Kleinigkeiten zum ManniRaspiRadio

Netzwerk

Der Kleine kann sowohl WLAN wie auch Wired. Ein WLAN-Adapter ist schon drin. Er macht DHCP. SSH ist aktiv.

Zugangsdaten

Nutzer: pi
Passwort: raspberry

Web Server

Der Kleine hat einen Web Server und ist darüber steuerbar.
http://<IP-Adresse>
Außerdem gibt es Steuer-Apps für iOS und Android.

Shutdown

Tricky! Der Netzschalter wird nach einschalten durch ein Relais gesteuert vom Arduino überbrückt. Am Schalter ist mechanisch auch ein Input des Arduino gekoppelt. Wenn der Netzschalter ausgeschaltet wird, sendet der Arduino ein shutdown-Signal an der RasPi (GPIO #19) und wartet darauf, dass der sich abmeldet (GPIO #26). Dann schaltet der Arduino das Relais aus.

Konfiguration des Shutdown-Script in RaspBMC

RaspBMC hat die Bibliothek **`RPi.GPIO`** normal nicht an Bord, sie muss nachinstalliert werden. Dazu wird **`Python PIP`** gebraucht.

```
sudo apt-get install python-pip
sudo pip install RPi.GPIO
```

Mit einem Script fährt der Arduino den RasPi herunter. Es kann im Home-Verzeichnis mit **`nano`** als Nutzer **`pi`** erzeugt werden. Das Script muss danach als **`root`** in **`/usr/local/bin`** abgelegt werden. Weiter müssen die Zugriffsrechte und Besitz eingestellt werden:

```
sudo mv shutdown.py /usr/local/bin/shutdown.py
sudo chown root:root /usr/local/bin/shutdown.py
sudo chmod 0755 /usr/local/bin/shutdown.py
```

Folgendes Script lässt den RasPi runterfahren (Python):

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#
# Creation:    09.06.2013
# Last Update: 07.04.2015
#
# Copyright (c) 2013-2015 by Georg Kainzbauer <http://www.gtkdb.de>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
```

```
#

# import required modules
import time
import subprocess
import RPi.GPIO as GPIO

# set GPIO pin with connected button
DownPin = 26
LEDPin = 19
# main function
def main():
	value = 0

	while True:
		# Hochzählen, wenn Null
		if not GPIO.input(DownPin):
			value += 0.5
		else:
			value = 0

		# shutdown gewählt wenn value >= 3
		if value >= 3:
			subprocess.call(["shutdown", "-h", "now"])
			#print "Jetzt Shutdown", value
			return 0
		# wait 500ms
		time.sleep(0.5)

	return 0

if __name__ == '__main__':
	# use GPIO pin numbering convention
	GPIO.setmode(GPIO.BCM)

	# setze GPIO pin DOWNPin als input mit Pull-Up
	GPIO.setup(DownPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
	# setze LEDPin als Output (Shutdown fertig wenn low)
	GPIO.setup(LEDPin, GPIO.OUT)
	GPIO.output(LEDPin, GPIO.HIGH)
	# call main function
	main()
```

Es muss bei Systemstart als Daemon gestartet werden. Da RaspBMC die Datei **`rc.local`** beim Start nicht ausführt, habe ich folgenden Workaround entwickelt:

Im Verzeichnis **`/etc/init`** muss die Datei **`shutdown.conf`** erzeugt werden:

```
# shutdown.conf
#
# This task is run on startup to prepare the system for an external shutdown
# command
# c ottO 2015

description     "Bereitet shutdown durch arduino vor"

start on startup

task
exec /usr/local/bin/shutdown.py &
```

Nach einem Reboot mit **`sudo reboot`** ist der Arduino-Shutdown betriebsbereit.

Achtung: Die Eingänge des RasPi vertragen keine 5V! Daher wird der Eingang über eine Diode auf Null gezogen.

Display

Das Display hat eine Auflösung von 1024x600 und wird von RaspBMC nativ nicht unterstützt. Um das einzuschalten, muss die Datei **/boot/config.txt** angepasst werden. So soll die dann aussehen:

```
# uncomment to force a console size. By default it will be display's size minus
# overscan.
framebuffer_width=1024
framebuffer_height=600

hdmi_force_hotplug=1
#hdmi_ignore_edid=0xa5000080
hdmi_cvt=1024 600 60 3 0 0 0
```

Um das Blanking des Displays zu verhindern, muss die Datei **/etc/kbd/config** an folgender Stelle geändert werden:

```
#  **** screen saver/DPMS settings: all VCs ****
# These settings are commented by default to avoid the chance of damage to
# very old monitors that don't support DPMS signalling.

# screen blanking timeout.  monitor remains on, but the screen is cleared to
# range: 0-60 min (0==never)  kernels I've looked at default to 10 minutes.
# (see linux/drivers/char/console.c)
BLANK_TIME=0
```

Audio

In der Hoffnung, ein besseres Audio zu bekommen, wird der Audio-Ausgang des Diplays statt des RasPi verwendet. Es wird für den Vorverstärker eine China-Baugruppe mit LM1036 (DC-gesteuert) und einer China-2 x 10W-Class D-Endstufe mit PAM8610 genutzt. Die Lichtorgel basiert auf einem Grafik-Analyser MSGEQ7.

Radio

Wenn das Ding schon ManniRasPiRadio heißt, dann muss es auch ein Radio haben. Und zwar ein I2C-gesteuertes auf Basis RRD102. Es wird aber nicht wie ein normales Radio abgestimmt, sondern kennt nur ein paar fest programmierte Stationen, die ich aus einer Senderliste für Schiffdorf rausgesucht habe. Hoffentlich klappt’s.

Eingabe

Es gibt zwei Quellen für die Eingabe (außer der Maus…): Die alte Tastatur und die alten DrehKos. An allen Tasten sind mechanisch kleine Schalter gekoppelt, die über ein Schieberegister in den Arduino eingelesen werden. Bisher sind nur drei Tasten wirklich aktiv:

- UKW
  schaltet das UKW-Radio ein
- LW,MW,KW1,KW2
  schalten auf den RasPi
- B
  schaltet den Bass Boost für das Radio ein

Nachdem auf dem Arduino noch rund die Hälfte Platz ist, könnte mit den Schaltern noch ein wenig angestellt werden. Ich hatte nur keine Lust mehr…

Arduino

Nachdem ja auf dem Arduino einiges los ist, habe ich einen Mini-Scheduler gesucht und gefunden. Das Ding heißt Scoop. Es laufen insgesamt fünf Tasks im kooperativen Multitasking.

Daneben kommen Libraries für

- WS2801 von Adafruit für den LED-Streifen
- Frequenzmesser für die DrehKos
- Radio für das UKW-Radio

zum Einsatz. Für den Rest siehe die Quellen.