Entwicklung von Anwendungen mit dem RedPitaya

Paul Hill

28.05.2016

Inhaltsverzeichnis

1 Vorwort und Überblick

Diese Dokumentation soll den Einstieg in das Entwickeln von Anwendungen, insbesondere von Web-Anwendungen für den RedPitaya, erleichtern. Sie bezieht sich auf die OS-Version 0.95 (April 16), einige Dinge mögen sich mit kommenden Versionen also ändern. Neben C, kann der RedPitaya auch über andere Wege "programmiert" werden (zb Matlab), auf welche ich hier aber nicht näher eingehen werde.
Wenn Sie einen Fehler gefunden oder sonstige Fragen oder Vorschläge haben, können Sie mich gerne unter paul.hill@stud.uni-heidelberg.de kontaktieren.

Für einen guten Überblick sei zunächst auf die RedPitaya-Wiki in der aktuellen[1] und der alten Version[2] verwiesen. In letzterer finden sich ein Benutzerhandbuch und eine Entwicklerdokumentation, die als erster Einstieg in die Materie genutzt werden sollten.

2 Installation und Vorbereitung der Entwicklungsumgebung

Nachdem die SD-Karte wie in der Wiki beschrieben vorbereitet wurde, ist der RedPitaya bereits einsatzbereit.
Möchte man selber Anwendungen entwickeln, müssen jedoch zunächst noch einige Vorbereitungen getroffen werden. Der Aufbau der Entwicklungsumgebung ist vom verwendeten Betriebssystem des Arbeitsrechners abhängig, eine Anleitung findet sich auch hier in der Wiki.

Als erster Schritt sollte die Git-Repository[3] des Red-Pitaya Projektes heruntergeladen werden, diese enthält alle notwendigen Quelldateien der bestehenden Anwendung (samt dem NGINX-Modul und dem Verilog-Projekt für den FPGA) und Beispiele, die für die Entwicklung benötigt werden. Im Folgenden müssen noch diverse Software-Pakete (Linaro Toolchain, Xilinx, ..) installiert werden. Hier kann nach der Anleitung der Wiki vorgegangen werden, es sollten aber unbedingt auch die entprechenden README-Dateien aus der Repository gelesen werden! Bei Abweichungen sollte man den READMEs folgen, da die Informationen in der Wiki womöglich veraltet sind. Achten Sie auch darauf, dass Sie die richtige Version des Cross-Compilers installieren, denn diese muss zum benutzten RedPitaya passen (man unterscheidet zwischen hardfloated "hf" und softfloated). An dieser Stelle sei darauf hingewiesen, dass es beim Vorbereiten der IDE zu Problemen (d.h insbesondere zu Fehlern bei der Kompilierung) kommen kann, die den Prozess stark hinauszögern können. In solchen Fällen sollte man sich vergewissern, streng nach den Readme-Dateien vorgegangen zu sein und notfalls noch einmal von vorne beginnen.

Kommt man nicht weiter, ist als generelle Anlaufstelle das RedPitaya Forum[4] zu nen-

[1] http://wiki.redpitaya.com/index.php?title=Main_Page
[2] http://wiki.redpitaya.com/index.php/OLD_OS_Main_Page
[3] https://github.com/RedPitaya/RedPitaya
[4] http://forum.redpitaya.com/

nen, in dem man kompetente Hilfe findet.

Da die Vorbereitung der Entwicklungsumgebung mitunter mühselig ist, kann bei kleineren Projekten auch direkt auf dem RedPitaya gearbeitet werden. Per ssh-Verbindung können Quelldateien hier direkt mit gcc kompiliert werden. Eine ausführlichere Beschreibung findet sich in der Wiki.

3 Die RedPitaya Api

Die standardmäßig auf dem RedPitaya enthaltenen Webanwendungen kommunizieren über in C geschriebene Module direkt mit dem FPGA, da bei deren Entwicklung noch keine Api bereitstand. Mittlerweile exisitiert allerdings eine solche Api, die diese Kommunikation abstrahiert und den Zugriff auf die auf dem FPGA implementierten Funktionalitäten erheblich erleichtert (für die Funktionalität der PIDs muss aber nach wie vor auf ein entsprechendes C-Modul zurückgegriffen werden, welches jedoch sehr überschaubar ist). Für die Api exisitiert eine Dokumentation[5] und es bestehen einige Beispiele in der Repository ("Examples/C").
Die Beispiele und die Dokumenation sind an sich sehr instruktiv, deswegen soll an dieser Stelle nicht allzu sehr auf deren Funktionalität eingangen werden. Es sei doch auch darauf hingewiesen, dass es an manchen Stellen Unstimmigkeiten in der Dokumenation und den Beispielen gibt. Bei Schwierigkeiten sollte man sich den Quellcode ("api/rpbase/src/") der API direkt anschauen (notfalls auch den Verilog-Code).

Programme, die die Api benutzen, müssen, wenn sie in einer Shell aufgerufen werden, mit 'LD_LIBRARY_PATH=/opt/redpitaya/lib' gestartet werden.

3.1 Oszilloskop-Funktion

Insbesondere das Acquirieren von Samples über den (Fast-) ADC, stellte sich als schwieriger heraus, weswegen dies nun genauer betrachtet werden soll.
Schauen Sie sich dazu das Programm "acq_api_test" an. In dem gleichnamigen Projektordner (in meinem Archiv) liegt eine Txt-Datei mit einer Ausgabe des Programms. Beide Dateien finden Sie auch im Anhang.

Intern werden die gelesenen Samples in einem zirkulären Buffer (je Channel) der Länge 16384 gespeichert. Der Index des aktuellsten Samples steht in dem sogenannten "Write Pointer". Die Samples hinter dem Write Pointer sind am ältesten und werden als nächstes überschrieben werden.
Mit dem Befehl 'rp_AcqStart()' wird mit dem Einlesen von Samples begonnen. Wird nun eine Triggerquelle mit 'rp_AcqSetTriggerSrc(..)' gesetzt, ist das "Oszilloskop" scharf

[5] http://libdoc.redpitaya.com/rp_8h.html

gestellt. Wird der Trigger ausgelöst, wechselt der Triggerstatus zu “triggered“. Nachdem die zuvor mit ’rp_AcqSetTriggerDelay(..)’ angebene Samplezahl (+interner Offset!) acquiriert wurde, wird die Triggerquelle auf “disabled“ gesetzt und der Triggerstatus wechselt wieder zu “waiting“. Nun werden keine weiteren Samples mehr eingelesen und es kann mit dem Auslesen der Daten begonnen werden. Ob man den internen Offset für das Triggerdelay umgehen sollte, hängt von der Art der Anwendung ab. Setzt man das Triggerdelay auf 0, sorgt der interne Offset (von + 16384/2) dafür, dass genauso viele “Pre-Trigger-“ wie “Post-Trigger-Samples“ zur Verfügung stehen. Dies kann zB bei einer Oszilloskop ähnlichen App von Nutzen sein.
Es ist wissenswert, dass ’rp_AcqReset()’ das Einlesen von Samples nicht beendet (es werden nur die gesetzten Parameter zurückgesetzt)! Im obigen Programm würde ein Aufruf von ’rp_AcqStart()’ am Ende der main-Funktion dazu führen, dass bei einem erneuten Programmstart bereits vor ’rp_AcqStart()’ Samples acquiriert werden würden.
Desweiteren kann es, je nach verwendeter Triggerquelle (zb “AWG“), Probleme mit dem Triggern geben. Dabei mag es helfen, einen ’sleep’-Befehl zwischen ’rp_AcqStart()’ und ’rp_AcqSetTriggerSrc()’ zu setzen (siehe z.B. “Laser Stabilizer“-App).

3.2 Kompilierung (Ubuntu)

Zum Kompilieren der App werden Makefiles verwendet. Beispiele dafür finden sich in den Projektordnern in “Examples/C/“ in der Repository. Eine typische Makefile sieht dann bspw. so aus:

Listing 1: Makefile

```
#Cross compiler definition
CC = $(CROSS_COMPILE)gcc

CFLAGS  = -std=gnu99 -Wall -Werror -g $(INCLUDE)
CFLAGS += -I../../api/include
CFLAGS += -L ../../api/lib -lm -lpthread -lrp


OBJS=main.c fpga_pid.c pid.c

all:
        $(CC) $(CFLAGS) $(OBJS) -o lock

clean:
        $(RM) *.o
        $(RM) lock
```

Im jeweiligen Verzeichnis kann das Programm dann im Terminal mit
’make CROSS_COMPILE=arm-linux-gnueabihf- all’ (hardfloated Version) kompiliert werden. Hat man zuvor das Skript “settings.sh“ (Repository) ausgeführt, reicht der Befehl ’make all’ aus. Vor dem erneuten Kompilieren kann es notwendig sein, zuerst alle zuvor bei der Kompilierung erzeugten Datein zu löschen (’make clean’).

4 Web-Apps

Eine wichtige Funktionalität des RedPitayas sind die sogenannten Web-Apps:
Eine Web-Oberfläche, die über einen Browser zugänglich ist, dient als graphische Schnittstelle zum Nutzer, sodass dieser auf sehr einfachem Wege Einstellungen/Befehle an der laufenden Anwendung vornehmen kann und direkt eine visuelle Darstellung der generierten Daten erhält. Dazu läuft auf dem RedPitaya der modularisierbare Web-Server NGINX. Das entsprechende Modul für die Webanwendungen ist in C geschrieben und findet sich hier[6]. So werden HTTP-Anfragen spezieller Art vom Web-Server nicht "wie gewohnt" verarbeitet, sondern über das Modul an die derzeit laufende Web-Anwendung weitergeleitet. Selbige besteht dann aus einem serverseitigen Programm (ebenfalls in C), das einige Funktionen ("entry points") implementieren und als shared-object Datei kompiliert werden muss. Innerhalb dieses C-Programms kann dann wie gewohnt (zB mit der Api) gearbeitet werden.
Der Sourcecode der bereits existierenden Webanwendungen findet sich in der Repository unter "apps-free" in den jeweiligen Projektordnern.
Auch die Wiki bietet einen Überblick über Web-Apps[7].
Für die Entwicklung kann es sehr hilfreich sein, den Traffic der Anwendungsoberfläche einsehen zu können. Eine solche Funktionalität sollte in nahezu jedem Browser vorhanden sein.
Eine sehr simple "Hello World!"-Anwendung findet sich im Ordner "web-apps/test" in meinem Archiv.

4.1 Aufbau, benötigte Dateien und Installation

Eine Web-App besteht zunächst aus einem Ordner mit einem eindeutigen Namen, der App-Id (zb "pid+gen"). Dazu werden mindestens fünf weitere Dateien benötigt:

- fpga.conf ..Verweis auf die zu verwendende FPGA-Datei. Falls keine eigene verwendet werden soll, zb '/opt/redpitaya/fpga/fpga_0.94.bit' (Version!)
- index.html ..Die HTML-Datei für die Weboberfläche
- controllerhf.so ..Die eigentliche Anwendungsdatei, die vom Webserver angesprochen wird (Version für hardfloated Gerät)
- info/icon.png ..Ein Icon für die Weboberfläche
- info/info.json ..Enthält Metainformation im JSON-Format, zb:

```
{
    "name": "Laser Stabilizer",
    "version": "0.94-BUILD_NUMBER",
    "revision": "REVISION",
    "description": "Scan & PID Controller for laser stabilization."
}
```

[6] https://github.com/RedPitaya/RedPitaya/tree/master/Bazaar/nginx/ngx_ext_modules/ngx_http_rp_module
[7] http://wiki.redpitaya.com/index.php?title=Red_Pitaya_WEB_application_structure_detailed_overview

Es können natürlich auch zusätzliche Dateien (zb css, js, ...) verwendet werden.
Zum Installieren der App muss der Ordner mit den obigen Dateien in das Verzeichnis "/opt/redpitaya/www/apps/" des RedPitayas kopiert werden (gegebenenfalls muss der Ordner 'appId' erst noch erstellt werden).

4.2 Controller-Datei und Kommunikation zwischen Server- und Clientseite

Auf der Clientseite können mittels Ajax HTTP GET- und POST-Anfragen an den Server gestellt werden. Bei entsprechender URL, werden sie von NGINX über das NGINX-RedPitaya-Modul an unsere controller-Datei weitergegeben. Es stehen im wesentlichen die folgenden vier Anfragen zur Verfügung ('root_url ' ist der jeweilige Hostname des RedPitayas):

- root_url + '/bazaar?start=' + app_id ..Initialisiert die Anwendung, app_id muss gleich dem Namen des Projektordners sein
- root_url + '/bazaar?start=' ..Stoppt die Anwendung
- POST root_url + '/data' ..Zum Senden der aktuellen Parametereinstellungen. Die Antwort enthält die aktuellen, serverseitigen Parameter. Die Daten werden dabei im JSON-Format übertragen. Eine typische Antwort kann bspw. so aussehen:

```
{
  "app":{
          "id":"stabilizer"
  },
  "datasets":{
          "params":{
                  "op_mode":0,
                  "xmin":25,
                  "xmax":75,
                  "trig_delay":1145.469727,
                  ...
          }
  },
  "status":"OK"
}
```

- GET root_url + '/data' ..Fordert den Server dazu auf, die aktuellen "Signale" und Parameter zu schicken. Die Antwort enthält entsprechende Signale und Parameter. Bei den Signalen handelt es sich um zwei zweidimensionale Arrays von jeweils max. 2048 Elementen. Üblicherweise enthält jedes der Arrays die Messergebnisse für je einen der Analogeingänge, wobei ein Messpunkt aus einem Zeit- und Spannungswert besteht. Die Signale werden ebenfalls im JSON-Format beschrieben, eine typische Antwort könnte so aussehen:

```
{
  "app":{
          "id":"stabilizer"
  },
```

```
    "datasets":{
            "g1":[{"data":[[0.0000,-0.0247],[25.0655,-0.0248],...]},
                {"data":[[0.0000,-0.0159],[25.0655,-0.0159],...]}],
            "params":...
    }
    "status":"OK"
  }
```

Intern (NGINX-RP-Modul) spielt die folgende Struktur eine zentrale Rolle:

```
typedef struct rp_app_params_s {
    char        *name;
    float       value;
    int         fpga_update;
    int         read_only;
    float       min_val;
    float       max_val;
} rp_app_params_t;
```

Diese Struktur repräsentiert genau einen Parameter, wobei nur die ersten beiden Variablen für die Kommunikation mit der Clientseite verwendet werden. Die restlichen vier Variablen sind für den Gebrauch innerhalb der Controller-Datei reserviert. Die aus der JSON-Anfrage geparsten Parameter werden zur weiteren Verwendung in einem Array gespeichert. Z.B.:

```
rp_app_params_t rp_main_params[..];
```

In der Controller-Datei muss die Struktur erneut definiert werden, da die Definition aus dem NGINX-RP-Modul hier nicht bekannt ist. Das ermöglicht gewisse Freiheiten bei den Variablentypen und Namen, denn Controller-Datei und NGINX-Module tauschen im Prinzip nur einen Void-Zeiger auf obiges Array miteinander aus, interpretieren (casten) aber alles Weitere mit ihrer lokalen Definition der Struktur. (Mindestens) die Gesamtgröße der Struktur in Byte muss in beiden Definitionen also zwingend übereinstimmen.

Die Controller-Datei muss neben der Parameter-Strukur einige Funktionen implementieren, die als Schnittstelle zwischen dem eigentlichen Programm und dem NGINX-Modul fungieren:

- int rp_app_init(void) ..wird aufgerufen, wenn Start-URL aufgerufen wird. Bei einem Fehler sollte -1, ansonsten 0 zurückgegeben werden.
- int rp_app_exit(void) ..wird aufgerufen, wenn Stop-URL aufgerufen wird. Bei einem Fehler sollte -1, ansonsten 0 zurückgegeben werden.
- int rp_set_params(rp_app_params_t *p, int len) ..wird aufgerufen, wenn Parameter an den Server gesendet wurden. p ist der Pointer auf das Parameter-Array und len dessen Länge. Bei einem Fehler sollte -1, ansonsten 0 zurückgegeben werden.
- int rp_get_params(rp_app_params_t **p) ..wird vom NGINX-Modul aufgerufen, wenn aktuelle Parameter gesendet werden sollen. p ist ein Zeiger auf ein Array,

das innerhalb dieser Funktion allokiert werden muss (Kopie des lokalen Parameter-Arrays). Es muss ein Element mehr allokiert werden, dessen 'name'-Variable auf 'NULL' gesetzt werden muss (markiert Ende des Arrays). Der Rückgabewert sollte der Anzahl an Parametern entsprechen. Außerdem versucht das NGNIX-Modul anschließend das 'char* name'-Array jedes Parameters in *p zu deallokieren. Damit das nicht fehlschlägt, muss jedes dieser Arrays beim Kopieren der lokalen Parameter mitallokiert werden.

- int rp_get_signals(float ***s, int *sig_num, int *sig_len) ..wird aufgerufen, wenn aktuelle Signale angefordert werden. s ist ein Zeiger auf ein 3D-Array. Die drei Arrays enthalten typischerweise die Messwerte für die beiden Eingänge, sowie ein "Zeitarray" mit den entsprechenden Zeitpunkten. Innerhalb des NGINX-Moduls werden die 3 Arrays dann zu zwei 2D-Arrays zusammengefügt. Die Arrays sind bereits allokiert. Mit sig_len kann die Länge der Arrays angegeben werden (max 2048). sig_num ist bedeutungslos, da immer 3. Über den Rückgabewert kann Folgendes mitgeteilt werden:
  -2 ..die Signale sind alt → die Clientseite kann darauf reagieren
  -1 ..die Signale sind noch nicht fertig verarbeitet → Das NGINX-Modul wiederholt den Aufruf der Funktion nach kurzer Zeit erneut (max. 200-mal mit 1ms Abstand)
  0 ..OK

- const char *rp_app_desc(void) ..wird eigentlich gar nicht aufgerufen

4.2.1 Kompilierung

Zur Kompilierung werden analog zur Api Makefiles verwendet. Beispiele finden sich in der Repository im Verzeichnis "apps-free" im jeweiligen Projektordner. In der Regel liegen die Quelldateien dort in dem Unterordner "src/" , der ein zweites Makefile enthält. Die beiden Makefiles könnten bspw. so aussehen (diese Web-App benutzt auch die Api):

Listing 2: Makefile 1

```
#
# $Id: Makefile 1235 2014-02-21 16:44:10Z ales.bardorfer $
#
# Red Pitaya specific application Makefile.
#

APP=$(notdir $(CURDIR:%/=%))

# Versioning system
BUILD_NUMBER ?= 0
REVISION ?= devbuild
VER:=$(shell cat info/info.json | grep version | sed -e 's/.*:\ *\"//' |
    sed -e 's/-.*//')

INSTALL_DIR ?= ../../build

CONTROLLERHF = controllerhf.so
```

```
CFLAGS += -DVERSION=$(VER)-$(BUILD_NUMBER) -DREVISION=$(REVISION)
export CFLAGS

all: $(CONTROLLERHF)

$(CONTROLLERHF):
        $(MAKE) -C src

clean:
        $(MAKE) -C src clean
        -$(RM) target -rf
        -$(RM) *.so
```

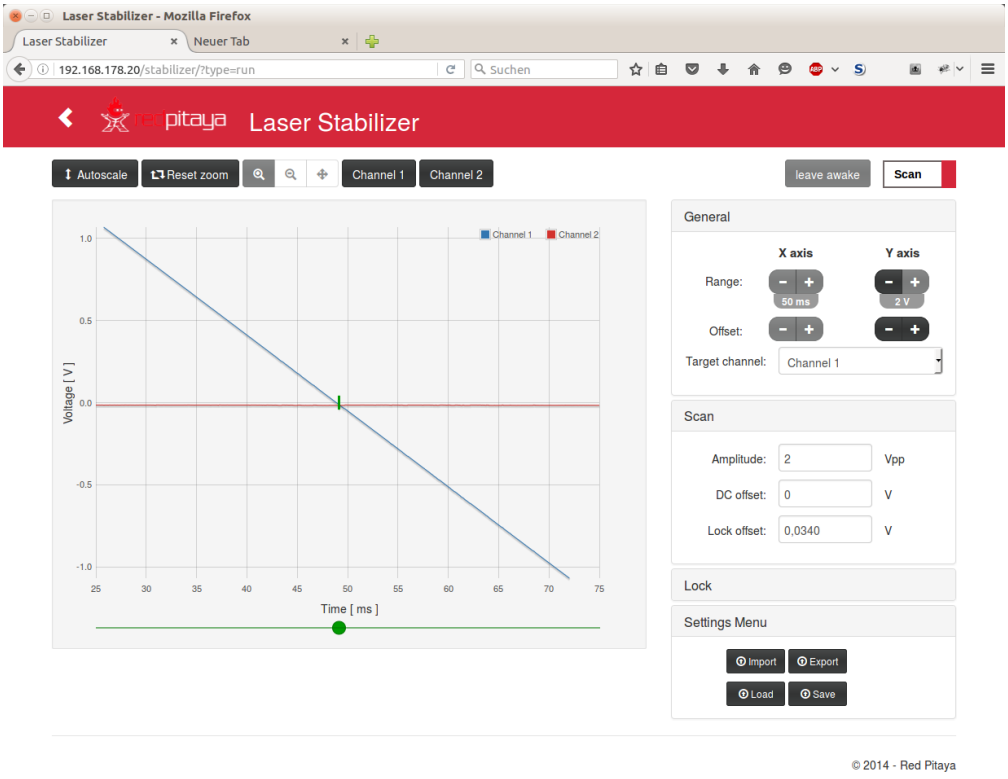
Listing 3: Makefile 2 in Unterordner “src“

```
CC=$(CROSS_COMPILE)gcc
RM=rm

OBJECTS=main.o worker.o pid.o fpga_pid.o

CFLAGS+= -Wall -Werror -g -fPIC $(INCLUDE)
CFLAGS += -I../../../../api/include
CFLAGS += -L ../../../../api/lib -lm -lpthread -lrp
LDFLAGS=-shared

CONTROLLER = ../controllerhf.so

all: $(CONTROLLER)

$(CONTROLLER): $(OBJECTS)
        $(CC) -o $(CONTROLLER) $(OBJECTS) $(CFLAGS) $(LDFLAGS)

clean:
        -$(RM) -f $(OBJECTS)
```

4.3 Debug Ausgabe

Da Web-Apps nicht im Terminal ausgeführt werden, steht auch keine direkte Ausgabemöglichkeit zur Verfügung. Debuginformationen müssen dazu in ein Log-File geschrieben werden: ’fprintf(stderr, ”debug info”);’. Die entsprechende Log-Datei liegt in dem Verzeichnis “/var/log/redpitaya_nginx/ “ .

5 Laser Stabilizer App



Screenshot der Weboberfläche der Stabilizer App im **Scan Mode**

Im **Scan-Mode** generiert die Stabilizer App ein 10Hz Dreieckssignal (Scansignal) variabler **Amplitude** an dem **Target-Channel** und misst die anliegenden Spannungen (Fehlersignale) an den beiden Input-Channeln über eine volle fallende Flanke. Mit einem Schieberegler kann nun eine Stelle im Fehlersignal markiert werden. Dabei wird der Zeitwert der x-Position in die dazu korrespondierende Spannung des Scansignals umgerechnet. Dieser **Lock-Offset** dient später als Spannungsoffset für die PIDs. Hat man eine passende Stelle im Fehlersignal gefunden und die PIDs konfiguriert, kann in den **Lock-Mode** umgeschaltet werden. Nun wird ein DC Offset mit Wert des **Lock-Offsets** an dem **Target-Channel** generiert und die PIDs gestartet. Im Lock-Mode können jetzt auch Änderungen an der Zeitbasis etc. vorgenommen werden. Außerdem lässt sich die App im **Lock-Mode** über die Schaltfläche **leave awake** ohne Beenden verlassen.
Im Menu **Settings** können die vorgenommenen Einstellungen exportiert oder auf dem RedPitaya gespeichert werden. Die gespeicherten Dateien werden dabei mit einem Erstellungsdatum versehen, sodass alte Dateien beim Speichern nicht überschrieben werden. Sie liegen auf dem RedPitaya im Verzeichnis “/var/opt/stabilizer/“ . Die dortige Da-

tei “settings_meta“ enthält den Namen der aktuellen Einstellungsdatei, sodass manuell durch Ändern der Meta-Datei alte Einstellungen wieder eingespielt werden können. Auch Importieren und Laden (der auf dem RedPitaya gespeicherten Daten) ist möglich.

Das gesamte Projekt findet sich unter “web-apps/stabilizer/“ in meinem Archiv.

6 Nützliches

6.1 Exportieren von Dateien auf den RedPitaya (Ubuntu)

Das Dateisystem des RedPitayas ist bis auf wenige Verzeichnisse wie “/var“ schreibgeschützt. Mit dem Befehl (per ssh ausführen) ’ro’ und ’rw’ wird das Dateisystem nicht-beschreibbar, bzw. beschreibbar. Auch sollte man bei Aktualisierungen innerhalb der Webanwendungen zuvor den WebServer mit ’systemctl stop redpitaya_nginx.service’ stoppen und anschließend wieder mit ’systemctl start redpitaya_nginx.service’ starten. Um mir das etwas zu vereinfachen, habe ich ein kurzes bash-Skript (für Linux) geschrieben:

Listing 4: rp_push_file.sh

```
#!/bin/bash
if [ $# -lt 4 ]
then
        echo "usage: rp_push_file password hostname destination source"
        exit
fi
SRC=$4
PSWD=$1
HOST=$2
DEST=$3
while [ $# -gt 4 ]
do
        SRC+=" "$5
        shift
done
echo "stop NGINX and make filesystem writable"
sshpass -p $PSWD ssh root@$HOST 'source /etc/profile; systemctl stop
    redpitaya_nginx.service; rw; exit'
echo "pushing "$SRC" to "$DEST
sshpass -p $PSWD scp $SRC root@$HOST:$DEST
echo "make filesystem read only and restart NGINX"
sshpass -p $PSWD ssh root@$HOST 'source /etc/profile; ro; systemctl start
     redpitaya_nginx.service; exit'
```

6.2 Schutz der laufenden Web-App vor versehentlichem Beenden

Öffnet ein zweiter Nutzer eine schon laufende App, so kommunizieren beide Nutzer mit der selben “Instanz“ des Controller-Programms.
Startet er jedoch eine andere App, führt dies zum sofortigen Beenden der laufenden

Anwendung. Im Falle von Anwendungen, die über längere Zeit laufen müssen, ist dies natürlich fatal. Leider zeigt das Hauptmenü der Weboberfläche des RedPitayas nicht an, ob eine, und wenn ja welche, Anwendung gerade läuft.

Um dieses Problem zu lösen, muss das Hauptmenü entsprechend angepasst werden, dessen Dateien im Verzeichnis "/opt/redpitaya/www/apps/" auf dem RedPitaya liegen. Insbesondere die Datei "index.html" und die darin verlinkten Javascript-Dateien müssen bearbeitet werden. Da die Weboberfläche sich für die verschiedenen Versionen des RedPitaya OS ändern kann, müssen die Änderungen bei einer neuen Versionen erneut und gegebenfalls abgändert vorgenommen werden. Glücklicherweise sollte folgendes Grundgerüst versionsunabhängig funktionieren:

1. Zunächst ist herauszufinden, welche und/oder ob eine App gerade läuft. Hierzu kann verwendet werden, dass eine laufende App auf eine Get-Anfrage an die Adresse 'root_url' + '/data' antwortet. Erhält man eine Antwort, ist unmittelbar einsehbar welche App gerade läuft, da die Antwort deren Id enthält. Dieses Verfahren funktioniert allerdings nicht für Pro-Apps...

2. Nun kann z.B durch Einblenden einer Warnung auf eine evtl. laufende App reagiert werden.

Wie gesagt, muss diese Methode versionsabhängig implementiert werden, einige Implementierungen finden sich allerdings bereits im Verzeichnis
"web-apps/prevent_user_from_killing_apps" meines Archivs.

7 Anhang

7.1 acq_api_test

Listing 5: acq_api_test

```
/*
 * main.c
 *
 *  Created on: 22.03.2016
 *      Author: Paul Hill
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <math.h>
#include "redpitaya/rp.h"

int main(){
        if(rp_Init() != RP_OK){
```

```
        fprintf(stderr, "Rp api init failed!\n");
        return 0;
}

printf("Be sure that OUT2 is connected to IN2.\n");
printf("trigger source:\n\t0: RP_TRIG_SRC_DISABLED\n\t4:
    RP_TRIG_SRC_CHB_PE\ntrigger state:\n\t%d:
    RP_TRIG_STATE_TRIGGERED\n\t%d: RP_TRIG_STATE_WAITING\n",
    RP_TRIG_STATE_TRIGGERED,RP_TRIG_STATE_WAITING);

uint32_t len = 2000;            //samples to be acquired after
    trigger

//prepare scope
rp_AcqReset();
rp_AcqSetDecimation(RP_DEC_64);
rp_AcqSetTriggerLevel(0);
rp_AcqSetTriggerDelay(len-ADC_BUFFER_SIZE/2);  //we want to
    acquire exactly 2000 samples after trigger event, so we have
    to bypass the internal offset

//generate 'triangle' on channel 2
rp_GenReset();
rp_GenFreq(RP_CH_2, 200.0);
rp_GenAmp(RP_CH_2, 1.0);
rp_GenWaveform(RP_CH_2, RP_WAVEFORM_TRIANGLE);
rp_GenOutEnable(RP_CH_2);


uint32_t pos, tr_pos;
rp_acq_trig_src_t src = 0;
rp_acq_trig_state_t state = 0;
uint32_t adclen = ADC_BUFFER_SIZE;                      //size of
     buffer

printf("lets look how the write pointer moves after rp_AcqReset()
    \n");
int k = 2;
while(k>0){
        usleep(10);
        --k;
        rp_AcqGetWritePointerAtTrig(&tr_pos);
        rp_AcqGetWritePointer(&pos);
        rp_AcqGetTriggerSrc(&src);
        rp_AcqGetTriggerState(&state);
        printf("\tpos: %d; pos when triggered: %d; trigger source
            : %d; trigger state: %u\n",pos, tr_pos, src, state);
}

printf("start acquiring samples\n");
rp_AcqStart();

printf("lets look how the write pointer moves\n");
k=3;
```

```
while(k>0){
        usleep(10);
        --k;
        rp_AcqGetWritePointerAtTrig(&tr_pos);
        rp_AcqGetWritePointer(&pos);
        rp_AcqGetTriggerSrc(&src);
        rp_AcqGetTriggerState(&state);
        printf("\twrite pointer: %d; write pointer at trigger: %d
            ; trigger source: %d; trigger state: %d\n",pos, tr_pos
            , src, state);
}

printf("arm scope\n");
rp_AcqSetTriggerSrc(RP_TRIG_SRC_CHB_PE);

printf("wait for trigger (0V posedge on Channel2)\n");
while(true){
        rp_AcqGetWritePointerAtTrig(&tr_pos);
        rp_AcqGetWritePointer(&pos);
        rp_AcqGetTriggerSrc(&src);
        rp_AcqGetTriggerState(&state);
        printf("\twrite pointer: %d; write pointer at trigger: %d
            ; trigger source: %d; trigger state: %d\n",pos, tr_pos
            , src, state);
        if(src==RP_TRIG_SRC_DISABLED)
                break;
                    //all samples are acquired
        usleep(1);
}
printf("scope is done with acquiring (2000) samples. Does the
    write pointer move?\n");

k=2;
while(k>0){
        usleep(10);
        --k;
        rp_AcqGetWritePointerAtTrig(&tr_pos);
        rp_AcqGetWritePointer(&pos);
        rp_AcqGetTriggerSrc(&src);
        rp_AcqGetTriggerState(&state);
        printf("\twrite pointer: %d; write pointer at trigger: %d
            ; trigger source: %d; trigger state: %d\n",pos, tr_pos
            , src, state);
}

//we can now read out the buffer

//latest samples
float *latest = (float *)malloc(len * sizeof(float));
rp_AcqGetLatestDataV(RP_CH_2, &len, latest);

//oldest samples
float *oldest = (float *)malloc(len * sizeof(float));
rp_AcqGetOldestDataV(RP_CH_2, &len, oldest);
```

```
        //all samples
        float *all = (float *)malloc(adclen * sizeof(float));
        rp_AcqGetDataV(RP_CH_2, 0, &adclen, all);


        //works only with small len. For large len it's likely to '
            overflow'
        printf("\nread out interesting part of buffer\ni\t\tdata\t\
            tlatest\t\toldest\n");
        int n;
        for(n=tr_pos-5;n<pos+len+6 && n<adclen;++n){
                if(n==tr_pos+7)
                        printf("...\n");
                if(n<pos-5 &&n>tr_pos+5)
                        continue;
                if(n==pos+len-4)
                        printf("...\n");
                if(n>pos+5 &&n<pos-4+len)
                        continue;
                if(n==pos)
                        printf("~~~~~~~~~\nwrite pointer\n");
                if(n==tr_pos)
                        printf("~~~~~~~~~\nwrite pointer at trigger\n");
                if(n<pos && n>pos-len-1)
                        printf("%d\t\t%f\t\t%f\t\t-\n",n, all[n],latest[n
                            -pos+len]);
                else if(n>pos && n<pos+1+len)
                        printf("%d\t\t%f\t\t-\t\t%f\n",n, all[n],oldest[n
                            -pos-1]);
                else
                        printf("%d\t\t%f\t\t-\t\t-\n",n, all[n]);
                if(n==pos || n==tr_pos)
                        printf("~~~~~~~~~\n");
        }
        free(latest);
        free(oldest);
        free(all);
        rp_Release();
}
```

Das Programm produzierte die folgende Ausgabe:

```
redpitaya> LD_LIBRARY_PATH=/opt/redpitaya/lib ./acq_api_test
Be sure that OUT2 is connected to IN2.
trigger source:
0: RP_TRIG_SRC_DISABLED
4: RP_TRIG_SRC_CHB_PE
trigger state:
0: RP_TRIG_STATE_TRIGGERED
1: RP_TRIG_STATE_WAITING
lets look how the write pointer moves after rp_AcqReset()
pos: 0; pos when triggered: 0; trigger source: 0; trigger state: 1
```

```
pos: 0; pos when triggered: 0; trigger source: 0; trigger state: 1
start acquiring samples
lets look how the write pointer moves
write pointer: 236; write pointer at trigger: 0; trigger source: 0; trigger state: 1
write pointer: 531; write pointer at trigger: 0; trigger source: 0; trigger state: 1
write pointer: 800; write pointer at trigger: 0; trigger source: 0; trigger state: 1
arm scope
wait for trigger (0V posedge on Channel2)
write pointer: 1047; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 1308; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 1553; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 1810; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 2074; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 2322; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 2604; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 2870; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 3134; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 3390; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 3652; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 3910; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 4168; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 4428; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 4686; write pointer at trigger: 0; trigger source: 4; trigger state: 1
write pointer: 4942; write pointer at trigger: 4779; trigger source: 4; trigger state: 0
write pointer: 5199; write pointer at trigger: 4779; trigger source: 4; trigger state: 0
write pointer: 5456; write pointer at trigger: 4779; trigger source: 4; trigger state: 0
write pointer: 5718; write pointer at trigger: 4779; trigger source: 4; trigger state: 0
write pointer: 5976; write pointer at trigger: 4779; trigger source: 4; trigger state: 0
write pointer: 6233; write pointer at trigger: 4779; trigger source: 4; trigger state: 0
write pointer: 6489; write pointer at trigger: 4779; trigger source: 4; trigger state: 0
write pointer: 6746; write pointer at trigger: 4779; trigger source: 4; trigger state: 0
write pointer: 6779; write pointer at trigger: 4779; trigger source: 0; trigger state: 1
scope is done with acquiring (2000) samples. Does the write pointer move?
write pointer: 6779; write pointer at trigger: 4779; trigger source: 0; trigger state: 1
write pointer: 6779; write pointer at trigger: 4779; trigger source: 0; trigger state: 1

read out interesting part of buffer
i data latest oldest
4774 -0.001733 --
4775 -0.001444 --
4776 -0.001300 --
4777 -0.000722 --
4778 -0.000433 --
~~~~~~~~
```

```
write pointer at trigger
4779 0.000289 0.000289 -
~~~~~~~~
4780 0.000722 0.000722 -
4781 0.001300 0.001300 -
4782 0.001733 0.001733 -
4783 0.002166 0.002166 -
4784 0.002744 0.002744 -
...
6774 0.957362 0.957362 -
6775 0.957939 0.957939 -
6776 0.958517 0.958517 -
6777 0.958806 0.958806 -
6778 0.959384 0.959384 -
~~~~~~~~
write pointer
6779 0.959673 --
~~~~~~~~
6780 0.938587 -0.938587
6781 0.939020 -0.939020
6782 0.939598 -0.939598
6783 0.939887 -0.939887
6784 0.940320 -0.940320
...
8775 0.374915 -0.374915
8776 0.374482 -0.374482
8777 0.374048 -0.374048
8778 0.373471 -0.373471
8779 0.372893 -0.372893
8780 0.372460 --
8781 0.372027 --
8782 0.371593 --
8783 0.371016 --
8784 0.370727 --
```