Decoder selbst bauen – Teil 2

DECODIERUNG

Im heutigen zweiten Teil der Artikelreihe decodieren wir ein Sx-Signal mit Hilfe eines Mikrocontrollers. Zum Einsatz kommt ein Arduino, eine beliebte und preiswerte Hardwareplattform für eigene Projekte (siehe auch Seite 46 dieses Hefts). Die Decoder-Programmierung erfolgt in der Programmiersprache C.

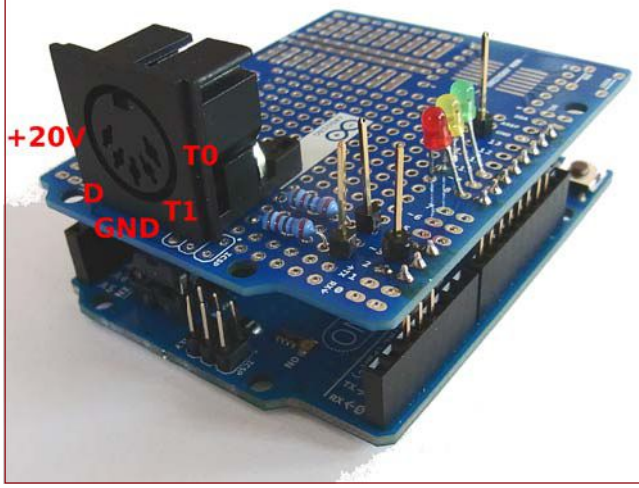
Eine beliebte „Open-Source"-Mikrocontrollerplattform ist der Arduino, dessen Dokumentation (sowohl Hard- als auch Software) komplett offengelegt ist. Es gibt verschiedene Versionen der Hardware mit unterschiedlichen Mikrocontrollern. Gemeinsam ist allen aktuellen Arduino-Boards die eingebaute USB-Schnittstelle und damit die Möglichkeit, den Baustein mit einem PC zu programmieren.

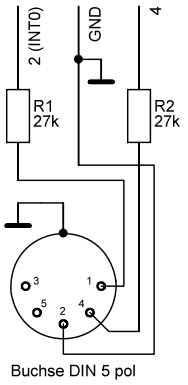
Die nötige Entwicklungsumgebung ist für Windows, Linux und Mac verfügbar. Die sinnvolle Nutzung der Programmierumgebung setzt Grundkenntnisse in C voraus, die man sich aber auch mit Hilfe der Tutorials auf http://arduino.cc aneignen kann.

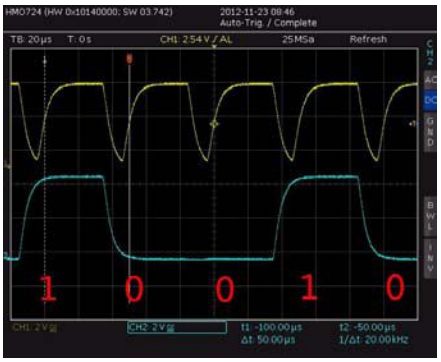
Ein Arduino-Board verfügt über genügend externe Anschlussmöglichkeiten (Ports), um zum Beispiel das Sx-Signal hineinzufüttern, zu decodieren und entsprechende Weichen oder Signale zu steuern. Da das Board über den USB Port angeschlossen wird, kann es vom PC mit 5 V versorgt werden, man braucht keine externe Spannung. Erst wenn man einen Decoder zum Ansteuern von Weichenmotoren oder Signalen baut, muss man sich über die Stromversorgung Gedanken machen.

Zur Steuerung von Weichen, Signalen und zur Abfrage von Handreglern und Belegtmeldern verwendet Selectrix den sogenannten SX-Bus mit fünfpoligen DIN-Steckern und -Buchsen. An den fünf Anschlüssen liegen Masse, +20 V Gleichspannung (Achtung, die 20 V bekommen dem µC nicht gut!) und drei di-

Das Sx-Interface ist auf einer universell verwendbaren Arduino-Shield-Platine aufgebaut. Diese Shields werden auf die eigentliche Controllerplatine ausgesteckt. Zusätzlich zu den im Schaltplan genannten Bauteilen ist das „Sx Shield“ mit LEDs an den Arduino-Pins 8, 9 und 10 ausgestattet, damit man ein optisches Feedback einstellen kann.



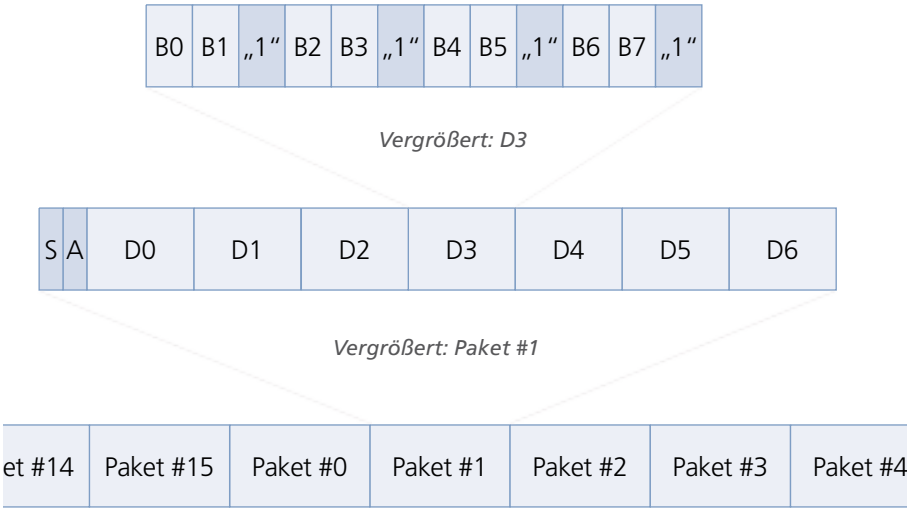




Die 27-kΩ-Widerstände des Interface bilden zusammen mit Leitungs- und sonstigen Kapazitäten jeweils einen Tiefpass, die das Takt- (gelb oben) und das Datensignal (blau unten) verschleifen. Aber auch, wenn die Signale nicht mehr ideal rechteckig sind, können sie eindeutig gelesen werden. Die Abtastung muss jeweils bei der steigenden Flanke des Takts (siehe die senkrechten Linien) erfolgen. Im Beispiel ergibt sich die Bitfolge 1 0 0 1 0.

gitale Signale, die alle 5-V-Pegel haben. Diese können wir über Vorwiderstände zum Schutz des Prozessors direkt anschließen. Das SX-„Interface“ ist daher ausgesprochen einfach und besteht nur aus der Buchse und zwei Widerständen. Die digitalen Signale, die wir brauchen, sind das Takt- (T0) und das Datensignal (T1), das die Zentrale sendet. Das Rücksignal „D“, welches z.B. die Handregler auf den Bus legen, verwenden wir im Moment noch nicht.

Das Grundprinzip des Selectrix-Decoders ist, dass wir bei jedem Taktpuls den Wert des Datensignals T1 einlesen (0 oder 1). Dies ist der Selectrix „Datenstrom“, aus dem wir die Werte der einzelnen Selectrix-Kanäle berechnen.



Aufbau des SX Datenstroms. Dargestellt ist nur ein Teil der 16 ausgesandten Pakete. Es bedeuten: B0 .. B7 = Bits 0–7; „1“ = log. 1; S = Sync (0 0 0 1); A = Spannungs- und Trennbits und Basisadresse; D0 .. D6 = Unterpakete.

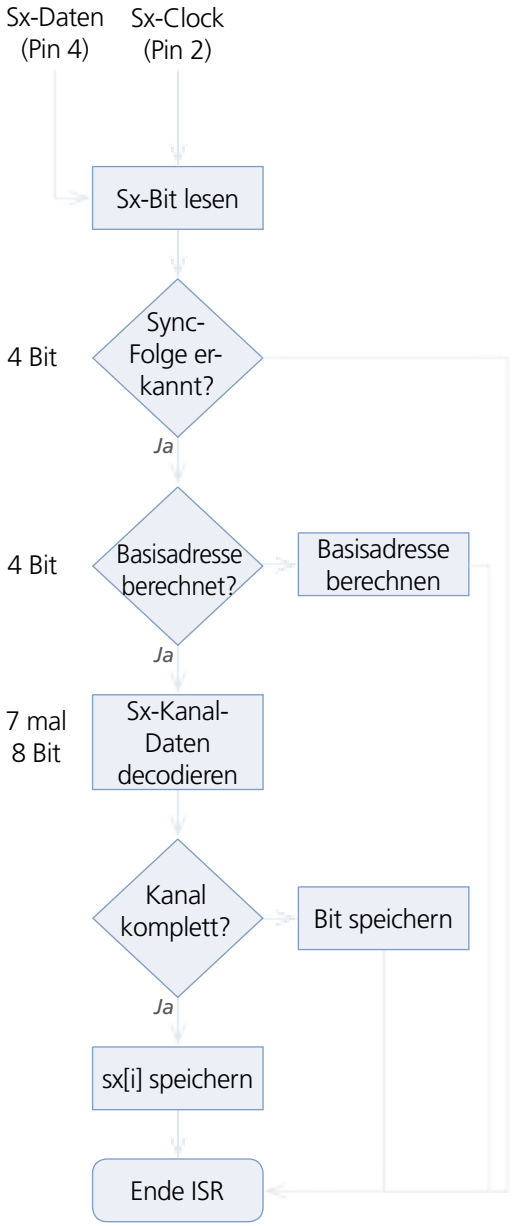
DATENSTROM DECODIERUNG

Selectrix sendet die 112 Kanäle (= Adressen) in 16 verschiedenen Datenpaketen zu jeweils sieben Kanälen, (7 * 16 = 112). Dies erfolgt in einem festen Zeitraster, so dass sich die Datenpakete alle 77 Millisekunden wiederholen. Die ständige Wiederholung bewirkt automatisch eine gewisse Störsicherheit, die SX-Zentrale entscheidet nicht (anders als bei z.B. DCC), wie häufig ein Kanal wiederholt werden muss.

Das Taktsignal ist jeweils 10 µsec auf 0 (= 0 V), dann 40 µsec auf 1 (= +5 V). Das heißt, die Taktfrequenz beträgt 20 kHz. Dies ist also nicht allzu viel, verglichen mit der typischen Taktfrequenz eines AVR-Controllers von 8 Mhz. Die Decodierung ist daher im Timing unproblematisch und man kann sie, der Einfachheit halber, in der Hochsprache C umsetzen und muss nicht in Assembler programmieren.

Um in einem seriellen Datenstrom feststellen zu können, wo man sich gerade befindet, benötigt man eine Synchronisation, einen Anfang des Datenpakets. Das SX-Protokoll verwendet hier die Bitfolge 0 0 0 1 – die sonst natürlich nicht vorkommen darf.

Mit dieser Bitfolge (genannt SYNC) beginnt jedes der 16 Datenpakete. Danach folgt ein Bit, das angibt, ob gerade eine Spannung auf dem Gleis liegt, gefolgt von einem Trennbit, das immer den Wert 1 hat. Die nächsten vier Bits (Wertebereich 0 – 15) enthalten die Basisadresse, die aussagt, welches der 16 Pakete folgt. Die Codierung (Bitreihenfolge) ist hier [Bit3] [Bit2] [1] [bit1] [bit0] [1], also



Das Flussdiagramm des Beispielprogramms zeigt den Ablauf der Decodierung. Mit der steigenden Flanke vom Taktsignal SX-Clock wird der Interrupt #0 (INT0) des µControllers getriggert. Im Programm-Listing erfolgt die entsprechende Zuweisung mit `attachInterrupt(0, sxisr, RISING);`
Entsprechend wird die Routine `sxisr()` *(bzw.* `isr()` *in der Bibliothek) alle 50 µs aufgerufen.*

Es folgt:

```
_bit=digitalRead(SX_DATA);
bitWrite(_data,0,_bit);
```

Die Funktion `digitalRead()` *liest das jeweils zugehörige Datenbit ein,* `bitWrite()` *schreibt es dann an die Position 0 im Byte* `_data`*.*

Im weiteren Ablauf der Interruptroutine wird die bisher empfangene Bitfolge analysiert und decodiert. Sind komplette Sx-Daten zusammengekommen, werden diese zur weiteren Verwendung gespeichert.

mit zwei zwischen die Datenbits eingeschobenen Bits mit Wert 1.

Warum kommt die SYNC-Bitfolge nicht im sonstigen Bitstrom vor? Ein Daten-Byte (= 8 Bits) wird bei SX nicht als acht aufeinander folgende Bits gesendet, sondern es wird jeweils nach zwei Bit Nutzdaten ein Bit mit Wert 1 eingefügt. Beispiel: Das Wert (dezimal) 43 soll gesendet werden, die Bitfolge ist also 43 = (32+8+2+1) = 0 0 1 0 1 0 1 1. Daraus wird bei SX 0 0 1 1 0 1 1 0 1 1 1 1, also ein auf zwölf Bit verlängertes Daten-"Bytes". Die SYNC-Folge 0 0 0 1 kann nun im Datenstrom nicht mehr vorkommen, da nach spätestens zwei Datenbits eine 1 gesendet wird.

DAS C-PROGRAMM

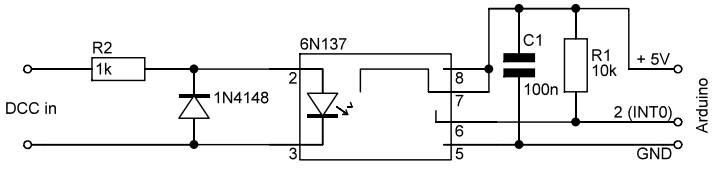
Da der SX-Datenstrom immer dem gleichen Muster folgt, bietet es sich an, seine Decodierung in einer immer wiederverwendbaren „Library" für den Arduino vorzunehmen. Diese Bibliothek kann von http://www.oscale.net/selectrix-arduino kostenlos geladen werden und wird mit #include "SX.h" in eigene Projekte eingebunden.

Wie bei vielen Mikrocontroller-Projekten ist das Hauptprogramm, das für die Ausgabe zuständig ist, in einer Endlosschleife eingebettet. Diese wird von einer Interrupt-Funktion unterbrochen, sobald an einem Eingangspin neue Daten vorliegen.

Nach dem Ausführen von Initialisierungs-Befehlen im setup()-Unter-

DCC UND ARDUINO

Für den Arduino gibt es ebenfalls eine Bibliothek für DCC. Zu ihrer Verwendung muss ebenfalls eine passende Interface-Elektronik aufgebaut werden. Da bei DCC keine Differenzierung zwischen Gleis- und Bus-Signal gemacht wird, stehen zur Auswertung nur die +/- 15 bis 20 Volt zur Auswertung zur Verfügung, die die Zentrale als Gleissignal bereitstellt. Daher benötigt man für das DCC-Interface etwas mehr Hardware, um die Zentralenspannung in ein 0/5-V-Signal umzuwandeln. Typischerweise wird hier ein (schneller) Optokoppler im Eingang verwendet. Dies sorgt auch gleich für eine galvanische Trennung des Gleissignals vom PC.
Auch hier wird der Eingang Pin2 (INT0) des Arduino verwendet und jeweils die



Dauer eines Pulses gemessen. Bei DCC werden die logischen Signale 0 und 1 durch lange und kurze Pulse repräsentiert.
Ein Arduino lässt sich auch zum Generieren von DCC-Signalen verwenden. Eine sehr einfache Arduino-DCC-„Zentrale" (Steuerung nur einer einzigen Lok) findet sich unter http://www.oscale.net/simpledcc. Hier finden DCC-Freunde auch weitere Links als Startpunkt für eigene Projekte.

DCC-Bibliothek für Arduino: http://mrrwa.org

DCC-"Zentrale" für Arduino: http://www.oascale.net/simpledcc

programm – hier wird zum Beispiel die Baudrate der Seriellen Schnittstelle eingestellt und es wird definiert, welche Pins des Arduinos Eingänge oder Ausgänge sein sollen – läuft das Hauptprogramm loop() in einer Endlosschleife. In loop() wird getestet, ob sich ein SX-Kanal geändert hat. Falls ja, wird die neue Information über die serielle Schnittstelle (USB) an den PC geschickt.

Die Interrupt-Routine isr() übernimmt die eigentliche Decodierung. Der Interrupt (= Unterbrechung) wird alle 50 µsec durch die steigende Flanke des Taktsignal am Pin INT0 angestoßen. Die Hauptprogramm-Schleife wird unterbrochen, an ihrer statt erfolgt die Abarbeitung der Interrupt-Service-Routine. isr() muss innerhalb der 50 µsec bis zum nächsten Interrupt beendet sein.

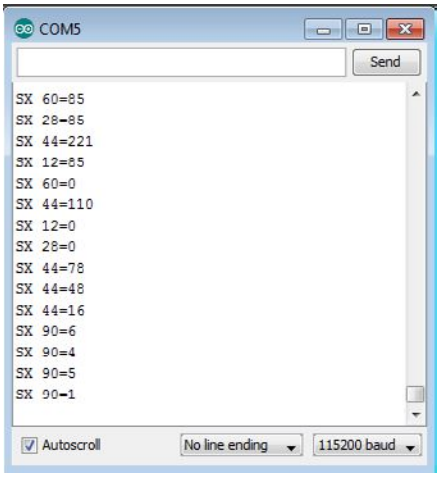
Die isr()-Funktion befindet sich immer in einem von verschiedenen Zuständen:

- warten auf eine SYNC-Bitfolge (drei „0"-Bits, die von einem „1"-Bit gefolgt werden).
- Auswerten der Bits für eine neue Basisadresse, switchAdr().
- Auswerten der Bits der Unterpakete (= Daten), switchData(). Die Datenbits werden in dieser Funktion an der richtige Bitposition von _data abgelegt.

Aus Basisadresse und Nummer des Unterpakets wird, wenn die acht Bits eines Unterpakets Dn (n = 0–6) komplett gelesen sind, jeweils die Kanalnummer (0–111) berechnet und in einem Array sx[Kanal] gespeichert. Dabei ist Kanal = (15 - baseAdr) + (6 - n) * 16. Diese nicht gerade naheliegende Art der Kanalberechnung hat vermutlich historische Gründe. Für einen einfachen Weichendecoder müssen allerdings nicht alle Kanäle gespeichert werden, sondern nur die Daten des Kanals, auf dem der Decoder arbeitet.

Michael Blank

In der nächsten Folge werden wir einen Selectrix Weichen- und Signaldecoder (mit einem ATtiny2313) entwickeln und auch detaillierter auf Hardwarefragen eingehen.



Hier ist der PC-Output des mit der Bibliothek mitgelieferten Beispielprogramms dargestellt. Das Programm kann die Bits eines ausgewählten SX-Kanals auf LEDs ausgeben und sendet außerdem alle Änderungen von SX-Kanälen an den über USB angeschlossenen PC. Im Screenshot „COM5" sieht man, wie eine Lok auf Adresse 44 gesteuert wird. Adresse 90 gehört zu einem Belegtmelder. Da verschiedene Blockabschnitte durchfahren werden, ändern sich diese Daten ebenfalls.

(Fotos: Michael Blank und Wolfgang Bertle)