Jörg Pleumann

Arduino steuert Märklin-Modellbahn

Moderne digitale Steuerungen für Modellbahnen von Märklin verwenden intern den CAN-Bus zur Kommunikation zwischen einzelnen Endgeräten. Mit einem Arduino und einem passenden Shield kann man diese Welt für interessante Experimente öffnen. Einer eigenen Steuerung steht damit nichts im Weg.

Kurzinfo

Zeitaufwand:
2 Stunden

Kosten:
ab 25 Euro

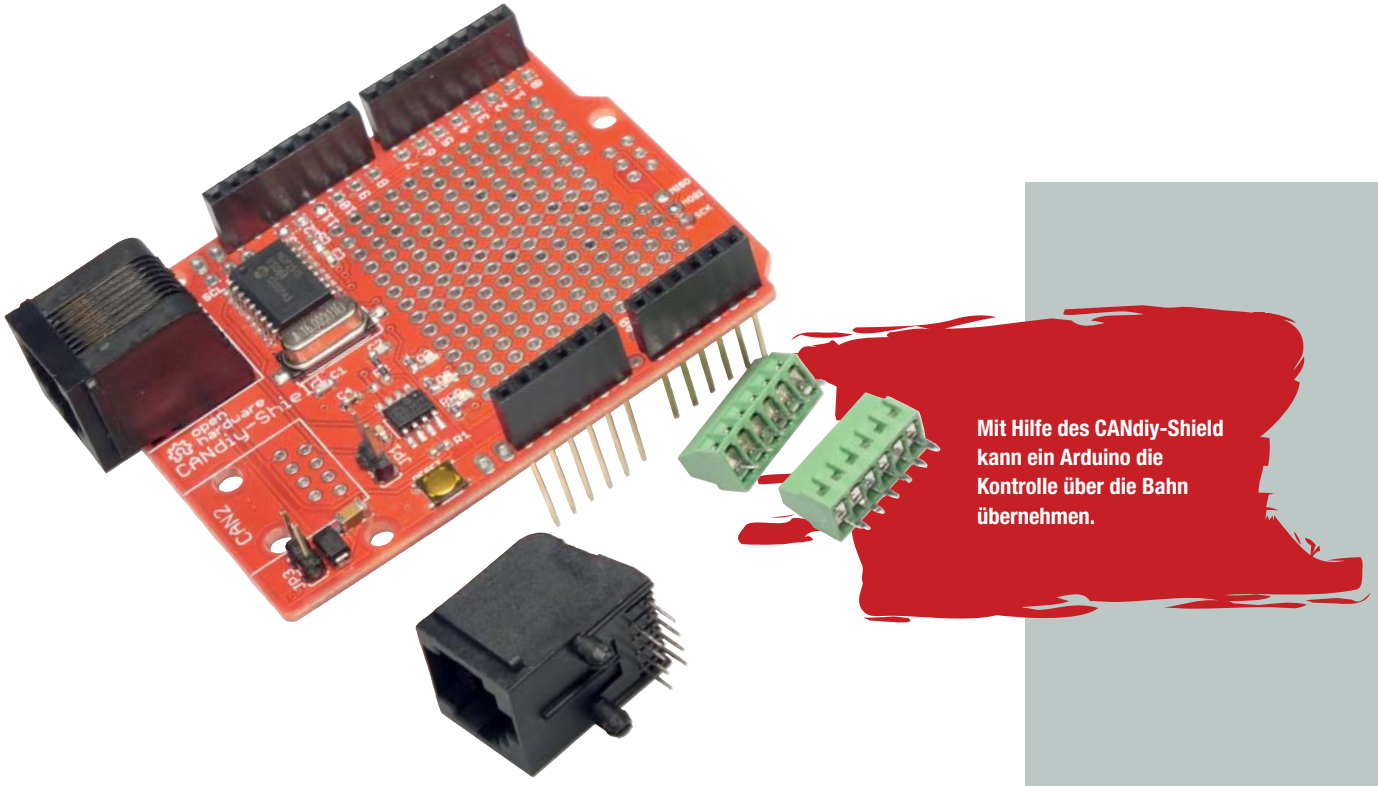
Programmieren:
Arduino SDK

Löten:
Kabel löten

Elektronik:
Grundkenntnisse

Schwierigkeitsgrad

leicht — schwer



Mit Hilfe des CANdiy-Shield kann ein Arduino die Kontrolle über die Bahn übernehmen.

Bis in die frühen 1980er Jahre waren Modellbahnen ein rein analoges Hobby. Das Fahrgerät – also ein schwergewichtiger Transformator mit Drehregler – legte beispielsweise eine Spannung von 0 bis 12 Volt ans Gleis, die direkt an den Motor der auf dem Gleis befindlichen Lokomotive weitergegeben wurde. Je nach System handelte es sich um eine Gleich- oder Wechselspannung, was zwar im Detail Konsequenzen hatte (und zu Grabenkriegen zwischen verschiedenen Lagern führte, die offenbar in keinem Hobby fehlen dürfen), aber am Gesamtbild nichts änderte: Die Geschwindigkeit der Lokomotive war direkt proportional zur angelegten Spannung. Hatte die Lokomotive eine Beleuchtung, so leuchtete das Licht bei hoher Geschwindigkeit heller. Wollte man mehr als eine Lokomotive gleichzeitig fahren lassen, brauchte man voneinander isolierte Gleisabschnitte und mehrere Transformatoren. Ferngesteuerte Weichen und Signale wurden mit zentralen Stellpulten verbunden. Der Kabelsalat war vorprogrammiert!

Zutaten

- Anschlussbox 60113 oder 60112 für digital Modellbahn von Märklin, Trix oder LGB
- Arduino Uno, Leonardo oder kompatibel
- CANDiy-Shield
- CAT5-Patchkabel
- spezieller 10-poliger Mini-DIN-Stecker
- (optional) Microchip MCP 23S08 oder MCP 23S17
- (optional) 2 × Pfostenleiste 6 Pins 2,54 mm
- (optional) S88-Decoder
- (optional) Bluetooth Modul
- (optional) Android-Telefon oder -Tablet

Auch wenn es im 21. Jahrhundert noch möglich ist, Modellbahnen nach diesem klassischen Schema zu kaufen und zu betreiben, dürfte die Mehrzahl der heutigen Modellbahnen digitalisiert sein. Zwar gibt es nach wie vor Unterschiede im Detail, aber diese sind geringer als früher. Zum Beispiel existiert die Trennung in „Gleichstrombahner“ und „Wechselstrombahner“ nicht mehr (die verschiedenen Lager müssen sich andere Themen suchen, über die sie streiten können). Stattdessen liegt am Gleis eine pulsierende Gleichspannung von 18 V an. Diese dient sowohl als Versorgungsspannung für die Lokomotiven als auch als Datensignal, über das die Lokomotiven mit Kommandos versorgt werden. Das hat eine Reihe von Konsequenzen: Die Geschwindigkeit der Lokomotive hängt nicht mehr direkt von der Spannung ab, und ihre Beleuchtung hat nun eine konstante Helligkeit. Auch ist eine Isolation von Gleisabschnitten nicht mehr unbedingt nötig, um mehrere Züge gleichzeitig fahren zu lassen – besser noch, sogar Weichen und Signale können mit an dem zentralen Datenbus hängen, den das Gleis nun bildet.

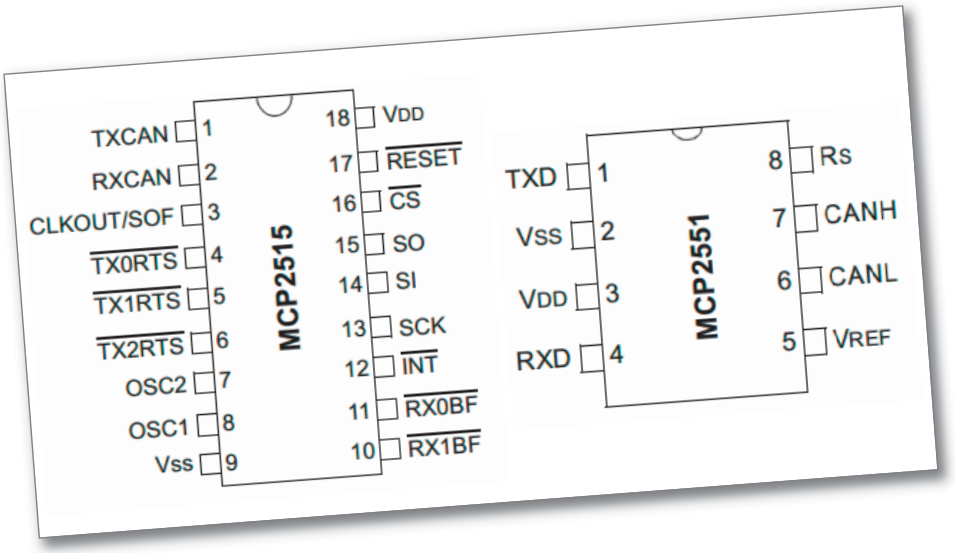
Allerlei Protokolle

Was ein bisschen nach Powerline-Ethernet klingt, hat tatsächlich einige Gemeinsamkeiten mit einem Rechnernetzwerk über die heimische Steckdose. Alle Teilnehmer, die an diesem Bus hängen, benötigen nun ein spezielles Stück Elektronik, einen sogenannten Decoder. Dieser übersetzt die Kommandos auf dem Gleis in konkrete Fahr- oder Schaltbefehle. Damit nicht alle Lokomotiven gleichzeitig losfahren, brauchen sie eine eindeutige Adresse, auf die der Decoder hört. Realisiert wird ein solcher Decoder üblicherweise durch einen Microcontroller, der auf einer daumennagelgroßen Platine in der Lok sitzt.

Für die Übertragung der Kommandos auf dem Gleis haben sich – je nach Hersteller – verschiedene Protokolle etabliert. Die größte Verbreitung haben das proprietäre Märklin-Motorola-Protokoll, kurz MM (meist mit der Versionsnummer 2 dahinter), und das aus dem amerikanischen Raum stammende Digital

Protokolle

Auch alte Märklin-Loks lassen sich digital aufrüsten. Dazu eignen sich unter anderem Multiprotokolldecoder für MM2 und DCC. Mit DCC hat man eine feine Fahrstufensteuerung. Daneben gibt es noch MFX. Wer seine Bahn bereits auf eines dieser Protokolle eingestellt hat, muss die Adresse in den Sketches anpassen. Beim Rausfinden der Adresse hilft der Sketch „Address“ unter Misc.



Der Stand-alone-CAN-Bus-Controller MCP2515 unterstützt das Protokoll V2.0B. Der Transceiver MCP2551 sorgt für die elektrische Anbindung.

TIPP

Die Adressierung ist eine prima Fehlerquelle. Wenn man die falsche Basisadresse wählt oder diese ganz vergisst, bewegt sich sicher nicht die gewünschte Lok, möglicherweise aber eine ganz andere. Wenn ein Sketch mal nicht funktioniert, lohnt es sich daher, die Adressen zu überprüfen. Die Herstellerdokumentation der Züge enthält ebenfalls meist einen Hinweis.

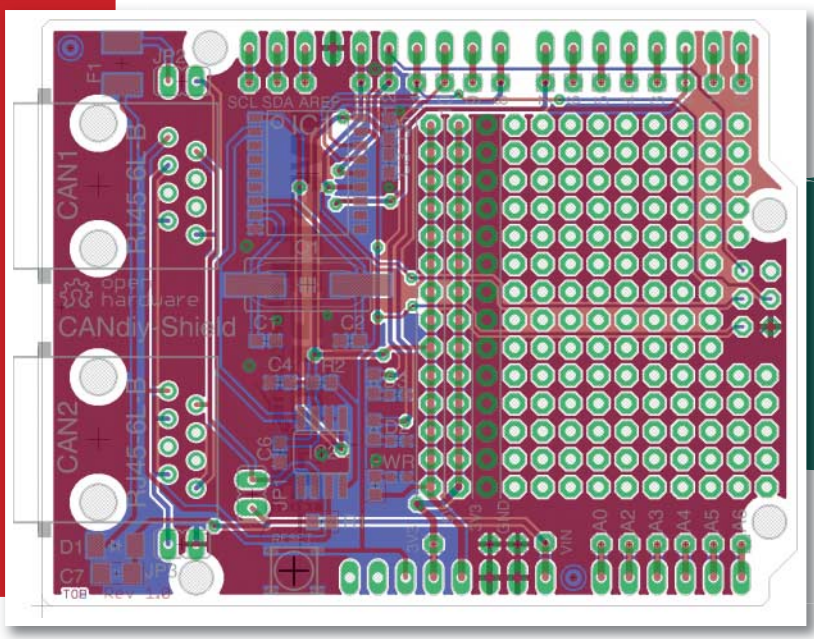
Command Control, kurz DCC. Moderne Systeme sprechen oft beide Protokolle. Daneben gibt es noch MFX, das automatisch für den Nutzer eine Adresse vergibt, sobald eine Lok zum ersten Mal auf dem Gleis steht. Erzeugt wird das Gleissignal von einer Komponente, die bei Märklin Gleisformatprozessor (GFP) genannt wird und eine entsprechend zentrale Rolle in der Anlage einnimmt. Interessanterweise schickt dieser GFP viele der Kommandos nicht einmalig, sondern wiederholt sie ständig, bis Ihre Daten überholt sind. Auf dem Gleis ist also ständig eine ganze Menge los! Dadurch nimmt aber eine Lokomotive, die dank verschmutzter Schienen kurz stromlos geworden ist, ihre alte Geschwindigkeit auf, sobald sie das Gleissignal wieder „empfängt".

Bei Märklin sitzt der GFP entweder in der relativ teuren Central Station 2, die praktisch ein spezialisierter Linux-Rechner ist, oder in der deutlich günstigeren Digital-Anschlussbox, die zusammen mit einer Mobile Station 2 als Handgerät in vielen Startpackungen enthalten ist – und wahrscheinlich auch zu Weihnachten unter dem einen oder anderen Baum lag. Damit haben auch wir unser Projekt getestet.

Mobile Station 2 und Anschlussbox nutzen den aus dem Auto bekannten CAN-Bus (siehe Kasten S. 60) zur Kommunikation miteinander. Das darauf zwischen Station und Lokomotive eingesetzte Anwendungsprotokoll ist von Märklin teilweise offen gelegt (siehe S. 61). Praktischerweise hat die Anschlussbox eine zweite unbenutzte Buchse, an die man weitere Steuerungen anschließen kann.

Hardware

Die Hardware, mit der wir unsere Modellbahn steuern wollen, setzt auf dem Arduino Uno auf. Der Kern der Schaltung ist sehr einfach (siehe Schaltbild auf S. 62). Im Wesentlichen werden drei relativ günstige ICs und



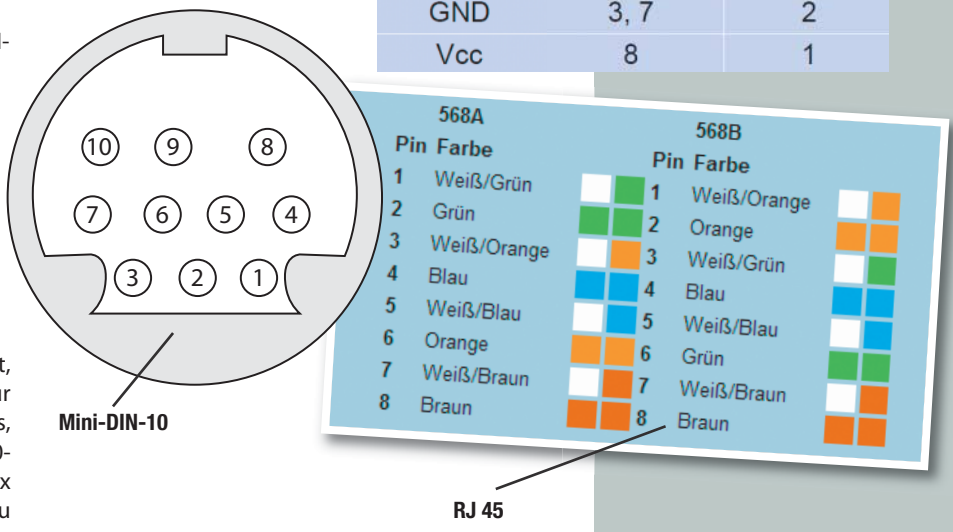
Das c't-CANdiy-Shield lässt sich dank der Proto-Area leicht erweitern.

ein paar Kleinteile benötigt: Ein CAN-Controller MCP2515, der sich um das CAN-Protokoll kümmert, also Pakete senden und empfangen kann. Hinzu kommt ein CAN-Transceiver MCP2551, der die 5V-Signale des Controllers in die differenziellen Signale übersetzt, die auf dem Bus übertragen werden.

Arduino und Controller sprechen über die SPI-Schnittstelle miteinander. Auf einem Breadboard lässt sich schnell ein Prototyp aufbauen. Wer es etwas professioneller oder langlebiger mag, für den hält c't-Hacks in Zusammenarbeit mit Watterott ein günstiges Shield bereit, das auch über dieses Projekt hinaus einsetzbar ist und im nächsten Heft erneut Verwendung finden wird. Eine weitere Alternative ist der CAN-Shield von Sparkfun, der über die einschlägigen Versender auch in Deutschland erhältlich ist.

Sind Breadboard oder CAN-Shield fertig bestückt, dann beschränkt sich die restliche Bastelarbeit für dieses Projekt auf das Konfektionieren eines Kabels, das unsere Schaltung mit der etwas untypischen 10-poligen Mini-DIN-Buchse der Märklin-Anschlussbox verbindet. Die Belegung ist in Bild rechts oben zu sehen. Bei unserem CAN-Shield zeigt sich schnell der Vorteil der verwendeten RJ45-Buchse: Ein passendes Patchkabel kostet im Internet etwa einen Euro, und einer der beiden Stecker ist schnell abgetrennt, um die einzelnen Adern mit dem 10-poligen Stecker zu verlöten.

Signal	RJ45	Mini-DIN
CAN_L	1	8
CAN_H	2	4
GND	3, 7	2
Vcc	8	1



Die Belegung der jeweiligen Stecker am Adapterkabel

Software

Auf der Software-Seite könnten wir zum Beispiel mit der exzellenten CAN-Bibliothek von Fabian Greif arbeiten, die unter anderem den MCP2515 unterstützt. Allerdings würden wir dann mit sämtlichen Bits und Bytes des Märklin-Protokolls konfrontiert. Das ist zwar durchaus spannend, aber nicht jedermanns Sache. Wer sich dafür interessiert, der findet im Kasten auf S. 61 eine Übersicht. Alle anderen laden sich stattdessen direkt die Bibliothek „Railuino" aus dem Netz (siehe Link am Ende). Diese kapselt die grundlegende Kommunikation mit dem CAN-Bus und stellt sowohl ein einfach zu nutzendes API als auch zahlreiche Beispiele zu Verfügung. Die Bibliothek wird – wie üblich – im „libraries"-Ordner von Arduino abgelegt und taucht nach einem Neustart der Entwicklungsumgebung im Menü auf.

Railuino-Sketches besitzen verschiedene wiederkehrende Elemente, die im einfachen Beispiel in Listing 1 zu sehen sind. Grundvoraussetzung ist jeweils die Einbindung der Bibliothek über `#include <Railuino.h>`.

Im Kopf des Sketches definiert man verschiedene Konstanten, die im weiteren Verlauf öfter genutzt werden. Dazu zählen zum Beispiel Adressen für Lokomotiven sowie Weichen und Signale. Bei der Adressierung ist es wichtig, das verwendete Protokoll des Decoders zu kennen. Die komplette Adresse ergibt sich aus einer Basisadresse des Protokolls und der eigentlichen Adresse. Die beiden werden einfach addiert. Bei einem frisch ausgepackten Märklin-Starterset „Güterzug" ist standardmäßig das Märklin-Protokoll eingestellt und die Lok reagiert auf Adresse 78. Im Sketch ist die Lok-Adresse als `ADDR_MM2 + 78` eingetragen. Sofern man DCC als Protokoll einsetzt, sind die Konstanten auf `ADDR_DCC + xy` anzupassen. Zubehörteile verwenden eine eigene Basisadresse.

Anschließend wird ein TrackController-Objekt erzeugt und initialisiert: `TrackController ctrl(0xdf24, TRUE);`. Es kapselt die Funktionen der Anschlussbox und kümmert sich selbstständig um die Übersetzung von Daten in CAN-Botschaften und das Auswerten der Antworten. Bei der Initialisierung können zwei Parameter übergeben werden: Der erste Parameter gibt unserer Station ein eindeutiges Adress-Hash in Verbund der Märklin-Geräte. Dieser ist jedoch nicht mit den Adressen von MM2 oder DCC zu verwechseln. Der Hash-Wert im Beispiel ist willkürlich gewählt. Die Regeln für die Bildung des Hashes sind in der Märklin-Dokumentation zu finden. Bei einem Wert von 0 wählt der TrackController automatisch einen passenden Hash.

Listing 1

```
#include <Railuino.h>
const word    LOCO  = ADDR_MM2 + 78;
const word    SPEED = 100;
const word    TIME  = 2000;
TrackController ctrl(0xdf24, true);
void setup() {
  Serial.begin(9600);
  ctrl.begin();
  ctrl.setPower(true);
  ctrl.setLocoFunction(LOCO, 0, 1);
}
void loop() {
  ctrl.setLocoSpeed(LOCO, SPEED);
  delay(TIME);
  ctrl.setLocoSpeed(LOCO, 0);
  delay(TIME);
  ctrl.toggleLocoDirection(LOCO);
}
```

TIPP

Damit die Arduino-Sketches fehlerfrei kompilieren, gehört die Railuino-Bibliothek an die richtige Stelle. Dazu legt man im Ordner der Arduino-Installation unter libraries den Ordner Railuino an. Dorthin kopiert man aus dem Archiv alle Ordner und Dateien, die sich im Ordner src finden.

Die Anschlussbox hat eine unbenutzte Buchse, an die man eigene Steuerungen anschließen kann. Im Bild ist noch ein Prototyp des Shields mit Bluetooth zu sehen.

Der zweite Parameter versetzt den Controller in den Debug-Modus, in dem er sämtliche ein- und ausgehenden CAN-Botschaften auf der seriellen Schnittstelle im Klartext ausgibt. Mit dem seriellen Monitor der Arduino-Konsole lässt sich so beobachten, was gerade auf dem Bus passiert. Nach der Initialisierung wird innerhalb von `setup()` der Controller über `begin()` gestartet und die Anschlussbox mit `setPower()` angewiesen, Spannung an das Gleis zu legen. Zudem schaltet `ctrl.setLocoFunction(LOCO, 0, 1);` das Licht unserer Lok an. Die einmalige Vorbereitung ist damit erledigt.

In der zyklisch durchlaufenen Schleife `loop()` werden nun zwei Kommandos an unsere Lokomotive geschickt: Mittels `setLocoSpeed()` lassen wir die Lokomotive bis zu einer vorgegebenen Geschwindigkeit beschleunigen. Mögliche Werte liegen hier im Bereich 0 bis 1023. Nach kurzer Fahrzeit bremsen wir sie mit dem gleichen Kommando wieder ab. Anschließend ändern wir mittels `toggleLocoDirection()` die Richtung. Wird dieser Sketch auf den Arduino geladen, sollte unsere Lokomotive brav ein Stückchen hin- und herfahren. Damit sind die Grundlagen für eine Steuerung von Lokomotiven gelegt. Ab hier lohnt es sich, entweder mit den Möglichkeiten von TrackController zu experimentieren oder die verschiedenen Beispiele auszuprobieren, die mit der Bibliothek kommen.

TIPP

Achtung: Die fertigen Gleise gehen üblicherweise davon aus, das ein einzelnes „Rückmeldekabel" existiert, das im Kontaktfall mit der Masse des Gleises verbunden wird. Sollte der Strom mal „falsch abfließen", werden die 18 V des Gleises einen Arduino schnell in den Elektronikhimmel befördern. Wer auf Nummer sicher gehen will, der verwendet eine galvanisch getrennte Lösung. Die Schaltgleise von Märklin lassen sich sehr einfach galvanisch trennen.

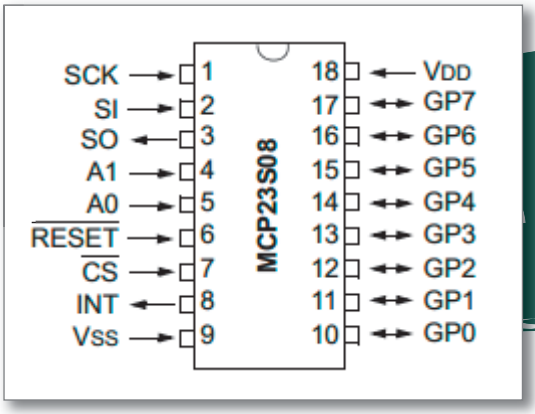
Rückmeldung

Wenn man eine (teil-)autonome Steuerung der Modellbahn bauen möchte, benötigt man zusätzlich zur Steuerung noch ein weiteres Element: Man muss wissen, wo ein Zug gerade ist. Dies wird über Rückmeldekontakte realisiert, die an verschiedenen Stellen der Anlage verbaut werden und ein Signal auslösen, wenn sie von einem Zug passiert werden. Üblich sind hier Magnetkontakte oder Lichtschranken. Märklin bietet auch fertige Gleise mit speziellen Kontakten oder Schaltern an.

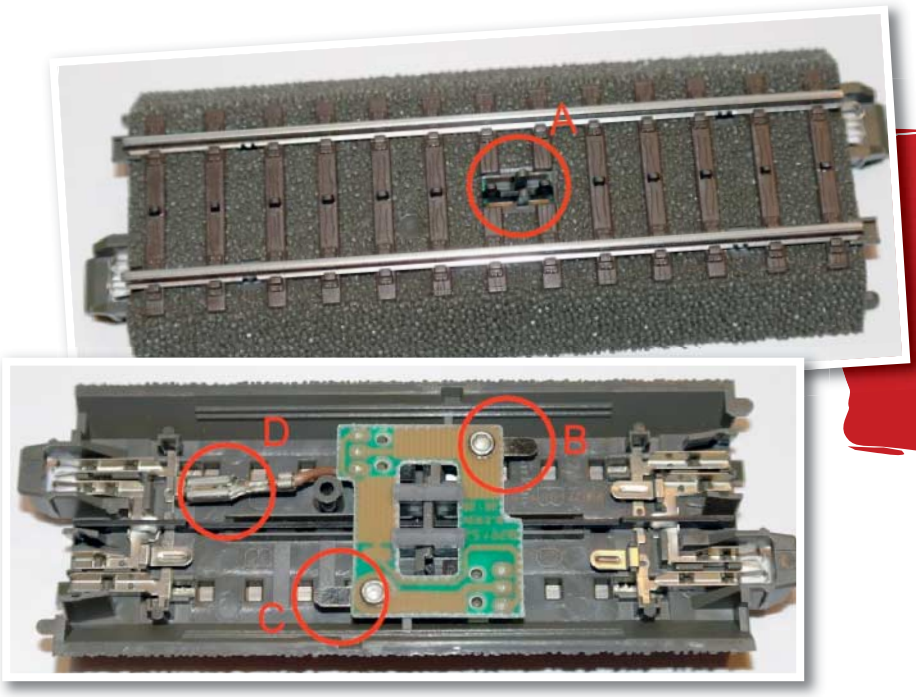
Wie wertet man nun diese Kontakte aus? Im einfachsten Fall kann man direkt die IO-Pins des Arduino benutzen, allerdings sind davon nicht endlos viele vorhanden. Ein zweiter Arduino oder ein Arduino Mega kann für kleine Anlagen eine Lösung sein, skaliert jedoch auch nicht beliebig. Zudem besteht das Problem, dass man die Eingänge quasi permanent beobachten muss – sonst kann ein schneller Zug möglicherweise den Kontakt passieren, ohne bemerkt zu werden. Der Arduino wäre in diesem Fall völlig ausgelastet und hätte keine Zeit mehr, den CAN-Bus zu bedienen.

Eine sehr einfache Lösung lässt sich auf Basis der IO-Extender MCP 23S08 oder MCP 23S17 konstruieren. Sie stellen 8 beziehungsweise 16 bidirektionale, gepufferte Eingänge/Ausgänge bereit, die Änderungen sogar auf Wunsch per Interrupt mitteilen können. Beide ICs werden – genau wie der CAN Controller – per SPI an den Arduino angebunden. Da sie zudem über zwei Adressleitungen verfügen, lassen sich vier dieser Extender an einem gemeinsamen Chip Select betrieben. Man kommt also problemlos (und für etwa fünf Euro) auf 64 parallele Ein-/Ausgänge. Das dürfte für die meisten Anlagen reichen.

Die Railuino-Bibliothek enthält bereits Unterstützung für die beiden Bausteine, was die Programmierung stark vereinfacht. Zunächst wird mit `TrackReporter IOXrpt (1)` ein TrackReporter mit einem IO-Expander initialisiert (D6 ist Chip Select, D3 ist Interrupt, Adresse 0). Jeder folgende Aufruf von `refresh()` liest nun die Werte aller Eingänge in den Speicher und setzt gleichzeitig die Puffer im Baustein zurück. Über `getValue()` kann anschließend der gelesene Wert eines einzelnen Kontakts erfragt werden. Mit diesem Rüstzeug ausgestattet lässt sich unser Beispielsketch aus dem vorangehenden Abschnitt leicht erweitern,



Die I/O-Extender melden über den Interrupt-Ausgang, wenn ein Eingangssignal seinen Zustand geändert hat. Damit spart man sich das Software-Polling auf dem Arduino.



A: Mikroschalter-Paar, wird in beiden Richtungen vom Schleifer der Lok ausgelöst

B: Kontakt für Linksfahrt

C: Kontakt für Rechtsfahrt

D: Gemeinsame Masse. Vom Gleis trennen, längeres Kabel anlöten und zur Arduino-Masse führen (gut isolieren!)

siehe Listing 2. Wenn alles richtig verdrahtet ist, sollte die Lokomotive jetzt zwischen zwei Kontakten hin- und herfahren.

Die große weite Welt

Auf der Basis der bislang beschriebenen Mittel ist es möglich, Teile einer Modellbahn – etwa den Schattenbahnhof – zu automatisieren, eigene Steuergeräte zu entwickeln oder sogar eine komplette Bahn automatisch ablaufen zu lassen. Irgendwann kommt möglicherweise der Wunsch auf, die Modellbahn mit einer komfortablen graphischen Steuerung via PC, Mobiltelefon oder Tablet auszustatten. Auch das ist möglich. Auf der mobilen Seite hat Android gegenüber der Apple-Welt ein paar Vorteile durch seine Offenheit.

Für den Arduino existieren verschiedene Shields, die Unterstützung für TCP/IP via Ethernet oder WLAN mitbringen. Damit wird der Arduino zu einem gewöhnlichen Netzwerkgerät und kann als solches mit anderen Geräten im heimischen Netz oder im Internet kommunizieren. Eine Alternative ist das Accessory Development Kit (ADK), eine spezielle Va-

Listing 2

```
#include <Railuino.h>
const word    LOCO  = ADDR_MM2 + 78;
const word    SPEED = 100;
const word    TIME  = 2000;
TrackController ctrl(0xdf24, true);
TrackReporter rprt(1);
void setup() {
  Serial.begin(9600);
  ctrl.begin();
  ctrl.setPower(true);
  ctrl.setLocoFunction(LOCO, 0, 1);
}
void waitForContact(word index) {
  Serial.print("Waiting for contact ");
  Serial.println(index);

  rprt.refresh();
  while (!rprt.getValue(index)) {
    rprt.refresh();
  }

  Serial.println("Ok");
}
void drivePastContact(int dir, int contact) {
  ctrl.setLocoDirection(LOCO, dir);
  ctrl.setLocoSpeed(LOCO, SPEED);
  waitForContact(contact);
  Serial.println("V200 stop");
  ctrl.setLocoSpeed(LOCO, 0);
}
void loop() {
  drivePastContact(DIR_FORWARD, 1);
  delay(TIME);

  drivePastContact(DIR_REVERSE, 2);
  delay(TIME);
}
```

Aktuelle Version?

Um den vollen Funktionsumfang von Railuino zu unterstützen, benötigt die Anschlussbox unter Umständen ein Firmware-Upgrade. Ob das so ist, lässt sich an der Software-Version der Mobile Station 2 ablesen. Diese findet sich im Eintrag „Informationen“ bei den Einstellungen (Shift+Schraubenschlüssel).

Aktuell und empfohlen ist die Version 1.81. Damit wurden alle Railuino-Beispiele erfolgreich getestet. Auch mit der Version 1.5 sollten alle Beispiele laufen. Erst bei älteren Versionen wie 1.3 gibt es kleinere Einschränkungen.

Zum Aktualisieren auf die aktuelle Version muss die Mobile Station einmal zum Modellbahnhändler gebracht werden, damit sie dort ein Update bekommt, was normalerweise weder Zeit noch Geld kostet. Die Mobile Station aktualisiert anschließend zuhause die Anschlussbox.

S88

Eine Variante der Rückmeldung, die in der Märklin-Welt starke Verbreitung hat, ist der sogenannte S88-Bus. Dabei handelt es sich um einen Rückmeldebus, der mehrere kaskadierte Rückmeldemodule mit parallelen Eingängen seriell abfragt. Klingt kompliziert, ist im Prinzip aber nur ein großes Schieberegister. Arduino-Experten werden hier schnell das Standardbeispiel „ShiftIn“ aus der Arduino-Dokumentation wiedererkennen. Der S88-Bus ergänzt dieses im Prinzip um ein Flip Flop, das dafür sorgt, dass Eingaben bis zu ihrer Abfrage gespeichert werden. Der S88-Bus ist zwar nicht extrem schnell und schon gar nicht Interrupt-fähig, kann dafür aber mit nur sechs Leitungen eine große Anzahl von Modulen bedienen.

Module für den S88-Bus sind sowohl in fertiger Form als auch als Bausatz oder Schaltplan leicht verfügbar, sodass an dieser Stelle auf Details verzichtet und stattdessen ein Modul „von der Stange“ verwendet werden soll. Die Abbildung auf Seite 62 zeigt die Verbindung des Moduls mit unserem Arduino. Die Klasse TrackReporterS88 der Railuino-Bibliothek setzt es bereits um. Die Verwendung entspricht dem zuvor beschriebenen TrackReporter IOX. Einzig die Initialisierung sieht etwas anders aus: TrackReporterS88 rprt (1); für ein Modul.

Beispiel-Sketches

Der Artikel hat aus Platzgründen nur einen kleinen Teil der Möglichkeiten anreißen können, die sich ergeben, wenn man sich mittels Arduino und CAN Shield in die Märklin Modellbahn „hackt". Weitere Beispiele liefert die Bibliothek, die im Quellcode unter der GNU Lesser General Public License (LGPL) Version 2 verfügbar ist. Richtig spannend wird es natürlich erst, wenn man damit eine reale Anlage automatisiert steuert oder aber mit einem PC oder Mobilgerät verbindet, um dort eine komfortable grafische Steuerung zu realisieren oder vorhandene Steuerungsprogramme an Railuino anzupassen. Der Autor ist sehr gespannt, was die Leser alles aus dieser kleinen Bastelei machen.

riante von Arduino, die von Google entwickelt wurde. Das ADK erlaubt den Anschluss einer Arduino-Schaltung an ein Android-Gerät via USB. Es wurde in c't Hacks 02/12 vorgestellt. Beide Lösungen sind jedoch mehr oder minder kostspielig.

Es geht allerdings auch deutlich günstiger. Aktuelle Geräte mit Android ab 3.0 bieten meist Unterstützung für den USB-Hostmodus. Mit einem Adapterkabel, das nur wenige Euro kostet, lässt sich dann ein gewöhnlicher Arduino direkt anschließen. Die beiden Geräte kommunizieren seriell über das USB-Unterprotokoll Communication Device Class (CDC). Eine ebenso günstige, jedoch flexiblere Lösung stellt Bluetooth dar. Für etwa 10 Euro bekommt man inzwischen Bluetooth-Adapter aus China, die sich direkt an den Arduino anschließen lassen. Nach dem Pairing kommunizieren Smartphone und Arduino über eine serielle Bluetooth-Verbindung miteinander.

Die Railuino-Bibliothek enthält einige Beispiele für Sketches, mit denen USB- oder Bluetooth-Gateways in wenigen Zeilen Code realisiert werden können. Als Gegenstelle bietet sich zum Beispiel die Railuino App an, die aus dem Google Play Store heruntergeladen werden kann. (dab)

Links und Foren
www.ct.de/ch1301054

Bezugsquelle

Das fast fertige CANdiy-Shield in SMD-Ausführung kann beim Elektronik-Versender Watterott für 15 Euro unter der Artikelnummer 20110452 bezogen werden. Die Stiftleisten und RJ-45-Buchsen liegen lose bei und müssen selbst verlötet werden. Wenn nur eine Buchse montiert ist, passt das Shield noch in das Original-Arduino-Gehäuse. Der Bausatz zum Herstellen des Kabels hat die Artikelnummer 20110460 und kostet 4 Euro.

Der CAN-Bus

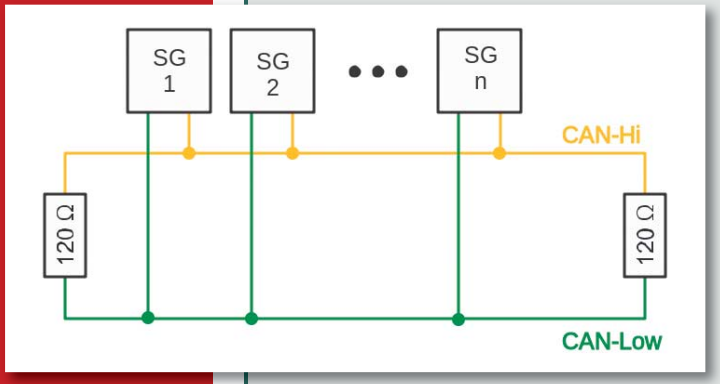
Der Controller Area Network Bus, kurz CAN-Bus, wurde in den 1980er Jahren von Bosch entwickelt, um dem zunehmenden Einsatz von elektronischen Steuergeräten im Auto und deren Vernetzung untereinander Herr zu werden. Es ist praktisch in allen modernen Fahrzeugen zu finden.

Da es sich um einen Bus handelt, sind die Teilnehmer direkt miteinander verbunden – jeder sieht also alle Nachrichten, und es kann dazu kommen, dass mehrere Teilnehmer gleichzeitig zu senden versuchen. Um dies zu lösen, wird zum einen eine Kennzeichnung der Nachrichten über IDs verwendet, zum anderen eine spezielle Art von Kollisionserkennung: Beginnen zwei Teilnehmer gleichzeitig mit dem Senden einer Nachricht, wird dies erkannt, und der Teilnehmer mit der höheren ID bricht seine Übertragung ab. Im Unterschied zur Kollisionserkennung beim alten Ethernet über Koaxialkabel müssen also nicht beide Teilnehmer ihre Nachricht neu senden, sondern einer von beiden sendet weiter.

Was den CAN-Bus auch für Anwendungen jenseits des Fahrzeugs interessant macht, ist seine relativ geringe Störempfindlichkeit bei hohen Übertragungsraten (bis zu 1 Mbit/s) und langen Leitungen (mehrere hundert Meter). Erreicht wird dies unter anderem durch die Verwendung differenzieller Übertragung: Die häufigste Verkabelung nutzt zwei Datenleitungen, CAN-HIGH und CAN-LOW. Entscheidend für den Wert eines Bits ist nicht der absolute Pegel einer Leitung gegen Masse, sondern die Differenz der beiden Leitungen. Da sich Störungen auf beide Leitungen gleich auswirken, bleibt diese Differenz im Normalfall unberührt. Eine Verdrillung der Kabel und eine Abschirmung verbessern die Eigenschaften noch, so dass sich als Übertragungsmedium z. B. CAT5-Kabel anbietet.

Die typische CAN-Topologie verwendet zwei Leitungen, die an den Enden mit Widerständen von je 120 Ohm terminiert sind, um Reflektionen zu vermeiden. Stichleitungen geringer Länge sind erlaubt. Die Kommunikation auf dem CAN-Bus erfolgt seriell und paketweise.

Dank seiner differenziellen Übertragung ist der CAN-Bus relativ störunempfindlich.

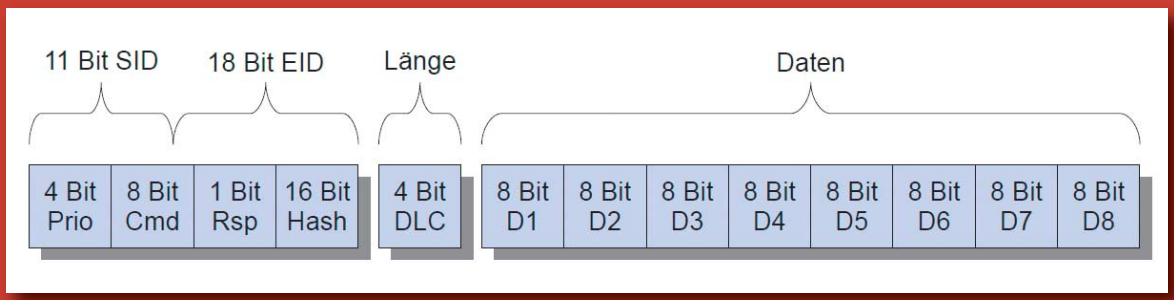


Märklin-Protokoll

Das auf dem CAN-Bus aufsetzende Protokoll von Märklin ist dankenswerterweise in zwei PDFs-Dokumenten erklärt, die im Netz frei verfügbar sind. Das Protokoll nutzt die Variante mit erweiterten IDs, also einer Kennung von insgesamt 29 Bit. Diese 29 Bit sind weiter unterstrukturiert.

Die ersten vier Bit geben eine Priorität für die Nachricht an, die jedoch keine Rolle spielt und einfach 0 sein kann. Das nächste Bit besagt, ob es sich bei der Nachricht um eine Antwort handelt. Der Hash dient zur eindeutigen Identifizierung einer Station.

Das CAN-typische RTR-Bit wird von Märklin nicht genutzt. Stattdessen wird eine Nachricht – sei es Anfrage oder auszuführendes Kommando – in der Regel von der Station, welche die Nachricht behandelt, zurückgesendet. Der Hash wird dann entsprechend verändert, das Antwortbit gesetzt und möglicherweise der Inhalt der Daten angepasst.



Railuino-Examples

Controller – Beispiele zur Steuerung mit dem TrackController

Power: Ein-/Ausschalten der Spannung am Gleis

Headlight: Ein-/Ausschalten des Spitzenlichts einer Lok

Direction: Richtungswechsel einer Lok (im Stand)

Speed: Lok fährt ein Stück und bremst dann wieder ab

CV: zeigt Konfigurationswerte des Decoders einer Lok an

Turnout: schaltet eine Weiche hin und her

Reporter – Beispiele zur Rückmeldung mit dem TrackReporter

Monitor: zeigt die Werte der ersten 16 Kontakte periodisch an

PingPong: wartet auf wechselweises Auslösen zweier Kontakte

Automation – Beispiele zur Kombination von Steuern und Melden zur Automatisierung

Commuter1: Eine Lok fährt über ein Schaltgleis hin- und her.

Commuter2: wie Commuter1, aber mit Weiche und Abstellgleis

Gateway – Sketches zur Anbindung von PC etc.

Serial: einfaches, zeilenweises und textbasiertes Protokoll via serielle Leitung

Bluetooth: gleiches Protokoll über Bluetooth Pin 4: RX – Pin 5: TX

LowLevel: schickt CAN-Pakete als Hexdump, work in progress, derzeit irrelevant

Misc – verschiedene Sketches

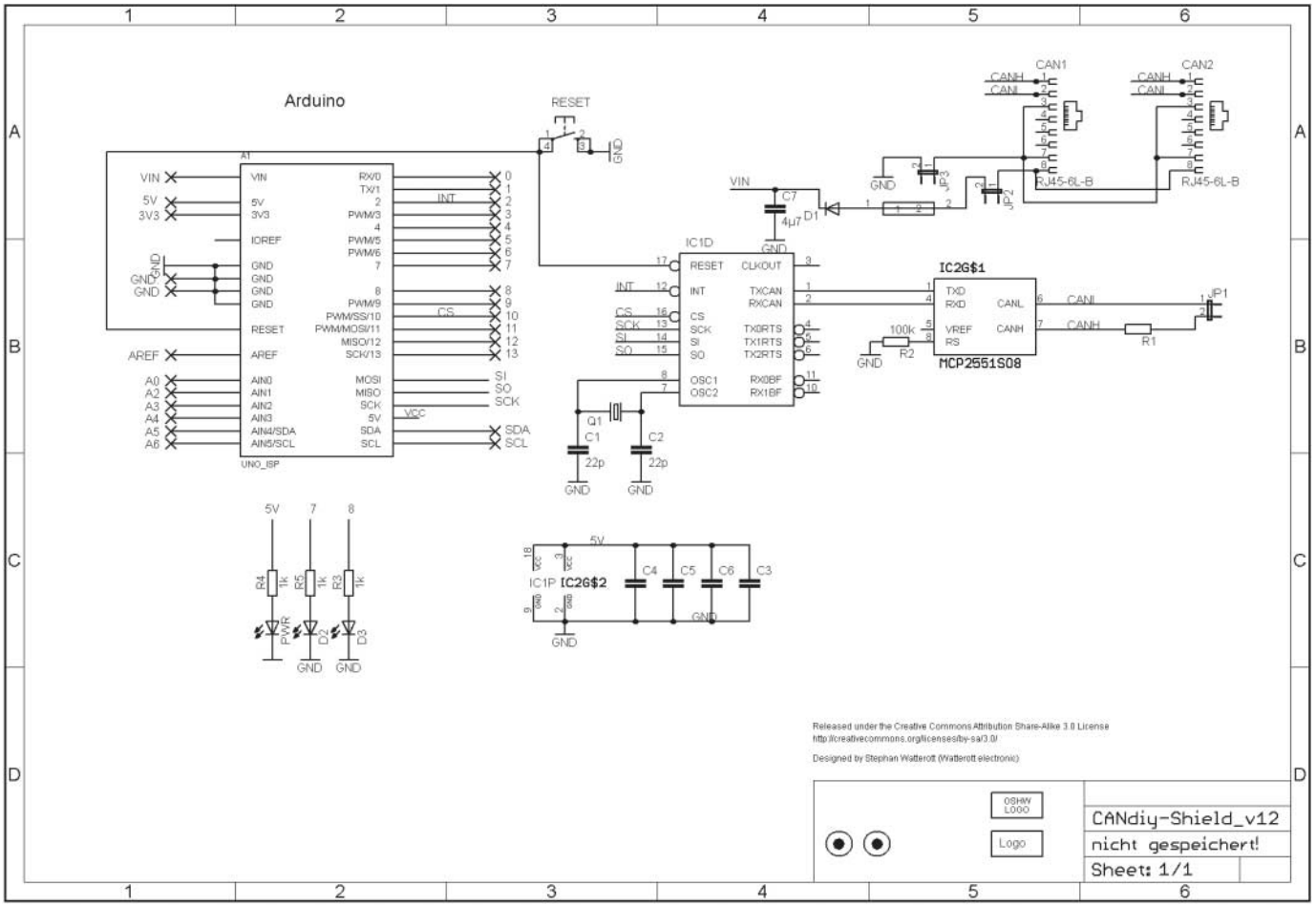
Address: Herausfinden der Adresse, auf die eine Lok hört (Lok muss über MS2 bewegt werden)

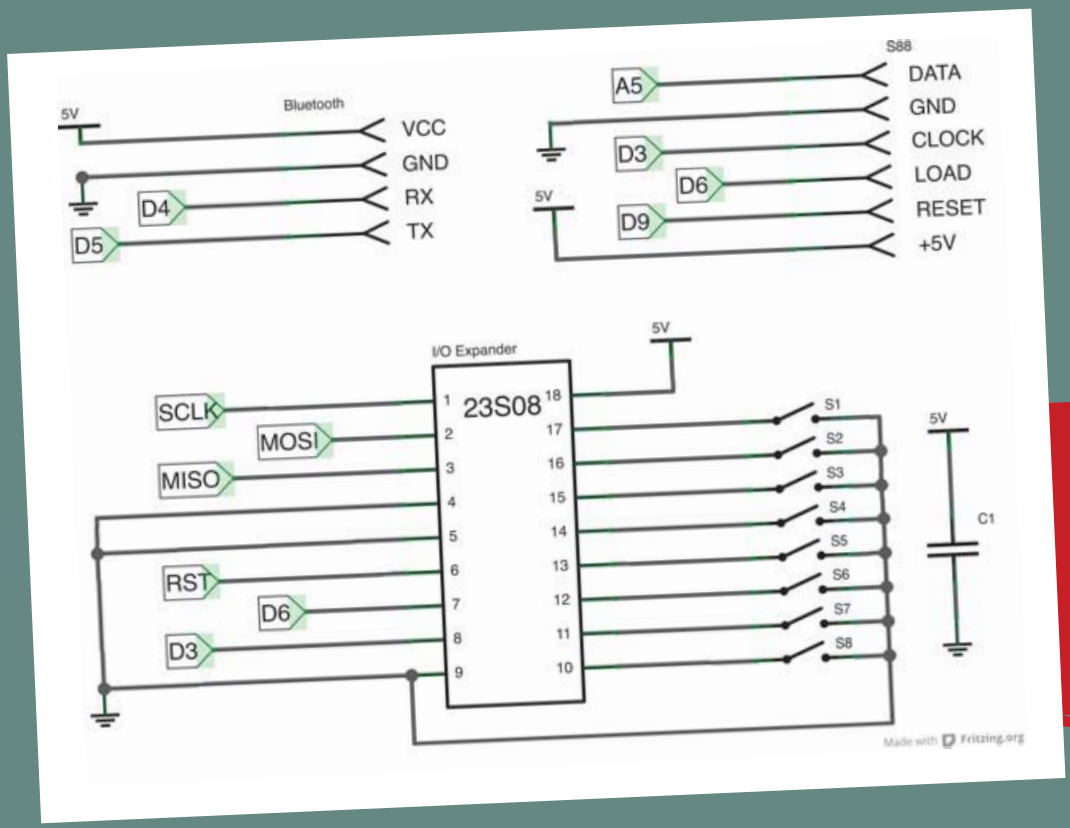
Console: menübasierte Steuerung der Bahn über Arduino-Terminal

Joystick: steuert eine Lok und eine Weiche über A0 – A5 (Joystick beim Sparkfun Shield)

Sniffer: zeigt alle Pakete an, die auf dem CAN-Bus ausgetauscht werden

Tests: führt einen Selbsttest des Systems durch. Erfordert mindestens angeschlossene Gleisbox.





Wer sein CANdiy-Shield noch erweitern will, findet im Schaltplan die Zuordnung der Arduino-Pins zu weiteren Modulen oder Chips.