DCC-Dekoder

In Bearbeitung

DCC-Funktions-Dekoder mit ATtiny85

Funktions-Dekoder mit Digispark-Board

Auf dieser Seite wird ein Funktionsdekoder mit einem ATtiny85-Digispark-Board vorgestellt. Diesen Ansatz habe ich verwendet und die Hardware ergänzt, damit auch CVs in Wagen, die mit diesem Decoder ausgerüstet sind, auch auf dem Programmiergleis programmiert und zurückgelesen werden können. Der damit ausgestattete Dekoder hat dann nur noch 3 Ausgänge, da der 4. Port für das ACK-Signal zum CV-Lesen genutzt wird.

Der ergänzte Arduino-Sketch für die 3-Port-Variante mit ACK ist auf obiger Seite abgelegt, ebenso ein Sketch für 4 Ports ohne CV-Lesemöglichkeit. Das Programmieren kann aber „blind“ auf dem Programmiergleis erfolgen.

Eine gute Informationsquelle zur Programmierung des ATtiny85 ist Wolles Elektronikkiste.

Ich verwende nach ersten Test mit dem im Link erwähnten Upgrade auf die 300ms-Bootzeit nur noch die ISP-Programmierung ohne den Micronucleus-Bootloader.

Zusatzplatine 3-Port-Variante

Nach ersten positiven Versuchen mit einem fliegendem Aufbau und einer ersten Leiterplatte habe ich festgestellt, dass das Zurücklesen nicht immer zuverlässig klappt, wenn ein Stützkondensator angeschlossen ist. Eine zusätzliche Diode verhindert, dass der ACK-Impuls seine Energie aus dem Stützkondensator bezieht.
Die kleine Zusatzplatine wird rückseitig mit Stiften aufgelötet. Nachfolgend der überarbeitete LP-Entwurf mit der zusätzlichen Diode.

Schaltplan
Bestückungsplan
Stückliste
Bestellmöglichkeit der Leiterplatte für Eigenbau bei Aisler
Eine kommerzielle Nutzung ist untersagt.

Hier ein Einbaubeispiel der ersten Version der Platine im Pwgs/Daa-Wagen. Nachträglich wurde die zusätzliche Diode eingefügt, damit auch mit dem Stützkondensator ein CV-Lesen mit dem ACK-Signal zuverlässig funktioniert.

Die oben verlinkte Software-Variante wurde von mir weiterentwickelt. Da ich die USB-Funktion im endgültigen Einbau nicht benötige, verwende ich das Board ohne den Micronucleus-Bootloader sondern mit ISP-Programmierung ohne Bootloader mit dem STK500-Programmer. Für das Board habe ich eine Programmier-Adapter erstellt.

Ohne Bootloader sind ca. 20% mehr Programmspeicherplatz vorhanden, der für zusätzliche Funktionalität genutzt werden kann. Der aktuelle Sketch würde mit 98% Programmspeicher-Nutzung aber noch in den Programmspeicher mit Micronucleus-Bootloader passen.

Diese Funktionen wurden zugefügt:

- Funktionsmapping für zwei Ports.
- Dimmung der drei Ports

Informationen und Arduino-Sketch sind auf Github verfügbar.
Dort liegt auch eine ausführliche Funktionsbeschreibung: DCC-Funktionsdekoder (3-Port)

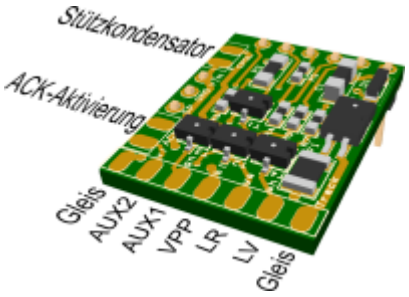
Folgende Funktionen sind aktuell als default konfiguriert:

- F0 Taste schaltet richtungsabhängig beide Schluss-LEDs an PB0 (LV) und PB4 (LR)
- F1 Taste schaltet Wagenbeleuchtung an PB1 (AUX1), die beiden Onboard-LEDs des Digispark-Boards wurden durch Entlöten des Vorwiderstands deaktiviert.

Zusatzplatine 4-Port-Variante

Wenn auf das Lesen der Konfigurationsvariablen verzichtet werden kann oder ein optionaler Schalter/Jumper möglich ist, dann kann auch Port PB3 als Ausgang genutzt werden. Mit PB3 (AUX2) ist aber kein Dimmen mit PWM über das analogWrite() möglich, dies wird an diesem Port nicht unterstützt. Das Dimmen ist über Software in der loop-Funktion realisiert worden.

Die Zusatzplatine wird rückseitig mit Stiften auf das Digispark-Board aufgelötet.



Schaltplan
Bestückungsplan
Stückliste
Bestellmöglichkeit der Leiterplatte für Eigenbau bei Aisler
Eine kommerzielle Nutzung ist untersagt.

Die Platine kann auch für die oben beschriebene 3-Port-Variante genutzt werden. Der Anschluss ACK muss dann überbrückt werden, PB3 wird dann nur für das ACK-Signal verwendet, AUX2 bleibt offen.

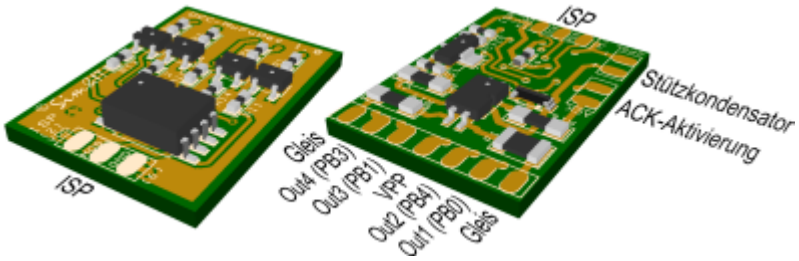
Funktions-Dekoder auf Lichtleiste

Auf der Basis der oben genannten Hard- und Software wurde eine Lichtleiste mit einem ATtiny85 aufgebaut. Diese benötigt ebenfalls keinen Bootloader, wie das Digispark-Board. Die Programierung erfolgt über einen 6-poligen ISP-Stecker mit dem STK500-Programmer. Es wird der gleiche Sketch wie bei der 3-Port-Variante verwendet. Es werden drei Ausgänge genutzt und das ACK-Signal wird beim CV-Lesen erzeugt.

Die Lichtleiste wird in SCHICHT-Rekowagen eingesetzt. Dort sind auch die Leiterplattendaten abgelegt.

Multi-Funktions-Dekoder 4-Port

Aus den obigen Erfahrungen wurde eine komplette Dekoder-Platine entwickelt, mit der die Sketche für Funktions-, Zubehör- und Signal-Dekoder ausgeführt werden können.



Dabei wurde die Reihenfolge der Zuordnung wie bei den beiden Zusatzplatinen beibehalten, so dass für einen Funktionsdekoder folgendes gilt:

- PB0 = Out1 = LV
- PB4 = Out2 = LR
- PB1 = Out3 = AUX1
- PB3 = Out4 = AUX2 / ACK

Die ISP-Programmierung erfolgt mit dem STK500-Programmer. Für die sechs Programmierpads habe ich eine Stiftleiste so modifiziert/gebogen, dass diese auf den Pads straff aufliegt. Für die einmalige/gelegentliche Verwendung ist das ausreichend.

Schaltplan
Bestückungsplan
Stückliste
Bestellmöglichkeit der Leiterplatte für Eigenbau bei Aisler
Eine kommerzielle Nutzung ist untersagt.

Zu den 4-Port-Varianten gibt es auf Github Informationen.
Dort liegt auch eine ausführliche Funktionsbeschreibung: DCC-Funktionsdekoder (4-Port)

Für die Nutzung mit der Software für die 3-Port-Variante muss zwischen den ACK-Lötpads eine Brücke gelegt werden, das Anschluss-Pad Out4 (PB3) wird nicht genutzt. PB3 wird dann nur für das ACK-Signal verwendet.

DCC-Zubehör-Dekoder mit ATtiny85

Entwicklungsboard

Nachdem ich den Funktionsdekoder auf Basis des ATtiny85 für die Beleuchtung meiner Wagen erfolgreich in Betrieb genommen hatte (s.o.), habe ich auf gleicher Basis die Entwicklung eines Sketches für einen Zubehördekoders begonnen, um damit Ausgänge u.a. für Beleuchtungen zu steuern.
Als ersten Ansatz dafür war ein Arduino-Beispielsketch der NmraDcc-Bibliothek von MRRWA. Dieser wurde dann um die Funktionen zur Ansteuerung der Ausgänge ausgebaut.

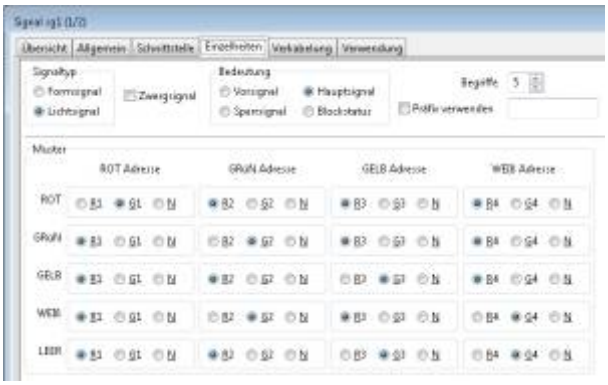
Mit den 4-Port-Platinen wie oben wird aus dem DCC-Signal die Spanung für das ATtiny85-Board generiert. An die Ausgänge des ATtiny sind Transistoren angeschlossen, um Verbraucher mit der gleichgerichteten DCC-Spannung versorgen zu können. Es kann optional ein Stützkondensator angeschlossen werden, was für stationäre Verbraucher aber eher nicht notwendig ist. Mit dem weiteren Anschluss ACK kann optional auch ein Lastwiderstand zugeschaltet werden, mit dem einen Ausgang zur Erzeugung des DCC-ACK-Signal genutzt wird. Damit ist dann bei geschlossenem ACK-Ausgang auch Lesen der CV-Werte mit dem Programmiergleis-Anschluss möglich.

Die vier Ausgänge (Ports) können mit DCC-Befehlen aktiviert und deaktiviert werden. Mit der Adresse+1 kann der Port als blinkend eingeschaltet werden. Die Blinkperiode (für alle Ausgänge gleich) ist konfigurierbar von 0,25 ... 3,75s.

PORT1 und PORT2 können als alternierend verbunden werden (z.B. Weichen, Signale) und es besteht auch die Möglichkeit, bei PORT1 und PORT2 einen Impuls definierter Länge (0,25 ... 8s) zu erzeugen, z.B. bei Weichen ohne Endabschaltung.

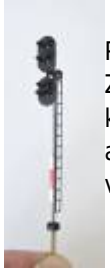
Für die Adressierung gibt es unterschiedliche Varianten. Ich habe mit Rocrail und der DCC-Commandstation von DCC-Ex die MADA-Adressierung ohne Probleme verwenden können. Andere Adressierungsarten kann ich mit dieser Commandstation nicht testen.
An einer Z21 sollte es mit der PADA-Adressierung funktionieren, wahrscheinlich jedoch mit einem Offset von +4.

Z.B. sind mit dem Zubehör-Dekoder auch Signalbilder im Rocrail mit den Einstellungen für Mustern für die 4 Ausgänge/Ports möglich. Es sind 5 Muster einstellbar, wenn man Muster „Leer“ auch belegt.

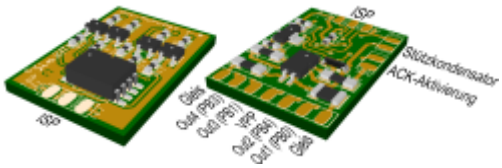


Link zu Github
Dort sind auch weitere Informationen verfügbar.

DCC-Signal-Dekoder mit ATtiny85



Die Hardware ist die gleiche, wie für die oben beschriebenen 4-Port-Funktions- bzw. -Zubehör-Dekoder. Es kann sowohl die Zusatzplatine zum DigiSpark-Board als auch die Platine für den kompletten DCC-Decoder verwendet werden. Auf letzterer sind aber Bestückungsplätze für die Vorwiderstände der LEDs vorgesehen.



Der Sketch verwendet die NmraDcc-Bibliothek von MRRWA die über die Arduino-Bibliotheksverwaltung eingebunden werden kann.

Es wird das erweiterte DCC-Paket-Format für Zubehör - Extended Accessory Decoder Control Packet Format - nach NMRA s-9.2.1, Abs. 2.4.2 verwendet. Der Dekoder nutzt nur eine Zubehör-Adresse. Damit sind in den meisten Systemen, z.B. auch DCC-Ex, 32 Signalbegriffe möglich. Der einzige in der Norm definierte Begriff ist der Begriff 0 für Stopp. Das ist hier der Signalbegriff Hl13 bzw. Hp0 mit 1x Rot. Alle weiteren Begriffe sind anwenderspezifisch frei definierbar.

Informationen und Beschreibung zum Dekoder auch bei Github.

Für EZMG-Signale mit Hl-Signal-Begriffen, die bei der DR auf Nebenbahnstrecken eingesetzt wurden, sind lt. Eisenbahnpraxis 6/1977 folgende Zuordnungen realisierbar:

- Einfahrsignal: Hl1, Hl9a, Hl10, Hl12a, Hl13 (Hp0) + Zs1
- Ausfahrsignal: Hl1, Hl3a, Hl10, Hl12a, Hl13 (Hp0) + Zs1, Ra12
- Vorsignal: Hl1, Hl7, Hl10

Bei den EZMG-Ausfahrsignalen sind im Realbetrieb aber meist nur zwei (Hp0, Hl3a) oder drei (Hp0, Hl1, Hl3a) Signalbegriffe mit Zusatzsignalen ausgeführt worden. Nicht benötigte Signalleuchten wurden abgedeckt.

Folgende Signal-Begriffe sind mit den 4 LEDs (statisch und blinkend) für eine Nebenbahn-Strecke möglich:

- Hp0 - Rot = Halt
- Hl1 - Grün = Fahrt mit Streckenhöchstgeschwindigkeit
- Hl3a - Grün + Gelb oben = Fahrt mit 40 km/h, dann mit Streckenhöchstgeschwindigkeit
- Hl7 - Gelb blinkend = Vorsignal - Höchstgeschwindigkeit auf 40 km/h ermäßigen
- Hl9a - Gelb oben blinkend + Gelb unten = Fahrt mit 40 km/h, Fahrt mit 40 km/h erwarten
- Hl10 - Gelb oben = Halt erwarten
- Hl12a - Gelb + Gelb = Fahrt mit 40 km/h, Halt erwarten

Das gelbe obere Licht ist bei den Ausfahrsignalen meist abgedeckt. Dieser Ausgang kann dann bei der Verdrahtung des Signals für 2x Weiß für das Rangiersignal Ra12 genutzt werden, wenn die Signalbilder Hl7 ... Hl12a nicht benutzt werden. Dieses Signalbild ist als Option für diesen Zweck auch im Sketch umgesetzt.

- Hp0 + Ra12 - Rot + 2x Weiß = Rangierfahrt erlaubt

Auf einer weiteren Seite wird der Bau der EZMG-Signale, die mit diesem Dekoder angesteuert werden, beschrieben.

Im Arduino-Sketch sind folgenden Konfigurationsvariablen (CVs) sind vorhanden:

- CV1 = 8 bit LSB, default 1
- CV9 = 3 Bit MSB, default 0
  - Dekoder-Adresse = LSB + MSB*256
- CV7 Versionsnummer, im Sketch eingestellt
- CV8 Hersteller-ID
  - entsprechend nmra-Bibliothek 13 für DIY
  - Schreiben auf CV8 führt einen Decoder Reset mit Default-Werten aus
- CV29 Configuration, default 192
  - Bit6=1 = Output Address Mode
  - Bit7=1 = Accessory Decoder Mode
- CV34 Blink-Periode - default 4 für 1 s Blink-Frequenz (4 bit für Blink-Periode in s (0.25 ... 3.75 s))
- CV51 Dim-Wert Rot
- CV52 Dim-Wert Grün
- CV53 Dim-Wert Gelb-oben bzw. 2x Weiß bei Ra12
- CV54 Dim-Wert Gelb-unten

Ansteuerung des Dekoders

Auf der Seite für die EZMG-Signale werden Beispiele für die Steuerung mit Rocrail gezeigt. Ich nutze die DCC-Ex-Commandstation. Die Signalbegriffe können auch mit einem Konsolenprogramm für die serielle Schnittstelle getestet werden.

Das Kommando dafür ist hier beschrieben: DCC-EX Native Commands Summary Reference