V15.2.8

LED-BASIC

KOMPONENTEN, EDITOR, BEFEHLSSATZ

Erwin Reuß und Folker Stange

LED-BASIC

Komponenten, Editor, Befehlssatz

Inhaltsverzeichnis

LED-Basic? 5
NixieCron© 5
Installation und Programmstart 6
Windows 10 und die Sicherheit 6
Automatische und manuelle Updates 6
LED-Basic Editor 7
Editor-Funktionen 7
Info-Fenster 9
Konfiguration 9
System 9
Komponenten 10
LED-Basic Grundbefehlssatz 12
Grundsätzliche Regeln für LED-Basic 12
Hinweise zur Benutzung von Variablen 12
Eingabe von Zahlwerten: 13
Arithmetische Operatoren: 13
Bit-Operatoren: 13
Vergleichs-Operatoren: 13
Logische-Operatoren: 13
Syntaxbeschreibung 14
REM 14
END 14
LET 14
FOR-NEXT-SCHLEIFE 14
IF-THEN-ELSE 15
GOTO 15
GOSUB/RETURN 15
RANDOM 15
DELAY 15
PRINT 15
DATA/READ 16
Hardware-Konfiguration 17
KONFIGURATIONSZEILE 17
LED-Zusatzbefehle für LED-Basic-Komponenten 18
LED-Befehle für serielle RGB-LEDs mit PWM (z.B. WS2812, APA102, SK6812) 18
LED.SHOW 18
LED.LRGB 18
LED.LHSV 18
LED.IRGB 18
LED.IHSV 18
LED.ILED 19
LED.IALL 19
LED.IRANGE 19
LED.RAINBOW 19

LED.COPY 19
LED.REPEAT 19
LED.SHIFT 20
LED.MIRROR 20
LED.BLACKOUT 20
LED-Befehle für RGB-LEDs ohne PWM und Einzel-LEDs 21
LED.SETLED 21
LED.SETALL 21
LED-Befehle für mehrere LED-Typen 21
LED.BRIGHT 21
LED-Befehle für 7-SEGMENT-DISPLAY 22
LED.CLEAR 22
LED.PCHAR 22
LED.PRAW 22
LED.ADP 22
LED.PHEX 23
LED.PDEZ 23
LED.UPDATE 23
LED-Befehle nur für 4-stelliges 7-SEGMENT-DISPLAY 23
LED.ACHAR 23
LED.ARAW 23
MATRIX-Befehle (ab Version 15.2.5) 24
MATRIX.SETXY 24
MATRIX.LINE 24
MATRIX.RECT 24
MATRIX.CIRCLE 24
MATRIX.SHIFT 24
MATRIX.SETFONT 24
MATRIX.CHAR 25
MATRIX.PIC 25
MATRIX.SIZE 25
MATRIX.SELECT 25
IO-Zusatzbefehle für LED-Basic-Komponenten 26
IO.WAITKEY 26
IO.GETKEY 26
IO.KEYSTATE 27
IO.SETPORT 27
IO.CLRPORT 27
IO.GETRTC 27
IO.SETRTC 28
IO.GETLDR 28
IO.GETIR 28
IO.GETTEMP 29
IO.XTEMPCNT 29
IO.XTEMPVAL 29

IO.BEEP 30
IO.GETENC 30
IO.SETENC 30
IO.GETPOTI 30
IO.GETADC 31
IO.EEREAD 31
IO.EEWRITE 31
IO.SYS 32
IO.BT 34
Laufzeit-Fehlermeldungen 35
Led-Basic-Komponenten 36
Treiberinstallation 36
Unterschiedliche Komponenten 36
Liste der von LED-Basic unterstützten Komponenten 37
Komponenten-Bootloader 37
LED-Badge (12 LEDs) 38
LED-Badge (16 LEDs) 38
Basic-Pentagon-Board 39
Basic-Budget-Board 40
Cronios 1 41
Cronios-Segmenta 42
Basic-Booster 43
Cortex-Clock 44
Temperatur-Sensor Interface 45
Lauflicht mit 16 LEDs 46
All-In-One Power-M4-Board 47
LED-Box 49
APA-Booster 50
RC-Box (Radio controlled box) 51
Touch-Lamp 53
LED-Tube-Clock 54
NixieCron Komponenten 55
NixieCron - Cronios 2 55
NixieCron - Cronios 3 (Sound) 56
NixieCron - LED-Nixie-M4 57
NixieCron - LED-Tube-Clock 58
NixieCron – Flame-Clock 59
NixieCron – Matrix- und Segment-Tube-Clock 60
LED-BASIC-PICO Komponenten 61
PICO: Basismodul Version 1 61
PICO2: Basismodul Version 2 62
PICO: Anschluss von WS2812 LEDs 65
PICO: Eingabetaste 65
PICO: Joystick 65
PICO: Drehimpulsgeber 66

PICO: Potentiometer 66
PICO: Infrarot-Empfänger 66
PICO: Fotodiode 66
PICO: Temperatursensor 67
PICO: Vibrationssensor 67
PICO: Beeper 67
PICO: RTC-Modul mit EEPROM 67
PICO: 7-Segment-Anzeige mit Taster 68
LED-BASIC-PICO PROJEKTE 70
PICO: NixieCron - SINGLE-DIGIT-Clock 70
PICO: NixieCron – S4-„Jukebox“-Clock 71
PICO2: Running Light, Lauflicht 72
Zubehör, Erweiterungen 73
Prog-SB, Seriell-Basic Programmier- und Terminaladapter 73
DCF77-Zeitmodul 74
GPS -> DCF-Zeitmodul 75
WLAN -> DCF-Zeitmodul 77
Versionen 78
V15.2.8 78
V15.2.7 78
V15.2.6 78
V15.2.5 78
V15.2.4 78
V15.2.3 78
V15.2.2 78
V15.2.1 79
V15.2.0 79
V15.1.15 79
V15.1.14 79
V15.1.12 79
V15.1.11 80
V15.1.10 80
V15.1.9 80
V15.1.8 80
V15.1.7 80
V15.1.6 80
V15.1.5 80
Fehler, Bugs 81
Hinweise 82
Links 82

LED-Basic?

Hinter der Idee von LED-Basic stecken langjährige Erfahrungen im Umgang mit Leuchtdioden und deren Ansteuerung mit dem PC oder als selbstständige Module wie die verschiedensten LED-Player oder LED-Controller. Alle hatten das Problem, dass die Daten für die LEDs von irgendeinem Programm erzeugt werden mussten. Meist wurde hierfür eines der Programme JINX! oder GLEDiator benutzt, die jedoch hauptsächlich für LED-Matrizen ausgelegt sind und teilweise recht umständlich zu konfigurieren und zu bedienen sind. Um die erzeugten Daten auf dem LED-Player benutzen zu können, mussten diese auf eine SD-Karte oder USB-Stick kopiert und in den Player gesteckt werden. Für kleine, einfache Projekte ist dieser Vorgang zu aufwändig und man ist auf die Effekte der benutzten Programme angewiesen. Eigene Kreationen sind nur im begrenzen Maße durch Änderungen der Effektparameter möglich.

LED-Basic ist hauptsächlich für die Ansteuerung von Einzel-LEDs oder LED-Stripes ausgelegt. Die maximale Anzahl der anzusteuernden LEDs hängt weitgehend von der Leistung des Microcontrollers und der Größe des Speichers der verwendeten LED-Basic-Komponenten ab.

LED-Basic muss nicht für Anwendungen mit LEDs benutzt werden. Steuerungen, die z.B. Sensoren abfragen und abhängig von den gemessenen Werten Signale auf IO-Ports ausgeben sind ebenso möglich (siehe Temperatur-Sensor-Interface).

Erweiterungen?
LED-Basic soll einfach zu benutzen sein und die Befehle soll sich jeder Anwender leicht merken können. Deswegen wurde auf eine aufwändige Benutzeroberfläche und Basic-Befehlen mit umständlicher Syntax absichtlich verzichtet. Sinnvolle Erweiterungen, die von den Anwendern vorgeschlagen werden, können sicher in zukünftige Versionen integriert werden. Ihre Mitarbeit ist also erwünscht. Senden Sie uns Ihre Vorschläge und auch eventuelle Fehlermeldungen an feedback@led-basic.de.

NixieCron©

Komponenten und Uhren mit dem NixieCron-Label besitzen den hochgenauen Uhrenchip DS3231. Dies bedeutet, eine geringe Abweichung von nur wenigen Sekunden im Monat auch ohne DCF-, GPS- oder WLAN/NTP-Synchronisation. Die Einstellung der Uhr erfolgt über den PC oder alternativ über den DCF77-Anschluß (nicht bei jeder Komponente vorhanden).

Installation und Programmstart

Laden Sie das Installationspaket „LedBasic_aa.b.c.exe“ von unserem Server herunter und installieren Sie dieses auf ihrem PC (aa.b.c = aktuelle Version).

Im Ordner „SOURCE“ des Installationsordners befinden sich mehrere Ordner mit Beispieldateien für verschiedene LED-Basic Komponenten. Sie können diese Dateien als Anreiz für Ihre eigenen Kreationen benutzen und nach Belieben verändern, sollten Ihre eigenen Versionen aber unter einem anderen neuen Namen speichern. Bei einem Update könnten die Originaldateien überschrieben werden.

Im Installationsverzeichnis befinden sich auch die USB-Treiberdatei und diese Bedienungsanleitung im PDF-Format.

Windows 10 und die Sicherheit

Sollte es unter Windows 10 Fehlermeldungen wie „Zugriff verweigert“ geben, starten Sie das Programm bitte mit Administrator-Rechten (mit der rechten Maustaste auf das LED-Basic-Icon klicken und „Als Administrator ausführen“ auswählen).

Automatische und manuelle Updates

Der LED-Basic Editor testet automatisch bei Programmstart, ob eine neue Version vorhanden ist (wenn diese Option in der Konfiguration aktiviert ist). Sie können dies auch jederzeit über das Hilfe-Menü -> Teste auf Aktualisierung überprüfen. Sollte eine aktualisierte Version vorhanden sein, wird diese automatisch installiert und gestartet. Voraussetzung hierfür ist natürlich, dass Ihr PC mit dem Internet verbunden ist. Sie können die aktuelle Version natürlich auch von unserem Server herunterladen und manuell installieren.

Zusammen mit der aktuellen Software wird auch diese PDF-Anleitung aktualisiert. Diese wird ebenfalls bei einem Update heruntergeladen und ersetzt die alte Version.

LED-Basic Editor

Der LED-Basic-Editor ist gleichzeitig auch ein sogenannter „Tokenizer“. Er übersetzt den Basic-Code aus dem Klartext in „Tokens“, dies spart Speicherplatz auf der LED-Basic-Hardware und bringt einen enormen Geschwindigkeitsvorteil bei der Ausführung des Programmes. Der „Tokenizer“ prüft die Syntax des eingegeben Basic-Codes, wandelt Sprungbefehle und Labels in absolute Adressen um und checkt die korrekte Verwendung von Variablen und die Anzahl der Parameter bei LED- und IO-Befehlen.

Es wird vorausgesetzt, dass sich der Anwender mit der Programmiersprache BASIC auskennt und die Unterschiede zwischen Variablen, Konstanten und Ausdrücken versteht. Wenn nicht, es gibt im Internet etliche Literatur über diese recht einfache Programmiersprache. Die Syntax und einige Besonderheiten des LED-Basic befinden sich in der Beschreibung des Grundbefehlssatzes und der Zusatzbefehle. Außerdem wird es mit der Zeit jede Menge Beispielprogramme geben, wo man sich die Programmiertechniken abschauen kann.

Die Version des LED-Basic Editor (z.B. 15.2.0) ergibt sich aus der aktuellen Basic-Version (v15) und der Version des Editor (v2.0). Sollte die Version des Editors nicht mit der Basic-Version der verwendeten LED-Basic-Komponente übereinstimmen, wird automatisch eine Aktualisierung des Bios durchgeführt.

Editor-Funktionen

Die Bedienung eines Texteditors sollte allgemein bekannt sein. Der LED-Basic-Editor unterstützt die Standard-Funktionen Ausschneiden, Kopieren und Einfügen sowie eine Rückgängig- und Wiederherstellen-Funktion (Undo/Redo) mit bis zu 100 Schritten. Beim Speichern einer Quelldatei wird immer eine Sicherheitskopie der vorhandenen Datei mit der Endung .bak angelegt.

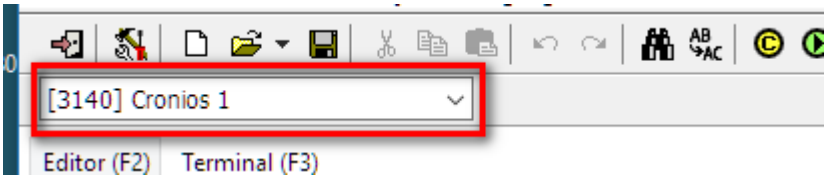
Die Sprache der Benutzeroberfläche des Editors kann über den Menüpunkt „Language“ gewählt werden. Beachten Sie, dass die Sprache der Datei- oder Suchen-Dialoge von Ihrer installierten Windows-Version abhängig ist.

Da LED-Basic keine Include-Dateien unterstützt, kann der Editor nur eine Datei gleichzeitig öffnen und bearbeiten.

Über das Menü, die Icon-Leiste oder über Tasten-Kurzbefehle können die wichtigsten Funktionen des Editors schnell aufgerufen werden.

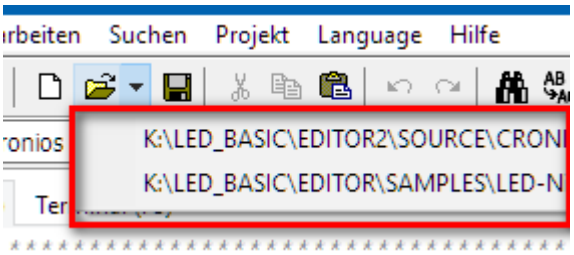
	F9	Übersetzt den Basic-Code und zeigt eventuelle Fehler im Info-Fenster an.
	Umsch + F9 oder F11	Übersetzt den Basic-Code und zeigt eventuelle Fehler Info-Fenster an. Nach erfolgreicher Übersetzung wird der Code zur LED-Basic-Komponente übertragen.
	F12	Löst einen Neustart des laufenden Basic-Programmes auf der LED-Basic-Komponente aus.
	Umsch + F11	Stellen der Echtzeituhr (falls von der LED-Basic-Komponente unterstützt).
		Hochladen von Zusatzdaten (nicht bei jeder Komponente verfügbar), z.B. Sound-Daten bei Cronios3

Ab Editor Version 15.2.0 kann die Komponente direkt vom Hauptbildschirm geändert werden.

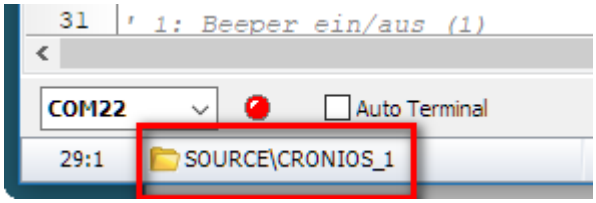


Durch Ändern der Komponente, werden folgende Parameter automatisch verändert:

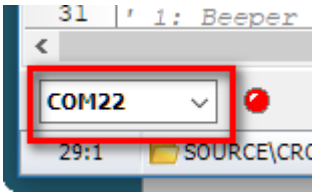
1. Die Liste der zuletzt verwendeten Dateien.



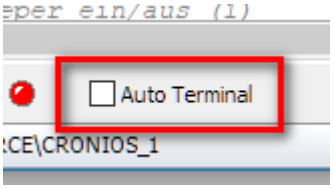
2. Der Verzeichnispfad für Source-Dateien für diese Komponente. Dieser wird immer in der Statuszeile angezeigt.



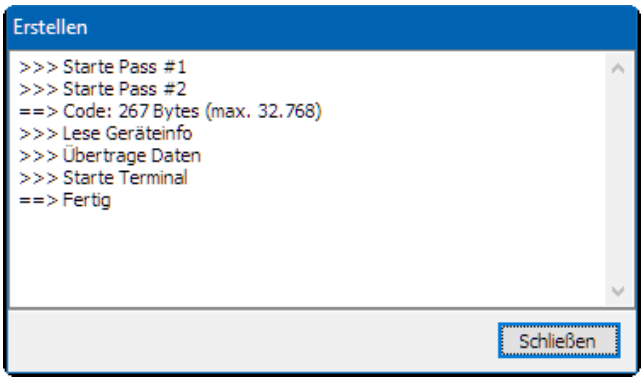
3. Der für diese Komponente eingestellte COM-Port.



4. Die Einstellung, ob das Terminal nach erfolgreicher Übertragung zur Komponente geöffnet werden soll.



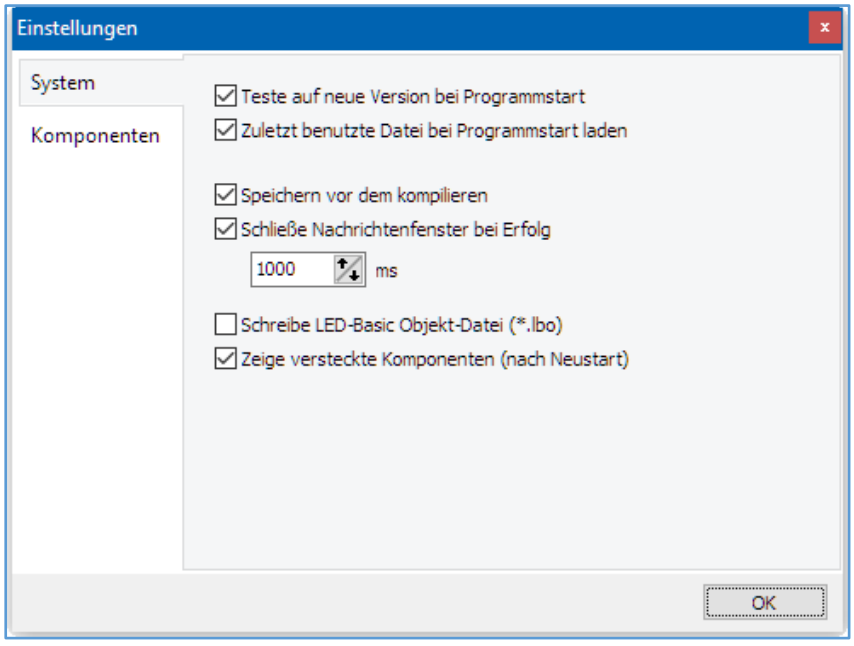
Info-Fenster



Im Info-Fenster werden eventuelle Fehler während des Übersetzens des Basic-Quellcodes oder der Datenübertragung zur LED-Basic-Komponente angezeigt. Wenn der Vorgang fehlerfrei durchgeführt wurde, schließt sich das Fenster nach der in der Konfiguration eingestellten Zeit von selber oder muss manuell geschlossen werden. Sollte ein Fehler aufgetreten sein, bleibt das Fenster immer geöffnet und muss manuell geschlossen werden.

Konfiguration

System



Teste auf neue Version bei Programmstart:

Wenn eine Internet-Verbindung existiert, wird bei Start des LED-Basic Editors automatisch nachgeschaut, ob eine neue Version existiert. Diese kann dann heruntergeladen und installiert werden. Eine manuelle Abfrage kann jederzeit über das Hilfe-Menü vorgenommen werden.

Zuletzt benutzte Datei bei Programmstart laden:

Die zuletzt bearbeitete Datei wird bei Programmstart wieder eingeladen. Wenn dieser Menüpunkt nicht angewählt ist, wird der LED-Basic Editor mit einem leeren Dokument geöffnet.

Speichern vor dem kompilieren:

Vor Erstellen/Kompilieren des Basic-Codes wird die aktuelle Datei automatisch gespeichert. Zusätzlich wird ein Sicherung der vorherigen Datei angelegt (*.bak).

Schließe Nachrichtenfenster bei Erfolg:

Bei einer fehlerfreien Übersetzung des Basic-Codes wird das Nachrichtenfenster nach der eingestellten Zeit automatisch geschlossen. Wenn dieser Punkt nicht gewählt ist, bleibt das Nachrichtenfenster so lange geöffnet, bis es manuell geschlossen wird.

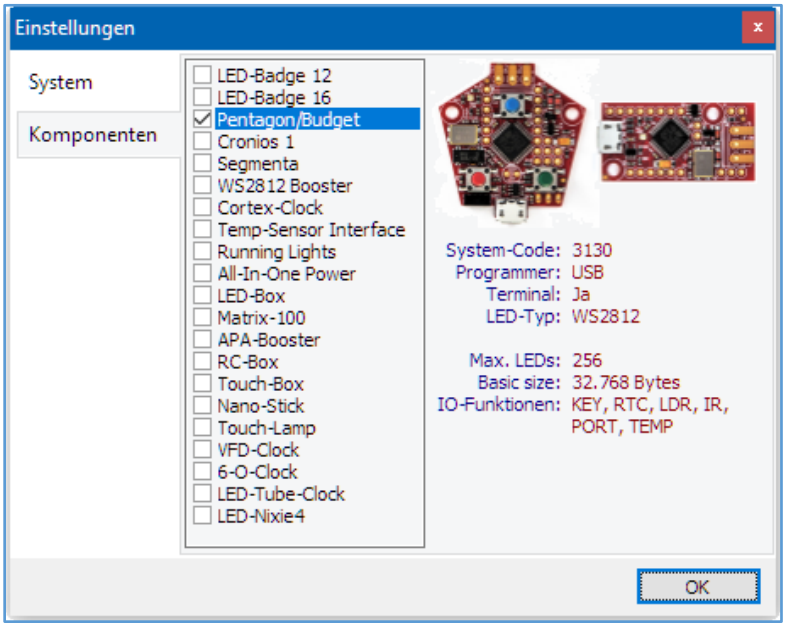
Schreibe LED-Basic Objekt-Datei (*.lbo):

Die vom Tokenizer erzeugte Objekt-Datei wird im selben Verzeichnis wie der Basic-Code abgelegt. Diese kann mit einem externen Programmiertool zur LedBasic-Komponente übertragen werden (in Planung).

Zeige versteckte Komponenten (nach Neustart):

Komponenten, die es leider (noch) nicht in die Serienfertigung geschafft haben, findet man hier. Damit die versteckten Komponenten angezeigt bzw. ausgeblendet werden, ist ein Neustart des Programmes erforderlich.

Komponenten



Wählen Sie die passende LED-Basic-Komponente aus der Liste aus. Sie können sich alle Komponenten zur Auswahl anschauen. Wichtig! Nur wenn die Komponente mit einem Haken markiert ist, wird es auch benutzt.

Unter dem Foto der Komponente finden Sie dessen Spezifikationen:

System-Code:

Dieser Code wird zur Identifikation der LED-Basic-Komponente benutzt. Wird eine Komponente angeschlossen, die nicht zum eingestellten System-Code passt, kann der erzeugte Basic-Code nicht hochgeladen werden.

Programmer:

Hier wird angezeigt, ob ein Programmieradapter (z.B. Prog_SB) benutzt werden muss, um das den Basic-Code in die Komponente hochzuladen.

Terminal:

Ist eine Print- und Fehlerausgabe über das Terminal möglich?

LED-Typ:

Welche LED-Typen werden von der Komponente unterstützt?

RGB	Einfache RGB-LEDs ohne PWM, 7 verschiedene Farben sind möglich.
WS2812	Serielle LEDs mit integriertem PWM-Controller des Typs WS2811 (RGB oder RGBW), auch SK6812 oder APA104/106. Merkmal: 3 Anschlussleitungen (+5V, DATA, GND)
APA102	Serielle LEDs mit integriertem PWM-Controller des Typs WS2801 (RGB). Merkmal: 4 Anschlussleitungen (+5V, DATA, CLOCK, GND)
7-SEGMENT	7-Segment-Anzeige, z.B. bei Cortex-Clock
SIMPLE-LED	Einzel-Leuchtdioden, z.B. bei Lauflicht mit 16 LEDs

Max. LEDs:

Maximale Anzahl der LEDs, die von der Komponente unterstützt werden.

Basic size:

Maximale Größe des Basic-Programmes?

IO-Funktionen:

Welche IO-Funktionen werden von der Komponente unterstützt?

KEY	Tasten und Eingangsports (KEY_x)
PORT	Ausgangsports (PORT_x)
RTC	Echtzeituhr
LDR	Helligkeitssensor
TEMP	Temperatursensor (1 x DS18B20), nur Abfrage der Temperatur
XTEMP	Temperatursensor (max. 8 x DS18B20), Temperatur- und Parameterabfrage
IR	Sensor für Infrarot-Fernbedienung
BEEP	Tonausgabe über Lautsprecher
ENC	Drehimpulgeber
POTI	Analoge Drehregler (z.B. Potentiometer)
ADC	Analogwerte abfragen (z.B. Batteriespannung)
EEP	EEPROM, speichert Daten, die auch nach entfernen der Stromversorgung erhalten bleiben.
SYS	System-Funktionen aufrufen, System-Variablen lesen oder schreiben
DATA	Zusatzdaten im FLASH-Speicher
BT	Befehle für Senden und Empfangen von Daten über Bluetooth

LED-Basic Grundbefehlssatz

Der Sprachumfang des LED-Basic beschränkt sich auf wenige Funktionen, die von der BASIC-Programmiersprache her bekannt sind. Spezielle LED- und IO-Befehle erweitern das Basic. Durch das Umwandeln der Befehle in „Tokens“ wird für eine hohe Abarbeitungsgeschwindigkeit von über 10000 Zeilen pro Sekunde (abhängig vom verwendeten Microcontroller) gesorgt.

Grundsätzliche Regeln für LED-Basic

- Groß- und Kleinschreibung wird nicht unterschieden
- Es gibt 26 globale Variablen a…z bzw. A…Z (a ist identisch mit A)
- LED-Basic rechnet mit maximal 16 Bit, -32768…+32767
- Es sind keine Fließkommawerte möglich
- Es gibt keine String-Variablen
- Maximal 4 verschachtelte GOSUB sind möglich
- Maximal 4 verschachtelte FOR-NEXT-Schleifen sind möglich
- Alle Befehle nach END werden ignoriert
- Bei einem ‘ (Hochkomma) am Anfang einer Zeile wird diese komplett ignoriert (wie bei REM)
- Über ein Terminalprogramm (z.B. das aus dem LED-Basic-Editor) werden PRINT-Ausgaben sowie Fehlermeldungen des LED-Basic-Players angezeigt, wenn dies von der LED-Basic-Komponente unterstützt wird

Es gibt in LED-Basic keine Zeilennummern, wie es von anderen Basic-Varianten bekannt ist. Zeilennummern werden dennoch als Sprungmarken für GOTO, GOSUB oder als Index auf DATA-Werte eingesetzt. Eine Sprungmarke besteht aus einer Zahl zwischen 0 und 32766 mit nachfolgendem Doppelpunkt

Beispiele

```
1000:
  i = 123
  ...
  Return
```

```
30000: a = 0
  ...
  Return
```

Die Reihenfolge der Nummern ist beliebig, muss jedoch im gesamten Quelltext einmalig sein. Der nachfolgende Befehl kann in derselben oder der darauffolgenden Zeile stehen.

Hinweise zur Benutzung von Variablen

Die 26 Variablen a..z (A..Z) sind global gültig. Es gibt keine lokalen Variablen, wenn Sie Variablen und Unterroutinen benutzen, sollten Sie darauf achten, dass Sie nicht dieselben Variablen wie in der aufrufenden Routine verwenden. Machen Sie Sich eine Liste oder schreiben die verwendeten Variablen als Kommentar in den Code.

Eingabe von Zahlwerten:

Dezimalzahlen:	0 1234 -2000
Hexadezimalzahlen:	0x1234 0xAB
Binärzahlen:	0b10101010 0b10

Arithmetische Operatoren:

+	Addition, 3 + 4
-	Subtraktion, 5 - 2
/	Division, 9 / 3
*	Multiplikation, 10 * 5
%	Modulo (Divisionsrest), 6 % 5

Bit-Operatoren:

|	bitweise OR-Verknüpfung, 0x05 | 0x80
&	bitweise AND-Verknüpfung, 0xAE & 0x07

Vergleichs-Operatoren:

<	kleiner als
>	größer als
=	gleich
<>	ungleich
<=	kleiner gleich
>=	größer gleich

Logische-Operatoren:

or	logische OR-Verknüpfung, if a=1 or a=3 then ...
and	logische AND-Verknüpfung, if a=2 and b=3 then ...

Syntaxbeschreibung

Für die folgende Syntaxbeschreibung der LED-Basic-Befehl gilt:

- *VAL* = numerischer Wert (123, 0x100, 0b101010)
- *STR* = String, muss in Anführungszeichen eingeschlossen sein (“HALLO WELT“)
  (nur im PRINT-Befehl möglich)
- *EXPR* = Ein Ausdruck mit numerischem Ergebnis (1 + 2, a * 3)
- *VAR* = Variable (a, A, z, Z)
- *REL* = Vergleichsoperation (a > 4, x = y)
- *LABEL* = Sprungziel oder Datenbasis (1234:)
- . . . = beliebige Anweisung
- <xxx> = Eingabe erforderlich
- xxx|yyy = xxx oder yyy sind möglich
- <VAL|EXPR|VAR> = hier kann ein numerischer Wert, ein Ausdruck oder eine Variable stehen
- [xxx] = kann optional hinzugefügt werden

REM

```
rem ...
´ ...
```

Remark, Anmerkung, alle Anweisungen hinter **rem** oder ` (Hochkomma) werden ignoriert.

END

```
end
```

Alle Befehle hinter **end** werden ignoriert

LET

```
[let] <var>=<VAL|EXPR|VAR>
<var>=<VAL|EXPR|VAR>
```

Zuweisung von Werten an eine Variable.

let ist optional und kann auch weggelassen werden.

FOR-NEXT-SCHLEIFE

```
for <VAR>=<VAL|EXPR|VAR> to|downto <VAL|EXPR|VAR> [step
<VAL|EXPR|VAR>]

...

next VAR
```

Bitte darauf achten, dass der rechte Wert nach **to** niemals kleiner wird als der Wert vor dem **to** und dass der rechte Wert nach **downto** niemals größer wird als der Wert vor dem **downto.**

Der Wert für **step** ist immer positiv, auch beim **downto**.

Ohne **step** ist die Schrittweite immer 1

Eine Verschachtelung von maximal 4 FOR-NEXT-Schleifen ist möglich. Dabei immer darauf achten dass sich der zur FOR-Variable passende NEXT-Befehl innerhalb derselben Ebene befindet.

IF-THEN-ELSE

```
if <VAL|EXPR|VAR> <REL> <VAL|EXPR|VAR> [then] ... [else ...]
```

then ist nicht unbedingt erforderlich, muss aber vorhanden sein, wenn auch **else** verwendet wird.

else kann optional benutzt werden.

Nach **then** oder **else** ist nur ein Befehl oder Ausdruck möglich, sollen mehrere Befehle ausgeführt werden, kann das über eine Unterroutine mit **gosub/return** gelöst werden.

GOTO

```
goto <VAL>
```

Es ist kein Ausdruck möglich, nur ein numerischer Wert.

GOSUB/RETURN

```
gosub <VAL>
...
<LABEL>
Return
```

Es ist kein Ausdruck möglich, nur ein numerischer Wert.

Eine Verschachtelungstiefe von maximal 4 **gosub**-Anweisungen sind möglich.

RANDOM

```
random
```

Erzeugt eine positive Zufallszahl zwischen 0 und 32767

Eine Initialisierung des Random-Generators wie in anderen Basic-Varianten mit „randomize“ ist nicht erforderlich.

DELAY

```
delay <VAL|EXPR|VAR>
```

Der Programmablauf wird um xx Millisekunden angehalten.

Der Wert für **delay** darf nicht 0 enthalten, Maximalwert ist 32767 (entspricht ca. 32,8 Sekunden)

PRINT

```
print <VAL|EXPR|VAR|STR> [;][,] [<VAL|EXPR|VAR|STR>]
```

Dient zur Status- oder Debug-Ausgabe über den Terminalausgang (nicht bei jeder LED-Basic-Komponente verfügbar).

Mehrere Werte oder Texte können hintereinander ausgegeben werden. Bei einem Komma wird ein Leerzeichen eingefügt, bei einem Semikolon wird kein Leerzeichen eingefügt. Die maximale Länge des mit **print** ausgegebenen Textes ist 256 Zeichen. Längere Texte werden abgeschnitten.

Mit diesem Befehl sollten nur wenige Daten ausgegeben werden, da dieser die Ausführungsgeschwindigkeit des LED-Basic reduziert. Über den Konfigurationsparameter P0 kann die PRINT-Funktion global deaktiviert werden. Werden die Daten zu schnell hintereinander gesendet, kann es zur Blockade des Terminals aufgrund eines Speicherüberlaufes oder zu einer fehlerhaften Anzeige kommen. Sollte sich das LED-Basic-Programm hierdurch nicht mehr bedienen lassen, ziehen Sie kurz den USB-Stecker von der Komponente oder Prog_SB ab.

DATA/READ

```
<LABEL>
data <VAL>[,<VAL>][,<VAL>]
...
read <VAL>,<VAL|EXPR|VAR>
```

Achtung! Die hier verwendete Syntax ist nicht kompatibel mit anderen Basic-Varianten und speziell für LED-Basic optimiert.

Hinter einem Label können maximal 127 Tabellenwerte definiert werden. Dieses dürfen nur numerische Werte sein und können nachträglich nicht verändert werden. Es können auch mehrere data-Zeilen hintereinander folgen, insgesamt jedoch nur 126 Werte. Alle Werte dürfen maximal 16-Bit groß sein (-32768...32767).

Beispiel:

```
100:
data  10, 20, 30, 40
data  100, 200, 400, 800


200:
data  0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80
```

Data-Zeilen müssen immer am Anfang des Programmes, spätestens jedoch vor ihrer ersten Benutzung definiert werden.

Mit dem **read**-Befehl wird gezielt auf die einzelnen **data**-Werte zugegriffen.

Beispiel:

```
[1] a = read 100,0
[2] x = read 200,i
```

[1] Liest den ersten Data-Wert hinter Label 100 (Zählung beginnt immer mit 0)

[2] Liest den Data-Wert hinter Label 200 mit dem Index in der Variablen i

Wenn versucht wird, hinter den letzten Data-Wert zuzugreifen, wird kein Fehler ausgegeben und als Wert 0 gelesen.

Hinter der Data-Zeile darf kein Kommentar stehen. Schreiben Sie den Kommentar einfach vor die Zeile.

Hardware-Konfiguration

Über die Konfigurationszeile können verschiedene Einstellungen global geändert werden. Beachten Sie bitte, dass nicht jede LED-Basic-Komponente alle Parameter unterstützt. Nicht unterstützte Parameter werden ignoriert.

KONFIGURATIONSZEILE

```
### L64 CRGB M100 P1 S3 T0 A3 F25
```

Diese Zeile muss zwingend direkt am Anfang des BASIC-Codes eingefügt werden, wenn die Standardwerte nicht benutzt werden sollen. Beginnend mit **###** sind mehrere Konfigurationsparameter möglich:

Parameter	Beschreibung	Standardwert
`Lxxx`	Anzahl der angeschlossenen LEDs (MAX_LED) Gültige Werte: 8..x (x = abhängig von der verwendeten Komponente)	L256, L128, L64
Cxxxy	Farbanordnung der angeschlossenen LEDs GRB = WS2812, RGB = SK6812, APA102, APA106 Für RGBW-Leds, muss hier CRGBW eingetragen werden	CGRB
Mxxx	Master-Helligkeit in % Gültige Werte: 1..100	M100
Px	Printausgabe global ein/ausschalten Gültige Werte: 0,1 (0 = aus, 1 = ein) Hinweis: Laufzeit-Fehlermeldungen werden immer ausgegeben und können nicht ausgeschaltet werden.	P1
Sx	System-Leds ein/ausschalten Gültige Werte: 0..3 (0 = aus, 1 = Ausgabe-LED ein, 2 = Warte-LED ein, 3 = Alle LEDs ein)	S3
Tx	LED-Typ auswählen (bei LED-Basic-Komponenten, die mehrere LED-Typen unterstützen) Gültige Werte: Siehe Hinweis bei den Komponenten	T0
Ax	Bitrate für die LED-Typen APA102 auswählen (SPI-Takt) Je höher der Wert, desto niedriger die Bitrate. Gültige Werte: 0..7 (Die Bitraten sind bei den Komponenten aufgelistet)	A3
Fxx	Framerate bei seriellen LEDs, die den LED.show() – Befehl nutzen. Bei zu niedrigen Werte erfolgt die Ausgabe ruckelnd und flackernd. Bei zu hohen Werten kann es zu Störungen der LED-Anzeige kommen und es kann sich die Ausführungsgeschwindigkeit des LED-Basic reduzieren. Gültige Werte: 1..100	F25

LED-Zusatzbefehle für LED-Basic-Komponenten

Anzahl und Parameter der Zusatzbefehle sind abhängig von der verwendeten LED-Basic-Komponente und können sich in jeder Version ändern. Bitte deshalb unbedingt die Hinweise bei den Updates beachten.

Die Anzahl der Parameter muss unbedingt stimmen. Ist kein Parameter erforderlich, müssen jedoch wie in der C-Syntax die Klammern dem Befehl folgen. Die Wertbereiche der Parameter werden nicht überprüft. Werden falsche Werte übermittelt, kann es zu fehlerhaften LED-Anzeigen kommen.

Als Parameter können neben numerischen Werten natürlich auch Variablen oder berechnete Ausdrücke eingesetzt werden.

Die Funktion einiger Befehle kann am besten durch ausprobieren herausgefunden werden. Beispiele befinden sich auch im Installationspaket und auf der Support-Homepage (Link am Ende).

In der Beschreibung der LED-Basic-Komponenten finden Sie die Liste der unterstützten LED-Befehle.

LED-Befehle für serielle RGB-LEDs mit PWM (z.B. WS2812, APA102, SK6812)

LED.SHOW

```
LED.show()
```

Die Ausgabe zu den LEDs wird gestartet. Aufgrund der seriellen Übertragung zu den LEDs ist dies nur standardmäßig alle 40ms möglich (25 Frames / Sekunde) und wird automatisch auf diese Geschwindigkeit begrenzt. Ohne diesen Befehl ist keine Anzeige auf den LEDs möglich. Die Framerate kann bei einigen Komponenten über den Konfigurationszeilen-Parameter Fxx geändert werden.

LED.LRGB

```
LED.lrgb(led, r, g, b)
```

LED an Position **led** wird mit den Werten in **r**, **g** und **b** gesetzt.

Werte: led=[0..MAX_LED-1], r/g/b=[0..255]

LED.LHSV

```
LED.lhsv(led, h, s, v)
```

LED an Position **led** wird mit den Werten in **h** (HUE, Farbwert), **s** (SAT, Sättigung) und **v** (VOL, Helligkeit) gesetzt.

Werte: led=[0..MAX_LED-1], h=[0..359], s=[0..255], v=[0..255]

LED.IRGB

```
LED.irgb(idx, r, g, b)
```

Bis zu 10 LED-Farbwerte können in Index-Registern gespeichert werden. Farbindex **idx** wird auf die Werte in **r**, **g** und **b** gesetzt.

Werte: idx=[0..9], r/g/b=[0..255]

LED.IHSV

```
LED.ihsv(idx, h, s, v)
```

Bis zu 10 LED-Farbwerte können in Index-Registern gespeichert werden. Farbindex **idx** wird auf die Werte in **h**, **s** und **v** gesetzt.

Werte: idx=[0..9], h=[0..359], s=[0..255], v=[0..255]

LED.ILED

```
LED.iled(idx, led)
```

LED an Position **led** wird mit dem Wert im Farbindex **idx** gesetzt.

Werte: led=[0..MAX_LED-1], idx=[0..9]

LED.IALL

```
LED.iall(idx)
```

Alle LEDs werden mit dem Wert im Farbindex **idx** gesetzt.

Werte: idx=[0..9]

LED.IRANGE

```
LED.irange(idx, beg, end)
```

Die LEDs im Bereich von **beg** bis **end** werden mit dem Wert im Farbindex **idx** gesetzt.

Werte: idx=[0..9], beg=[0..MAX_LEDS-1], end=[beg..MAX_LEDS-1]

LED.RAINBOW

```
LED.rainbow(h, s, v, beg, end, inc)
```

Ein Regenbogen-Effekt wird über einen definierten LED-Bereich (**beg..end**) erzeugt. Der Start-Farbwert wird über **h**, **s**, und **v** festgelegt, die Stärke des Farbverlaufes über den Wert in **inc**. Zu einem bewegten Farbverlauf kommt es, wenn der Startwert **h** ständig verändert wird (siehe Beispiel auf der Support-Homepage).

Werte: h=[0..359], s=[0..255], v=[0..255], beg=[0..MAX_LEDS-1], end=[beg..MAX_LEDS-1], inc=[0..100]

LED.COPY

```
LED.copy(from, to)
```

Eine einzelne LED wird von Position **from** an Position **to** kopiert. Die LED an Position **from** bleibt dabei unverändert.

Werte: from=[0..MAX_LEDS-1], to=[0..MAX_LEDS-1]

LED.REPEAT

```
LED.repeat(beg, end, count)
```

Der LED-Bereich **beg** bis **end** wird um die Anzahl in **count** wiederholt.

Werte: beg=[0..MAX_LEDS-1], end=[beg..MAX_LEDS-1], count=[1..x]

Beispiel:

```
LED.repeat(0,5,3)
Led-Anordnung vor Aufruf:
0 1 2 3 4 5
Led-Anordnung nach Aufruf (3-fache Wiederholung):
0 1 2 3 4 5 0 1 2 3 4 5 0 1 2 3 4 5 0 1 2 3 4 5
```

Es erfolgt automatisch eine obere Begrenzung bei MAX_LEDS.

LED.SHIFT

```
LED.shift(beg, end, to)
```

Der LED-Bereich **beg** bis **end** wird nach **to** verschoben.

Werte: beg=[0..MAX_LEDS-1], end=[beg..MAX_LEDS-1], to=[0..MAX_LEDS-1]

Beispiele:

```
LED.shift(0,4,2)
Led-Anordnung vor Aufruf:
0 1 2 3 4 x x
Led-Anordnung nach Aufruf:
0 1 0 1 2 3 4
```

```
LED.shift(6,9,3)
Led-Anordnung vor Aufruf:
x x x x x x 6 7 8 9
Led-Anordnung nach Aufruf:
x x x 6 7 8 9 7 8 9
```

Die LEDs im frei gewordenen Bereich behalten ihren Ursprungswert und müssen bei Bedarf auf neue Werte gesetzt werden.

LED.MIRROR

```
LED.mirror(beg, end, to)
```

Der LED-Bereich **beg** bis **end** nach **to** gespiegelt. Um unerwünschte Effekte zu vermeiden, sollte darauf geachtet werden, dass sich die Bereiche **beg...end** und **to** nicht überschneiden.

Werte: beg=[0..MAX_LEDS-1], end=[beg..MAX_LEDS-1], to=[0..MAX_LEDS-1]

Beispiel:

```
LED.mirror(0,5,6)
Led-Anordnung vor Aufruf:
0 1 2 3 4 5
Led-Anordnung nach Aufruf:
0 1 2 3 4 5 5 4 3 2 1 0
```

LED.BLACKOUT

```
LED.blackout()
```

Alle LEDs werden ausgeschaltet. Ein LED.show() ist nicht zusätzlich erforderlich.

LED-Befehle für RGB-LEDs ohne PWM und Einzel-LEDs

LED.SETLED

```
LED.setled(led, color)
```

LED **led** wird auf **color** gesetzt

Werte: led=0…max (abhängig von der Komponente), color 0…x (abhängig von der Komponente)

LED.SETALL

```
LED.setall(color)
```

Alle LEDs werden auf **color** gesetzt

Werte: color=0…7

COLOR	FARBE
0	Aus
1	Rot
2	Grün
3	Gelb
4	Blau
5	Magenta
6	Cyan
7	Weiß

LED-Befehle für mehrere LED-Typen

LED.BRIGHT

```
LED.bright(value)
```

Helligkeit der LEDs oder des Displays einstellen. Hinweis: Helligkeitsänderungen im oberen Bereich sind vom menschlichen Auge kaum wahrnehmbar, verändern jedoch die Stromaufnahme von LEDs enorm.

Werte: value = [0..x], x ist abhängig von der verwendeten Komponente und ist in der Liste der LED-Befehle angegeben.

LED-Befehle für 7-SEGMENT-DISPLAY

LED.CLEAR

```
LED.clear()
```

Die Anzeige wird gelöscht, alle Segmente aus. Um ein Flackern der Anzeige zu vermeiden, sollte dieser Befehl nicht in einer Ausgabeschleife stehen.

LED.PCHAR

```
LED.pchar(pos, char)
```

Gibt das vordefinierte Zeichen **char** an Position **pos** (+1) aus. Pos gibt die 7-Segment-Stelle an, an der das Zeichen ausgegeben werden soll, von links nach rechts. Die Anzeige der Dezimalpunkte bleibt unbeeinflusst.

Werte: pos=0…x, char=0…29 (x = Anzahl der Stellen – 1)

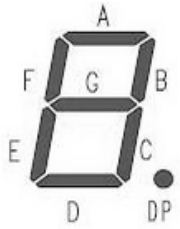
char	Zeichen	char	Zeichen	char	Zeichen
0	0	10	A	20	-
1	1	11	b	21	SPACE
2	2	12	C	22	i
3	3	13	d	23	n
4	4	14	E	24	r
5	5	15	F	25	N
6	6	16	H	26	t
7	7	17	L	27	o
8	8	18	P	28	G
9	9	19	U	29	Y

LED.PRAW

```
LED.praw(pos, raw)
```

Gibt das Zeichen in **raw** an Position **pos** (+1) aus. Pos gibt die 7-Segment-Stelle an, an der das Zeichen ausgegeben werden soll, von links nach rechts. Die Anzeige der Dezimalpunkte bleibt unbeeinflusst.

Werte (x = abhängig von der Komponente):
pos=0…x, raw=0…127 (0x7F)



Der Wert in raw gibt die leuchtenden Segmente bitkodiert an.

Bit 0 = A (0x01), Bit 1 = B (0x02), Bit 2 = C (0x04), Bit 3 = D (0x08),

Bit 4 = E (0x10), Bit 5 = F (0x20), Bit 6 = G (0x40)

Der Dezimalpunkt kann nur über den Befehl LED.ADP angesteuert werden.

LED.ADP

```
LED.adp(dp)
```

Setzt die Dezimalpunkte bitkodiert. Bit 0 = Position 0, Bit 1 = Position 1, usw. Die Anzeige aller anderen Segmente bleibt unberührt.

Werte: dp=0…255 (0xFF) (Abhängig von der Anzahl der Anzeigen)

LED.PHEX

```
LED.phex(pos, value, width)
```

Gibt den Wert in **value** als Hexzahl auf dem Display an Position **pos** (+1) aus. Der Wert in **width** (+1) gibt die Breite der Ausgabe an. Die Anzeige der Hexzahl geschieht immer mit Vornullen.

Beispiel: Eine 2-stellige Hexzahl (0x00..0xFF) soll auf den beiden rechten Anzeigen ausgegeben werden: LED.phex(2, value, 1).

Werte (x = abhängig von der Komponente):
pos=0…x, value=0…65536 (0xFFFF), width=0..4

LED.PDEZ

```
LED.pdez(pos, value, width, lzero)
```

Gibt den Wert in **value** als Dezimalzahl auf dem Display an Position **pos** (+1) aus. Der Wert in **width** (+1) gibt die Breite der Ausgabe an. Der Wert in lzero gibt an, ob die Vornullen unterdrückt werden sollen (0 = JA, 1 = NEIN). Wird ein Wert größer 9999 übergeben, erfolgt keine Anzeige.

Beispiel: Eine 3-stellige Dezimalzahl (0..999) soll rechtsbündig mit unterdrückten Vornullen ausgegeben werden: LED.pdez(1, value, 2, 0).

Werte (x = abhängig von der Komponente):
pos=0…x, value=0…9999, width=0..3, lzero=0..1
pos=0…x, value=0…65535, width=0..4, lzero=0..1

LED.UPDATE

```
LED.update()
```

Dieser Befehl wird bei einigen 7-Segment-Displays benötigt, um die Anzeige zu aktualisieren. Dieser Befehl sollte nur nach Änderung des Display-Inhaltes aufgerufen werden.

LED-Befehle nur für 4-stelliges 7-SEGMENT-DISPLAY

LED.ACHAR

```
LED.achar(ch1, ch2, ch3, ch4)
```

Gibt alle 4 Zeichen mit einem einzigen Befehl gleichzeitig aus. Es müssen die Werte aller 4 Zeichen übergeben werden. Die Werte sind identisch mit denen in der Tabelle bei LED.PCHAR. Die Anzeige der Dezimalpunkte bleibt unbeeinflusst.

Werte: ch1, ch2, ch3, ch4=0…29

LED.ARAW

```
LED.araw(raw1, raw2, raw3, raw4)
```

Gibt alle 4 Zeichen mit einem einzigen Befehl gleichzeitig aus. Es müssen die Werte aller 4 Zeichen übergeben werden. Die Werte sind identisch mit denen bei LED.PRAW. Die Anzeige der Dezimalpunkte bleibt unbeeinflusst.

Werte: raw1, raw2, raw3, raw4=0…127 (0x7F)

MATRIX-Befehle (ab Version 15.2.5)

Hinweise für alle folgenden MATRIX-Befehle:
Bei Objekten sind positive und negative Positionsangaben zulässig und können sich auch außerhalb der sichtbaren Matrix befinden. Dies ist z.B. beim Scrolling von Text/Zeichen sinnvoll.

Der Wert von col ist abhängig von der Komponente. Bei einfarbigen LEDs kann dies die Helligkeit oder der Indexwert auf ein Farbregister sein.

MATRIX.SETXY

```
MATRIX.setxy(x, y, col)
```

Setzt eine einzelne LED in einer Matrix an Position x, y.

Werte: x, y = -n … +n, col = abhängig von der Komponente

MATRIX.LINE

```
MATRIX.line(x0, y0, x1, y1, col)
```

Erzeugt eine Linie von x0, y0 nach x1, y1. Horizontale und vertikale Linien werden über einfache Schleifen erzeugt, für diagonale Linien wird ein Bresenham-Algorithmus benutzt.

Werte: x0, y0, x1, y1 = -n … +n, col = abhängig von der Komponente

MATRIX.RECT

```
MATRIX.rect(x0, y0, x1, y1, fill, col)
```

Erzeugt eine Rechteck von x0, y0 nach x1, y1. Bei fill = 1 wird der Inhalt des Rechtecks ausgefüllt.

Werte: x0, y0, x1, y1 = -n … +n, fill = 0/1, col = abhängig von der Komponente

MATRIX.CIRCLE

```
MATRIX.circle(x, y, r, fill, col)
```

Erzeugt einen Kreis an Mittenposition x, y mit Radius r. Bei fill = 1 wird der Inhalt des Kreises ausgefüllt.

Werte: x, y = -n … +n, fill = 0/1, col = abhängig von der Komponente

MATRIX.SHIFT

```
MATRIX.shift(dir, count)
```

Schiebt den gesamten Display-Inhalt um count-Positionen in Richtung dir. Bitte beachten, dass die frei gewordene Spalte/Zeile nicht von diesem Befehl gelöscht wird.

Werte: dir = 0: left, 1: right, 2: up, 3: down, count = abhängig von der Komponente

MATRIX.SETFONT

```
MATRIX.setfont(font)
```

Definiert den im folgenden Befehl benutzen Zeichensatz.

Werte: font = 0…n (abhängig von der Komponente)

MATRIX.CHAR

```
MATRIX.char(x, y, char, col)
```

Setzt Zeichen char aus dem mit MATRIX.setfont(n) ausgewählten Zeichensatz an Position x, y. Wenn eine Koordinate außerhalb des sichtbaren Bereichs angegeben wird, kann nur ein Teil des Zeichens dargestellt werden (z.B. für Scrolling).

Werte: x, y = -n … +n, char = 0…255, col = abhängig von der Komponente

MATRIX.PIC

```
MATRIX.pic(frame, opt)
```

Zeigt den gewählten Frame aus dem Speicher auf dem Display an (nicht bei allen Komponenten mit MATRIX-Befehlen verfügbar)

Werte: frame = 0…n, opt = abhängig von der Komponente

MATRIX.SIZE

```
MATRIX.size(x, y, cnt, res1, res2)
```

Definiert die Anzahl und Größe der Matrizen in einem Array (z.B. bei Matrix-Tube-Clock).

Werte: x = 1..n (Anzahl horizontal), y = 1..n (Anzahl vertikal), cnt = 1..n (Anzahl Matrizen), res1/res2 = reserviert (0).

MATRIX.SELECT

```
MATRIX.select(num)
```

Wählt die Matrix aus dem Array aus, auf der die folgenden Matrix-Befehle angewendet werden.

Werte: num = 0..n, n darf nicht größer als der definierte Wert cnt - 1 in MATRIX.SIZE sein.

IO-Zusatzbefehle für LED-Basic-Komponenten

Anzahl und Parameter der Zusatzbefehle können sich in jeder Version ändern. Bitte deshalb unbedingt die Hinweise bei den Updates beachten. Beachten Sie bitte, dass die benutzte LED-Basic-Komponente eventuell nicht alle Funktionen ermöglicht.

Die Anzahl der Parameter muss unbedingt stimmen. Ist kein Parameter erforderlich, müssen jedoch wie in der C-Syntax die Klammern dem Befehl folgen.

Als Parameter können neben numerischen Werten natürlich auch Variablen oder berechnete Ausdrücke eingesetzt werden.

In der Beschreibung der LED-Basic-Komponenten finden Sie die Liste der unterstützten IO-Befehle.

IO.WAITKEY

```
IO.waitkey()
```

Es wird auf einen beliebigen Tastendruck gewartet. Die Warte-LED blinkt, wenn dies nicht über die Konfigurationszeile mit Sx abgeschaltet ist.

IO.GETKEY

```
<VAR>=IO.getkey()
```

Während des Programmablaufes kann der Status der Tasten abgefragt werden. Die Anzahl der Tasten ist von der verwendeten LED-Basic-Komponente abhängig.

Der Tastendruck bleibt so lange gespeichert, bis eine Abfrage erfolgte.

Taste	Wert
Keine	0
1	1 (Bit 0)
2	2 (Bit 1)
3	4 (Bit 2)
4	8 (Bit 3)
…	…

FALSCH:

```
if IO.getkey() = 1 then goto 1000
if IO.getkey() = 2 then goto 2000
if IO.getkey() = 4 then goto 3000
```

Durch die erste **getkey**-Abfrage wird der Tastaturstatus zurückgesetzt, die zweite Abfrage wird also niemals ausgeführt, auch wenn Taste 2 zuvor gedrückt war.

RICHTIG:

```
k = IO.getkey()
if k = 1 then goto 1000
if k = 2 then goto 2000
if k = 4 then goto 3000
```

Hier wird der Tastaturstatus in eine Variable eingelesen und kann dadurch auf mehrere verschiedene Werte abgefragt werden.

IO.KEYSTATE

```
<VAR>=IO.keystate()
```

Im Gegensatz zu **getkey** kann mit dieser Funktion abgefragt werden, ob eine oder mehrere Tasten gedrückt sind. Hierbei werden die Tasten bitweise (siehe IO.getkey) verknüpft. Der Rückgabewert bleibt so lange auf dem Tastaturwert, wie die Taste gedrückt ist. Erst wenn die Taste losgelassen wird, ist der Rückgabewert Null. Es gelten dieselben Wertebereiche wie bei **getkey**. Diese Funktion ist besonders für Eingangsports wichtig, wenn anliegende Signale (High, Low) verarbeitet werden sollen. Alle Eingangsports besitzen integrierte Pull-Up Widerstände und sind per Sofware entprellt.

IO.SETPORT

```
IO.setport(port)
```

Setzt die Ausgangsports auf High-Pegel (ca. 3.3V). Welche der Ports beeinflusst werden, bestimmt der Wert in **port.** Die Anzahl der Ports ist abhängig von der LED-Basic-Komponente.

Port 1 = Bit 0, Port 2 = Bit 1, Port 3 = Bit 2, usw.

IO.CLRPORT

```
IO.clrport(port)
```

Setzt die Ausgangsports auf Low-Pegel (0 V). Welche der Ports beeinflusst werden, bestimmt der Wert in **port**.

Port 1 = Bit 0, Port 2 = Bit 1, Port 3 = Bit 2, usw.

IO.GETRTC

```
<VAR>=IO.getrtc(idx)
```

Liest die Werte der Echtzeituhr (RTC) aus (falls vorhanden).

Werte: idx = [0..6], [0..7] bei Komponenten mit AutoDST

idx	Bedeutung	Bereich
0	Sekunde	0 - 59
1	Minute	0 - 59
2	Stunde	0 - 23
3	Tag	1 - 28/29/30/31
4	Monat	1 - 12
5	Jahr	2000 - 20xx
6	Wochentag	1 - 7 (1 = Montag)
7*	DST-Kennung	0x80 = AutoDST, 0x01 = IsDST

Hinweis für NixieCron-Komponenten: Um die Uhrzeit abzufragen immer erst die Sekunde lesen. Hierdurch werden auch alle anderen Zeit/Datum-Werte aus dem Uhrenchip gelesen und zwischengespeichert. Damit wird ein eventueller Zeitwechsel zwischen dem Lesen verschiedener Register vermieden.

Beispiel:
s = IO.getrtc(0)
m = IO.getrtc(1)
Wenn nur die Minute benötigt wird, erst die Sekunde lesen.

*Wenn die Komponente die automatische Sommer-/Winterzeitumschaltung unterstützt (AutoDST), kann idx(7) als Kennung abgefragt werden. Bit 7 ist gesetzt, wenn die automatische Umschaltung aktiv ist. Bit 0 ist 0 bei Winterzeit und 1 bei Sommerzeit.

IO.SETRTC

```
IO.setrtc(idx, val)
```

Setzt die Werte der Echtzeituhr (falls vorhanden). Hier ist es sinnvoll zunächst den gewünschten Wert zu lesen, diesen zu verändern und dann wieder zu schreiben. Wenn mehrere Werte geschrieben werden sollen, mit den niedrigsten **idx**-Werten beginnen.

Werte: idx = [0..5] (wie bei **GETRTC,** 6 (Wochentag) wird berechnet und kann nicht gesetzt werden)

*Wenn die Komponente die automatische Sommer-/Winterzeitumschaltung unterstützt (AutoDST), wird die Umschaltung mit Setzen von Bit 7 von idx 7 aktiviert.

IO.GETLDR

```
<VAR>=IO.getldr()
```

Liest den Wert des Helligkeitssensors (LDR) aus (falls vorhanden). Die Ergebniswerte liegen je nach Helligkeit zwischen 0 und 255, wobei technisch bedingt die obersten und untersten Werte nie erreicht werden.

Hinweis: In neueren Komponenten werden Fototransistoren oder Fotodioden anstatt LDR eingesetzt. Der Befehl und die Rückgabewerte sind jedoch identisch.

IO.GETIR

```
<VAR>=IO.getir()
```

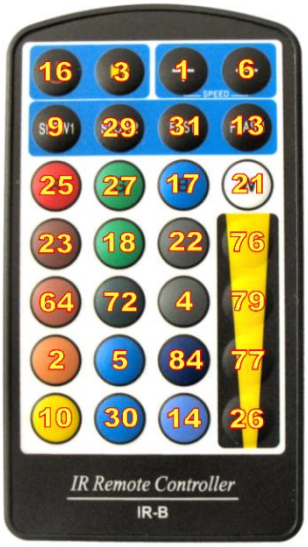
Liest die Wert des Infrarot-Empfängers aus (falls vorhanden). Ein Wert 0 bedeutet, dass keine Daten vorhanden sind. Der gelesene Wert sollte in eine Variable geschrieben werden (wie bei der GETKEY-Funktion). Es wird ein 16-Bit-Wert ausgelesen, der eigentliche Tastenwert steht dabei in den unteren 8 Bits des Wertes, in den oberen 8 Bit ist die Anzahl der Wiederholungen zu finden, wenn eine Taste länger gedrückt wird.

Beispiel:

```
z = IO.getir()
k = z & 0xFF      ' Tastenwert ausmaskieren
n = z / 256       ' In n steht der Wiederholungswert
```

Die Standard-Infrarot-Fernbedienung für alle Komponenten, die einen Infrarot-Empfänger besitzen. Die dezimalen Tastenwerte sind auf der Abbildung rechts zu finden. Die Werte sind vom Hersteller der Fernbedienung vorgegeben und können nicht verändert werden. Alle LED-Basic-Komponenten mit Infrarot-Sensor können nur Signale dieser Fernbedienung empfangen und verarbeiten.

Beachten Sie bitte, dass Farbabweichungen der LEDs gegenüber den Farbtasten möglich sind.



IO.GETTEMP

```
<VAR>=IO.gettemp()
```

Liest den Wert des Temperatursensors DS18B20 aus (falls vorhanden). Ergebnis ist ein Wert in 0,1°C Auflösung. Aufgrund der relativ langen Messdauer sollte die Abfrage nicht in kürzeren Abständen als 1 Sekunde erfolgen. Erst ein Lesen des Wertes startet einen neuen Messvorgang. Der erste gelesene Wert sollte deswegen verworfen werden.

Beispiele: 0 = 0°C, 217 = 21.7°C, -81 = -8.1°C

Hinweis: Eine höhere Auflösung als 0,1°C ist aufgrund des verwendeten Temperatursensors nicht möglich. Die Genauigkeit wird vom Hersteller mit ±0,5°C im Bereich -10°C - +85°C angegeben.

IO.XTEMPCNT

```
<VAR>=IO.xtempcnt()
```

Bei Komponenten, die mehrere Temperatursensoren des Typs DS18B20 unterstützen (z.B. Temperatur-Sensor-Interface) wird mit diesem Befehl die Anzahl der erkannten Sensoren ausgegeben. Maximal 8 Sensoren können angeschlossen werden. Wenn die Sensoren mit parasitärer Stromversorgung betrieben werden, sind aufgrund der höheren Belastung der Datenleitung eventuell nur weniger Sensoren möglich (ausprobieren). Beachten Sie bitte, dass die Sensoren nur bei einem Neustart eingelesen und identifiziert werden. Um Beschädigungen an der Hardware und an den Sensoren zu vermeiden sollten Sie die Sensoren nur im stromlosen Zustand anschließen oder entfernen.

IO.XTEMPVAL

```
<VAR>=IO.xtempval(nr, idx)
```

Bei Komponenten, die mehrere Temperatursensoren des Typs DS18B20 unterstützen, können mit diesem Befehl für jeden Sensor (**nr)** verschiedene Werte (**idx**) ausgelesen werden.

Werte: nr = [0..x] (x = Wert aus IO.xtempcnt – 1), idx = 0..10

idx	Bedeutung
0	0 = Temperaturwert nicht gültig, <> 0 = Temperaturwert gültig
1	Temperaturwert in 0.1°C Auflösung (siehe IO.gettemp)
2	0 = parasitäre Stromversorgung, 1 = externe Stromversorgung
3..10	ROM-ID des Sensors

IO.BEEP

```
IO.beep(val)
```

Erzeugt einen Ton über den Lautsprecher (falls vorhanden).

val	Bedeutung
0	Ton aus
1..36	Note
ab 200	Frequenz = val Hz

Frequenzen unter 200 Hz können systembedingt nicht erzeugt werden.

Folgende Notenwerte sind fest gespeichert:

val	Note	val	Note	val	Note
1	C2	13	C3	25	C4
2	C2#	14	C3#	26	C4#
3	D2	15	D3	27	D4
4	D2#	16	D3#	28	D4#
5	E2	17	E3	29	E4
6	F2	18	F3	30	F4
7	F2#	19	F3#	31	F4#
8	G2	20	G3	32	G4
9	G2#	21	G3#	33	G4#
10	A2	22	A3	34	A4
11	A2#	23	A3#	35	A4#
12	H2	24	H3	36	H4

IO.GETENC

```
<VAR>=IO.getenc()
```

Liest den Wert des Drehimpulsgebers aus (falls vorhanden). Der Wertebereich wird über den folgenden Befehl **IO.SETENC** festgelegt.

IO.SETENC

```
IO.setenc(pos, max, stop)
```

Setzt die Werte-Parameter für den Drehimpulsgeber (falls vorhanden).

Werte: pos=[0..max], max=[1..65535], stop=[0/1]

Der Drehimpulsgeber liefert Werte zwischen 0 und **max**. Der aktuelle Startwert wird mit **pos** festgelegt, dieser muss immer <= **max** sein. Bei **stop** = 1 bleibt der Maximalwert bestehen, wenn der Impulsgeber weiter gedreht wird. Wenn **stop** = 0 beginnt der Zähler wieder bei 0 wenn der Maximalwert überschritten ist und wieder bei **max** wenn 0 unterschritten werden soll.

IO.GETPOTI

```
<var>=IO.getpoti(idx)
```

Liest den Wert eines Potentiometers oder dessen umgewandelten Wert aus (falls vorhanden). Welche und wie viele Werte ausgelesen werden können, hängt von der Komponente ab.

Werte: idx=[0..x], x ist abhängig von der verwendeten Komponente

IO.GETADC

```
<var>=IO.getadc(idx)
```

Liest einen Analogwert. Welche Werte ausgelesen werden können, hängt von der Komponente ab.

Werte: idx=[0..x], x ist abhängig von der verwendeten Komponente

IO.EEREAD

```
<VAR>=IO.eeread(adr)
```

Liest den Inhalt des EEPROM an Adresse **adr** aus (falls vorhanden). Rückgabe ist ein 16-Bit-Wert. Im Auslieferzustand sind alle Speicherstellen des EEPROM auf 0xFFFF (-1). Ein ungültiger Wert von **adr** liefert immer den Wert 0.

Werte: adr=[0..x], x ist abhängig von der verwendeten Komponente und ist in der Liste der IO-Befehle angegeben.

IO.EEWRITE

```
IO.eewrite(adr, data)
```

Schreibt das EEPROM an Adresse **adr** mit dem 16-Bit Wert in **data** (falls vorhanden). Beachten Sie bitte, dass der Schreibvorgang einer Speicherstelle eventuell mehrere Millisekunden benötigt und damit den Programmablauf verzögert. Um die Lebensdauer des EEPROM zu verlängern, sollten Schreibvorgänge vermieden werden und nur geänderte Werte gespeichert werden.

Werte: adr=[0..x], x ist abhängig von der verwendeten Komponente und ist in der Liste der IO-Befehle angegeben. data=[-32768..32767]

IO.SYS

```
<VAR>=IO.sys(par1, par2)
```

Universeller Befehl zum Setzen oder Lesen von Systemparametern (falls vorhanden). Es müssen immer beide Parameter übergeben werden, auch wenn sie nicht benötigt werden.

Die gängigen Werte für **par1** sind hier aufgeführt. Sollte eine Komponente noch über andere Werte verfügen, sind diese dort extra beschrieben.

[COM] Communication, 0..64

Über diese Befehle können einzelne oder mehrere Zeichen an die Komponente gesendet werden. Diese kann per LED-Basic Befehle die Daten auswerten und eventuell Ergebnisse über den Print-Befehl zurückgeben.

<var> = IO.sys(0, 0)
Liest die Anzahl der über den virtuellen COM-Port empfangenen Zeichen aus.
0 = keine Zeichen empfangen
1..64 = Zeichen befinden sich im Empfangsspeicher.

IO.sys(0, 1)
Setzt die Anzahl der empfangenen Zeichen auf null.

<var> = IO.sys(n, 0)
Liest den Empfangsspeicher an Position n aus.
Gültige Werte für n: 1..x (x = Anzahl der empfangenen Zeichen, max. 64).

Über das Terminal (ab Version 15.1.14) können die ASCII-Werte der gedrückten Taste übermittelt werden. Es wird dabei jeweils 1 Zeichen in den Empfangsspeicher geschrieben.
Beispiel: Taste A wird gedrückt. IO.sys(0,0) liefert den Wert 1 = 1 Wert im Speicher. IO.sys(1, 0) liefert den Wert 65 (0x41) = ASCII-Code für „A“. Danach sollte der Empfangszähler mit IO.sys(0, 1) auf null gesetzt werden.

[TEMP] Lese Temperatur - Sensor, 97

<var> = IO.sys(97, 0)

Liest den Temperatur-Wert aus dem Uhrenchip DS3231. Der Rückgabewert ist die aktuell gemessene Temperatur in 0.25° Auflösung. Der ganzzahlige Temperaturwert ergibt sich aus **var / 4**, die Nachkommastelle aus **var & 3** (0 = 0.25, 1 = 0.5, 2 = 0.75). Es werden Werte bis 127.75° ausgegeben, Werte >= 128 bedeutet Minusgrade.

HINWEISE: Nur bei einigen Uhren mit DS3231-Chip. Aufgrund der Eigenerwärmung der Schaltung wird meist ein zu hoher Wert ausgegeben. Sehen Sie deshalb in Ihrem Programm die Einstellung eines Differenzwertes vor. Achtung: Der Temperaturwert wird nur alle 64 Sekunden aktualisiert.

[CALIB] Calibration, 99

IO.sys(99, c)
Kalibrierung der Echtzeituhr. Sollte die Integrierte Echtzeituhr vor- oder nachgehen, kann dies hier ausgeglichen werden.
Gültige Werte für c: -510 bis 510
Ein negativer Wert lässt die Uhr langsamer, ein positiver Wert schneller laufen.

Ändern Sie die Werte in 10er, 20er oder 50er Schritten und synchronisieren dann die Uhr mit dem PC. Da die Änderungen nur eine geringe Auswirkung auf die Geschwindigkeit der Uhr haben, kann es mehrere Stunden oder sogar Tage dauern, bis man eine Änderung der Geschwindigkeit erkennt.

[DFP] DFPlayer Mini, 1000…1101

Der DFPlayer Mini spielt MP3-Dateien von einer Micro-SD-Karte ab.

<val>=IO.sys(1000, 0)	Status lesen, **val**: 0 = busy
IO.sys(1000, 1)	Status zurücksetzen
<val>=IO.sys(1000, 1)	Rückgabewert lesen und Status zurücksetzen
IO.sys(1000 + cmd, par)	**cmd:** entspricht CMD 0x01 bis 0x4D des DFPlayer Mini **par:** 16 Bit Parameter (0 wenn nicht benötigt)
IO.sys(1100, 0)	Warteschlange (Queue) löschen
IO.sys(1100, 1)	Warteschlange ausführen
IO.sys(1101, par)	Datei zur Warteschlange hinzufügen (max. 16 Einträge)

Beispiele:

```
IO.sys(1012, 0)          DF-Player Reset (knackt zweimal)
IO.sys(1006, 20)         Stelle Lautstärke auf 20 (gültige Werte 0..30)
```

Nach diesen Befehlen muss der Busy-Status abgefragt werden.

```
IO.sys(1015, (2 * 256) + 45)     Spiele MP3-Datei [045.mp3] in Verzeichnis [02] ab
```

Busy kann abgefragt werden um das Ende der Datei abzuwarten.
Die laufende Datei wird sofort beendet, wenn ein neuer Befehl gesendet wird.

```
IO.sys(1100, 0)                  Warteschlange löschen
IO.sys(1101, (2 * 256) + 1)      MP3-Datei [001.mp3] in Verzeichnis [02] hinzufügen
IO.sys(1101, (2 * 256) + 3)      MP3-Datei [003.mp3] in Verzeichnis [02] hinzufügen
IO.sys(1101, (2 * 256) + 24)     MP3-Datei [024.mp3] in Verzeichnis [02] hinzufügen
IO.sys(1101, (2 * 256) + 31)     MP3-Datei [031.mp3] in Verzeichnis [02] hinzufügen
IO.sys(1101, (1 * 256) + 1)      MP3-Datei [001.mp3] in Verzeichnis [01] hinzufügen
IO.sys(1100, 1)                  Warteschlange ausführen
```

Beispiel für Busy-Status lesen:

```
1000:
        s = IO.sys(1000, 0)      Status lesen
        if s = 0 goto 1000       Wenn busy = 0, status lesen wiederholen
        IO.sys(1000, 1)          Status zurücksetzen
        return                   und zurück
```

Viele Befehle können nur ausgeführt werden, wenn eine SD-Karte im DF-Player eingesteckt ist.

Link zur DF-Player Anleitung (englisch):
https://www.dfrobot.com/wiki/index.php/DFPlayer_Mini_SKU:DFR0299

IO.BT

```
<VAR>=IO.sys(par1, par2)
```

Daten vom Bluetooth-Modul (BLE) empfangen und senden. Über die Parameter par1 und par2 werden verschiedene Funktionen ausgewählt:

par1	par2	Funktion	Rückgabewert
0	0	Verbindungsstatus	0=nicht verbunden 1=verbunden
0	1	Lösche Empfangs-FIFO	0 (kein Wert)
0	2	Anzahl der Bytes im Empfangs-FIFO lesen	max. 256
0	3	Lese Byte (8-Bit) aus Empfangs-FIFO	8-Bit Wert
0	4	Lese Wort (16-Bit) aus Empfangs-FIFO	16-Bit Wert
0	11	Lösche Sende-FIFO	0 (kein Wert)
0	12	Anzahl der Bytes im Sende-FIFO lesen	Max. 256
0	15	Daten im Sende-FIFO senden	0 (kein Wert)
0	20	BtCode-Low-Wert (siehe Beschreibung unten)	8-Bit Wert
0	21	BtCode-Mid-Wert (siehe Beschreibung unten)	8-Bit Wert
0	22	BtCode-High-Wert (siehe Beschreibung unten)	8-Bit Wert
1	Byte	Schreibe Byte (8-Bit) in Sende-FIFO	0 (kein Wert)
2	Wort	Schreibe Wort (16-Bit) in Sende-FIFO	0 (kein Wert)

Die Bluetooth-Module sind über das BLE-Protokoll (Bluetooth Low Energy) von jedem aktuellen Smartphone erreichbar. Die Kennung (BtCode) für die Verbindung ist folgendermaßen aufgebaut:

CRBT-123456

CRBT steht für „Cronios Bluetooth“, die 6-stellige Ziffernfolge ist bei jedem Modul unterschiedlich und als hexadezimaler Wert zu interpretieren. Da LED-Basic nur mit maximal 16-Bit Werten rechnen kann, kann der BtCode-Wert als drei einzelne 8-Bit-Werte gelesen werden: 12 = High-Wert, 34 = Mid-Wert, 56 = Low-Wert.

Die Sende- und Empfangs-FIFOs haben eine Kapazität von 256 Bytes. Ist ein FIFO voll, werden neue Daten ignoriert.

Die Bluetooth-Module haben eine relativ lange Startzeit. Bevor der Verbindungsstatus gelesen und Daten empfangen oder gesendet werden, sollte nach Programmstart eine Wartezeit von mindestens 700ms eingehalten werden.

Laufzeit-Fehlermeldungen

Über das Terminal (wenn es von der LED-Basic-Komponente unterstützt wird) werden Laufzeitfehler ausgegeben.

```
?ERROR xx IN LINE zzzzz
```

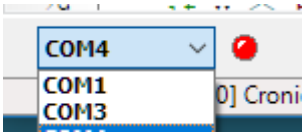
Fehler	Bedeutung
11	Unbekannter Token (sollte nicht vorkommen)
12	Falsche Adresse (sollte nicht vorkommen)
13	Zu viele verschachtelte GOSUB-Befehle
14	RETURN ohne GOSUB
15	Wert darf nicht 0 sein
16	Zu viele verschachtelte FOR-NEXT-Schleifen
17	Falsche Werte bei TO/DOWNTO
18	Next-Variable ungültig
19	Falscher Wert oder ungültiger LED-Befehl
20	Falscher Wert oder ungültiger IO-Befehl
22	Falscher Wert oder ungültiger MATRIX-Befehl

Bei LED-Basic-Komponenten ohne Terminal (z.B. LED-Button 12) wird der Fehlercode durch die Anzahl der rot blinkenden LEDs (+9) angezeigt (6 blinkende LEDs bedeutet Fehler 15).

Led-Basic-Komponenten

Treiberinstallation

Jede LED-Basic-Komponente mit USB-Anschluss und auch der Programmieradapter „Prog-SB“ werden über einen virtuellen COM-Port angesprochen. Unter Windows 7 und 8.x muss hierzu die Treiberdatei „LedBasic.inf“ installiert werden. Im Gerätemanager von Windows den entsprechenden Eintrag anklicken, im Reiter "Treiber" auf "Treiber aktualisieren". Dann die Datei LED-Basic.inf aus dem LED-Basic Installationsverzeichnis auswählen.



Sollten Sie noch Windows 8.x benutzen, müssen Sie eventuell das „Erzwingen der Treibersignatur deaktivieren“. Wie das funktioniert, können sie leicht mit einer Google-Suche herausfinden. Unter Windows 10 ist keine Installation eines Treibers erforderlich. Hier meldet sich jede Komponente als „Serielles USB-Gerät (COMx)“ an. Welcher COM-Port für Ihre verwendetes Komponente gültig ist, können Sie am besten testen, indem Sie im LED-Basic-Editor die Liste der COM-Ports anklicken bevor und nachdem Sie die Komponente angesteckt haben. Der COM-Port der hinzugekommen ist, sollte ausgewählt werden.

Unterschiedliche Komponenten

Für LED-Basic sind verschiedene Komponenten verfügbar. Diese unterscheiden sich in der Regel durch die Möglichkeiten, unterschiedliche Typen und Mengen von LEDs anzusteuern sowie der Anschluss von Sensoren, Tastern und der internen Peripherie wie Echtzeituhr oder IO-Ports.

Jede LED-Basic-Komponente benutzt denselben BASIC-Grundbefehlssatz. Es unterscheiden sich nur die LED-, IO- und MATRIX-Zusatzbefehle. Welche Befehle Ihre LED-Basic -Komponente versteht, entnehmen Sie bitte der weiter unten folgenden Beschreibung. LED-Basic ist universell, es muss auch keine LED-Ansteuerung vorhanden sein, um LED-Basic zu verwenden. LED-Basic kann auch in Komponenten verwendet werden, die nur Sensoren abfragt und Schaltvorgänge auslöst.

Es gibt eine Reihe von LED-Basic Komponenten, die über einen USB-Anschluss verfügen. Dieser wird dann in der Regel gleichzeitig zur Programmierung und als Terminal-Ausgang verwendet. Sollte die Komponente über keinen USB-Anschluss verfügen, ist ein USB-Programmieradapter „Prog-SB“ erforderlich. Dieser verbindet den PC mit der 6-poligen Programmierleiste der LED-Basic -Komponente.

Liste der von LED-Basic unterstützten Komponenten

Es folgt eine Beschreibung der von LED-Basic-Editor unterstützten Komponenten. Beachten Sie bitte, dass es teilweise identische Komponenten in unterschiedlichen Bauformen gibt. Neben der Anschlussbelegung zu jeder LED-Basic-Komponente folgt eine Liste der möglichen LED- und IO-Befehle sowie der Parameter in der Konfigurationszeile.

Hinweise: Alle Input/Output-Pins haben einen Pegel von maximal 3,3 Volt. Um eine Zerstörung der Hardware vorzubeugen, sollten Sie Vorwiderstände oder Pegelwandler benutzen, wenn höhere Spannungen angelegt werden. Input-Pins haben integrierte Pullup-Widerstände, ein Taster wird einfach vom Input-Pin gegen GND angeschlossen. Output-Pins liegen im Ruhezustand auf Low-Pegel (0 Volt). Sie können im High-Pegel nur wenige Milliampere treiben, eine LED kann direkt über einen passenden Vorwiderstand angeschlossen werden. Sollten Sie einen höheren Strom benötigen (z.B. zum Ansteuern eines Relais), sind unbedingt Transistoren oder IC-Treiber erforderlich.

LED-Ausgänge für serielle RGB-LEDs (WS2812, APA102 o.A.) haben immer einen Pegel von 5 Volt.

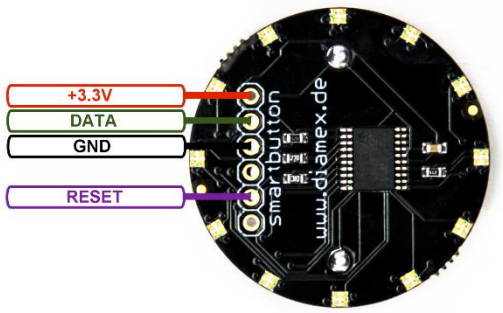
Alle nicht bezeichneten Pins sind unbenutzt oder dienen zur Programmierung und zum Test bei der Herstellung.

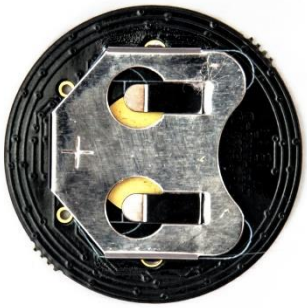
Komponenten-Bootloader

In allen LED-Basic-Komponenten mit USB-Port befindet sich ein USB-Bootloader, über den das Bios und die LED-Basic-Daten verwaltet und hochgeladen werden kann. Dieser Bootloader ist immer aktiv nach Anlegen der Stromversorgung, es wird geprüft, ob sich ein gültiges Bios im Speicher befindet und ob LED-Basic-Daten vorhanden sind, ist alles in Ordnung, wird der Bootloader verlassen und das Bios zusammen mit den LED-Basic-Daten gestartet. Sollte es im Extremfall vorkommen, dass sich das Bios durch inkonsistente Daten „aufhängt“, gibt es bei allen Komponenten einen „Notschalter“ um den Bootloader zu aktivieren und dann über die LED-Basic Entwicklungsumgebung neue Bios- oder LED-Basic-Daten hochzuladen. Dieser „Notschalter“ befindet sich bei den Komponenten auf einer der Taster und/oder auf einem der IO-Pins. Halten Sie den Taster gedrückt oder legen den IO-Pin auf Massepotential (GND) und legen die Stromversorgung an. In der Regel wird durch regelmäßiges Blinken einer LED angezeigt, dass der Bootloader gestartet ist. Sind mehrere Taster vorhanden, testen Sie am besten alle Taster durch, welches der Richtige ist.

Bei Komponenten mit seriellem Bootloader, die über Prog-SB programmiert werden, wird der Bootloader immer automatisch durch den Programmer aktiviert.

LED-Badge (12 LEDs)





BADGE = Abzeichen, Sticker, Plakette

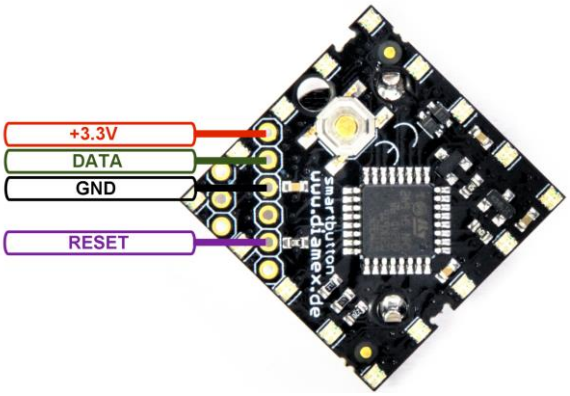
12 RGB-LEDs ohne PWM. Ein- und Ausschalten durch Einsetzen und Herausnehmen der Batterie möglich.

Programmierung über Prog-SB (+3.3V, DATA, GND, RESET).

System-Code: 3110

Konfigurationszeile	LED-Befehle	IO-Befehle
KEINE	SETLED, SETALL	KEINE

LED-Badge (16 LEDs)



Unterschiedliche Bauformen sind erhältlich. Die Pinbelegung für den Programmieradapter ist bei allen Bauformen identisch.

BADGE = Abzeichen, Sticker, Plakette

16 RGB-LEDs ohne PWM. Einschalten durch kurzen Tastendruck, Ausschalten durch Tastendruck länger als 2 Sekunden. Eine Tastenabfrage ist per Basic-Befehl möglich. Terminal Print- und Fehler-Ausgabe über Prog-SB.

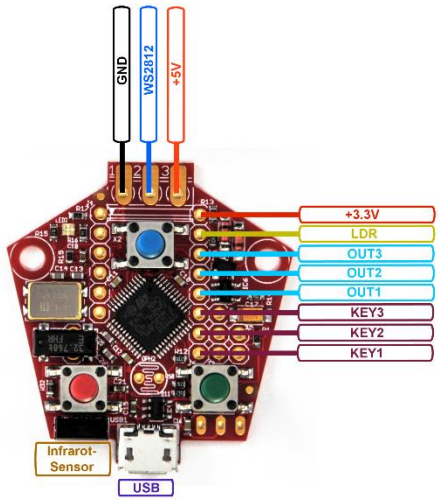
Programmierung über Prog-SB (+3.3V, DATA, GND, RESET, TERMINAL).

System-Code: 3120

Konfigurationszeile	LED-Befehle	IO-Befehle
P…	SETLED, SETALL	WAITKEY, GETKEY CLRPORT

Durch IO.clrport(0) wird der LED-Button abgeschaltet. Hiermit kann z.B. nach einer per Basic vorgegebenen Anzahl von Schleifendurchläufen die Komponente abgeschaltet und damit Energie gespart werden.

Basic-Pentagon-Board



Die Basis-Schaltung für LED-Basic. Für maximal 256 LEDs mit integrierter PWM (WS2812, SK6812 und kompatible). Auch für RGBW-LEDs geeignet. Optional mit Infrarot-Fernbedienung steuerbar. Terminal Print- und Fehler-Ausgabe über USB. 3 Tasten oder Eingangspins, 3 programmierbare Ausgangspins.

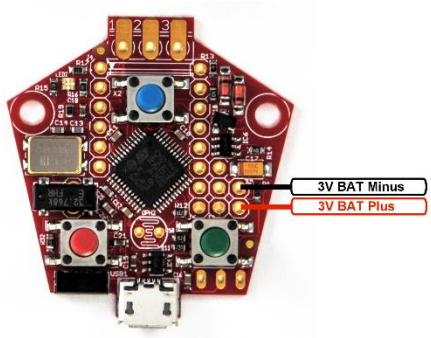
Stromversorgung über USB- oder LED-Anschluss möglich.

Ein LDR (Fotowiderstand) kann an direkt auf die Platine (OPH2) aufgelötet oder an die Pins LDR und +3,3V angeschlossen werden.

Programmierung über USB-Anschluss.

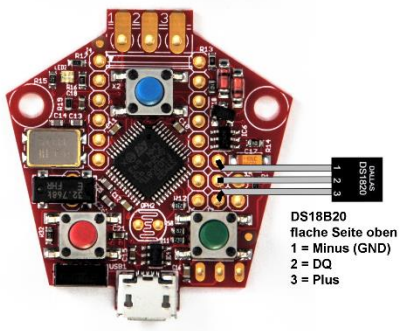
System-Code: 3130

Konfigurationszeile	LED-Befehle	IO-Befehle
L... C... M... P... S... F... Lxxx: Maximum = 256	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR GETIR SETPORT, CLRPORT GETTEMP

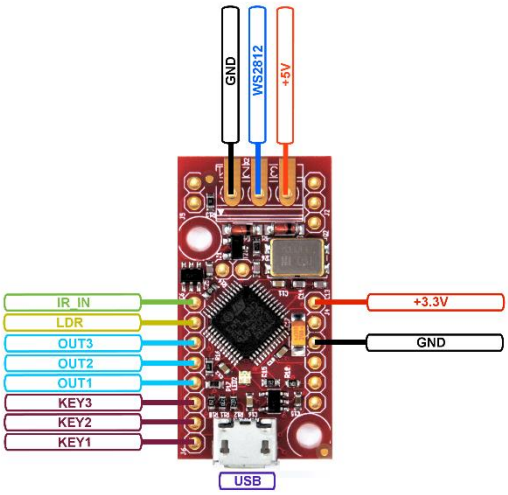


Wenn die Echtzeituhr benutzt werden soll, kann eine 3 Volt Stützbatterie (z.B. CR2032 oder 2 x AAA-Zellen) an die bezeichneten Pins angeschlossen werden.

Anschluss eines DS18B20 Temperatursensors an das Pentagon-Board.



Basic-Budget-Board



Die Basis-Schaltung für LED-Basic in besonders leichter Ausführung (z.B. für Flugmodelle geeignet). Softwarekompatibel mit dem Basic-Pentagon-Board. Für maximal 256 LEDs mit integrierter PWM (WS2812, SK6812 und kompatible). Auch für RGBW-LEDs geeignet. Terminal Print- und Fehler-Ausgabe über USB. 3 Tasten oder Eingangspins, 3 programmierbare Ausgangspins.

Stromversorgung über USB- oder LED-Anschluss möglich.

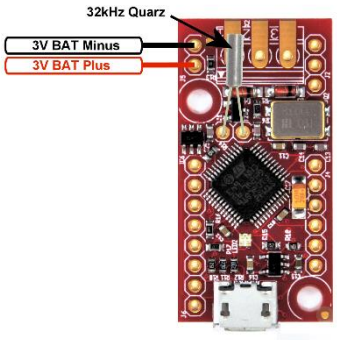
Ein LDR (Fotowiderstand) kann an die Pins LDR und +3,3V angeschlossen werden.

Ein Infrarot-Empfänger kann an den Pin IR_IN angeschlossen werden (siehe Anschlussplan beim Basic-Booster).

Programmierung über USB-Anschluss.

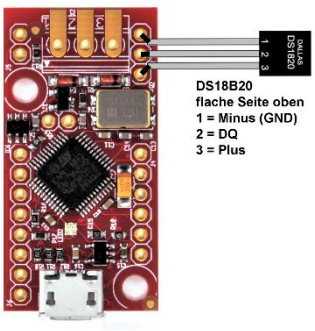
System-Code: 3130

Konfigurationszeile	LED-Befehle	IO-Befehle
L... C... M... P... S... F... Lxxx: Maximum = 256	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR GETIR SETPORT, CLRPORT GETTEMP

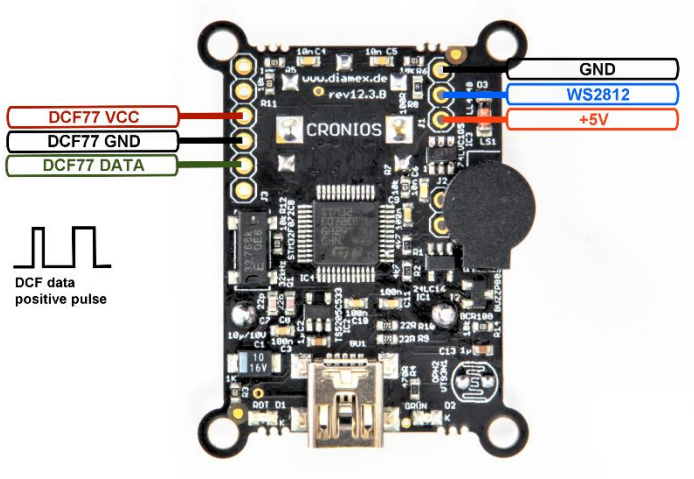


Wenn die Echtzeituhr benutzt werden soll, kann eine 3 Volt Stützbatterie (z.B. CR2032 oder 2 x AAA-Zellen) an die bezeichneten Pins angeschlossen werden. Zusätzlich muss ein 32,768 kHz Quarz aufgelötet werden.

Anschluss eines DS18B20 Temperatursensors an das Budget-Board.



Cronios 1



Die Cronios 1 Komponente mit LED-Basic-Software. Für maximal 256 LEDs mit integrierter PWM (WS2812, SK6812 und kompatible). Auch für RGBW-LEDs geeignet. Terminal Print- und Fehler-Ausgabe über USB. Integrierte Echtzeituhr mit Stützbatterie, Drehimpulsgeber mit Tastenfunktion, LDR und Lautsprecher sind auf dem Board vorhanden. EEPROM mit 1024 Speicherstellen. Stromversorgung über USB- oder LED-Anschluss möglich. Ab LED-Basic-Version 15.1.12 kann zur Zeitsynchronisation ein DCF77-Modul angeschlossen werden.

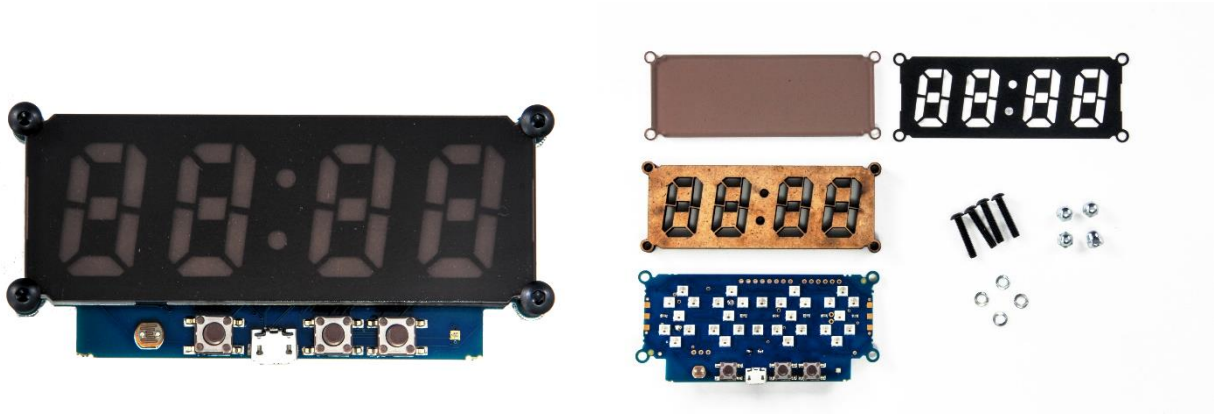
Programmierung über USB-Anschluss.

System-Code: 3140

Konfigurationszeile	LED-Befehle	IO-Befehle
L... C... M... P... S... F... Lxxx: Maximum = 256	Alle Befehle Für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR BEEP GETENC, SETENC EEREAD, EEWRITE [0..1023] SYS [COM, CALIB]

Hinweis: An die Cronios 1-Komponente kann nur das DCF77-Modul angeschlossen werden.
Das GPS->DCF-Modul und das WLAN->DCF-Modul funktioniert aufgrund der sehr hohen Stromaufnahme nicht.

Cronios-Segmenta



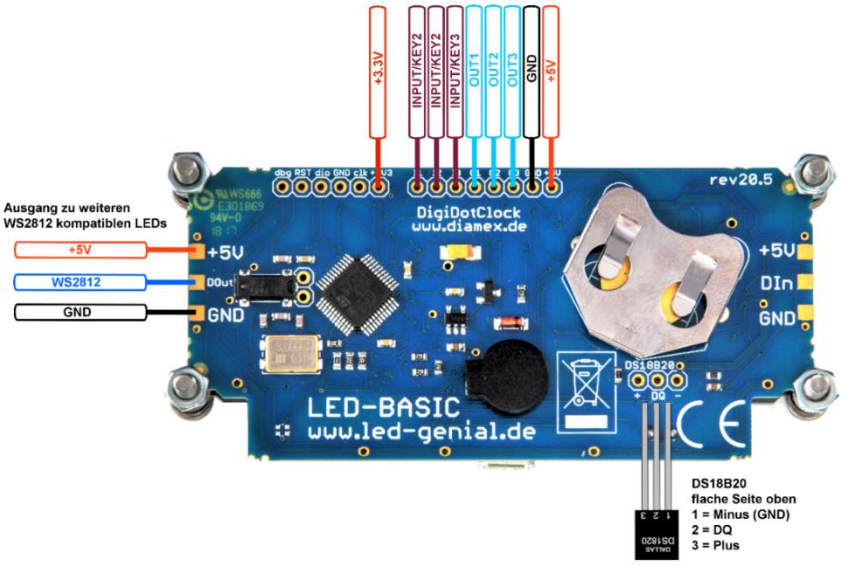
Bausatz für eine 4-stellige 7-Segment-Uhr mit farbigen Ziffern, bestehend aus SK6812-Mini-LEDs. Terminal Print- und Fehler-Ausgabe über USB. Integrierte Echtzeituhr mit Stützbatterie, 3 Tasten oder Eingangspins, 3 Ausgangspins für Schaltsignale, LDR und Lautsprecher sind auf dem Board vorhanden. Anschluss für optionalen Temperatursensor DS18B20. Stromversorgung über USB-Anschluss. Zusätzlich zu fest verdrahteten 30 LEDs können über den Ausgang (Dout) bis zu insgesamt 256 LEDs angesteuert werden.

Programmierung über USB-Anschluss.

System-Code: 3150

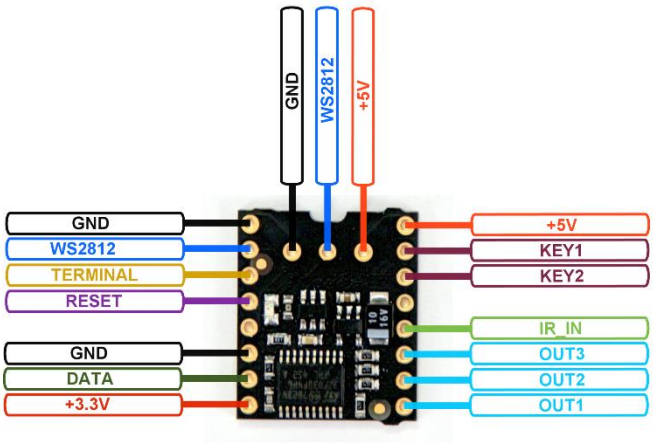
Konfigurationszeile	LED-Befehle	IO-Befehle
L… M… P… S… F… Lxxx: Maximum = 256	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR SETPORT, CLRPORT GETTEMP BEEP EEREAD, EEWRITE [0..15]

Hinweis: Aufgrund der recht hohen Stromaufnahme kann es passieren, dass die Anzeige nicht sichtbar ist. Versuchen Sie es dann an einem anderen USB-Anschluss oder verwenden ein USB-Hub mit eigenem Netzteil.



Basic-Booster

Mini-Interface für LED-Basic. Für maximal 64 LEDs mit integrierter PWM (WS2812, SK6812 und kompatible). Auch für RGBW-LEDs geeignet. Terminal Print- und Fehler-Ausgabe über Prog-SB. Über 2 Eingangspins für Taster (KEY1,2) können Funktionen ausgelöst werden, 3 Ausgangspins (OUT1,2,3) sind für Schaltvorgänge vorhanden. Optional kann ein Infrarot-Empfänger an IR_IN angeschlossen werden, die hierfür erforderlichen Bauteile sind im Set mit der Fernbedienung erhältlich. Die Stromversorgung geschieht über einen der +5V-Anschlüsse.



Programmierung über Prog-SB (+3.3V, DATA, GND, RESET, TERMINAL).

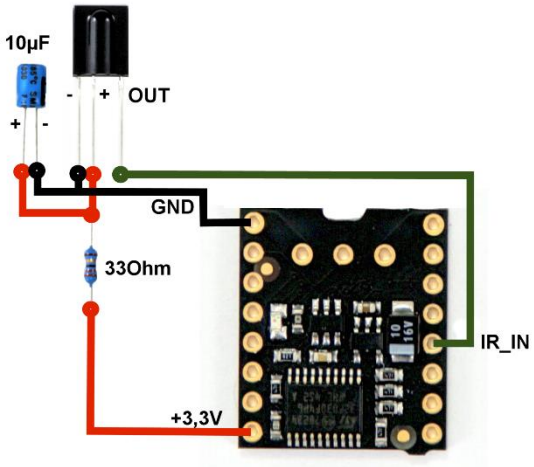
System-Code: 3160

Konfigurationszeile	LED-Befehle	IO-Befehle
L... C... M... P... S... F... Lxxx: Maximum = 64	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT GETIR

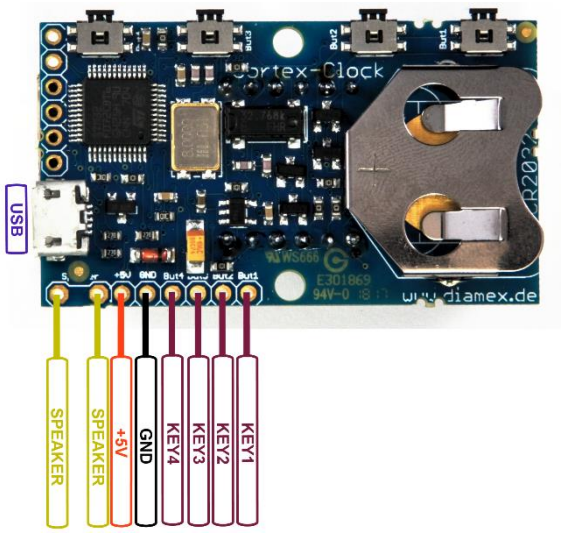
Anschluss eines Infrarot-Empfängers an den Basic-Booster

Zum Lieferumfang der Infrarot-Erweiterung gehören neben der Infrarot-Fernbedienung auch der passende Infrarot-Empfänger TSOP31436, ein Widerstand 33 Ohm und ein Elko 10 µF. Widerstand und Elko dienen zur Störunterdrückung aus der Betriebsspannung und damit zu sicherem Empfang der Infrarot-Signale. Bitte nicht die Polarität des Elkos vertauschen, das kürzere Beinchen (schwarze Markierung auf dem Gehäuse) kommt an GND. Schließen Sie die Bauteile nach der folgenden Zeichnung an den Basic-Booster an.

Tipp: Die Basic-Booster Platine lässt sich mit passenden Stiftleisten auf einem Steckbrett (Breadboard) platzieren. Die Bauteile für den Infrarot-Empfänger lassen sich damit bequem ohne Lötverbindungen verdrahten.



Cortex-Clock



4-stelliges 7-Segment-Uhr Modul. Terminal Print- und Fehler-Ausgabe über USB. Integrierte Echtzeituhr mit Stützbatterie. 4 Taster auch über Lötpunkte als Eingangspins zu verwenden. Anschluss für Miniatur-Lautsprecher zur Signalausgabe ist auf dem Board vorhanden. Stromversorgung über USB-Anschluss oder über die Lötpunkte +5V und GND.

Die Anwendung dieses Moduls ist nicht auf die Uhrzeit-Anzeige beschränkt. Durch Programmierung sind unter anderem auch solche Anwendungen wie Kurzzeitwecker oder Eieruhr, Stoppuhr, Schrittzähler und mehr möglich.

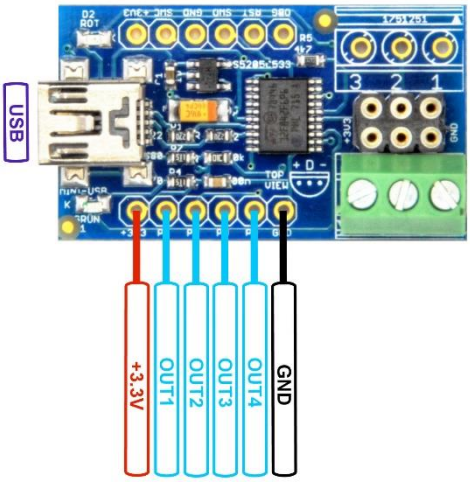
Programmierung über USB-Anschluss.

System-Code: 3170

Konfigurationszeile	LED-Befehle	IO-Befehle
P…	Befehle für 7-SEGMENT-DISPLAY, BRIGHT [0..15]	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC BEEP EEREAD, EEWRITE [0..15]

Temperatur-Sensor Interface

Interface für maximal 8 Temperatur-Sensoren des Typs DS18B20 von Dallas. Über 4 Ausgangsports können Schaltvorgänge ausgelöst werden. Terminal Print- und Fehler-Ausgabe über USB. Die Stromversorgung geschieht über den USB-Anschluss. Steckbrett-geeignet mit passenden Stiftleisten.



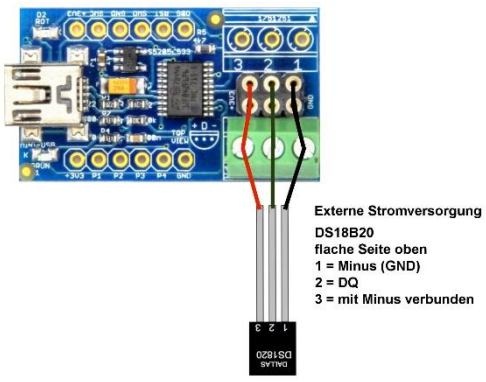
Programmierung über USB-Anschluss.

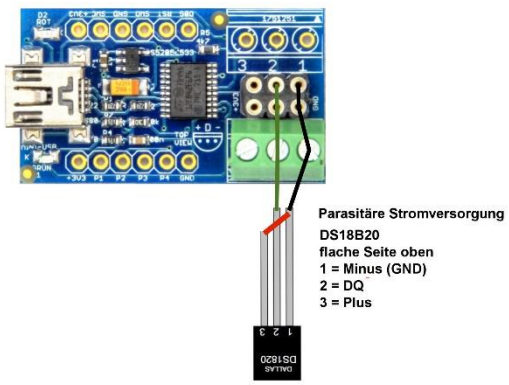
System-Code: 3180

Konfigurationszeile	LED-Befehle	IO-Befehle
P… S…	Keine	XTEMPCNT, XTEMPVAL SETPORT, CLRPORT

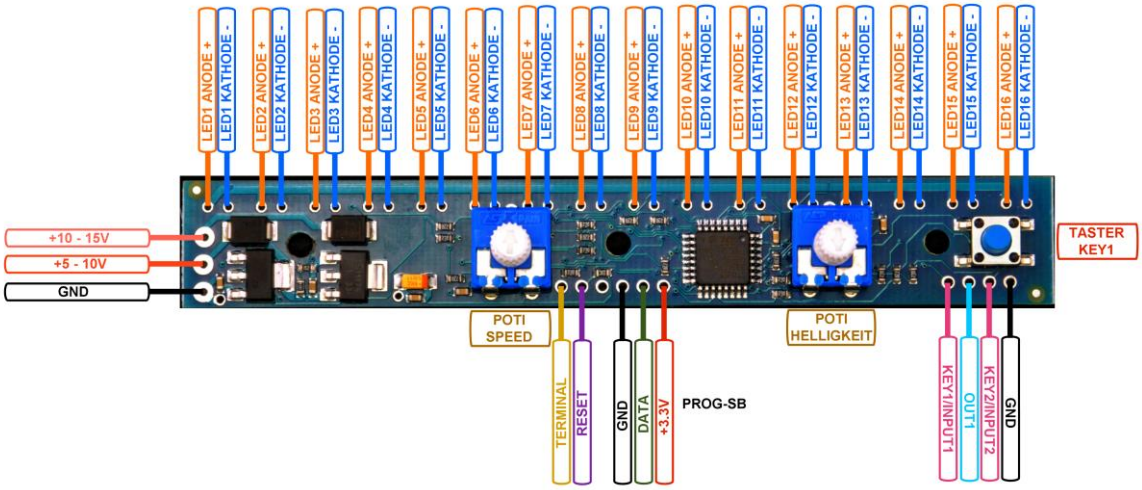
Hinweise zur Benutzung:

Keine Sensoren bei angeschlossener Stromversorgung anschließen oder entfernen.
Die Sensoren nicht verpolen, sie werden hierdurch zerstört.
Bei Anschluss mit parasitärer Stromversorgung über nur 2 Leitungen ist eventuell nicht die maximale Anzahl von Sensoren möglich (ausprobieren).
Alle Sensoren werden nacheinander im Abstand von ca. 1 Sekunde abgefragt, es kann also bis zu 8 Sekunden dauern, bis der gewünschte Temperaturwert aktualisiert wird (bei 8 angeschlossenen Sensoren).
Die Reihenfolge der erkannten Sensoren kann sich beim Austausch von Sensoren verändern.
Alle Sensor-Anschlüsse auf der Platine sind parallel geschaltet, die Reihenfolge der Sensoren ist somit unabhängig davon, an welchen Pins sie angeschlossen sind.
Die grüne LED blinkt beim Lesen eines Sensors kurz auf, dies kann durch S0 in der Konfigurationszeile abgeschaltet werden.
Die rote LED zeigt durch Blinken an, dass ein Laufzeitfehler aufgetreten ist. Die genaue Fehlermeldung wird über das Terminal ausgegeben.





Lauflicht mit 16 LEDs



Lauflicht für 16 einfarbige LEDs. 2 Potentiometer zur Geschwindigkeit- und Helligkeitsregelung (oder andere Funktion programmierbar). 1 Taster und ein zusätzlicher Eingangsport vorhanden. Über einen Ausgangsport können Schaltvorgänge ausgelöst werden. Terminal Print- und Fehler-Ausgabe über Prog-SB. Die Stromversorgung (5-10V / 10-15V) geschieht über Lötanschlüsse.

Programmierung über Prog-SB (+3.3V, DATA, GND, RESET, TERMINAL).

System-Code: 3190

Konfigurationszeile	LED-Befehle	IO-Befehle
P…	SETLED, SETALL BRIGHT	GETKEY, WAITKEY, KEYSTATE SETPORT, CLRPORT GETPOTI EEREAD, EEWRITE [0..15]

LED.setled(pos, val)
val: 0 = LED an Position **pos** aus, 1 = LED an Position **pos** ein
pos: 0..15

LED.setall(val)
val: 0 = alle LEDs aus, 1 = alle LEDs ein

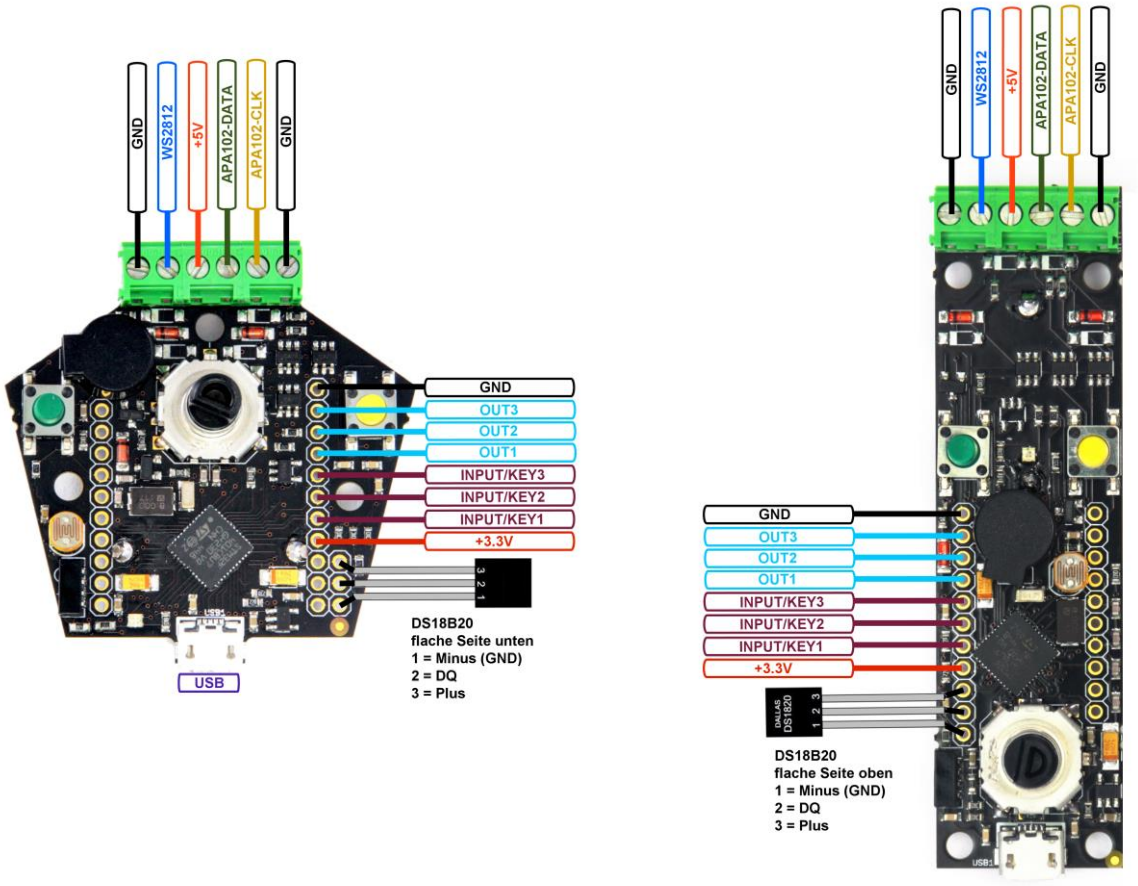
LED.bright(val)
val: 0..255 = alle LED auf Helligkeit **val** setzen, >255 = Helligkeit wird mit Poti geregelt (Standard)

IO.getpoti(idx)
Liest die Werte der Potentiometer (Poti) oder deren umgewandelten Werte aus einer Tabelle aus.

Gültige idx-Werte:
0 = Speed-Poti-Wert (0..255) direkt lesen
1 = Umgewandelter Speed-Poti-Wert aus Tabelle (0..15) lesen
2 = Helligkeits-Poti-Wert (0..255) direkt lesen
3 = Umgewandelter Helligkeits-Poti-Wert aus logarithmischer Tabelle (0..255) lesen
Bei einem ungültigen **idx**-Wert wird Null gelesen.

Das Auslesen der Potentiometer-Werte geschieht über einen Analog-Digital-Wandler im Microcontroller des Lauflichtes. Um einen gleichmäßigen Wert zu ermitteln, werden mehrere Werte hintereinander gemessen und miteinander verglichen. Der erste mit der **getpoti**-Funktion gelesene Wert kann deshalb noch ungültig sein (Wert = 0).

All-In-One Power-M4-Board



Die Power-Schaltung mit vielen Funktionen auf Cortex-M4-Basis. Für maximal 1024 LEDs mit integrierter PWM (WS2812, SK6812 und kompatible sowie APA102 und kompatible). Auch für RGBW-LEDs geeignet. Optional mit Infrarot-Fernbedienung steuerbar. Terminal Print- und Fehler-Ausgabe über USB. 2 Tasten, 3 Eingangspins, 3 programmierbare Ausgangspins. Drehimpulsgeber mit Taste, Lautsprecher und LDR (Fotowiderstand), Echtzeituhr (RTC) mit Batterie, Anschlussmöglichkeit für Temperatursensor DS18B20, Beeper. EEPROM mit 1024 Speicherstellen. Stromversorgung über USB- oder LED-Anschluss möglich.

Programmierung über USB-Anschluss.

System-Code: 3210

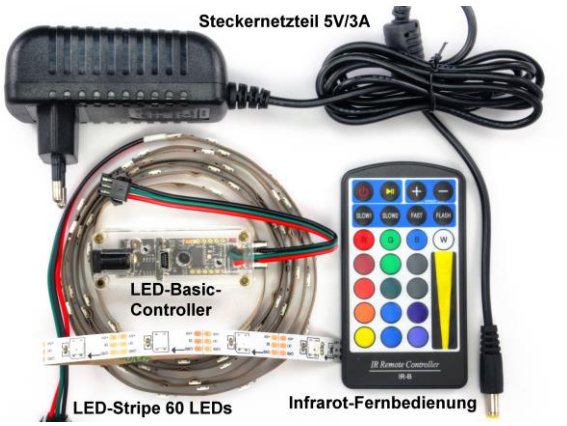
Konfigurationszeile	LED-Befehle	IO-Befehle
L... C... M... P... S... T... A... F... Lxxx: Maximum = 1024 Tx: 0 = WS2812, 1 = APA102	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR GETIR SETENC, GETENC SETPORT, CLRPORT GETTEMP BEEP EEREAD, EEWRITE [0..1023] SYS [COM]

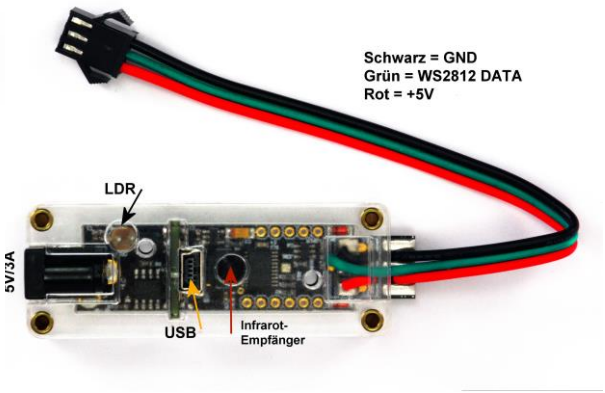
In der Konfigurationszeile kann mit dem Wert **Ax** die Bitrate für die Ansteuerung der APA102-LEDs eingestellt werden. Je höher der Wert, desto niedriger die Bitrate. Bei längeren Verbindungsleitungen zu den LEDs kann es erforderlich sein, eine geringere Bitrate einzustellen, wenn Störungen auftreten. Bei Verwendung von WS2812-LEDs hat dieser Wert keine Bedeutung.

A0 = 42 Mbit	A1 = 21 Mbit	A2 = 10,5 Mbit	A3 = 5,25 Mbit
A4 = 2,6 Mbit (Std.)	A5 = 1,3 MBit	A6 = 656 kBit	A7 = 328 kBit

Bei Werten oberhalb von 10 Mbit kann es zum Flackern der LEDs kommen.

LED-Box





Spezialschaltung zur Steuerung von WS2812-Stripes mit Infrarot-Fernbedienung. Die LED-Box beinhaltet den LED-Basic-Controller, einen LED-Stripe mit 60 LEDs, Infrarot-Fernbedienung und Netzteil. Ideal für Effektbeleuchtungen in der Wohnung. Über den LDR sind Helligkeitsanpassungen des Stripes abhängig von der Umgebungshelligkeit möglich. EEPROM mit 8 Speicherstellen. Terminal Print- und Fehler-Ausgabe über USB. Stromversorgung über 5V/3A Steckernetzteil.

Programmierung über USB-Anschluss.

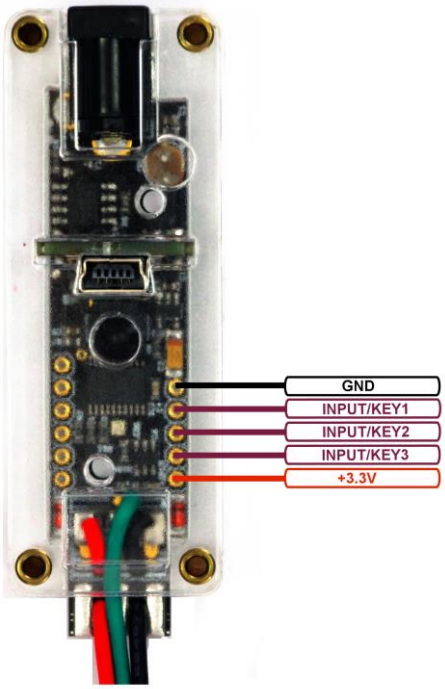
System-Code: 3220

Konfigurationszeile	LED-Befehle	IO-Befehle
L… M… P… S… F… Lxxx: Maximum = 64	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT GETIR GETLDR EEREAD, EEWRITE [0..7]

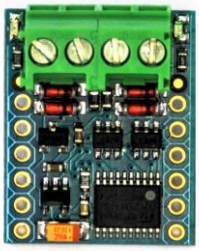
Da die LEDs auch im ausgeschalteten Zustand einen Ruhestrom aufnehmen, kann die Stromversorgung der LEDs über den Befehl **IO.setport(1)** ein- und über den Befehl **IO.clrport(1)** ausgeschaltet werden.

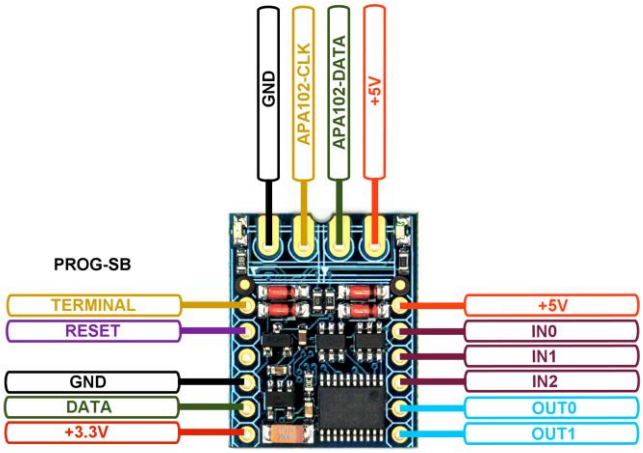
Es kann passieren, dass die LEDs nach dem Einschalten kurz flackern oder sogar ständig leuchten. Es sollte deshalb sofort nach dem Einschalten ein Befehl zum Löschen aller LEDs geschickt werden (z.B. **LED.blackout()**)

Drei Eingangs-pins (Input/Key) können im LED-Basic abgefragt werden um z.B. bestimmte LED-Sequenzen über ein externes Signal auszulösen.



APA-Booster





Spezial-Mini-Interface für LED-Basic. Für maximal 256 RGB-LEDs mit integrierter PWM vom Typ APA102 (DATA- und CLOCK-Leitung). Terminal Print- und Fehler-Ausgabe über Prog-SB. Über 3 Eingangspins (IN0..3) können Funktionen ausgelöst werden, 2 Ausgangspins (OUT0..1) sind für Schaltvorgänge vorhanden. Die Stromversorgung geschieht über den +5V-LED-Anschluss.

Diese Komponente ist nicht für WS2812/SK6812-LEDs geeignet!

Programmierung über Prog-SB (+3.3V, DATA, GND, RESET, TERMINAL).

System-Code: 3230

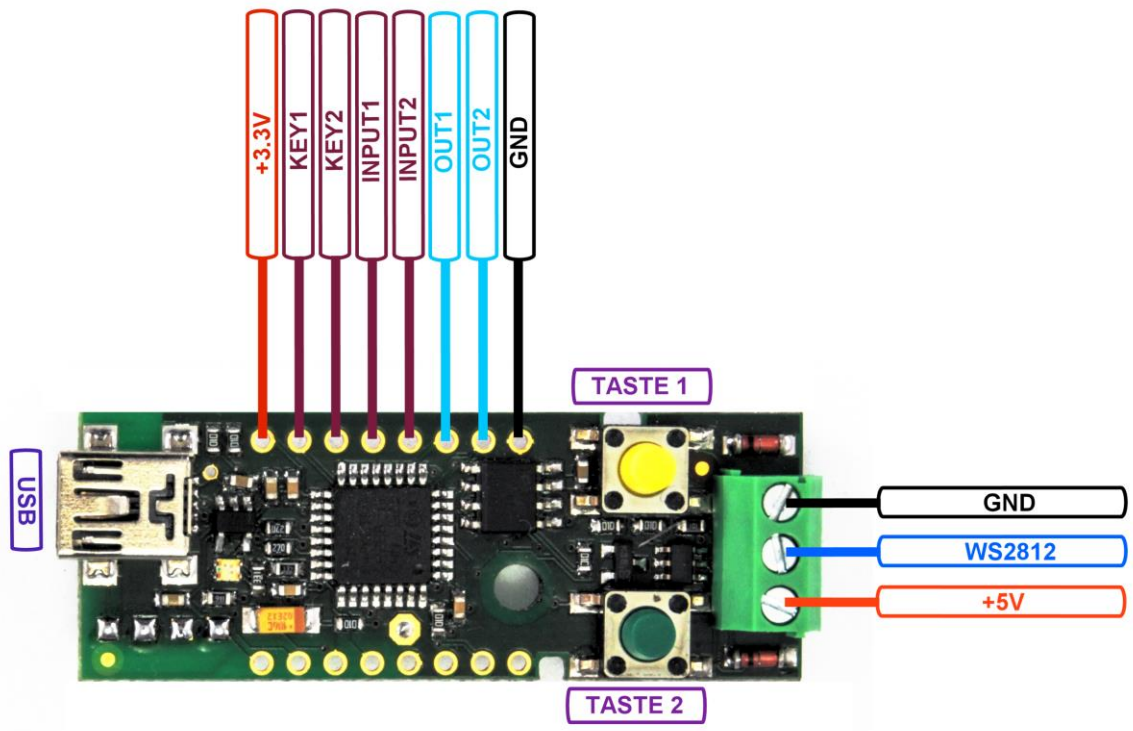
Konfigurationszeile	LED-Befehle	IO-Befehle
L… CRGB P… S… A… F… Lxxx: Maximum = 256	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT

In der Konfigurationszeile kann mit dem Wert **Ax** die Bitrate für die Ansteuerung der LEDs eingestellt werden. Je höher der Wert, desto niedriger die Bitrate. Bei längeren Verbindungsleitungen zu den LEDs kann es erforderlich sein, eine geringere Bitrate einzustellen, wenn Störungen auftreten.

A0 = 24 Mbit	A1 = 12 Mbit	A2 = 6 Mbit	A3 = 3 Mbit
A4 = 1,5 Mbit (Std.)	A5 = 750 kBit	A6 = 375 kBit	A7 = 187,5 kBit

Bei Werten oberhalb von 10 Mbit kann es zum Flackern der LEDs kommen.

RC-Box (Radio controlled box)



Spezialschaltung zur Steuerung von WS2812-LEDs/Stripes mit 1- oder 4-Kanal Funk-Fernbedienung. EEPROM mit 16 Speicherstellen. Terminal Print- und Fehler-Ausgabe über USB. Über 2 Eingangspins können Funktionen ausgelöst werden, 2 Ausgangspins sind für Schaltvorgänge vorhanden. Stromversorgung über USB oder LED-Anschluss.

Programmierung über USB-Anschluss.

System-Code: 3240

Konfigurationszeile	LED-Befehle	IO-Befehle
L… M… P… S… F… Lxxx: Maximum = 128	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT GETADC GETIR EEREAD, EEWRITE [0..15]

Da die LEDs auch im ausgeschalteten Zustand einen Ruhestrom aufnehmen, kann die Stromversorgung der LEDs über den Befehl **IO.setport(8)** ein- und über den Befehl **IO.clrport(8)** ausgeschaltet werden.

Es kann passieren, dass die LEDs nach dem Einschalten kurz flackern oder sogar ständig leuchten. Es sollte deshalb sofort nach dem Einschalten ein Befehl zum Löschen aller LEDs geschickt werden (z.B. **LED.blackout()**)

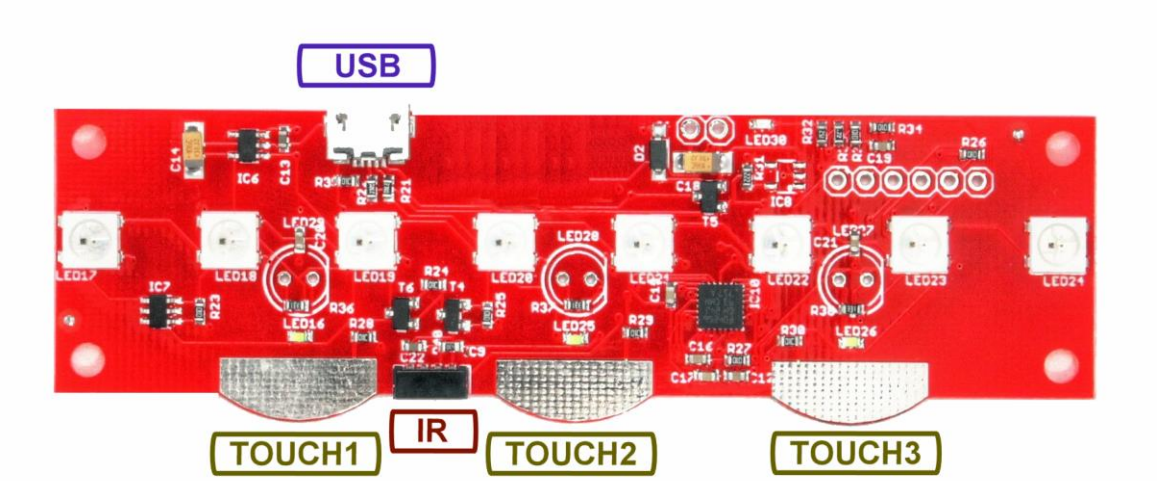
Die RC-Box wurde für den Betrieb mit 5V Powerbanks vorgesehen. Über **IO.getadc(0)** wird die Betriebsspannung abgefragt. Ein Rückgabewert von z.B. 475 bedeutet 4,75 Volt.

Anlernen der Funk-Fernbedienung:
Stromversorgung trennen (USB Stecker ziehen).
Taste 1 drücken und festhalten.
USB-Stecker einstecken, LED blinkt grün.
Eine oder mehrere Tasten auf der Fernbedienung gleichzeitig drücken bis die LED dauerhaft grün leuchtet (ca. 2 Sekunden).
USB-Stecker ziehen und wieder einstecken.

Die 4-Kanal Funk-Fernbedienung kann mit einer oder mehreren Tasten an einen Empfänger oder mit jeder Taste an verschiedene Empfänger angelernt werden.
Es können nicht mehrere Fernbedienungen an einen Empfänger angelernt werden.

Abfrage der Funk-Fernbedienung:
Der Infrarot-Befehl **IO.getir()** dient zur Abfrage der Funk-Fernbedienungssignale.
Bei der 1-Kanal Fernbedienung wird bei gedrückter Taste der Wert 1 zurückgegeben.
Bei der 4-Kanal Fernbedienung wird abhängig von der angelernten Taste folgender Wert zurückgegeben: Taste A = 8, Taste B = 4, Taste C = 2, Taste D = 1
Eine Abfrage, wie lange eine Taste auf der Fernbedienung gedrückt wurde oder eine Tasten-Wiederholfunktion ist aus technischen Gründen nicht möglich.

Touch-Lamp



Effektlampe mit Sensor-Tasten und Fernbedienung. 8 Integrierte WS2812-LEDs sind in Farbe und Helligkeit veränderbar und beleuchten ein Plexiglas-Display. EEPROM mit 8 Speicherstellen. Terminal Print- und Fehler-Ausgabe über USB. Stromversorgung über USB. Steuerung auch über Infrarot-Fernbedienung möglich.

Programmierung über USB-Anschluss.

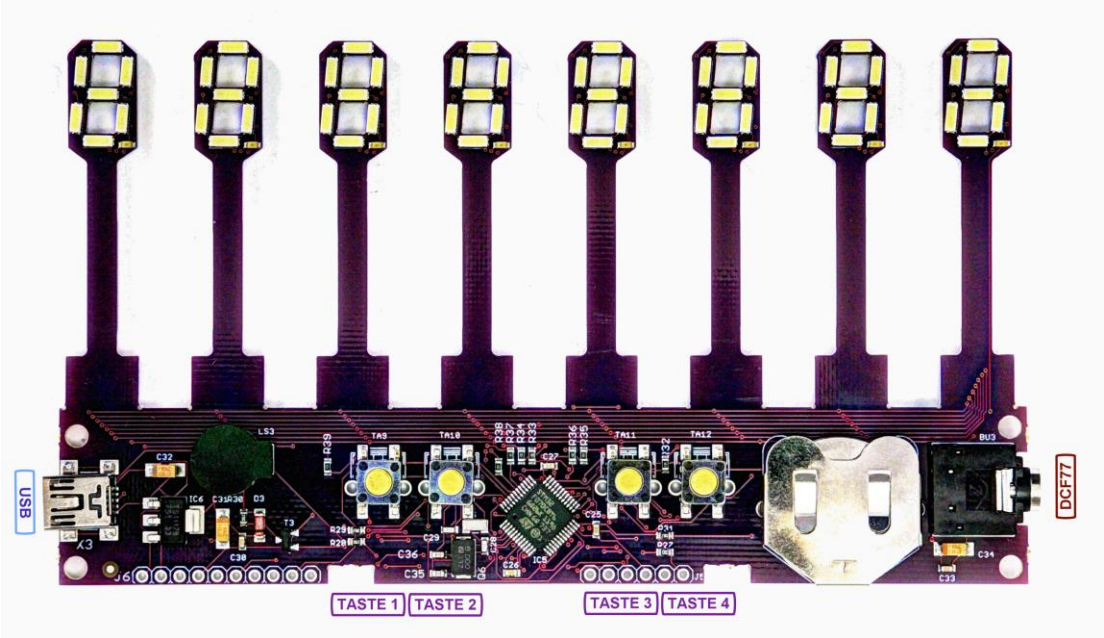
System-Code: 3270

Konfigurationszeile	LED-Befehle	IO-Befehle
M… P…	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT GETIR EEREAD, EEWRITE [0..7]

Da die LEDs auch im ausgeschalteten Zustand einen Ruhestrom aufnehmen, kann die Stromversorgung der LEDs über den Befehl **IO.setport(1)** ein- und über den Befehl **IO.clrport(1)** ausgeschaltet werden.

Die Bedienung der Touch-Lamp ist abhängig von der verwendeten Basic-Software. Sie finden eine Kurzanleitung für die Touch-Sensoren und die Fernbedienung im Basic-Sourcecode.

LED-Tube-Clock



8-stellige Uhr mit 7-Segment LED-Anzeigen im Tube-Design. In verschiedenen LED-Farben erhältlich. Terminal Print- und Fehler-Ausgabe über USB. Integrierte Echtzeituhr mit Stützbatterie. DCF-77 Anschluss mit Klinkenbuchse. 4 Taster, Beeper, Stromversorgung über USB-Anschluss, Stromaufnahme ca. 30mA.

Programmierung über USB-Anschluss.

System-Code: 3300

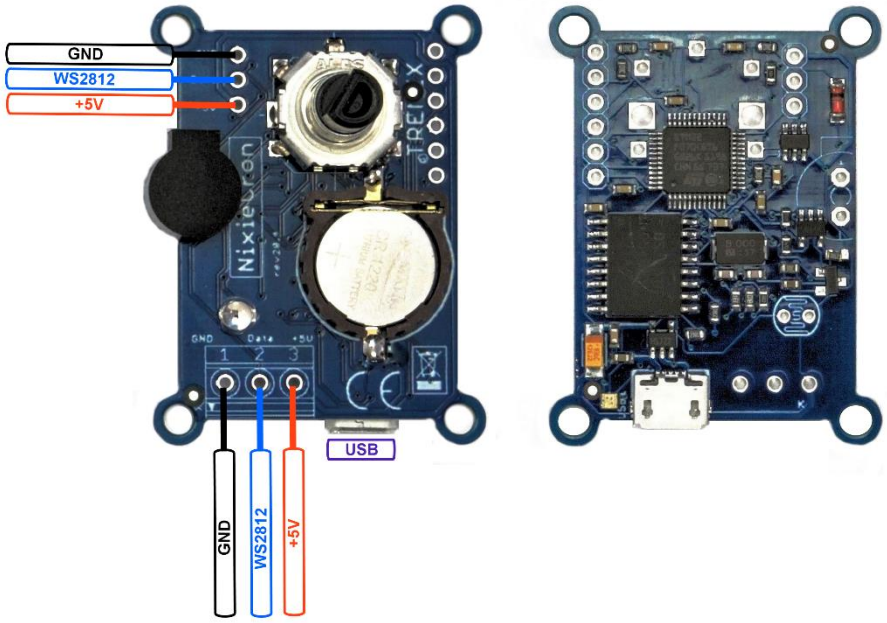
Konfigurationszeile	LED-Befehle	IO-Befehle
P…	Befehle für 7-SEGMENT-DISPLAY BRIGHT [0..15]	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC BEEP EEREAD, EEWRITE [0..15] SYS [COM, CALIB]

LED-Tube im Plexiglas-Gehäuse.

Hinweis zur Montage:
Um die Röhren von innen nicht zu verkratzen, diese bitte beim Aufstecken nicht drehen!

NixieCron Komponenten

NixieCron - Cronios 2



Die NixieCron-Cronios 2 Komponente mit LED-Basic-Software. Für maximal 512 LEDs mit integrierter PWM (WS2812, SK6812 und kompatible). Auch für RGBW-LEDs geeignet. Terminal Print- und Fehler-Ausgabe über USB. Integrierte hochgenaue Echtzeituhr (DS3231) mit Stützbatterie, Drehimpulsgeber mit Tastenfunktion, LDR und Lautsprecher sind auf dem Board vorhanden. EEPROM mit 1024 Speicherstellen. Stromversorgung über USB- oder LED-Anschluss möglich.

Neue Version mit doppelter LED-Anzahl gegenüber Cronios 1 und hochgenauem integrierten Uhrenchip. Durch Temperaturkompensation sind Ganggenauigkeiten von nur wenigen Sekunden Abweichung pro Monat möglich.

Programmierung über USB-Anschluss.

System-Code: 3350

Konfigurationszeile	LED-Befehle	IO-Befehle
L… C… M… P… S… F… Lxxx: Maximum = 512	Alle Befehle Für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR (Photodiode) BEEP GETENC, SETENC EEREAD, EEWRITE [0..1023] SYS [COM, CALIB, TEMP]

Hinweis:
Eine erweiterte Cronios 2 – Version mit DCF77-Anschluss ist verfügbar.

NixieCron - Cronios 3 (Sound)

Die NixieCron-Cronios 3 Komponente mit LED-Basic-Software. Für maximal 512 LEDs mit integrierter PWM (WS2812, SK6812 und kompatible). Auch für RGBW-LEDs geeignet. Terminal Print- und Fehler-Ausgabe über USB. Integrierte hochgenaue Echtzeituhr (DS3231) mit Stützbatterie, Drehimpulsgeber mit Tastenfunktion, zusätzliche Taste (z.B. zur Ansage der Uhrzeit) und LDR/Photodiode sind auf dem Board vorhanden. EEPROM mit 1024 Speicherstellen.

Cronios 3 mit Sound-Ausgabe: Aus dem 8 Megabyte großen Flash-Speicher werden Sound-Daten im MP3-Format abgespielt und über den eingebauten Lautsprecher ausgegeben. Hier können Zeit- und Temperaturansagen sowie Stundensignale abgelegt und über LED-Basic ausgewählt und ausgegeben werden.

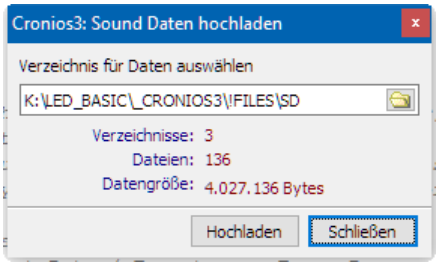
Stromversorgung über USB- oder LED-Anschluss möglich.

Programmierung über USB-Anschluss.

System-Code: 3400

Konfigurationszeile	LED-Befehle	IO-Befehle
L… C… M… P… S… F… Lxxx: Maximum = 512	Alle Befehle Für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR (Photodiode) BEEP GETENC, SETENC EEREAD, EEWRITE [0..1023] SYS [COM, CALIB, DFP]

Die Befehle für die Ausgabe der Sound-Daten sind identisch mit den **IO.sys** – Befehlen für den DF-Player-Mini. Ein Reset und die Busy-Abfrage bei Befehlen sind jedoch beim Cronios3 nicht erforderlich. Der Equalizer ist nicht vorhanden, Befehle hierfür werden ignoriert. Die Lautstärken-Einstellung wirkt sich auch auf **IO.beep** aus.



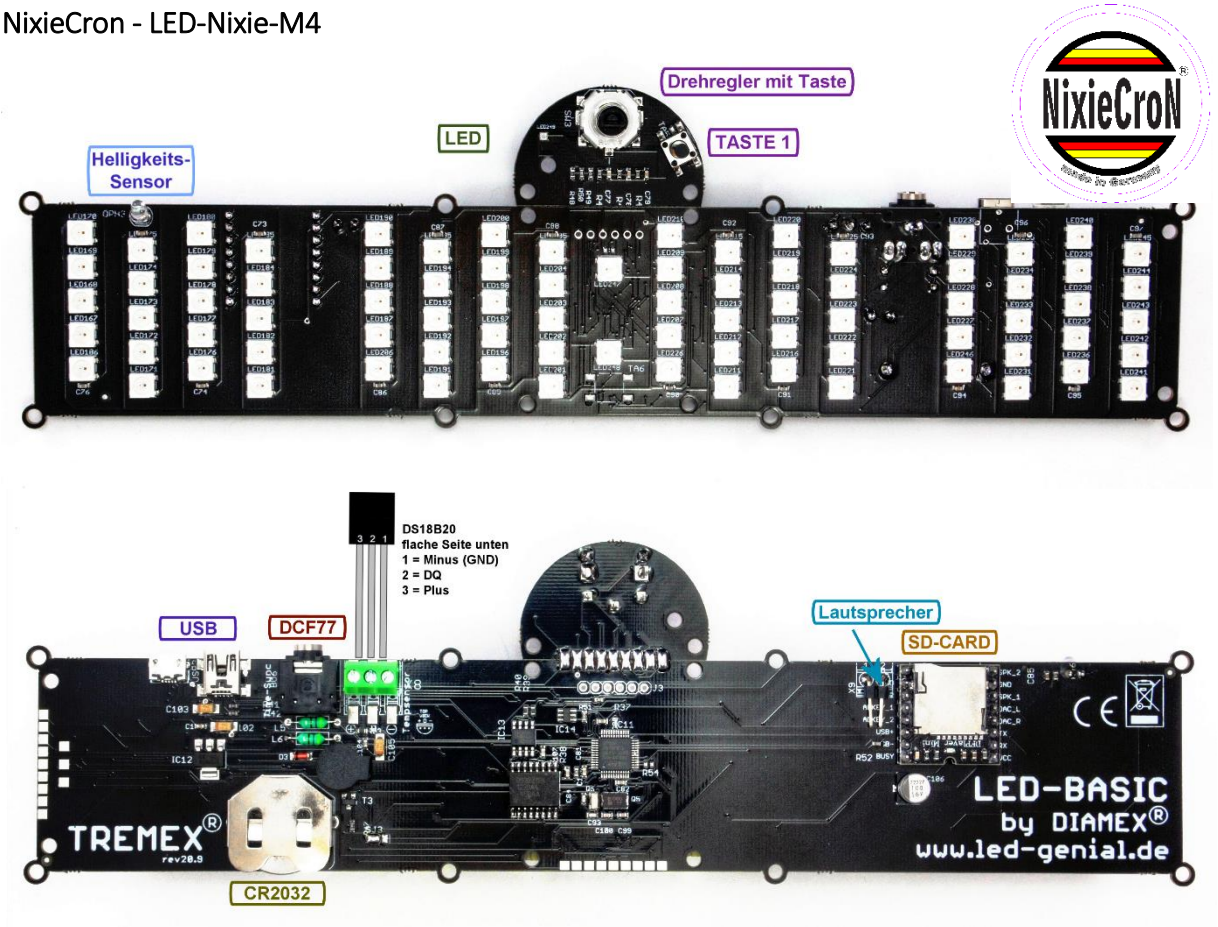
Die Sound-Daten werden über die LED-Basic Entwicklungsumgebung hochgeladen, je nach Datenmenge kann dies mehrere Minuten dauern. Hierzu müssen sich alle MP3-Dateien in einem beliebigen Verzeichnis und den darin befindlichen Unterverzeichnissen befinden. Die Namen der Unterverzeichnisse dürfen nur aus Nummern bestehen: 01, 12 usw. In jedem Unterverzeichnis dürfen sich maximal 255 Dateien befinden: 001.mp3 … 255.mp3 usw. Über den Befehl

IO.sys(1101, (Dir * 256) + Datei)

wird **Datei** aus **Dir** zur Warteschlange hinzugefügt. Wenn eine Datei nicht gefunden wird, wird diese ohne Fehlermeldung ignoriert.

Um Speicherplatz zu sparen, sollten alle MP3-Dateien mit maximal 128 kBit und in Mono gespeichert werden. ID3-Tags müssen nicht vorhanden sein. Beispiele der Sound-Daten für die LED-Nixie-Software (kompatibel für die SD-Karte der LED-Nixie-M4) befinden sich im LED-Basic-Programmpaket.

NixieCron - LED-Nixie-M4



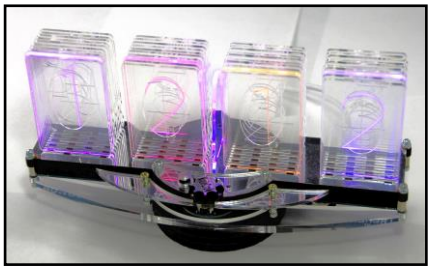
LED-Nixie Größe M mit 4 Stellen.. Terminal Print- und Fehler-Ausgabe über USB. Integrierte hochpräzise Echtzeituhr (DS3231) mit Stützbatterie. DCF-77 Anschluss mit Klinkenbuchse zur Zeitsynchronisation. Drehimpulsgeber mit Taster, 2. Taster, Beeper, Anschluss für DS18B20 Temperatursensor, Helligkeitssensor, DF-Player zur Ausgabe von Sounddaten von SD-Karte mit Anschluss für Lautsprecher. Stromversorgung über USB-Anschluss.

Programmierung über USB-Anschluss.

System-Code: 3320

Konfigurationszeile	LED-Befehle	IO-Befehle
P… M… S… L… F… Lxxx: Maximum = 128	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC BEEP GETLDR (Photodiode) GETTEMP GETENC, SETENC EEREAD, EEWRITE [0..1023] SYS [COM, CALIB, DFP]

Aufbauvideo: https://www.nixiecron.de/video/nixie4



NixieCron - LED-Tube-Clock

8-stellige Uhr mit 7-Segment LED-Anzeigen im VFD-Tube-Design. In verschiedenen LED-Farben erhältlich. Terminal Print- und Fehler-Ausgabe über USB. Integrierte hochpräzise Echtzeituhr DS3231 mit Stützbatterie. DCF-77 Anschluss mit Klinkenbuchse. 4 Taster, Beeper, Stromversorgung über USB-Anschluss, Stromaufnahme ca. 30mA.

Programmierung über USB-Anschluss.

System-Code: 3300

Konfigurationszeile	LED-Befehle	IO-Befehle
P…	Befehle für 7-SEGMENT-DISPLAY BRIGHT [0..15]	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC BEEP EEREAD, EEWRITE [0..15] SYS [COM]

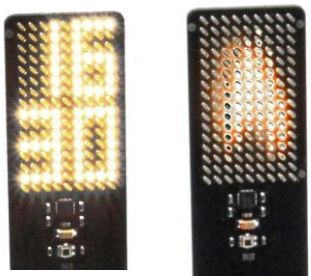
Hinweis: In der LED-Basic Entwicklungsumgebung bitte LED-Tube-Clock auswählen.

NixieCron – Flame-Clock

LED-Matrix-Anzeige (9 x 16 Pixel) als Kerzen-Flammen-Simulation oder zur Uhranzeige Terminal Print- und Fehler-Ausgabe über USB. Integrierte hochpräzise Echtzeituhr DS3231 mit Stützbatterie. 3 Taster (2 x vorhanden), Beeper, Helligkeitssensor mit Photodiode.

Stromversorgung und Programmierung über USB-Anschluss.

System-Code: 3390



Konfigurationszeile	LED-Befehle	IO-Befehle
P…	Befehle für LEDs ohne PWM und MATRIX-Befehle + LED.update()	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC BEEP GETLDR (Photodiode) EEREAD, EEWRITE [0..1023] SYS [COM]

Hinweise:

Nach Änderungen der Anzeige muss der Befehl **LED.update()** gesendet werden.

Bei Matrixbefehlen gibt der **col**-Wert die Helligkeit der LEDs an (Werte: 0...255).

Bei **MATRIX.shift(dir, count)** ist der maximale count-Wert für left/right = 9, für up/down = 16

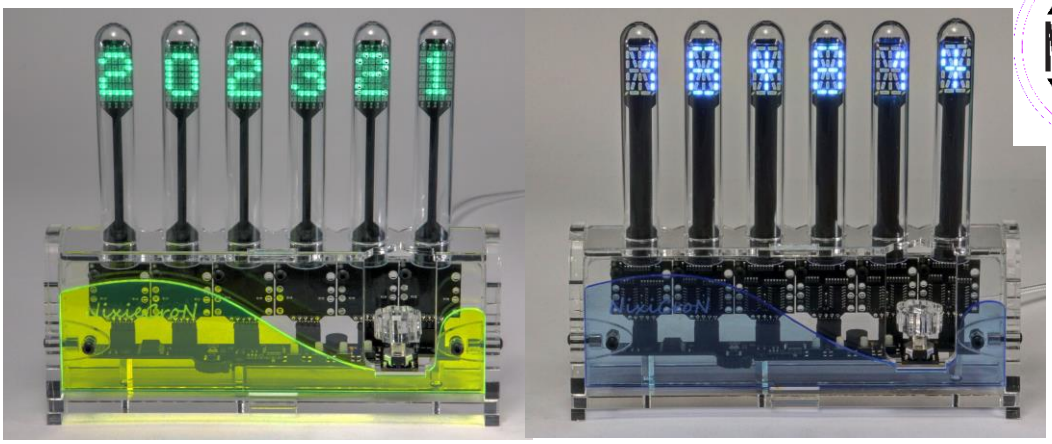
Mit **MATRIX.pic(frame, opt)** wird ein Flammen-Frame aus dem internen Speicher angezeigt. Mit opt wird die Helligkeit eingestellt. (Werte: frame = 0…268, opt = 0…4).

Font 0 = 3 x 5 Pixel, Zeichen: 0..9, °, C

Font 1 = 4 x 7 Pixel, Zeichen: 0..9, °, C

Font 2 = 5 x 8 Pixel, Zeichen: 0...9, A...Z, + - . :

NixieCron – Matrix- und Segment-Tube-Clock



1. 6-Stellige Matrix-Tube-Clock mit 5 x 8 Matrix-Anzeigen, WS2812-RGB LEDs (multi-color).
2. 6-Stellige Segment-Tube-Clock mit 24-Segment-Anzeigen, monochrom (hellblau).

Terminal Print- und Fehler-Ausgabe über USB. Integrierte hochpräzise Echtzeituhr DS3231 mit Stützbatterie. Bedienung über Drehimpulsgeber mit Taste, Beeper, Helligkeitssensor mit Photodiode, Stromversorgung und Programmierung über USB-Anschluss.

System-Code: 3410

Konfigurationszeile	LED-Befehle	IO-Befehle
L… C… M… P… S… F…	Alle Befehle für RGB-LEDs mit PWM MATRIX-Befehle bei Matrix-Tube-Clock (ohne MATRIX.PIC)	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC BEEP GETLDR (Photodiode) GETENC, SETENC EEREAD, EEWRITE [0..1023] SYS [COM, CALIB, TEMP]

Hinweise zur Segment-Tube-Clock:

Aufgrund der verwendeten WS2812-kompatiblen LED-Treiber ist die Ansteuerung der einzelnen Segmente etwas umständlich. Rot, Grün und Blau steuern jeweils ein Segment der Anzeige an. Schauen Sie bitte in den Demodateien nach, wie die Umwandlung per Basic-Befehl funktioniert und wie Zeichen definiert werden.

Hinweise zur Matrix-Tube-Clock:

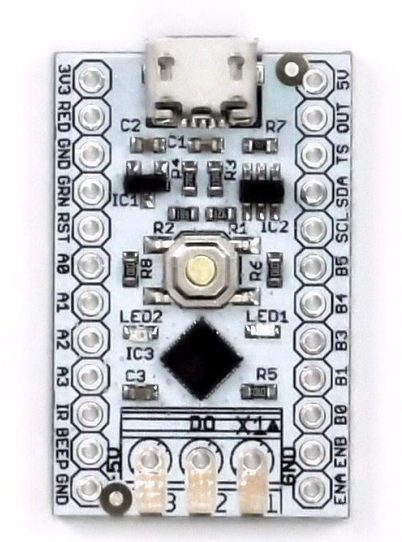
Diese Komponente hat aufgrund der benutzten WS2812-LEDs eine sehr hohe Stromaufnahme, wenn die LEDs sehr hell eingestellt sind. Dies kann den USB-Port eines PC überlasten. Bitte arbeiten Sie während der Programmierung nur mit geringer Helligkeit und benutzen ein entsprechend starkes Netzteil im laufenden Betrieb.
Der Befehl **MATRIX.size(x, y, cnt, res1, res2)** muss bei der Matrix-Tube-Clock nicht verwendet werden, hier sind die Standardwerte x = 5, y = 8, cnt = 6 bereits voreingestellt.

LED-BASIC-PICO Komponenten

LED-BASIC-PICO basiert auf einem Mini-LED-BASIC-Controller mit leistungsstarkem STM32-Controller mit 48MHz Taktfrequenz. Trotz seiner geringen Größe werden viele Peripheriebausteine unterstützt und bis zu 64 WS2812-LEDs können direkt angesteuert werden. Viele Sensor-, Ein- und Ausgabemodule sind erhältlich. Alle sind direkt Steckbrett-Kompatibel und können so einfach miteinander verdrahtet werden. Über die LED-BASIC-Benutzeroberfläche kann LED-BASIC-PICO leicht programmiert und so viele verschiedene Anwendungen einfach realisiert werden. Beispiele zur Ansteuerung und Auswertung der Module sind im SOURCE-Ordner von LED-Basic zu finden.

PICO: Basismodul Version 1

Universal-Mini-Basic-Modul mit vielen Funktionen. Maximal 64 WS2812-LEDs (RGB und RGBW) sind mit diesem Modul direkt anzusteuern. Taste zur Eingabe auf dem Board. Direkter Anschluss von Temperatur-Sensor DS18B20, Infrarot-Empfänger, Drehimpulsgeber, Beeper. Über IO-Pins zusätzliche Ein- und Ausgabegeräte anschließbar.



Stromversorgung über USB-Buchse oder LED-Anschluss.

Programmierung über USB-Anschluss.

System-Code: 3370

Konfigurationszeile	LED-Befehle	IO-Befehle
L… C… M… P… S… F… Lxxx: Maximum = 64	Alle Befehle für RGB-LEDs mit PWM Alle Befehl für 7-Segment-LEDs (inkl. UPDATE)[1]	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT GETADC[1] GETIR[1] SETENC, GETENC[1] GETTEMP[1] BEEP[1] EEREAD, EEWRITE [0..2047][1] GETRTC, SETRTC[1] SYS (siehe: Sys-Befehle)

[1]Funktion benötigt Erweiterungsmodul

PICO2: Basismodul Version 2

BILDER FOLGEN

Universal-Mini-Basic-Modul mit vielen Funktionen. Maximal 512 WS2812-LEDs (RGB und RGBW) sind mit diesem Modul direkt anzusteuern. Taste zur Eingabe auf dem Board. Direkter Anschluss von Temperatur-Sensor DS18B20, Infrarot-Empfänger, Drehimpulsgeber, Beeper. Über IO-Pins zusätzliche Ein- und Ausgabegeräte anschließbar. Interne EEPROM-Funktion auch ohne Erweiterungsmodul.

Stromversorgung über USB-Buchse oder LED-Anschluss.

Programmierung über USB-Anschluss.

System-Code: 3430

Konfigurationszeile	LED-Befehle	IO-Befehle
L… C… M… P… S… F… Lxxx: Maximum = 512	Alle Befehle für RGB-LEDs mit PWM Alle Befehl für 7-Segment-LEDs (inkl. UPDATE)[1]	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT GETADC[1] GETIR[1] SETENC, GETENC[1] GETTEMP[1] BEEP[1] EEREAD, EEWRITE [0..2047][1] EEREAD, EEWRITE [0..31][2] GETRTC, SETRTC[1] SYS (siehe: Sys-Befehle)

[1]Funktion benötigt Erweiterungsmodul

[2]Das interne EEPROM wird über die Adressen 10000 … 10031 angesprochen.

Die Version 2 des PICO-Moduls ist Pin- und funktionskompatibel zur Version 1. Aufgrund des mit einem größeren Speicher ausgestatteten Microcontrollers können bis zu 512 LEDs angesteuert werden und es ist zusätzlich ein interner EEPROM-Speicher vorhanden.

PINBELEGUNG DER PICO BASISPLATINE (VERSION 1 + 2)

3V3	3,3V Versorgungsspannung Ausgang für externe Module
RED	Rote LED
GND	Masse
GRN	Grüne LED
RST	RESET, Schaltung wird zurückgesetzt wenn Pin auf Masse gelegt wird
A0...A3	Universelle Eingänge
IR	Eingang für Infrarot-Empfänger-Modul oder universeller Eingang
BEEP	Ausgang für Lausprecher-Modul (BEEPER)
GND	Masse
5V	5V Stromversorgung für PICO-Basic Platine
OUT	WS2812 Signalausgang
TS	Eingang für Temperatursensor-Modul mit DS18B20
SDA	I2C-Bus Datenleitung
SCL	I2C-Bus Taktleitung
B0...B5	Universelle Ausgänge oder Spezialpins für Module (B2 existiert nicht)
ENA,ENB	Eingänge für Drehimpulsgeber oder universelle Eingänge

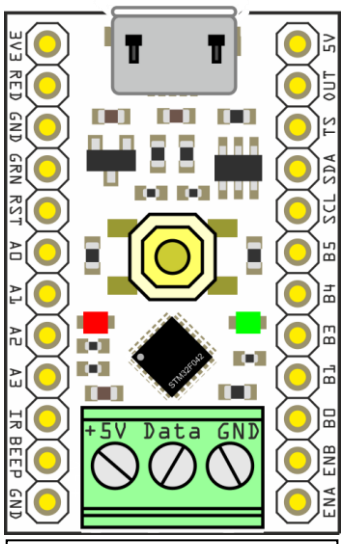


Abbildung: PICO1
PICO2 ist Pin kompatibel

An die optional aufzulötende Schraubklemme (+5V, Data, GND) kann direkt ein WS2812-LED-Strip angeschlossen werden. Die Stromversorgung für das PICO-Basic-Modul ist über den 5V-Anschluss der Schraubklemme dem 5V-Pin oder die USB-Buchse möglich.

Die Eingabeports werden mit IO.getkey, IO.waitkey und IO.keystate abgefragt. Alle Eingänge haben interne Pullup-Widerstände und müssen gegen Masse geschaltet werden. Wenn der Infrarot-Sensor oder der Drehimpulsgeber nicht benötigt werden, können auch diese Pins zusätzlich als Eingabeports benutzt werden. A0=1, A1=2, A2=4, A3=8, Taste auf dem Modul=16, IR=32, ENA=64, ENB=128

PICO VERSION 1 + 2: SYS-BEFEHLE

Über den SYS-Befehl können die IO-Pins in ihrer Funktion gewechselt und verschiedene Werte der Erweiterungsplatinen gesetzt oder abgerufen werden.

IO.sys(500, 0)
Setzt A0, A1, A2, A3 auf Eingabeports zurück (Standardeinstellung nach Start)

IO.sys(500, port)
Setzt A0, A1, A2, A3 auf Analogport (Bitkodiert: A0 = 0x01, A1 = 0x02, A2 = 0x04, A3 = 0x08)
Abfrage des Analogwertes mit **IO.getadc(port)**. Port = 0..3

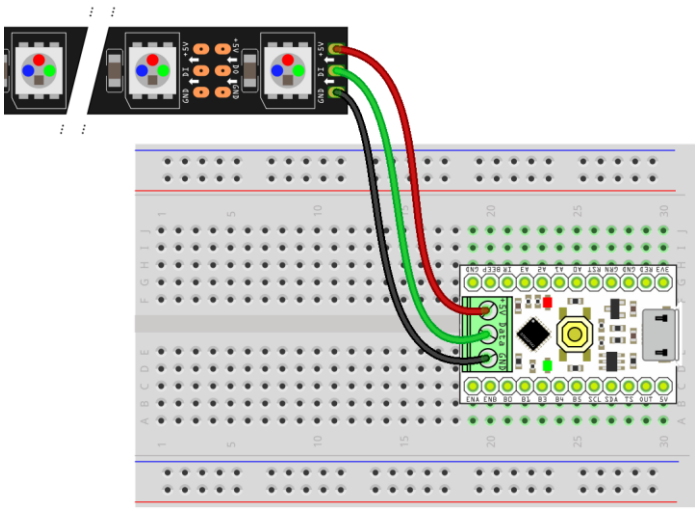
IO.sys(501, 0)
Setzt B0, B1, B3, B4, B5 auf Ausgabeports zurück (Standardeinstellung nach Start)

IO.sys(501, 1)
B3 = CLK, B4 = STB, B5 = DIO für 4/6/8-stelliges Display-Modul mit 4/6/8 Taster
B0 und B1 bleiben unverändert.

<var> = IO.sys(502, 0)
Liest die Tastenwerte des angeschlossenen 4/6/8-stelligen Display-Moduls aus
Rückgabe ist ein 8-Bit Wert, Bit0 = Taste1, Bit1 = Taste2, usw.
Die Bits sind so lange gesetzt, wie die Tasten gedrückt sind. Auch mehrere Tasten gleichzeitig sind möglich.

PICO: Anschluss von WS2812 LEDs

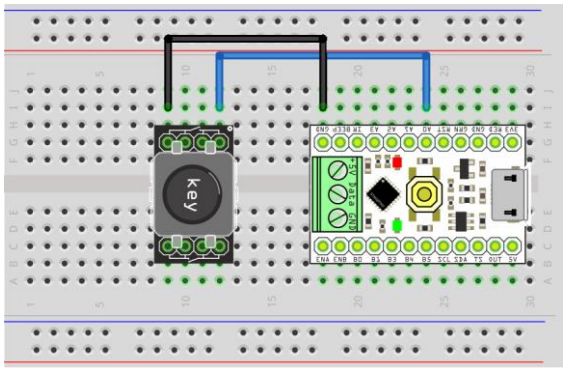
WS2812 LEDs und Stripes können direkt an die 3-poligen Kontakte +5V, Data, GND angelötet oder über eine aufzulötende 3-polige Schraubklemme angeschlossen werden. Das WS2812-Datensignal steht zusätzlich am OUT-Pin zur Verfügung.



Es folgt eine Auflistung aller zum PICO kompatiblen Ein-, Ausgabe- und Sensormodule. In den Tabellen finden sich die erforderlichen Verbindungsleitungen zwischen Modul (links) und UNI-Basic-Platine.

PICO: Eingabetaste

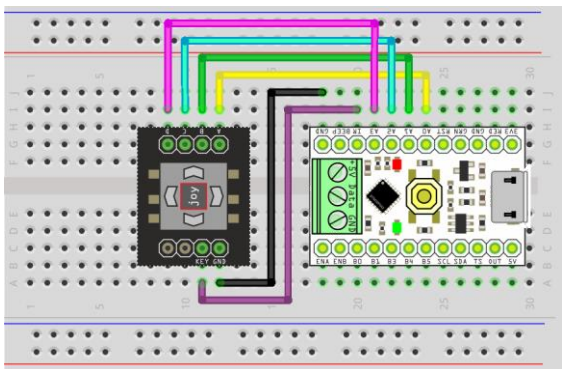
Kontakt 1	GND
Kontakt 2	A0, A1, A2, A3, (IR, ENA, ENB)



Universelle Eingabetaste. Kann an jeden freien Eingabepin angeschlossen werden. Abfrage mit **IO.getkey**, **IO.waitkey** und **IO.keystate**. Beispiel: Taster an A0.

PICO: Joystick

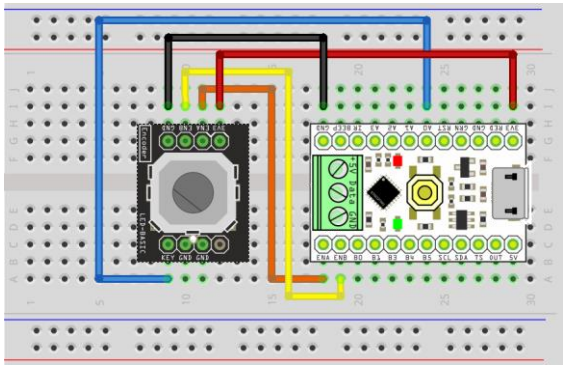
GND	GND
KEY	A0, A1, A2, A3, (IR, ENA, ENB)
A,B,C,D	A0, A1, A2, A3, (IR, ENA, ENB)



4-Wege-Mini-Joystick mit zentraler Taste. Die Kontakte können an jede freien Eingabepins angeschlossen werden. Abfrage mit **IO.getkey**, **IO.waitkey** und **IO.keystate**. Beispiel: A...D an A0...A3, KEY an IR.

PICO: Drehimpulsgeber

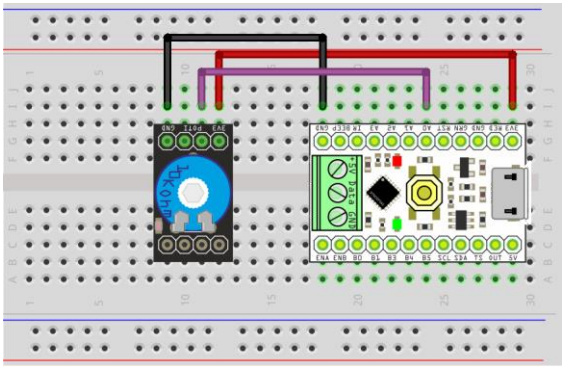
GND	GND
3V3	3V3
KEY	A0, A1, A2, A3, (IR)
ENA	ENA
ENB	ENB



Drehimpulsgeber mit zentraler Taste. KEY kann an jeden beliebigen freien Eingangspin angeschlossen werden. Abfrage der Taste **mit IO.getkey**, **IO.waitkey und IO.keystate**.. Setzen und Abfrage des Impulsgebers mit **IO.setenc** und **IO.getenc**. Beispiel: KEY an A0.

PICO: Potentiometer

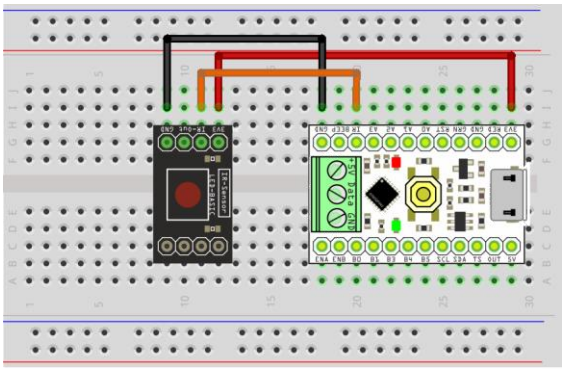
GND	GND
3V3	3V3
POTI	A0, A1, A2, A3



Analog-Potentiometer zur Abfrage mit **IO.getadc(port)**. Initialisierung des Ports: Siehe **IO.sys(500, port)**. Hinweis: Die Bauform des Potentiometers kann variieren. Beispiel: Poti an A0.

PICO: Infrarot-Empfänger

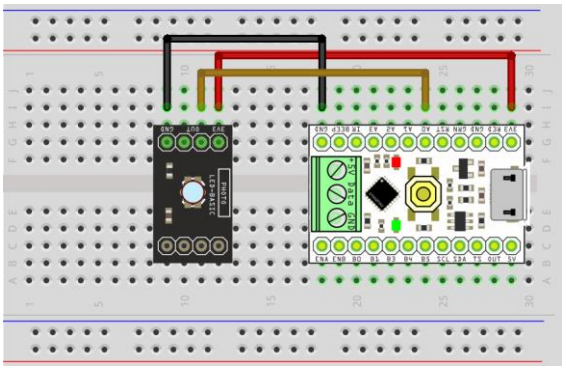
GND	GND
3V3	3V3
IR-OUT	IR



Infrarot-Empfänger für die LED-Basic-Universal-Fernbedienung. Abfrage mit **IO.getir**. PICO erkennt automatisch das Vorhandensein des Infrarot-Empfängers und konfiguriert den Port entsprechend.

PICO: Fotodiode

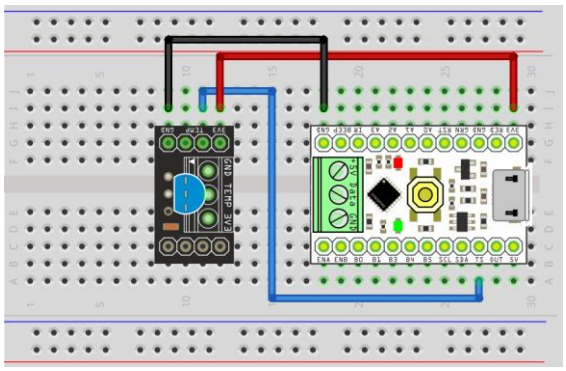
GND	GND
3V3	3V3
OUT	A0, A1, A2, A3



Helligkeitssensor mit Fotodiode. Abfrage mit **Io.getadc(port)**. Initialisierung des Ports: Siehe **IO.sys(500, port)**. Beispiel: Fotodiode an A0.

PICO: Temperatursensor

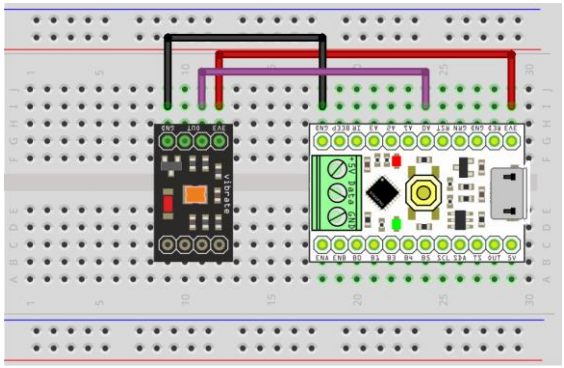
GND	GND
3V3	3V3
TEMP	TS



Temperatursensor mit DS18B20. Abfrage mit **IO.gettemp**.

PICO: Vibrationssensor

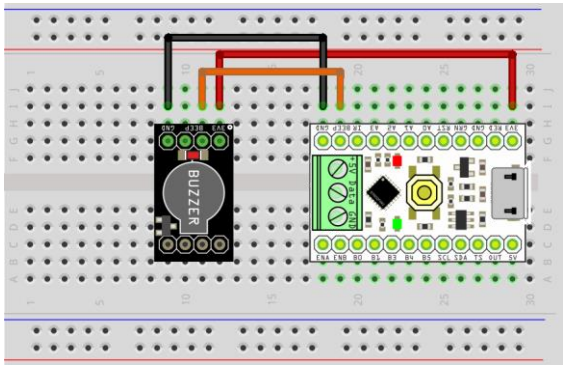
GND	GND
3V3	3V3
OUT	A0, A1, A2, A3, (IR, ENA, ENB)



OUT kann an jeden beliebigen freien Eingangspin angeschlossen werden. Abfrage mit **IO.getkey**, **IO.waitkey** und **IO.keystate**.. Durch Vibration wird das OUT-Signal auf Massepotential gezogen. In Ruhestellung ist das Ausgangssignal auf High-Signal (3,3V). Die Signalumschaltung erfolgt leicht Zeitverzögert um einen ständigen Wechseln des Signalpegels zu verhindern. Beispiel: Vibrationssensor an A0.

PICO: Beeper

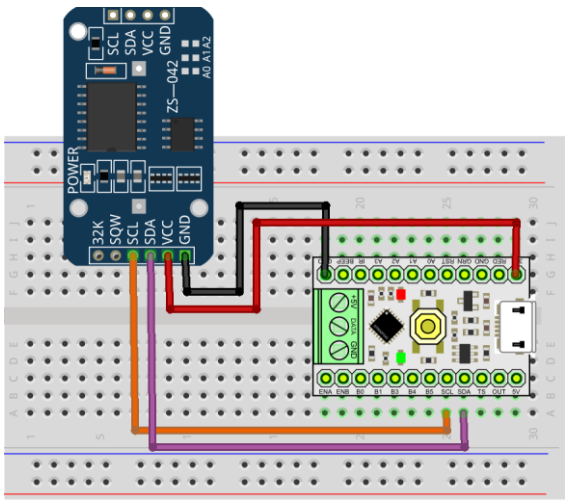
GND	GND
3V3	3V3
BEEPER	BEEP



Ausgabe von Signaltönen über **IO.beep**.

PICO: RTC-Modul mit EEPROM

GND	GND
3V3	3V3
SCL	SCL
SDA	SDA



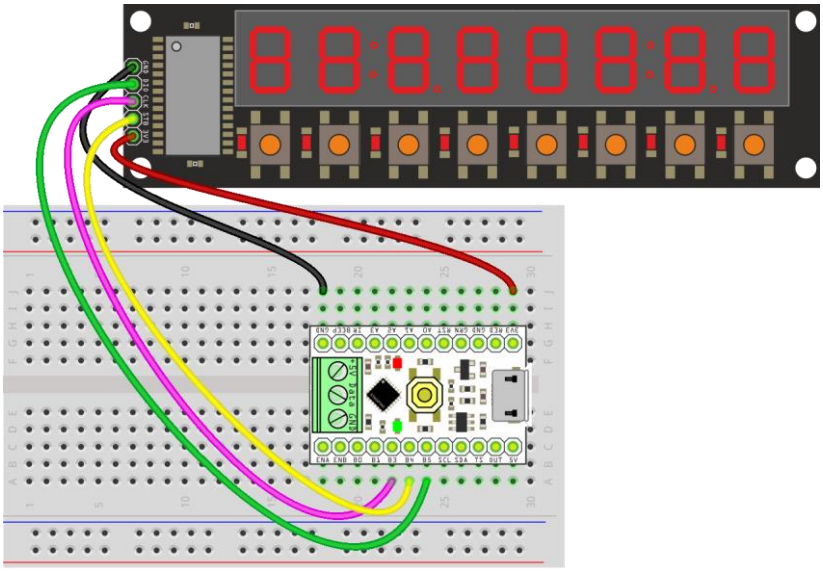
Hochgenaues Uhrmodul mit DS3231 und EEPROM 24C32. Uhrzeit-Abfrage und Setzen mit **IO.getrtc** und **IO.setrtc**. Lesen und Schreiben des EEPROM mit **IO.eeread** und **IO.eewrite**. Zeit und Datum können auch über die LED-Basic-Benutzeroberfläche mit dem PC synchronisiert werden.

PICO: 7-Segment-Anzeige mit Taster

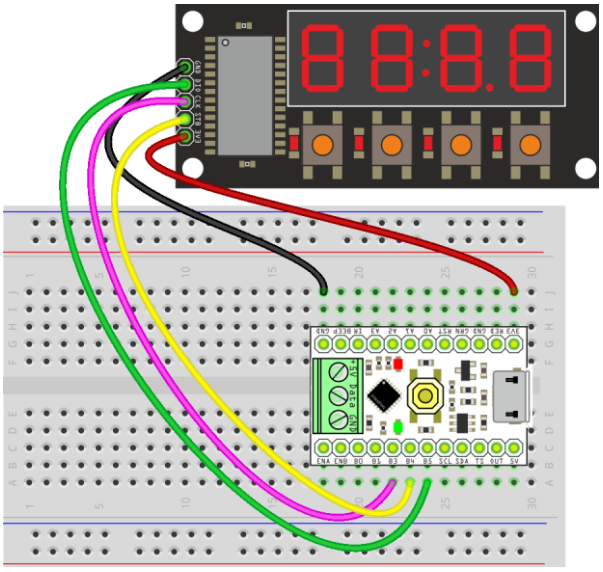
GND	GND
3V3	3V3
CLK	B3
STB	B4
DIO	B5

Verschiedene Varianten erhältlich, 8 Anzeigen mit 8 Taster, 6 Anzeigen mit 6 Taster, 4 Anzeigen mit 4 Taster. Anschlussbelegung und Abfrage erfolgt bei allen Varianten identisch. Die Ausgangsports müssen zunächst mit **IO.sys(501, 1)** initialisiert werden. B3, B4 und B5 stehen damit nicht mehr als normale Ausgangsports zur Verfügung. Ansteuerung der 7-Segment-Anzeige mit den LED-Befehlen für 7-Segment-Display. Die Anzeige muss mit **LED.update()** aktualisiert werden. Abfrage der Tasten mit **<var>=IO.sys(502, 0)**. Infos zu den SYS-Befehlen bei der PICO-Basic-Basisplatine. Die Helligkeit der Anzeige wird mit **LED.bright(x)** eingestellt (x = 0...4).

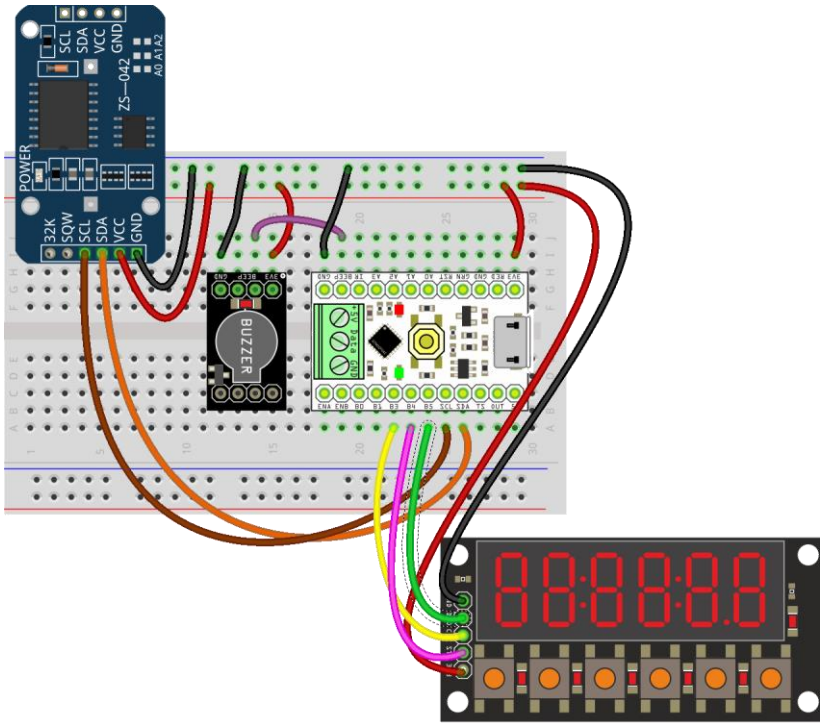
Beispiel: 8-Stellige Anzeige mit 8 Taster

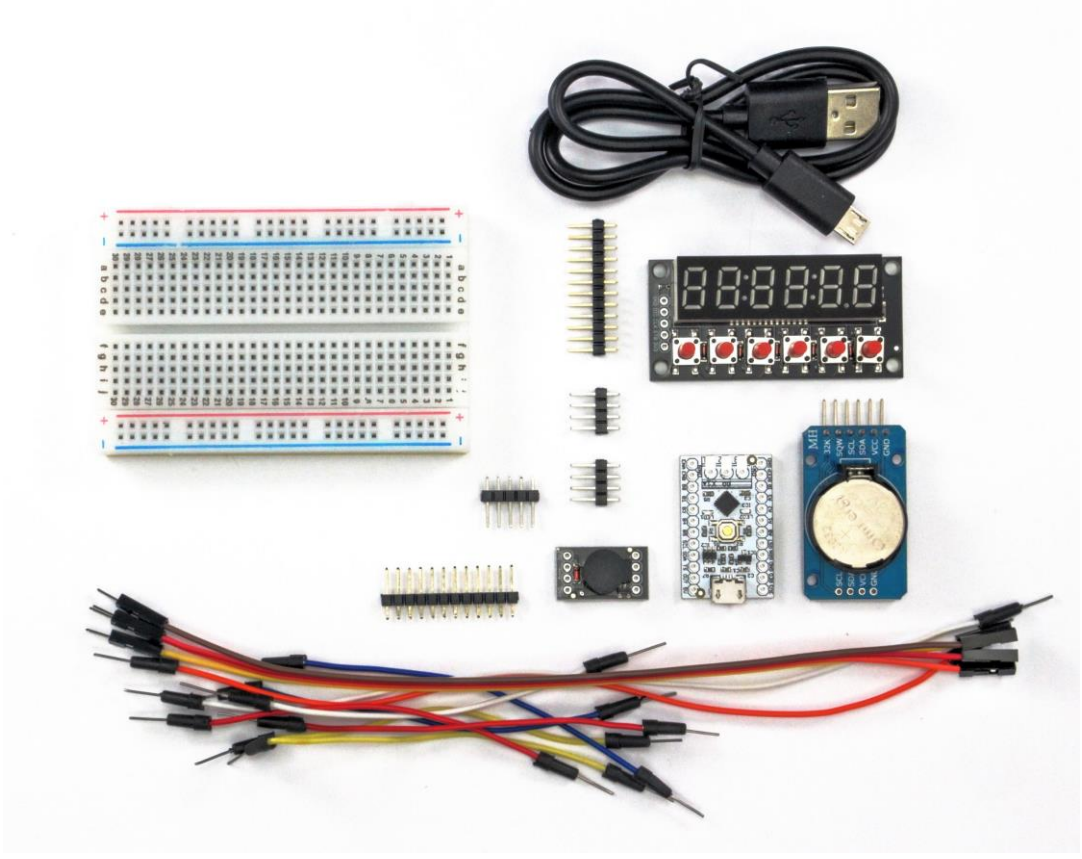


Beispiel: 4-Stellige Anzeige mit 4 Taster



Beispiel: Starterkit-Uhr mit 6-Stelliger Anzeige, RTC-Modul und Beeper
Demo-Software im PICO-SOURCE-Ordner von LED-BASIC.





Hinweis: Die Farben der gelieferten Verbindungskabel können von den Abbildungen abweichen.

LED-BASIC-PICO PROJEKTE

Die folgenden Projekte basieren auf der LED-BASIC-PICO- bzw. PICO2-Technik. Zum Programmieren der Projekte und anschließendem Hochladen der Software wählen Sie bitte in der LED-BASIC Entwicklungsumgebung die Komponente „PICO-Controller“ bzw. „PICO2-Controller“ aus. Beispiele und fertige Software für die Komponenten finden Sie im SOURCE-Verzeichnis.

PICO: NixieCron - SINGLE-DIGIT-Clock

Einstellige NixieCron-Uhr in verschiedenen Designs. Die Anzeige von Zeit und Datum erfolgt sequentiell.

Terminal Print- und Fehler-Ausgabe über USB. Integrierte hochpräzise Echtzeituhr DS3231 mit Stützbatterie. Bedienung und Einstellung über Drehimpulsgeber mit Taste, Beeper,

Stromversorgung und Programmierung über Micro-USB-Anschluss.

System-Code: 3370

Konfigurationszeile	LED-Befehle	IO-Befehle
L… C… M… P… S… F… Lxxx: Maximum = 64	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE SETENC, GETENC BEEP EEREAD, EEWRITE [0..2047][1] GETADC (Photodiode) GETRTC, SETRTC

Die Photodiode ist an A0 des PICO-Moduls angeschlossen. Zunächst mit **IO.sys(500, 0x01)** den Analog-Port auf A0 aktivieren, dann kann der Wert mit **IO.getadc(0)** gelesen werden.

Die Encoder-Taste ist an der Taste des PICO-Moduls angeschlossen, bei Druck wird der Wert 16 an die Funktionen IO.waitkey, IO.getkey und IO.keystate übergeben.

PICO: NixieCron – S4-„Jukebox“-Clock

4-stellige NixieCron-Uhr. Terminal Print- und Fehler-Ausgabe über USB. Integrierte hochpräzise Echtzeituhr DS3231 mit Stützbatterie. Bedienung und Einstellung über Drehimpulsgeber mit Taste, Beeper.

Stromversorgung und Programmierung über Micro-USB-Anschluss.

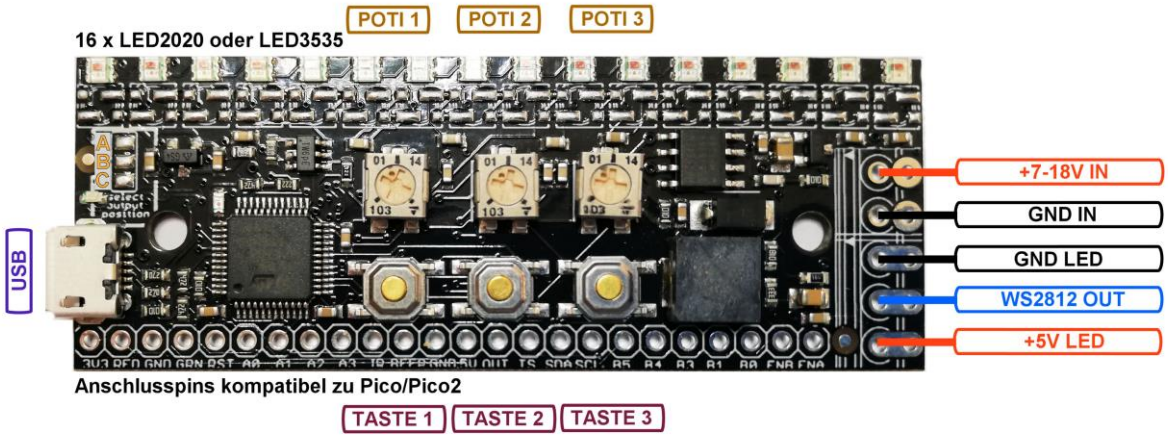
System-Code: 3370

Konfigurationszeile	LED-Befehle	IO-Befehle
L… C… M… P… S… F… Lxxx: Maximum = 64	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE SETENC, GETENC BEEP EEREAD, EEWRITE [0..2047][1] GETADC (Photodiode) GETRTC, SETRTC

Die Photodiode ist an A0 des PICO-Moduls angeschlossen. Zunächst mit **IO.sys(500, 0x01)** den Analog-Port auf A0 aktivieren, dann kann der Wert mit **IO.getadc(0)** gelesen werden.

Die Encoder-Taste ist an der Taste des PICO-Moduls angeschlossen, bei Druck wird der Wert 16 an die Funktionen IO.waitkey, IO.getkey und IO.keystate übergeben.

PICO2: Running Light, Lauflicht



Lauflicht-Modul auf PICO2-Basis für WS2812 kompatible LEDs. On-Board mit 16 RGB-LEDs in den Größen 2020 oder 3535 bestückt. Über den externen LED-Ausgang können bis zu 512 WS2812-kompatible RGB- oder RGBW-LEDs angesteuert werden. Schaltregler für die Versorgung des Moduls mit Spannungen bis zu 18 Volt. 3 Taster und 3 Potentiometer zur universellen Programmierung. Alle Schnittstellen des PICO2-Basismoduls sind zusätzlich an Lötanschlüssen herausgeführt.

Stromversorgung über USB-Buchse, 5 Volt über LED-Extern oder 7-15 Volt Anschluss.

Programmierung über USB-Anschluss.

System-Code: 3430

Konfigurationszeile	LED-Befehle	IO-Befehle
L… C… M… P… S… F… Lxxx: Maximum = 512	Alle Befehle für RGB-LEDs mit PWM	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT GETADC EEREAD, EEWRITE [0..31][1] SYS (siehe: Sys-Befehle) Alle weiteren Befehle für PICO2 mit zusätzlichen Modulen.

[1]Das interne EEPROM wird über die Adressen 10000 … 10031 angesprochen.

Die drei Potentiometer (Poti, regelbarer Widerstand) sind fest mit den Anschlüssen IOA0, IOA1 und IOA2 verdrahtet. Die drei Taster befinden sich an den Anschlüssen KEY_1 (16), ENC_A (64) und ENC_B (128). Die Funktionen der Taster und Potis entnehmen Sie bitte der Beschreibung zur Software.

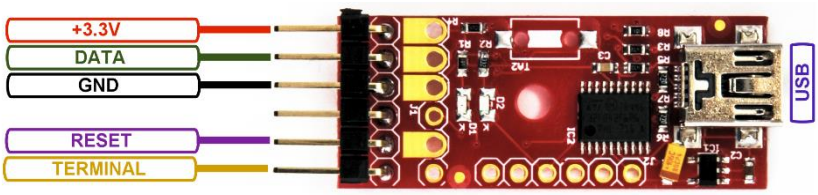
Über die Lötbrücke wird ausgewählt, welches LED-Signal am 3-poligen externen LED-Ausgang zur Verfügung steht. Sind Kontakte B-C verbunden, ist die erste LED am externen Anschluss auch die erste LED in der Lauflicht-Kette (#0). Sind Kontakte A-B verbunden, ist die erste LED am externen Anschluss die 17. LED in der Lauflicht-Kette (#16).

Der integrierte Schaltregler 7-18 Volt nach 5 Volt kann nur bis maximal 1A belastet werden, achten Sie bitte darauf, wenn Sie externe LEDs über den 3-poligen LED-Anschluss versorgen wollen. Wir ein höherer Strom benötigt, benutzen Sie bitte ein entsprechend starkes 5 Volt–Netzteil und versorgen die externen LEDs hiermit.

Zur Programmierung in der LED-Basic Entwicklungsumgebung „PICO2“ einstellen.

Zubehör, Erweiterungen

Prog-SB, Seriell-Basic Programmier- und Terminaladapter



LED-Basic Komponenten (z.B. Button12, Button16, Lauflicht oder Basic-Booster), die über keinen eigenen USB-Port verfügen, benötigen diesen Programmieradapter. Mit diesem sind dieselben Funktionen möglich, die auch bei Komponenten mit eigenem USB-Port möglich sind: Aufspielen des LED-Basic-Codes, eventuelles Bios-Update sowie Print- und Fehlerausgaben über das Terminal (nicht bei Badge12).

Bei der Verbindung mit Jumperkabeln benötigen Sie 4 oder 5 Verbindungen. Zum hochladen des LED-Basic-Codes und zum Bios-Update werden 4 Leitungen benötigt:

- **(PIN1) +3,3V**
- **(PIN2) DATA**
- **(PIN3) GND**
- **(PIN5) RESET**

(Pin 4 und 6 sind nicht beschaltet)

Wenn die Print- und Fehlerausgabe über das Terminal benutzt werden soll, muss zusätzlich noch PIN6 angeschlossen werden (bei Badge12 ohne Funktion).

- **(PIN6) TERMINAL**

(Pin 4 ist nicht beschaltet)

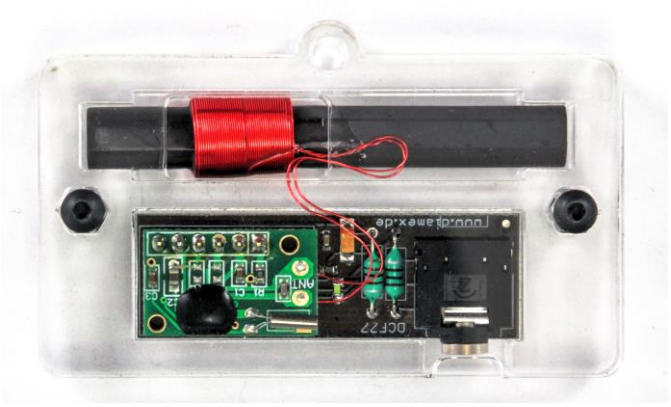
Die LED-Basic Komponenten werden über den Programmieradapter mit Strom versorgt, bitte entfernen Sie vor der Programmierung eine eventuell eingesteckte Batterie (bei Badge12/16).

Hinweise:

- TA2 wird auf der Prog-SB-Platine für die Seriell-Basic-Funktion nicht benötigt und ist aus diesem Grund nicht vorhanden.
- Bei der Programmierung können auch alle 6 Leitungen verbunden werden, die unbenutzten Leitungen beeinflussen die Programmierung nicht.

Installieren Sie den Treiber, der sich im LED-Basic Installationspaket befindet, unter Windows 7 und 8.x, wie im Kapitel „Treiberinstallation“ beschrieben. Für Windows 10 ist keine Treiberinstallation erforderlich.

DCF77-Zeitmodul



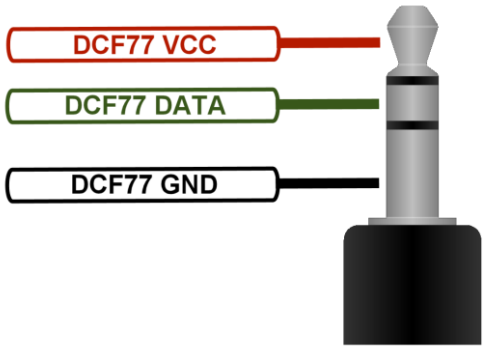
Dieses DCF77-Modul ist für alle LED-Basic-Komponenten mit Uhr-Funktion geeignet, die eine Bios-Version besitzen, in der DCF77 unterstützt wird.

LEDs mit integrierter PWM (z.B. WS2812, APA102) stören den DCF77-Empfang. Platzieren Sie den Empfänger möglichst weit entfernt von diesen LEDs. Die LED im DCF77-Empfängermodul muss im Sekundenrhythmus blinken und darf nicht flackern.

Wenn das Empfangssignal sauber ist, sollte sich die LED-Basic-Komponente nach maximal 3 Minuten auf die aktuelle Zeit synchronisieren.

- Ausgangssignal: positive Pulse in Höhe der Betriebsspannung
  (kein Pullup-Widerstand erforderlich)
- LED-Anzeige zur Empfangskontrolle
- Stromversorgung und Daten über 3,5mm Stereo-Klinkenbuchse
- Spannungsversorgung: 1,5 - 3,5V (ca. 50µA)

Steckerbelegung für alle Komponenten, die DCF77 unterstützen



Hinweis: Bitte achten Sie darauf, nur hochwertige Kabel mit 3,5mm Klinkenstecker zu benutzen. Viele preiswerte Kabel haben zu dünne Innenleiter oder sind nicht abgeschirmt.

GPS -> DCF-Zeitmodul

Wenn ein DCF77-Empfänger kein Signal empfängt, weil z.B. die nähere Umgebung mit zu viel Störstrahlung verseucht ist, kann dieses Modul helfen. Sobald freie Sicht zu den GPS-Satelliten existiert, empfängt dieses Modul die genaue über GPS übertragene GMT-Zeit und wandelt diese in ein DCF77-kompatibles Signal um.

- Ausgangssignal: positive oder negative Pulse in Höhe der Betriebsspannung (umschaltbar)
- Zeitdifferenz zu GMT maximal plus/minus 7 Stunden einstellbar
- Automatische Sommerzeit-Korrektur (DST) zuschaltbar
- 2 LEDs zur Empfangs- und Datenkontrolle
- Stromversorgung und Daten über 3,5mm Stereo-Klinkenbuchse (wie beim DCF77-Modul)
- Spannungsversorgung: 3,3V (ca. 60mA)

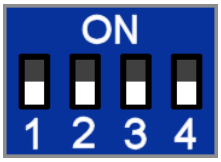
LED-Funktionen

GRÜN blinkt	Kein GPS-Empfang, es wurde noch keine gültige Zeit empfangen.
GRÜN leuchtet konstant	GPS-Empfang erfolgreich, Uhrzeit und Datum wurde empfangen.
ROT blinkt im Sekundentakt	Daten im DCF77-Format werden gesendet. Die LED zeigt das DCF77-Signal an, wie es zur Komponente gesendet wird: 100ms an bedeutet Bit=0, 200ms an bedeutet Bit=1.

Beachten Sie bitte, dass es ja nach Empfangsverhältnissen mehrere Minuten dauern kann, bis das Modul einen sauberen GPS-Empfang hat und die grüne LED konstant leuchtet.

DIP-Schalter A, Einstellung der Zeitzone

1:off, 2:off, 3:off, 4:off	Zeit = GMT
1:on, 2:off, 3:off, 4:off	Zeit = GMT+1 (Deutschland)
1:off, 2:on, 3:off, 4:off	Zeit = GMT+2
1:on, 2:on, 3:off, 4:off	Zeit = GMT+3
1:off, 2:off, 3:on, 4:off	Zeit = GMT+4
1:on, 2:off, 3:on, 4:off	Zeit = GMT+5
1:off, 2:on, 3:on, 4:off	Zeit = GMT+6
1:on, 2:on, 3:on, 4:off	Zeit = GMT+7
1:on, 2:off, 3:off, 4:on	Zeit = GMT-1
1:off, 2:on, 3:off, 4:on	Zeit = GMT-2
1:on, 2:on, 3:off, 4:on	Zeit = GMT-3
1:off, 2:off, 3:on, 4:on	Zeit = GMT-4
1:on, 2:off, 3:on, 4:on	Zeit = GMT-5
1:off, 2:on, 3:on, 4:on	Zeit = GMT-6
1:on, 2:on, 3:on, 4:on	Zeit = GMT-7



DIP-Schalter B, Konfiguration

1	Automatische Sommerzeitkorrektur (DST) wenn ON
2	Polarität des Ausgangssignal, OFF = Positiv, ON = Negativ

Alle LED-Basic-Komponenten benötigen ein positives Ausgangssignal (Schalter 2 = OFF)

Nach Ändern der Schalterstellungen muss das Modul neu gestartet werden, um die Einstellungen zu übernehmen (Klinkenstecker abziehen und wieder einstecken).

Hinweise:

- *Dieses Modul kann aufgrund der hohen Stromaufnahme nicht an die Cronios 1-Komponente angeschlossen werden.*
- *Bitte nur hochwertige abgeschirmte Klinkenkabel benutzen.*
- *Der GPS-Empfang ist in Innenräumen nicht möglich. Versuchen Sie es auf dem Fensterbrett oder unter schrägen Dachfenstern.*

WLAN -> DCF-Zeitmodul

Wenn DCF77 und GPS keine Chance haben, dann vielleicht dieses Modul. Über WLAN wird die aktuelle Zeit vom NTP-Server empfangen und in ein zu-DCF77 kompatibles Signal umgewandelt. Zur Konfiguration des WLAN-Zuganges ist ein Smartphone mit WLAN und Web-Browser erforderlich.

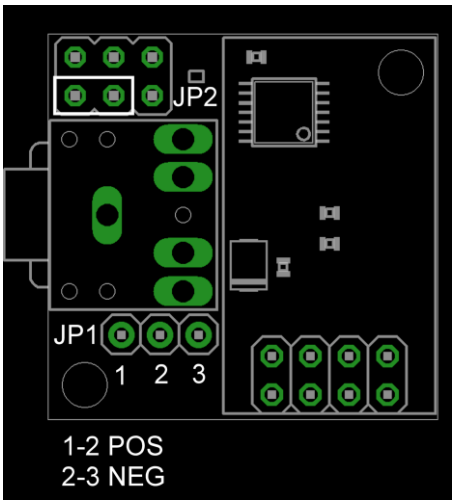
- Ausgangssignal: positive oder negative Pulse in Höhe der Betriebsspannung (mit Jumper einstellbar)
- LED zur Datenkontrolle
- Stromversorgung und Daten über 3,5mm Stereo-Klinkenbuchse (wie beim DCF77-Modul)
- Spannungsversorgung: 3,3V (max. ca. 200 mA)

Eine Verbindung zum WLAN-Router wird durch Leuchten der **blauen LED** auf dem WLAN-Modul angezeigt. Die **rote Kontroll-LED** blinkt gleichmäßig, wenn keine gültigen Zeitdaten empfangen wurden. Nach maximal einer Minute sollte die rote LED konstant leuchten und im Sekundenrhythmus flackern. Wenn keine Verbindung zu einem WLAN-Router besteht, wird automatisch ein WLAN-Zugangspunkt „DCF77_EMU“ gestartet. Wählen Sie mit Ihrem Smartphone diesen WLAN-Zugang aus, es ist kein Passwort erforderlich. Nach Öffnen des Internet-Browsers auf dem Smartphone und dem Aufruf einer beliebigen Internet-Seite erscheint das Konfigurationsmenü in dem Sie die Zugangsdaten für Ihren Router sowie die Zeitzone und die Sommer-/Winterzeit-Umschaltung konfigurieren können.

Hinweise:

- *Dieses Modul kann aufgrund der hohen Stromaufnahme nicht an die Cronios 1-Komponente angeschlossen werden.*
- *Bitte nur hochwertige abgeschirmte Klinkenkabel benutzen.*

JP2	Manueller Start des WLAN Konfigurations-Menüs.



JP1: 1-2	Positive Pulse (für Basic-Komponenten)
JP1: 2-3	Negative Pulse

Versionen

Hier finden Sie eine Auflistung aller veröffentlichten LED-Basic-Versionen.

V15.2.8

Neue Komponenten:
PICO2 – PICO1-Kompatibles Modul mit größerem Speicher für bis zu 512 LEDs.
PICO2-Lauflicht: Auf PICO2 basierendes Lauflicht mit 16 fest aufgelöteten LEDs.

V15.2.7

Neue Komponenten:
NixieCron - 24-Segment-Tube-Clock, die LED-Tube-Clock mit 24-Segment-Anzeige.
NixieCron - Matrix-Tube-Clock, die LED-Tube-Clock mit Matrix-Anzeige.
PICO-Projekte – Single-Digit-Clock, S4-Mini-Clock
Update:
Auf mehrfachen Wunsch wurde die Einstellung der Master-Helligkeit (Mxxx) in der Konfigurationszeile auf 1...100 geändert.
Neuer Befehl:
IO.BT für zukünftige Komponenten mit Bluetooth-Steuerung.

V15.2.6

Neue Komponente:
NixieCron - Cronios 3, die Cronios-Komponente mit Sound.
Update:
Upload-Funktion für Sound-Daten in der LED-Basic-Entwicklungsumgebung.

V15.2.5

Neue Komponente:
Basic-Flame, LED-Matrix-Anzeige als Kerzen-Simulation und Uhr.
Update:
MATRIX-Befehle für Komponenten mit LED-Matrizen (z.B. Basic-Flame)

V15.2.4

Bugfix: Bios-Update für PICO, LED-Tube und LED-Nixie-4.

V15.2.3

Bugfix: Bios-Update für Cronios 2 und LED-Nixie-4. In Verbindung mit einem DCF77-Empfänger kam es sporadisch zu Hängern des Betriebssystems.

V15.2.2

Neue Komponenten:
PICO, die Mini-Basic-Komponente mit vielen Erweiterungsmodulen.
DS3231->DCF77 Zeitmodul.

V15.2.1

Bugfix: Das neue Bios der LED-Tube-Clock führte zu einem Absturz der Uhr. Sollte dies passiert sein, kann die Uhr folgendermaßen wiederbelebt werden: USB-Stecker ziehen. 3. Taste von links drücken und halten. USB-Stecker einstecken. Mit der neuen LED-Basic-Version die Software hochladen, das Bios wird automatisch auf den neuesten Stand gebracht.

V15.2.0

Update:
Änderungen an der Entwicklungsumgebung. Achtung, neue Struktur der Beispieldateien. Sollte es nach dem Update zu Problemen mit den Beispieldateien kommen, deinstallieren Sie bitte die alte Version komplett und installieren die neue Version. Vergessen Sie bitte nicht, vor der Deinstallation ihre eigenen Dateien zu sichern.
Neue Komponenten:
RC-Box, Controller mit 1- oder 4-Kanal Funkfernbedienung
LED-Nixie-M4, 4-stellige LED-Nixie-Uhr
Hinweis:
Einige Komponenten wurden aus dieser Anleitung entfernt, die noch in Version 15.1.15 vorhanden waren. Grund hierfür sind Probleme bei der Entwicklung und Produktion der Komponenten. Diese Komponenten werden jedoch noch von der LED-Basic Software unterstützt und sind dort weiterhin vorhanden.

V15.1.15

Neue Komponenten:
LED-Tube, die LED-Uhr im VFD-Tube-Design.
Update: Echtzeituhr-Kalibrierung über IO.sys-Befehl bei Cronios-Controller.
Bugfix: Fehler bei IO.setenc beseitigt.

V15.1.14

Update: 16-Kanal-Lauflicht: Bug im Bios beseitigt. Anschlusspins für Prog-SB hinzugefügt.
APA-Booster: Bilder hinzugefügt und aktualisiert.
Neue Komponenten:
VFD-Clock, die Nostalgie-Uhr mit VFD-Röhren.
Touch-Lamp, die Effekt-Lampe mit Sensor-Tasten und Fernbedienung.
6-O-Clock, die 6-stellige LED-Uhr.
Nano-Stick, der Mini-USB-Stick mit 5 RGB-LEDs und programmierbarer Intelligenz.
Neuer Befehl: IO.sys(a, b) zum Lesen und Setzen von Systemparametern, bei einigen neuen Komponenten vorhanden.

V15.1.12

Update: Neue Version des 16-Kanal Lauflicht. Neue und aktualisierte Beispieldateien Cortex-Clock, Cronixie, Lauflicht
Neu: Unterstützung des DCF77-Moduls zur Zeitsynchronisierung bei Chronios1 (andere Komponenten folgen).

V15.1.11

Update: Cortex-Clock und Cronios-Segmenta besitzen jetzt eine EEPROM-Funktion. Damit können Helligkeiten, Farben und Alarmzeiten fest abgespeichert werden.
Neu und Korrektur: Anschlussbild LED-Box hinzugefügt. Das EEPROM in der LED-Box hat 8 Speicherstellen (0..7).
Änderungen: Button 12 und Button 16 heißen jetzt Badge 12 und Badge 16 (Badge = Abzeichen, Sticker, Plakette)

V15.1.10

Bugfix: In Data-Zeilen konnten keine negativen Werte benutzt werden.

V15.1.9

Bugfix: REM nach einem LABEL erzeugte einen Laufzeitfehler
Neu: APA-Booster hinzugefügt.
Update: Neuer Konfigurationswert **Ax** zum Einstellen der Bitrate bei APA102-Leds.

V15.1.8

Neu: LED-Matrix 10x10, All-In-One Cortex-M4-Power-Board und Led-Box hinzugefügt.
Update: Cronios1: EEPROM-Lesen/Schreiben hinzugefügt (1024 Adressen).

V15.1.7

Bugfix: Fehler im Lauflicht-Bios beseitigt.

V15.1.6

Neu: Budget-Board, Cortex-Clock, Temperatur-Sensor-Interface und Lauflicht hinzugefügt.

V15.1.5

Erste veröffentlichte Version

Fehler, Bugs

Wird in einem if-Statement eine IO-Funktion mit Parameter verwendet, erzeugt dieser einen Laufzeitfehler 11.

Beispiel:
```
if IO.eeread(0) <> 10 then …
```

Lösung 1:
```
t = IO.eeread(0)
if t <> 10 then …
```

Lösung 2:
```
if (IO.eeread(0)) <> 10 then …
```

Hinweise

Als Anregung für LED-Basic diente der µBasic-Interpreter von Adam Dunkel. Aufgrund der Auslagerung des Tokenizers auf den PC mit einem selbst entwickelten „Token-Code“ sowie der Hinzufügung von LED- und IO-Routinen ist daraus ein nahezu vollständiger eigener Code entstanden.

Obwohl diese Anleitung mehrfach überarbeitet und zur Kontrolle gelesen wurde, kann sie immer noch sachliche oder grammatikalische Fehler beinhalten. Über Hinweise zu Fehlern oder Verbesserungsvorschlägen sind wir Ihnen als Anwender sehr dankbar. E-Mail: feedback@led-basic.de

Links

LED-Basic Homepage
http://www.led-basic.de

LED-Genial Online-Shop
http://www.led-genial.de

Diamex-Shop
http://www.diamex.de

µBasic von Adam Dunkel
http://dunkels.com/adam/ubasic/

VERTRIEB

DIAMEX Produktion und Handel GmbH

Innovationspark Wuhlheide
Köpenicker Straße 325, Haus 41
12555 Berlin

Telefon: 030-65762631

E-Mail: info@diamex.de
Homepage: http://www.diamex.de

HERSTELLUNG

www.tremex.de
Köpenicker Str. 325 12555 Berlin
Tel.030-65762631
Hersteller: Tremex GmbH
DIAMEX * OBD-DIAG * TREMEX
WEE-Reg.Nr. DE 51673403