dazu, die steilen Schaltflanken abzuflachen. Dadurch werden die beim Schaltvorgang erzeugten Anteile von Hochfrequenz auf den Anschlußkabeln deutlich reduziert. Bei PWM-Betrieb sollte der Kondensator eingebaut werden, um evtl. EMV-Problemen (Störung der RC-Anlage!) vorzubeugen. Wenn aber nur einzelne Schaltvorgänge stattfinden, ohne PWM-Modulation des LED-Stromes, kann der Kondensator auch weggelassen werden.



Bipolarer Transistor als Konstantstromquelle

Wird die LED über einen simplen Vorwiderstand aus einem Akku oder einer Batterie gespeist, hängt der LED-Strom auch von der Batteriespannung ab. Damit sinkt die Helligkeit der LED bei fortschreitender Entladung der Batterie, und bei Modellen, bei denen die LEDs vom Antriebsakku mit versorgt werden, kann es leicht zu unerwünschten Helligkeitsschwankungen in Abhängigkeit von der momentanen Stromaufnahme des Antriebs kommen. Diese Probleme können durch eine Konstantstrom-Schaltung umgangen werden. Bei der hier gezeigten Schaltung wird davon ausgegangen, daß der Controller mit einer stabilisierten Spannung von +5V versorgt wird, z.B. durch einen BEC-Schaltkreis von der RC-Empfangsanlage. Dann liegt im eingeschalteten Zustand an der Basis des Transistors eine Spannung von etwa 1,7V an. Der Wert für R sollte nun so gewählt werden, daß bei dem gewünschten LED-Strom über R 1V Spannung liegt. Also z.B., wenn durch die LED ein Strom von 100mA fließen soll, ergibt das



R = 1V/0,1A = 10 Ohm

Die Schaltung regelt den LED-Strom unabhängig von +U so ein, daß über R immer ca. 1V (= Basisspannung - Durchlasspannung der B-E Diodenstrecke) anliegt. Auch bei dieser Schaltung sollten für PWM-Betrieb ggf. durch den Einbau des Kondensators die hochfrequenten Signalanteile reduziert werden. Die Verlustleistung des Transistors sollte hier nicht außer Acht gelassen werden, da der Transistor hier nicht als Schalter, sondern "analog" arbeitet. Deshalb ist die Schaltung in der hier gezeigten Ausführung nur für kleinere Ströme (bis ca. 100mA) geeignet. Die Verlustleistung ergibt sich aus U * I, wobei die Spannung über dem Transistor ist:

$U_{Transistor} = U_{Batt} - U_{LED} - U_R$

Beispiel:

Der Akku hat im geladenen Zustand 14V, die LED besteht aus zwei in Serie geschalteten 4-Chip LEDs mit je:
U_f = 3,5V, I_f = 100mA. Über dem Transistor liegt dann eine Spannung von:

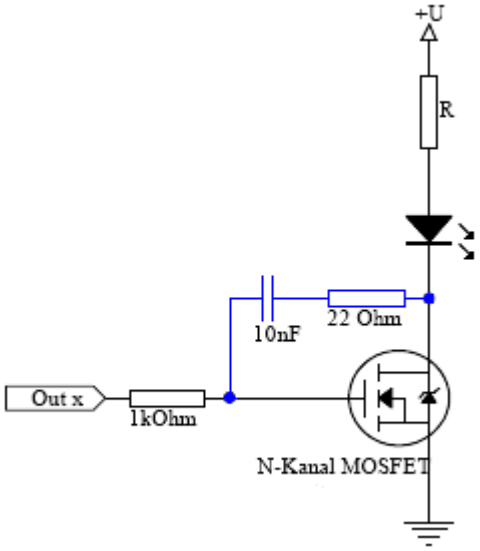
$U_{Transistor} = 14V - (2 * 3,5V) - 1V = 6V$

Der Transistor muß dann also U * I = 0,6W Leistung "verbraten". Das ist für diesen Transistor schon etwas grenzwertig, da er laut Datenblatt 625mW verbraten darf. Um den Transistor etwas zu entlasten, können wir noch einen Widerstand von 10 Ohm in Serie zu den LEDs schalten. An diesem fallen 10Ohm * 0,1A = 1V Spannung ab, wodurch sich die Spannung am Transistor nun auf 5V und die Verlustleistung auf 500mW verringert. Am Stromfluß durch die LEDs ändert sich durch den zusätzlichen Widerstand nichts!

An den LEDs incl. allen Widerständen fallen nun zusammen 9V ab. Die Akkuspannung sollte immer etwas über diesem Wert bleiben, da sonst der eingestellte Strom nicht mehr erreicht werden kann.

MOSFET als LED-Treiber

Zum Schalten größerer Leistungen bieten sich MOSFETs an. Auch hier wird durch den Kondensator die HF-Erzeugung reduziert, was besonders für PWM-Betrieb wichtig ist. Der 22-Ohm Widerstand hilft gegen Schwingneigungen im Mhz-Bereich.



Als MOSFETs eignen sich zum Beispiel:

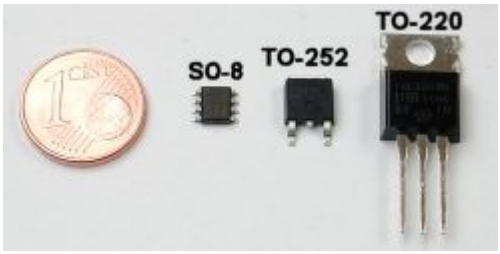
IRF7413: MOSFET im SO-8 Gehäuse, max. 9,2A, bis 30V, R_{DSon}= 0,02 Ohm (bei U_{GS}=4,5V)

IRF7103: zwei MOSFETs in einem SO-8 Gehäuse, jeweils max. 2,3A, bis 50V, R_{DSon}= 0,16 Ohm (bei U_{GS}=4,5V)

IRF7303: zwei MOSFETs in einem SO-8 Gehäuse, jeweils max. 3,9A, bis 30V, R_{DSon}= 0,08 Ohm (bei U_{GS}=4,5V)

IRLR024N: MOSFET im TO-252 Gehäuse, max. 12A, bis 55V, R_{DSon}= 0,11 Ohm (bei U_{GS}=4V)

IRL2203: MOSFET im TO-220 Gehäuse, max. 80A, bis 30V, R_{DSon}= 0,01 Ohm (bei U_{GS}=4,5V)und viele andere...



Transistor-Arrays als LED-Treiber

Für die Ansteuerung von mehreren LED-Kanälen mit kleiner bis mittlerer Leistung können auch entsprechende integrierte Schaltkreise verwendet werden, z.B. ULN2003 mit sieben Darlington Transistorschaltungen für max. 500mA/Kanal bei max. 50V, oder ULN2803 mit 8 solchen Treibern. Die Eingänge dieses ICs können direkt mit den Ausgängen des Microcontrollers verbunden werden, die Ausgänge schalten nach Masse (mit einer Restspannung von ca. 0,8 bis 1,5V durch die Darlington-Schaltung).