RADCELF User Guide

Rev	Date	Description
1	20160826	init
6	20200819	new-style clock diagram, spell out CLK/TRG cmds
7	20200920	support GPS ntp, web counters page
10	20201028	FAT.
11	20210910	chirp --stop

Document created using OpenOffice.Org www.openoffice.org.

This document and D-TACQ Software comprising platform Linux port, Linux kernel modules and most applications are released under GNU GPL/FDL:

Document:

Software:

Table of Contents

1 Introduction

1.1 Scope

Covers software control of RADCELF.

RADCELF has the following programmable devices:

1) AD9854 DDS : ddsA, ddsB, ddsC
2) AD9512 CLKD chips clkdA, clkdB
3) AD7417 temperature and slow ADC
4) 8 bit slow DIO buffer.

1.2 References

1) RADCELF Block Diagram
2) AD9854 Data Sheet
3) AD9854 Quick Ref
4) AD512 Data Sheet
5) 4GUG

1.3 Software concept

Each device is represented by a device driver that presents every register as a virtual file or “knob”. The knobs have meaningful names and are set with simple ascii strings. Where possible, knobs are read/write. These features make for easy local scripting. Scripting languages available on the ACQ400 series include : shell, TCL, python.

In addition, it's ACQ400 convention to present every user-controllable knob on a “site service” where it's accessible by connecting a TCP socket to a well-known port. This makes writing a remote automation script extremely easy. The site service may also be accessed locally using the **`set.site`** command.

Then, to examine and optionally set the complete state of the device, virtual files are provided to dump / load all the registers in the device. This information is also presented in a dynamic web page for easy monitoring.

2 AD9854 DDS

2.1 Virtual Devices

2.1.1 Top level directory for RADCELF

```
ls /dev/radcelf
clkdA  clkdB  ddsA   ddsB   ddsC
```

2.1.2 One virtual file per register

```
cd /dev/radcelf/ddsA
ls [A-Z][A-Z]*
CR     DFR    FTW1   FTW2   IPDMR  POTW1  POTW2  QDACR  QPDMR  RRCR   SKRR   UCR
```

2.1.3 Knobs accessed from site service

Chip	Site	Port	Local Command	Remote Command
ddsA	4	4224	set.site 4	nc ACQ 4224
ddsB	5	4225	set.site 5	nc ACQ 4225
ddsC	6	4226	set.site 6	nc ACQ 4226

2.1.4 Example Site Service Usage

Shows a get followed by a set. It's a good idea to check that the set worked, at least in debug by following it up with another get. Or refer to the web page.

```
acq1001_068> set.site 4
FTW1
100000000000
FTW1=110000000000
FTW1
110000000000

Baadd=B0409
FTW1
090000000000
```

So either way you can set FTW1, however, apart from being less legible, “Baadd” is <u>really</u> inefficient, because not only are 6 commands required to complete the word, each command has to write the full 7 byte command..

2.1.5 Socket with complete low level emulation

Ports 4244, 4245, 4346 emulate the Baadd protocol directly, this is provided so that customer's existing client software may run directly.

Configuration:

```
acq1001_068> cat /etc/inetd.radcelf.conf
4244 stream tcp nowait root set.Baadd set.Baadd 4
4245 stream tcp nowait root set.Baadd set.Baadd 5
4346 stream tcp nowait root set.Baadd set.Baadd 6
```

Example:

```
a-host> nc acq1001_068 4244
B0411
B0500
B0600
B0700
B0800
B0900

Equivalent to

nc acq1001_068 4224
FTW1=110000000000
```

2.1.6 Recommended Scripting Style

Convenience commands are provided to easy local scripting:

```
set.ddsA    FTW1  0123456789abc

set.clkdB   UPDATE      01
```

2.1.7 Help

```
acq1001_068> set.site 4 help2
BPSK                    : rw
      0 FPGA Enable BPSK
CR                      : rw
      [7,4] Control Register
DFR                     : rw
      [4,6] Delta Frequency Register
FTW1                    : rw
      [2,6] Frequency Tuning Word #1
FTW2                    : rw
      [3,6] Frequency Tuning Word #2
IPDMR                   : rw
      [8,2] I Path Digital Multiplier Register
OSK                     : rw
      0 FPGA Enable OSK
POTW1                   : rw
      [0,2] Phase Offset Tuning Word Register #1
POTW2                   : rw
      [1,2] Phase Offset Tuning Word Register #2
QDACR                   : rw
      [B,2] Q DAC Register 2 Bytes
QPDMR                   : rw
      [9,2] Q Path Digital Multiplier Register
RRCR                    : rw
      [6,3] Ramp Rate Clock Register
SKRR                    : rw
      [A,1] Shaped On/Off Keying Ramp Rate Register
UCR                     : rw
      [5,4] Update Clock Rate Register
strobe                  : rw
      0 manually strobes update if required
strobe_mode             : rw
      0   0:manual 1:self 2:group (hit strobe to assert group)
help                    :
      help
help2                   :
      /usr/share/doc/acq400_help0:help2
```

2.1.8 Register Dumps

```
acq1001_068> cat /proc/driver/ad9854/ddsA/rare
0000
0000
110000000000
000000000000
000000000000
00000000
000000
000f0041
0000
0000
00
0000
```

2.1.9 Load/Restore state

```
cat /proc/driver/ad9854/ddsA/rare >/mnt/local/ddsA_saved_state

… later
```

```
cat /mnt/local/ddsA_saved_state > /proc/driver/ad9854/ddsA/rare
```

2.1.10 Strobe

AD9854 requires the IOUD signal to clock register settings through to the device.

Each DDS driver provides the **`strobe_mode`** knob to control this operation.

Values for **`strobe_mode`** are:

0 : `SPI_STROBE_NONE`

no action, a separate write to the **`strobe`** knob pulses IOUD

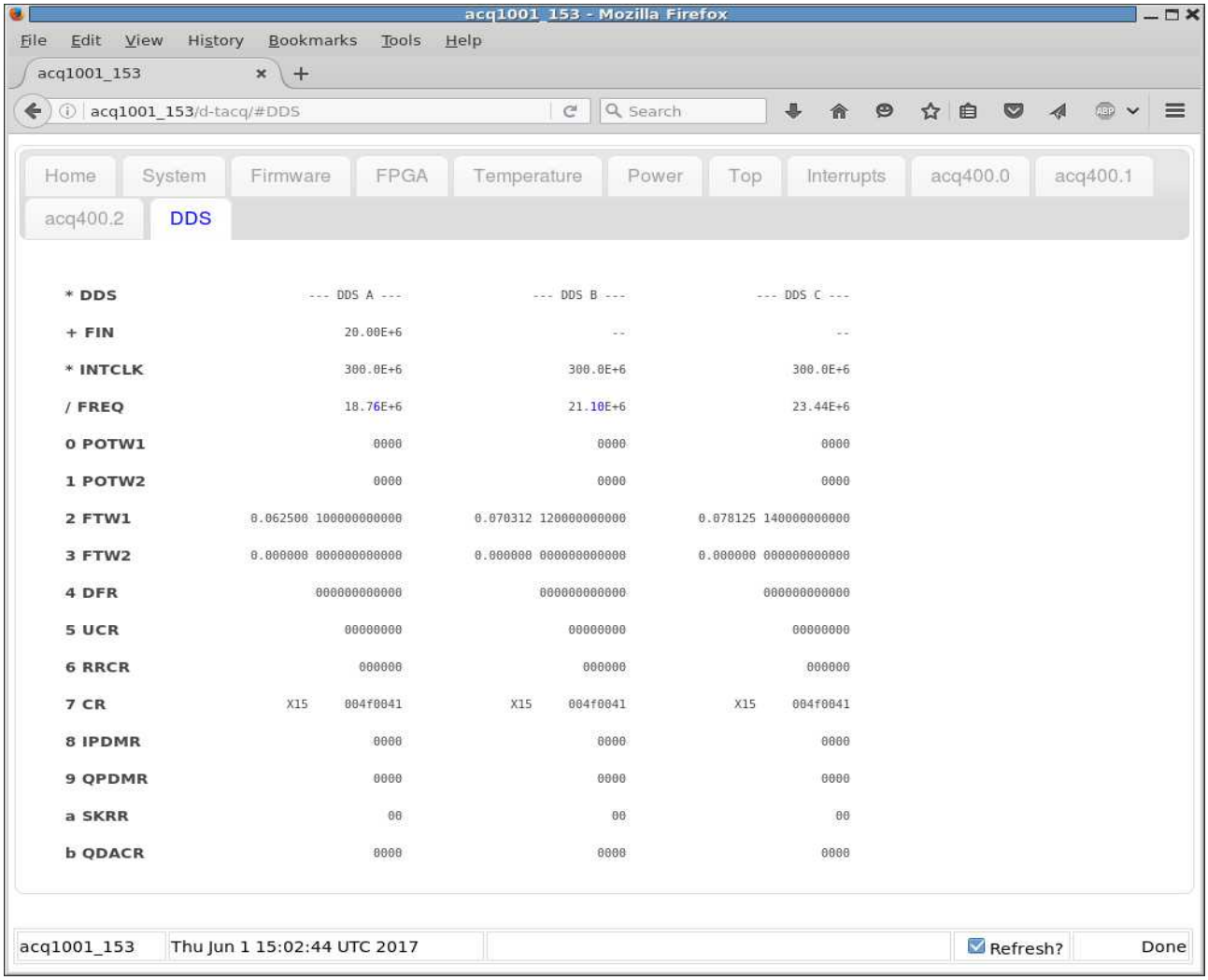
1: `SPI_STROBE_SELF`

automatic IOUDD pulse after each command [default]

2: `SPI_STROBE_GROUP`
ddsA, ddsB constitute a group, in this mode, a write to the **`strobe`** knob pulses IOUD on both dds devices to enable synchronous update

2.2 Web page.

Shows complete register state of all 3 DDS, + measured output frequency



3 AD9512 CLKD

3.1 Virtual Devices

3.1.1 Top level directory for RADCELF

```
ls /dev/radcelf
clkdA  clkdB  ddsA   ddsB   ddsC
```

3.1.2 One virtual file per register

```
acq1001_068> cd /dev/radcelf/clkdA/
acq1001_068> ls [A-Z][A-Z]*
CSPD     DFA4     DIV0     DIV2     DIV4     LVDS3    LVPECL0  LVPECL2  UPDATE
DBP4     DFS4     DIV1     DIV3     FUNPS    LVDS4    LVPECL1  SCPC
```

3.1.3 Knobs accessed from site service

Chip	Site	Port	Local Command	Remote Command
clkdA	7	4227	set.site 7	nc ACQ 4227
clkdB	8	4228	set.site 8	nc ACQ 4228

3.1.4 Example Site Service Usage

Shows a get followed by a set. It's a good idea to check that the set worked, at least in debug by following it up with another get. Or refer to the web page.

```
acq1001_068> set.site 4
FTW1
100000000000
FTW1=110000000000
FTW1
110000000000
```

3.1.5 Help

```
acq1001_068> set.site 7
help2
CSPD                : rw
      [45 1] Clocks Select Power Down
DBP4                : rw
      [34 1] Delay Bypass 4
DFA4                : rw
      [36 1] Delay Fine Adjust 4
DFS4                : rw
      [35 1] Delay Full Scale 4
DIV0                : rw
      [4A 2] Divider 0  8000 => Bypass
DIV1                : rw
```

```
      [4C 2] Divider 1  8000 => Bypass
DIV2                  : rw
      [4E 2] Divider 2  8000 => Bypass
DIV3                  : rw
      [50 2] Divider 3  8000 => Bypass
DIV4                  : rw
      [52 2] Divider 4  8000 => Bypass
FUNPS                 : rw
      [58 1] FUNCTION Pin and Sync
LVDS3                 : rw
      [40 1] LVDS CMOS Out 3
LVDS4                 : rw
      [41 1] LVDS CMOS Out 4
LVPECL0               : rw
      [3D 1] LVPECL Out 0
LVPECL1               : rw
      [3E 1] LVPECL Out 1
LVPECL2               : rw
      [3F 1] LVPECL Out 2
SCPC                  : rw
      [0 1] Serial Port Control Configuration
UPDATE                : rw
      [5A 1] Update Registers
```

4 Voltage Monitor and DIO

Knobs available on site 3.

d? : dio bit

in? : analog inputs

temp?: temperature.

```
acq1001_153> get.site 3
help
ai1
ai2
ai3
ai4
ai5
ai6
ai7
ai8
d0
d1
d2
d3
d4
d5
d6
d7
temp
temp2
```

4.1 DIO direction

Default: out, set by modifying file on boot as follows (script in /mnt/local/rc.user).

```
acq1001_153> cat /dev/gpio/CELF/.direction.d0
out
acq1001_153> echo in > /dev/gpio/CELF/.direction.d0
```

5 Boot Time Initialisation

`/usr/local/init/RAD-CELF-init`

Enables the DDS devices, configures a 25MHz source clock and sets the DDS to run internally at 300MHz.

6 Test routines

6.1 Sets some default clocks

/usr/local/CARE/radcelf-init-123

Configures ddsA , ddsB, ddsC with test frequencies. Use Web page to confirm it's working.

6.2 Set a single chirp

shell script runs on ACQ1001, sets single chirp

```
#!/bin/sh
#
# SETTING KAKA'AKOS CHIRP
#
# Set AD9854 clock remap to 25 MHz
export PATH=$PATH:/usr/local/bin
set.ddsC        CR      004C0041
set.ddsC        FTW1    1AAAAAAAAAAA
# Program AD9512 secondary clock to choose 25 MHz from the AD9854 remap
set.clkdB       CSPD    02
set.clkdB       UPDATE  01
# Program the chirp using Kaka'ako parameters
set.site 2 ddsA_upd_clk_fpga 1
set.ddsA        CR      004F0061
set.ddsA        FTW1    172B020C49BA
set.ddsA        DFR     0000000021D1
set.ddsA        UCR     01F01FD0
set.ddsA        RRCR    000001
set.ddsA        IPDMR   0FFF
set.ddsA        QPDMR   0FFF
set.ddsA        CR      004C8761
# Set the trigger
set.site 2 ddsA_upd_clk_fpga 0
# lera_acq_setup
# we assume a 25MHz from ddsC
# trigger from site 3 ddsA
set.site 1 trg 1,3,1
set.site 1 clk 1,3,1
set.site 1 hi_res_mode 1
# 25 MHz/4 = 6.25MHz / 512 = SR 12207
set.site 1 CLKDIV 4
```

7 Remote Scripting

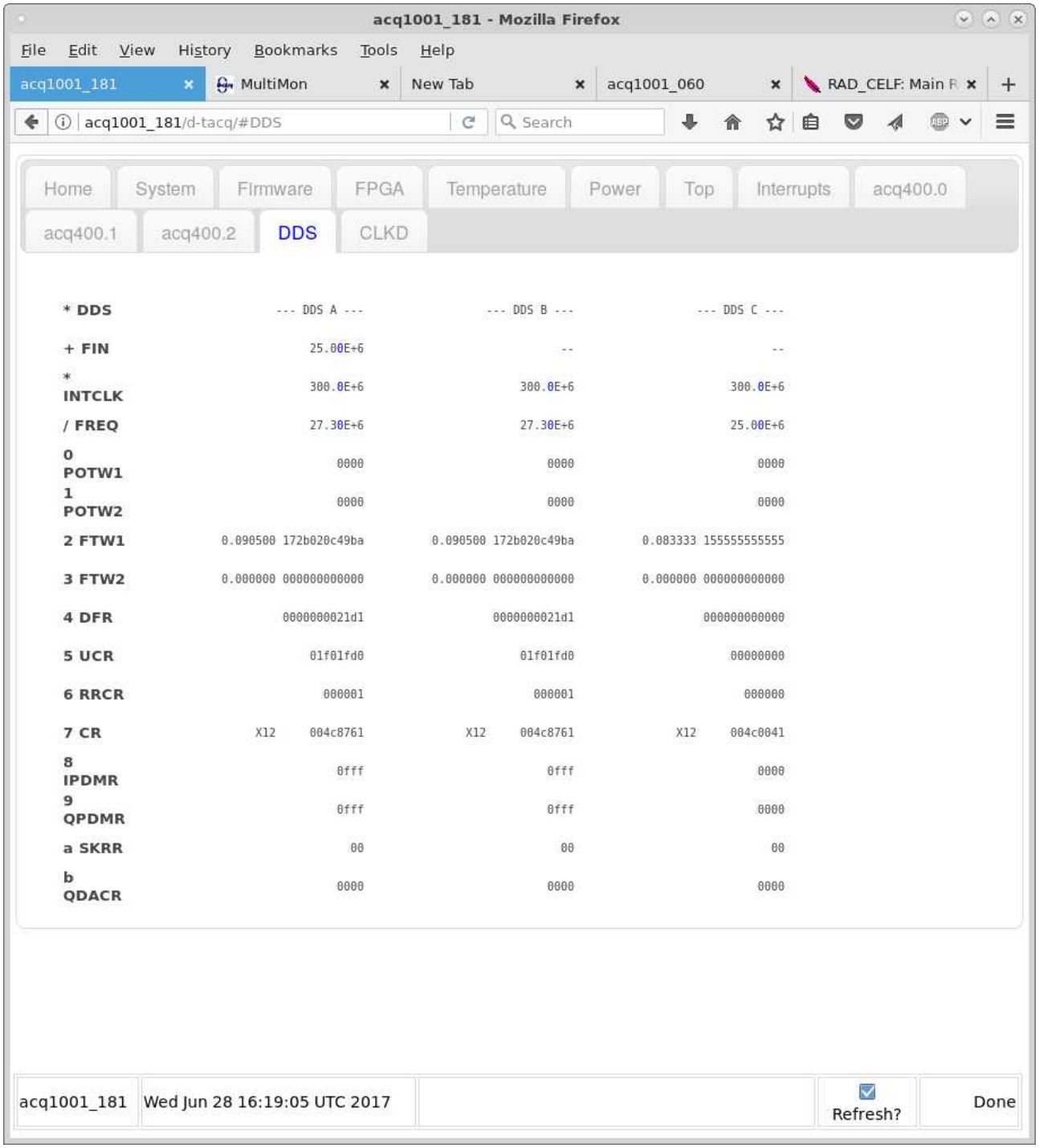
D-TACQ offers a host side api “HAPI”. This brings all the site services together in one consistent object API. https://github.com/petermilne/acq400_hapi/test_apps

Example below “radcelf-chirp-init.py” sets up a chirp on either or both of ddsA or ddsB

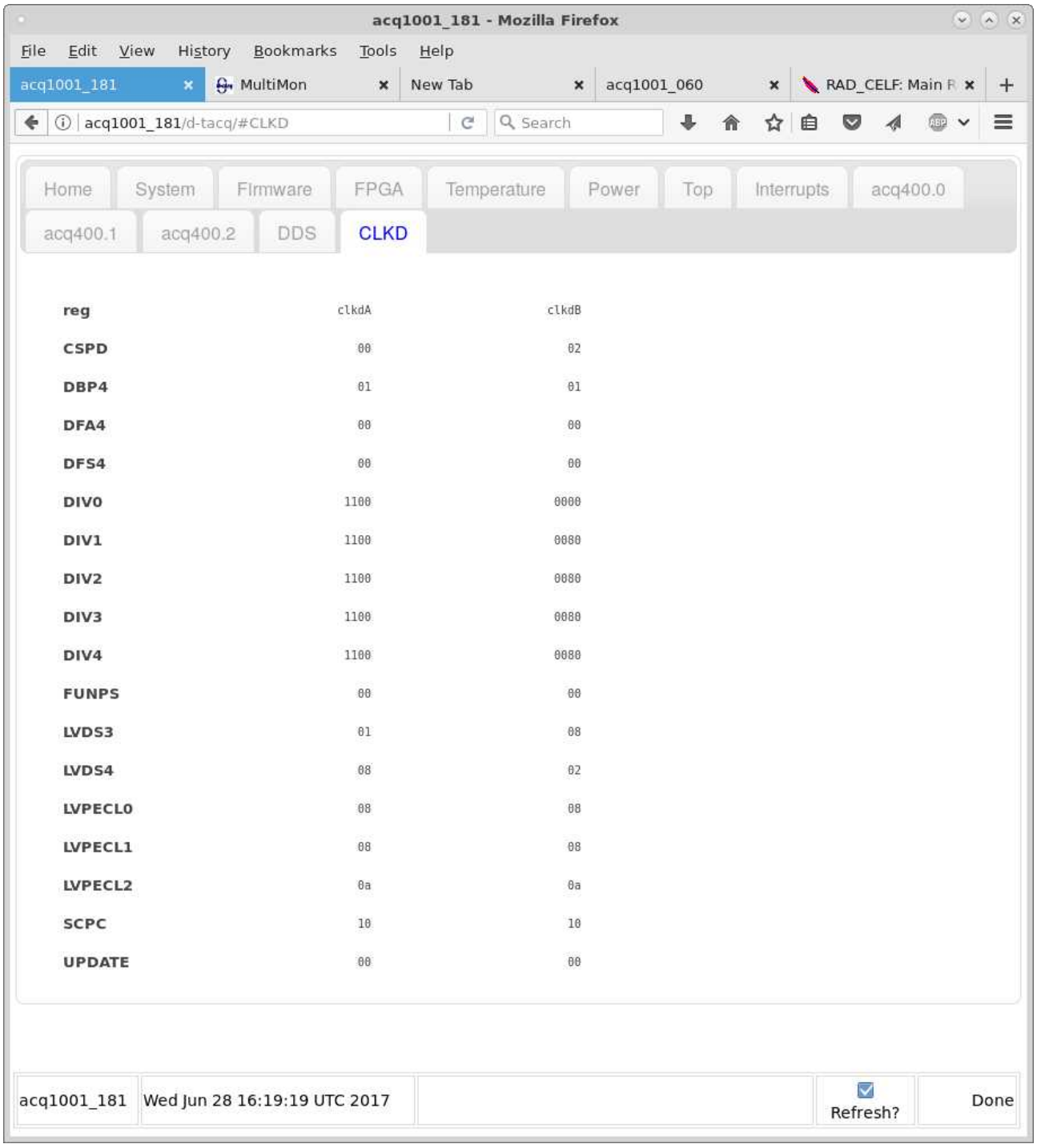
Note, it's basically identical to the shell script example, but with a richer capability. HAPI scripts may also be run on the embedded system, which includes a full Python 3 from 2020.

```
def init_chirp(uut, idds):
# SETTING KAKA'AKOS CHIRP
#
# Set AD9854 clock remap to 25 MHz
    dds = uut.ddsA if idds == 0 else uut.ddsB

    uut.ddsC.CR     = '004C0041'
    uut.ddsC.FTW1   = FTW1(1.0/12.0)
# Program AD9512 secondary clock to choose 25 MHz from the AD9854 remap
    uut.clkdB.CSPD  = '02'
    uut.clkdB.UPDATE = '01'
# Program the chirp using Kaka'ako parameters
    set_upd_clk_fpga(uut, idds, '1')
    dds.CR     = '004F0061'
    dds.FTW1   = '172B020C49BA'
    dds.DFR    = '0000000021D1'
    dds.UCR    = '01F01FD0'
    dds.RRCR   = '000001'
    dds.IPDMR  = '0FFF'
    dds.QPDMR  = '0FFF'
    dds.CR     = '004C8761'
    set_upd_clk_fpga(uut, idds, '0')

# Set the trigger
# lera_acq_setup
# we assume a 25MHz from ddsC
# trigger from site 3 ddsA
    uut.s1.trg  = '1,3,1'
    uut.s1.clk  = '1,3,1'
    uut.s1.hi_res_mode = '1'
# 25 MHz/4 = 6.25MHz / 512 = SR 12207
    uut.s1.CLKDIV = '4'
# create an object instance and init_chirp
init_chirp(acq400_hapi.RAD3DDS("acq1001_181"), 0)
```

7.1 Monitoring After Chirp : Web pages:



7.2 Monitoring CLKD

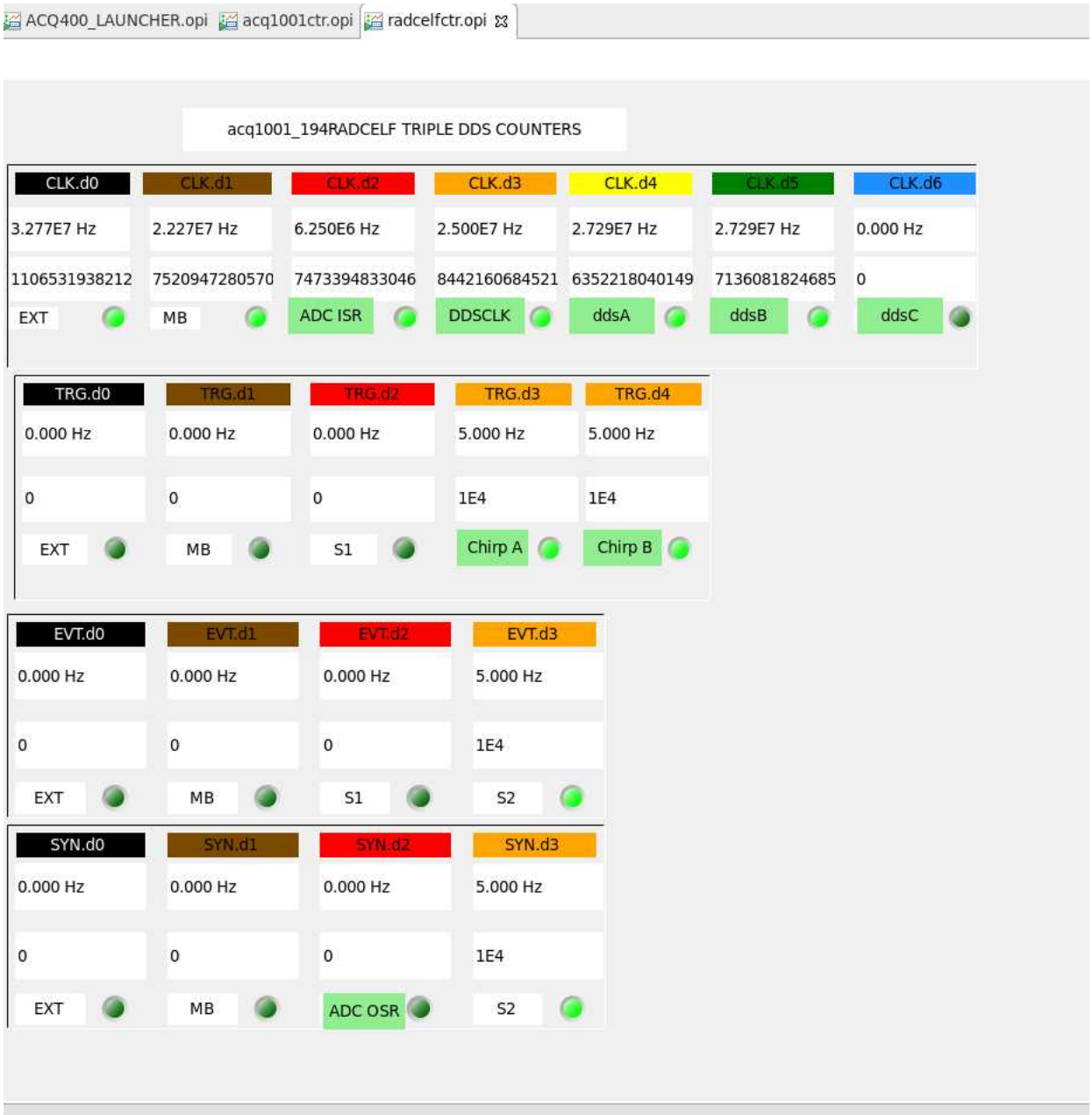


Web page shows current CLKD register settings.

7.3 Monitoring Chirp

7.3.1 OPI

Users with cs-studio get a colour mimic

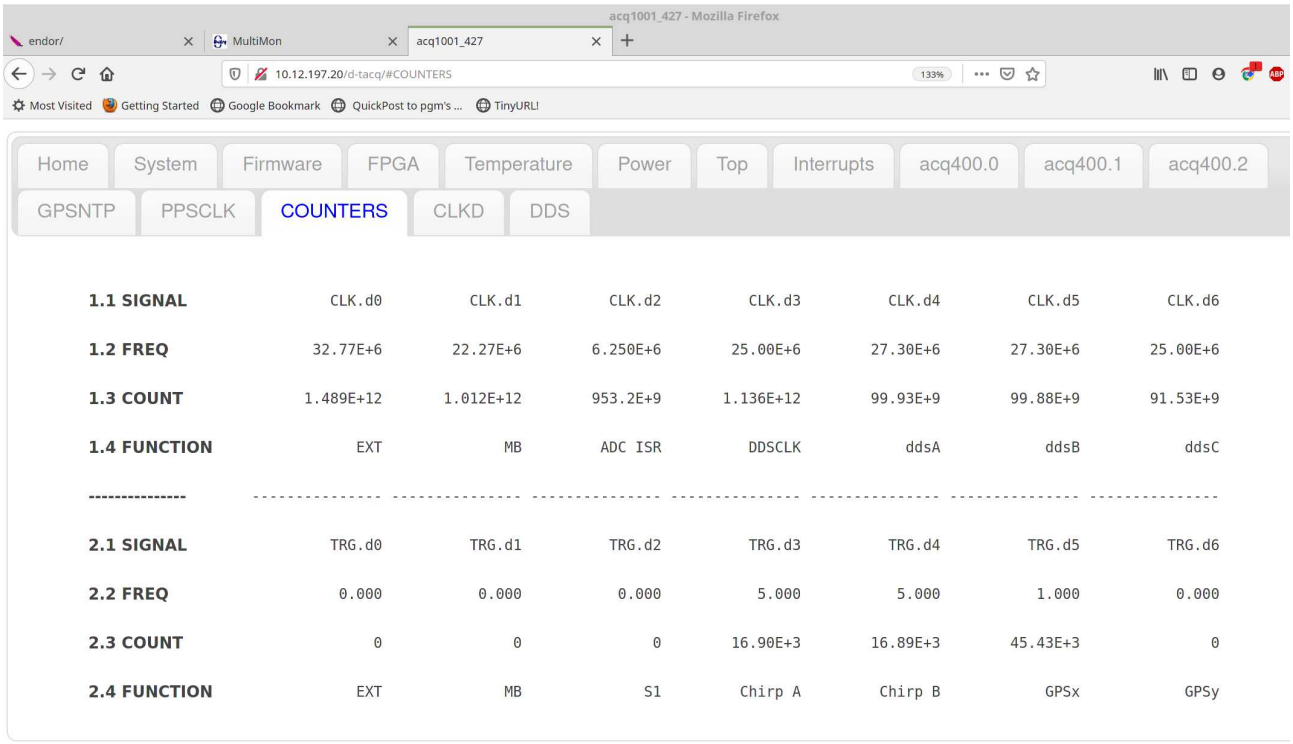


** These counters are good to about 50MHz. Higher frequencies, or rapidly fluctuating frequencies will NOT record correctly.

7.3.2 Web

NEW from 2020 - 09 -20 : web display shows the counters without need for cs-studio:

The web page style is very basic, but it does update dynamically.



1.1 SIGNAL	CLK.d0	CLK.d1	CLK.d2	CLK.d3	CLK.d4	CLK.d5	CLK.d6
1.2 FREQ	32.77E+6	22.27E+6	6.250E+6	25.00E+6	27.30E+6	27.30E+6	25.00E+6
1.3 COUNT	1.489E+12	1.012E+12	953.2E+9	1.136E+12	99.93E+9	99.88E+9	91.53E+9
1.4 FUNCTION	EXT	MB	ADC ISR	DDSCLK	ddsA	ddsB	ddsC
----------	----------	----------	----------	----------	----------	----------	----------
2.1 SIGNAL	TRG.d0	TRG.d1	TRG.d2	TRG.d3	TRG.d4	TRG.d5	TRG.d6
2.2 FREQ	0.000	0.000	0.000	5.000	5.000	1.000	0.000
2.3 COUNT	0	0	0	16.90E+3	16.89E+3	45.43E+3	0
2.4 FUNCTION	EXT	MB	S1	Chirp A	Chirp B	GPSx	GPSy

7.4 Remote monitoring

The frequency counters are aliased on site 2 with meaningful names:

```
[pgm@hoy4 acq400_hapi_tests]$ nc acq1001_181 4222
*SIG*
SIG:DDS:A:CHIRP:COUNT 595852
SIG:DDS:A:CHIRP:FREQ 0
SIG:DDS:A:FREQ 0
SIG:DDS:B:CHIRP:COUNT 77397
SIG:DDS:B:CHIRP:FREQ 0
SIG:DDS:B:FREQ 0
SIG:DDS:C:FREQ 0
SIG:DDS:INP:FREQ 25004591
```

The counters may be cleared from this knob:

```
clear_counts 1
```

Clearing the counts is useful to track “number of chirp cycles from start”.

The DDS and CLKD system may be restored to power up state:

```
RADCELF_init 1
```

7.5 Hooking RADCELF outputs on ZYNQ FPGA

7.5.1 Clock Signals

Signals appearing on the clock bus. The most common usage is to link a clock source to the ADC in site 1. eg

- set.site 1 clk=1,1,1 # source from the MBCLOCK (ACQ435 default mode)
- set.site 1 clk=1,3,1 # source from DDSCLK
- set.site 1 clk=1,6,1 # source from ddsC

Name	Clock Bus	Comment
EXT	d0	Front panel clock
MB	d1	Motherboard Clock MBCLK
ADC	d2	ADC modulator clock
DDSCLK	d3	DDS source clock (clkdB output)
ddsA	d4	
ddsB	d5	
ddsC	d6	

For full understanding of motherboard clock routing, please refer to CLKTREE OPI

7.5.2 Trigger Signals

Signals appearing on the TRG bus. Likely use cases:

- set.site 1 trg=1,0,1 # set external start trigger
- set.site 1 event0=1,3,1 # embedded timing event on ChirpA
- set.site 1 RGM:DX=d3 # Burst Mode, trigger on ChirpA

Name	TRG Bus	Comment
EXT	d0	Front panel trigger
SOFT	d1	Soft trigger
GPG2	d2	GPG output
ChirpA	d3	DDSA Chirp trigger
ChirpB	d4	DDSB Chirp trigger
PPS	d5	GPS PPS Output (1Hz)
PPS2	d6	GPS PPS2 Output (100Hz)
GPG7	d7	GPG output
GPG: Gate Pulse generator, programmable intervals from trigger		

8 GNSS Integration

Feature set available from 2020-09-21

8.1 GNSS Unit

uBlox RCB-F9T-0-01 precision time module may be integrated with RADCELF

The GNSS unit provides an NMEA feed to the ACQ1001 on *dev*ttySP1 and a ONE-PPS signal that is mapped by the FPGA to TRG.d5 and GPIO453

8.2 Theory Of Operation

This document describes the FPGA changes and software methods used to demonstrate GPS synchronisation in the RAD-CELF of the ACQ435 Digitiser to within a small fraction of the sample clock.

The GPS RAD-CELF synchronisation involves frequency synchronisation across multiple RAD-CELF modules by measuring the frequency of the local oscillator after passing through the Clock Remapping DDS (DDS-C). This frequency is then adjusted to precisely 25 MHz.

To do this the FPGA counts the number clock cycles between the PPS pulses from the GPS module. This is then averaged over 10,30 and 100 seconds to obtain up to 0.01 Hz resolution on the frequency. The FTW1 on the remapping DDS is adjusted up or down until the target 25 MHz frequency is reached. The precision of the FTW on the DDS device allows this precision to be reached. The limiting factors are GPS PPS precision, and both GPS and Oscillator drift in both time and temperature.

The FPGA logic provides the Oscillator count from DDS-C and software provides this count and the average of 10, 30 and 100 seconds.

The script **`radcelf-tune-pps.py`** is provided to adjust the DDS FTW to achieve 25 MHz to within 0.02 Hz accuracy

Running **`radcelf-tune-pps.py`** `--fine=1,` program continuously after the main script to attempt to maintain the frequency over a long period, during the shot, if necessary.

Once the Frequency has been adjusted the system can then be started using the new FPGA logic that synchronises the Chirp start to the GPS PPS logic.

This achieves synchronous chirp generation but the ACQ435 is a sigma-delta ADC which divides down the system clock by 512 to achieve the sample clock,

this means that another change to the FPGA is needed to adjust the “phase” of this sample clock divider to achieve ADC synchronisation across multiple devices.

The problem is that the current setup results in a sample frequency of

25MHz/(4*512) of 12207.03 kHz. This means that the ADC sample point is constantly rotating with respect to the GPS PPS signal.

The method chosen was to use a frequency that is a multiple of 512Hz rather than the nominal 25 MHz. This requires the frequency of DDS-C to move to 24.999936 MHz. This changes the default sample rate of 12207.03 Hz to exactly 12207 Hz.

With this integer number of samples per second the GPS PPS signal can then be used to reset the

phase of the 512 clock divider to the nearest oscillator clock. This achieves a 1/512 skew between two ACQ435 digitisers synchronised this way.

The FPGA was changed so that prior to starting a capture the ADC divider synchronises to every PPS pulse. Once a capture is started the synchronisation is disabled and subject to any drift between the clocks.

Due to the ADC pipeline there is a 40 sample settling time for the FIR filter in the ADC before data capture can commence, this is a minor issue as it is normally several seconds before the ADC is triggered.

This method is a proof of concept to show synchronous sampling of the ADC if the frequency change is an issue further mechanisms would be explored.

The script **`radcelf-tune-pps.py`** takes an optional user-supplied DDSCLK frequency as first argument [default 25000000]. The script runs by default in a mode `--best=1`, this will adjust the supplied frequency to the nearest multiple of 512Hz. Users can disable this feature by running

--best=0.

8.3 Time Multiplexing

RCB-F9T actually has two PPS outputs. PPS1 is used for the main synchronization procedure below. PPS2 is set on boot to run at 100Hz. It's not currently used, but we suggest it could be the basis of a Time Domain Multiplex system, where multiple transmitters share the same frequency, but are allocated time slots. For example, if one second is divided into 100 timeslots, then Radar A could be set to chirp on PPS2#33 aka Slot 33 and Radar B could be set to chirp on PPS2#34 aka Slot 34 and in this way they don't interfere.

We have a mechanism that allows a radar to select a given 1PPS edge; this isn't in the current firmware, but this mechanism could be extended to select a given 100PPS edge.

8.4 GPSNTP

Enable by setting optional package : 99-custom_gpsntp-2009201746.tgz or newer

8.4.1 Customisation

A default ntp customisation file is provided:

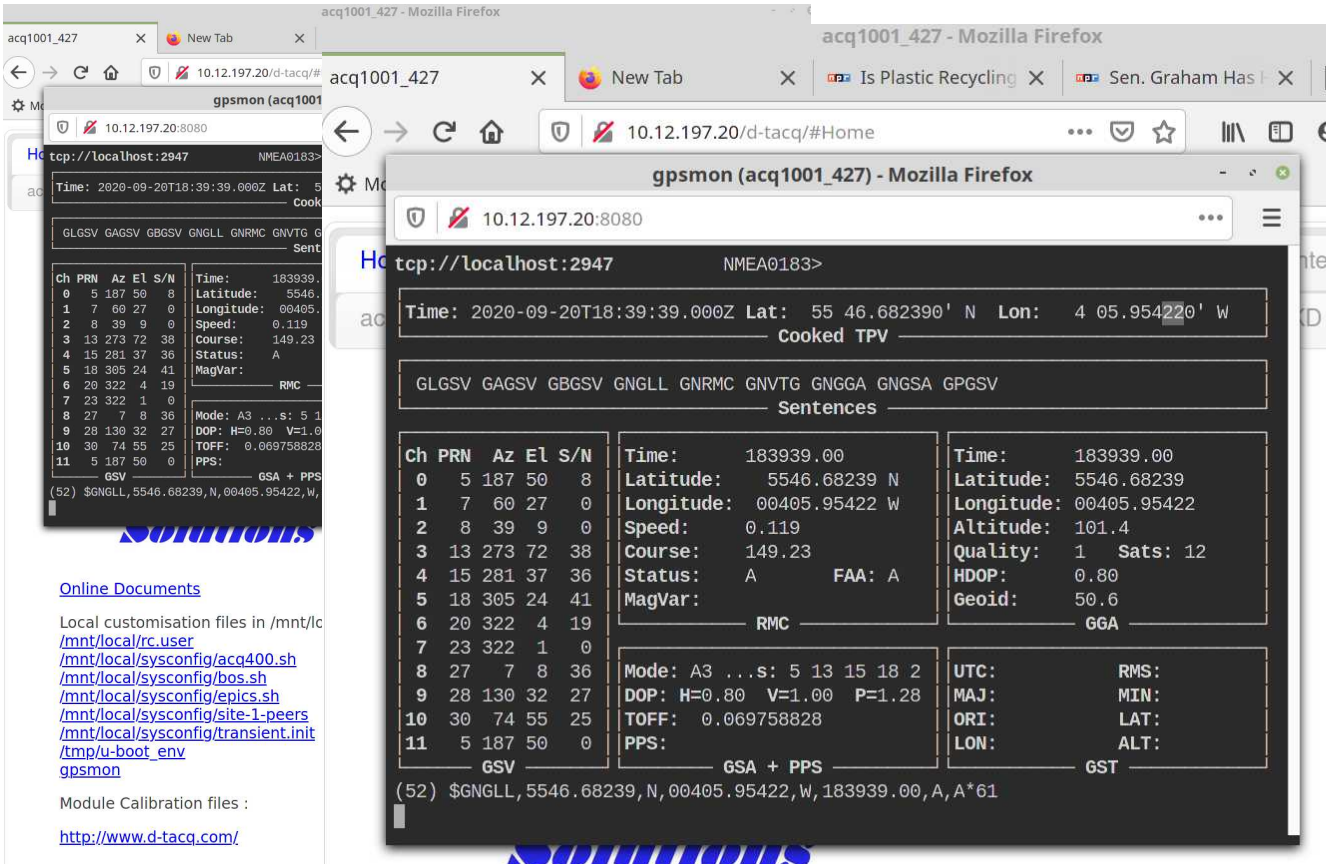
/usr/local/CARE/GPSNTP/ntp.conf, this is copied into place at /etc/ntp.conf on boot

Should this not be sufficient, the user if free to create a custom ntp.conf, to be installed on boot:

/mnt/local/ntp.conf, copied to /etc/ntp.conf on boot

8.4.2 Web Diagnostics : GPSMON

The “gpsmon” link on the home page launches an interactive gpsmon window.

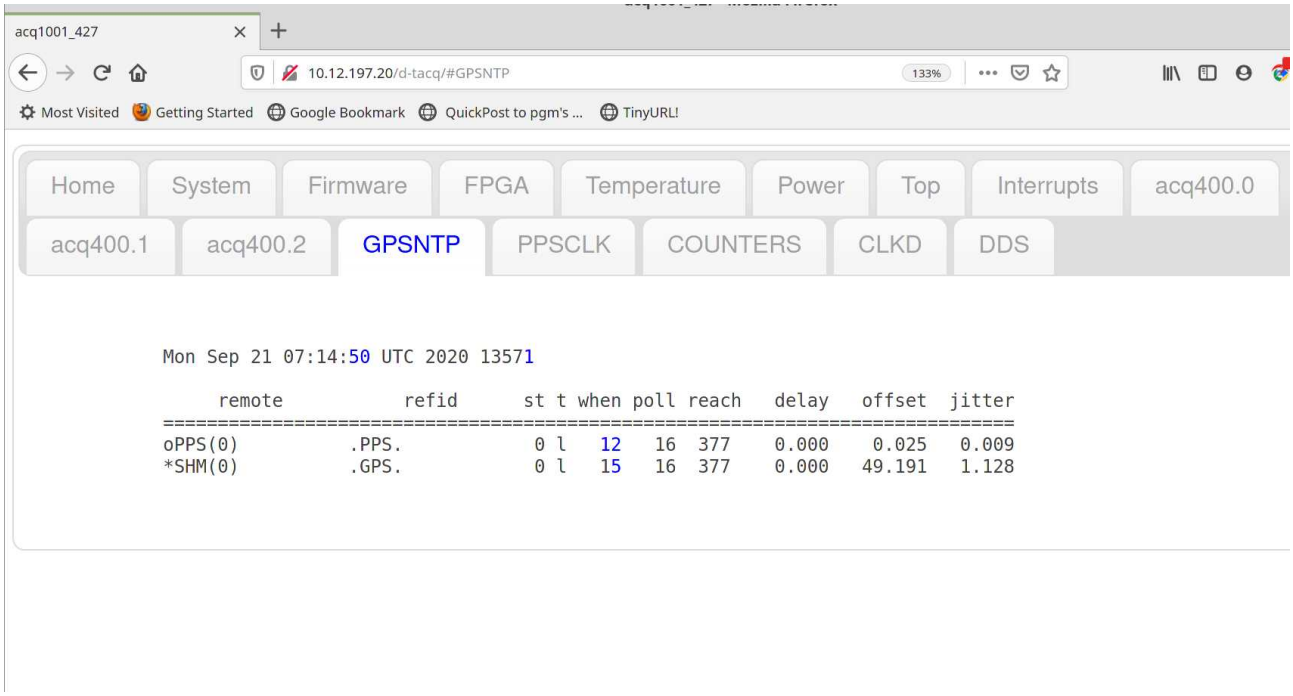


8.4.3 Web Diagnostics GPSNTP

The GPSNTP tab shows results on “ntpq -p” updated on a 3s schedule

The key figure of merit is PPS jitter, shoing 0.009 msec in the screenshot below:

The number after the date is the the ntp run time, normally it will need a few minutes to stabilize.



8.5 GPS RESET

example: reset ON, reset OFF:

```
acq1001_427> set.site 2 gps_reset 1
```

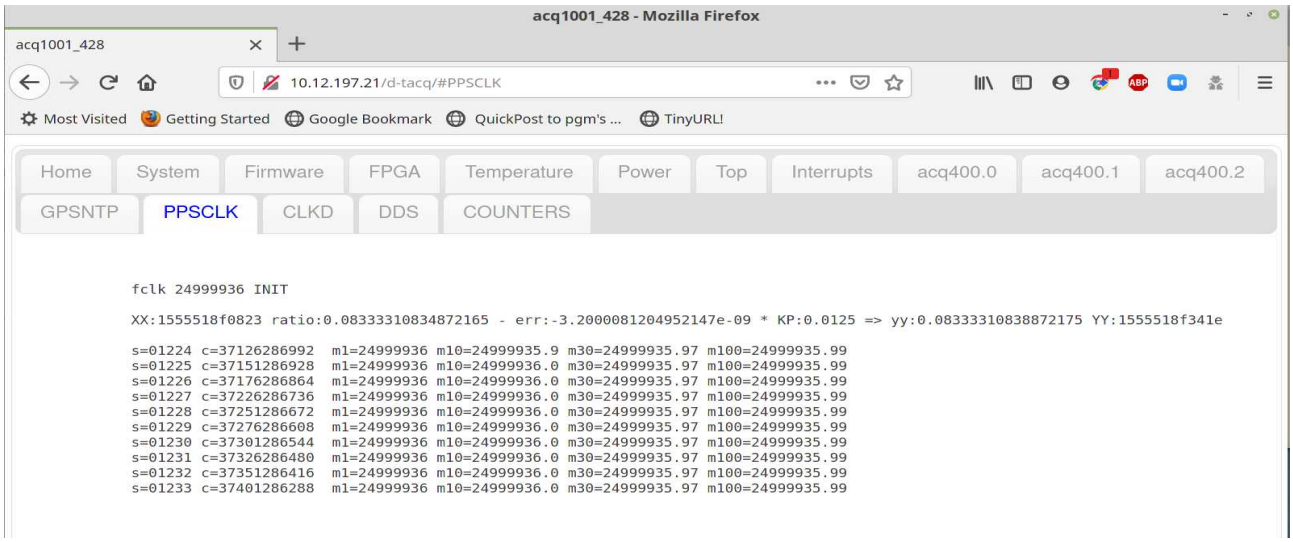
8.6 ubxtool

ubxtool(1) is provided to interrogate the u-blox, and it can apparently change defaults.

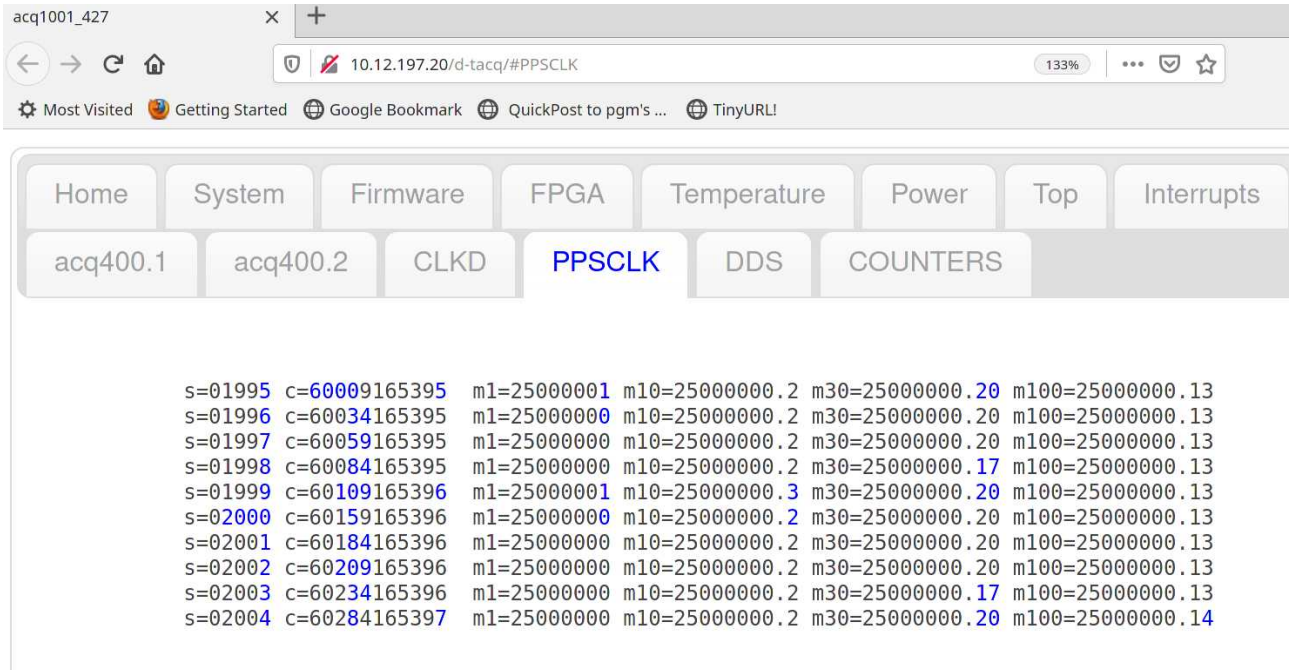
8.7 Clock conditioning to ONEPPS

FPGA hardware maintains a count of DDS-C output, latched every ONEPPS. The intention is that DDS-C can be tuned to an exact number of cycles per ONEPPS interval so that the DDS clock is exactly tuned to GNSS. The tuning may be conducted over a long interval.

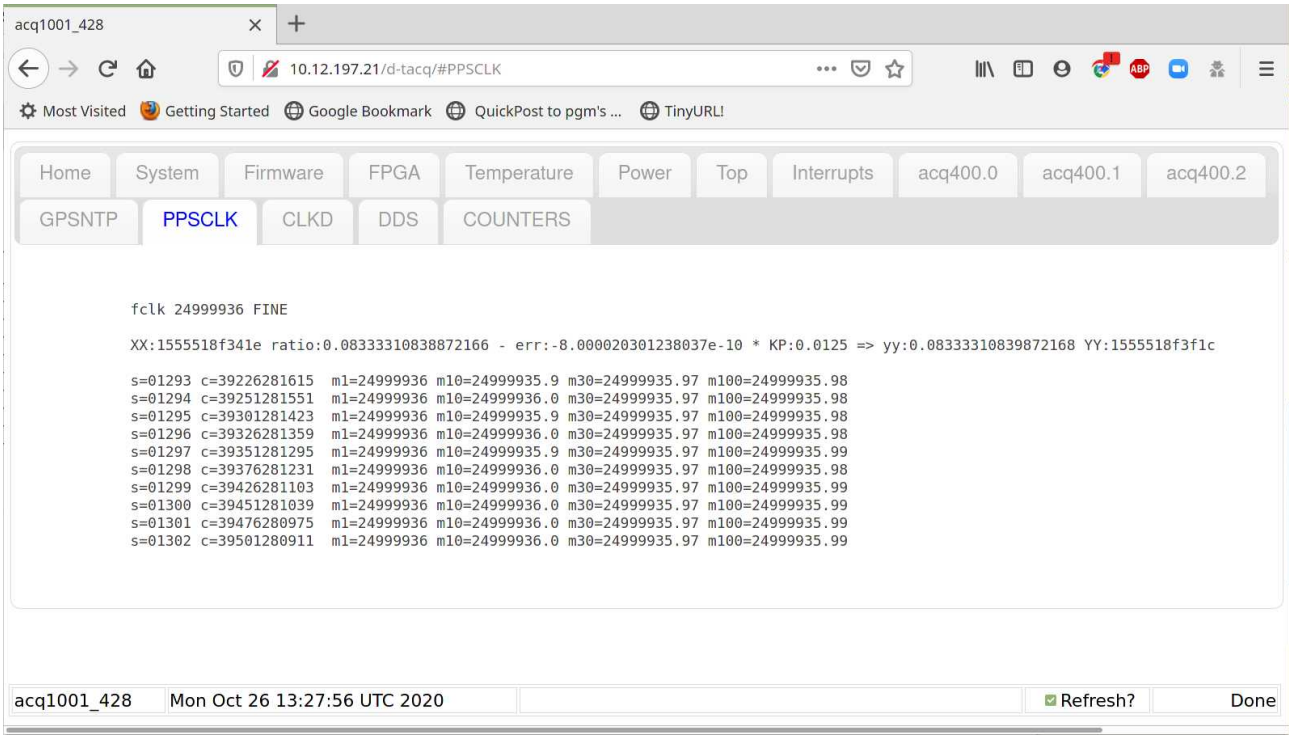
8.7.1 Before Tuning



8.7.2 After Tuning



DDS-C was tweaked from 155555555555 to 155555d8a555.



8.7.3 *Automated Frequency Tuning*

8.7.3.1 *From Boot*

Possible to run **`radcelf-tune-pps.py`** from//mnt/local/rc.user.

8.7.3.2 *Start a chirp*

```
acq1001_427> /usr/local/HAPI/test_apps/radcelf-chirp-init.py localhost
test:0  PASS SIG:TRG_S2:FREQ 0.000 SIG:TRG_S3:FREQ 0.000
```

8.7.3.3 Initial Local Clock Tuning

`radcelf-tune-pps.py` [--best[=1] DDSCLK] :Runs once and terminates

```
acq1001_427> radcelf-tune-pps.py
control m1
XX:155555555555 ratio:0.08333333333333215 - err:-3.6e-07 * KP:0.05 =>
yy:0.08333335133333215 YY:155555a2a48a
control m1
XX:155555a2a48a ratio:0.08333335133333009 - err:-4e-07 * KP:0.05 =>
yy:0.08333337133333009 YY:155555f88ac5
control m1
XX:155555f88ac5 ratio:0.08333337133332819 - err:1.2e-07 * KP:0.05 =>
yy:0.08333336533332819 YY:155555dec5b3
control m10
XX:155555dec5b3 ratio:0.08333336533332769 - err:-7.200000002980232e-08 *
KP:0.05 => yy:0.0833333689333277 YY:155555ee3bf0
control m10
XX:155555ee3bf0 ratio:0.08333336893332444 - err:5.200000002980232e-08 * KP:0.05
=> yy:0.08333336633332443 YY:155555e31135
control m10
XX:155555e31135 ratio:0.08333336633332422 - err:3.200000002980232e-08 * KP:0.05
=> yy:0.08333336473332421 YY:155555dc31fd
control m10
XX:155555dc31fd ratio:0.08333336473332409 - err:1.2000000029802323e-08 *
KP:0.05 => yy:0.08333336413332408 YY:155555d99e48
control m100
XX:155555d99e48 ratio:0.08333336413332404 - err:-5.999999940395355e-09 *
KP:0.05 => yy:0.08333336443332404 YY:155555dae822
```

NB: this is a simple proportional control servo; likely end-users will be able to come up with a better algorithm. Note the count converging to 25Mhz exactly over all time periods..

The last state of radcelf-tune-pps.py is presented on the PPSCLK web page.

- XX: Current 48bit FTW
- ratio: Current FTW as a ratio
- error: difference with desired value
- KP : proportional gain constant.
- yy: Next value as a ratio
- YY: Next 48 bit FTW.

The control algorithm doesn't claim to be the best. In particular, the very low Kp value was found by experiment. The author's suspicion is that because we're actually controlling a ratio, our control function is not a linear gain, but rather a reciprocal gain and therefore applying a linear constant doesn't work as expected. This should be a nice control problem to solve optimally.

8.7.3.4 Fine Clock Tuning

`radcelf-tune-pps.py` `--fine=1` : fine tuning that could run continuously.

8.8 Synchronized Chirp

8.8.1 Trigger_at

Starts a trigger on a given GPS second.

Usage:

`trigger_at` `+N` : trigger in +N seconds (actually, only useful for single UUT)

`trigger_at` `TIME_T` : trigger at this TIME_T (in the future)

`trigger_at today` `HH:MM[:SS] [HH:MM ...]` `Wall Clock Style`

- +N style is only helpful when testing a single UUT (think: delays on network)
- TIME_T style is ideal for script controlled testing (radcelf-chirp-init use it)
- Wall Clock style can be used for turnkey systems

8.8.2 Radcelf-Chirp-Init

radcelf-chirp-init.py [--opts] [uuts]

- default uut : localhost
- radcelf-chirp-init can set up several U.
- User can chose whether to use a central script setting many uuts, or to configure each uut individually.

```
acq1001_427> radcelf-chirp-init.py --help
usage: radcelf-chirp-init.py [-h] [--test TEST] [--debug DEBUG]...

radcelf-chirp-init

positional arguments:
  uuts                  uut

optional arguments:
  -h, --help            show this help message and exit
  --test TEST           set number of tests to run
  --debug DEBUG         1: trace 2: step
  --noverify NOVERIFY   do not verify (could be waiting gps)
  --ddsX DDSX           ddsA=1, ddsB=2, ddsA+ddsB=3
  --gps_sync GPS_SYNC   syncronize with GPSPPS >1: autotrigger at + gps_sync s
  --chirps_per_sec CHIRPS_PER_SEC
                        chirps per second
  --stop                stop chirp and quit [no value]
  --noidletone          ensure no tone output between chirp runs
  --power_down          after --stop, also power down
  --trigger_adc_dx TRIGGER_ADC_DX
                        trigger ACQ on ddsA or ddsB or dX [X=0,1,2,3,4,5,6]
  --init_trigger        --init_trigger : configure trigger only
```

- By default radcelf-chirp-init.py with no args will run exactly as per the original version, setting up ddsA, ddsB chirps at 5Hz, soft triggered on localhost.
- --ddsX=1 : init ddsA only
- --ddsX=2 :init ddsB only
- --chirps_per_sec=N may be used to set other chirp rates, to the limits of available parameters.
- --gps_sync N, N> : will autotrigger at +N seconds
- --gps_sync -1 : will configure for autotrigger, but leave the actual time of trigger for others to set up
- use SITECLIENT_TRACE to show the commands
- use --debug=1 to trace operation (prints names of functions on entry/exit)
- use --debug=2 to step operation (user breakpoint at start/end of every function.
- --stop [uuts] : STOP the chirp and capture on UUTS.

8.8.3 Test Case Example

```
SITECLIENT_TRACE=1 \
radcelf-chirp-init.py --chirps_per_sec=10,5.555 --noidletone --gps_sync=5 \
acq1001_427 acq1001_428
```

- configure 2 UUTs to run 10, 5.555 chirps per second, triggering 5s in the future
- suppress the main frequency generation between runs.

8.8.4 Stopping the chirp

Stopping the chirp will also set the DDS main frequency to zero, and optionally set PD (Power Down) bits as well:

```
radcelf-chirp-init.py [--ddsX=X] --stop --noidletone [UUTS]
radcelf-chirp-init.py [--ddsX=X] --stop --power_down [UUTS]
```

8.8.5 Script Trace and debug

Print all interactions with hardware:

```
SITECLIENT_TRACE=1 radcelf-chirp-init.py ...
```

Debug mode: print function entry and exit

```
SITECLIENT_TRACE=1 radcelf-chirp-init.py --debug=1 ...
```

Debug mode: step and print function entry and exit

```
SITECLIENT_TRACE=1 radcelf-chirp-init.py --debug=2 ...
```

8.8.6 Turnkey Example

All units are configured to boot turnkey, and to restart every hour:

Embed in a boot script eg //mnt/local/radcelf-turnkey-boot and call in the background from //mnt/local/rc.user

NB: this example is a little contrived. Users will probably do better, but it shows the concept:

Example uses at(1) to control setups and captures ahead of time, so the setup is complete and the capture is armed ahead of the precise trigger time.

```
#!/bin/sh
# radcelf-turnkey-boot, run a capture on the hours for the workday morning

sleep 300                 # allow GPSNTP to settle

radcelf-tune-pps.py       # tune the clock (once)

radcelf-tune-pps.py --fine=1 & # leave the fine tuning on in the background


# trigger every hour on the hour
trigger_at today 07:00 08:00 09:00 10:00 11:00 12:00

# setup radcelf-chip-init to reinitialise 5 minutes before hand.
echo radcelf-chirp-init.py --chirps_per_sec=10 | at 06:55
```

```
echo radcelf-chirp-init.py --chirps_per_sec=10 | at 07:55
echo radcelf-chirp-init.py --chirps_per_sec=10 | at 08:55
echo radcelf-chirp-init.py --chirps_per_sec=10 | at 09:55
echo radcelf-chirp-init.py --chirps_per_sec=10 | at 10:55
echo radcelf-chirp-init.py --chirps_per_sec=10 | at 11:55
```

On the HOST computer, there could be a similar arrangement to capture the data

```
echo acq400_stream.py --time=60min --output=UUT.0700.dat UUT | at 06:55
echo acq400_stream.py --time=60min --output=UUT.0800.dat UUT | at 07:55
echo acq400_stream.py --time=60min --output=UUT.0900.dat UUT | at 08:55
echo acq400_stream.py --time=60min --output=UUT.1000.dat UUT | at 09:55
echo acq400_stream.py --time=60min --output=UUT.1100.dat UUT | at 10:55
echo acq400_stream.py --time=60min --output=UUT.1200.dat UUT | at 11:55
```

All normal alternatives to HOST-PULL streaming are available.
For example, ACQ1001 can still remote mount an NFS disk and stream locally to the local mount

eg assuming an nfs mount /nfs

```
echo timeout -s 3600 nc localhost 4210 > /nfs/UUT.0700.dat | at 06:55
echo timeout -s 3600  nc localhost 4210 > /nfs/UUT.0700.dat | at 07:55
echo timeout -s 3600  nc localhost 4210 > /nfs/UUT.0700.dat | at 08:55
echo timeout -s 3600  nc localhost 4210 > /nfs/UUT.0700.dat | at 09:55
echo timeout -s 3600  nc localhost 4210 > /nfs/UUT.0700.dat | at 10:55
echo timeout -s 3600  nc localhost 4210 > /nfs/UUT.0700.dat | at 11:55
```

8.8.7 Boot time Configuration

```
acq1001_427> cat /mnt/local/sysconfig/transient.init
COOKED=0 NSAMPLES=100000 NCHAN=8 TYPE=LONG
transient PRE=2000 POST=2000 OSAM=1 DEMUX=1 SOFT_TRIGGER=1

# always trigger on CHIRPA
set.site 1 trg=1,3,1
# GPS sync
set.site 1 sync=1,0,1
# IOC not up yet, have to go low..
#set.site 0 SIG:SRC:SYNC:0=PPS
set.sys /dev/acq400.0.knobs/sig_src_route_sync_d0 3

/usr/local/CARE/make_spad_id
set.site 0 spad=1,4,0
set.site 0 spad1_us_clk_d1 1
set.site 0 spad1_us 1,3,1

spad2_monitor
spad3_monitor

run0 1
make-ch-server
```

8.8.8 In-Data Instrumentation

- Per sample, we have:
  - N columns of AI data, 32 bit
    - 24 bit signed ADC value {31:8}
    - 8 bit channel ID {7:0} 0x20, 0x21, 0x22.. 0x27
  - 4 columns Scratchpad (SPAD), each unsigned
    - SPAD[0] : Sample number
    - SPAD[1] : Clock count (6.25Mhz clock)
    - SPAD[2] : EVENT LATCH Count : sample count at PPS edge
    - SPAD[3] : KTIME_T, updated at 1kHz. Kernel ktime_t is 64 bit, we encode 32 bits as follows:
      - 20 bits time_t seconds {31:12}
      - 12 bits milliseconds {7:0} actual range : 0..999

The firmware includes a handy tool to check KTIME_T:

```
acq1001_427> /usr/local/CARE/spad3_timestamp_check
Wed Oct 21 18:37:36 UTC 2020 5f907ff0 0x07ff014d

Where :
5f907ff0          is the current time_t
   07ff014d       is the KTIME_T, with 20 bit seconds, followed by msec.
```

8.8.9 Factory Test

8.8.9.1 On Each Box

The text box represents a terminal session. Leave it running:

```
radcelf-tune-pps.py
radcelf-tune-pps.py --fine=1
```

8.8.9.2 On Master Box

```
acq1001_427> export SITECLIENT_TRACE=1
acq1001_427> radcelf-chirp-init.py --debug=1 --stop acq1001_427 acq1001_428
```

8.8.9.3 On Host

```
nc acq1001_427 53667 | pv > acq1001_427.5.dat
```

```
nc acq1001_428 53667 | pv > acq1001_428.5.dat
```

nb: nc UUT SPYPORT is shown for convenience, a real system might use **`acq400_stream2.py`** and it might also run on the UUT to stream to an NFS mount.

We propose using the SPYPORT to receive data rather than the STREAMPORT (4210), since opening the SPYPORT doesn't affect the start up sequence in the way that connecting to STREAMPORT does (ie it does not cause the capture system to ARM and possibly run); in otherwords, it's easier to set up a complex sequence, and the single program **`radcelf-chirp-init.py`** is then responsible for the entire start up sequence.

8.8.9.4 On Master Box

```
radcelf-chirp-init.py --chirps_per_sec=5 --gps_sync=5 acq1001_427 acq1001_428

sleep 60

radcelf-chirp-init.py --stop acq1001_427 acq1001_428
```

8.8.9.5 On Host

- First, look at AI data [0..7,8] and sample count

```
[peter@andros ~]$ hexdump -e '12/4 "%08x," "\n"' acq1001_427.5.dat | cut -d, -
f1-9 | head
30753820,007f1121,00816822,00760023,0076d124,00000f25,009a7826,00760327,00000001
338cb620,007e4621,00810722,0075a523,00768824,ffffab25,009a2b26,0070fb27,00000002
31680720,007a1921,007e3022,00722523,0073a124,ffff3025,00969626,00630e27,00000003
33306c20,00786121,007d0722,0070eb23,00727024,fffefd25,00954426,005ce527,00000004
31d5df20,0074ee21,007a9422,006e3f23,00700a24,fffe8225,00926926,00525227,00000005
```

- Second : look at Instrumentation Data [8,9,10,11,12]
- SPAD0 : Sample count : starts at zero, increments in 1's
- SPAD1: ADC modulator clock count (6.25MHz). Starts at 0x5000+ ???
- SPAD2 : SAMPLE count at Event Latch (Event1=CHIRP), starts at zero
- SPAD3 : KTIME_T

```
hexdump -e '12/4 "%08x," "\n"' acq1001_427.5.dat | cut -d, -f9,10,11,12 | less
00000001,00005139,00000000,082b5003
00000002,00005339,00000000,082b5003
00000003,00005539,00000000,082b5003
00000004,00005739,00000000,082b5003
00000005,00005939,00000000,082b5003
…
0000098b,00136539,00000000,082b50cb
0000098c,00136739,00000000,082b50cb
0000098d,00136939,00000989,082b50cb
0000098e,00136b39,00000989,082b50cb
0000098f,00136d39,00000989,082b50cb
```

8.8.9.6 Compare instrumentation on two boxes:

```
dt100@andros ~]$ cat bin/pasty
paste \
<(hexdump -e '12/4 "%08x," "\n"' $1 | cut -d, -f9,10,11,12)\
<(hexdump -e '12/4 "%08x," "\n"' $2 | cut -d, -f9,10,11,12)
```

- **Analysis of next page: START**

- 6Mhz clock count at start: 0x50d8, 0x50d9 : remote UUT's agree to 0.16usec.
- Initial count 0x50d0 is the 40 sample initial pipeline
- First PPS Latch is a relic from previous shot, please ignore
- NTP / KTIME-T agrees to 1msec.

- **First Event**
- First event latch value 0x989. This is inserted by SOFTWARE AFTER the event. The event position is found at sample index 0x989, 2441 Decimal.
- Time to first event : 0x989 => 2441 * 512/6.25M = 0.200 s
- Time to first event from NTP => 0xca-0x02 = 200msec

- **Second Event**
- Second event latch value 0x1312, This is inserted by SOFTWARE AFTER the event. The event position is found at sample index 0x1312, 4882 decimal.
- Time to second event: 0x1312 => 4882*512/6.25Mhz = 0.400s
- Time to second event NTP: 0x190 → 400msec.

A python analysis program is provided to validate over long periods: **`radcelf_sync_test.py`**

```
pasty acq1001_427.dat acq1001_428.dat | grep -n . | less


SampleD, SampleX, 6MHz clocks, PPS Latch, NTP Time

   1:00000001,000050d8,000eb703,98deb002       00000001,000050d9,000f7f48,98deb003
   2:00000002,000052d8,00000000,98deb002       00000002,000052d9,000f7f48,98deb003
   3:00000003,000054d8,00000000,98deb002       00000003,000054d9,000f7f48,98deb003
   4:00000004,000056d8,00000000,98deb002       00000004,000056d9,000f7f48,98deb003
   5:00000005,000058d8,00000000,98deb002       00000005,000058d9,000f7f48,98deb003
   6:00000006,00005ad8,00000000,98deb002       00000006,00005ad9,000f7f48,98deb003
   7:00000007,00005cd8,00000000,98deb003       00000007,00005cd9,000f7f48,98deb003
   8:00000008,00005ed8,00000000,98deb003       00000008,00005ed9,000f7f48,98deb003
   9:00000009,000060d8,00000000,98deb003       00000009,000060d9,000f7f48,98deb003

2441:00000989,001360d8,00000000,98deb0ca       00000989,001360d9,00000000,98deb0cb
2442:0000098a,001362d8,00000000,98deb0ca        0000098a,001362d9,00000000,98deb0cb
2443:0000098b,001364d8,00000000,98deb0ca        0000098b,001364d9,00000000,98deb0cb
2444:0000098c,001366d8,00000989,98deb0ca        0000098c,001366d9,00000000,98deb0cb
2445:0000098d,001368d8,00000989,98deb0ca        0000098d,001368d9,00000000,98deb0cb
2446:0000098e,00136ad8,00000989,98deb0ca        0000098e,00136ad9,00000000,98deb0cb
2447:0000098f,00136cd8,00000989,98deb0ca        0000098f,00136cd9,00000000,98deb0cb
2448:00000990,00136ed8,00000989,98deb0cb        00000990,00136ed9,00000000,98deb0cb
2449:00000991,001370d8,00000989,98deb0cb        00000991,001370d9,00000000,98deb0cb
2450:00000992,001372d8,00000989,98deb0cb        00000992,001372d9,00000000,98deb0cb
2451:00000993,001374d8,00000989,98deb0cb        00000993,001374d9,00000000,98deb0cb
2452:00000994,001376d8,00000989,98deb0cb        00000994,001376d9,00000000,98deb0cb
2453:00000995,001378d8,00000989,98deb0cb        00000995,001378d9,00000989,98deb0cb
2454:00000996,00137ad8,00000989,98deb0cb        00000996,00137ad9,00000989,98deb0cc
2455:00000997,00137cd8,00000989,98deb0cb        00000997,00137cd9,00000989,98deb0cc
2456:00000998,00137ed8,00000989,98deb0cb        00000998,00137ed9,00000989,98deb0cc
2457:00000999,001380d8,00000989,98deb0cb        00000999,001380d9,00000989,98deb0cc


4879:0000130f,00266cd8,00000989,98deb192        0000130f,00266cd9,00000989,98deb192
4880:00001310,00266ed8,00000989,98deb192        00001310,00266ed9,00000989,98deb192
4881:00001311,002670d8,00000989,98deb192        00001311,002670d9,00000989,98deb192
4882:00001312,002672d8,00000989,98deb192        00001312,002672d9,00000989,98deb192
4883:00001313,002674d8,00000989,98deb192        00001313,002674d9,00000989,98deb193
4884:00001314,002676d8,00000989,98deb192        00001314,002676d9,00000989,98deb193
4885:00001315,002678d8,00001312,98deb192        00001315,002678d9,00000989,98deb193
4886:00001316,00267ad8,00001312,98deb192        00001316,00267ad9,00000989,98deb193
4887:00001317,00267cd8,00001312,98deb192        00001317,00267cd9,00000989,98deb193
4888:00001318,00267ed8,00001312,98deb192        00001318,00267ed9,00000989,98deb193
4889:00001319,002680d8,00001312,98deb192        00001319,002680d9,00000989,98deb193
4890:0000131a,002682d8,00001312,98deb193        0000131a,002682d9,00000989,98deb193
4891:0000131b,002684d8,00001312,98deb193        0000131b,002684d9,00000989,98deb193
4892:0000131c,002686d8,00001312,98deb193        0000131c,002686d9,00000989,98deb193
4893:0000131d,002688d8,00001312,98deb193        0000131d,002688d9,00000989,98deb193
4894:0000131e,00268ad8,00001312,98deb193        0000131e,00268ad9,00001312,98deb193
4895:0000131f,00268cd8,00001312,98deb193        0000131f,00268cd9,00001312,98deb194
4896:00001320,00268ed8,00001312,98deb193        00001320,00268ed9,00001312,98deb194
4897:00001321,002690d8,00001312,98deb193        00001321,002690d9,00001312,98deb194
```

9 Appendix

9.1 Software Install

9.1.1 Install Fresh Release

acq400-268-20201012082516 or newer

Customize after install as follows:

```
mv /mnt/packages.opt/35-radcelf* /mnt/packages
mv /mnt/packages.opt/38-custom_hapi-* /mnt/packages
mv /mnt/packages.opt/99-*gpsntp* /mnt/packages
```

9.1.2 U-Boot settings.

NB: all units shipped with RADCELF fitted already have this selection, so it's NOT necessary to do it again

```
Connect serial console
Press <SPACE> within 3s of power up
from the u-boot prompt:
u-boot> setenv devicetree_image dtb.d/acq1002r.dtb
u-boot> saveenv
u-boot> boot
```

10 Appendix: Production Self-Tests

10.1 DDS

- run this script: /usr/local/CARE/radcelf-init-123
- Check this result:
- 20MHz becomes 18.75, 21.09 and 23.4 MHz on ddsA, ddsB, ddsC, respectively.

See Web Page 2.2

10.2 Aux DIO

- connect a 16 way header cable from J1 to J2.
- run this script:

```
acq1001_153> /usr/local/CARE/rad_aux_test
loopback J1 to J2
make all dX outputs, active high
OUT:0 0 0 0 0 0 0 0 BACK:0 0 0 0 0 0 0 0 PASS 0 0 0 0 0 0 0 0
OUT:1 0 0 0 0 0 0 0 BACK:1 0 0 0 0 0 0 0 PASS 2476 0 0 0 0 0 0 0
OUT:0 1 0 0 0 0 0 0 BACK:0 1 0 0 0 0 0 0 PASS 0 2478 0 0 0 0 0 0
OUT:0 0 1 0 0 0 0 0 BACK:0 0 1 0 0 0 0 0 PASS 0 0 2476 0 0 0 0 0
OUT:0 0 0 1 0 0 0 0 BACK:0 0 0 1 0 0 0 0 PASS 0 0 0 2476 0 0 0 0
OUT:0 0 0 0 1 0 0 0 BACK:0 0 0 0 1 0 0 0 PASS 0 0 0 0 2485 0 0 0
OUT:0 0 0 0 0 1 0 0 BACK:0 0 0 0 0 1 0 0 PASS 0 0 0 0 0 2493 0 0
OUT:0 0 0 0 0 0 1 0 BACK:0 0 0 0 0 0 1 0 PASS 0 0 0 0 0 0 2480 0
OUT:0 0 0 0 0 0 0 0 BACK:0 0 0 0 0 0 0 0 PASS 0 0 0 0 0 0 0 0
PASS: 9/9 PASS
```

10.3 USB

Connect USB stick and confirm it's available.

It's necessary to set bConfigurationValue to force power up.

```
echo 1 > /sys/bus/usb/devices/1-1.1/bConfigurationValue
mkdir /sd
mount /dev/sda1 /sd
```

10.4 Check all signal outputs

Run the dual chirp and use a scope to monitor (not covered in this guide)

11 Appendix: AD9854 Quickref

Table IV. Register Layout

Parallel Address	Serial Address	AD9854 Register Layout								
Hex	Hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Default Value
00 01	0 POTW1	Phase Adjust Register #1 <13:8> (Bits 15, 14 don't care) Phase Adjust Register #1 <7:0>					Phase 1			00h 00h
02 03	1 POTW2	Phase Adjust Register #2 <13:8:> (Bits 15, 14 don't care) Phase Adjust Register #2 <7:0>					Phase 2			00h 00h
04 05 06 07 08 09	2 FTW1	Frequency Tuning Word 1 <47:40> Frequency Tuning Word 1 <39:32> Frequency Tuning Word 1 <31:24> Frequency Tuning Word 1 <23:16> Frequency Tuning Word 1 <15:8> Frequency Tuning Word 1 <7:0>					Frequency 1			00h 00h 00h 00h 00h 00h
0A 0B 0C 0D 0E 0F	3 FTW2	Frequency Tuning Word 2 <47:40> Frequency Tuning Word 2 <39:32> Frequency Tuning Word 2 <31:24> Frequency Tuning Word 2 <23:16> Frequency Tuning Word 2 <15:8> Frequency Tuning Word 2 <7:0>					Frequency 2			00h 00h 00h 00h 00h 00h
10 11 12 13 14 15	4 DFR	Delta Frequency Word <47:40> Delta Frequency Word <39:32> Delta Frequency Word <31:24> Delta Frequency Word <23:16> Delta Frequency Word <15:8> Delta Frequency Word <7:0>								00h 00h 00h 00h 00h 00h
16 17 18 19	5 UCR	Update Clock <31:24> Update Clock <23:16> Update Clock <15:8> Update Clock <7:0>								00h 00h 00h 40h
1A 1B 1C	6 RRCR	Ramp Rate Clock <19:16> (Bits 23, 22, 21, 20 don't care) Ramp Rate Clock <15:8> Ramp Rate Clock <7:0>								00h 00h 00h
1D	7 CR	Don't Care CR [31]	Don't Care	Don't Care	Comp PD	Reserved, Always Low	QDAC PD	DAC PD	DIG PD	10h
1E		Don't Care	PLL Range	Bypass PLL	Ref Mult 4	Ref Mult 3	Ref Mult 2	Ref Mult 1	Ref Mult 0	64h
1F		CLR ACC 1	CLR ACC 2	Triangle	SRC QDAC	Mode 2	Mode 1	Mode 0	INT/EXT Update Clk	01h
20		Don't Care	Bypass Inv Sinc	OSK EN	OSK INT	Don't Care	Don't Care	LSB First	SDO Active CR [0]	20h
21 22	8 IPDMR	Output Shape Key I Mult <11:8> (Bits 15, 14, 13, 12 don't care) Output Shape Key I Mult <7:0>								00h 00h
23 24	9 QPDMR	Output Shape Key Q Mult <11:8> (Bits 15, 14, 13, 12 don't care) Output Shape Key Q Mult <7:0>								00h 00h
25	A SKRR	Output Shape Key Ramp Rate <7:0>								80h
26 27	B QDACR	QDAC <11:8> (Bits 15, 14, 13, 12 don't care) QDAC <7:0> (Data is required to be in two's complement format)								00h 0