nrao_adc5g_test.py

Usage:

>>> python2.7 nrao_adc5g_test.py -r *vegasr2-8* -b *<bofffile_of_current_mode>*

(alternate set up):

>>> python2.7 nrao_adc5g_test -r srbsr2-1 -l -3

This will program the roach with designated bof file, do MMCM calibration, check the synthesizer remote control is working and we can see sine wave shapes in snapshots (manual input required), and do OGP calibration (with default test tone of 18.3105MHz at 3.0dBm) and finally INL adjustment. If these three adjustments completed successfully, there's option for manually injecting different test tones (frequencies and amplitudes) and showing the raw adc snapshot and spectrum.

Notes on manual mode operations:

1. If the OGP calibration is enabled (as default) or the manual testing mode is active, the program will first try to establish a connection with the synthesizer and take a snapshot of the raw ADC outputs. Please verify that the raw data plot is roughly at the correct frequency (testfreq option) and its amplitude is of reasonable range. Then please close the plots to proceed.
2. If prompted to select between 'Y' and 'N', please enter only one of those (sorry for the sloppy coding)...
3. Quite a few plots will pop up in this mode.

Notes on options:

- To skip the manual synthesizer verification and manual testing after adjustments, add "-m 0". The code should complete automatically without requiring any manual input. We can check the result of calibration later.
- To skip testing after the adjustments, add "-t 0"
- To use a higher power test tone (default at 3.0), use "-l *<ampl>*"
- To use some specific frequency sine wave as test tone, use "-f *<freq>*"
- If for some reason OGP failed and we need to try again but don't want to repeat the "program the FPGA and do the MMCM adjustment" part, use "-p" to skip those (go straight into the OGP calibration).
- To calibrate only one of the ADCs, use "-z *<zdok>*" (default=2 referring to both ADCs)
- To skip OGP adjustment, use "-o 0"
- To skip INL adjustment, use "-i 0" (INL will only be done if OGP has been completed)
- To skip saving *.PNG plots, use -S 0 (capital S)
- To skip viewing the plots interactively, use -V 0
- More elaborated, step-by-step manual adjustments will be described in another section (step-by-step, scroll down a few pages).
- Examples:
    - Read the help file (usage, default values, etc.)
      >>> python2.7 nrao_adc5g_test.py -h

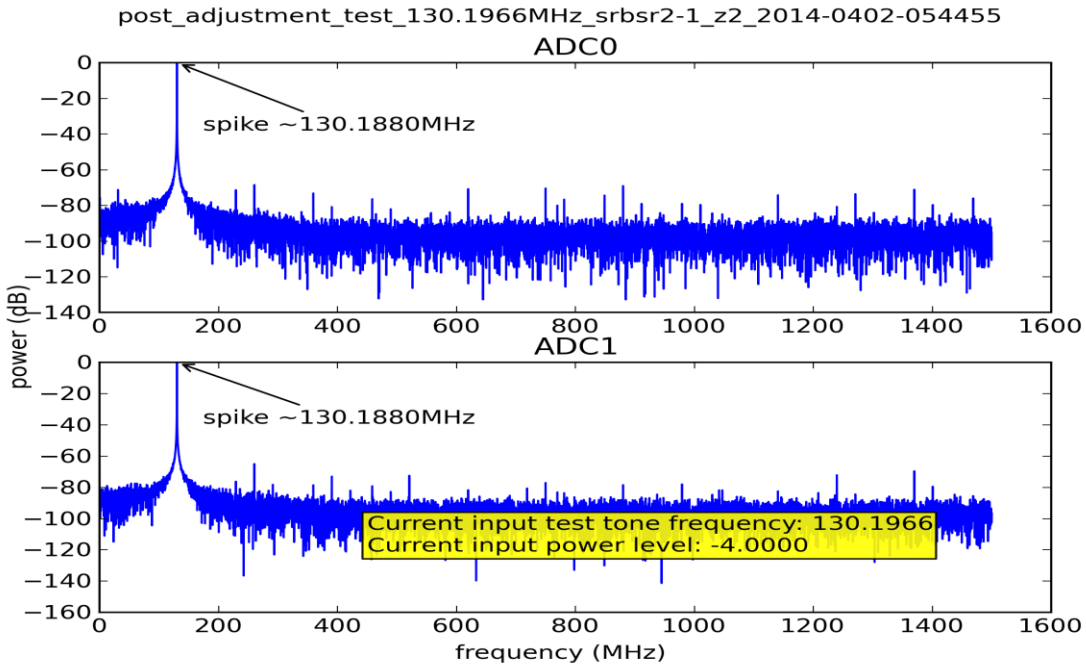
- Do MMCM calibration for ADC0 (skip OGP)
  >>> python2.7 nrao_adc5g_test.py -z 0 -o 0
- Do MMCM/IODELAYS/OGP for ADC1
  >>> python2.7 nrao_adc5g_test.py -z 1 -d 1
  We default to skip IODELAYS adjustment as its effect for our case (~1.5G clock rate) hasn't been consistent or particularly helpful in my experiments, so I would recommend that we opt not to do it.
- **TEST only (right after a calibration is done and assume the roach is programmed with one of the vegas bof file)**:
  - >>> python2.7 nrao_adc5g_test.py -r <hostname> -l -3 -p -o 0 -i 0
  - Notes: **-r <hostname>** to select roach name/IP; **-l -3** to set power level to -3; **-p** to skip reprogramming the roach; **-o 0** to skip the OGP calibration; **-i 0** to skip the INL calibration.
- **Automate everything (no use input required, run through all calibration and some testing)**
  - >>> python2.7 nrao_adc5g_test.py -r <hostname> -l -3 -m 0
  - Notes: **-m 0** to disabled manual mode

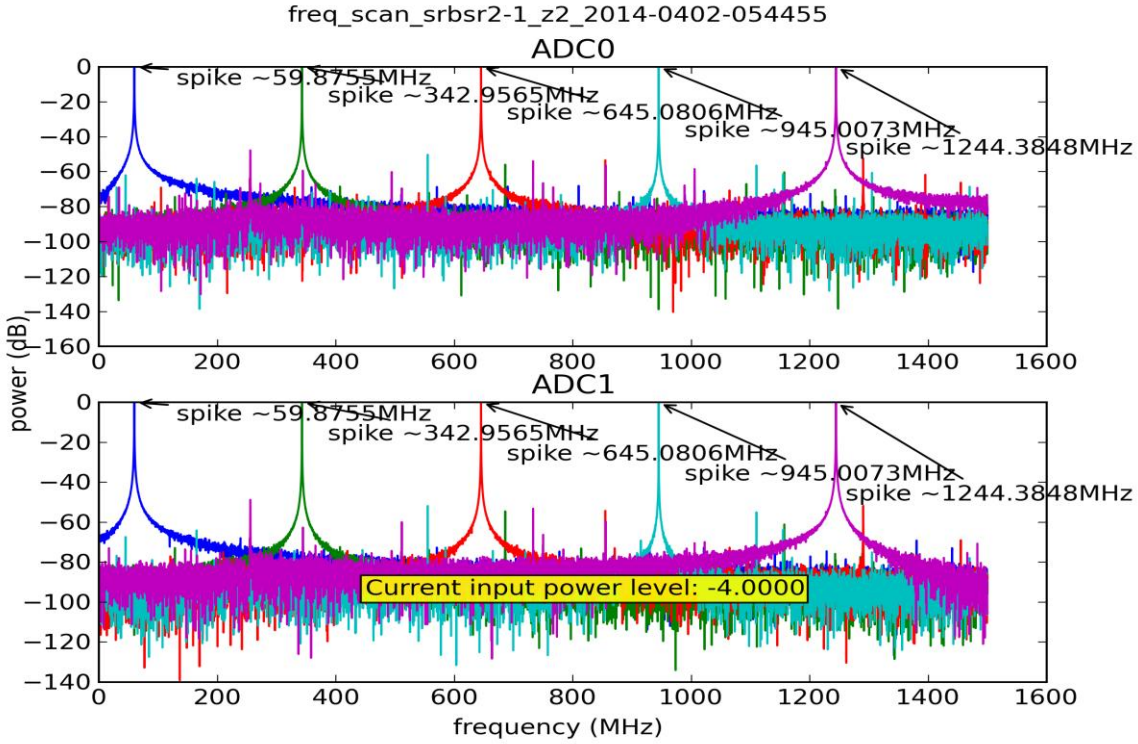
Testing procedures:

After the adjustments, there are options for testing (or we can also skip to the testing part directly). At this moment it's quite naive - injecting a test tone, and show the spectrum (of a snapshot of data, 16382 points). The program will try to find the location of the highest spike and annotate it on the plots and in the log file.
We can also do a frequency scan which is select a series of test tones with frequency going from 0 to nyquist (with slight randomization) and plot multiple spectra in one plot.

Logs and plots:

A log file will be generated as, with the verbosity level set by the argument -v. Watch out for “WARNING”s as they might hint some possible problems such as input power level too high, etrc. If “save” argument is active (as default), then plots will be automatically saved in *.PNG format.





Some of the output files include:

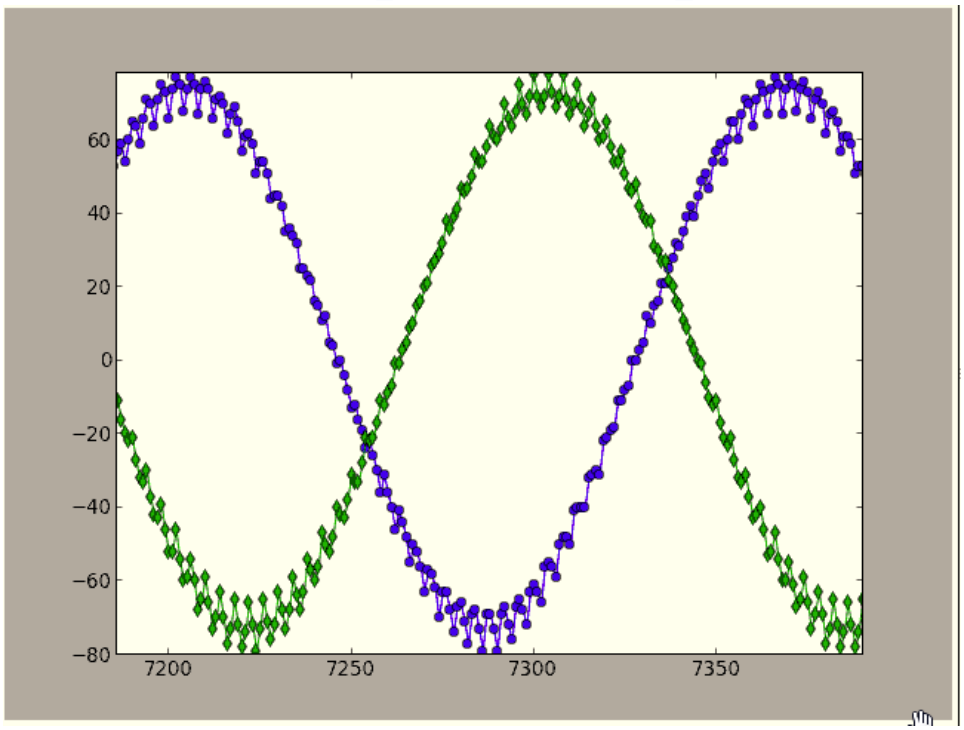
- adc5g_adjmt_<roach-name/ip>_z<zdok>_<timestamp>.log:
    - log file (text)
- ramp_check_<roach-name/ip>_z<zdok>_<timestamp>.png:
    - If MMCM is carried out, this on is to verify that the output ramp in test mode shows no glitch (ramps running from 0 to 255)
- post_ramp_check_<roach-namae/ip>_z<zdok>_<timestamp>.png:
    - This one follows the previous ramp_check_<>.png to make sure the ADC is unset from test mode (it should look like something that resembles to the actual input to the ADCs - synthesizer/noise/actual antenna input/etc.)
- raw_startup_<roach-name/ip>_z<zdok>_<timestamp>.png:
    - If OGP/INL or post-adjustment test is enabled, this will be used to verify that the synthesizer is connected correctly and the remote script is working. This should look like a sine wave at test_freq (default at 18.3105MHz) with reasonable amplitude (depending on the input parameter -l).
- snapshot_adc<0/1>_raw_<roach-name/ip>_z<zdok>_<timestamp>.dat<other_id>, ogp_<roach-name/ip>_z<zdok>_<timestamp>, inl_<roach-name/ip>_z<zdok>_<timestamp><other_id>:
    - Data files containing the measured results produced during OGP/INL
- post_adjustment_test_<test_freq>_<roach-name/ip>_z<zdok>_<timestamp>.png:
    - Shows the spectrum of a snapshot after injecting a certain kind of test tone. An attempt to locate the test frequency from the spectrum will be made and the result annotated (and logged) as well as the info of the test tone.
- freq_scan_<roach-name/ip>_z<zdok>_<timestamp>.png:
    - Selected spectra of several slightly randomized test tone spanning 0 to nyquist.

Expected results:

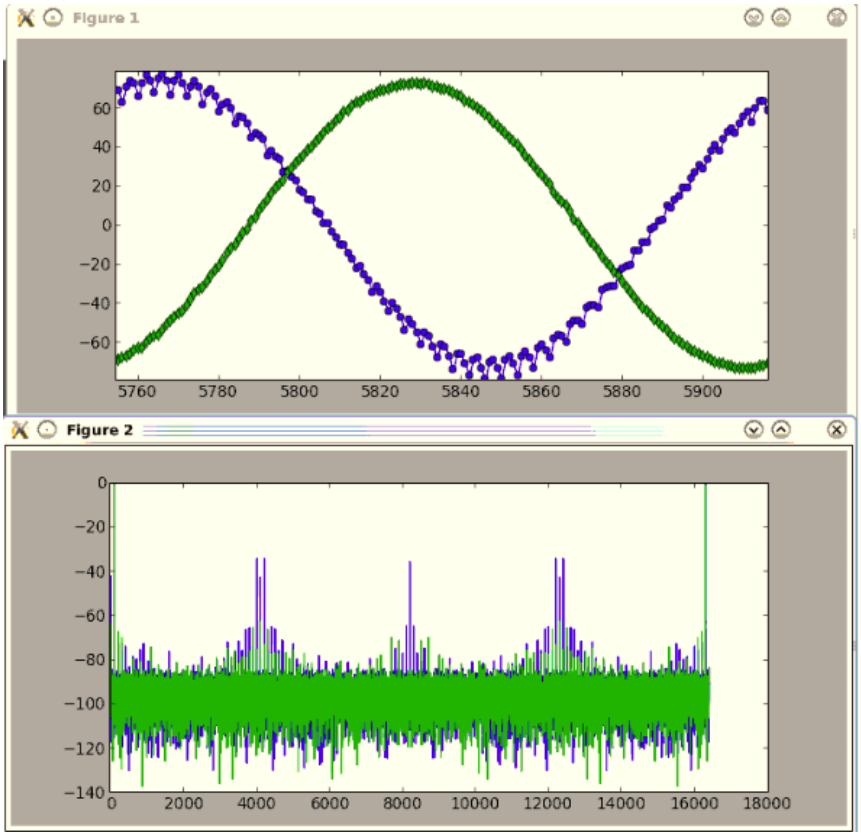
With low power level sine wave signals (raw data amplitude <20), all noise spikes (including those resulted from interleaving) should still be consistently below -35dB. Snapshot taken with 16384 data points for a single spectrum. (This is not meant to simulate the FPGA designs as it doesn't take into account of number of FFT channels for different modes, acc_len, etc.)

- MMCM - Ideally we would like to have some monitor unit in the design, but I recall that many of the VEGAS designs are quite full and re-compiling might be a pain. For the experiments I've done with vegasr2-8 and at BWRC, I don't really see them slipping away from calibrated state (confirming Jack Hickish's observations), so my guess is we can do without them.
- OGP - The results have been good with this (some hacking here and there in the code which might add some robustness problem, but I've tried to keep as close to the original repo as possible). We try to use a low frequency and relatively high power (amplitude >64, as we would like to use all 8 bits) test tone, default to 18.3105MHz (*quote **"picked to have an odd number of cycles in the 16384 point snapshot"** as in*

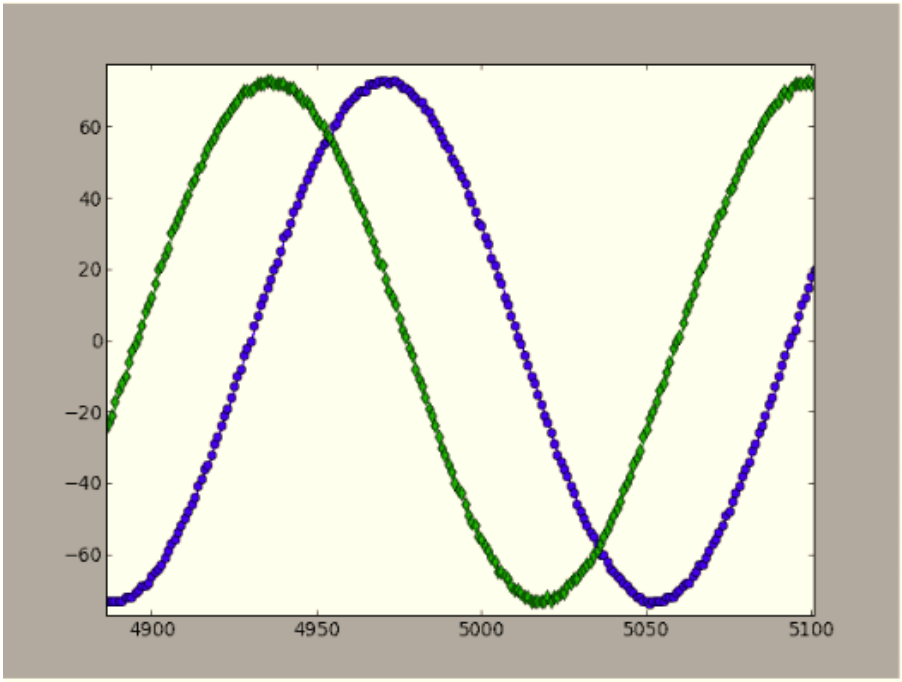
https://github.com/jack-h/adc_tests/blob/master/rww_tools.py#L221)



Before OGP calibration - we can see the 4 cores of each ADCs are not very well calibrated



After doing OGP calibration for ADC1



After OGP calibration

Step-by-step manual calibration:

1. Enter ipython (assume it's installed and uses python2.7)
2. >>> **execfile('nrao_util.py')**
3. >>> **fpga = corr.katcp_wrapper.FpgaClient(*<hostname>*, 7147)**
4. >>> **fpga.progdev(*'<boffile>'*)**
5. >>> **fpga.est_brd_clk()**
   Verify that the clock rate is about 187.5MHz
6. >>> **do_mmcm(*<zdok>*)**
   Use do_mmcm(2) to perform the MMCM adjustments for both ADCs. Otherwise use do_mmcm(0) and/or do_mmcm(1). For details about this adjustment, see quotes from Rurik below.
7. >>> **check_ramp()**
   Use this to check if the MMCM adjustment indeed succeeded. We should be able to see plots showing ramps running from 0 to 255 for the four cores in each ADC in test mode. Another plot showing the regular snapshot from ADCs in data capturing mode will also be present (sine wave shapes if the synthesizer is connected).
8. >>> **gpib_init(*<ampl>*, *<test_freq>*)**
   Initiate the synthesizer connection, use usual test tone power level (0dBm? 3dBm?) and frequency (18.3105MHz?)

   Note: If this takes a really long time, the default IP address to the synthesizer might not be correct. Type "addr" or open gpib.py to check for the IP address, modify as necessary.

9. >>> **check_raw()**
   Check raw ADC data with the given input (ampl and test_freq). We may see some interleaving issue among the four cores of each ADCs.
10. >>> **check_spec()**
    Use this function the check the spectrum of the raw ADC input (16384 points FFT done with numpy.fft.fft). It's very likely we see some spikes resulted from not-well-calibrated interleaving, most notably at 0Hz, ¼ clock frequency (~4096 in the plot), ½ clock frequency - test_freq, etc.
11. >>> **ampl_setup()**
    Use this function to adjust the power level of the test signal (we want something that has big enough amplitude that uses all 8 bits, but not too big). Repeat and experiment a bit to find a suitable power level for out testing purpose. This function uses a global variable **ampl**.
12. >>> **freq_setup()**
    Use this function to adjust the test tone frequency. This function uses a global variable **test_freq**.
13. >>> **do_ogp(2, test_freq)**
    Do OGP adjustments for both ADCs. Use do_ogp(0, test_freq) if doing OGP for ADC0 only. If all goes well, the code should run without error. (python2.6 might throw some warning complaining about format code masking, but as far as I can see that's not a big issue). If this well, we can use check_raw() and check_spec() to see whether those artifacts spikes are suppressed to an acceptable level. The second argument **test_freq** here is the global variable we used for freq_setup().
14. >>> **do_inl(2)**
15. >>> **res0, res1, f0, f1, freqs = freq_scan()**
    Injecting slightly randomized test tone between 0 to clockrate (1.5GHz, half of sampling rate) and save the spectrum. freqs is an array of the test tones injected. f0 is an array of spectra from ADC0 (use plot(f0[i], 0<i<50 to plot a spectrum), similarly for f1. Or use something like
    >>> **for i in range(0, 49, 5):**
    plot(f0[i])
    to see if levels of undesirable spikes are consistently low enough across the band.
16. >>> **ampl_setup()**
    Adjust the power level and see the performance with lower power inputs.
17. Alternate/Extra:
    >>> **do_multifreq(0)**
    >>> **do_multifreq(1)**
    This will do OGP and INL at different test frequencies… It takes longer, and I haven't observed any significant improvement comparing to do_ogp() and do_inl(). Sometimes it might actually turn out worse. So I do not recommend as I haven't find ways to produce consistent results with it.
18.

Oddities and troubleshooting:

- MMCM: sometimes it fails to find an optimal phase; or check_ramp() might show that we still have glitchy ramps even after the do_mmcm() is successfully completed.
    - A few things to check:
        - Does the clock frequency agree with the frequency for which the bof file was compiled for?
        - Are all the cables connected securely and the ADC cards firmly inserted in the ZDOK slots?
        - Is the bof file used a verified “good” one?
    - A few things to try:
        - Reprogram the roach and try do_mmcm(2) again...
        - Try a different bof file and repeat the MMCM adjustment process.
        - Look at the snapshots of raw ADC data (in normal data capturing mode) and see if it makes sense.
        - Repeat the MMCM adjustments a couple times, check_ramp() after each attempts, see if the problem is persistent, if it’s manifested in only one or both ADCs.
        - ?
    - If all those failed and the test ramp is still glitchy… we might have a real problem here.
- OGP: this may fail for different reasons…
    - A few things to check
        - Is the synthesizer remote control working correctly? The actual input test tone must be the same as the ‘fr’ argument for function rww_tools.dosnap(fr=...)
        - Is the second argument of function do_ogp() assigned correctly?
        - What’s the power level of the input test tone? This should be in reasonable range, if it’s too small (barely distinguishable sine wave shape) or too big (clipping), it’s very likely to fail.
        - Is the MMCM adjustment completed successfully before attempting OGP adjustments? (check_ramp() no glitch)
    - A few things to try:
        -
- INL: In my experiments this almost never cause any problem. Not that it brings dramatic improvements, though.
- IODELAYS: At 1500MHz clock rate we can probably do without this. In fact this adjustment, when successfully completed, doesn’t make much difference. I have had a lot of consistency issues
- Write-verification error?: Hopefully more attempts, or the step-by-step method, or a reboot of the roach will work out...
- Other? Sometimes even when all available adjustment methods are attempted and seemingly completed without problem, there are still some persistent problem observable in the raw ADC data/spectrum.

- A few things to check:
  - Input?
    - Filter?
    - Interference?
    - ...
  - Is there any pattern in the undesirable spikes?
    - 0Hz?
    - ½ clock rate?
    - ¼ clock rate?
    - ½ clock rate - test tone?
    - multiples of the test tone?
  - BOF file?
    - Compiled for what clock rate?
    - Timing report?
    - ...

Quote from Rurik Primiani (7/9/13):

> First, the MMCM clock phase must be shifted to the center of the data eye. This is done by using the test ramp output of the ADC to check for glitches and choosing the optimal phase by eliminating glitches in the ramp. I've discussed this with Hong and Mark and there are some changes coming to this procedure to make it more robust; there are both yellow block and software changes in the pipeline but what's presently on Github should work.
>
> Secondly, there's the offset, gain, phase, and INL adjustments of the individual interleaved cores. This is a much more complicated business and it's something that Nimesh Patel and Bob Wilson have been working. We've had good results on this and there's a paper in the works for JAI that will go into the details of this procedure. There is also software hosted on Github that does this.