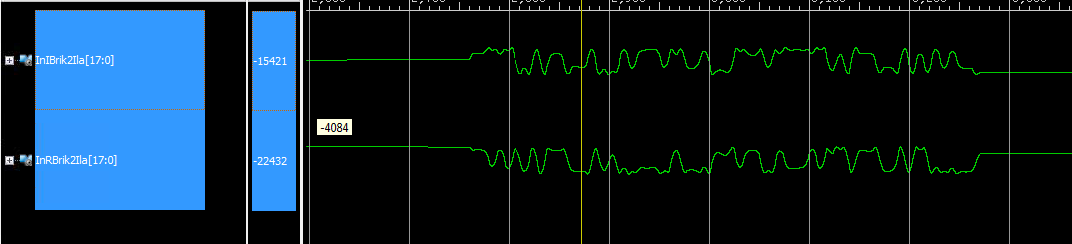
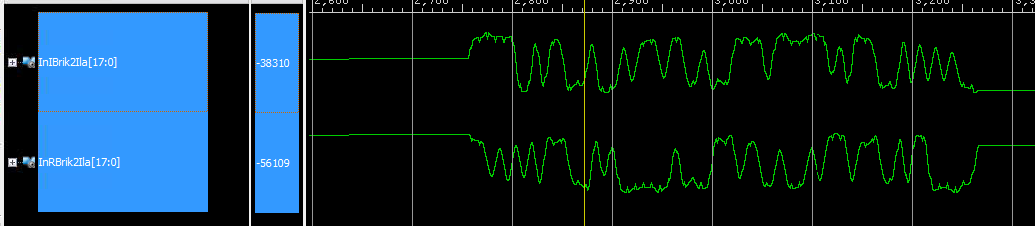
STC Issues and Status

Shown below is the pilot signal with the BYU resampler, after I fixed the HDL, looking as expected. Notice the amplitude at the yellow line as shown in the second blue column as -22422 at the near peak.



This is the same signal with the KTS resampler. It’s twice the amplitude and has periodic ticks toward zero. We’ve seen this working in simulation, so I’m curious if it’s a synthesis issue and if so what else is Xilinx doing to us. Not sure of the gain since I didn’t check the amplitude on the simulations. The waveforms will change over time due to phase offsets, so the two figures may not appear exact.

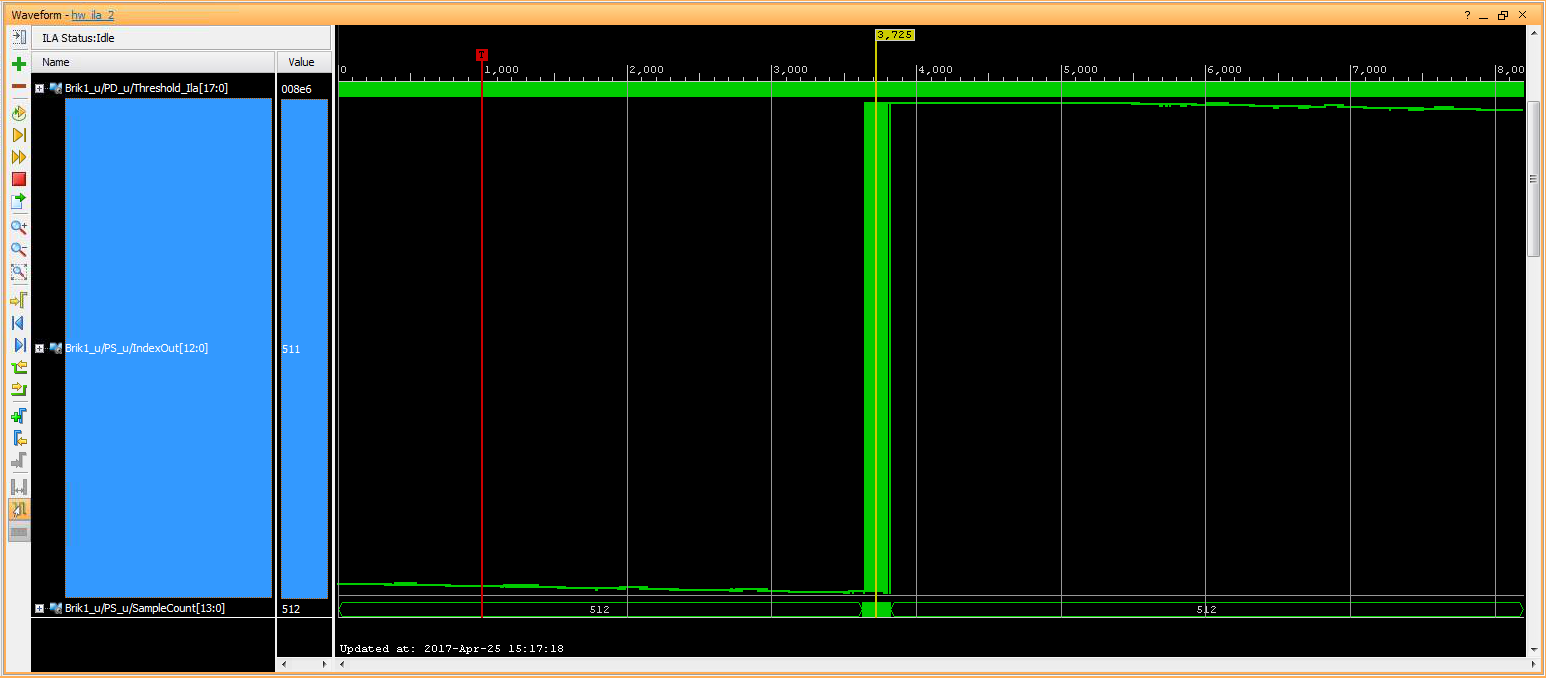


Another issue I’m seeing is the pilot detector glitching as the pilot phase shifts over time. The detector packs 512 samples (I call them packets) into a FIFO then performs a 1K FFT on the packet at 186MHz system clock speed regardless of data rate (10Mbps at moment). These are then cross correlated (complex multiplied) by FFTs of the pilot at 0Hz and ±66KHz, then iFFT’d and peak detected with overlap and add. The address of the peak is the offset of the pilot. The offset feeds a variable length FIFO and generates a Start of Frame pulse for the estimators. The offset has to be accurate within two samples for the trellis which the Time estimate will correct but being off at all impedes the accuracy of the three estimators. The 128 bit pilot, at 4 samples per bit, will fill a packet but the packetizer is completely asynchronous (ie ignorant) to the data stream and will start randomly at boot up, which is why the overlap and add is used. Since the ArbGen and receiver are on separate 10MHz references, some slippage over time is expected. Another issue the C simulation doesn’t address.

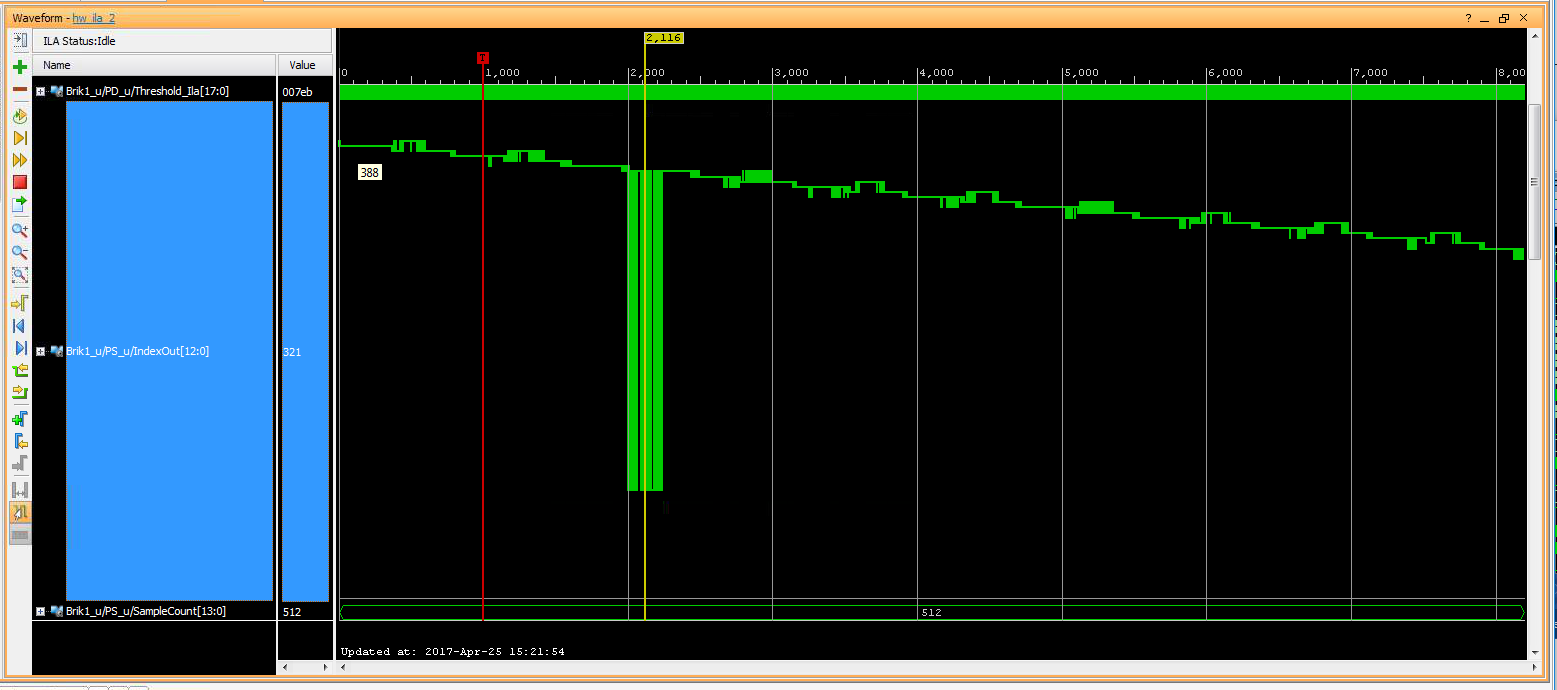
Shown below is the real time (Ocala hardware) pilot offset for 8192 frames @ 3125Hz frame rate or 2.6 seconds (days of simulation). The graph vertical is scaled to cover the 512 offset range. This capture shows the offset transitioning from 263 down to 243 with jumps to 318 or 193 as it passes through 256.



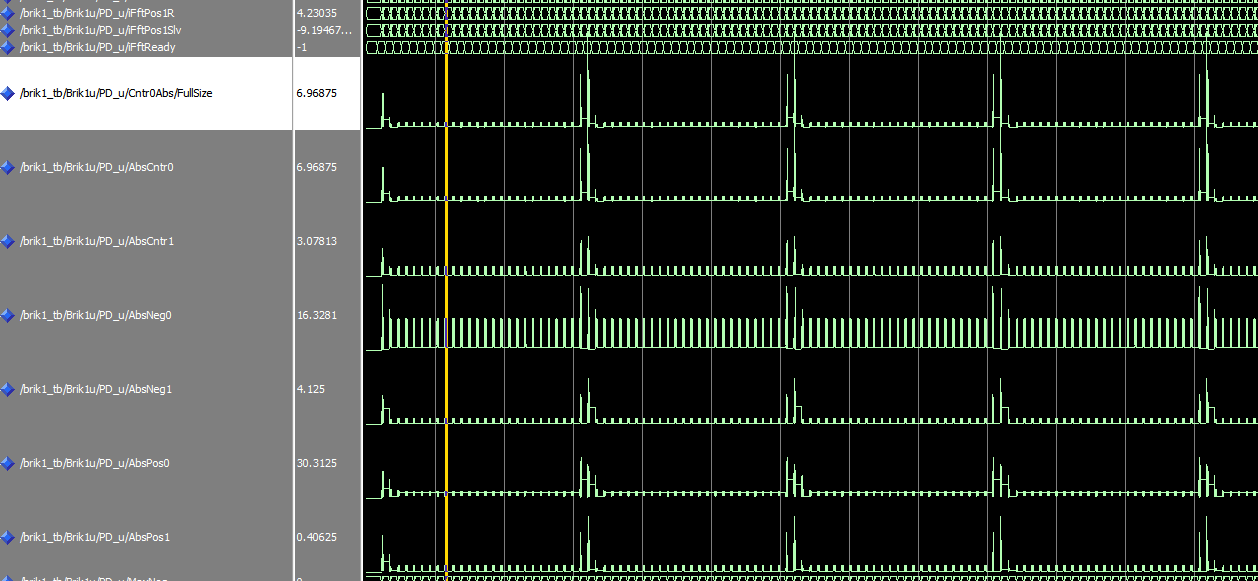
However, wrapping from 12 to 0 to 511 to 504 chatters between 0 and 511 at the transition but no odd values. I expected this to be worse.

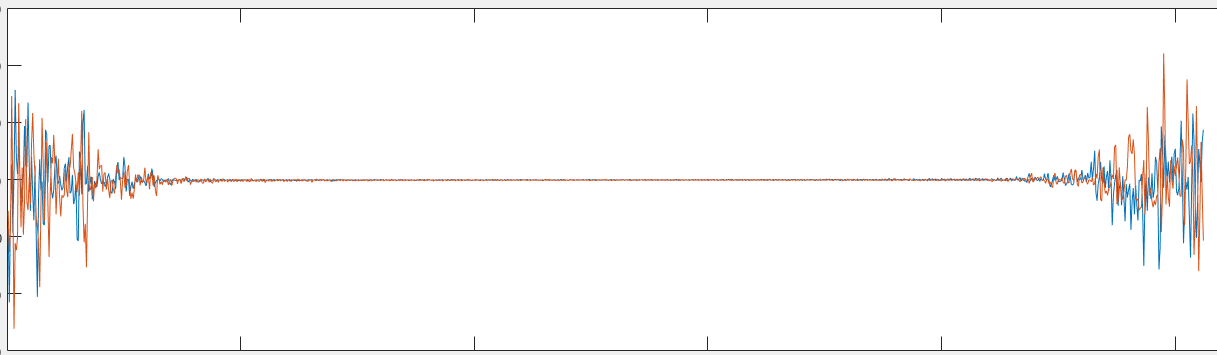


This capture starts at 388 going down to 368 with negative peaks of 321 with occasional 369s if zoomed in to the large pulses. I have the vertical set to a much smaller scale to examine the noise. The smaller noise pulses are ±2 rather than ±1. I haven’t seen these on the other scans.

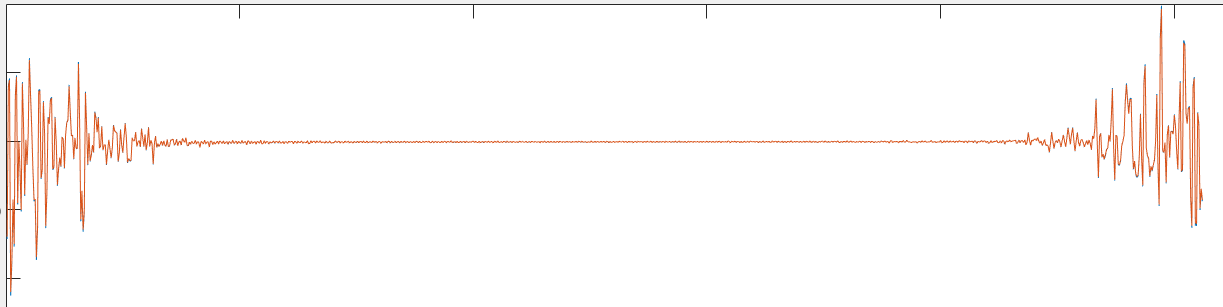


Another issue I’ve discovered in the simulations relates to the negative 66KHz Pilot Detection path. Shown are the six results, real/imaginary and Center, Negative and Positive frequency offset. The FullSize signal on the top is just verifying the accuracy of the AbsCntr0 signal below, so ignore it aside from the fact that they look identical. You can see the pilot going off every 26 frames with a little ‘prepulse’ followed a larger main pulse and little pulses between. The little pulses are the data, which is all 1’s at the moment. The prepulse is due to the pilot having partial energy in the previous packet, but the main pulse will set the threshold between the two amplitudes so the sync will ignore the first pulse. I have all seven signals scaled identically to emphasize the size of the large AbsNeg0 data pulses. I don’t see this in the C or Simulink simulations, but that brought out another issue.



The phase detector uses an FFT of the pilot signal frequency shifted up and down 66KHz which is a simple data file. The Simulink doesn’t use the same pattern for positive and negative as the C code. I’m using the C code pattern, but I don’t see the problem when running the C code. Shown below are the Positive patterns, the negatives are similar.

The Center patterns match between C and Simulink as shown below. I’m assuming due to the largest pulses on the outer edges that these are natural versus bit reverse FFTs. All the complex multiplies, iFFTs and Complex Absolute/Overlap Add are HDL modules. The only difference in the code are the template file and signal names. The positive pulses look fine, so why are the negatives so big??? Ah, found a possible cause for the noise, but the C vs Simulink issue is annoying. I’m beginning to wonder is the Simulink turned into the master program and the C used to verify most issues. I used the C as my gold standard and ignored the Simulink, especially the estimates and pilot detect. This may have been a bad direction, since the Simulink is what was supposed to fly. Looking at Brik2, I had issues correlating it to the C.



List of known issues and status. If fixed, I say so upfront thus the Dones.

1. Done. Timing is changing just after Trellis Starts. Trellis was triggered on FreqDone but ChanEst is last with FreqDFT
2. FreqEst hiccups.
3. Trellis Start with TrellisCount of 780. Get ChanEst change. Probably from PD double starts, which I think I’ve fixed but haven’t verified.
4. Done. Always get two spare nibbles at start regardless of interpolator offset up to 13. needs 16 per nibble. 21 seems optimal.
5. Option to move Time and Chan Est before DF
6. Done. FreqLpf is -1315 when FreqCoarse = -1346
7. Done. Transmitter data is iffy, Data fades mid frame twice. Replaced with Arb Gen
8. StartTime rising on TrellisCount 21 due to Start offset. See item 3 above.
9. Verify hreads are correct.
10. The H0Neg templates don’t match with the Simulink, which may be bit reversed but the edges grow like natural. H0Cntr are Ok. The Neg iFft is much larger than the center during non-pilot packets.
11. -4dBm is maximum input level to prevent clipping, yet -7 needed for DFs. The KTS resampler seems to have gain of two.
12. The Detection Filters may have gain, but just at quick glance while chasing other issues.
13. Setting PilotSyncOffset to 1396 gives desired all F's output data but wrong frequency estimates. Freq was forced to 0 since no offset on input. This also yields H0=0.011 H1=0.029 with no H1 input. Tau0=-1 Tau1=1. The H estimates are too small and will affect the frequency estimate. The pilot I/Q signals, after a phase compensation, are mirror images of each other. I believe the pilot wants to start 2 samples into the packet following the Start pulse from the PilotSync. The matlab script PilotSyncCompare will read in the two I/Q 512 sample packets and rotate them by the given phase offset. I typically call it from a for loop of ‘for i=0:10:360; PilotSyncCompare(i, 2); pause; end’ to get close to the correct phase, then adjust the offset.

Had Arb Generator programmed with one frame if data at 46.6MHz DDC rate. This is 14.933.333 samples, 13312 \* 175 / 156. Need three frames to compensate, else Pilot Detection Index moves every 3 frames so generator now runs three frames repeatedly. The Arb Gen was connected to Doug’s old laptop in web control mode via local intranet. The ‘toolkit’ to program the Gen is on the laptop. This allows remote access via TightVnc. The Toolkit icon was on the desktop, it has a red bar across the bottom with an antenna radiation pattern on a blue background. Use most recent config, then under ‘Waveform Setup’ set the Source file type to CVS, Source File to RealFile.txt, Use separate Q = yes, Q file to ImagFile.txt, Sample rate to 46.666….MHz. Enter a Arb Destination file. All files are in c://engineering/stc. Click Instrument tab and set Frequency to 70.0MHz, -8dbM. Then hit the download/play button on the top left.

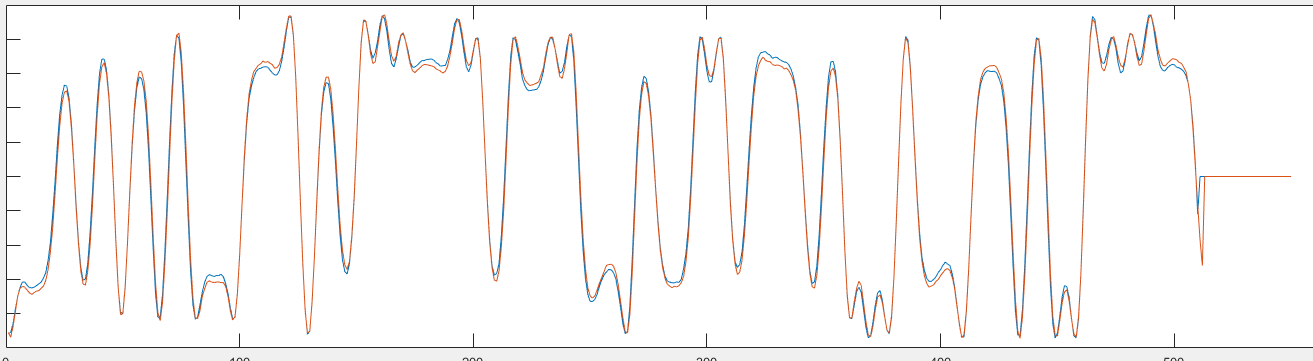
Hardware is setup with the KCU105 running on its wall wart. Another wall wart drives the AD9746 A/D converter plugged into the FMC LP connector. The ArbGen drives the input while a second RF source at 93.333…MHz +13dBm drives the clock. The HDL for the A/D is operational but not optimal.

Connect the KCU105 to the laptop via the SuperSpeed USB or the waveforms load very slowly. Run the hardware manager in Vivado 2016.3 or newer. The hw\_vios are used to control the system real time. Setting the FreqResolution to anything but d123 forces the mixer oscillator to said freq, typically 0 or 1.

Probe7 is the MiscBits register define in the semco\_pkg.vhd file, usually set to x17 or x07 for BYU or KTS resampler. The LSB must be set for the pilot detector to work with the conjugate bit set. Leave ‘0’ for simulation.

Probe8 is the interpolator offset and usually x015, d021.

To\_ufixed is the PilotSyncOffset and set to 1386 or 1388. Monitor the PilotSync/SyncSum line for minimal value. This will change as the phase rotates so try to capture similar ReadR/ReadI signals for a closer reading. Below is a properly aligned H0 pilot waveform from PilotSyncOffset(158,2);



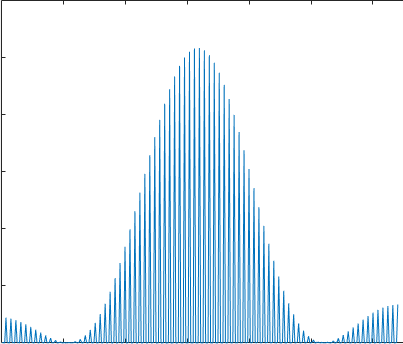
The other VIOs are used at the moment since they aren’t connected in STC.vhd. They will be used to program the other bit rates when the time comes. The PrecisionReqmt.xlsx file is used to calculate the values. Enter the bit rate in D39, currently 10e6. The VIO values are in J48:55 but not in proper order. The control software in the radio can eventually use the equations in D48:55 to calculate the values. All the variable bit rate registers are in the structure Variables in STC.vhd. The file currently uses the VIO\_0 IP core, which will need removed. Move the Variables signal from the signal section line 232 to the module inputs line 53 and tie to the external controller. Should be all that’s required for other bit rates, except for the PIlotSyncOffset which may change depending on the number of packets required to calculate the PS/IndexOut. The higher the rate, the less dead time between packets and thus the issue. The calculation time should be the same regardless. Debated added a bypassable 256 clock DelayLine module in the PilotSync path should the offset get near the edge of a packet to move the transition to the middle of the next packet.

The PilotDetect module first packs the next 512 I/Q resampled samples into packets at four times the bit rate. These are then burst out at 186MHz into a 1K FFT. The FFT data consist of the 512 samples and 512 zeros. The FFT outputs, in natural order, are complex multiplied by the stored FFT of the pilot I and Q signals then inverse FFT’d to form a cross correlation (multiplying in frequency is correlating in time). After a delayed copy of the previous packet is added to form an Overlap and Add function, the Complex Absolute Value is calculated over the first 512 samples only. These outputs are searched for the Index (address, count) of the maximum value over all 6 iFFTs. The Index is fed to the PilotSync module with aligns the Pilot burst with the next output packet with a leading StartOut pulse. The last 25 maximums are checked for the two largest (Peak1 and Peak2). Their average sets the threshold for the next maximum. One issue with this is signal level dependence. The average of 0 and 0 is 0, so PilotDetect will go high. May need a minimum amplitude detector to prevent this or ignore rather than drop frames. The StartOut signal is triggered in the middle of its packet since it has to check all of the first 512 values. The last 512 values are ignored.

Each time the magnitude exceeds the threshold, a GoodPilot counter is incremented to a max value of 3 and PilotDetect is declared. The next 25 packets should not exceed the threshold which increments the BadPilot counter 25 times. If BadPilot reaches 128, then PilotDetect is negated. Each GoodPilot increment clears the BadPilot counter.

Current issue with the PilotDetect module is the noise floor is higher than the software models. The peak should be 8-10x the noise.

The PilotSync module stores the packetized data in memory. The Pilot Index then determines the offset into the incoming packet to start the next out going packet. A frame (pilot plus data) consist of a start pulse followed by 26 packets with a data valid signal bracketing each packet. The first packet will contain the pilot signal followed by 25 data packets. Where in memory the new start is contained is a function of propagation delays through the PilotDetect. A PilotSyncOffset variable is set to correct. This ends the “Brik1” section of the code.

The Brik2 function contains all the estimation logic. The FreqEst starts with a packing fifo that holds the 512 samples then generates a Start signal. A Training Sequence is calculated by multiplying each sample of a ROM addressed P0/P1 complex signal by the H0/H1 estimates from ChannelEstimate module. The template is then complex multiplied by the FIFO data per clock to form PilotTemplate3. The last three sets of FIFO data are also stored in buffers to form a 4 frame memory. The previous data is also multiplied by the current ChannelEstimate. These four sets are then decimated by 64 down to 8 samples per pilot. These 32 samples are spread across 832 zeros simulating the data packets in between. In the software, they are fed into a 4K double precision FFT just to find the peak value/index which determines the center frequency of the pilots and thus the frequency offset for which to correct. This takes 8325 clocks but is just a starting point. The frequency spectra produced is shown at the right resembling a bell curve picket fence, note the spacing between pickets. Since the next software step is to search for the maxima via a DFT, I used 8 parallel DFTs to scan across the spectrum at a quarter of the picket spacing. Note that the spectrum could slide left or right depending on the offset. The quarter picket spacing allows the results to be at least ~80% of the peak. I keep a running maximum Index to find the two peak DFTs much like the FFT would but much faster, partially due to parallel processing but with comparable hardware. FFTs are expensive.

These two peaks then perform the DftSearch the FFT would have called next. The DftSearch does a sort of successive approximation technique around the frequency of interest searching at Offset = half a picket distance. It does three DFTs, one Offset below, another Offset above and at center frequency. The peak of these three determine the new center and the Offset is cut in half. This continues till the offset goes to zero, with the result sent back to the main routine.

Another cost saving with the DFT is the software calculates all 832 values of the “FFT” data while rotating a phase relative to the frequency of interest. Since 800 of the 832 values are zero, I skip over them and calculate the phase 200 clocks between groups of eight.

Since the top of the curve is all but flat, I thought of adding a check that the two sides of the curve are even since they have the steepest slope, but not yet implemented.

# Time Estimate

The timing estimate compares the last half of the pilot to a reference of H0 and H1 and generates a 2D map of the surface searching for the bottom. The point of the timing estimate is to compensate for arrival times of the two airplane antennas since the plane can rotate away from the receiver increasing the physical relative distance to the radio. The C code did some complicated triangle flipping that, I believe, looked for the highest of the three points of the triangle, then flipped the triangle along the hypotenuse of that point. Kind of rolling downhill. This just continues for a fixed count and stops.

If you think of the 2D array as a square bowl that is 65 by 65 cells, dividing the big square into four 32 by 32 smaller squares, ignoring the far edges for now, I came up with a 2D successive approximation that does four parallel calculations starting at the center of each smaller square. The smallest result becomes the center of the next calculation and the dimensions drop in half to 16 by 16. This continues in decreasing powers of 2 till we’re testing adjoining cells. At this point, keeping track of the minima, the search continues 1 step at a time a couple more iterations which will hit the 65s if needed.

Both approaches assume that the bowl in monotonic (only one low spot). If there’s a dimple in the bowl, the result could be in the wrong spot. I haven’t mapped the bowl in Matlab yet to determine validity.