31 July 2003

Restarted software development after long hiatus. To keep track of what I have completed, I have decided to start a debugging log

Software is starting up and beeping out continuity status. No other activity noted, i.e. no launch detect. Therefore, check out the ADC/sensor system. In order to get some data out of the altimeter, I start with the code written to beep out AGL altitude so that I can use it to beep out other numbers. After fixing a few errors, the beeping code is functional.

Accelerometer data does not respond to changes in attitude and the value decays slowly. To make sure that this is actually accel data and not an error in the ADC code, I try to read the battery voltage. This reveals mistake in the ADC code. A loop counter is initialized with a hex value of 0x13 instead of decimal 13. After fixing this, the ADC code works correctly and the value beeped out for the battery voltage is within one count of the computed value. It also works for the pressure sensor.

The slow decay in the accel data remains. Close inspection of the circuit card reveals that one pin on the op-amp is not soldered to the card. Fixing this solves the problem and I am getting good accel data.

UP: 1992 counts
DOWN: 2054 counts
SIDE: 2022 counts

Sensitivity is about 30 counts/G

ADC/sensor system is now functional.

Next: Verify module accel.asm is functioning properly.

1 August 2003

Call to get_acceleration results in continuous series of beeps. No idea why this is happening. Further investigation narrows it down to the a_filter code as being the problem.

This turned out to be an indexing problem with the use of the FSR register and the IRP bit. I ended up addressing an area of data memory that was undefined (FSR = 0xA0 and IRP = 1, address = 0x1A0) why this altered program flow I have no clue.

Get_accel now returns values that almost make sense. If the accel does not change position from power up, a value of zero is beeped out. Change results in a negative number in one direction and zero in the other. Clearly this is not correct.

Get-accel subtracts a delayed average of the acceleration reading before applying the gain. A quick check reveals that these are being initialized with a value that is very different from what is being read later. I cannot find anything in the ADC datasheet to indicate a wake up period. I suspect that this might be caused by the low pass filter on the accel output. Therefore I will add a power up delay before reading the accel and see if that helps.

Nope, that wasn't it. I then hacked the code to beep out the value read for initialization and it is whacked. No idea why at this point.

The problem was a sample time issue. The ADC timing is completely determined by the serial interface timing and that is being bit banged. Although I copied sample code that I thought was OK, the timing proved to be off. I lengthened the delay in the basic serial timing loop and added some extra delay at the sampling time. All is now well. I hope. ☺

Getting back to the get_acceleration routine, I still get a value of zero for one direction.

It looks like subtracting the average value from the current reading is going OK so the next thing to check is the multiplication by the gain factor.

It looks like the internal eedata read routine is getting a value of 0x4a4a but the Warp13 device programmer shows that the correct value has been programmed.

Time to check out the eedata routine.

A mess of errors in this one. First, I hadn't set up my debug code properly so I wasn't hearing the correct value. Second I had a transcription error in the eedata routine having to do with those damn data page bits. Third, the eedata memory wasn't really being programmed because I messed up the config bits. (I should have READ the data back in to verify this previously.)

But even after all of that, I am still getting a value of zero. Shit.

It turns out that the code example in the Microchip 16F628 data sheet is in error. The EEADDR and EEDATA registers are now in data page 1 while the example assumes they are page 0. eedata routine working normally now!

The result from multiplying the gain by the offset adjusted ADC reading now makes perfect sense. The values are in the range of about +/- 9 m/s. Cool.

After putting the code back to normal configuration, I now get a launch detect! Woo Hoo!

It is starting to look like I need to begin working on the code that I have been avoiding: the serial communications routines.

2 Aug.

Checkout of barometric system. First get raw number from ADC, which is 3258 and looks reasonable. Then check the computed altitude which is 1048 meters which is not reasonable. It looks like I need to revise the altitude conversion data in the face of finally having real data from the sensor.

7 Aug.

Finally started messing around with the serial communications code. I first had to cobble up an interface so that the altimeter could talk with a serial port. I used a MAX232CPE that I had on hand

Found a logic error in the txchar code. I was checking for the wrong state. Changing a btfsc instruction to btfss fixed this.

The code is transmitting the initial characters but pukes when it transistions to sending data. Lobotomize the serial EEPROM read routine and I get a correct HEX dump of the same data over and over. Time to check the serial EE routines.

Wait a minute, it seems to be working correctly. Data blocks are sent, resent if no acknowledge, and address advances (and data changes slightly) when ack character sent. But I have no idea if the data is any good or not.

Ok, data is altitude state, velocity state, acceleration state, and pressure.

S 0000 ACDE 0012 0069 1FFF 23
S 0001 ACDC 0017 0043 1FFF 01
S 0002 ACDC 001B 002A 1FFF ED
S 0003 ACDC 001D 0019 1FFF DF
S 0004 ACDD 001E 000E 1FFF D7
S 0005 ACDE 001F 0006 1FFF D2

Altitude is much too high, velocity too high for resting, same with acceleration, and the pressure reading is off. Time to check and see if the data is being written and read correctly. First step is to try and write a fixed pattern of data and see if it comes back.

That answers that question. The fixed pattern was not read back. So which part of the code is broke? Reading? Writing? Both?

8 Aug.

I found a bug in the Hyperterminal program. I was receiving data fine but the altimeter was not responding to serial input. I checked and no serial data was coming over the port. It turns out that I had mistakenly left hardware flow control on. Changing it had no effect until I shut down the program and restarted, making sure to get it right the first time.

I modified the serial eeprom code to treat the data line as open drain. I did this by setting the data out to a value of zero. Then I just toggle the TRISB register to either drive the pin low or let it float high (thanks to the pull-up resistor).

Didn't help the operation at all. It looks like I need to sit down and go through the code line by line. Yuck.

While looking at the code I can't find where the clock line is set to an output. This could be serious trouble.

Success! Sort of. My end of preflight data marker is located in the first four words of the data that I read back. This means that at least some of the data read and write code is working!

The routine to write the pre-launch information is testing the wrong bits of the timer. I do this to only record data at 32 samples/second instead of 128.

Result!

Prelaunch data is recorded and there is a launch detect marker! OK, now fix the prelaunch code to get rid of the fixed data and go back to the normal data.

First post launch detect data is:

S0100 0420 000F 0005 1CB5 0A
S0108 0421 000F 0005 1CB5 13
S0110 0422 000F 0005 1CB5 1C
S0118 0423 0010 0005 1CB5 26

Which looks quite reasonable except for the initial altitude except that it is consistent with the un-calibrated pressure to altitude conversion.

It looks like the velocity part of the launch detect is working since the last sample before DEADBEEF is 14 m/s and the first after is 15 m/s. The acceleration test is not working because the acceleration is 6m/s/s. Ahh, there is a goto instruction present that bypasses the acceleration check.

The data shows that the code also transitions to state 2 at apogee but it skips state 3 and goes straight to state 4 after that. Nope, that is correct. I have a state for "POST_APOGEE" but it is not used. The code heads straight for state "DROGUE" like it should.

There are also transitions to states 5 (MAIN) and 6 (POST_MAIN) and that is where it ends because I haven't written the code for detecting landing yet!

Woo Hoo!

I can also see that I haven't coded the change of Kalman filter gains at apogee yet.

Details, Details.

Things to do before flight test.

| | | |
|---|---|---|
| 1 | Update altitude conversion table | X |
| 2 | Landing detection code. | X |
| 3 | Altitude beeping code | X |
| 4 | Measure sensor statistics and update Kalman gains | X |
| 5 | Calibrate accelerometer sensitivity | X |
| 6 | Install standoffs and mounting hardware | |
| 7 | Test pyro outputs | X |
| 8 | Kalman gain change at apogee code | X |
| 9 | Write code to download flight data and process to GNUPlot compatible file. | X |
| | | |

While mucking about with the gain change code, I noticed that the software is not properly updating the averages of the at rest acceleration. This needs more work.

Accleration data:

Average = 2054.712 counts
Std dev = .751541 counts

Average2 = 1993.034 counts

2G delta = 2054.712 – 1993.034 = 61.68 counts

calibration factor = 0.317769 m/s/s / count ( 1.0425 ft/sec/sec per count)

noise = .2388 m/s/s

Output of kgain32

```
Input noise values used (standard deviation):
#Altitude       -       1.000000 feet
#Acceleration   -       0.250000 feet/sec/sec
#Model noise    -       0.025000 feet/sec/sec
#Sample rate = 128
#Estimated output first order statistics (standard deviation):
#Altitude     -       0.074850 feet
#Velocity     -       0.037176 feet/sec
#Acceleration -       0.077098 feet/sec/sec
#Kalman gains converged after 990 iterations.
#       0.005603        0.000521        0.001958        0.007058        0.000034
        0.095109
#0.16 bit fixed point gains (fraction only) in hex
```

```
K1_HI = 1
K1_LO = 6f
K2_HI = 0
K2_LO = 80
K3_HI = 0
K3_LO = 2
K4_HI = 0
K4_LO = 22
K5_HI = 1
K5_LO = ce
K6_HI = 18
K6_LO = 59
```

Output of kgain31

```
D:\usr\local\Rockets\AltimeterProject>kgain31 128 1 1000
Input noise values used (standard deviation):
#Altitude       -       1.000000
#Model noise    -       0.031623
#
#Kalman gains converged after 105 iterations.
#
K1 =    0.0540431133
K2 =    0.1900927756
K3 =    0.2961767648
16 bit fixed point gains (fraction only) in hex
K1_HI = d
K1_LO = d5
K2_HI = 30
K2_LO = a9
K3_HI = 4b
K3_LO = d2
```

12 Aug.

Working on various items including gain change and landing detect.

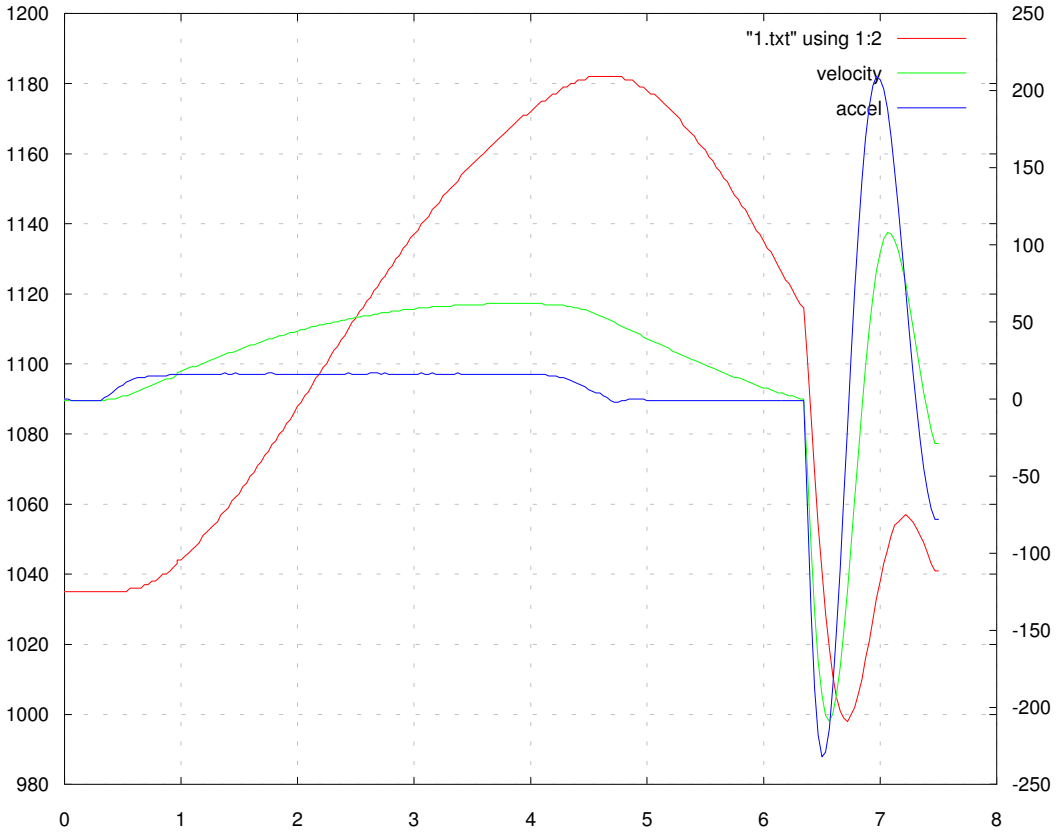
The gain change code is working. Mostly. It looks like after the gains change, the acceleration state never changes. This might happen if the delta between measured and estimated altitude was very small and the gain was small as well. The delta in this case will be large and the gain is reasonable. Perhaps my code to ignore the pressure reading at high altitudes and velocities is crap. ☹

No, it was disabled by a return instruction. While looking at this code I realize that I need to do something for high altitude flights. Right now the Kalman code is ignoring the pressure data if the altitude is too high. This will result in the Kalman state doing weird things. Not sure what to do about this but it can wait.

I have also noticed that I do not always get a start of flight marker in the pre-flight ring buffer. This looks to be a timing issue. The first thing I do after the Kalman state update is write the data to EEPROM. After determining that it is time to advance the state, I write the marker. Because I skip writing data 3 out of 4 times, I have a 25% chance of starting another write while the EEPROM is busy with the previous write. I just need to rearrange the order of things a bit.

Now that the pressure data is being flagged as valid I am getting other bits of weird behavior. Checking further I find that I am checking the least significant (fractional) byte of the velocity in my Mach code instead of the most significant. Doh!

OK. This looks much better.




It is still a little off because the pressure reading is constant. But that is hard to account for without an actual flight.

One of the things that I am curious about is how much time is left after completing the Kalman filter code. So I mangled up a special version of the code that recorded the value in Timer 1 to the serial EEPROM instead of the pressure and state data.

There is a bit of jitter in the data. Mainly because the multiplication algorithm takes differing amounts of time to complete depending on the data values. But none were over 0xE300. The counter goes up and generates an interrupt when it overflows. So the number of timer ticks left is 0x10000 – 0xE300 or 7424 decimal.

This is a shockingly high number given that I load this counter with 7812 ticks to interrupt. That means that only 388 timer ticks are required to complete. I am certain that I do not believe that. The timer is running at 1 tick per instruction so that is only 388 instructions executed.

The problem was in the code to read a free running timer. The code error should have resulted in random errors but Murphy stepped in.

Now I am reading a value in the range of 0xF240 to 0xF260. using 0xF300 results in 3328 timer ticks left. So I am using about 42% of the available time on a 4MHz PIC processor. Not too bad.

13 Aug.

Pressure to altitude conversion time. I noticed that the altimeter is showing an altitude of about 1000 meters when my actual altitude is around 600 feet. Not a terrible problem but something that isn't too hard to fix.

Pressure sensor: sensitivity is stated as 45.9mV/kPa at 5.1 Volts excitation. Scaling this to a 5V excitation gives 45mV/kPa sensitivity. Because the ADC uses the power supply for its reference, I don't care what the exact voltage level is. If it goes up or down the ADC range will track it and the net result is same as having a perfect 5V supply. Ratiometric conversions cover up a lot of things that would otherwise cause trouble.

Static reading was 3258 counts with a station pressure 99.1kPa. I need to know what pressure would result in zero counts.

$Y = mx + b$

I know m (the sensitivity) and have a known x and y. The rest is just algebra.

$B = y - mx$ = 99.1kPa – (3258 counts * 5000mV/4096 counts)/45mV/kPa

B = 10.721 kPa

Output of Program pdata.c

```
Altitude, slope
15443.899319  -11.273465
12557.892173 -8.097953
10484.816148 -6.415236
8842.515676 -5.356959
7471.134202 -4.623736
6287.457812 -4.082775
5242.267358 -3.665604
4303.872816 -3.333130
3450.591485 -3.061326
2666.892149 -2.834563
1941.243986 -2.642219
1264.835897 -2.476806
630.773592 -2.332887
33.554554 -2.206414
-531.287541 -2.094310
-1067.430875 -1.994188


Data suitable for MPASM
        ;
        ;       Pressure to altitude conversion tables
        ;
        ;       Base altitude is a 16 bit integer
        ;
        ;       Slopes are 8.8 signed fixed point numbers
        ;
        ;       Base Pressure, high byte
        ;
        de      0x3c, 0x31, 0x28, 0x22
        de      0x1d, 0x18, 0x14, 0x10
        de      0x0d, 0x0a, 0x07, 0x04
        de      0x02, 0000, 0xfd, 0xfb
        ;
        ;       Base Pressure, low byte
        ;
        de      0x53, 0x0d, 0xf4, 0x8a
        de      0x2f, 0x8f, 0x7a, 0xcf
        de      0x7a, 0x6a, 0x95, 0xf0
        de      0x76, 0x21, 0xed, 0xd5
        ;
        ;       Slope, high byte
        ;
        de      0xf4, 0xf7, 0xf9, 0xfa
```

```
        de      0xfb, 0xfb, 0xfc, 0xfc
        de      0xfc, 0xfd, 0xfd, 0xfd
        de      0xfd, 0xfd, 0xfd, 0xfe
        ;
        ;       Slope, low byte
        ;
        de      0xf4, 0xf8, 0xfa, 0xfb
        de      0xfb, 0xfc, 0xfd, 0xfd
        de      0xfc, 0xfe, 0xfe, 0xfe
        de      0xfd, 0xfe, 0xfe, 0xff
```

The list of things to do is getting short!

I just checked the pyro outputs using LED's. The apogee LED did not come on but the main did. A quick check of the code reveals that I was toggling the wrong bit. An easy fix and now it works great.

Note that an output capacitor will have to be added to the igniter board before it can be used to fire E-matches.

20 Oct. 2003

DARS was finally able to schedule a launch and not be rained out. Yeah! The first flight test for the altimeter was in my trusty Aerotech Initiator that I added a payload section to. The motor of choice was the G64-7. Visually the flight was only marred by the motor ejection being a bit early. This was annoying because I really wanted it to be late so the altimeter code could do its apogee detection thing.

Tonight I downloaded the data and processed it. Yuck. Something bad happened well before apogee.

Everything looks to be fine through the motor burn but at 4.6 seconds into the data, something very bad happens and the acceleration plot goes crazy. While I only recorded the raw pressure data, I am still pretty sure that the accelerometer did not go crazy and do this. The cause is most likely an overflow or some such during the computations. This will be hard to check.

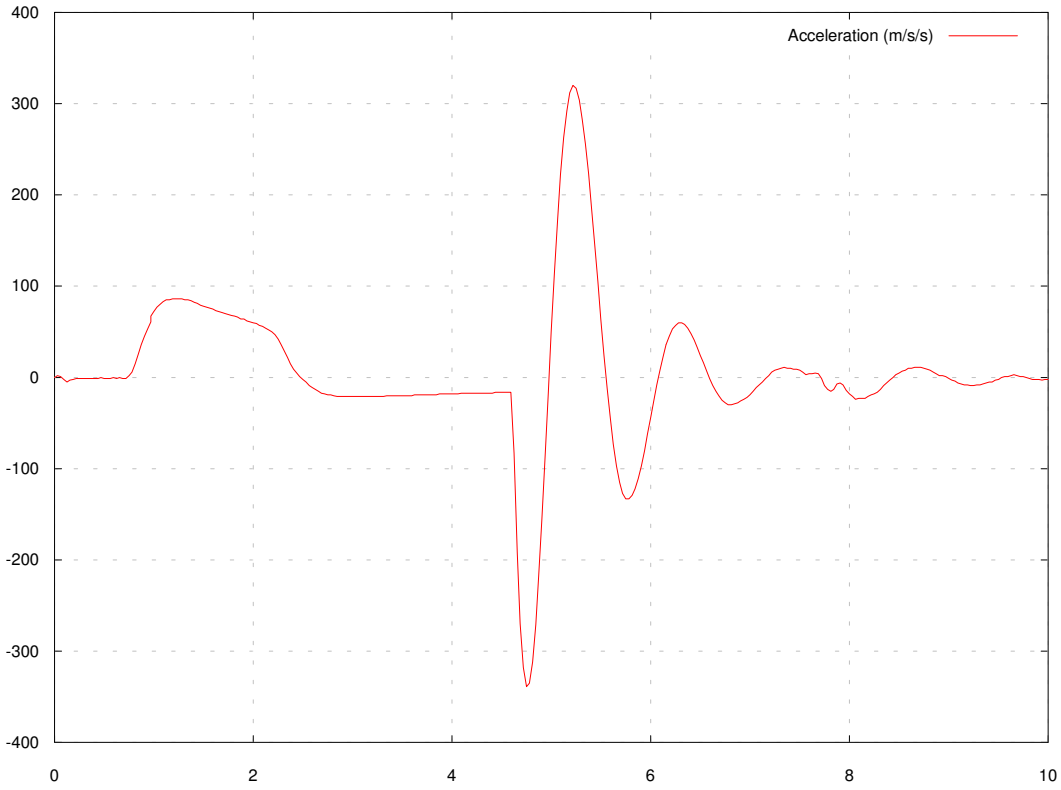



**Figure 1, Acceleration plot**

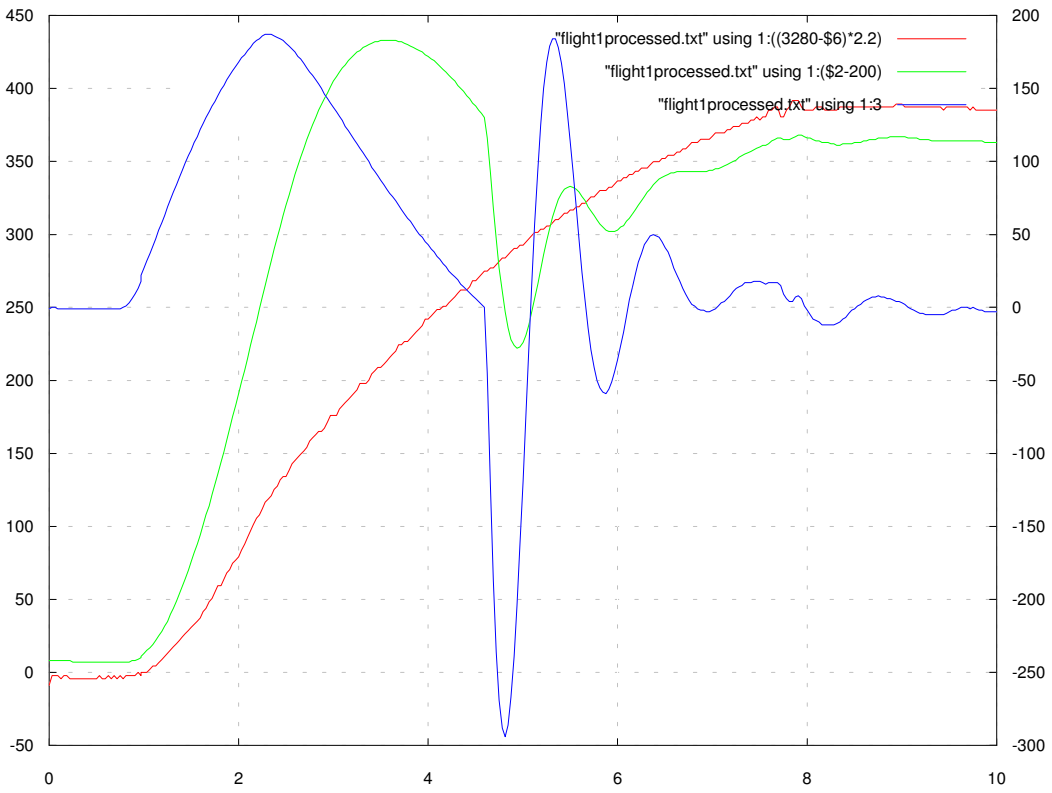



**Figure 2**

I generated another plot that reveals that I only have one problem. Good. Looking at Figure 2, I can see that the reason for the break in performance and wild oscillations occurs when the software detects apogee. This results in a change of Kalman gains (zeroing out the acceleration gains) so that only the pressure measurement is used. This results in a large error between the filtered altitude and the measured altitude causing a large correction. This is the cause of the wild oscillations.

But I still have the problem that the initial response (at least it detected liftoff correctly) is wrong. The altitude increases much too quickly and then peaks too soon. This might be caused by whacked Kalman gains so I will double check those.

The Kalman gains looked OK so I started checking some other things. I found a problem in the state update code. This was set up for 64 samples per second and when I switched to 128 (sometime long long ago) I forgot to change the right shift counts here. This causes nasty things to happen because the time between samples is 1/128 second but the dynamic equations are being stepped by 1/64 second. Yuck. This is easily and quickly fixed.

I am going to also change the data storage code. I was storing 8 bytes 16 times a second. I am going to change this to 16 bytes and do it much more quickly. At apogee, the time between stored samples will drop back to 1 per second. In addition I am going to tweak it so that a sample will be stored every time a flight event is detected. This will be a bit complicated but should be well worth the effort.

4 Nov. 2003

Very bad news. While working on the new data storage and processing code, I ran into a few problems. The really nasty one has to do with time.

As part of the data stored I include the least significant byte of the time counter that gets incremented with every timer interrupt. This is used by the processing software to keep track of elapsed time since the time between recorded samples is not uniform.

But it turns out to be less uniform than desired. I was getting alternating (in not quite a pattern) time increments of one or two ticks. This is really bad because it could mean that the code is not executing as fast as I thought it is. The simple way to check this is to change the timer interrupt routine so that the Kalman code only executes every other interrupt.

I did this and the recorded time information has become quite uniform. This is really bad news because it means that the code is not executing as fast as I thought.

The next step is put back the code that reads the timer at the end of processing the data (actually, I do it in the data storage code.) Then I can check to see what the execution time is. I will do this with the skip every other interrupt code in place.

Change of plan. I realized that I added another ADC conversion in the middle of the data storage code. I will disable that and see what happens first.

Nope that didn't help.

Ok, I added the timer code to see how much time is left. Yuck.

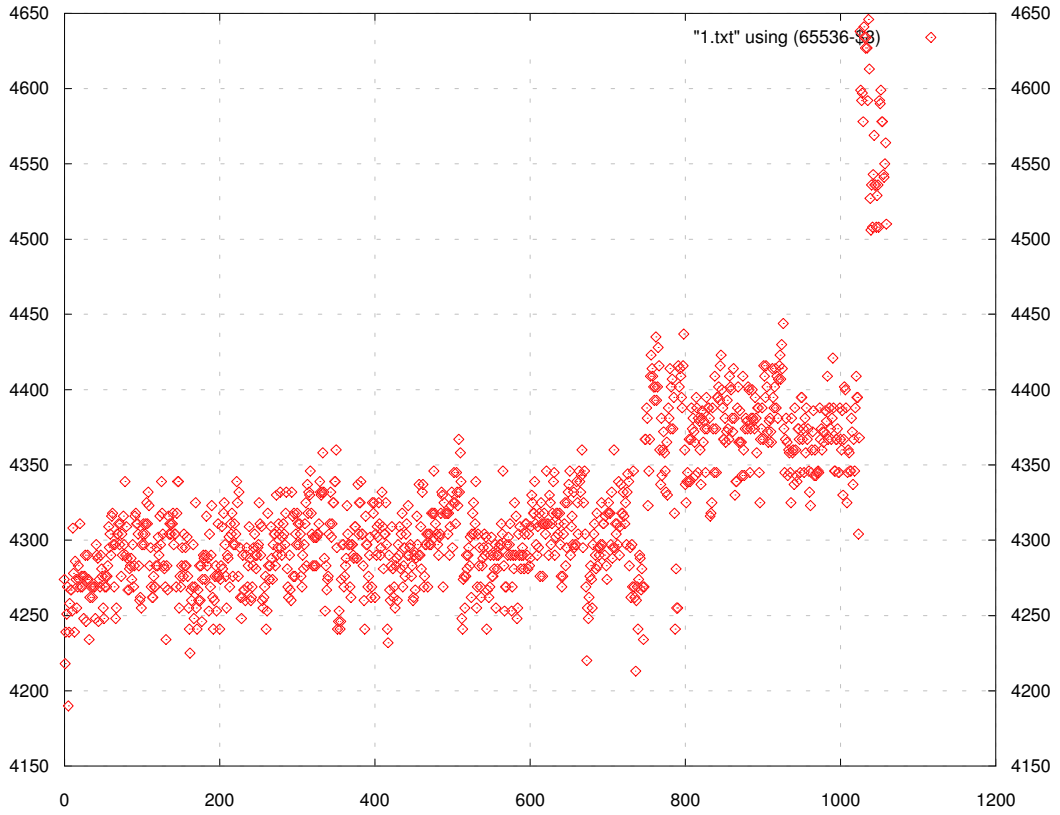



**Figure 3, Time left**

This is an absolute disaster. I started the timer with 15624 ticks until interrupt and when I finish processing there are ~4300 ticks remaining. This means that I cannot run at 128 samples per second and must run at 64.

The filter will still perform well but I was really hoping to get 128 SPS out of the hardware.

Shit! (and the rest of the seven words you can never say on television)

OK, mangle the code to 64 SPS (timer interrupt is at 64 already) and re-compute the Kalman gains for 64 SPS. It is already far too late this evening and I am feeling pretty stupid so this is a project for later.

5 Nov. 2003

I will skip the gory details. The bugs (known anyway) are fixed and the new data storage code is in place.

I altered the data storage code to give me 2 seconds of preflight data. This is recorded at 64 SPS as is the next 14 seconds of data. After that the rate decreases to 2 SPS until the end of flight. Data capacity is now 16 seconds at 64 SPS and then 512 seconds at 2 SPS.

I also realized that with the flight state information recorded I did not need to record a special marker for start and end of flight. The flight state change marked those events just fine.

Here is a plot of  data from a bench test.

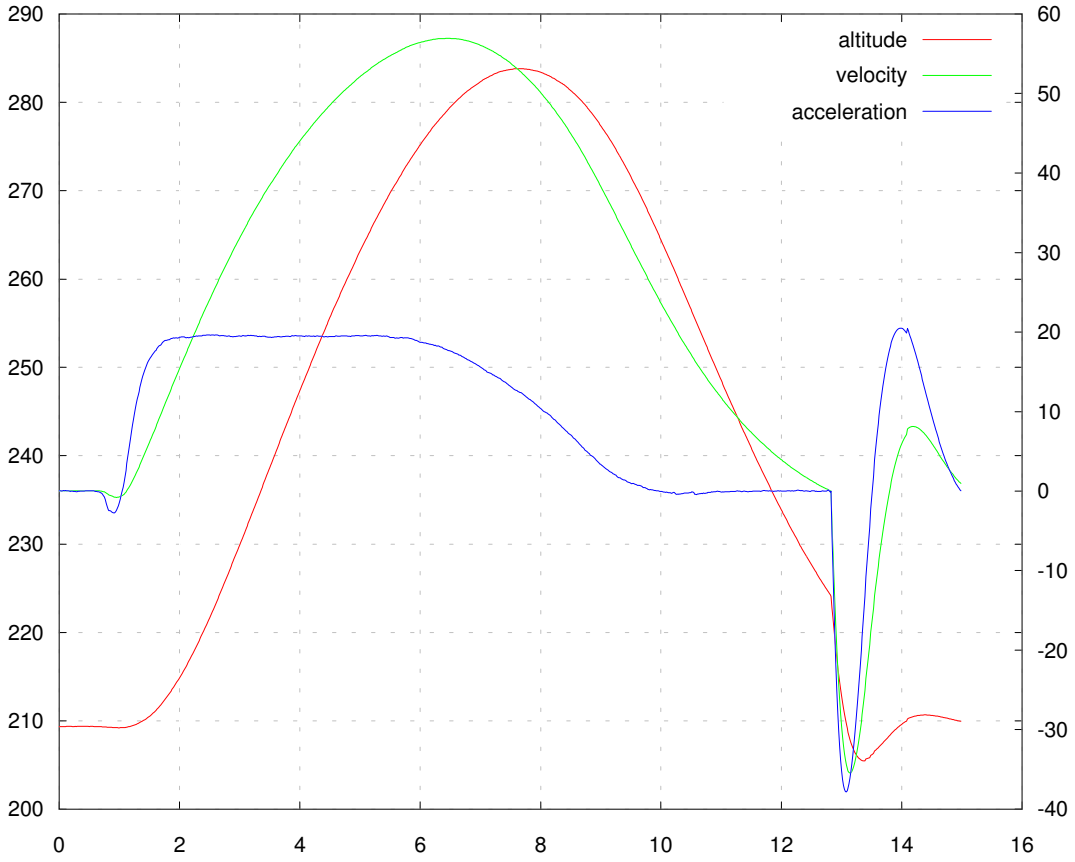



**Figure 4, Bench Test Data**

Now that the recorded data include 8 fractional bits, resolution is greatly improved. It is nice to see good Kalman filtered data coming straight from the altimeter. No post flight processing required!

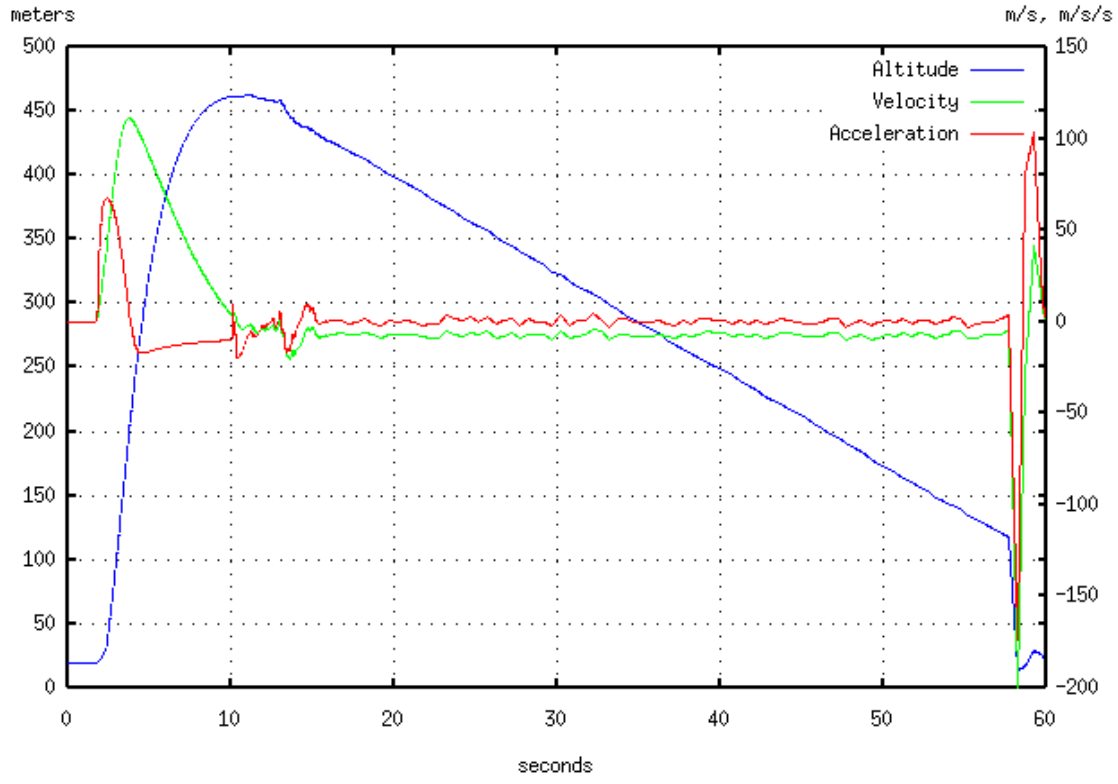
I wondered about the dip in the acceleration curve at about 1 second for a while. Then I realized that this was the result of my picking up the altimeter to turn it over.

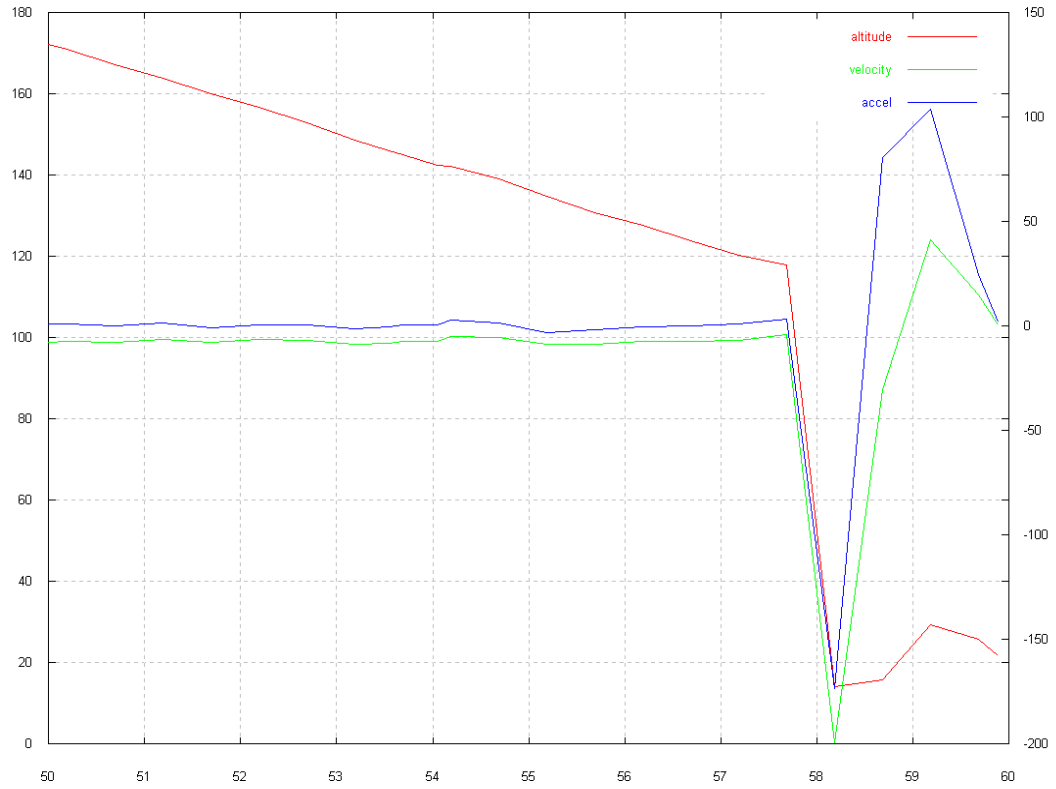
One thing I might change is the criteria for declaring liftoff. It took about one second at two G's before the required velocity was achieved in this test. That feels too long for some reason. Or I may just leave it alone.

21 Dec. 2003

Another flight test done. The flight went well but the altitude beeped out afterwards was incorrect. This problem was quickly found and was apparently the result of using a movf instruction instead of the needed movwf instruction.

The downloaded flight data looks pretty good except towards the end when something weird happens.






Taking a closer look at the data after 50 seconds, it looks like the altitude suddenly changed to what it was reading at ground level. But there is not a similar change in the pressure measurement. Possible causes?

| | |
|---|---|
| Corrupted data | Previous data recorded at base altitudes higher than this location. Therefore no prior data recorded at this altitude. Measured pressure and acceleration looks fine. Of course, 31 of 32 samples are not recorded. |
| Processor reset | Program would have started recording pre-launch data again. Which is located in the first 2 seconds. No evidence that this occurred. |
| Program stopped running for a while. | Timing data included with recorded data indicates no problems. |

| | |
|---|---|
| Corrupted data sample. | A single corrupted sample would not cause this sort of behavior. (Compare the acceleration spike measured at deployment to the filtered data.) The filter seems to be converging to the ground level altitude. |
| | |
| | |

Things to do/look at:

Because the ADC will repeat the data in reverse order if you continue to clock it after getting the lsb of the result, it is possible to perform a simple check to make sure there is not a data transfer error. It isn't a perfect check but it is better than nothing.

It is possible that this is caused by some internal program flaw that only appears randomly or in a data dependant fashion. The only real way to check this is to fly the altimeter or to simulate flights somehow. The simulation capabilities of MPLAB have improved a lot from the last version and it is now possible to feed realistic pressure/acceleration measurements to the program during simulation. In order to do this I need to have a source of realistic flight data. I will need to drag out an old copy of the RASP source code and hack it so that it outputs appropriate data suitable for MPLAB. This will be extremely slow. The alternative is to build hardware which will output simulated sensor data for input to the altimeter's ADC. Oh, and build a new sensor board without sensors.

The program currently checks to see when the rocket lands and then stops recording data. I could alter this so that data is recorded until EEPROM space runs out. Then I would at least be able to see what is going on for a bit longer. And since I don't really need 512(+16) seconds of data, I could increase the stored sample rate. By going to 16SPS I would get 64(+16) or 80 seconds of flight data.

1 Jan. 2004

It suddenly occurred to me to look at the altitude conversion routine. I remember going over my data tables pretty carefully to make sure that I didn't have any discontinuities but something might have crept in. Plotting the data shows that right where the trouble begins at 58 seconds, the pressure measurement is passing through 3328 counts which is one of the points in my piecewise linear table. I also noticed a distinct slope change in the filtered altitude at the beginning of the flight at this pressure reading. Nothing so

dramatic as at 58 seconds but then in the early part of the flight the filter is pretty dominated by the acceleration measurement. So now I have to check my altitude conversion to make sure that nothing weird is going on.

Man, this was easy to spot. Now.

I looked at the data table and noticed an anomaly in the slopes. The fractional portions all looked a lot like the integer portions and were all very nearly the same value. They should have looked much more like random numbers. So I looked into the pdata.c code that generates these tables and low and behold there is a glaring error in my code to extract the fractional part of the slope and convert it to a hex value. This should be very easy to fix.

Which is good.

I fixed the problem with the pdata.c program and generated new slopes:

```
;       Slope, high byte
;
de      0xf4, 0xf7, 0xf9, 0xfa
de      0xfb, 0xfb, 0xfc, 0xfc
de      0xfc, 0xfd, 0xfd, 0xfd
de      0xfd, 0xfd, 0xfd, 0xfe
;
;       Slope, low byte
;
de      0xba, 0xe7, 0x96, 0xa5
de      0x61, 0xeb, 0x56, 0xab
de      0xf1, 0x2b, 0x5c, 0x86
de      0xab, 0xcc, 0xe8, 0x02
```

Comparing this to the values earlier in this document shows quite a change in the low bytes.

4 May 2004

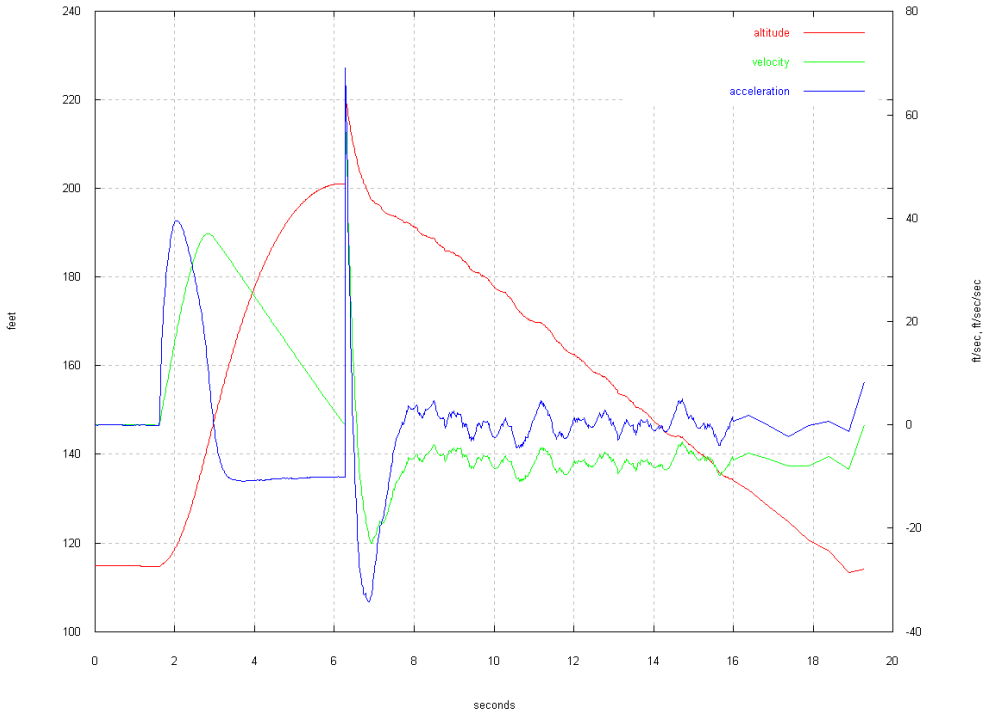
DARS is into one of their long runs of rainouts so I have not had a chance for another flight test. I assembled an F39 motor before the scheduled January launch and still haven't had a chance to use it. The payload bay has been modified so that I can use the altimeter for apogee ejection.

There is a good chance that I can hitch a ride on a very high performance rocket in June. Project Aurora will be flying on a P motor again to 30,000' or so.

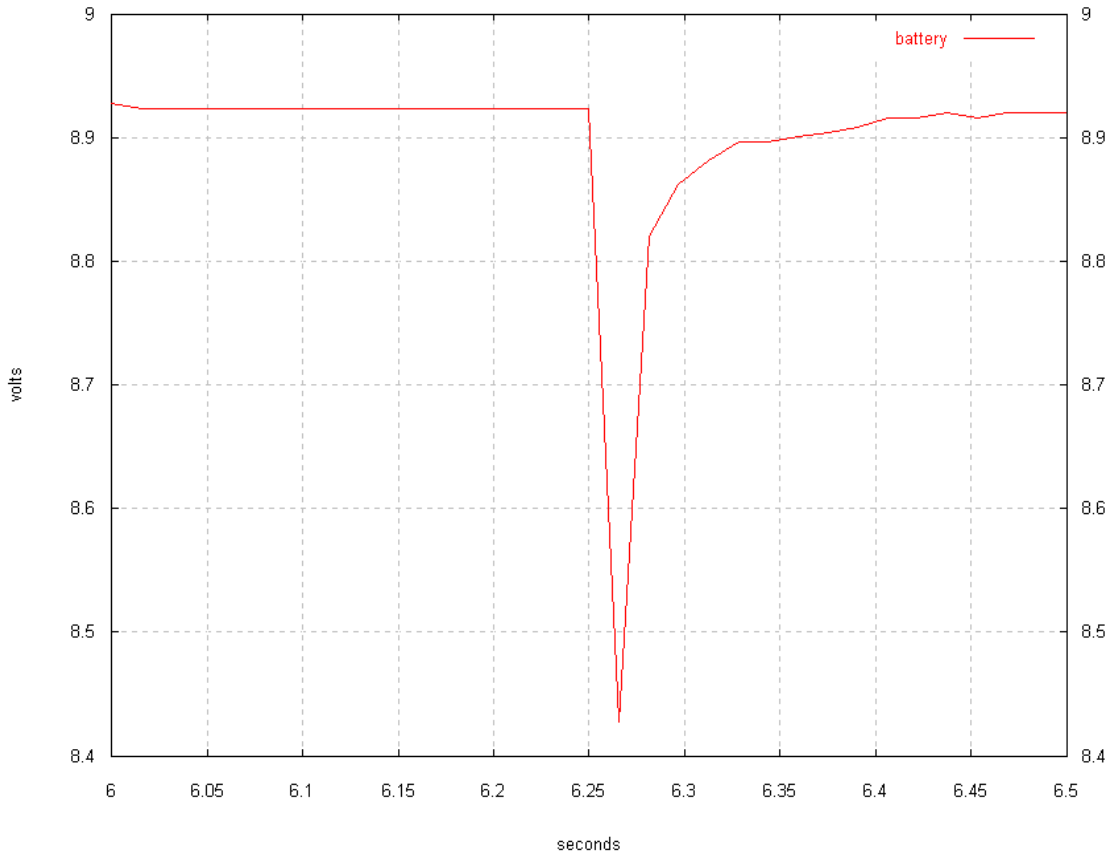
8 May 2004

Finally a launch that isn't rained out.

The flight on a F39 looked perfect with deployment right at apogee. The altimeter was beeping out an altitude of 87 meters when I picked it up.



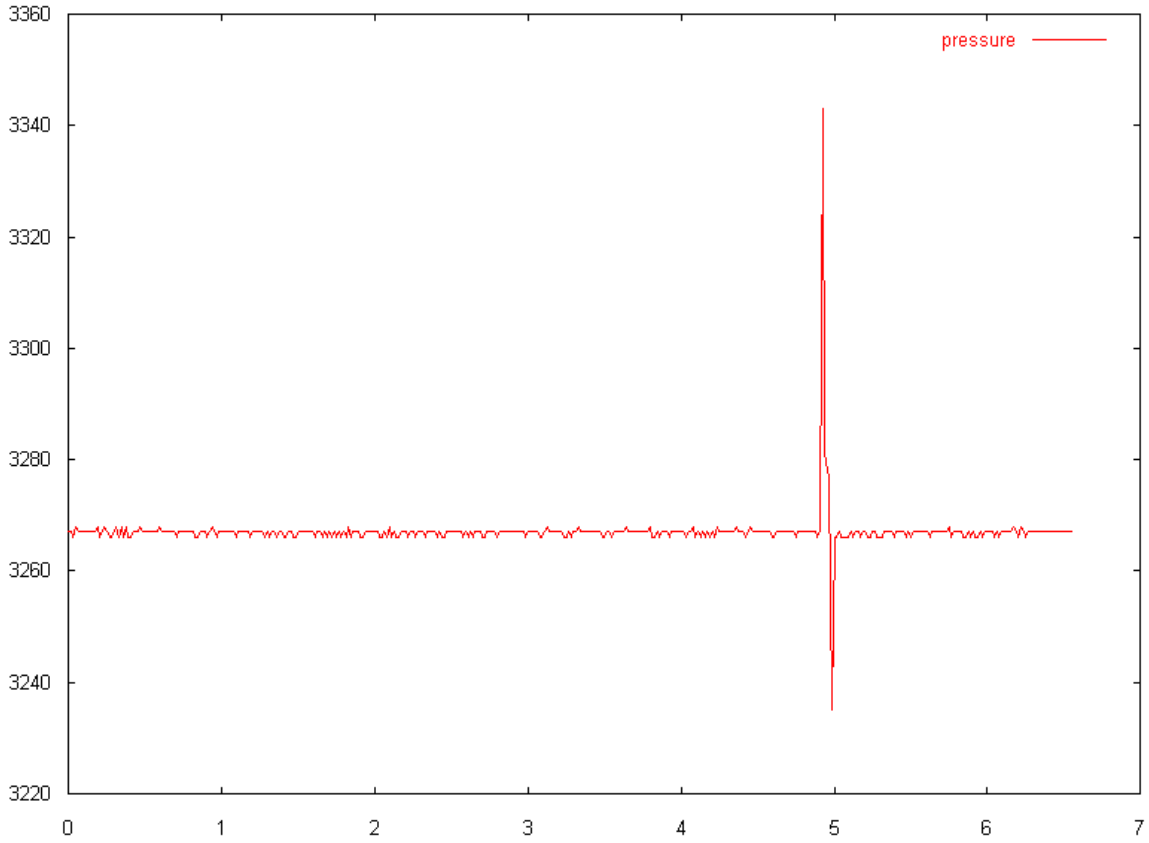

The plotted data looks good with the notable exception of a large anomaly at apogee. A quick check of the raw sensor data shows that both acceleration and pressure took large hits at this time. Because the electric match was being driven directly from the battery (with a 2 ohm resistor in series to limit current) I suspected that there was a power problem. Since I was also recording battery voltage on this flight, I looked at it next.




The battery voltage as measured took a substantial dip when the output FET was turned on. This resulted in the data from the sensors being briefly corrupted.

I decided to perform a few tests to try and isolate the precise cause of the corrupt data. I just shorted the pyro output which meant that the only load was the two ohm resistor. Therefore the current draw will be larger than if an electric match were attached. The first test connected a second 9 volt battery for the pyro outputs. This resulted (file btest1.txt) in no spikes in the sensor data. To make sure that my test setup is OK, I then returned to a single battery to try and replicate the problem.

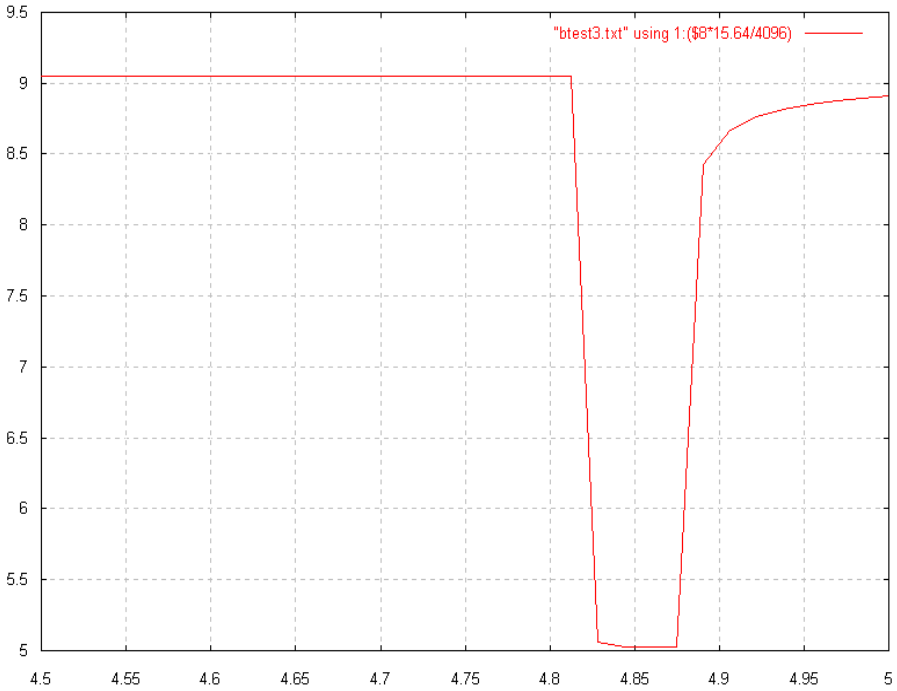
I disconnected the second battery and tied the battery 2 positive terminal on the harness board to the battery 1 positive terminal. I left the battery 2 negative terminal unconnected. This produced a very definite spike in the pressure data. (Btest2.txt)




So for the next test I tried connecting the battery 2 negative terminal to the battery 1 negative terminal. This connects the analog and digital ground nets together on the harness board. This resulted in no change to the voltage spikes. (Btest3.txt) A check of the battery voltage shows a deep drop in voltage.

The battery voltage dropped to almost 5 volts. Keep in mind that it might have gone lower as the output of the 5 volt regulator is used as the reference for the ADC. If the 5 volt rail dropped out of regulation then this voltage measurement is corrupted. I am surprised that the micro controller continued to operate with this deep a voltage drop.

The result is that this altimeter cannot use a single battery to both run the altimeter and to fire the charges directly. I will change the harness board so that it uses a charged capacitor to fire the outputs in the future.




I repeated the test using a total of 4 ohms for the output load. Strangely, the measured battery voltage dropped to the same value: just above 5 volts. This got me to thinking and I now think I know what is happening.

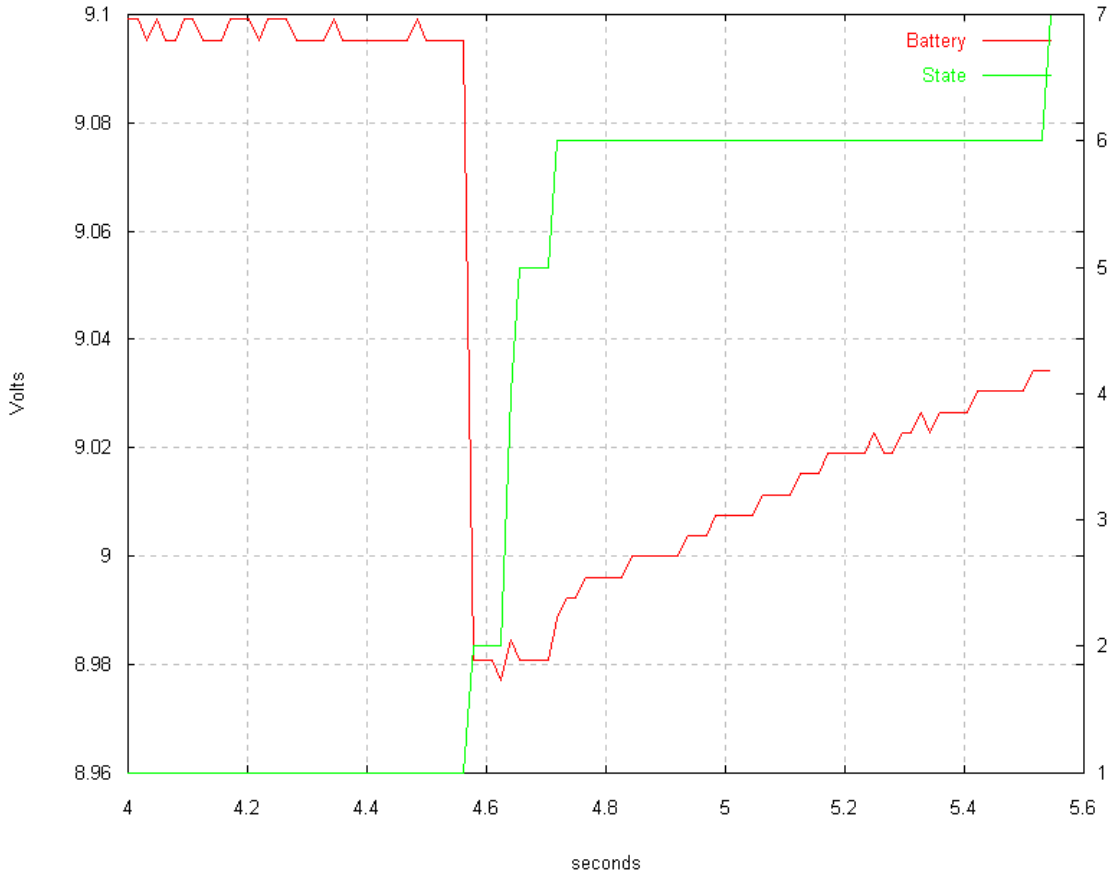
The dropout voltage for the 5V regulator is around 100mV at 50mA load current and 10mV at 100uA load. So the measured battery voltage is just too close to the dropout voltage for the regulator for it to be a coincidence. So what is happening is that the battery drops to where the regulator is unable to maintain a 5V output and begins dropping. But the ADC is using that value as the reference for conversions so the measured battery voltage now stays fixed at the dropout voltage.

The 16F628 is specified to operate to 3V and the ADC to 2.7 so they are fine even after the regulator drops below 5 volts. So why doesn't the voltage collapse below 3V? Good question. The answer is that the ability of the IRF7101 output device to sink current depends on the gate to source voltage. As soon as the gate voltage starts dropping when the regulator drops below 5V, the current the FET can sink starts dropping as well. Eventually this self regulates at some particular battery voltage. Which just happens to be above 3V.

The measurement anomalies result from the ADC using the supply voltage rail as the reference. The pressure sensor has a simple RC low pass filter on its output. While the pressure sensor is to some extent ratiometric (the output is proportional to the supply voltage), the capacitor voltage will not track this instantly. Thus the capacitor voltage

will be higher than it should be if it were as ratiometric as the sensor. Because it is higher than it ought to be, the ADC measures it as being too high and the result is a spike in the data. In the bench test data where the voltage dropped for the full tenth of a second or so that the output is on, there was also a downward spike in the pressure data when the supply voltage recovered to its nominal 5 volts. This didn't occur in the flight data because the dropout lasted for such a brief time (one sample) since the electric match bridge wire opened up quite quickly.

I have rewired the altimeter with a 1K resistor and 1000uF capacitor to provide the pyro power. The capacitor will have (1000uF * 5V^2)/2 = 12.5 milli-joules of energy when the battery is at 5 volts. The stated all-fire energy for an E-match (from the Daveyfire-Works web site) is 3mJ/ohm. Using a value of 2 ohms for the igniter means that it needs about 6mJ to make sure that it fires. So I have plenty of margin. A quick bench test reveals much.

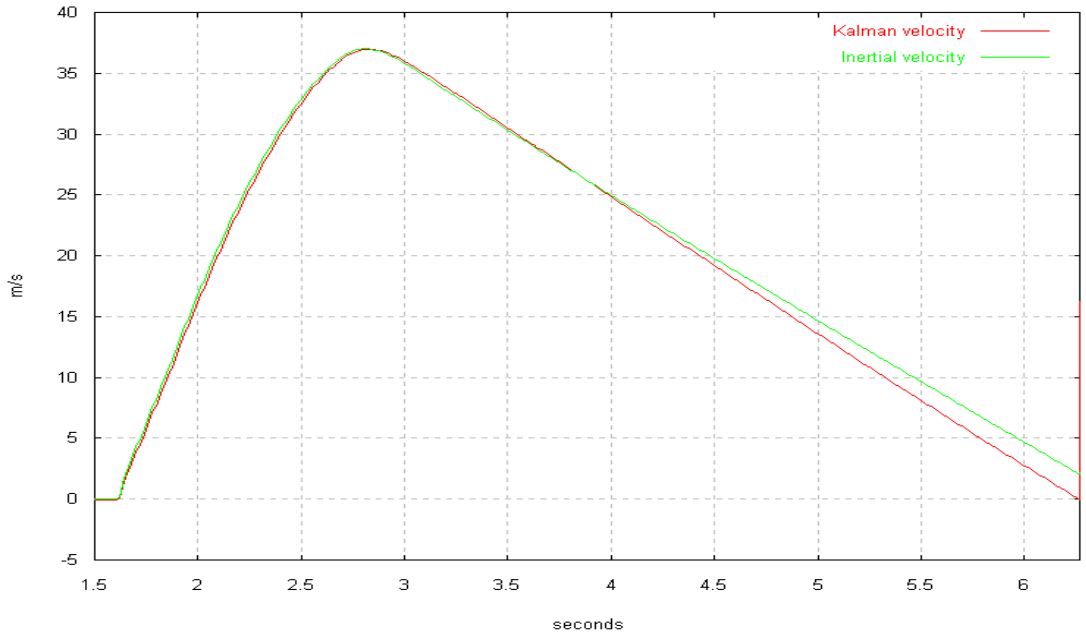



The battery voltage dips about 0.03 volts when the output is turned on. A little bit of quick math reveals that this lithium battery has an internal resistance of about 3.3 ohms. This would result in an output voltage of 9* 4/(3.3+4)= 4.93 volts when the output was turned on into a 4 ohm load and 3.4 volts into a 2 ohm load. This would definitely cause the voltage regulator to drop out of regulation and produce sensor glitches.

# Comparison to inertial velocity

To get an idea of how the Kalman filter compared to a strictly inertial algorithm, I imported the flight data into a spreadsheet to play with. The first step was to determine the 1G offset. Because I did not record the value reached by the altimeter, I averaged all of the pre-launch acceleration values to get 1992.952. I then entered a few formulas to convert the ADC values to acceleration in m/s/s and to then integrate these to get velocity. The result was that at the time of deployment, the integrated velocity was 2.02 meters/second. This would have resulted in the deployment being 2.02/9.8 = 0.2 seconds late. Which doesn't seem too bad except that this was a pretty short flight. (First motion at 1.625 seconds and apogee at 6.265 seconds for a 4.6 second flight.) The error was increasing with time and on a typical 15 to 20 second high power flight, the error would have been much greater.

Comparing the Kalman and inertial velocities shows that they track very closely until peak velocity. After that the begin to diverge. If the problem were due to a non-vertical flight after motor burnout, the inertial value would have been less than the Kalman value but this is not the case. What is probably happening is that both the Kalman and inertial velocities end up being too high at motor burnout. The Kalman velocity is then corrected by the pressure altitude.




To make sure that my approximated 1G offset was not a significant source of error, I also used a value of 1992. This resulted in a velocity at apogee of 0.12 m/s. It is highly unlikely that the 1G offset was anywhere near this low so the offset was not a problem.

Things to do:

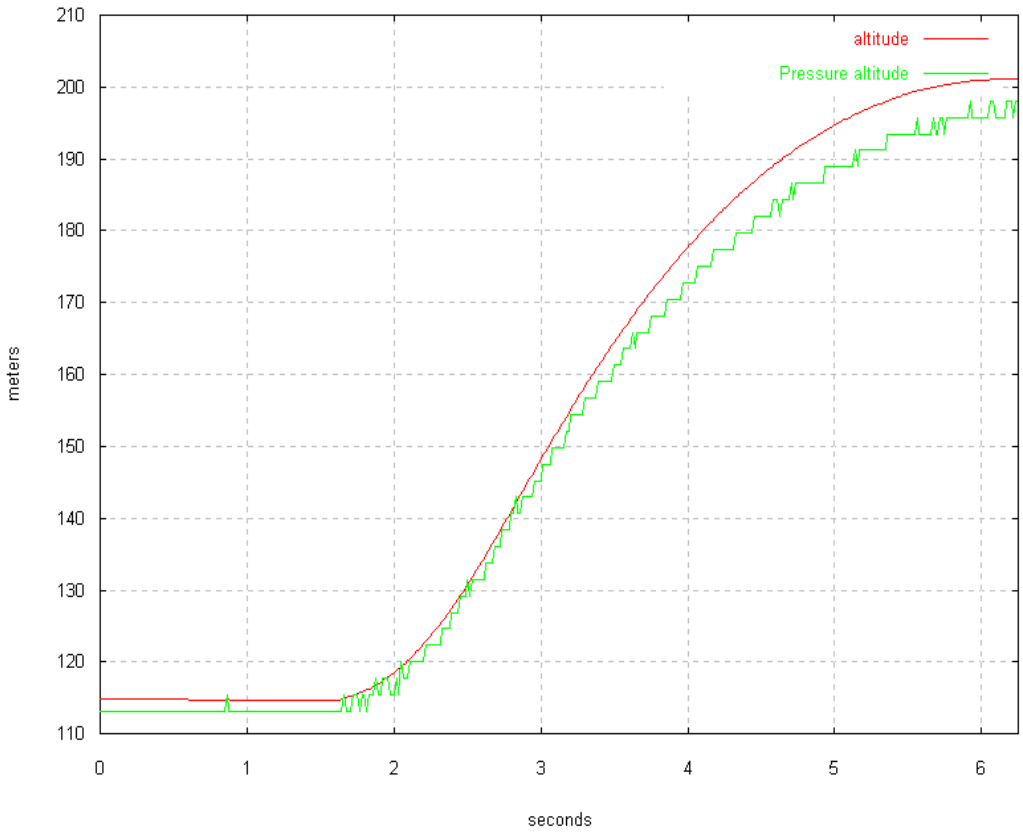
Modify flight code to write an additional data record once it decides that the rocket is on the ground. This record will include:

Pre-launch pressure altitude. (4 bytes)
Pre-launch 1G offset (2 bytes)
Apogee altitude (4bytes)

Modify the program "process.c" to convert the pressure reading and acceleration reading to engineering units. Perhaps toss in inertial velocity and altitude as well for comparison.

13 May 2004

Modified the process.c program to compute pressure altitude using the full strength exponential form, acceleration, inertial velocity, and inertial altitude.



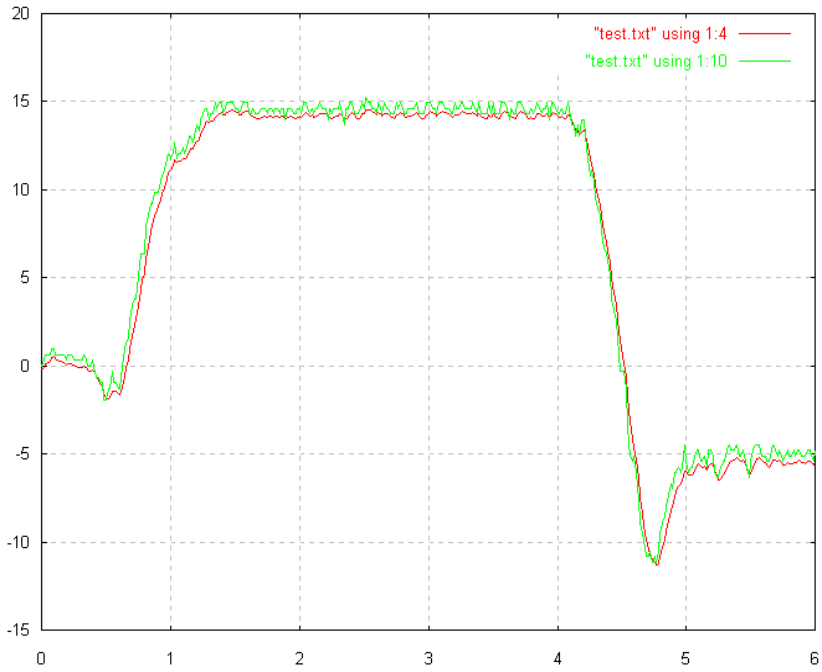

From this plot of the altimeters version of altitude versus the pressure altitude, it is pretty obvious that the linear interpolation introduces a bit of error. I am almost willing to consider working on a second order curve fit to try and improve on this. But that will

require an additional two multiplies in the flight code to implement and I am not sure it is worth the effort.

I have also noticed while looking at this graph that the pressure data really doesn't have enough noise in it. I will have to consider decreasing R6 on the sensor board from its current 10K value.

I located a 5.1K resistor and since R6 was easily accesible, I replaced it. After a quick check I could see that the noise was still too low. Not having anything less than a 5K surface mount resistor around, I just dropped another 5K resistor on top of the first. Now with 2.5K for R6 the noise is about where I like it.

I recomputed the Kalman gains with a higher model noise. This increased the ability of the filter to track the measured acceleration. In fact it might be tracking it too closely but I want to get in a flight test with these gains before I decide.

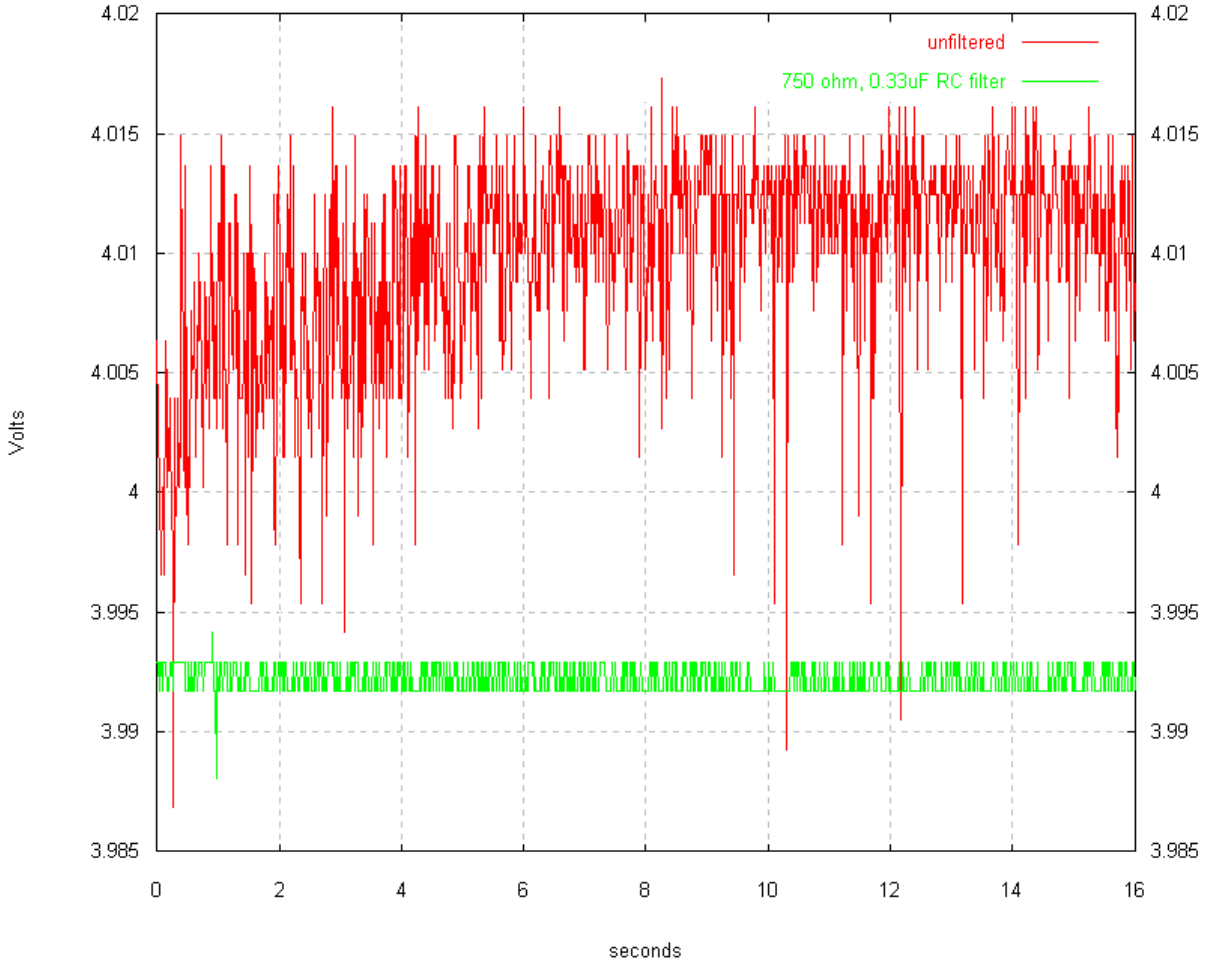



As you can see from this plot of a bench "flight", the Kalman filtered acceleration (in red) tracks the measured value pretty closely.

19 May 2004

I have been trying to adjust the RC filter on the output of the pressure sensor to get the amount of noise that I want in the data. I started out with values of 10K and 0.33uF. I based these values on the recommended filter values of 750 Ohms and 0.33uF and the noise present in the RDAS pressure data.

The RDAS uses a 10 bit ADC and has perhaps a bit to much noise in the data. Since this altimeter uses a 12 bit ADC, the noise would have to be reduced by at least a factor of 4 which would mean at least a 3K resistor. The 10K was obviously too high so I started dropping this value. I first tried 5.1K but that wasn't it. Then 2.5K and that didn't provide enough noise. So finally I removed the capacitor which resulted in unfiltered data and then I installed a 750 Ohm and 0.33uF filter again. Here is a plot of the pressure sensor output for these last two tests:




The data has been scaled to volts. The unfiltered data looks very much like the sample plot in the AN1646 application note from Motorola. But the results of filtering are much better than would be expected. The Motorola plot of filtered data shows a peak to peak amplitude of around 3 mV where I show a bit over 1mV (apart from the spikes early in the data).

I can't explain it but since this is about the amount of noise that I want, this is the filter I will use. The filter would actually perform OK with the unfiltered pressure data but I would have to change the Kalman gains to match the higher level of noise.

That might be something to do on another flight test. But I will need to purchase some more 0.33uF capacitors. When I removed it to get the unfiltered data it flew off never to be seen again and I only had one extra one available. I would hate to be stuck with unfiltered data.

14 July

I had a thought about a way to simplify the computations even more for the pressure only variation of the filter. If I round the Kalman gains to the nearest power of two, I can eliminate all of the multiplications. I troundled out an old piece of C code to test this out and it actually seems to work OK. I had no idea that the Kalman filter would be this insensitive to gain changes. I just have this feeling that there is something waiting to reach up an bite me.

I would like to hack my existing altimeter code into a pressure only version in time to flight test it this weekend. I will up the sample rate to 256 samples per second but I will store data to the EEPROM at the same old 64Hz rate.

Output from kgain31:

```
Input noise values used (standard deviation):
#Altitude      -       1.000000
#Model noise   -       0.010000
#
#Kalman gains converged after 184 iterations.
#
K1 =    0.0314081353
K2 =    0.1272046810
K3 =    0.2283155447
16 bit fixed point gains (fraction only) in hex
K1_HI = 8
K1_LO = a
K2_HI = 20
K2_LO = 90
K3_HI = 3a
K3_LO = 72
```

So K1 will be rounded to 0x08 or 1/16, K2 to 0x20 or 1/4, and K3 to 0x40 or 1/2

This should be very interesting.

First hack at the code reduced the size to slightly less than 1K. Not that I would recommend this for a PIC with only 1K of code space. The only code deleted (besides the multiply) is the stuff to beep out altitude after touchdown. This still requires the multiply routine.

The code isn't working and I have a bug. A really nasty one.

I noticed that the altimeter would go into its normal preflight beeping when I had the RS232 converter hooked up but not when it was disconnected. In addition, the data recorded was bizarre. Some was good but a bunch showed the same wackiness that I saw in the Aurora data.

16 July

OK. After tilting at various windmills in the code, I am getting closer. One very nasty bug that took me a while to figure out was that 256SPS was too fast for the serial EEPROM. It has a maximum write cycle time of 5ms. Which is a bit too slow for 256 SPS that I was writing at. The result was that it only wrote every other 16 byte sample. I didn't notice this until I reloaded the known working two measurement filter code, recorded some data, loaded the barometric only code, and recorded data.

After I got that working, things looked pretty reasonable. Until I looked at the timer value. I had re-enabled the timer code so I could see what the execution time was. It looked to be taking far too long and showed a lot of variation. Which just wasn't right.

This is when I thought that it would be really nice if the MPLAB simulator had a way to count instructions executed. It does. A nifty little gadget called a stopwatch.

So I set a couple of breakpoints at the beginning and end of the code for each sample. Ran the code to the first breakpoint, zeroed the stopwatch, and ran to the second breakpoint. I was surprised to see that it was taking about 15ms to execute. Which was too slow for even 64 SPS.

The main program loop does two things: run the Kalman filter and then run the state machine. I found that each was taking about 7 to 8 milliseconds to execute. Way too long.

So I started stepping deeper into the code. This is when I found that I had messed up when I rewrote the code to go from 256SPS to 128. I had to change from using byte shifts to calling my right shift code. Except that I forgot to store the loop count into LOOPCOUNT. Doh!

That fix reduced the Kalman filter execution time to 1.8ms. Much better!

But the state machine section was still taking too much time. The only part of this code that really takes any time at all is the stuff to store data into the serial EEPROM. And that is where all of the time is vanishing.

This code repeatedly calls a 10uS delay routine for each bit. This is where the time was vanishing to. A quick check of the EEPROM data sheet shows that this delay is not

required. The maximum clock rate is 400KHz. Which my code is not going to exceed because each instruction takes 1us. The delay loop goes.

Good. That dropped the execution time down to 6ms. It still seems a bit high. At 400KHz 16 bytes takes 320 us.

Most of the time is vanishing in the routine to write a byte. I played with it and it only takes about 2.91ms to execute. I can't see how to tighten this up anymore. Hardware I2C support would help a lot.

Main loop execution time is now 4.7ms. Which is plenty fast for 128SPS but not fast enough for 256SPS.

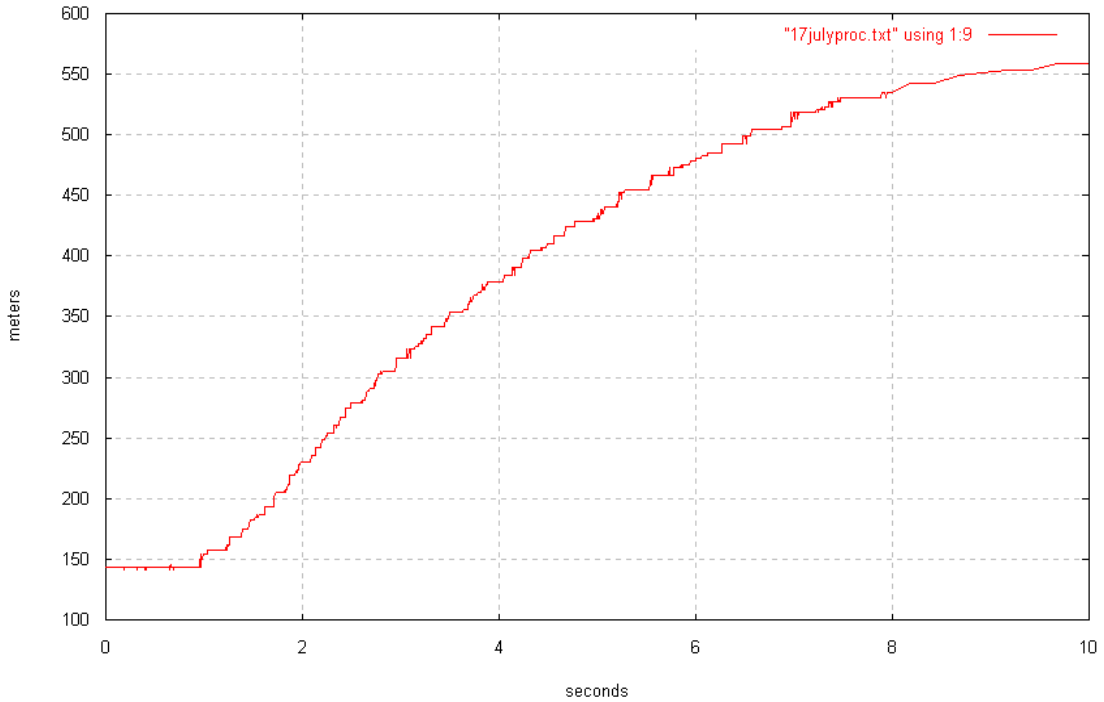
Enough futzing around with the simulator. Time to see if it runs on the hardware.

OK. It works fine. Execution time is running at 4 to 4.3ms. Which is fine.

But the filtered velocity and acceleration seem a little twitchy and the gains seem too high. I need to double check kgain31.c to make sure it is OK.

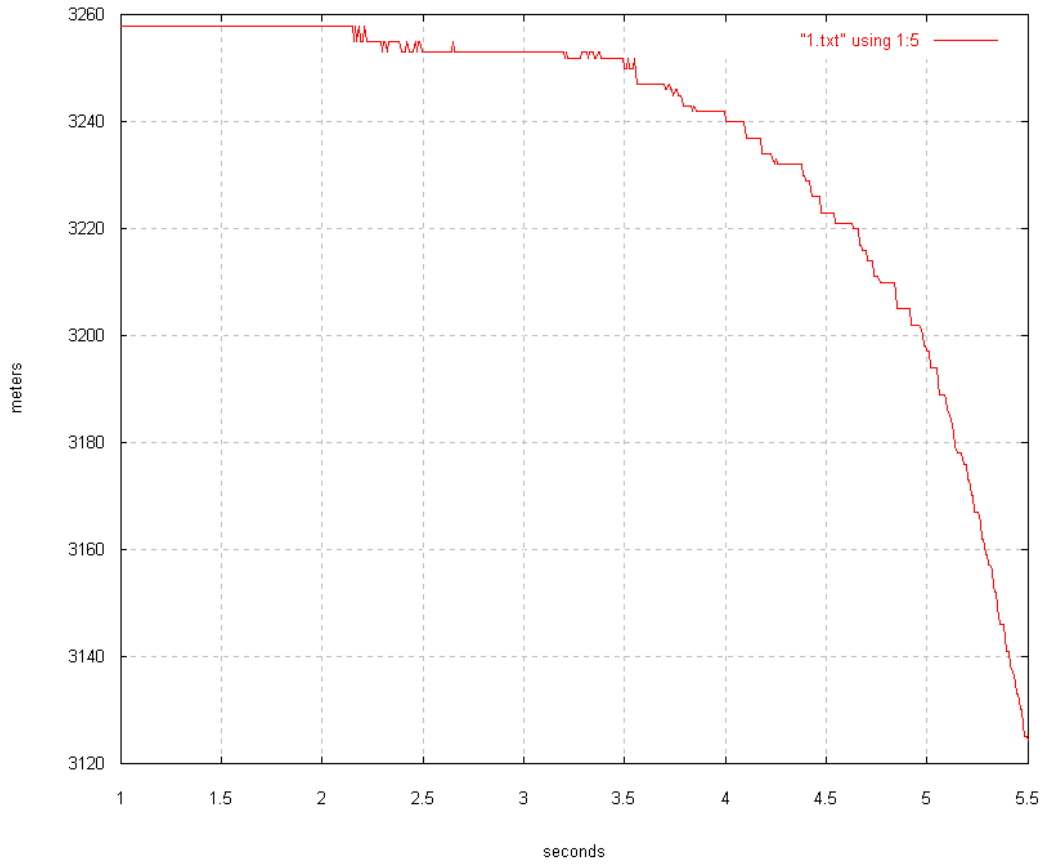
18 July

Flight test results of barometric firmware are mixed. Apogee time was correct but the pressure data looks awful.



I messed around with the ADC code quite a bit so I obviously broke something. I am certain that I haven't violated any of the basic parameters of the ADC serial interface so this is almost certainly a sampling time issue. The code currently sets a sampling time of about 30us.

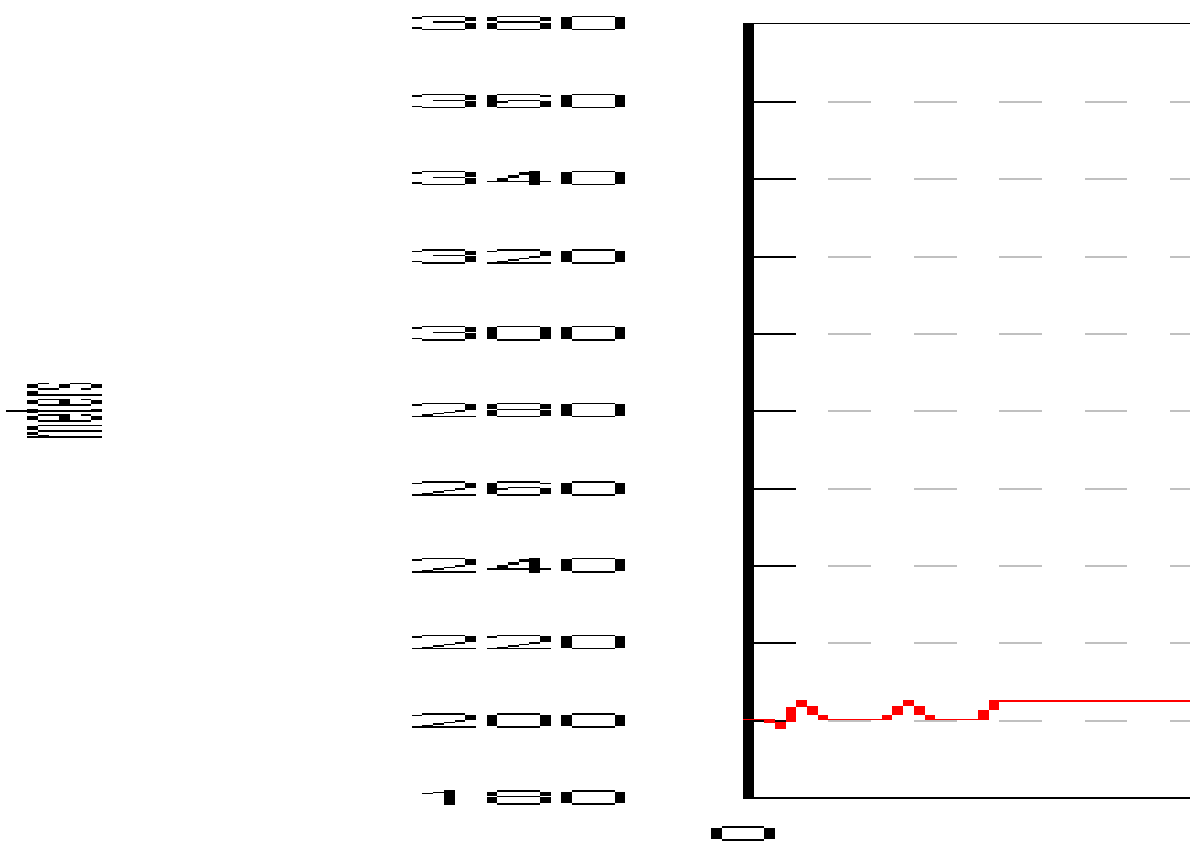
I am using my Foodsaver to simulate flight. Here is the baseline using the current ADC code:



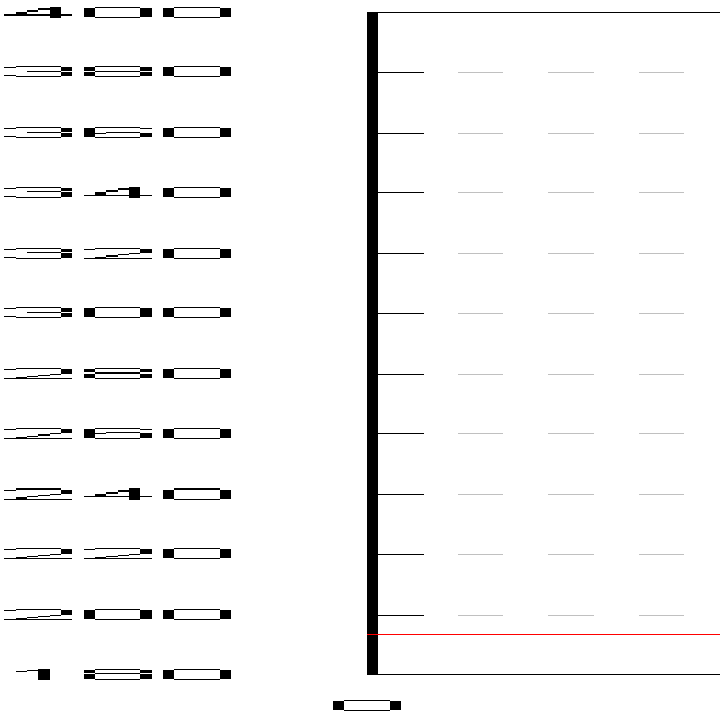

Baseline

After altering the ADC code to decrease the sampling time to just about as little as possible, I did another test. Or tried to. The altimeter was so sensitive that I couldn't even get it into the bag without it detecting launch. This rubbed my nose into the gain issue again.

After looking at the code and running it with various settings, I noticed that it exhibited a problem that I thought I had previously taken care of: premature convergence. So I added a quick check to make sure that it went for at least 1000 iterations. Now the gains make a lot more sense.



Minimum sampling time



Long sampling time