External Adjustment for Atmel/e2v iADC

Hong Chen, Mark Wagner
August 4, 2010

1.Introduction

With interleaving ADC's, higher sampling rates can be achieved, but in the process of interleaving, multiple matching problems may occur and prevent the interleaved ADC's from working as desired. This memo investigates this issue on a dual 8-bit 1Gsps ADC board(AT84AD001B)[1] made by e2v and implements python code, which adjust the ADC gain, offset and delay for significant improvements in performance.

A wiki page containing all source code mentioned in this memo has been created. (https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC)

2.Simulation

The three most common types of mismatches in a dual interleaved ADC are: Gain mismatch, Offset (DC bias), and Phase difference.

The Matlab code *(adc_simulation.m* https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#Simulation*)* performs the simulation for all three types of effect simultaneously and plots the spectrum to show the result. This allows us to be able to identify the error caused by offset, gain and delay when we are performing actual measurement and gives us a better idea of what we're looking for.

1. More information about the iADC, including data sheet, schematics, layouts, etc. is available at *http://casper.berkeley.edu/wiki/ADC2x1000-8*
   Datasheets are also available at
   https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#Datasheet

Conclusion of simulation:

Letting *Fs* be the combined sampling rate and *freq* be the input frequency of the test tone, we observe that:

1. The DC bias, or offset mismatch, will result in spikes at 0 Hz and *Fs*/2 Hz.

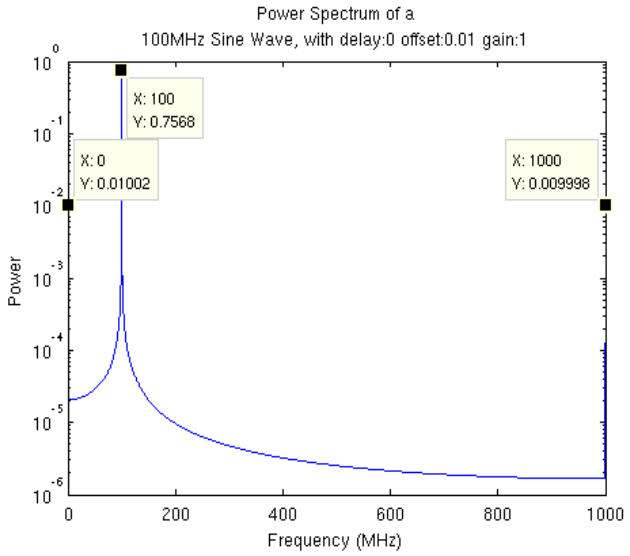


Figure 2-1

2. The gain produces spikes at the position of the supplementary angle, which is *Fs*/2-freq Hz.

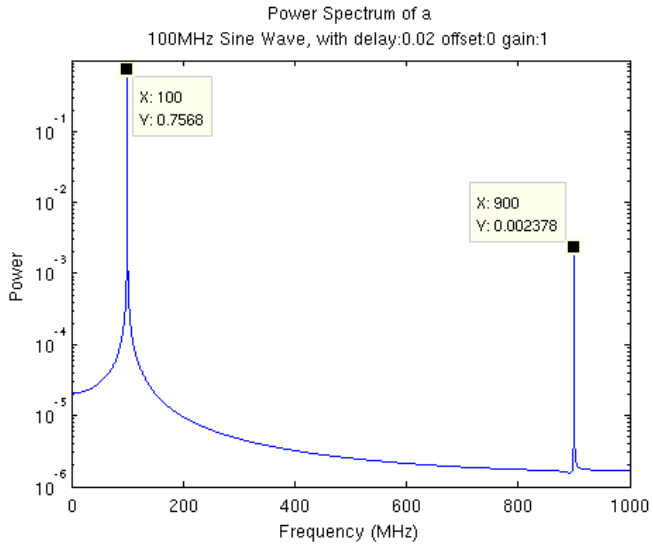


Figure 2-2

3. The delay also produces spikes at the position of the supplementary angle, which is *Fs*/2-freq Hz.

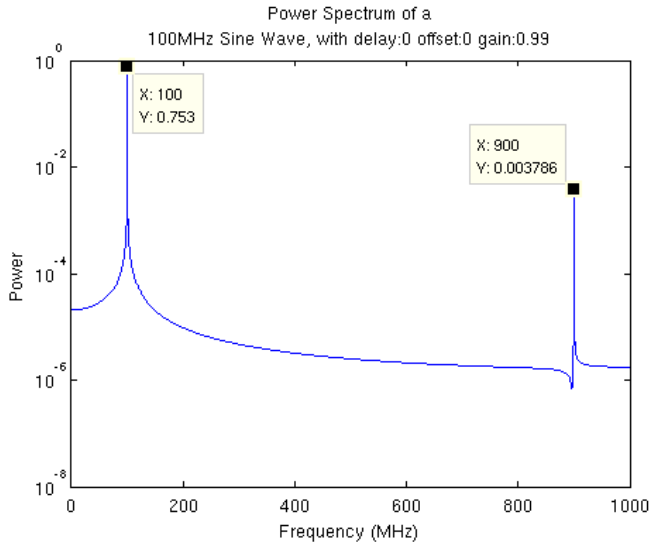


Figure 2-3

4. When three types of effects are simulated simultaneously, the above statements are still true, and no other significant spikes can be observed.

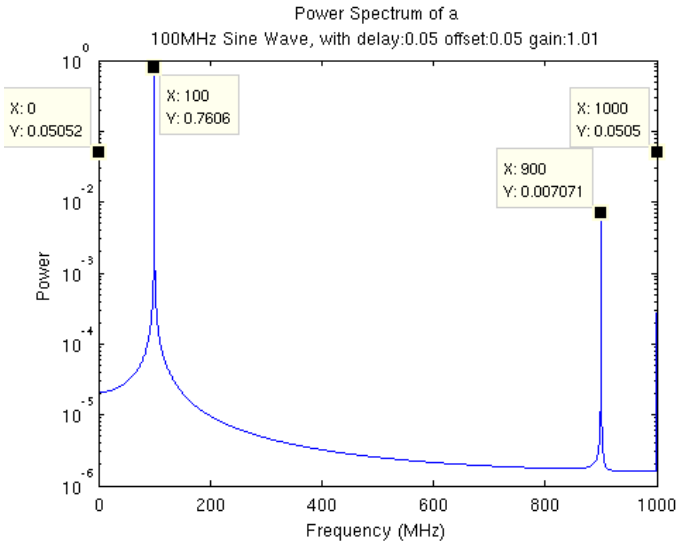


Figure 2-4

For more code, analysis and plots:
https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#More_Simulation

3.Test

To measure the performance of the interleaved ADC's, we connect it to a ROACH board, and run a design which can capture data and write it into shared memory for futher test and analysis on a CPU.

The .mdl file and .bof file for the design:

*(*https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#Test_Design*)*

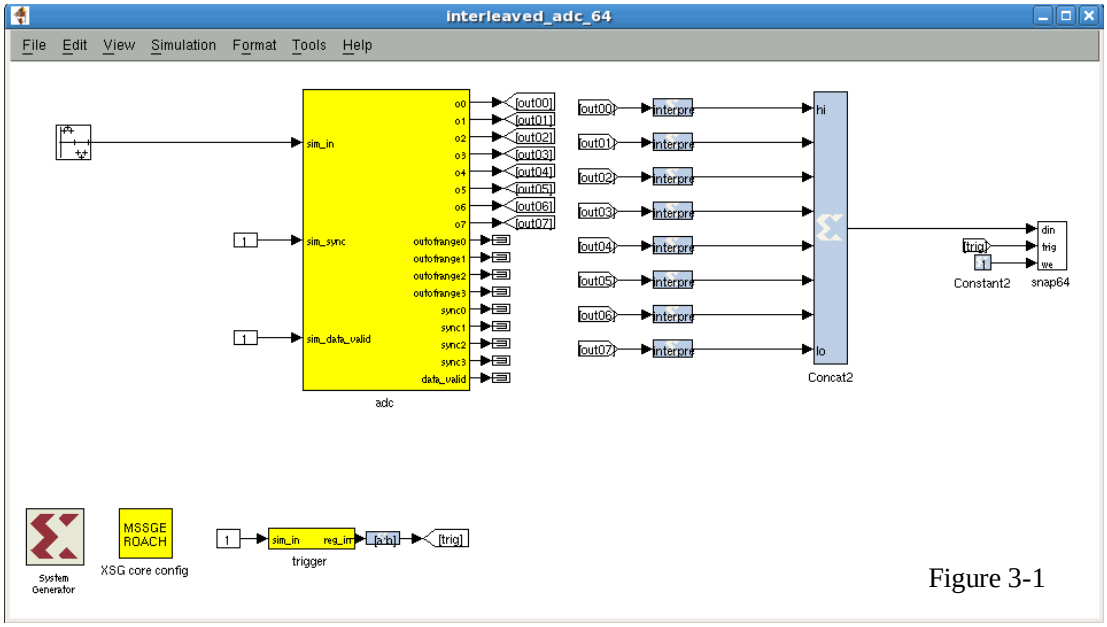


Figure 3-1

The data type coming out from the iADC[2] is signed 8.7. In order to put the data into shared memory, we reinterpret it as 8-bit unsigned 8.0. To get the maximal amount of sample points, we concatenate every eight samples and make a 64-bit unsigned number and send it to the CASPER snap64 block[3]. We also set the depth for snap64 block to 16, which is the largest integer allowed on ROACH. Now we will get 8*(2^16) sample points in each snapshot.

Once we get the .mdl file compiled successfully, we upload its BOF file to a ROACH board and run it. Interaction with the registers is to be done in python using the CORR and KATCP packages. We read data out from the snap64 block and convert it back to signed 8.7. We then use matlab to analyse the data and plot the spectrum. [Supplement Notes 1]

2. See documentation for CASPER adc block at *http://casper.berkeley.edu/wiki/Adc*
3. See documentation for CASPER snap64 block at *http://casper.berkeley.edu/wiki/Snap64*

Test Procedure with Python and Matlab code:

First we put in an arbitrary signal, then read raw data out of the snap64 block and write it to a data file.(*https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#maker.py*)

We then read the data file, performe an FFT and plot it.
(https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#fft_plot.m)

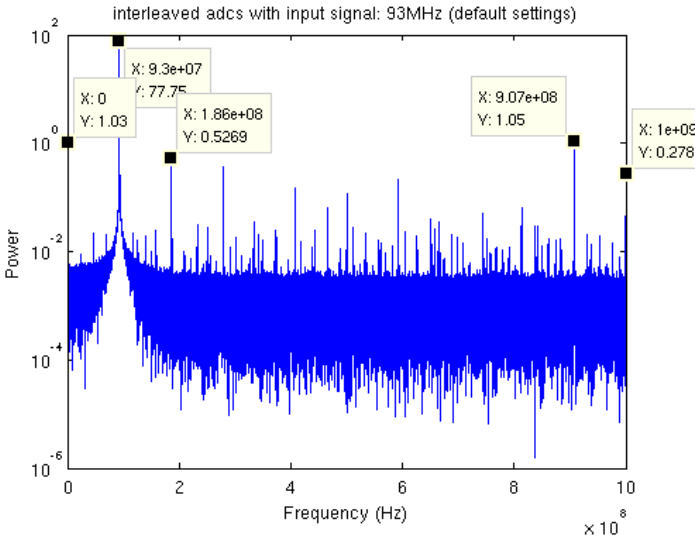
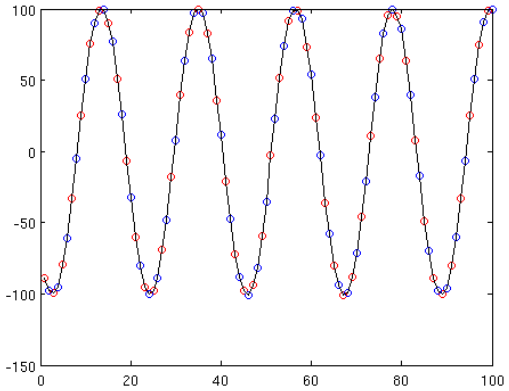


Figure 3-2

While the frequency for the test tone is 93MHz and the sampling rate is 2GHz, from the plot we observed some relatively high spikes compare to the major spike, which are more than 1 percent. Those spikes are at position 0Hz and 907MHz (= *Fs*/2 - *freq*), which match the pattern of our simulation results.

We plot the raw data to verify that both ADCs are working correctly. The plot should have the shape of the sine wave. To look at the raw data, I personally recommend putting in a relatively low frequency signal(~10MHz) to get enough sample points in each cycle. If the shape of the plot doesn't looks good, then the ADC probably is having some issue with hardware.

Matlab Code: https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#zoom.m

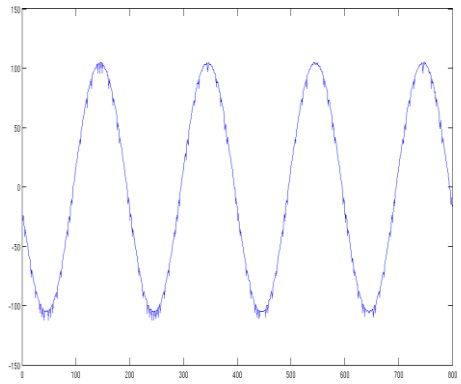


Plot Description:
Raw data from iADC

Figure 3-3

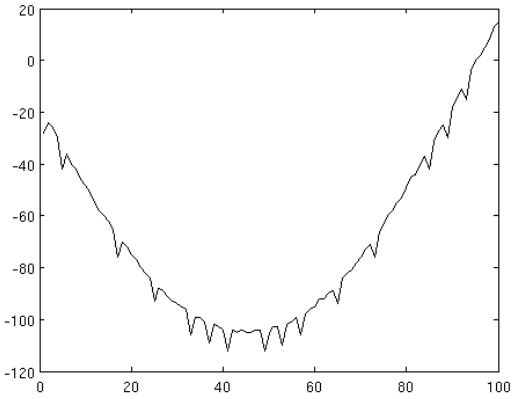
Figure 3-4~7 show a case when we came across a problematic iADC. Here we can see the two ADCs show very different plots.

Figure 3-4



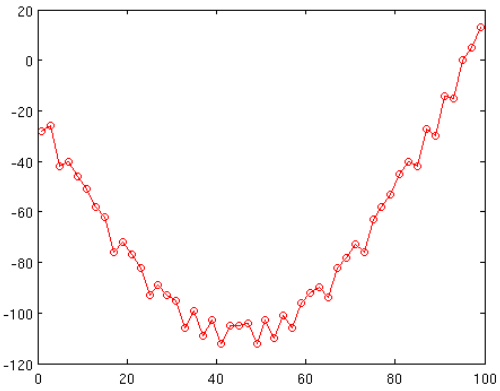
raw data from input

Figure 3-5



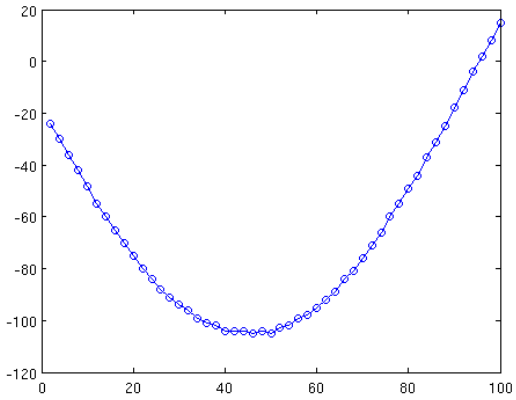
raw data from input with lower frequency input signal

Figure 3-6



Input data from ADC I

Figure 3-7



Input data from ADC Q

4. Set Up

From the test results, we can identify that there are some mismatching issues. The iADC offers internal digital calibration and external adjustment for gain, offset and delay. When the iADC is booted, the internal calibration will be automatically executed. Since after the internal digital calibration, the iADC still has some interleaving issues unresolved according to our test, we would like to adjust it externally to improve performance.

The data sheet for AT84AD001B has descriptions about the iADC's external control features starting on page 37 (page 32 for AT84AD001C)[4]. The registers on the iADC are not readable. In order to adjust them, we use the BORPH 'iadc_controller' software register. To quote David George:

> "To write data D into address A on ADC0, where D0 is the least significant byte and D1 is the most significant:
> fpga.blindwrite('iadc_controller','%c%c%c%c'%(D1,D0,A,0x1),offset=0x4)" [5]

A Memo about the 'iadc_controller' by David George can be found at *(*https://casper.berkeley.edu/wiki/images/0/0f/IADC-opb-controller.pdf*)*.
The values of address and DATA can be found in data sheet for the iADC.

The Python scripts used for our analysis are listed below:
1.config_setup.py:https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#config_setup.py
2.start.py: https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#start.py
3.analysis_function.py:
https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#analysis_function.py
4.tester.py: https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#tester.py
5.fft_out: https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#fft_out.py
6.iadc.py: https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#iadc.py
7.auto_adjust.py:
https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#auto_adjust.py

They can also be downloaded here.

4. Supplement Notes 1: Making sense of the data sheet

5. By David George.
For more examples and descriptions, please take a look at
https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#iadc.py

5. Adjust

Automatic Adjustment:

1. Run config_setup.py to connect to the ROACH board and run the design.
2. Run auto_adjust.py to perform the external adjustment.

Adjustment Procedure:

1. *Offset mismatch*[6]: We do this first so that we will get more accurate results in next step where we calculate the sine wave's amplitude.

In order to eliminate offset mismatch to the lowest possible level, we go through loops that adjusts offset compensation and checks the effect after each step until the result is reasonably good. We do this to each ADC separately. To check the effect, our approach is to capture the data, perform an FFT and measure the height of the spike at 0Hz. For each ADC, if the offset is 0, then there shouldn't be a spike at 0Hz.

2. *Gain mismatch*[7]: In this step we also deal with each ADC separately. We find the amplitudes for signals coming from both ADC's by calculating the RMS of the two data streams. We then calculate their difference, and adjust until a best match is found.

To get the smallest step, we adjust the gain by adjusting the "Gain Compensation" feature rather than "Analog gain adjustment".

6. Offset Compensation: page 38 in data sheet for AT84AD001B.
7. Gain Compensation: page 38 in data sheet for AT84AD001B.

3. <u>Phase mismatch:</u> According to the software simulation, we know that the phase mismatch would result in a spike at the position of the supplementary angle, which is (Fs/2-freq) Hertz.

Figure 5-1

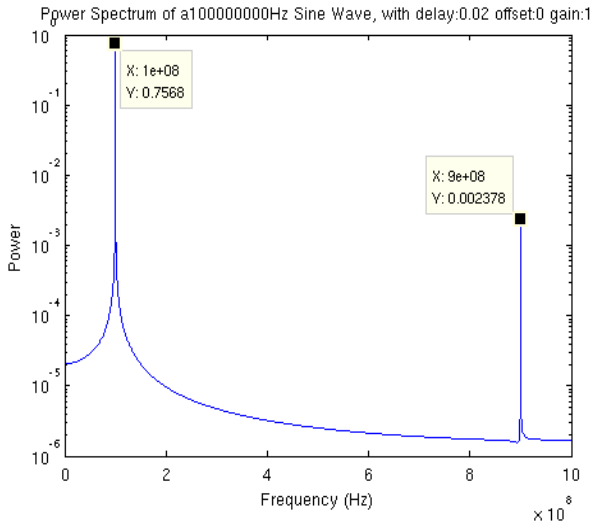
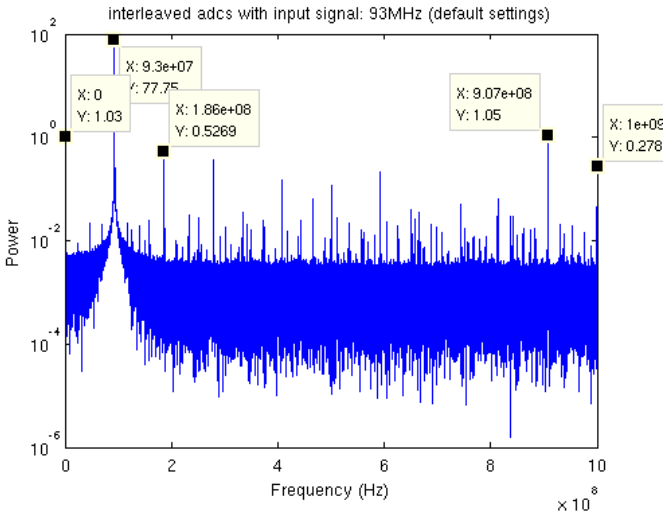


Figure 5-2



For our input signal of 93MHz, we observe a high spike at 907MHz, which is equal to (2GHz/2 - 93MHz). In order to find a best match for the phases, we perform the *Fine Sampling Delay Adjustment*[8] and check the specific spike in each step in our loops.

If all other spikes are low enough to be ignored in comparison to the major spike, then we can assume that the adjustment is successful.

8. Fine Sampling Delay Adjustment (FiSDA): page 39 in data sheet for AT84AD001B.

6.Result

1. Internal Calibration (either automatically executed or manually initiated):

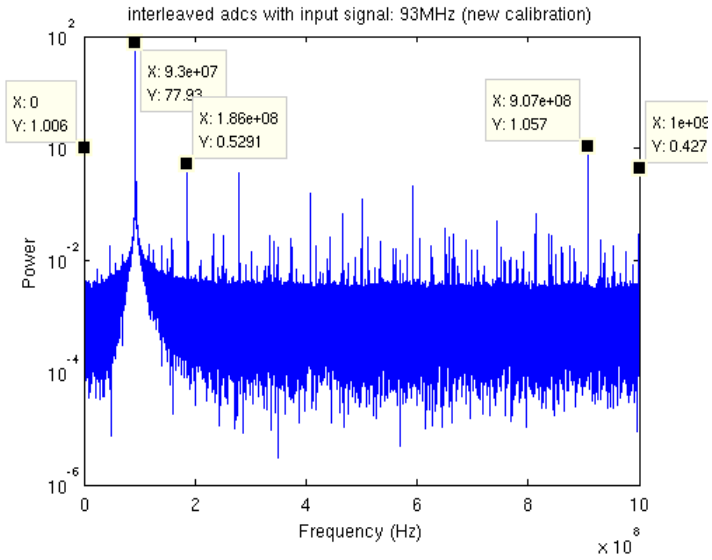


Figure 6-1

2. No Calibration Mode[9]:

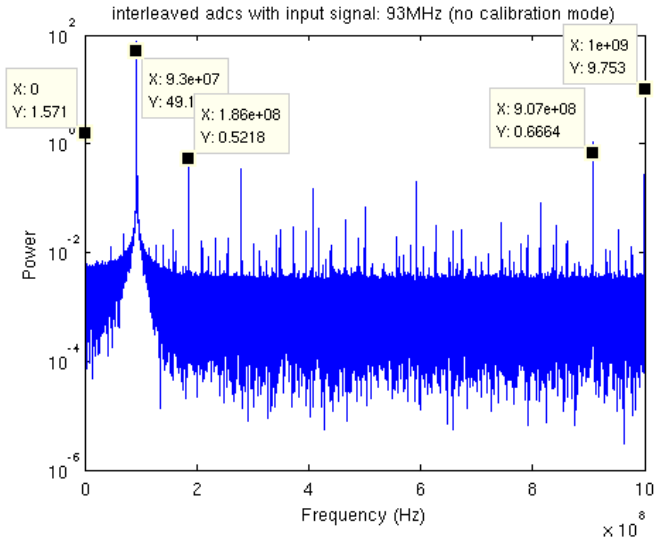


Figure 6-2

9. No Calibration Mode: page 40/41 in data sheet for AT84AD001B.

3. **External Adjustment** (by running "auto_adjust.py"):

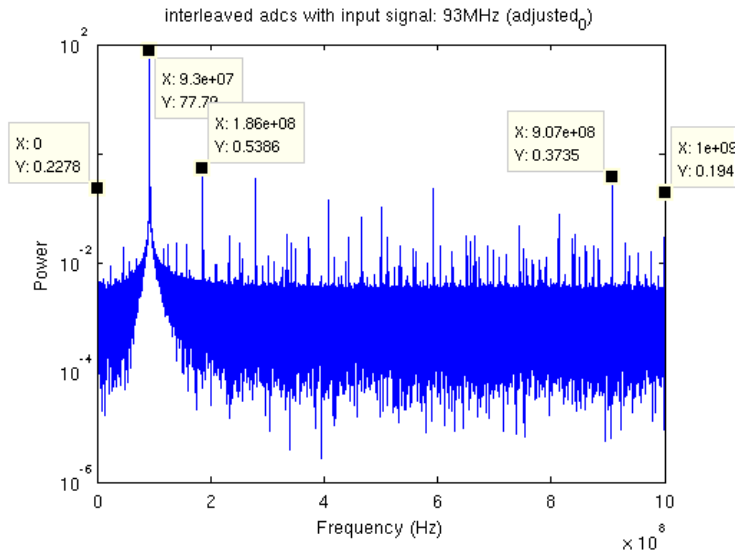


Figure 6-3

4. External Adjustment with different input signals:

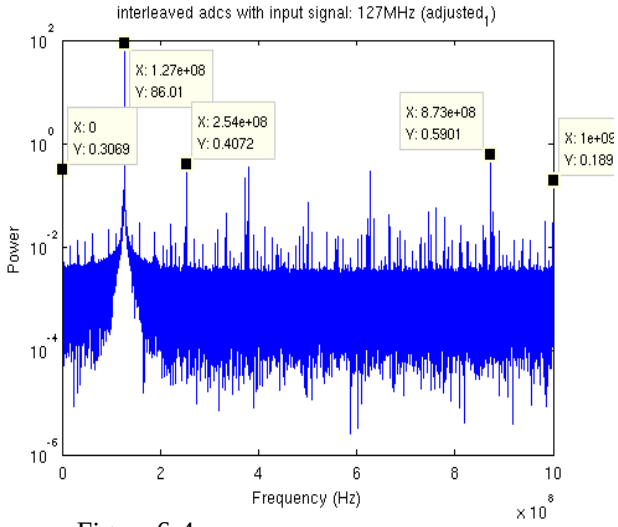


Figure 6-4

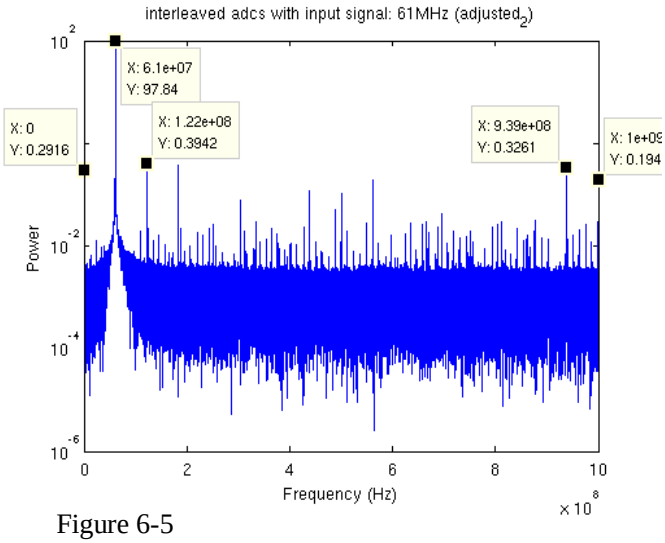


Figure 6-5

Conclusion of Adjustment:

Comparing to the internal calibration, our external adjustment is able to reduce the error by approximately 60%~85%. After the external adjustment, all other spikes will be at most 0.7% of the height of the major spike.

7.Supplement Notes:

1. Making sense of the data sheet:
In table 10-3 in data sheet for AT84AD001B/AT84AD001C, the "x's" don't mean "don't care" as we originally thought, but rather mean "not related to this feature".

For example, the first four rows are grouped together, and it shows DATA0 and DATA1 is related to them.

Since we have to write all 16 bits at a time, and the register is not readable, all we can do is figure out what we want for each setting and write corresponding value to that specific bit.

Table 10-3. 3-wire Serial Interface Data Setting Description

Setting for Address: 000	D15	D14	D13	D12	D11	D10	D9[1]	D8	D7	D6	D5	D4	D3	D2	D1	D0
Full standby mode	X	X	X	X	X	X	0	X	X	X	X	X	X	X	1	1
Standby channel I[2]	X	X	X	X	X	X	0	X	X	X	X	X	X	X	0	1
Standby channel Q[3]	X	X	X	X	X	X	0	X	X	X	X	X	X	X	1	0
No standby mode	X	X	X	X	X	X	0	X	X	X	X	X	X	X	0	0
Binary output mode	X	X	X	X	X	X	0	X	X	X	X	X	X	1	X	X

Figure 7-1

2. Issues with "Analog gain adjustment":
We didn't use this feature to adjust the gain of the iADC, but during our testing we found something may be worthwhile to note.

001	**Analog gain adjustment** Data7 to Data0: gain channel I Data15 to Data8: gain channel Q Code 00000000: –1.5 dB Code 10000000: 0 dB Code 11111111: 1.5 dB Steps: 0.011 dB

Figure 7-2

As the data sheet states, when DATA code goes from 00000000 to 11111111, the gain should keep increasing. But our testing shows different result. (note: the program takes a few minutes to run)

Code: **anag_gain_test.py**

https://casper.berkeley.edu/wiki/External_Adjustment_for_the_Atmel_iADC#Supplement_Notes

Plot:

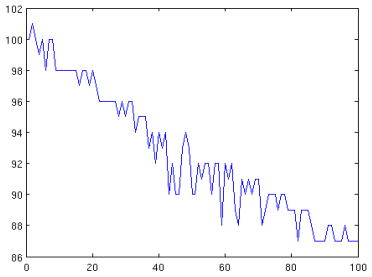


Figure 7-3

Data file:
(click here to download)
1.(format: gain_vi max_of_the_data)
analog_gain_test_result.txt
2.(format: max_of_the_data)
analog_gain_test_plot.txt

3. About gain compensation:

011	**Gain compensation** Data6 to Data0: channel I/Q (Q is matched to I) Code 11111111b: –0.315 dB Code 10000000b: 0 dB Code 0000000b: 0 dB Code 0111111b: 0.315 dB Steps: 0.005 dB Data6: sign bit

Figure 7-4

It's worthwhile to note that different from most other settings, gain compensation uses only 7 bits(DATA0 to DATA6), and the example code 10000000b: 0 dB should be 01000000b.

8. Acknowledgements:

I want to express my appreciation to David George, Mark Wagner, Dan Werthimer, Daniel Blakley and Edward Wishnow. Without their help and suggestions this Memo wouldn't have been possible.