Red Pitaya scripting summary (Basics)

Interesting Links

http://redpitaya.readthedocs.io/en/latest/quickStart/quickStart.html

Pavel Denim's notes:
https://pavel-demin.github.io/red-pitaya-notes/

Alpine Linux:
https://alpinelinux.org/
https://wiki.alpinelinux.org/wiki/Main_Page

Pitaya in education:
http://red-pitaya-active-learning.readthedocs.io/en/latest/index.html

Pitaya Remote control:
http://redpitaya.readthedocs.io/en/latest/appsFeatures/remoteControl/remoteControl.html

Pitaya active learning (Pitaya for school):
http://red-pitaya-active-learning.readthedocs.io/en/latest/index.html

Preparing the SD card

http://redpitaya.readthedocs.io/en/latest/quickStart/SDcard/SDcard.html
https://pavel-demin.github.io/red-pitaya-notes/alpine/

In short (under Linux):

- Use df -h to see the available devices.

  The SD card should be listed as something similar to
  ```
  /dev/sde1         60G   32K   60G   1% /media/jcf/23D7-CD2E
  ```

- Unmount the SD card partition(s):
  ```
  sudo umount /dev/sde1
  ```

- Write the image with dd:
  ```
  sudo dd bs=1M if=red_pitaya_image_file.img of=/dev/sde
  ```

  Take care:
  - Doublecheck the partition name! (Don't overwrite your PC's hard disk!)
  - Use the name of the partition without the number at the end
    (/dev/sde instaed of /dev/sde1)

- Wait until the process has finished.
  To see the progress, you can use the option status=progress for dd.

- Use sync to force the writing of the buffer to the card, before removing the card.

Connect to the Pitaya

There are (more than) two possibilities:

- **by webbrowser**
  Use the address rp-xxxxxx (e.g. rp-f052a6) where xxxxxx are the last digits of the hex number noted on the pitaya board

  or

  Use the IP address of the pitaya.
  To find the address, use
  `sudo arp-scan -l`

- **by SSH**
  http://redpitaya.readthedocs.io/en/latest/developerGuide/os/ssh/ssh.html

  `ssh root@<IP address of Pitaya>`
  e.g.
  `ssh root@192.168.0.103`

  I had the idea of installing geany on the Pitaya to have a fine editor for writing python scripts and using ssh with the -X option to see it on the host computer. Forget it! The Pitaya (I could have thought of it) has no grafical interface on board, so it doesn't work.

Use Python scripts to do something useful

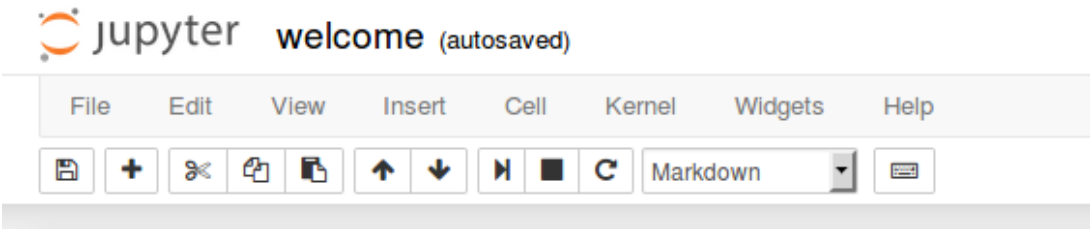
There are two possibilities:

- Use the Jupyter notebook to interactively execute code and immediately see the result.
  This is a good possibiliy for studying, as you can directly see the result.

- Use an editor (without grafical interface, like nano or mcedit !) on the Pitaya and write the scripts using SSH.
  I found it a good idea to use the midnight commander mc and it's editor.
  This can be installed on the Pitay via
  `apt-get install mc`

Hello world (Blink) via Jupyter

- Connect to the Pitaya via the web interface
- From the web applications, open Jupyter notebooks.
- In the Jupyter menu



select File – New Notebook Python 3

- Write this to the input cell

```
In [ ]: from redpitaya.overlay.mercury import mercury as FPGA
overlay = FPGA()

LED = FPGA.led
led1 = LED(3, 0)          # LED nr.3, initial state = 0

# blink 10 times
import time
for _ in range(10):
    led1.write(1)
    time.sleep(0.1)
    led1.write(0)
    time.sleep(0.1)

led1.close()
```

- Place the cursor into the cell and do <Shift><Enter> to execute the cell.
  After a short transmission time, the 3rd LED on the Pitaya should blink 10 times.

Hello world (Blink) via SSH

This hello world should be a blink program, written and executed on the Pitaya.
It is inspired from the Jupyter sample programs.

Let's suppose the IP address of the Pitaya is 192.168.0.103.

First we have to connect to the Pitaya, so, on the host computer, do in a terminal:

```
ssh root@192.168.0.103
```

```
root@192.168.0.103's password: root
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-xilinx armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
##############################################################################
##
# Red Pitaya GNU/Linux Ecosystem
# Version: 0.98
# Build: 617
# Branch:
```

```
# Commit: 1819a1d704b949065c09b981ee84cd912179a72c
# U-Boot: "redpitaya-v2016.4"
# Linux Kernel: "redpitaya-v2016.2-RP4"
# Pro Applications:  15fbd007cc7246e710f0b0f39f91d9f824a42f48 Applications
(v0.97-RC6-3-g15fbd00)
##############################################################################
##
Last login: Mon Oct 30 13:05:58 2017 from 192.168.0.102
root@rp-f052a6:~#
```

Now we are connected as root on the Pitaya.
Every command will be executed on the Pitaya, not on the host computer.

We are now in the folder /root.
I created a folder /home/jcf where my scripts shall reside:

```
root@rp-f052a6:~# ls
bin
root@rp-f052a6:~# cd ..
root@rp-f052a6:/# ls
bin  boot  buildlog.txt  dev  etc  home  lib  lost+found  media  mnt  opt
proc  root  run  sbin  srv  sys  tmp  usr  var
root@rp-f052a6:/# cd home
root@rp-f052a6:/home# ls
izi  jcf  jupyter  redpitaya
root@rp-f052a6:/home# cd jcf
root@rp-f052a6:/home/jcf#
```

If not done already, this is the moment to install mc:
sudo apt-get install mc

After that, mcedit can be used to write the script led.py:

```
from redpitaya.overlay.mercury import mercury as FPGA
overlay = FPGA()

LED = FPGA.led
led1 = LED(3, 0)        # LED nr.3, initial state = 0

# blink 10 times
import time
for _ in range(10):
    led1.write(1)
    time.sleep(0.1)
    led1.write(0)
    time.sleep(0.1)

led1.close()
```

The script can be executed by doing

```
python3 led.py
```

in the terminal.

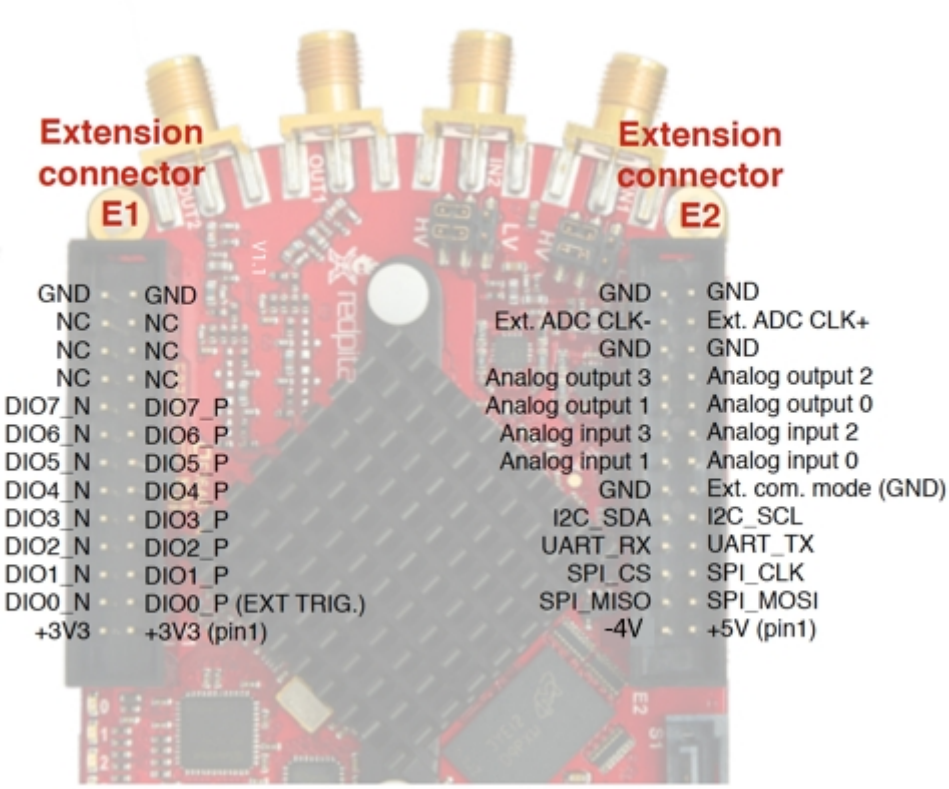
Take care to use Python 3 !
It takes a second or so to do the transfer, than the 3rd LED will blink 10 times on the Pitaya.

Tip:
To test a script, it is a good idea to use the Jupyter notebook, start a new Python file, write all the code in one cell, test it and only then transfer it to the Pitaya as Python script.

Digital IN and OUT (slow)

Using the GPIOs
There are 16 IO pins, managed by the linux subsystem.

Beware:
The digital High level is 3.3V, not TTL!



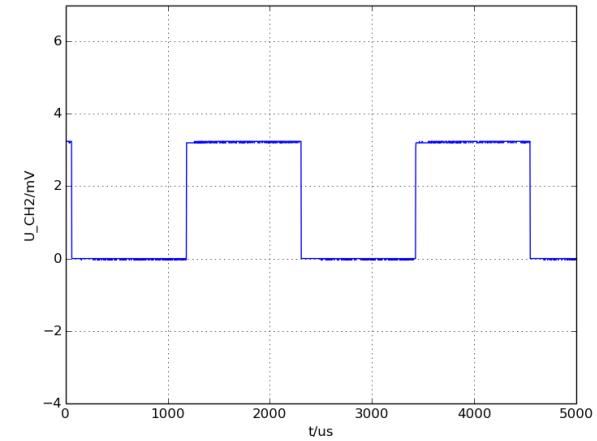
Reading a digital input in a loop:

```
# read_input.py
from redpitaya.overlay.mercury import mercury as FPGA
overlay = FPGA()
GPIO = FPGA.gpio
gpio_i = GPIO('n', 2, "in")         # DIO_N2

import time

while True:
    x=gpio_i.read()
    print(x)
    time.sleep(0.5)
```

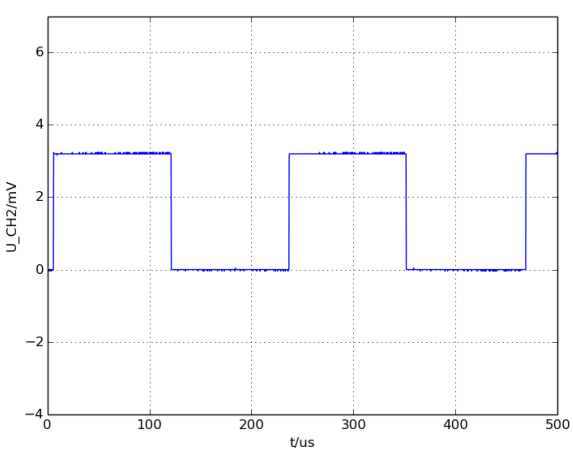
```
This will print the digital state of the input every 0.5s on the console
(supposing you are logged in by SSH and have started the script on the
Pitaya with  python3  read_input.py


Writing a digital output:

from redpitaya.overlay.mercury import mercury as FPGA
overlay = FPGA()
GPIO = FPGA.gpio
myout = GPIO('p', 0, "out")         # DIO_P0

import time
for i in range(0,10):
    myout.write(True)
    time.sleep(0.5)
    myout.write(False)
    time.sleep(0.5)
```

How fast is this?

I tried with different time intervals, and as expected the reaction time is rather slow:



dt = 1ms



dt = 1µs

There is a minimal latency of ca. 110µs, regardless of the time settings in the Python script.

So setting the outputs via Python is only useful for slow processes, from seconds down to the millisecond range.

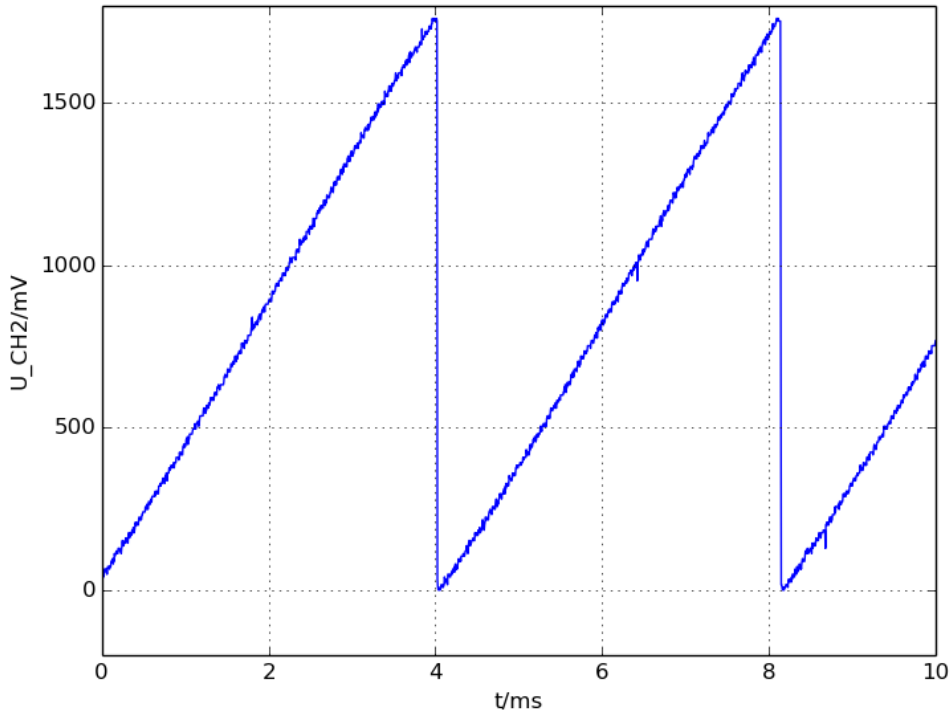
Slow analog output

```
There are 4 analog outputs with a voltage range from 0 to 1.8V

from redpitaya.overlay.mercury import mercury as FPGA
overlay = FPGA()

pdm = FPGA.analog_out()
voltage = 0.3                          # allowed: 0 to 1.8V
```

```
pdm.write(3, voltage)
```

The following program generates a sawtooth signal on analog output 3:

```
from redpitaya.overlay.mercury import mercury as FPGA
overlay = FPGA()
import numpy
pdm = FPGA.analog_out()
voltages=numpy.linspace(0,1.8, 100)

while True:
    for v in voltages:
        pdm.write(3, v)
```



There are 100 values that are generated per period.
A period is ca. 4ms wide (with some jitter!), so the latency time of the analog output is about 40µs.

If a voltage is set on the analog output, it stays there even when the script is stopped.

Slow analog input

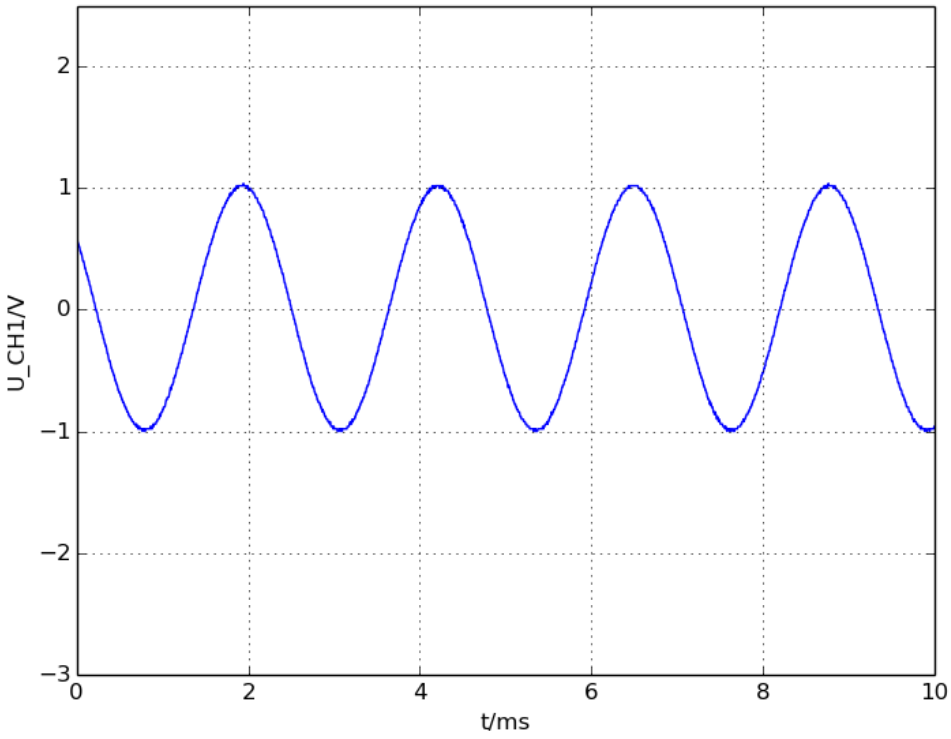
This reads the voltage on analog input 0:

```
from redpitaya.overlay.mercury import mercury as FPGA
overlay = FPGA()

myInput = FPGA.analog_in(0)
voltage = myInput.read()
print(voltage)
```

Sine wave generator

This generates a 440Hz sine wave:

```
from redpitaya.overlay.mercury import mercury as overlay
fpga = overlay()

gen0 = fpga.gen(0)
gen0.mode = 'PERIODIC'
gen0.waveform  = gen0.sin()
gen0.amplitude = 1.0
gen0.offset    = 0.0
gen0.frequency = 440

print(str(gen0.frequency) + "Hz")

gen0.start()
gen0.enable = True
gen0.trigger()
```



To stop the generator, use:

```
gen0.enable = False
```

Oscilloscope

For this experiment, I used a sine wave generator as described before, and looped the signal of OUT1 to IN1.

Voltage range of fast analog inputs on the Red Pitaya depends on gain setting that can be set by jumpers. HV setting is for input range to ±20V, while LV sets input range to ±1V.

Time length of the acquired signal depends on the time scale of a buffer that can be set with a decimation factor. Decimations and time scales of a buffer are given in the table from http://blog.redpitaya.com/examples-new/single-buffer-acquire/

Decimation factor	Effective Sampling Rate	Time scale/length of a buffer	Trigger delay in samples	Trigger delay in seconds
d	$S = 125\frac{MS/s}{d}$	$t = 16384 \cdot d \cdot 8\,ns$		
1	125 MS/s	131.072 us	- 8192 to *x*	*-6.554E-5 to x*
8	15.6 MS/s	1.049 ms	- 8192 to *x*	-5.243E-4 to *x*
64	1.9 MS/s	8.389 ms	- 8192 to *x*	-4.194E-3 to *x*
1024	122.0 MS/s	134.218 ms	- 8192 to *x*	-6.711E-2 to *x*
8192	15.2 kS/s	1.074 s	- 8192 to *x*	-5.369E-1 to *x*
65536	7.6 kS/s	8.590 s	- 8192 to *x*	-4.295E+0 to *x*

The decimation factor must be a power of 2.

The sampling rate of the Red Pitaya's ADC is always 125MS/s per channel.

So it takes $dt = \frac{1}{125 \cdot 10^6} s = 8\,ns$ to take 1 sample.

The incoming samples pass through an averager, which takes a number of samples (corresponding to the decimation factor) and writes the average of the whole lot into the output buffer.

So the effective sampling rate at which samples are written into the output buffer is 125MSps divided by the decimation factor.

The buffer length is 16K = 16384

To take a decimated sample it takes $d \cdot 8\,ns$

So to fill the buffer with 16384 samples the time is $t = 16384 \cdot d \cdot 8\,ns$

Beispielprogramm:

```
from redpitaya.overlay.mercury import mercury as overlay
fpga = overlay()

osc0 = fpga.osc(0, 1.0)       # channel index = 0 -> IN1    or 1 -> IN2
                              # voltage range = 1.0V (LV)   or 20.0V (HV)

# data rate decimation
osc0.decimation = 128

# trigger timing [sample periods]
N = osc0.buffer_size
osc0.trigger_pre  = 0
osc0.trigger_post = N
print (str(N) + " samples buffer size")

osc0.trig_src = 0              # disable hardware trigger sources

# synchronization and trigger sources are the default,
# which is the module itself
osc0.reset()
osc0.start()
osc0.trigger()

# wait for data
while (osc0.status_run()): pass
print ('triggered')

import matplotlib.pyplot as plt

# show only the part of the buffer requested by pre/post trigger timing
# plot data against sample number:
data = osc0.data(N)
plt.plot(data)
plt.show()

# plot data against time:
import numpy as np
t=np.linspace(0,16384*8E-9*osc0.decimation, 16384)
plt.plot(t,data)
plt.xlabel("t/s")
plt.ylabel("u/V")
plt.show()
```