

## Beeriger MIDI-Übersetzer – Raspberry Pi als MIDI-Router

Am Anfang des Projektes stand der Wunsch des Autors das vorhandene Digitalpiano Casio Celviano mit anderen Klangerzeugern zu koppeln, wofür seit 1985 der MIDI-Standard sorgt. Das Casio Digitalpiano hat allerdings die moderne MIDI-Interpretation über eine USB-Verbindung, während die meisten anderen Klangerzeuger des Autors – wie das hier gezeigte Yamaha QY70 – über die klassischen 5poligen MIDI-DIN-Buchsen verfügen.



Abbildung 2: Yamaha QY70 mit klassischem MIDI



Abbildung 1: Digitalpiano Casio Celviano mit USB-MIDI

Daher war es nötig, einen MIDI-Router zu entwerfen der das MIDI-USB-Protokoll in ein serielles MIDI-Protokoll in beide Richtungen übersetzen kann.

Natürlich könnte man das Problem über einen beliebigen Microcontroller lösen, der HW-seitig zumindest über eine USB-Schnittstelle und eine serielle Schnittstelle verfügt. SW-seitig ist aber einiges an Aufwand nötig, um einen USB-Protokollstack und einen MIDI-Protokollstack zu implementieren. All diese Voraussetzungen waren an einem ganz normalen PC mit USB-Anschlüssen und einem billigen externen USB-MIDI-Konverter erfüllt. Die SW-seitige Kopplung beider Schnittstellen ließ sich bei einem Debian-basierten Betriebssystem wie z.B. Ubuntu sehr einfach über die ALSA-Utilities und ein wenige Zeilen langes Perl-Skript darstellen. Etwas unhandlicher war es da aber schon jedes Mal den PC zu booten, wenn man vom Digitalpiano das QY70 spielen wollte. Also galt es etwas PC-ähnliches mit den Abmaßen eines Embedded Systems zu finden. Hier wurde der Autor schnell beim Raspberry Pi fündig.

## Die Software

Die eigentliche Software des MIDI-Routers besteht aus den folgend dargestellten wenigen Zeilen Perl-Kode.

Der Grund für diese Einfachheit liegt darin, dass umfangreiche und bereits vorhandene Systembibliotheken wie die ALSA<sup>1</sup>-Library für die MIDI-Kommunikation benutzt wurden. Außerdem wurden noch Perl-Module benutzt, die als Schnittstelle zwischen Perl und den Systembibliotheken fungieren. Diese Perl-Module wurden über <http://www.cpan.org> als Archive heruntergeladen und über das Befehlsstripel „perl Makefile.PL“, „make“ und „sudo make install“ kompiliert und in die Kommandoverzeichnisse kopiert.

Zum Einsatz kam zum einen „MIDI-ALSA-1.20“ (<https://metacpan.org/pod/MIDI::ALSA>) und zum anderen „Term-ANSIColor-4.03“ (<https://metacpan.org/pod/Term::ANSIColor>). Der Sourcecode des MIDI-Routers basiert zum größten Teil auf (1).

In den folgenden beiden Bildern ist der Teil des Perl-Skripts „midiRouter.pl“ dargestellt, der anfänglich auf dem Büro-PC des Autors entwickelt wurde, um die MIDI-Umsetzung zu realisieren. In der Abbildung 3 sieht man die Deklaration und Initialisierung der benutzten Variablen, sowie die Einbindung der benutzten Funktionen.

<sup>1</sup> ALSA = Advanced Linux Sound Architecture

In der Abbildung 4 sieht man die insgesamt 2 MIDI-Verbindungen, die im MIDI-Router vorgenommen werden, vom MIDI-Ausgang des Casio Celvianos zum MIDI-Eingang des QY70 (über den MIDI-Ausgang des USB-MIDI-Konverters) und vom MIDI-Ausgang des QY70 (über den MIDI-Eingang des USB-MIDI-Konverters) zum MIDI-

```

midiRouter.pl
MIDI::ALSA::connectfrom( 0, 'CASIO' ); # CASIO USB-MIDI-OUT
MIDI::ALSA::connectto( 1, 'USB2.0' ); # USB2.0-MIDI-IF-IN
MIDI::ALSA::connectfrom( 0, 'USB2.0' ); # USB2.0 USB-MIDI-OUT
MIDI::ALSA::connectto( 1, 'CASIO' ); # CASIO-MIDI-IF-IN

while ($processMidiEvents) {
    @alsaevent = MIDI::ALSA::input();
}

my_cBerry28 midiRouter.pl
use MIDI::ALSA (':CONSTS');
use Term::ANSIColor qw(:constants);
use strict;

local $Term::ANSIColor::AUTORESET = 1;
#---- Variables -----
my $programVersion = "2015-12-12_1.14";
my $debug = 0;
my $sk = "";
my $sv = "";
my %clientnumber2clientname;
my %clientnumber2howmanyports;
my $keyboardPort = 0;
my $synthiPort = 0;
my $stst = SND_SEQ_EVENT_NOTEON();
my $schannel = 0;
my $smidiByte1 = 0;
my $smidiByte2 = 0;
my $smidiByte3 = 0;
my $keyboardClientNo = 0;
my $synthiClientNo = 0;
my $noValidMidiClients = 1;
my $processMidiEvents = 1;
my $retVal = 0;
my @alsaevent;
#-----
sub wait_seconds($)
{
    my ($wait_time) = @_;
    my $start = 0;
    my $stop = 0;
    $start = time;
    $stop = time;
    while (($stop - $start) < $wait_time)
    {
        $stop = time;
    }
    return 0;
}
#---- Functions -----
#-----
Zelle: 42 Spalte: 22 Zeichen: 3882 EINF

```

```

midiRouter.pl [geändert]
my_cBerry28 *midiRouter.pl
MIDI::ALSA::connectfrom( 0, 'CASIO' ); # CASIO USB-MIDI-OUT
MIDI::ALSA::connectto( 1, 'USB2.0' ); # USB2.0-MIDI-IF-IN

while ($processMidiEvents) {
    @alsaevent = MIDI::ALSA::input();
    if ($alsaevent[0] == SND_SEQ_EVENT_PORT_UNSUBSCRIBED()) { last; }
    if ($alsaevent[0] == SND_SEQ_EVENT_NOTEON() ||
        $alsaevent[0] == SND_SEQ_EVENT_NOTEOFF()) {
        if ($alsaevent[5][0] == $keyboardClientNo)
        {
            print BOLD CYAN "CASIO ::";
        }
        elsif ($alsaevent[5][0] == $synthiClientNo)
        {
            print BOLD MAGENTA "USB2.0::";
        }
        $schannel = $alsaevent[7][0];
        if ($alsaevent[0] == SND_SEQ_EVENT_NOTEOFF())
        {
            $smidiByte1 = $schannel+0x80;
        }
        else
        {
            $smidiByte1 = $schannel+0x90;
        }
        $smidiByte2 = $alsaevent[7][1];
        $smidiByte3 = $alsaevent[7][2];
        print "mb1: $smidiByte1 | mb2: $smidiByte2 | mb3: $smidiByte3\r\n";
    }
    elsif ($alsaevent[0] == SND_SEQ_EVENT_CONTROLLER()) {
        if ($alsaevent[5][0] == $keyboardClientNo)
        {
            print BOLD CYAN "CASIO ::";
        }
        elsif ($alsaevent[5][0] == $synthiClientNo)
        {
            print BOLD MAGENTA "USB2.0::";
        }
        $schannel = $alsaevent[7][0];
        $smidiByte1 = $schannel+0x80;
        $smidiByte2 = $alsaevent[7][4];
        $smidiByte3 = $alsaevent[7][5];
        if (($smidiByte1 >= 0x80) && ($smidiByte2 == 0) && ($smidiByte3 == 0))
        {
            $processMidiEvents = 0;
        }
        print "mb1: $smidiByte1 | mb2: $smidiByte2 | mb3: $smidiByte3\r\n";
    }
}
MIDI::ALSA::output( @alsaevent );
MIDI::ALSA::disconnectfrom( 0, 'CASIO' ); # CASIO USB-MIDI
Zelle: 141 Spalte: 1 Zeichen: 3881 EINF

```

Abbildung 3: midiRouter.pl : Variablendeklaration + benutzte Funktionen

Abbildung 5: Hauptwandlungsroutine

Midi-Routers zu sehen. Die Wandlung wird für alle MIDI-Nachrichten der beiden Verbindungen durchgeführt, angezeigt werden aber nur MIDI-Note-On-, MIDI-Note-Off- und MIDI-Controller-Nachrichten. Je nachdem, ob die Daten vom QY70 oder vom Casio Celviano empfangen worden sind, wird der Text „CASIO ::“ bzw. „USB2.0::“ in der Farbe Cyan oder Magenta gefolgt von den übertragenen 3-MIDI-Bytes ausgegeben.

Die eigentliche Wandlung verbirgt sich hinter den beiden Zeilen „@alsaevent = MIDI::ALSA::input();“ und „MIDI::ALSA::output( @alsaevent );“.

Wie in der letzten Zeile von Abbildung 5 zu sehen ist, erfolgt beim Verlassen der „while (\$processEvents)“-Schleife eine Trennung der 2 anfangs aufgebauten MIDI-Verbindungen. Auch das Disconnect besteht aus insgesamt 4 Zeilen analog dem Connect in Abbildung 4, was hier lediglich aus Platzgründen nicht dargestellt ist.

## Die Hardware

Zum Einsatz kam ein Raspberry Pi B (Artikelnummer „RASPBERRY PI B“ bei Reichelt Elektronik) mit 512MB RAM, 2 x USB, 1 x Ethernet, HDMI und SD-Kartenfassung. Um ein Lebenszeichen des



Abbildung 6: Raspberry mit aufgestecktem Display in TEK-BERRY-Gehäuse



Abbildung 8: Micro-USB-Netzteil



Abbildung 7: Raspberry mit eingesteckter NOOBS-SD-Karte

laufenden MIDI-Übersetzers anzeigen zu können, kam noch ein aufsteckbares Grafikdisplay mit 2,8"-Diagonale und 320 x 240 Bildpunkten (Artikelnummer „RASP C-BERRY 28“ bei Reichelt Elektronik) dazu. Die Stromversorgung erfolgt über ein Micro-USB-Steckernetzteil mit 5V und 2A (Artikelnummer „HNP12 MICRO USB“ bei Reichelt Elektronik), als Gehäuse wurde ein transparentes TEK-BERRY-Gehäuse verwendet. Für den einfachen Start hat der Autor eine SD-Karte mit vorinstallierten, verschiedenen Raspberry-Pi-Betriebssystemen (NOOBS) unter anderem auch Raspbian verwendet (Artikelnummer „RASP OS 8GB 1.4“ bei Reichelt Elektronik).

Für die Verbindung zwischen dem USB des Raspberry und den klassischen MIDI-DIN-BUCHSEN sorgt ein Logilink-USB-MIDI-Interface (Artikelnummer „LOGILINK UA0037“ bei Reichelt Elektronik).



Abbildung 9: Logilink-Midi-Interface

## Erstinstallation

Für die Erstinstallation empfiehlt sich der Anschluss einer USB-Tastatur und eines Monitors oder Fernsehers mit HDMI-Anschluss. Nachdem die SD-Karte mit NOOBS ein- und das Netzteil angesteckt ist, bootet der Raspberry Pi und in einem Eingabeprompt kann man auswählen welches der verschiedenen Betriebssysteme auf der SD-Karte installiert werden soll. Für den MIDI-Router wurde Raspbian ausgewählt. Für weitere Details zur Installation von Raspbian möchte der Autor auf das Buch (2) verweisen, was er selbst für die ersten Schritte mit dem Pi verwendet hat. Gemäß Rezept 1.12 wurde der gesamte Speicher der SD-Karte genutzt, nach Rezept 1.15 das Passwort geändert und nach Rezept 2.7 der SSH-Daemon aktiviert. Alle weiteren Inbetriebnahmeschritte erfolgten dann nach einem Reboot des Raspberry Pi ohne angeschlossene Tastatur und ohne Monitor einfach über den „Putty SSH Client“ vom heimischen Büro-PC aus, wobei der Pi natürlich über seine Ethernetschnittstelle im Netzwerk eingebunden war.

## Installation des Perl-Skripts auf dem Pi

Im nächsten Schritt wurden die Entwicklungspakete von „libasound“ über den Befehl „`sudo apt-get install libasound2-dev`“ auf dem Pi installiert.

Von CPAN wurde das Perlmodul „MIDI-ALSA-1.20“ zuerst auf den Büro-PC heruntergeladen und entpackt. Die entpackten Inhalte wurden über gFTP vom Büro-PC auf den Pi kopiert (Verzeichnis: „/home/pi/Downloads/MIDI-ALSA-1.20“). Auf dem Pi wurde in dieses Verzeichnis gewechselt und das Modul wurde über die Befehle „`perl Makefile.PL`“, „`make`“ und „`sudo make install`“ auf dem Pi installiert.

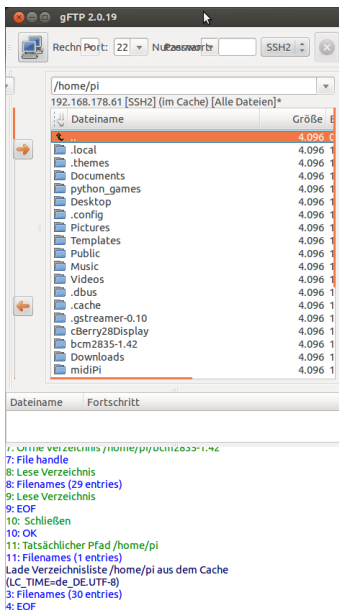


Abbildung 10: Verzeichnisstruktur von /home/pi nach vollständiger Installation

Von CPAN wurde das Perlmodul „Term-ANSIColor-4.03“ zuerst auf den Büro-PC heruntergeladen und entpackt. Die entpackten Inhalte wurden über gFTP vom Büro-PC auf den Pi kopiert (Verzeichnis: „/home/pi/Downloads/Term-ANSIColor-4.03“). Auf dem Pi wurde in dieses Verzeichnis gewechselt und das Modul wurde über die Befehle „*perl Makefile.PL*“, „*make*“ und „*sudo make install*“ auf dem Pi installiert. Das bereits auf dem Büro-PC lauffähige Perlskript „*midiRouter.pl*“ wurde nun über gFTP auf den Raspberry Pi kopiert (Verzeichnis : „/home/pi/midiPi“).

Nach Verbinden des MIDI-Interfaces und des CASIO Celviano mit dem Pi stand einem ersten Test des Midi-Routers auf dem Pi nichts mehr im Wege.

## Inbetriebnahme des Grafikdisplays

Um das Grafikdisplay ansteuern zu können, muss auf dem Pi die „bcm2835“-Library installiert sein. Die Installation erfolgte über die Eingabe und Ausführung der folgenden Befehle:

1. Start im Verzeichnis „/home/pi“
2. Dort „*wget*“ [http://www.airspayce.com/mikem/bcm2835/bcm2835-](http://www.airspayce.com/mikem/bcm2835/bcm2835-1.42.tar.gz)

[1.42.tar.gz](http://www.airspayce.com/mikem/bcm2835/bcm2835-1.42.tar.gz)“ ausführen

3. Entpacken des Archivs über „*tar xzvf bcm2835-1.42.tar.gz*“
4. Wechseln in das Verzeichnis „/home/pi/ bcm2835-1.42“
5. Dort folgende Konfigurationsbefehle ausführen:
  6. *./configure*
  7. *make*
  8. *sudo make check*
  9. *sudo make install*

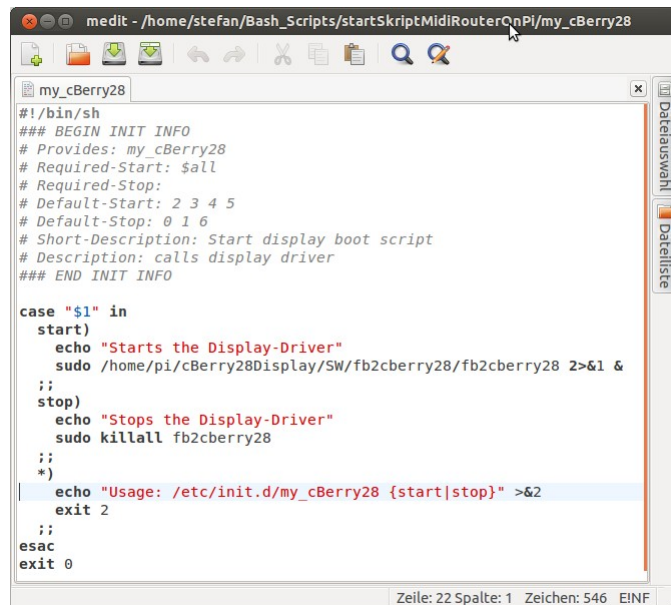
Im nächsten Schritt wurden die Treiber für das Grafikdisplay von der Herstellerseite über folgende Adresse „<http://admatec.de/sites/default/files/downloads/cBerry28.tar.gz>“ heruntergeladen und auf dem Büro-PC entpackt. Auf dem Pi wurde das Verzeichnis „/home/pi/cBerry28Display“ erstellt und die entpackten Inhalte wurden vom Büro-PC auf den Pi über gFTP kopiert.

In der Konfigurationsdatei des Pi „/boot/config.txt“ wurden folgende Änderungen vorgenommen:

1. enable            disable\_overscan = 1
2. disable           #hdmi\_force\_hotplug = 1
3. disable           #config\_hdmi\_boost = 4
4. disable           #overscan\_left = 24
5. disable           #overscan\_right = 24
6. disable           #overscan\_top = 16
7. disable           #overscan\_bottom = 16
8. disable           #disable\_overscan = 0
9. enable            frambuffer\_width= 320
10. enable           frambuffer\_height= 240

Nachdem die Datei gespeichert und der Pi neu gebootet wurde, wurde in das Verzeichnis „/home/pi/cBerry28Display/SW/fb2cberry28 “ gewechselt. Hier wurde über den Befehl „*sudo make*“ die Firmware des Grafikdisplays kompiliert. Ein erster Test wurde durch Eingabe von „*sudo ./fb2cberry28 &*“ durchgeführt. Wenn bis hierher alles mit der Installation des Grafikdisplaytreibers funktioniert hat, sollte das Display den Inhalt des Bildschirmpuffers anzeigen.

Als nächstes musste dafür gesorgt werden, dass der Treiber automatisch bei jedem Bootvorgang des Raspberry Pi automatisch geladen wurde. Dafür wurde die Datei „my\_cBerry28“ im Verzeichnis „/etc/init.d“ angelegt und folgende Inhalte hineingeschrieben:



```

my_cBerry28
#!/bin/sh
### BEGIN INIT INFO
# Provides: my_cBerry28
# Required-Start: $all
# Required-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Start display boot script
# Description: calls display driver
### END INIT INFO

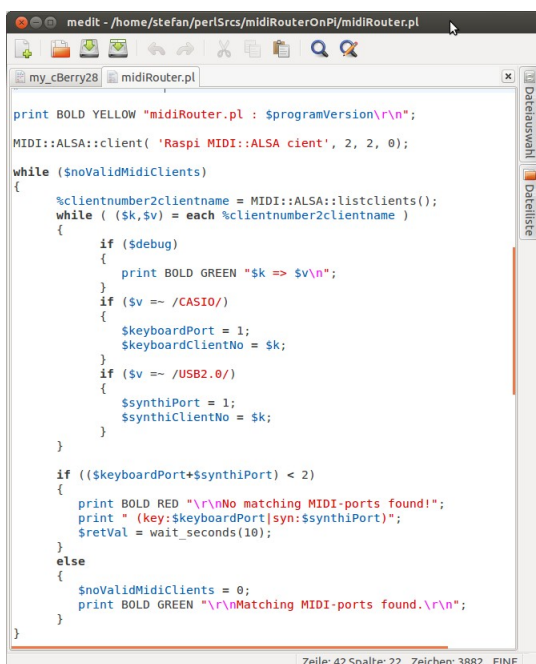
case "$1" in
  start)
    echo "Starts the Display-Driver"
    sudo /home/pi/cBerry28Display/SW/fb2cberry28/fb2cberry28 2>&1 &
    ;;
  stop)
    echo "Stops the Display-Driver"
    sudo killall fb2cberry28
    ;;
  *)
    echo "Usage: /etc/init.d/my_cBerry28 {start|stop}" >&2
    exit 2
    ;;
esac
exit 0

```

Abbildung 11: Inhalt der Datei "my\_cBerry28"

Über den Befehl „`sudo chmod a+x /etc/init.d/my_cBerry28`“ wurde die Datei ausführbar gemacht und über den Befehl „`sudo update-rc.d my_cBerry28 defaults`“ wurde sie in die Liste der Bootskripte eingefügt. Nach einem erneuten Reboot des Raspberry Pi sollte jetzt bereits der Bootvorgang auf dem Grafikdisplay angezeigt werden.

## Anpassung von midi-router.pl an den Einsatz mit dem Raspberry Pi



```

midiRouter.pl
print BOLD YELLOW "midiRouter.pl : $programVersion\r\n";
MIDI::ALSA::client( 'Raspi MIDI::ALSA client', 2, 2, 0);
while ($noValidMidiClients)
{
  %clientnumber2clientname = MIDI::ALSA::listclients();
  while ( ($k,$v) = each %clientnumber2clientname )
  {
    if ($debug)
    {
      print BOLD GREEN "$k => $v\r\n";
    }
    if ($v =~ /CASIO/)
    {
      $keyboardPort = 1;
      $keyboardClientNo = $k;
    }
    if ($v =~ /USB2.0/)
    {
      $synthiPort = 1;
      $synthiClientNo = $k;
    }
  }

  if (($keyboardPort+$synthiPort) < 2)
  {
    print BOLD RED "\r\nNo matching MIDI-ports found!";
    print " (key:$keyboardPort|syn:$synthiPort)";
    $retVal = wait_seconds(10);
  }
  else
  {
    $noValidMidiClients = 0;
    print BOLD GREEN "\r\nMatching MIDI-ports found.\r\n";
  }
}

```

Abbildung 12: Pi-spezifische Anpassung von midiRouter.pl

Das ursprüngliche Perlskript für den Büro-PC sah den Start innerhalb eines Terminalfensters vor. Das Skript war schon dahingehend ausgelegt in eine Endlosschleife zu laufen, die nur über das Stoppen des Perlinterpreters per „Strg+C“-Tastaturkombination angehalten werden konnte. Da am Raspberry Pi keine Tastatur vorgesehen war, musste das Skript automatisch am Ende des Bootvorganges gestartet werden. Außerdem konnte die eigentliche MIDI-Umsetzung im Skript erst dann erfolgen, wenn sowohl das Casio Celviano als auch das MIDI-Interface an den USB-Anschlüssen des Raspberry Pi angeschlossen waren. Im Skript des Büro-PC wurde beim Nichtfinden der Schnittstellen das Skript einfach abgebrochen, für den Einsatz im Pi wurde es dahingehend abgeändert, dass jetzt alle 10s wieder nach den Interfaces gesucht wird und das Skript erst aus dieser Suchschleife aussteigt und die MIDI-Übersetzung startet, wenn alle Interfaces gefunden worden sind. In Abbildung 12 ist der angepasste Teil des Quelltextes des Perl-Skriptes „midiRouter.pl“ zu sehen.

Um dafür zu sorgen, dass das Perl-Skript „midiRouter.pl“ am Ende des Bootvorganges automatisch aufgerufen wird, muss auf dem Pi noch die Datei „/etc/rc.local“, wie in Abbildung 13 gezeigt, geändert werden.

```

pi@raspberrypi:
GNU nano 2.2.6      Datei: /etc/rc.local

#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
#
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
printf "My IP address is %s\n" "$_IP"
fi
perl /home/pi/midiPi/midiRouter.pl
exit 0

```

Abbildung 13: Angepasste Datei "rc.local"

Nach einem Reboot des Pi und weder dem CASIO Celviano noch dem MIDI-Interface am Raspberry angeschlossen sollte sich das in Abbildung 14 gezeigte Bild auf dem Grafikdisplay ergeben.

Wie man dort sehen kann, wird zuerst das USB-MIDI-Interface angeschlossen (In der Zeile „No matching MIDI-ports found!“ wechselt zuerst syn von 0 auf den Wert 1).



Abbildung 14: Bootscreen MIDI-Router (keine angeschlossenen MIDI-Geräte)

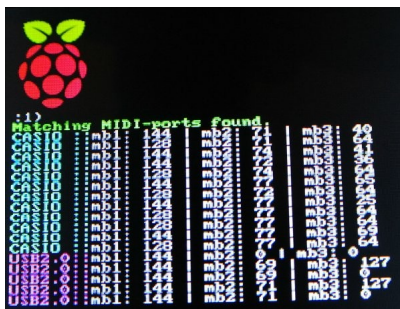


Abbildung 15: Applikationsscreen MIDI-Router

Sind beide Interfaces angeschlossen und erkannt, wechselt die Anzeige nach „Matching MIDI-ports found“.

Werden dann noch einzelne Noten auf dem Casio gespielt werden diese in Zeilen mit der Farbe „Cyan“ startend gezeigt, während Noten über das MIDI-Interface eingespielt in Zeilen mit der Farbe „Magenta“ startend angezeigt. All das kann man in Abbildung 15 sehen.

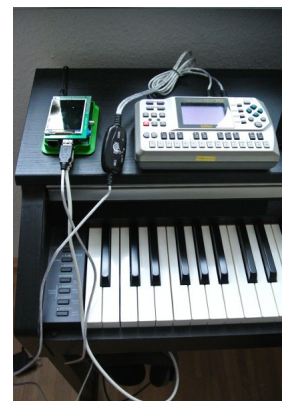


Abbildung 16: Mission Accomplished

Der fertig verkabelte Aufbau im Einsatz ist in Abbildung 16 zu sehen.

## Literaturverzeichnis

- 1: Peter Billam, MIDI-ALSA-1.20,
- 2: Simon Monk, Raspberry Pi Kochbuch, 2014