`InverseFilterbank`: FFT bases PFB inversion

Part of the PST signal model involves implementing a FFT based PFB inversion algorithm. This algorithm takes oversampled or critically sampled channelized data and produces a higher time resolution signal with a smaller number of channels. From a science perspective, this algorithm is desirable in that it allows for analyzing time series at higher time resolutions, potentially revealing fine temporal structure that would not be present in more channelized data. In the context of the SKA, introducing FFT based PFB inversion to the PST signal chain could remove an upstream processing step, namely the stage where the CBF performs a synthesis filterbank operation on high frequency resolution beamformed data streams.

FFT based PFB inversion, or simply PFB inversion, is distinct from the synthesis filterbank algorithm. Both algorithms represent synthesis operations, or recombining channelized data to produce higher time resolution data. The synthesis filterbank is highly constrained by polyphase filterbank filter design, making it unsuitable for use in some signal processing settings. On the other hand, PFB inversion is lossy, in that it will not perfectly reconstruct some upstream time series. If some time series is fed through a channelization algorithm, which is in turn passed to either the synthesis filterbank or PFB inversion algorithm, the synthesis filterbank can perfectly reconstruct the original time series, while the PFB inversion will introduce some error. With some tweaks, this error can be properly mitigated.

The purpose of this document is to highlight some of the modeling and implementation work we have done in order to meet the acceptance criteria for feature 129. To this end, we have implemented the PFB inversion algorithm in the context of the PST prototype DSPSR, a widely used, industry tested pulsar signal processing library. This implementation matches the updated Matlab PST signal model, developed as part of the CSP CDR, to within floating point numerical precision. Moreover, it meets the level three purity requirements as discussed in the PST Matlab signal model document. Moreover, we are able to show that PFB inversion is correct inside the larger pulsar processing capabilities of DSPSR.

Matlab PST Signal Model Updates

We introduced two modifications to the PFB inversion algorithm in order to meet the PST signal model requirements. The first involves correcting for the polyphase filterbank FIR passband and the second comprises applying FFT window functions and using an overlap discard technique to attenuate the effect of sharp cutoffs when moving between processing blocks. Moreover, we investigated how properly aligning the PFB inversion FFT window can affect phase discrepencies between input data and the output of the PFB inversion.

Combined, these modifications contribute to a significant increase in temporal and spectral purity.

Derippling

The Polyphase Filterbank algorithm uses a low-pass FIR filter as part of channelization. When the number of filter coefficients is relatively small, these filters have some bandpass ripple, which can play a role in diminishing spectral purity. We effectively correct for this during PFB inversion by dividing the results of forward FFTs with the frequency response of the FIR filter used during channelization. This derippling correction can have the same effect on spectral and temporal purity as dramatically increasing the number of filter taps, without making the same impact on computational efficiency.

FFT Window Functions

Previous work with the Matlab PST Signal Model code did not involve working with streaming data; only single processing blocks were used at any given time. Initial modeling work for PDR/CDR assumed that the processing block edge effects could be effectively mitigated with sufficient block overlap, but this had not been rigorously evaluated. We discovered that with no means of handling block edges in place, the PFB inversion algorithm results in significant temporal leakage at processing block edges. This leakage can be somewhat mitigated by implementing an overlap discard method. Even with overlap regions half the size of the processing block, the temporal leakage could not be appropriately diminshed. Ideally, the overlap region would be kept to a minimum, as it effectively increases the number of blocks the algorithm has to process. Using overlap discard in conjunction with FFT windows reduces the ringing introduced by the sharp-edged window function inherent in any time domain fourier analysis, allowing us to meet the PST temporal purity specification. Moreover, these FFT windows allow us to keep overlap regions to a minimum, decreasing computation time. Our modelling efforts suggest that a Tukey window function offers the best spectral and temporal performance.

PFB Sample Alignment

Care must be taken when choosing at which sample to start inverting oversampled, channelized data. If we don't have information regarding at which point the upstream channelization algorithm started, we can introduce a phase shift between the original input data and the inverted data. This phase shift cannot be corrected simply by offsetting the input or inverted output an expected number of samples – it is an artifact of oversampling, inherent in the inverted data due to our choice of FFT window alignment. In order to eliminate this

issue, the PFB inversion algorithm has to know on which sample any upstream channelizers started. Moreover, the sample on which PFB inversion begins has to be a multiple of the numerator of the oversampling factor plus any channelization induced offset. Ensuring that appropriate sample offsets can be computed requires careful bookkeeping of all upstream layers of channelization.

Matlab Signal Model Purity

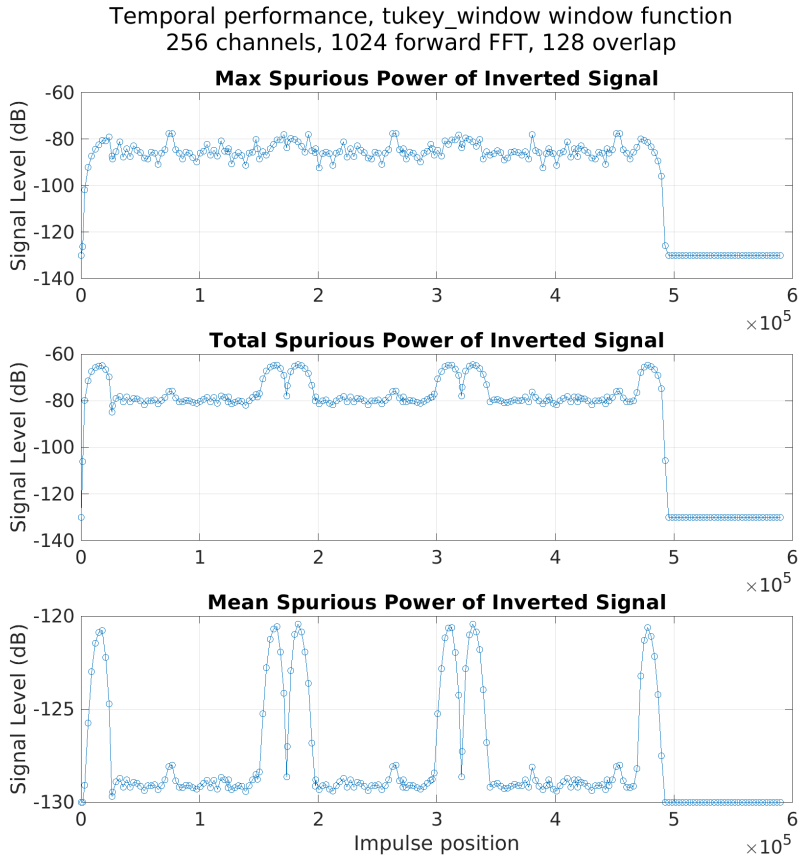

Figure 1: Current temporal purity of the Matlab PFB inversion implementation.

With the modifications we made to the PFB inversion algorithm, we were able to drive down error to the point where the algorithm meets PST signal model purity requirements when processing streaming data.

The general prescription for running purity tests consists of generating some

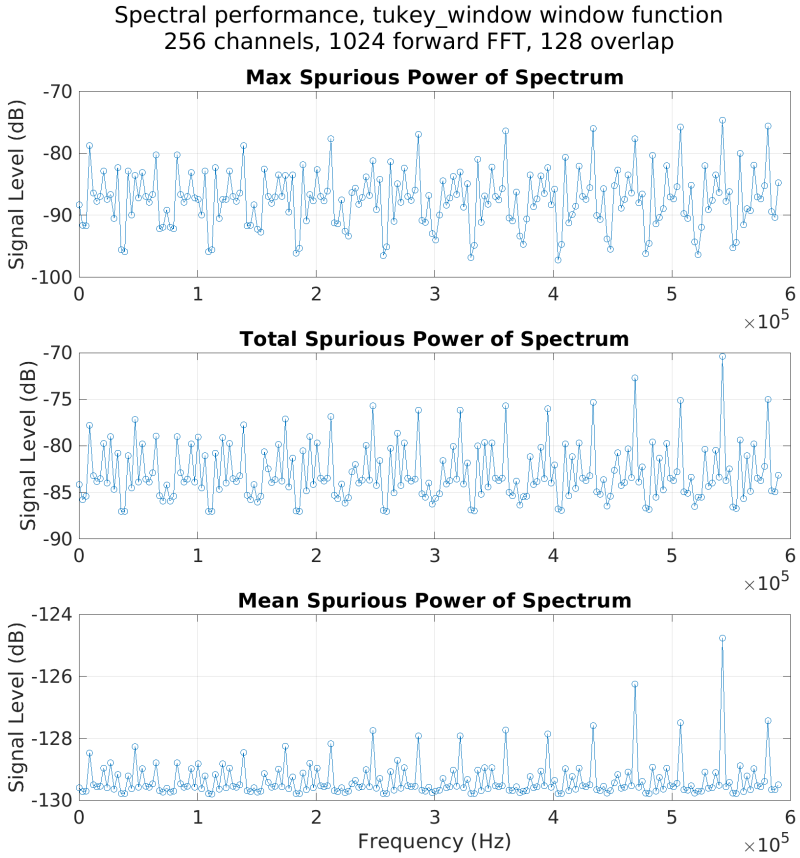

Figure 2: Current spectral purity of the Matlab PFB inversion implementation.

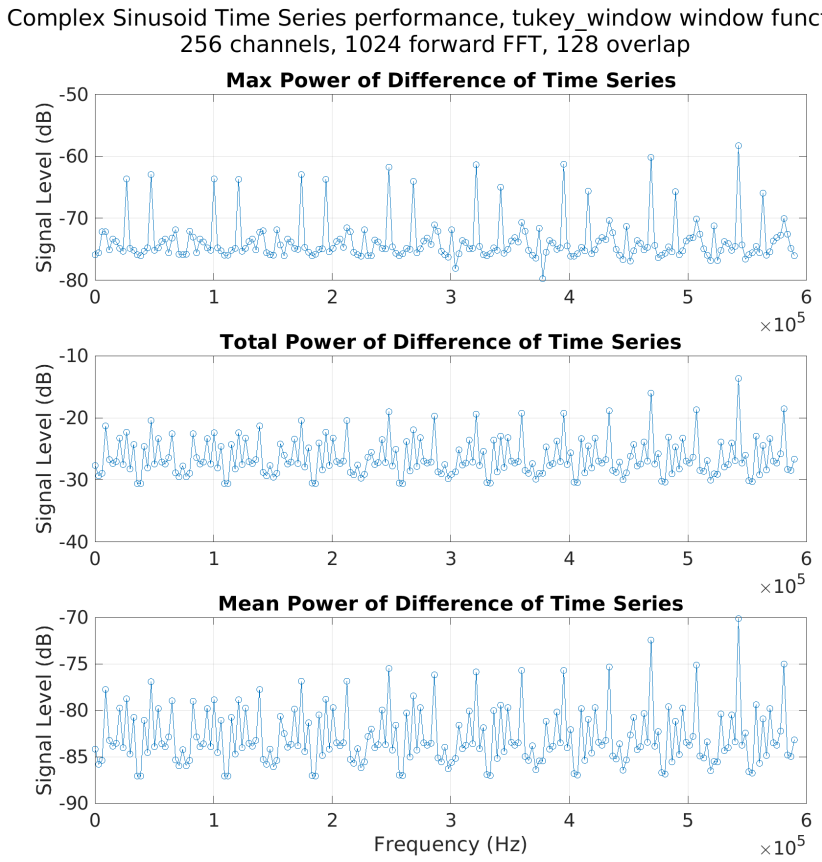

Figure 3: Max, sum and mean of power of complex difference of input and inverted complex sinusoids as a function of sinusoid frequency.

input signal, channelizing it, and then inverting the results of the channelizer. We then compute two purity measures, the max spurious power and the total spurious power. Here, spurious power is the power of the signal when the maximum value has been set to zero. Depending on whether the purity test is targeting the time or frequency domain, these measures will be computed with respect to the inverted time series, or an inverted power spectrum.

From the PST signal model document, the recommended levels for error measures are as follows.

1. max-max-spurious power – temporal (-70 dB)
2. max-max-spurious power – spectral (-70 dB)
3. max-total-spurious power – temporal (-60 dB)
4. max-total-spurious power – spectral (-60 dB)

The following section addresses each of these requirements individually, using figures 1, 2 and 3 as visual references. These figures were produced with a 256 channel PFB using 1024 sample forward FFT, and 128 sample overlap region in the PFB inversion step. Better performance can be acheived using larger FFT lengths.

Figure 1 shoes the temporal performance of the Matlab implementation of the PFB inversion algorithm for a variety of impulse positions across the operation domain. The three subplots in the figure show the max, total and mean spurious power of the inverted time series.

Figure 2 shows the spectral performance of the Matlab implementation of the PFB inversion algorithm for complex sinusoids of a range of frequencies. The three subplots in this figure show the max, total and mean spurious power of the spectrum of the inverted data.

Figure 3 shows the maximum, sum and mean of the power of the complex difference of input and inverted complex sinusoids for a range of frequencies. These performance indicators are not used in the PST specification, but they are useful for demonstrating the correspondance between spectral purity and the numerical similarity of time series.

1. The first subplot in figure 1 shows that the max spurious power is below -70 dB. Close to the block edges, the error does creep up close to the reference measure.
2. The first subplot in figure 2 shows the max spurious power for complex sinosoids. For all tested frequencies, the max spurious power is within specification.
3. The second subplot in figure 1 shows that the total spurious power is below -60 dB. As impulses approach the edges of processing blocks, the total spurious power does approach the specified limit.
4. The second subplot in figure 2 shows the total spurious power for complex sinosoids. This is below the -60 dB for all tested frequencies.

PFB inversion in DSPSR

The main objective of SP-129 was to implement the PFB inversion algorithm in DSPSR. We created several new classes inside DSPSR, in addition to implementing some architectural tweaks to accomodate the PFB inversion's synthesis paradigm. In the end, we added roughly 800 lines of C++ implementation code to the DSPSR codebase.

The most important classes to the DSPSR PFB inversion implementation are the `InverseFilterbank` and `InverseFilterbankEngineCPU` classes. The former prepares input and output data buffers for the operation, in addition to setting up any associated time or frequency domain windows. An example of a frequency domain window might be a dedispersion kernel. The `InverseFilterbankEngineCPU` class is what actually performs the PFB inversion algorithm. As it's name suggests, it is designed to work on a CPU. By splitting the preparation and operation stages of the the PFB inversion transformation, we open ourselves to creating algorithm implementations that target different hardware. In a subsequent program increment, we plan to implement a `InverseFilterbankEngineCUDA` engine which operates on CUDA enabled GPUs.

DSPSR has traditionally been used for channelizing time series, meaning that much of the code contains assumptions about the nature of the relationship between the input and output of transformation operations, namely that the number of output channels will be larger than the number of input channels.

In addition to the main `InverseFilterbank` and `InverseFilterbankEngineCPU` classes, we created a number of other adjacent classes that represent useful software representations of things like FIR filters and FFT windows.

`InverseFilterbank` validation

We created a Python test harness to validate the C++ PFB inversion implementation against the Matlab signal model code. This harness relies on dumping the state of the signal after the `InverseFilterbank` operation, allowing us to compare the results of the C++ and Matlab PFB inversion implementations. After testing complex sinusoids, time domain impulses and simulated pulsar data we were able to determine that the Matlab and C++ implementations match to within the bounds of single precision floating point accuracy. Figures 4, 5, and 6 show these differences.

Another aspect of the `InverseFilterbank` validation is ensuring that it works in the larger context of DSPSR. To test this, we generated some simulated pulsar data, and dedispersed it with DSPSR. For convenience, we call this DSPSR use case "vanilla" DSPSR. We compared vanilla DSPSR to the output of using DSPSR to invert and dedisperse a channelized version of that same simulated pulsar data. In order to compare the state of each of these signal chains, we

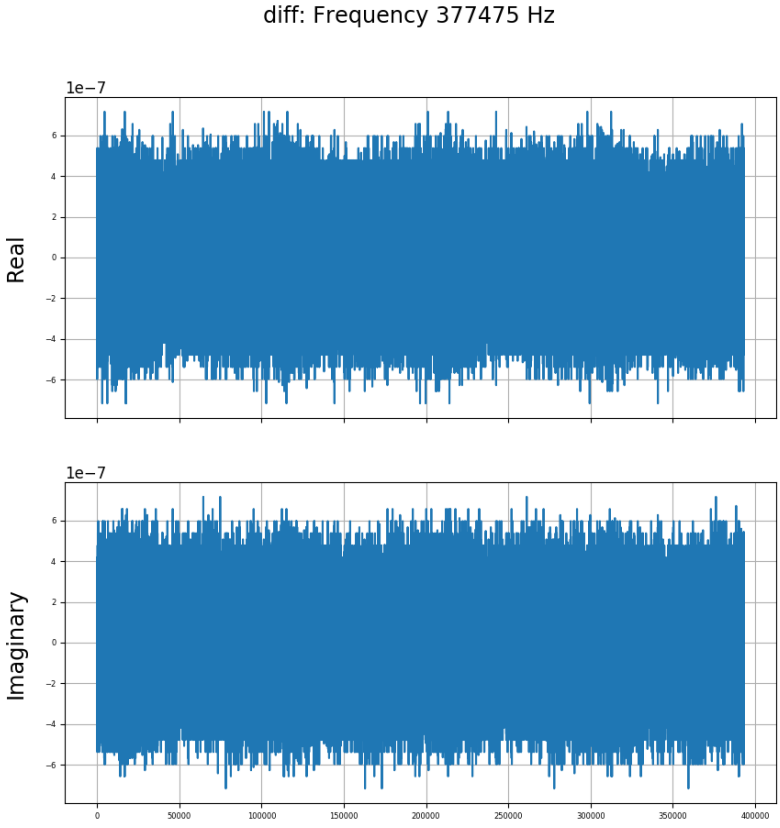

Figure 4: Difference between Matlab and DSPSR PFB inversion implementations for a complex sinusoid.

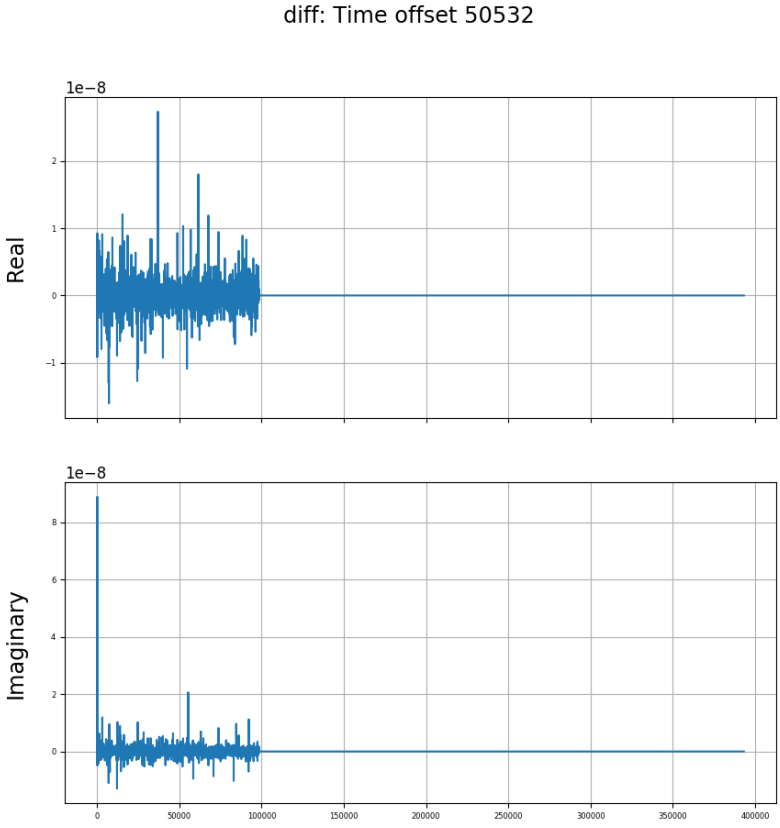

Figure 5: Difference between Matlab and DSPSR PFB inversion implementations for a time domain impulse located in the first procesing block.

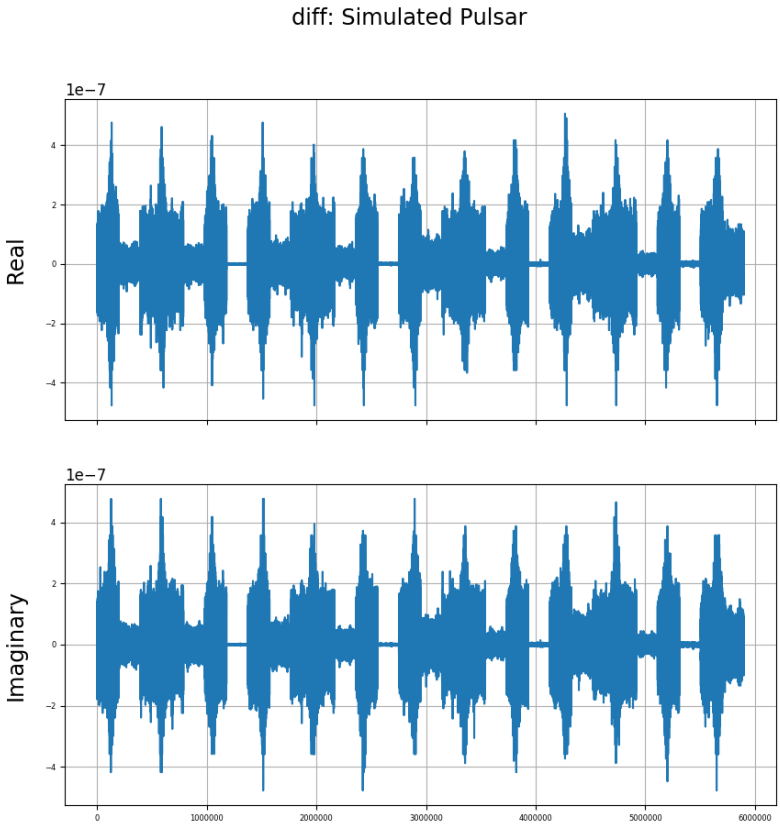

Figure 6: Difference between Matlab and DSPSR PFB inversion implementations for a simulated pulsar.

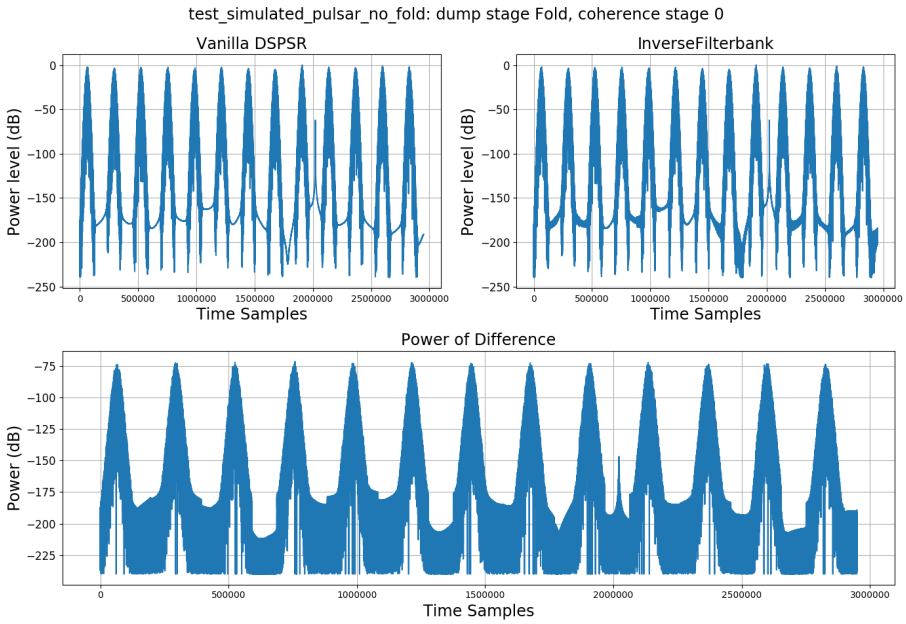

Figure 7: Comparison of "vanilla" DSPSR and DSPSR with PFB inversion and dedispersion enabled when operating on simulated pulsar data.

dumped the state of the signal before the Fold operation. Figure 7 shows one of the coherence components for this comparison. The other three Coherence components illustrate a similar point: the numerical difference between vanilla DSPSR and DSPSR with PFB inversion and dedisperson enabled is below -70 dB.