SRCPD RUST

Daniel Sigg, daniel@siggsoftware.ch
V1.0.0, 22.10.23

1 Allgemeines

Der in RUST geschriebene SRCP Server (https://siggsoftware.ch/wordpress/srcpd-in-rust/) ist im wesentlichen kompatibel mit dem original srcpd (https://srcpd.sourceforge.net/). Aktuell wird unterstützt:

- DDL über SPI
  - Märklin Motorola Protokolle, MM
  - DCC, inkl. Servicemode mit Haupt- und Programmiergleis
  - MFX, inkl. automatische Lokanmeldung, Servicemode
- S88 Bus über SPI

Gegenüber dem Original sind folgende Verbesserungen enthalten, die zum Teil natürlich auch schon in meiner srcpd Erweiterung (https://siggsoftware.ch/wordpress/srcpd-mit-mfx-und-anderen-erweiterungen/) enthalten sind:

- Optimierte Ausgabe über DCC, es werden keine unnötigen Pasuen in den Datenstrom eingefügt. Verlangt ein Protokoll eine Pause, wird sie für andere Telegramme verwendet.
- MFX
- Für Loks (GL): Priorisierung Haltkommandos
- (Alte) Schaltdekoder wie Märklin k83 können gleichzeitig nur ein Ausgang aktiv haben

2 Konfiguration

Standardmässig wird die Konfigurationsdatei „/etc/srcpd.conf“ verwendet und der srcpd läuft nach „fork“ im Hintergrund weiter. Mit dem Parameter „-n“ kann das „fork“ unterdrückt werden, mit dem Parameter „-f configfile“ kann beim Start eine andere Konfigurationsdatei angegeben werden.

2.1 Konfigurationsdatei

Die Konfigurationsdatei hat einen Abschnitt für den srcp-Server selbst und je einen für alle vorhandenen Server (aktuell S88 und DDL).

2.2 srcp Server

```
[srcp]
port = 12345
```

Einzige Angabe: Port, auf dem srcpd auf srcp Verbindungen wartet.

2.3 S88

```
[s88]
bus = 1
refresh = 50
repeat = 3
spiport = /dev/spidev1
spimode = 2
number_fb_1 = 18
number_fb_2 = 23
number_fb_3 = 0
number_fb_4 = 0
```

bus:
Erster verwendeter srcp Bus. Es werden ab da IMMER 4 Busse reserviert, in diesem Beispiel als 1 bis 4. Jeder der max. S88 Busse wird über einen eigenen srcp Bus gemeldet.

refresh:
Einleseintervall der S88 Busse. Angabe in ms.

repeat:
Ungerade Zahl, Anzahl Einlesen für Filterung. Bei 3 ist bei Ungleichheiten min. 2 mal das selbe notwendig. Damit können einzelne Bitfehler korrigiert werden.

spiport:
Der zu verwendende SPI Port. Füt den ersten S88 Bus wird dann spix.0, für den zweiten spix.1 verwendet.

spimode
Zu verwendender SPI Mode, siehe S88 SPI Schema. Eigentlich Mode 1, SPI1 auf den Raspberry PI unterstützt diesen aber nicht, deshalb Schaltung für Mode 2 verwenden.

number_fb_[1..4]
Die Anzahl S88 Module für Bus 1 bis 4. 0 Wenn Bus nicht verwendet wird.

2.4 DDL

```
[ddl]
bus = 5
spiport = /dev/spidev0
maerklin
dcc
mfx=1021970
mfx_reg_count_file = /etc/srcpd.regcount
siggmode
timeout_shortcut_power_off = 10000
shortcut_delay = 500
#watchdog
```

bus
Zu verwendender srcp Bus.

spiport
Zu verwendender SPI Bus für die Ausgabe.

maerklin
Wenn enthalten, wird das Märklin / Motorola Protokoll aktiviert.

dcc
Wenn enthalten, wird das DCC Protokoll aktiviert.

mfx=UIDZentrale
Wenn enthalten, wird das MFX Protokoll aktiviert. Es muss die für die Zentrale zu verwendende UID (32 Bit) angegeben werden.

mfx_reg_count_file
Angabe des Files in dem der MFX Neuanmeldezählerstand gespeichert wird.

siggmode
Wenn vorhanden wird der / die Booster über Impulse an RTS gestartet und DTS gestoppt. Ansonsten definiert RTS Booster Ein/Aus.

timeout_shortcut_power_off
Nur bei „siggmode“: wieviele Millisekunden muss der Booster eingeschaltet sein, damit bei einer Abschaltung (wegen Kurzschluss), einmalig versucht wird automatisch wieder einzuschalten.

shortcut_delay
Wenn nicht „siggmode“: Verzögerung in Millisekunden bis auf Überlastmeldung mit Ausschalten reagiert wird.

watchdog
Wenn vorhanden: Watchdog auf srcp Kommandos. Automatische Boosterausschaltung

wenn länger als 2s keine Kommando auf srcp Kommandosession empfangen wird.

3 SRCP Protokoll

Das SRCP Protokoll ist im wesentlich wie in der Spezifikation (https://srcpd.sourceforge.net/srcp/index.html) definiert implementiert. Jedoch NICHT:

- Device Groups DESCRIPTION, GM, LOCK, SERVER, TIME
- Für FB (S88 Bus) gilt, dass die Zustände über INFO automatisch bei Veränderungen gemeldet werden. Es werden KEINE Kommandos (SET, GET, WAIT, INIT, TERM) unterstützt.

3.1 GL Erweiterungen für MFX

Für MFX (Protokollbezeichner „X“) gilt folgender GL INIT:

```
INIT <bus> GL <addr> X „Lokname“ CodeF0 CodeF1 … CodeF15
```

Der Lokname muss in Anführungszeichen stehen (er kann Leerschläge enthalten), CodeFx sind 3 Bytes grosse Integer die die MFX Funktionscodes (siehe MFX Protokoll Beschreibung http://www.skrauss.de/modellbahn/Schienenformat.pdf) enthalten:

- Bit 0..7: S2
- Bit 8..15: S1
- Bit 16..23: Gruppe

Wenn eine neue Lok erkannt wird, dann wird dieser INIT automatisch über INFO Kanal gesendet.

3.2 Servicemode

Es wird Servicemode für DCC und MFX unterstützt. Es kann gleichzeitig IMMER nur ein Servicemode aktiv sein!

```
INIT <busnr> SM NMRA
INIT <busnr> SM MFX
TERM <busnr> SM
```

3.3 Servicemode DCC

Wie in SRCP Beschrieben:

```
SET <bus> SM <decoderaddress> <type> <1 or more values>
```

Jedoch nur: type CV und CVBIT. REG und PAGE werden NICHT unterstützt.

Wenn Booster eingeschaltet ist, wird Hauptgleisprogrammierung verwendet, d.h. nur SET möglich. Wenn der Booster ausgeschaltet ist, wird automatisch Programmiergleis verwendet. Es muss auch dann die Dekoderadresse > 0, die für DCC initialisiert ist, verwendet werden, nicht -1! Die Ausgabe für Programmiergleis startet bei

ausgeschaltetem Booster und aktiviertem DCC SM an SPI. Damit ist sichergestellt, dass die Anlage NIE Programmiergleisbefehle erhält.

VERIFY und GET werden unterstützt wenn ACK Impuls Schaltung mit Rückmeldung an RI (Raspberry PI GPIO 22) vorhanden ist.

3.4 Servicemode MFX

Für MFX Servicemode gilt:

```
SET <bus> SM <decoderaddress> <type> <block> <ca> <caindex> <index> <value>
VERIFY <bus> SM <decoderaddress> <type> <block> <ca> <caindex> <index> <value>
GET <bus> SM <decoderaddress> <type> <block> <ca> <caindex> <index>
```

type: immer „CAMFX“

block, ca, caindex, index: Siehe MFX Protokollbeschreibung
http://www.skrauss.de/modellbahn/Schienenformat.pdf

4 Konfiguration Raspberry PI

Auf dem Raspberry PI sind mit dem da vorhandenen BCM Chip die SPI Clocks abhängig vom Core-Clock. Da dieser normalerweise dynamisch der CPU Last angepasst wird (Cache der ARM CPU's hängt auch daran), verändert sich die SPI Ausgabe damit. Wenn SPI als SPI mit Clock Leitung verwendet wird, dann stört das normalerweise nicht so. Hier wird SPI aber „missbraucht“ um die MM, DCC und MFX Signale zu erzeugen. Da ist eine Clockänderung natürlich nicht zu gebrauchen. Deshalb muss diese ausgeschaltet werden.

Wenn SPI verwendet wird, dann kommt dazu, dass der kleinst mögliche SPI Clock gleich Core Clock geteilt 8192 ist. Das heisst, der Core Clock darf sicher nicht schneller als 250 MHz sein, damit haben wir dann 30.5 kHz SPI Clock, was noch stabil funktioniert (50 kHz funktioniert bei mir nicht mehr stabil).

Das bedeutet, dass in /boot/config.txt folgendes enthalten sein muss:

```
core_freq=250
core_freq_min=250
```