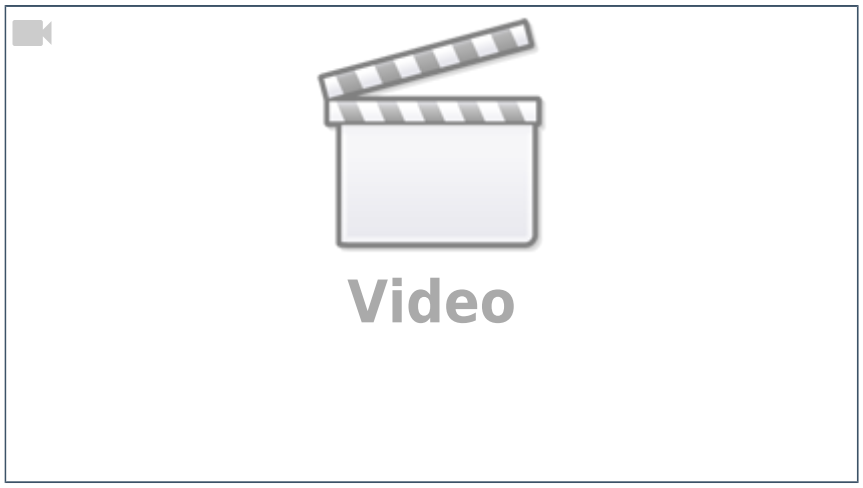
Rundumlicht & Baustellensicherungsanhänger

Rundumlichter für Einsatzfahrzeuge selbst gemacht

Um ein funktionierendes Rundumlicht selbst zu bauen, benötigt man nur vier gelbe LEDs vom Typ 0402 und ein bisschen Kupferlackdraht mit 0,1mm Stärke. Dazu selbstverständlich eine Lötstation mit passender Spitze, ein ruhiges Händchen und viele Ersatz-LEDs. Ist es einem dann aber doch gelungen, vier dieser winzigen LEDs in ein aufgebohrtes Rundumlicht zu kleben, geht es am Ende wieder mal an die Programmierung des Pattern Configurators.



Doch zuvor ein paar Tipps zum Einbau der LEDs:

- Das potentielle Rundumlicht wird zunächst mit 0,8mm und anschließend mit 1,5mm von Hand innen hohl gebohrt.
- Mit einem diamantbesetzten Schleifstift kann die Bohrspitze innerhalb des Lochs flacher gefräst werden.
- Ggf. kann mit diesem Fräser ein aus der Mitte geratenes Loch korrigiert werden, sodass man am Ende auf 1,7-1,8mm kommt.
- Hier ist Vorsicht geboten, die meisten Rundumlichter sind am oberen Ende nicht dicker als 2,2mm
- Das Rundumlicht wird mit Pressluft gereinigt (gut festhalten!).
- An jede LED 0402 werden zwei Kupferlackdrähte mit 0,1 mm Durchmesser gelötet, beide Litzen zeigen in dieselbe Richtung.
- Ggf. die LED von hinten dünn mit Klarlack isolieren, trocknen lassen! Das minimiert die Fehlerquote.
- Nun entweder die LEDs einzeln mit Sekundenkleber im Rundumlicht platzieren oder zunächst zum 4er-Bündel verkleben. Es wird eng!
- Einmal im Rundumlicht platziert, eignet sich zum Fixieren am besten transparentes Resin. Evtl. geht auch transparenter UV-Nagellack oder Bondic.
- Vor dem Fixieren unbedingt einen Funktionstest machen. Ggf. das Resin bei laufenden LEDs mit einer Nadel rein tropfen lassen und dann aushärten.
- Fertig!

Ein paar Worte zum verwendeten Kupferlackdraht:

- Kupferlackdraht mit 0,10 mm Durchmesser ist für 30 mA Strom geeignet.
- Kupferlackdraht mit 0,15 mm Durchmesser ist für 60 mA Strom geeignet.
- Kupferlackdraht mit 0,22 mm Durchmesser ist für 140 mA Strom geeignet.
- Da wir jede LED einzeln anschließen, reicht 0,1 mm. Die gemeinsame Rückleitung (ab Zusammenführung) muss nicht dicker als 0,15 mm sein, da nie mehr als zwei LEDs leuchten.

Nun wird programmiert. Wenn mehrere Fahrzeuge an einem Standort zu beleuchten sind, empfiehlt es sich, diese im Pattern Configurator zusammenzufassen. Das spart Speicher im Arduino. Für zwei Fahrzeuge legt man 16 Kanäle an. In diesem Fall reicht beim analogen Überblenden die „1“. Damit spart man 16 Byte RAM.

Damit die beiden Rundumlichter abwechselnd blinken, werden die Vollausschläge der Kanäle (LED Nr.) 5-8 um die Hälfte der Sequenz zu den Kanälen 1-4 verschoben. Das zweite Fahrzeug wird identisch aufgebaut und etwas verschoben, damit es nicht synchron blinkt. Voraussetzung ist, dass die LEDs identisch verkabelt sind.

Jede Spalte wird für 32 ms aktiviert. Dieser Wert gilt für alle zehn Spalten und muss nur in der ersten Spalte eingetragen werden. Insgesamt werden nur vier Helligkeitswerte benötigt (0, 33%, 67% und 100%). Es reichen also 2 Bits pro Wert.

Die Abfolge „1 - x - x - 1 - .“ lässt die LED insgesamt nur 160 ms leuchten, wobei sie meiste Zeit davon ein- bzw. ausgedimmt wird. Den Vollausschlag gibt es nur für 32 ms. Im folgenden Beispiel wurde mit 2 Bits pro Wert gearbeitet. Die „1“ entspricht daher 33% Helligkeit.

	Dauer	32									
	Flash Bedarf: 53 Bytes + - ☐ RGB LED										
LED Nr	Spalte Nr ->	1	2	3	4	5	6	7	8	9	10
1	LED1	.	.	.	.	.	1	x	x	1	.
2	LED2	1	.	.	.	.	.	.	1	x	x
3	LED3	x	x	1	.	.	.	.	.	.	1
4	LED1	.	1	x	x	1	.	.	.	.	.
5	LED2	x	x	1	.	.	.	.	.	.	1
6	LED3	.	1	x	x	1	.	.	.	.	.
7	LED1	.	.	.	1	x	x	1	.	.	.
8	LED2	.	.	.	.	.	1	x	x	1	.
9	LED3	.	.	1	x	x	1	.	.	.	.
10	LED1	.	.	.	.	1	x	x	1	.	.
11	LED2	.	.	.	.	.	.	1	x	x	1
12	LED3	x	1	.	.	.	.	.	.	1	x
13	LED1	1	.	.	.	.	.	.	1	x	x
14	LED2	x	x	1	.	.	.	.	.	.	1
15	LED3	.	1	x	x	1	.	.	.	.	.
16	LED1	.	.	.	1	x	x	1	.	.	.

Wie man effektiv Speicherbedarf reduzieren kann, zeigt dieses Beispiel.

Wenn der Ablauf soweit passt, geht es an die Helligkeitswerte. Beim Anhänger ist es nicht ganz so einfach wie beim Rundumlicht, weil die Helligkeitwerte der Blitzer evtl. andere sein können, als die des Pfeils. Also muss man sich ran tasten. Man beginnt am besten mit 4 Bits pro Wert, sprich 16 Helligkeitswerten (0-15) und mit einer maximalen Helligkeit von 255. Als Nächstes probiert man aus, ob Pfeil und Blitzlichter unterschiedliche Werte brauchen, indem man die Werte zwischen 1 und 15 testet. Wenn diese ähnlich sind, kann man die Bits pro Wert reduzieren und beides mit der gleichen Helligkeit ansteuern. Wenn sich am Ende herausstellt, dass beide mit ca. 30% der maximalen Helligkeit auskommen, so trägt man nicht wie oben die „1“ bei 2 Bits pro Wert ein, sondern man reduziert die maximale Helligkeit aller LEDs auf 30%, stellt auf 1 Bit pro Wert um und trägt ein „x“ in der Tabelle ein. So spart man auf einem Weg Speicher auf dem Arduino.

Im folgenden Screenshot ist der Ablauf der gesamten Sequenz mit einmaligem Blitzen der beiden Warnleuchten zu sehen. Alle Zeiten sind in diesem Beispiel in Millisekunden angegeben. Bei Millisekunden muss im Pattern Configurator keine Einheit angegeben werden. Für das im Video gezeigte Doppelblitzen werden die Spalten 1 bis 3 kopiert und zwischen den Spalten 4 und 5 eingefügt. Im unten gezeigten Beispiel ließe sich noch Speicher sparen, indem man die Spalten 4 und fünf zu einer zusammenfasst. Die beiden Spalten dienen als Gedankenstütze für die ermittelten Zeiten bei Verwendung des Doppelblitzes.

	Dauer	10	60	10	100	320	180	330	180	460	
Flash Bedarf: 40 Bytes + - ☐ RGB LED											
LED Nr	Spalte Nr ->	1	2	3	4	5	6	7	4	5	
1	LED1	x	x	.	.	.	.	.	.	.	
2	LED2	x	x	.	.	.	.	.	.	.	
3	LED3	.	.	.	.	.	x	x	.	.	
4	LED1	.	.	.	.	.	x	x	.	.	
5	LED2	.	.	.	.	.	x	x	.	.	
6	LED3	.	.	.	.	.	x	x	.	.	
7	LED1	.	.	.	.	.	x	x	.	.	
8	LED2	.	.	.	.	.	x	x	.	.	
9	LED3	.	.	.	.	.	x	x	.	.	