Software für Arduino:
Ein universeller Zubehördecoder mit DCC oder LocoNet Schnittstelle

Dies ist eine Software für einen DiY-Zubehördecoder auf Arduinobasis.
Für den Aufbau sind nur Grundkenntnisse in der Arduino-Programmierung notwendig. Der eigentliche Sketch muss normalerweise nicht angepasst werden. Das Verhalten kann über Konstantendefinitionen in einer Parameterdatei eingestellt werden.

Ab Version 6 kann der Decoder wahlweise mit DCC oder LocoNet Interface genutzt werden. Das Interface wird über ein #define in der Datei *interface.h* festgelegt. Die Funktionalität ist in beiden Fällen dieselbe.

Bei entsprechender externer Beschaltung und mit einer geeigneten Zentrale können die CV-Werte sowohl geschrieben, als auch gelesen werden. Beim LocoNet Interface gilt dasselbe für die SV-Werte. In der weitern Beschreibung wird allgemein von ‚CV' gesprochen. Dies gilt in gleicher Weise auch für die SV-Werte, soweit dies im Einzelnen nicht anders angegeben ist.
Für die SV-Programmierung beim LocoNet Interface ist es möglich passende JMRI Decoderfiles zu generieren. Die notwendigen Dateien sind im Verzeichnis *utility* enthalten (siehe Details im Abschnitt ‚INTERFACE'

Die Hardware muss selbst erstellt werden, ein Bausatz existiert nicht. Beispiel-Schaltpläne sind im Anhang vorhanden.

Die folgende Zubehörtypen können (auch gemischt) eingerichtet werden. Pro Typ können bis zu 3 Portausgänge konfiguriert werden.

- Servos (opt. mit Relais zur Polarisierung von Weichen)
- Doppelspulenantriebe
- statische Ausgänge
- blinkende Ausgänge (optional Wechselblinker)
- Lichtsignale

Alle belegten Zubehöradressen befinden sich in einem lückenlos aufeinanderfolgenden Block. Die 1. Adresse ist per Programmierung einstellbar. Die im Einzelfall nutzbare Zahl der Adressen ist von den verfügbaren Ausgängen abhängig. Es können aber maximal 16 Servos angesteuert werden.

Die parametrierbaren Eigenschaften des Decoders lassen sich über eine ‚Steuerdatei' (DCC_Zubehoerdecoder.h) einstellen, ohne den eigentlichen Sourcecode ändern zu müssen.

Parametrierbar sind:

- Portbelegung für Startverhalten (analoge Eingänge)
- Zahl und Typ der ansteuerbaren Zubehöradressen
- die Ports für die Zubehöranschlüsse
- initiale Grundeinstellungen für die jeweiligen Zubehörtypen (diese können per CV-Programmierung überschrieben werden.

Das Verhalten der konfigurierten Zubehörtypen wird über CV-Programmierung festgelegt:

- bei Servoausgängen die Endlagen und die Geschwindigkeit
- bei Doppelspulenantrieben die Einschaltzeit der Spulen.
- bei blinkenden Ausgängen das Blinkverhalten
- Bei Lichtsignalen die Zuordnung der Ausgangszustände zu den Signalzuständen.

Die Initialwerte dieser CV's werden ebenfalls in der Steuerdatei festgelegt. Nach dem erstmaligen Programmieren eines Arduino mit dem Decoder werden die Werte aus der Steuerdatei automatisch in die CV's übernommen.
Wird die Steuerdatei danach verändert und erneut übertragen, so werden die Werte aus der veränderten Steuerdatei nicht mehr automatisch in die CV-Werte übernommen. Die Übernahme muss dann explizit veranlasst werden (s. Kapitel Inbetriebnahme)

Inbetriebnahme

Zur Inbetriebnahme kann der Decoder in verschiedene Betriebsmodi gebracht werden. Dies wird über Spannungspegel an 2 analogen Eingängen beim Hochlauf bestimmt.
Die im folgenden mit A5 bzw. A4 bezeichneten Eingänge lassen sich per Einstellung in der .h-Datei auch auf andere Analogeingänge verlegen.

```
const byte betrModeP    =   A5;
const byte resModeP     =   A4;
```

A5:

- 5V (nur Pullup) normaler Betriebsmodus, kein PoM
- 3,3V (Spannungsteiler 1:2) PoM immer aktiv, Adresse immer aus defaults
- 1,6V (Spannungsteiler 2:1) IniMode: CV's werden immer auf init-Werte aus .h-Datei gesetzt
- 0V Programmiermodus / PoM (1. Empfangenes Telegramm bestimmt Adresse)

A4:

- Ist A4 beim Programmstart auf 0, werden alle CV's einschl. interner Statuswerte auf die Defaults (Werte aus der Steuerdatei) zurückgesetzt
- Im Betrieb können mit diesem Eingang zur Justierung die Servos in die Mittelstellung gebracht werden (nur wenn die Justierung per Encoder aktiv ist, s. Kapitel ‚Servo')

Das Zurücksetzen der CV's auf die Defaults kann auch von der Zentrale aus erfolgen. Dazu muss ein beliebiger Wert auf CV 8 (Herstellerkennung) geschrieben werden. Dabei wird dieser CV-Wert nicht geändert, vielmehr wird automatisch ein Decoder-Reset ausgelöst und alle CV-Werte auf die Werte aus der Steuerdatei gesetzt.

Definition der Zubehörfunktionen

Die gesamte Konfiguration des Zubehördecoders wird über Einstellungen in der Datei DCC_Zubehordecoder.h vorgenommen. Welche Zubehörfunktionen den jeweiligen Adressen zugeordnet werden, wird in Konstantendefinitionen festgelegt:

```
const byte iniTyp[]   = { FCOIL, FSIGNAL2, FSIGNAL0, FSERVO, FSERVO, FSTATIC, FSTATIC };
const byte out1Pins[] = {    A2,        9,       12,     A0,     A1,       7,       8 };
const byte out2Pins[] = {    A3,       10,        5,     NC,     NC,       6,      NC };
const byte out3Pins[] = {    NC,       11,       NC,     NC,     NC,      NC,      NC };
```

Die erste Zeile bestimmt die Funktion der jeweiligen Zubehöradresse, die darunterliegenden Zeilen die zugeordneten Ausgangspins. Es müssen nicht allen Adressen 3 Ausgänge zugeordnet werden. Nicht verwendete Ausgänge werden mit NC gekennzeichnet.

Das Verhalten der jeweiligen Funktionen wird über 4 (bei Lichtsignalen 5) CV-Werten pro Zubehöradresse angepasst. Für die erste Adresse sind dies CV50-53. Für alle weiteren Zubehöradressen gilt ein Offset von 5. D.h. z.B. für die 3. Zubehöradresse sind die CV's 60-63 zuständig.
Die Initialwerte dieser CV's werden ebenfalls in der Konfigurationsdatei festgelegt. Diese Werte können per CV-Programmierung überschrieben werden.

```
const byte iniFmode[] = {CAUTOOFF,       0, 0b10000, SAUTOOFF,    0, BLKMODE,        0 };
const byte iniPar1[]  = {      50, 0b00010, 0b10100,        0,    0,      50,        0 };
const byte iniPar2[]  = {      50, 0b10001, 0b11001,      180,  180,      50,        0 };
const byte iniPar3[]  = {       0,      50,       8,        8,    0,     100,        0 };
```

Funktionsbeschreibungen

Servos (initTyp = FSERVO)

Der Servo wird je nach Zustand der DCC-Adresse zwischen 2 konfigurierbaren Endpunkten gefahren. Die Geschwindigkeit ist ebenfalls konfigurierbar. Der Servoimpuls wird am 1. zugeordneten Output-Pin

ausgegeben. Es kann jeder Digitalpin verwendet werden.
Ist ein 2. Ouput-Pin defniert, so wird dieser in der Mitte des Verfahrweges des Servos umgeschaltet. Dies kann zur Weichenpolarisierung verwendet werden.
Ist auch der 3. Output-Pin definiert, so werden die Relais im Gegentakt angesteuert. Während der Servobewegung sind beide Relais abgefallen.

CV-Werte:

50+Offs	Fmode	Bit0 = 1: SAUTOOFF Servoimpulse werden bei Stillstand des Servo abgeschalter Bit1 = 1: SDIRECT das Servo kann auch während der Bewegung umgesteuert werden Bit3 = 1: NOPOSCHK Wird im Fall ‚SAUTOOFF' die aktuelle Position erneut angefahren, so werden erneut für 1sec Impulse ausgegeben (ist das Bit=0, so passiert in diesem Fall nichts).
51+Offs	Par1	Position des Servo für Weichenstellung '0' (in Grad, 0...180)
52+Offs	Par2	Position des Servo für Weichenstellung '1' (in Grad, 0...180)
53+Offs	Par3	Geschwindigkeit des Servo

Justierung der Servo-Ausgänge:

Die Justierung der Endlagen kann direkt durch Programmieren der CV-Werte erreicht werden. Wird der CV-Wert von Par1 oder Par2 verändert, so wird das Servo direkt auf diese neue Position gefahren. Dies erleichtert die Einstellung.

Alternativ kann zur Justierung ein Drehencoder angeschlossen werden. Dies vereinfacht die Einstellung noch weiter. Dazu müssen die Kommentarzeichen vor dem ‚#define ENCODER_AKTIV' entfernt werden. Die Pins des Encoders können dann nicht mehr für andere Zwecke genutzt werden.

```
//#define ENCODER_AKTIV       // Wird diese Zeile auskommentiert, wird der Encoder ...

const byte encode1P     =   A3;     // Eingang Drehencoder zur Justierung.
const byte encode2P     =   A2;
```

Der Drehencoder wirkt immer auf den Servo und die Servolage, die als letztes über einen DCC-Befehl eingestellt wurde. Sobald der Servo über einen DCC-Befehl wieder verstellt wird, wird die aktuell per Encoder eingestellte Position in den CV's gespeichert.
Wird A4 (Eingang ist konfigurierbar, s.o., Kapitel Inbetriebnahme) während der Einstellung auf 0 gezogen, wird der aktuell vom Drehencoder beeinflusste Servo in die Mittellage gebracht. Sobald der Encoder wieder bewegt wird, bewegt sich das Servo wieder zur vorhergehenden Position.

Doppelspulenantriebe (initTyp = FCOIL)

Diese Funktion dient zur Ansteuerung von Weichen mit Doppelspulenantrieben. Es müssen 2 Ausgangspins zugeordnet werden. Die Länge des Schaltimpulses und die minimale Pause zwischen 2 Impulsen kann konfiguriert werden.
Der Spulenausgang kann wahlweise automatisch oder über einen DCC-Befehl abgeschaltet werden.
Abschaltung per DCC-Befehl: Ist das Flag CAUTOOFF gesetzt, verhält es sich wie bisher. D.h. das Ausschalten wird automatisch entsprechend der in iniPar1 eingestellten Zeit gemacht. Ist das Flag nicht gesetzt, so reagiert der Ausgang auch auf den DCC-Abschaltbefehl
Inipar2 definiert wie bisher die Mindestpause zwischen 2 Pulsen um die Spule zu schonen. Bei Bedarf kann aber auch das auf 0 gesetzt werden.
Beide Flags können auch gleichzeitig gesetzt sein: NOPOSCHK|CAUTOOFF
NOPOSCHK=8 (Bit3) , CAUTOOFF=1 (Bit0)

CV-Werte:

50+offs	Fmode	Bit0 = 1: Spulenausgang nur automatisch abschalten = 0: Spulenausgang auch über DCC-Befehl abschalten.(das erste Ereignis, Zeit oder DCC, schaltet den Ausgang ab). Die Zeit kann deaktiviert werden, indem iniPar1 auf 0 gesetzt wird. Dann wird nur über den DCC-Befehl abgeschaltet.

		Bit3 = 0: Ändert sich die aktuelle Weichenpos. nicht, wird kein Impuls ausgegeben = 1: keine Überprüfung auf Weichenposition
51+offs	Par1	Einschaltdauer der Spule (in 10ms Einheiten)
52+offs	Par2	minimale Ausschaltdauer der Spule (in 10ms Einheiten)
53+offs	Par3	reserviert

Statische / Blinkende Ausgänge (initTyp = FSTATIC)

Mit dieser Funktion kann ein Ausgang statisch ein/ ausgeschaltet werden (z.B. für Beleuchtung). Ausserdem ist es möglich den Ausgang blinken zu lassen.
Werden 2 Ausgangspins zugeordnet, so schalten diese im Gegentakt um.
Im Blinkmodus blinken sie im Gegentakt (Wechselblinker) bei eingeschalteter Zubehörfunktion. Wird per DCC-Befehl ausgeschaltet, so sind beide Ausgänge inaktiv.

CV-Werte:

50+offs	Fmode	Bit0 = 1: Blinken, 0: statisch Bit1 = 1: Beim Blinken starten erst beide Leds dann Wechselblinken Bit2 = 1: mit weichem Auf/Abblenden Bits4..7: Risetime für das Auf/Abblenden in 50ms-Schritten. 0 bedeutet Default (500ms)
51+offs	Par1	Einschaltzeit des Blinkens (10ms Einheiten)
52+offs	Par2	Ausschaltzeit des Blinkens (10ms Einheiten)
53+offs	Par3	Erste Einschaltzeit beim Start des Blinkens (10ms Einheiten)

Lichtsignale (initTyp = FSIGNAL2 / FVORSIG)

Zur Ansteuerung von Lichtsignalen können bis zu 3 Zubehöradressen zu einer Signalfunktion zusammengefasst werden. Der ersten (oder einzigen) Adresse wird der Funktionstyp FSIGNAL2 oder FVORSIG zugeordnet, der bzw. den Folgeadressen der Typ FSIGNAL0.
Aus der Zusammenfassung von bis zu 3 Adressen ergeben sich 2 bis 6 mögliche Signalzustände.
Ausserdem können dem Signal bis zu 8 (bei 3 Adressen) Ausgänge zugeordnet werden.
Über CV-Parameter kann frei konfiguriert werden, welche Ausgangspins bei den jeweiligen Signalzuständen aktiv (= ON) werden sollen. Dabei ist jedem DCC-Befehl ein CV-Wert zugeordnet.
Ausserdem ist für jeden Ausgangspin einzeln einstellbar, ob er weich oder hart umschalten soll.

Die zu schaltenden Ausgangspins werden jeweils in den 8 Bits eines CV-Parameters konfiguriert.
Dabei folgende Zuordnung:
Bit0 out1pin der 1. Zubehöradresse des Signals
Bit1 out2pin der 1. Zubehöradresse des Signals
Bit2 out3pin der 1. Zubehöradresse des Signals
Bit3 out1pin der 2. Zubehöradresse des Signals
. usw
.
Bei den Ausgängen, die für eine ‚weiche' Umschaltung konfiguriert ist, wird zwischen altem und neuem Signalbild eine kurze Pause eingefügt, in der alle Ausgänge ausgeschaltet werden. Diese Überblendzeit ist global im .h File einstellbar. Dies gilt ebenso für die Auf bzw. Abblendzeit. Durch das ‚Nachglühen' bedeutet dies bei kurzen Überblendzeiten nicht, dass tatsächlich alle angeschlossenen Led's vollkommen dunkel werden.

Beim Programmstart sind alle Lichtsignale im Zustand 0, der Zustand beim Ausschalten des Decoders wird bei Lichtsignalen nicht gespeichert.

Vorsignale
Vorsignale verhalten sich im Prinzip wie Lichthauptsignale. Bei Vorsignalen kann jedoch eine zusätzliche Adresse parametriert werden, auf die das Signal auch reagiert. Dies ist in der der Regel die Adresse des Hauptsignals, das vom Vorsignal angekündigt wird. Damit kann das Vorsignal automatisch mit dem zugehörigen Hauptsignal geschaltet werden. Hat das Haupt- und Vorsignal mehr als 2 Begriffe, muss die

entsprechende Adresse auch bei den Folgeeinträgen angegeben werden

Dunkeltastung:
Befindet sich das Vorsignal zusammen mit einem Hauptsignal an einem Mast, kann es automatisch dunkelgeschaltet werden, wenn das Hauptsignal Hp0 / Hp00 anzeigt. Dazu wir beim Hauptsignal der Index des Vorsignals eingetragen (Position in der Funktionstabelle, beginnend mit 1, 0= kein Vorsignal). Ausserdem ein Bitmuster, das festlegt bei welchen Signalbildern des Hauptsignals das Vorsignal dunkelgeschaltet wird.

CV-Werte:

50+offs	Fmode	Bit7 = 1 : invertiert die Softled-Ausgänge (HIGH=OFF) (MobaTools ab V0.9) Bit 0..2: Bitmuster hard/soft gibt an, welche Ausgänge 'hart' umschalten (Bit=1) und welche Ausgänge weich überblenden (Bit=0) Bit0: out1Pin, … Bit2: out3Pin
51+offs	Par1	Bitmuster der Ausgänge für Befehl 1.Adresse 0 (rot) 255 bedeutet Befehl ignorieren
52+offs	Par2	Bitmuster der Ausgänge für Befehl 1.Adresse 1 (grün) 255 bedeutet Befehl ignorieren
53+offs	Par3	FVORSIG: low Byte der Hauptsignaladresse FSIGNAL: Index des Vorsignals am gleichen Mast (0 ….) 0= kein Vorsignal, Index zählt ab 1
54+offs		FVORSIG: high Byte der Hauptsignaladresse FSIGNAL: Bitmuster der Zustände, bei denen das Vorsignal dunkel ist: Bit 0: Befehl 1.Adresse 0 (rot) Bit 1: Befehl 1.Adresse 1 (grün) Bit 2: Befehl 2.Adresse 0 (rot) u.s.w.
CV‘s der 1. Folgeadresse:		
55+offs	Fmode	Bit 0..2: Bitmuster hard/soft gibt an, welche Ausgänge 'hart' umschalten (Bit=1) und welche Ausgänge weich überblenden (Bit=0) Bit0: out1Pin, … Bit2: out3Pin
56+offs	Par1	Bitmuster der Ausgänge für Befehl 2.Adresse 0 (rot)
57+offs	Par2	Bitmuster der Ausgänge für Befehl 2.Adresse 1 (grün)
58+offs	Par3	reserviert
59+offs		reserviert
CV‘s der 2. Folgeadresse:		
60+offs	Fmode	Bit 0..2: Bitmuster hard/soft gibt an, welche Ausgänge 'hart' umschalten (Bit=1) und welche Ausgänge weich überblenden (Bit=0) Bit0: out1Pin, … Bit2: out3Pin
61+offs	Par1	Bitmuster der Ausgänge für Befehl 3.Adresse 0 (rot)
62+offs	Par2	Bitmuster der Ausgänge für Befehl 3.Adresse 1 (grün)
63+offs	Par3	reserviert
64+offs		reserviert

Interface

Der Decoder kann wahlweise für ein DCC oder LocoNet Interface übersetzt werden. Die Interfacespezifischen Konfiguration wird in der Datei *interface.h* festgelegt. Dies ist zum einen die Art des Interfaces, und zum andern die Arduino-Pins, an denen das Interface angeschlossen ist.

```
//#define LOCONET // Wird dies auskommentiert, wird ein DCC-Interface eingebunden
```

Die obige Zeile definiert, ob ein DCC oder LocoNet Interface eingebunden wird. Um das LocoNet zu aktivieren, müssen die Kommentarzeilen am Anfang der Zeile entfernt werden.

DCC-Interface

Standardmäßig ist der Decoder mit einem DCC-Interface ausgerüstet. Für das DCC-Interface muss der Eingangspin (muss interuptfähig sein) und der AckPin für die Rückmeldung bei der CV Programmierung festgelegt werden. Im #else Zweig werden die Pins für die AVR-Arduinos festgelegt.

```
    #if defined ( ARDUINO_MAPLE_MINI )
        const byte dccPin       =   3;
        const byte ackPin       =   18;
    #elif defined (ARDUINO_GENERIC_STM32F103C)
        const byte dccPin       =   PB0;
        const byte ackPin       =   PB4;
    #else
        const uint8_t dccPin       =   2;
        const uint8_t ackPin       =   4;
    #endif
#endif
```

LocoNet-Interface

Die Variante mit LocoNet Interface ist nicht auf allen Plattformen lauffähig, auf denen das DCC Interface möglich ist. Das LocoNet Interface funktioniert nur auf AVR-Prozessoren, die über mindestens 2 16-Bit Timer verfügen. Dies gilt für die Varianten:

Arduino Leonardo, Micro und pro Micro (alle Varianten mit ATmega32U4)
Arduino Mega

Frei wählbar ist nur der Sendepin für das LocoNet Interface. Der Empfangspin ist in der LocNet Library festgelegt.

```
const uint8_t txPin        =   2;
// Das Empfangssignal MUSS auf dem pin 4 ( bei Micro/Leonardo ) oder pin 48
//( bei Mega ) liegen.
```

SV-Programmierung

Der CV-Programmierung beim DCC-Interface entspricht die SV-Programmierung bei LocoNet. Da es hier kein ‚Programmiergleis' gibt, sondern immer alle Geräte parallel am LocoNet Bus hängen, muss der Decoder für die SV-Programmierung individuell adressiert werden. Im Prinzip entspricht dies der POM-Programmierung bei DCC. Beim LocoNet Interface wird deshalb die POM-Adresse (CV 48/49) als LocoNet ID interpretiert. Um keine ungültigen Zwischenstände beim Umprogrammieren beider Teilwerte zu erhalten, wird der Wert bei Änderungen erst nach ca. 2Sek. Als neue LocoNet ID übernommen. In dieser Zeit mussen dann beide Werte auf die neue Adresse umgestellt werden.

JMRI

Zur komfortableren Konfigurierung der SV-Werte beim LocoNet Interface, können angepasste xml-Dateien für JMRI erstellt werden. Diese Dateien können dann in JMRI importiert werden, um dort einen neuen

Decoder einzurichten.Die notwendigen Dateien und Tools sind im Verzeichnis *utilities* enthalten. Zur Generierung der xml-Dateien wird das Programm ‚gpp.exe' (Windows) verwendet. Dies ist ebenfalls in diesem Verzeichnis enthalten, eine gesonderte Installation ist nicht notwendig.
Auf Linux-Systemen muss das Programmpaket GPP getrennt installiert werden (s. Datei gpp.pdf)

Um eine neu angepasste xml-Date für JMRI zu erstellen, muss ein CrConfig-JMRI* Datei erstellt werden. Dies ist eine Skriptdatei, die in den ersten Zeilen die Funktionsdefinitionen für den zu konfigurierenden Decoder enthält.
Dazu die Beispieldatei CrConfig-JMRI-DIY_Standard.cmd Datei kopieren, und so umbenennen, dass ‚DIY_Standad' in den neuen Decodertyp umbenannt wird. Die kopierte und umbenannte Datei muss sich im Gleichen Verzeichnis befinden, wie die Beispieldatei.
In dieser Datei den Decodernamen und die Funktionen entsprechend der Konfigurationsdatei des Decoders anpassen (s. Kommentare in der Beispieldatei). Danach die Datei durch Doppelklick ausführen.

Es entsteht dann im gleichen Verzeichnis die xml-Datei für JMRI. Diese Datei muss dann in JMRI importiert werden.
.

CV-Werte - Übersicht

folgende CV's werden vom Zubehördecoder verwendet:

CV-Nr	Bedeutung
1/9	1. Weichenadresse (nur bei DCC)
2	Enthält die Versionsnummer in Hex (0x62) (nur LocoNet)
7	Enthält die Versionsnummer in Hex (0x62) (nur DCC)
8	Herstellerkennung (=13, DIY-Decoder). Ein Schreiben auf diese CV führt zu einem Factory-Reset (laden aller CV's aus der Konfig-Datei, alle Ausgänge in Grundstellung)
17/18	1. Weichenadresse (nur bei LocoNet)
29	Betriebsarteneinstellung nach DCC (Decodertyp Zubehödecoder, Adressierungsart: Output-Adressierung). Der Inhalt von CV29 ist spezifisch für diesen Decoder und kann nicht verändert werden.
45	Kennung für Erstinitiierung. Da die Umsetzung SV→EEPROM-Adresse und CV→ EEPROM-Adresse um 2 versetzt ist, werden 2 um 2 versetzte Initkennungen verwendet. Damit ist sichergestellt, das bei Interfacewechsel sicher die Ungültigkeit der Daten erkannt wird. Bit7-4 = 0x50 bei DCC Bit7-4 = 0xA0 bei LocoNet andere Werte führen beim Hochlauf zur Grundinitiierung
47	Kennung für Erstinitiierung, allgemeine Optionen die für den gesamten Decoder gelten Bit7-4 = 0x50 bei DCC, Bit7-4 = 0xA0 bei LocoNet, andere Werte führen beim Hochlauf zur Grundinitiierung Bit0=1: Automatische Adresserkennung im Programmiermodus Bit1=1: Roco-Modus (dcc-Adresse 0 = Weichenadresse 1) Bit1=0: Std-Modus (dcc-Adresse 4 = Weichenadresse 1)
48/49	PoM-Adresse (bei DCC) / Loconet-Id (bei LocoNet)
50-54	Parameter für 1. Zubehöradresse
55-59	Parameter für 2. Zubehöradresse
...	

Bedeutung der CV's bei den verschiedenen Funktionen (CV-Nummern für 1. Weichenadresse)	
Servo:	
50	Bit0 = 1: AutoOff der Servoimpulse bei Stillstand des Servo Bit1 = 1: Direkte Umsteuerung des Servos möglich (ohne das Endlage erreicht wurde) Bit3 = 1: Wird im Fall ‚SAUTOOFF' die aktuelle Position erneut angefahren, so werden erneut für 1sec Impulse ausgegeben (ist das Bit=0, so passiert in diesem Fall nichts).
51	Position des Servo für Weichenstellung '0' (in Grad, 0...180)
52	Position des Servo für Weichenstellung '1' (in Grad, 0...180)
53	Geschwindigkeit des Servo
54	aktuelle Weichenstellung (nicht manuell verändern!)
Doppelspulenantrieb:	
50	Bit0 = 1: Spulenausgang automatisch abschalten = 0: Spulenausgang über DCC-Befehl abschalten Bit3 = 1: keine Überprüfunge der aktuellen Weichenposition = 0: ein Puls wird nur ausgegeben wenn sich die Weichenposition ändert
51	Einschaltdauer der Spule (in 10ms Einheiten) Ist der Wert 0, kann nur über einen DCC-Befehl abgeschaltet werden (Bit0 im ModeBit MUSS dann 0 sein)
52	minimale Ausschaltdauer der Spule (in 10ms Einheiten)
53	-
54	aktuelle Weichenstellung (nicht manuell verändern!)
statischer/Blinkender Ausgang	
50	Bit0 = 1: Blinken, 0: statisch Bit1 = 1: Beim Blinken starten erst beide Leds dann Wechselblinken Bit2 = 1: mit weichem Auf/Abblenden Bits4..7: Risetime für das Auf/Abblenden in 50ms-Schritten. 0 bedeutet Default (500ms)
51	Einschaltzeit des Blinkens (10ms Einheiten)
52	Ausschaltzeit des Blinkens (10ms Einheiten)
53	Erste Einschaltzeit beim Start des Blinkens
54	aktueller Zustand (nicht manuell verändern!)
Lichtsignale	
50	Signalmodus: Bit7=1 : invertiert die Softled-Ausgänge (HIGH=OFF) (MobaTools ab V0.9 Bit 0..2: Bitmuster hard/soft gibt an, welche Ausgänge 'hart' umschalten (Bit=1) und welche Ausgänge weich überblenden (Bit=0). Bit0: out1Pin, … Bit2: out3Pin
51	Bitmuster der Ausgänge für Befehl 1.Adresse 0 (rot)
52	Bitmuster der Ausgänge für Befehl 1.Adresse 1 (grün)
53	FVORSIG: low Byte der Hauptsignaladresse FSIGNAL: Index des Vorsignals am gleichen Mast (0 ….)
54	FVORSIG: high Byte der Hauptsignaladresse FSIGNAL: Bitmuster der Zustände, bei denen das Vorsignal dunkel ist: Bit 0: Befehl 1.Adresse 0 (rot) Bit 1: Befehl 1.Adresse 1 (grün) Bit 2: Befehl 2.Adresse 0 (rot)

	u.s.w.
55	Bit 0..2: Bitmuster hard/soft gibt an, welche Ausgänge 'hart' umschalten (Bit=1) und welche Ausgänge weich überblenden (Bit=0). Bit0: out1Pin, … Bit2: out3Pin
56	Bitmuster der Ausgänge für Befehl 2.Adresse 0 (rot)
57	Bitmuster der Ausgänge für Befehl 2.Adresse 1 (grün)
58	reserviert
59	reserviert
60	Bit 0..2: Bitmuster hard/soft gibt an, welche Ausgänge 'hart' umschalten (Bit=1) und welche Ausgänge weich überblenden (Bit=0). Bit0: out1Pin, … Bit2: out3Pin
61	Bitmuster der Ausgänge für Befehl 3.Adresse 0 (rot)
62	Bitmuster der Ausgänge für Befehl 3.Adresse 1 (grün)
63	reserviert
64	reserviert

Beispiel-Decoder (DCC)

Beispiel einer Decoderschaltung für 4 Servos (3 mit Polarisierung), 1 Doppelspulenantrieb und 2 statischen Ausgängen.
Die Schaltung kann wahlweise mit einem Arduino Uno oder einem Arduino Nano aufgebaut werden. Im Schaltbild sind beim Arduino deshalb keine Pin-Nummern, sondern nur die Pinnamen angegeben.

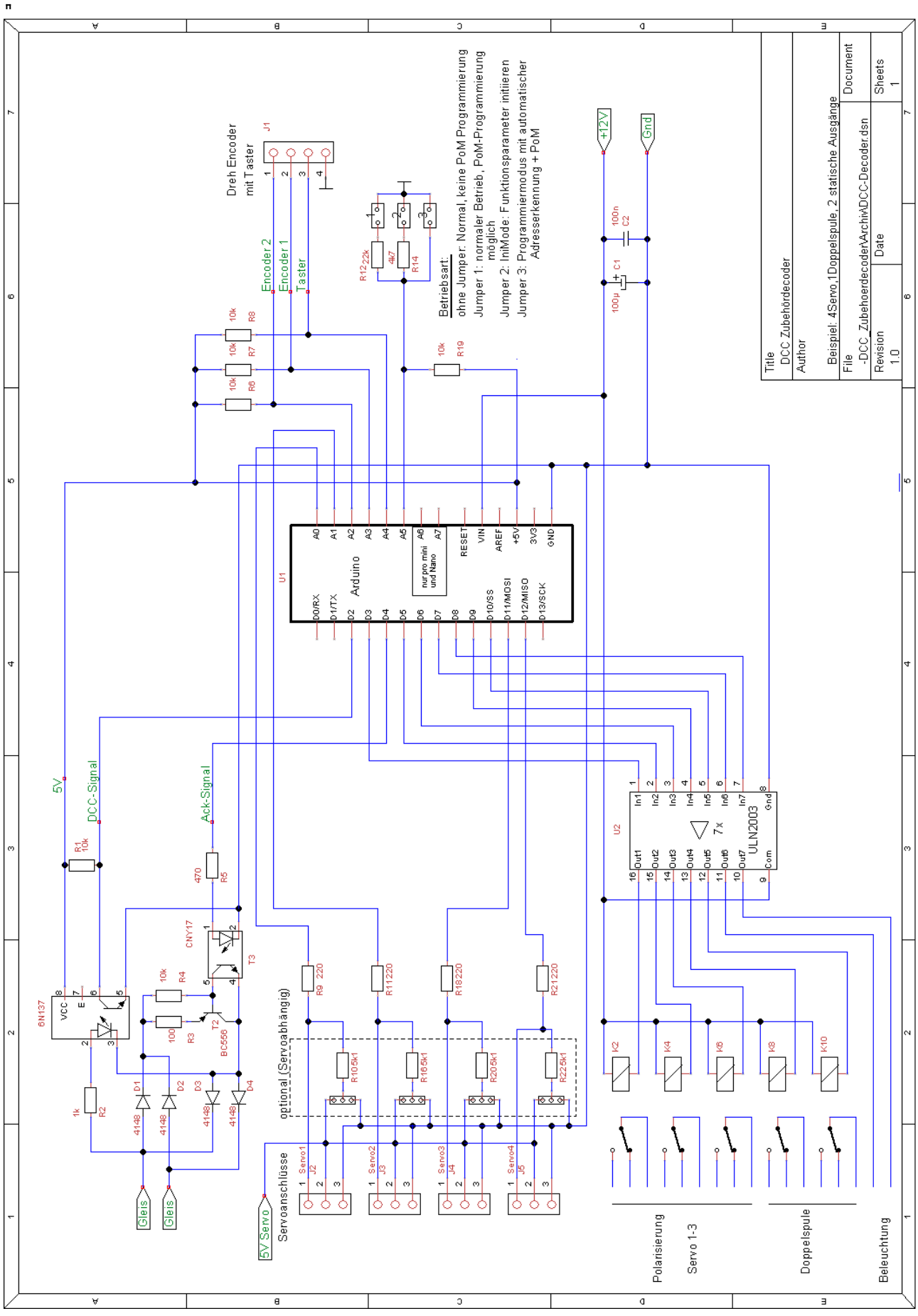
Zugehörige Konfiguration aus der .h-Datei:

```
const byte iniTyp[]   = {   FSERVO,   FSERVO,   FSERVO,   FSERVO,  FSTATIC,  FSTATIC,    FCOIL };
const byte out1Pins[] = {       A0,       A1,       11,       12,        7,        8,       10 };
const byte out2Pins[] = {        3,        5,       NC,       NC,        6,       NC,        9 };

const byte iniFmode[] = { SAUTOOFF, SAUTOOFF, SAUTOOFF,        0,  BLKMODE,        0, CAUTOOFF };
const byte iniPar1[]  = {        0,        0,        0,        0,       50,        0,       50 };
const byte iniPar2[]  = {      180,      180,      180,      180,       50,        0,       50 };
const byte iniPar3[]  = {        8,        8,        8,        0,      100,        0,        0 };
```

Die zugehörige DCC_Zubehoerdecoder.h findet sich im Verzeichnis ‚examples' unter den Namen ‚DCC_Zubehoerdecoder-Bsp1.h'

Schaltbild:


Dreh Encoder
mit Taster
Encoder 2
Encoder 1
Taster
Betriebsart:
ohne Jumper: Normal, keine PoM Programmierung
Jumper 1: normaler Betrieb, PoM-Programmierung möglich
Jumper 2: IniMode: Funktionsparameter initiieren
Jumper 3: Programmiermodus mit automatischer Adresserkennung + PoM
+12V
Gnd
Arduino
nur pro mini und Nano
ULN2003
7x
5V
DCC-Signal
Ack-Signal
optional (Servoabhängig)
5V Servo
Servoanschlüsse
Gleis
Gleis
Polarisierung
Servo 1-3
Doppelspule
Beleuchtung
Title
DCC Zubehördecoder
Author
Beispiel: 4Servo,1Doppelspule, 2 statische Ausgänge
File
-DCC_ZubehoerdecoderArchiv\DCC-Decoder.dsn
Revision
1.0
Date
Document
Sheets
1

Beispiel-Decoder (LocoNet)

Beispiel einer Decoderschaltung für 4 Servos (3 mit Polarisierung), 1 Doppelspulenantrieb und 2 statischen Ausgängen.

Zugehörige Konfiguration aus der .h-Datei:

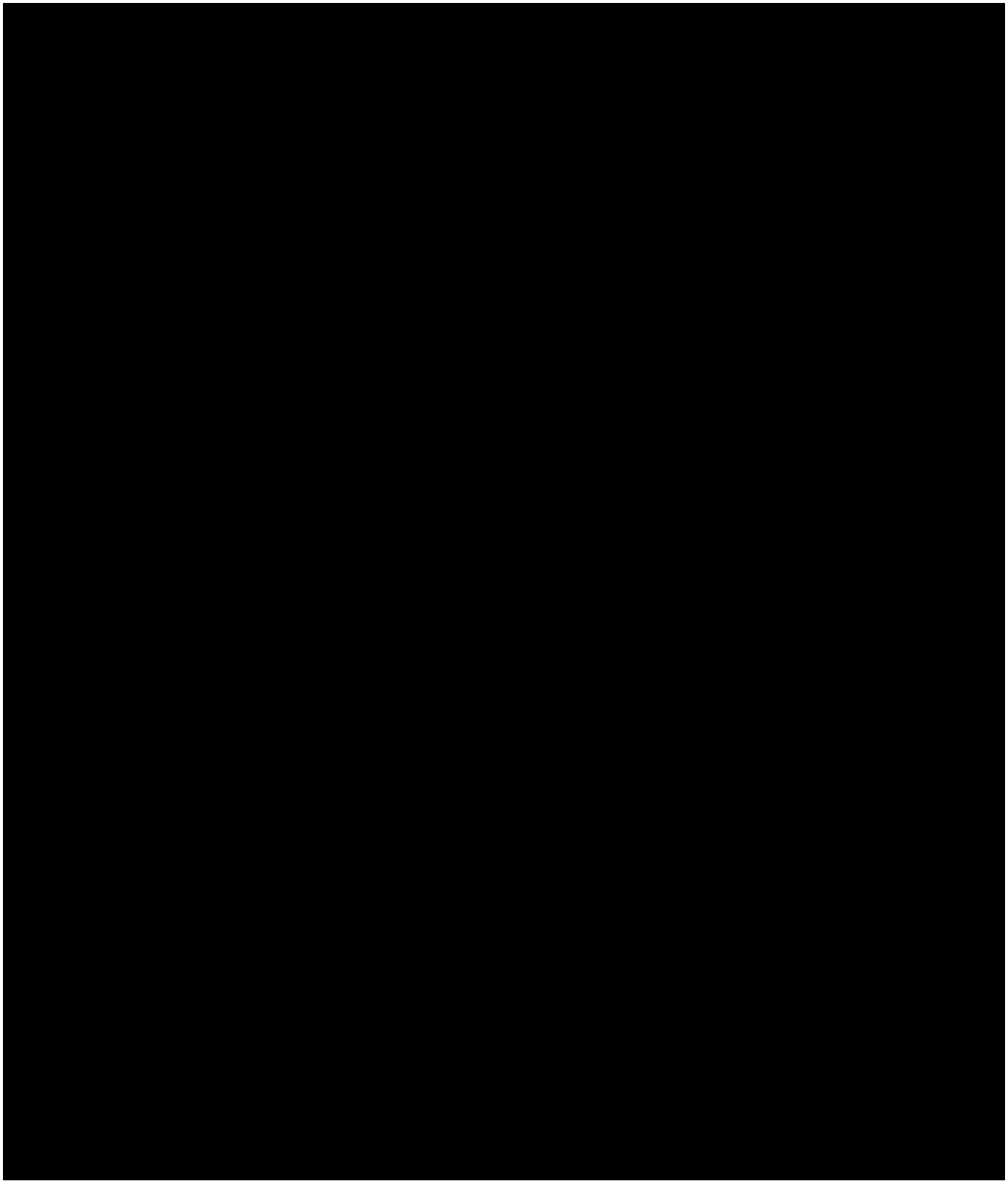
```
const byte iniTyp[]   = {   FSERVO,   FSERVO,   FSERVO,   FSERVO,  FSTATIC,  FSTATIC,    FCOIL };
const byte out1Pins[] = {        1,        0,        3,        5,       16,       14,       10 };
const byte out2Pins[] = {        6,        7,        8,       NC,       NC,       15,        9 };

const byte iniFmode[] = { SAUTOOFF, SAUTOOFF, SAUTOOFF,        0,  BLKMODE,        0, CAUTOOFF };
const byte iniPar1[]  = {        0,        0,        0,        0,       50,        0,       50 };
const byte iniPar2[]  = {      180,      180,      180,      180,       50,        0,       50 };
const byte iniPar3[]  = {        8,        8,        8,        0,      100,        0,        0 };
```

Die zugehörige DCC_Zubehoerdecoder.h findet sich im Verzeichnis ‚examples' unter den Namen ‚DCC_Zubehoerdecoder-Bsp2.h'

In der Datei Interface.h muss das LocoNet Interface aktiviert werden.

Schaltbild:



Beispieldecoder für Lichtsignale:

Beispiel einer Decoderschaltung für 5 3-begriffige Lichtsignale (HP0, HP1, HP2). Bei diesem Beispiel wird auf einen Rückkanal zum Auslesen von CV-Werten verzichtet, da bei Lichtsignalen normalerweise keine Konfigurationswerte verändert werden müssen. Alles wird fest in der .h Datei eingestellt. Das Beispiel ist für einen Arduino Nano ausgelegt. Im Verzeichnis ‚examples' ist die komplette Datei unter dem Namen DCC_Zubehoerdecoder-LS-Nano.h zu finden.

Auszug aus der .h-Datei:

```
const byte iniTyp[]   = { FSIGNAL2, FSIGNAL0, FSIGNAL2, FSIGNAL0, FSIGNAL2, FSIGNAL0, FSIGNAL2, FSIGNAL0,  FSIGNAL2, FSIGNAL0  };
const byte out1Pins[] = {        A0,       NC,       A3,       NC,        3,       NC,        6,       NC,         9,       NC  };
const byte out2Pins[] = {        A1,       NC,       A4,       NC,        4,       NC,        7,       NC,        10,       NC  };
const byte out3Pins[] = {        A2,       NC,       A5,       NC,        5,       NC,        8,       NC,        11,       NC  };

const byte iniFmode[] = {         0, 0b000100,        0, 0b000100,        0, 0b000100,        0, 0b000100,         0, 0b000100  };
const byte iniPar1[]  = { 0b001001, 0b110001, 0b001001, 0b110001, 0b001001, 0b110001, 0b001001, 0b110001,  0b001001, 0b110001  };
const byte iniPar2[]  = { 0b100010, 0b100110, 0b100010, 0b100110, 0b100010, 0b100110, 0b100010, 0b100110,  0b100010, 0b100110  };
const byte iniPar3[]  = {         0,        0,        0,        0,        0,        0,        0,        0,         0,        0  };
const byte iniPar4[]  = { 0b000101,        0, 0b000101,        0, 0b000101,        0, 0b000101,        0,  0b000101,        0  };
```

Die Konfiguration ist auch für 4-begriffige Signale (HP00, HP1, HP2, Sh1/Hp0) geeignet. Hierfür werden allerdings 5 Ausgänge benötigt, weshalb nur weniger Signale angesteuert werden können. Für diesen Zweck müssen ausschließlich die Arduino-Pins anders zugeordnet werden. Die Konfigurationsparameter iniFmode … iniPar4 müssen hierfür nicht angepasst werden.

Zuordnung der Ausgangspins:

	FSIGNAL2,	FSIGNAL0,	
Rote Led (HP0) →	A0,	NC,	← 2. rote Led (HP00)
Grüne Led →	A1,	NC,	← weisse Leds
Gelbe Led →	A2,	NC,	← Relais

Das Relais kann zu Zugbeeinflussung verwendet werden. Es ist nur im Zustand HP0 / HP00 abgefallen. In allen anderen Signalzuständen ist es angezogen.

Weitere Signaltypen
Sollen auch andere Signaltypen (Blocksignale, Vorsignale, Signale mit Vorsignal am gleichen Mast, Sperrsignale) angesteuert werden, so sind weitere Anpassungen notwendig. Dafür ist pro Signal immer ein Block aus 1-3 Spalten notwendig. Die folgenden beispielhaften Blöcke können für die jeweiligen Signaltypen direkt übernommen werden, lediglich die Ausgabepins müssen angepasst werden.
DCC-Adr in den Tabellen bezieht sich immer auf die Addresse, die sich aus der Spaltenposition in der Tabelle ergibt. (Ausnahme: DCC-Adresse des Hauptsignals, dass das Vorsignal ankündigt)

Einfaches Blocksignal		
iniTyp	FSIGNAL2	
out1Pins	5	← Pinnr rote Led
out2Pins	6	← Pinnr grüne Led
out3Pins	NC	← Pinnr Relais
iniFmode	0b100	← hard/softUmschaltung
iniPar1	0b001	← Leds für HP0 (DCC-Addr 0/rot)
iniPar2	0b110	← Leds für HP1 (Dcc-Addr 1/grün)
iniPar3	0	← Tabellenindex des Vorsignals am Mast(opt, 1...max)
iniPar4	0b01	← Vorsignaldunkeltastung (opt)

Ausfahrsignal (4-Begriffig) – benötigt 2 aufeinanderfolgende Spalten in der Tabelle			
	FSIGNAL2	FSIGNAL0	
Pinnr rote Led →	5	8	← Pinnr rote Led
Pinnr grüne Led →	6	9	← Pinnr weisse Leds
Pinnr gelbe Led →	7	NC	← Relais
hard/softUmschaltung →	0b000	0b100	← hard/softUmschaltung
(DCC-Addr 0/rot) Leds für HP00 →	0b001001	0b110001	← Leds für HP0/Sh1 (DCC-Addr 0/rot)
(Dcc-Addr 1/grün) Leds für HP1 →	0b100010	0b100110	← Leds für HP2 (Dcc-Addr 1/grün)
Tabellenindex des Vorsignals am Mast(opt, 1...max; 0= kein Vorsig.) →	0	0	
Vorsignaldunkeltastung (opt) →	0b0101	0	

Befindet sich am Mast auch ein Vorsignal, dass bei HP0 dunkelgetastet werden soll, so muss dieses Vorsignal mit dem gleichen Decoder geschaltet werden. D.h. dieses Signal muss sich auch in der Tabelle befinden. Der Position des Vorsignals in der Tabelle (ab 1…) muss beim Hauptsignal eingetragen werden.

Einfaches Vorsignal		
iniTyp	FVORSIG	
out1Pins	5	← Pinnr gelbe Leds
out2Pins	6	← Pinnr grüne Leds
out3Pins	NC	← nicht belegt
iniFmode	0b000	← hard/softUmschaltung
iniPar1	0b001	← Leds für Vr0 (DCC-Addr 0/rot)
iniPar2	0b010	← Leds für Vr1 (DCC-Addr 1/grün)
iniPar3	23	← low Byte der Hauptsignaladresse
iniPar4	0	← high Byte der Hauptsignaladresse

Bei Vorsignalen kann zusätzlich auch die DCC-Adresse des Signals eingetragen werden, das von diesem Vorsignal angekündigt wird. Damit schaltet das Vorsignal synchron mit dem Hauptsignal. Das Hauptsignal muss nicht mit dem gleichen Decoder geschaltet werden.
Auf die Adresse, die sich durch die Position des Vorsignals in der Tabelle ergibt, reagiert das Vorsignal immer.

Vorsignal (3-Begriffig) – benötigt 2 aufeinanderfolgende Spalten in der Tabelle			
	FVORSIG	FSIGNAL0	
Pinnr gelbe Led →	5	7	← Pinnr gelbe Led
Pinnr grüne Led →	6	8	← Pinnr grüne Led
	NC	NC	
hard/softUmschaltung →	0b00000	0b00000	
(DCC-Addr 0/rot) Leds für Vr0 →	0b01001	0b01010	← Leds für Vr2 (DCC-Addr 0/rot)
(Dcc-Addr 1/grün) Leds für Vr1 →	0b10010	255	Ungenutzt (Dcc-Addr 1/grün)
low Byte Adr. Hauptsig →	55	0	
high Byte Adr. Hauptsig →	0	0	

Schaltbild des Lichtsignaldecoders (examples\DCC_Zubehoerdecoder-LS-Nano.h):