ENEE 630: Advanced Digital Signal Processing

Project 01/02: Designing a $\pi/4$ BPSK TX/RX system

Anirudh Nakra, İsmail Coşandal

University of Maryland
College Park
Prof. Tahereh Fazel

November 30, 2021

Abstract

In this project, we create an efficient TX/RX system for $\pi/4$ BPSK modulation and then further try to optimise the receiver while reducing the errors in time and frequency delay estimations. We get meaningful results and find that our system estimates the frequency and time estimates ideally in a clean channel and the BER curves in the presence of a noisy AWGN channel with doppler and time shifts closely follow the ideal BER curve. We finish the project by using 4 major optimisation techniques to reduce resources and make it suitable for a fixed point implementation. We also analyse our system using a RX MIPS table.

In the second part at Section 4, we explore fixed point receiver and some other modifications.

1 Floating Point System Design

1.1 Transmitter

1.1.1 Signal Generation

We start with burst formatting and generate the frame we work on. From project specifications, we have been given the sampling rate which is 16000 Hz as well as the duration of each frame which is 50 ms. From this information, we find that a single frame is 800 bits in length. The frame is further subdivided into three parts. The start of the frame is a pilot signal that will be used to correctly estimate the reception of the frame as well as the change in time and frequencies that might occur due to the noisy channel. The pilot channel occupies 128 bits at the start of the frame and we have chosen a 0 tone to be our pilot for convenience purposes. This pilot should give us an impulse when looking at it in the frequency domain on the receiver side. We shall analyse it in detail in the coming sections.The next component of the frame is a key which is a 8 bit sequence of 0's and whose role would be made more clear in the subsequent phases of the project. After this comes 664 bits of binary data that we have randomly generated in MATLAB.

1.1.2 Modulation

From here we move on to the modulator. Our modulation of choice is pi/4 BPSK. This is one of the simplest PSK modulation schemes and our work to benchmark our floating point system will be focused around it. It is clear from the signal constellation diagrams for a traditional BPSK that the two symbols are phase shifted by pi and we are using $\pi/4$ BPSK which implies that our symbols rotate $\frac{\pi}{4}$ in each indice. We use this fact to perform the modulation of our randomly generated data. Several approaches can be taken to implement this. A simple formula is

$$X(k) = e^{j\pi k/4}(1 - 2Data[k])$$

but we have simply used the above mentioned property/definition to modulate the data rather than working in exponentials.

1.1.3 Upsampling and Filtering

The modulated data stream is then passed through the final phase of our transmitter. We first upsample the modulated data frame 16 times. This means that we now have 800 symbols of data that correspond to 16*800 samples rather than just 800 samples as before upsampling. This upsampled data stream is then further passed through a low-pass cosine raised filter. The cosine raised filter has a rolloff coefficient of 0.35 and covers $\pm$ 3 symbols with 97 taps. This enables us to define sps and span of the filter to implement it in MATLAB. The sps field is defined as the number of samples per symbol which for our system is 16. Our span is 6 because we truncate our signal to 6 symbols for filter application. The signal stream after the low-pass filter has been defined as our baseband signal.

1.2 Channel

We have designed a channel from scratch to test our receiver operation. The channel introduces four types

Table 1: Cost of Receiver Table

Operations	Lowpass Filter	Downsampling Filter	DFT	Time Estimation	Frequency Estimation	Phase Estimation	Demodulation
Multipication	16 x 880 x 97	800 x 16	128 x 128 +128 x 30	80 x 128	130	256	1600
Addition	16 x 880 x 96	800 x 15	128 x 127 + 128 x 29	-	1	117	800
Sine	-	-	-	-	-	256	2400
Cosine	-	-	-	-	-	256	2400
Max	-	-	-	81	1	-	-
Division	-	-	-	-	1	2	-
Square Root	-	-	-	-	4	-	-
Arctan	-	-	-	-	-	1	-

of perturbations. First of all, the channel has an additive white gaussian noise component. Further the channel has a doppler shift and time delay effect. This means we create a random unknown frequency delay to the baseband signal represented as f0 as well as time shift that we express as k0. According to the specification sheet, we were told to incorporate a frequency shift of the range [-1500, 1500] Hertz and a time delay of [-2.5, 2.5]ms in the baseband signal. We know that the sampling frequency given to us is 16000 Hz. Given that the duration is ± 2.5 ms, there is thus an error/uncertainity of

$$\pm 16000 * 2.5 * 10^{-3} = \pm 40$$

samples in the baseband signal. After upsampling we can easily analyse that this uncertainty now becomes ± 40 *symbols*. This means we have to introduce a time shift of exactly ± 40*16 in our baseband signal.

Further using the SNR of the signal, we can very easily model the WGN to be added to the doppler and time delayed baseband signal. We also incorporate a phase delay in the channel that we have initialised as the variable p0 in the MATLAB script. This concludes our channel. We now have a data stream that has been shifted by an unknown frequency f0, an unknown time delay of k0 samples and has an additional phase component p0 along with some WGN noise. The size of the frame has also been increased to 880*16.The channel thus has a behaviour similar to the given input output relation.

$$R(k) = e^{i\phi_0} e^{j2\pi f_0 k/f_s} S(k - k_0) + n(k)$$

1.3 Receiver

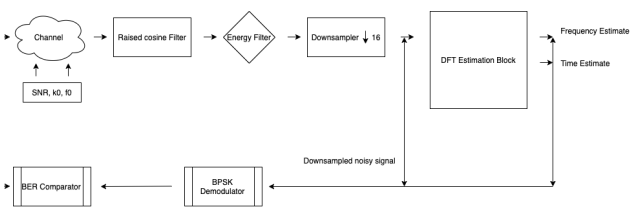


Figure 1: Block Diagram

1.3.1 Matched Filter and Energy Filter for downsampling

We start our receiver with a matched filter which is the same raised cosine filter that we designed on the transmitter side. The convolution of the channel passed signal and the raised cosine filter now needs to be downsampled. However this is not very straightforward.

We introduced redundancy in the signal using upsampling to make noise a non factor. Now we need to *downsample* our frame that we got from the channel and get 880 bits of data back. But the most important question that is still lingering is what is the ideal downsampling time? It needs to be understood that if we start downsampling at the incorrect moment, we will *always get 880 bit frames that have absolutely no data*!. Thus there needs to be some sort of sampling energy filter that identifies what the correct downsampling time is.Note that we are not completely eliminating the time shift using this filter. We will still need to do time estimation on the data frame that we get back.

The sampling filter can be created in multiple ways. We can check the DFT of all the data for each index in the first 16 points. We can then get the index that maximises the energy or correspondingly the DFT values to get the optimal downsampling index. However, similarly we can use the time domain expression of Parseval's relation *(as shown below)* to find the index with the maximum energy in the time domain and this is the algorithm that we have chosen to implement to reduce computational resources and not compute sines and cosines used in a standard DFT block. We will later see, we can in fact do this using a table of DFT coefficients to optimise the algorithm even further since the table call will be based on indexing and no actual computations would be required.

$$\sum_{n=-\infty}^{\infty} |x(n)|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} \left|X(e^{j\omega})\right|^2 d\omega$$

1.3.2 Pilot Estimation

The estimation of a pilot is not a very tough task. We have already applied this technique in the aforementioned sampling time formulation using the energy method. All we need to do to estimate the pilot is to

find the exact index where there is a spike corresponding to our zero tone pilot's frequency response. Since we know that the pilot is 128 bits in length, we can shift a window of size 128 bits and calculate the DFT of the window. The index where the spike occurs corresponds to the pilot and the time delay can be finally resolved. We optimised this algorithm further. Firstly we chose to implement a DFT matrix of our own and not rely on the inbuilt *fft()* function. This enabled us to use the created DFT table to extract the coefficients that we required and just refer to the memory where the coefficients are already stored.

1.3.3 Time and Frequency estimation

After we have found the location of the pilot, we take the DFT of the pilot. We know that the DFT of the pilot signal has a peak at 2kHz when we transmit it (due to the nature of $\pi/4$ BPSK modulation). After the channel, this peak has shifted $\pm$ 1500 Hz. Therefore, we expect a peak in the DFT of the pilot signal in the interval of 500Hz to 3500Hz.Resolution of the 128 point DFT is then expressed as:

$$\frac{2\pi}{128} = 125\,\text{Hz}$$

The optimisation of the indices relies on the following calculations:

$$\frac{2000-1500}{125} = 4 \quad \text{and} \quad \frac{2000+1500}{125} = 28$$

Our range thus corresponds to the 5 to 29th indices of the DFT coefficients. Multiplying with W128trunc gives us these indices without any need to use the fft() function.We then look for peaks from the DFT. The index of the maximum value gives us where the peak occurs. However, in order to get a more accurate solution, we also look at the second peak. If there exists some point larger than half of the peak value, we store this index as well.

If there is exactly one peak, 'the index number-13' multiplied with 125 Hz give us the frequency shift (Because when there is no shift, we expect a peak in the 13th index). On the other hand, if there are more than one peak, we apply the *linear interpolation*. Further, the square root operation give us the magnitude. The linear interpolation algorithm is as follows. Consider there two peaks in the DFT of the signal, and the magnitudes of peaks are m_1 and m_2, and indices are k_1 and k_2, for first and second peak respectively. Then linear interpolation can be written as:

$$k = \frac{m_1 k_1 + m_2 k_2}{m_1 + m_2}$$

After we find, we first decode pilot data. Because we know exactly what is our pilot data is, we can estimate the phase shift from that using a simple inverse tangent operation and then demodulate.

1.3.4 Phase Estimation

After we estimate time and frequency shift, we can find phase shift from the pilot signal. First we can reverse the effect of the frequency and time shifts. Then, we can demodulate pilot signal as $\hat{P}[n]$ by multiplying $\exp(-\frac{\pi}{4}n)$. We have observed that, first few terms of $\hat{P}[n]$ is different from the rest because of the non-ideal lowpass filter. On the other hand, we know that we know that our pilot is contains one bits, and in the constellation diagram it is correspond to complex number $p = (1 + j0)$. Because of the phase shift demodulated version of the pilot signal is different from the p. If we define $\hat{p} = \frac{1}{118}\sum_{k=11}^{128}\hat{P}[n]$, we can estimate the phase shift as

$$\phi = \arctan\left(\frac{\Im\{\hat{p}\}}{\Re\{\hat{p}\}}\right).$$

1.3.5 Demodulation

We can now exactly recover the 800 bits of data that we started with. These 800 bits are however modulated. We can now feed our frame into a π /4 BPSK demodulator to recover the actual data. This demodulator compares the recovered sample with the decision boundary of the constellation diagram of π /4 BPSK to identify whether the bit received is 1 or 0. An easier way of modelling the decision boundary is to map the π /4 demodulation to a simple BPSK demodulation and check the sign of the found symbol. We further map the symbol found to 2 and 1 corresponding to positive and negative data respectively.

BER can then be calculated using a simple comparator and over many iterations the FER can be initiated as a count that increases any time there is even one bit of error in a frame. The iterations are repeated according to the details provided in the project specifications.

2 Experiment Setup

2.1 Major Optimisations

- DFT Block We have implemented the DFT algorithm as a table, thereby getting rid of the need to use the fft() function.
- Index Truncation We have only computed the DFT coefficients that lie in our own range of frequencies i.e signal frequency $\frac{f_s}{8} \pm 1500$ Hz.
- Interpolation We have also used linear interpolation between the maximum and the second maximum peak to generalise the system to work for frequencies that are not integer multiples of the resolution i.e 125 Hz.
- Parseval's Theorem We have exploited Parsseval's relation to bypass the need to compute the frequency interpretation for analysing the best possible downsampling time.

2.2 Setup 1

Specifications:

- SNR = 100 dB *Clean channel*
- F0 = 0 Hz
- TIME OFFSET = -2.5 ms to 2.5 ms, Step = 0.0625 ms
- NUMBER OF SIMULATIONS = 100 per point

2.2.1 Observations

We notice that our system performs ideally under a clean channel. The standard deviation of the estimates is 0 and we get the perfect frequency and time estimates every time.

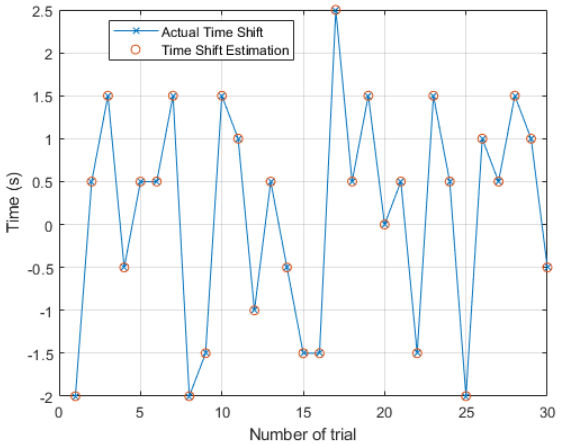


Figure 2: Part 1 Time Estimates and Random time shifts

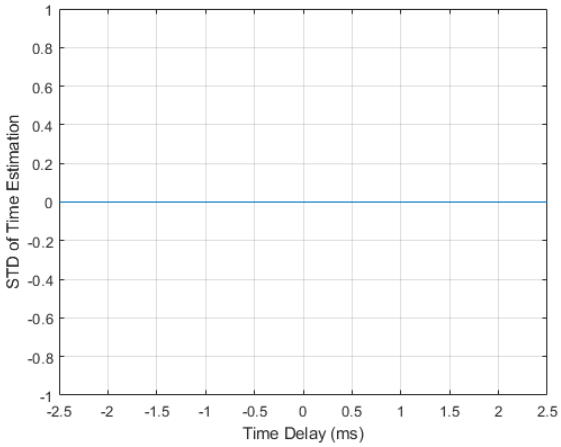


Figure 3: Part 1 Standard Deviation in the time estimate

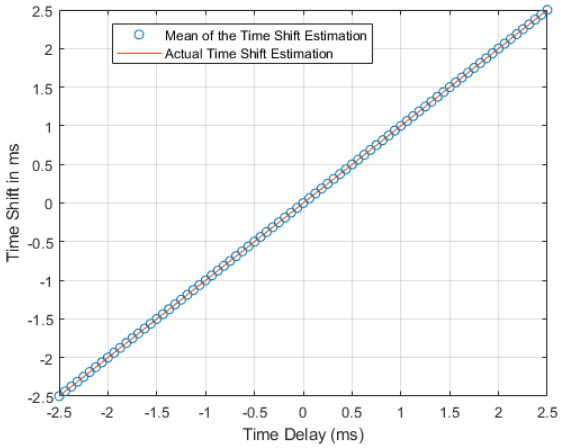


Figure 4: Part 1 Mean in the time estimate

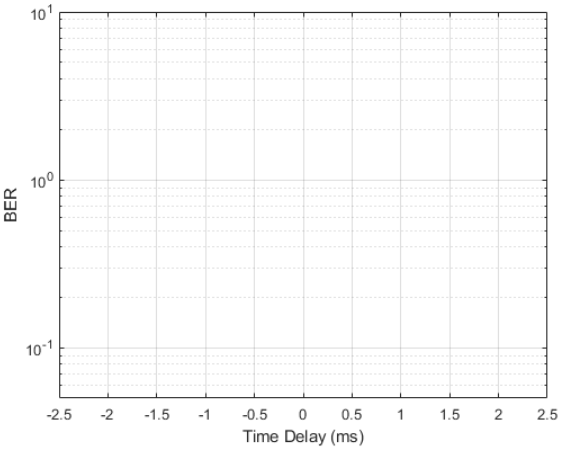


Figure 5: Part 1 BER is 0 for the Clean Channel

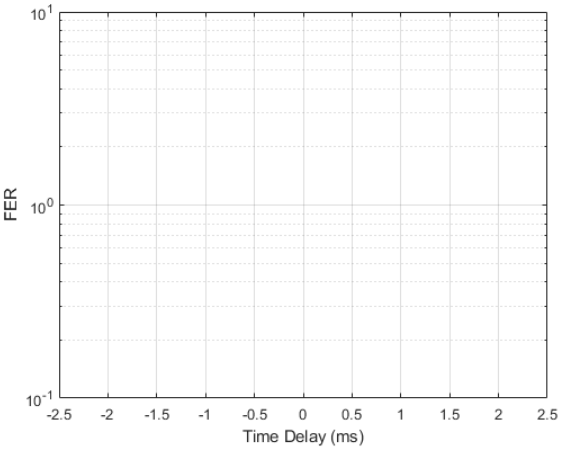


Figure 6: Part 1 FER is 0 for the Clean Channel

2.3 Setup 2

Specifications:

- SNR = 100 dB *Clean channel*
- f0 = -1500 Hz to 1500 Hz, Step= 125 Hz
- Time offset = 0 ms
- Number of simulations = 100 per point

2.3.1 Observations

We notice that our system performs ideally under a clean channel. The standard deviation of the estimates is 0 and we get the perfect frequency estimates every time. Also notice that the BER and FER are 0 for every iteration and this is in line with the choice of our channel which has a very high SNR of 100 db.

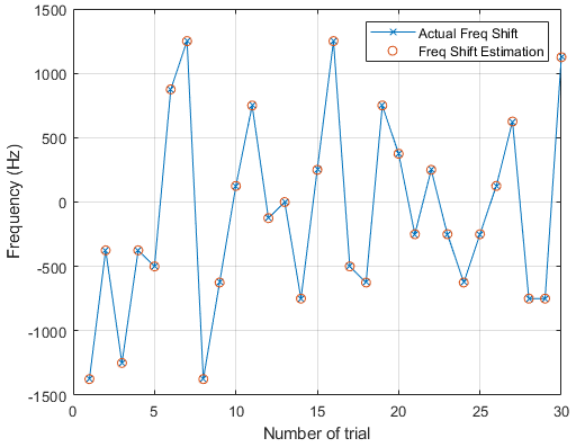


Figure 7: Part 2 Frequency Estimates and Random freq shifts

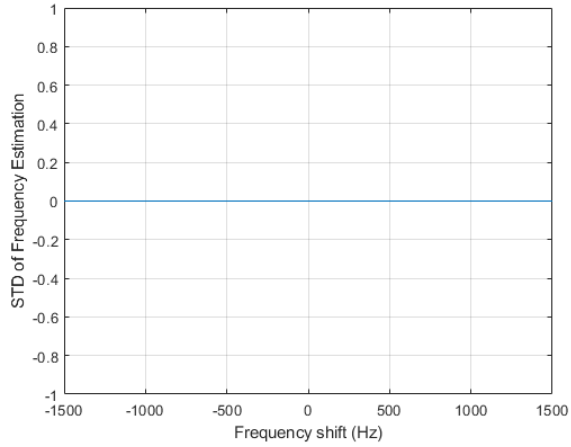


Figure 8: Part 2 Standard Deviation in the frequency estimate

2.4 Setup 3

Setup 3 includes 3 major experiments. In the first experiment, we do not time or frequency shift and we compare our curve to the ideal BPSK curve.

From Figures 17 and 21, we can see that it follows the theoretical bound very closely.

Specifications: (Run our system for given pairs of time and frequency offsets)

- SNR = -3 to 15 dB, Step= 0.5 dB

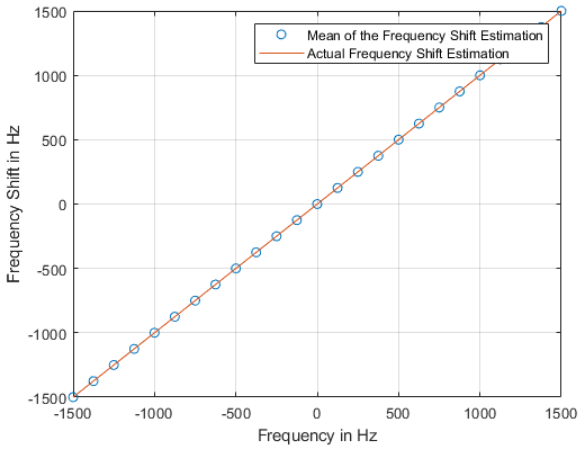


Figure 9: Part 2 Mean in the frequency estimate

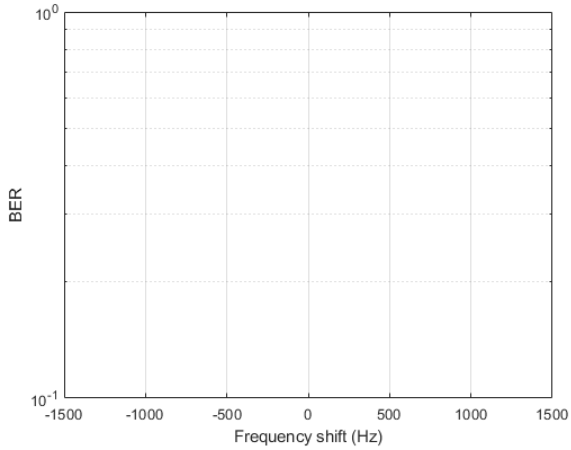


Figure 10: Part 2 BER is 0 for the Clean Channel

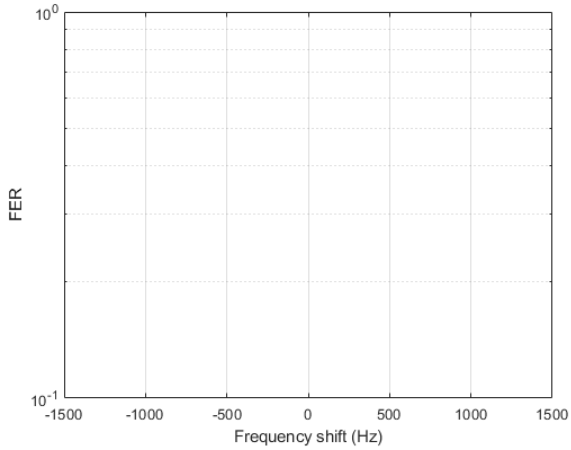


Figure 11: Part 2 FER is 0 Clean Channel: Setup 2

- f0 = 0 Hz, 625 Hz, 62.5 Hz
- Time offset = 0 ms, 0.25 ms, $\frac{2.5}{80}$ ms
- Number of simulations = >1000
- Frame Error Limit = >50

2.4.1 Observations

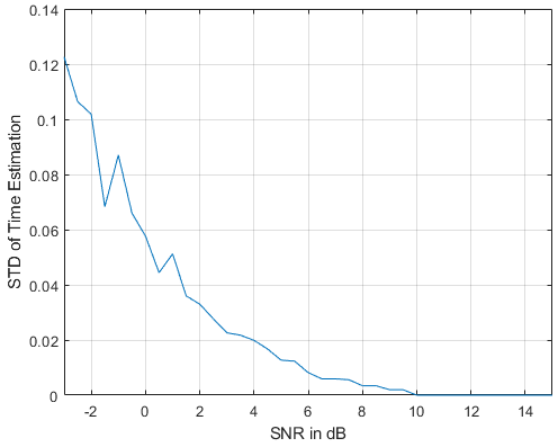


Figure 12: Part 3A Std Deviation of Time estimate

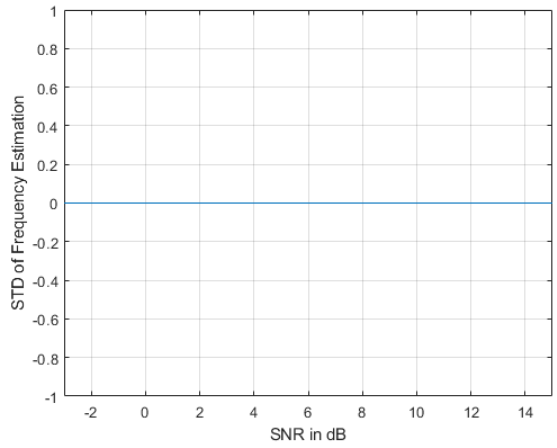


Figure 13: Part 3A Std Deviation of Freq estimate

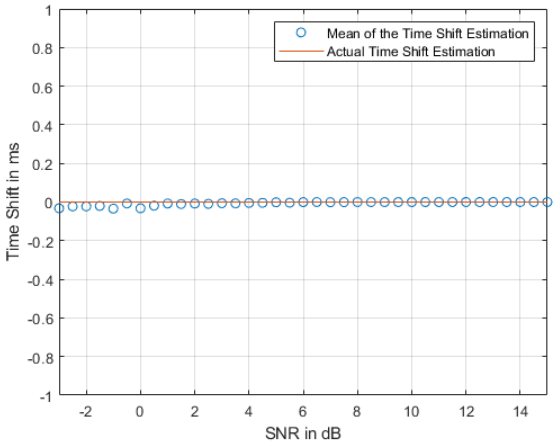


Figure 14: Part 3A Mean Deviation of Time estimate

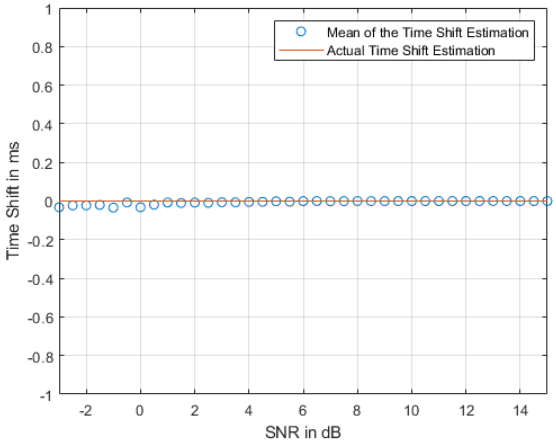


Figure 15: Part 3A Mean Deviation of Freq estimate

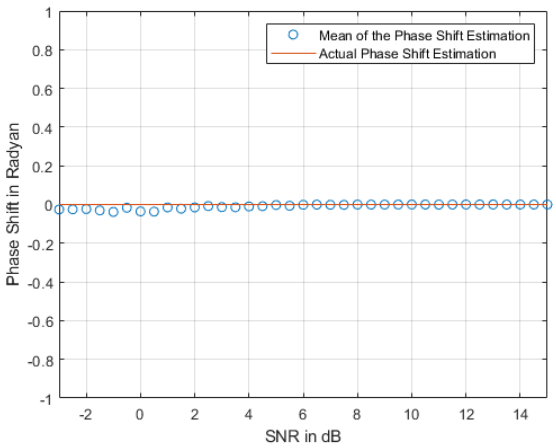


Figure 16: Part 3A Mean Deviation of Phase estimate

3 Key Conclusions

We can conclude that we have created an efficient floating point benchmarking system that accurately estimates both channel induced frequency and time shifts while also getting near ideal BER to SNR curves. We observe no BER and FER and perfect frequency and time estimates in the case of a clean channel. In the case of a noisy channel, SNR's of around 8 db are enough to completely eliminate the uncertainty in the performance of our design and it behaves similar to the clean channel case and the FER is completely eliminated at SNRs of around 10 dB.

The system proposed has also been optimised heavily using both algorithmic knowledge as well as some "tricks" that are common in DSP implementations. As a future work, we are planning to have more accurate and efficient algorithms for especially frequency and phase estimation, better parameter tuning and implement a faster *real time* $\pi/4$ BPSK system that can be put onto an FPGA to reduce computation times in the Part II.

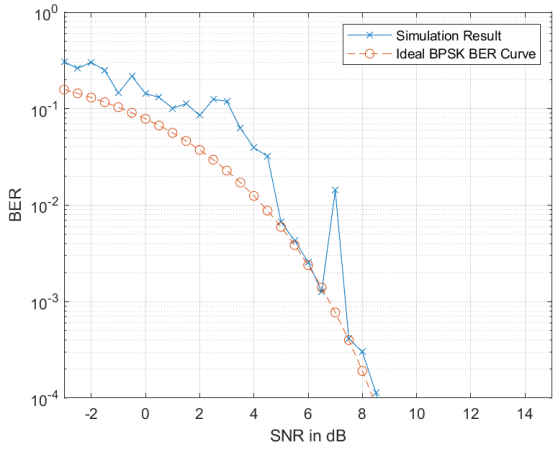


Figure 17: Part 3A BER estimate

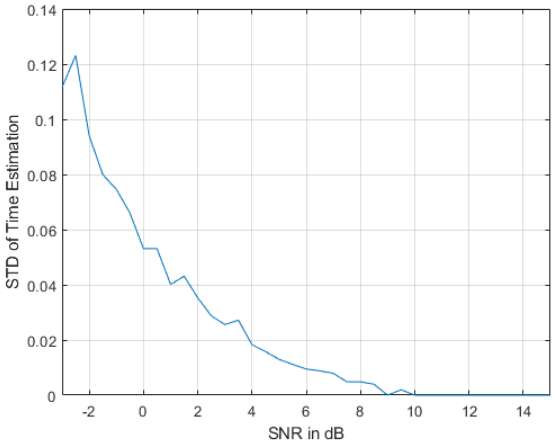


Figure 20: Part 3B Std Deviation of Time estimate

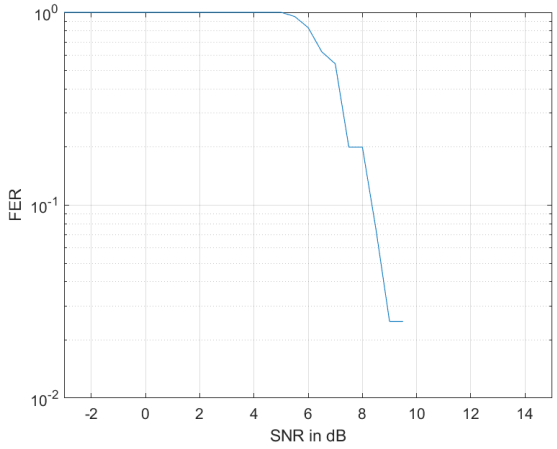


Figure 18: Part 3A FER estimate

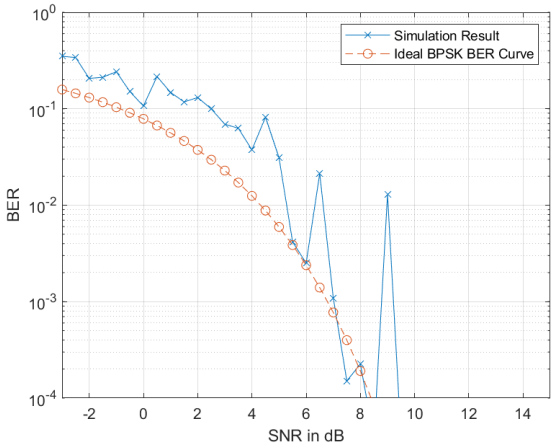


Figure 21: Part 3B BER estimate

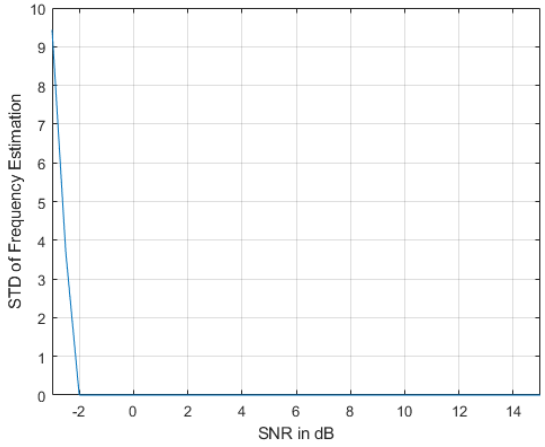


Figure 19: Part 3B Std Deviation of Freq estimate

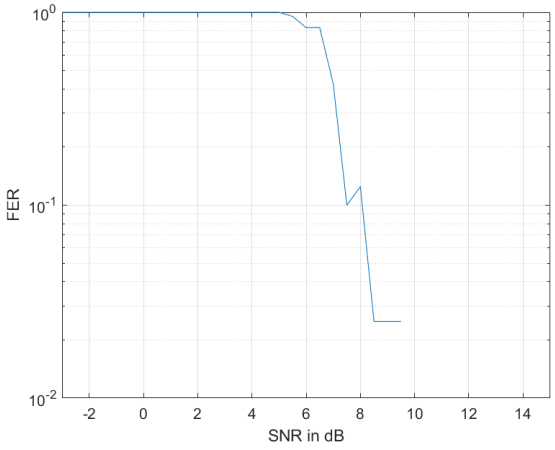


Figure 22: Part 3B FER estimate

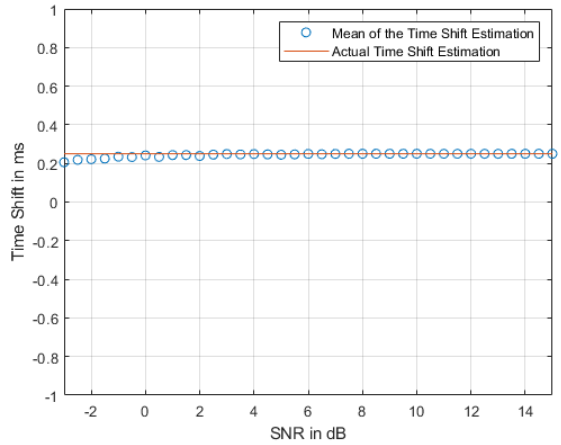


Figure 23: Part 3B Mean Deviation of Time estimate

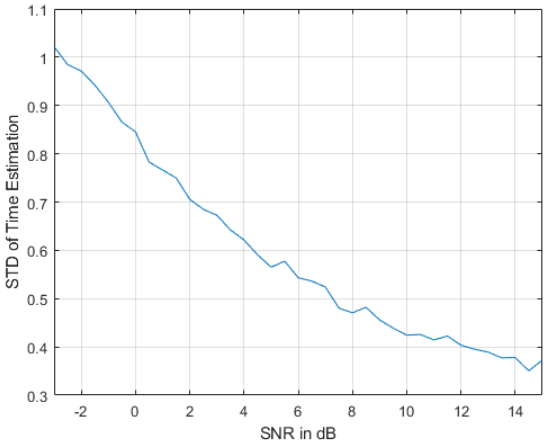


Figure 26: Part 3C Std Deviation of Time estimate

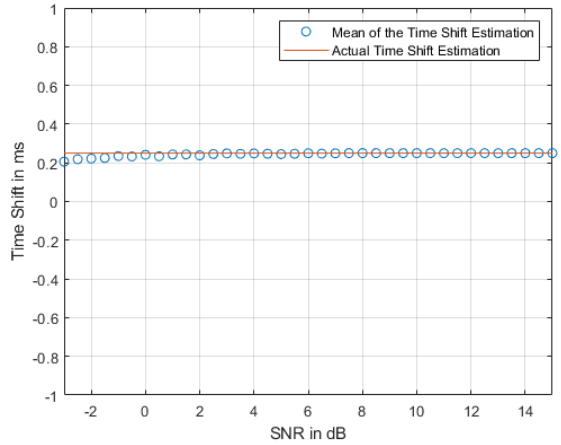


Figure 24: Part 3B Mean Deviation of Freq estimate

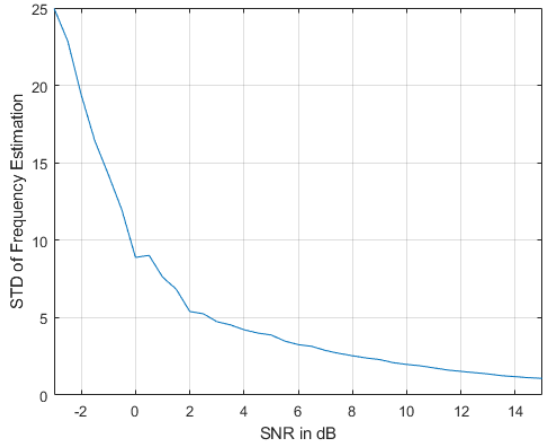


Figure 27: Part 3C Std Deviation of Freq estimate

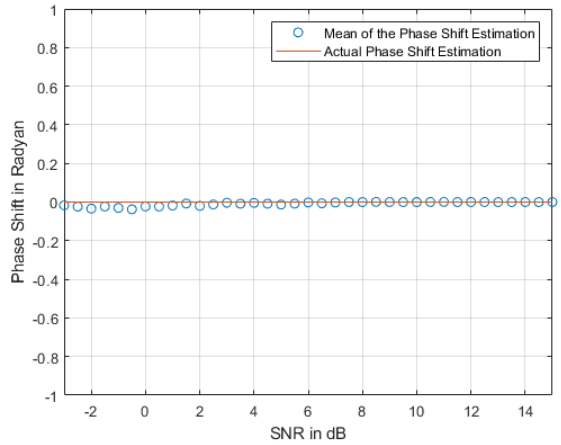


Figure 25: Part 3B Mean Deviation of Phase estimate

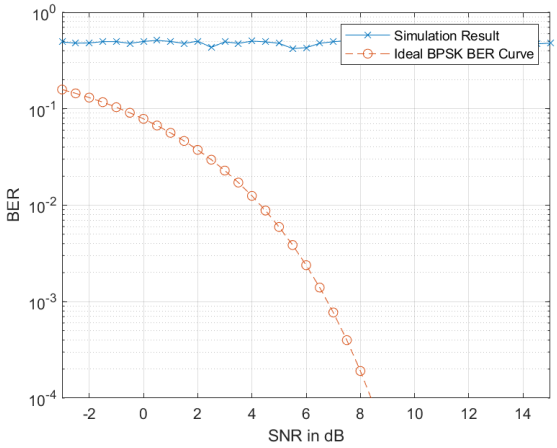


Figure 28: Part 3C BER estimate

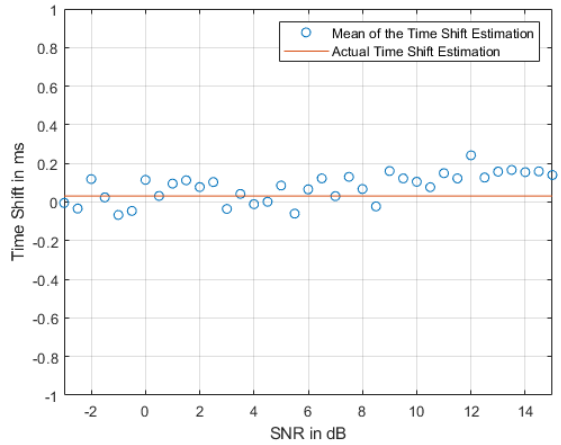


Figure 29: Part 3C Mean Deviation of Time estimate

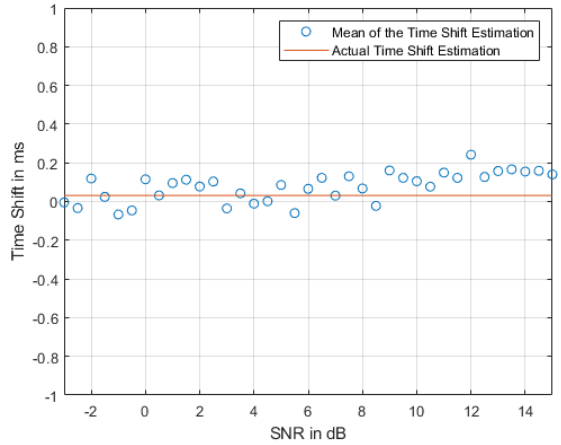


Figure 30: Part 3C Mean Deviation of Freq estimate

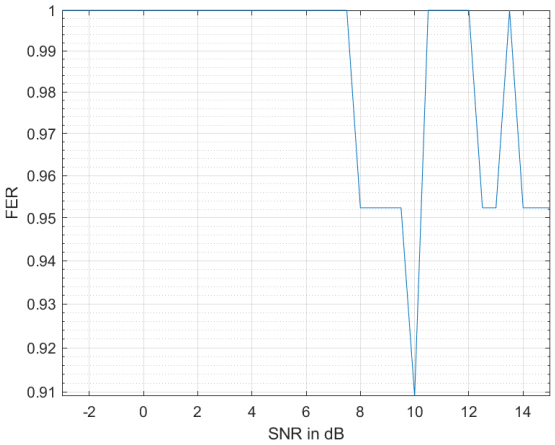


Figure 32: Part 3C FER estimate

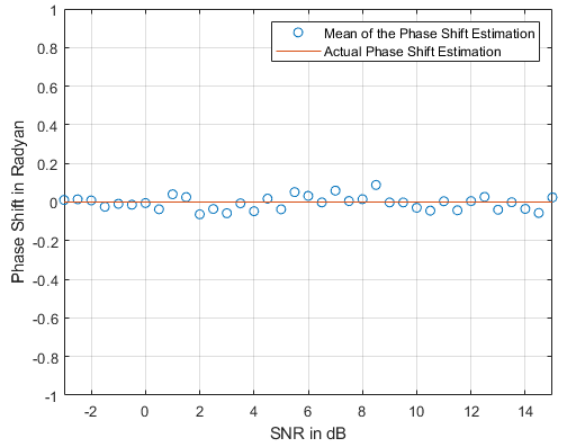


Figure 31: Part 3C Mean Deviation of Phase estimate

Project 1 Part 2

In the second part of the project, we focus on optimising our receiver side even more. Through both algorithmic improvements as well as the fixed point implementation of our receiver, we reduce the computational expense of our system while also maintaining a high level of precision in the working of the system. We also do some additional experiments to estimate the phase noise introduced in the channel.

4 Phase Delay Experiments

In our previous work, we have implemented a phase delay detection. However, in all simulations there is no phase delay. For this part, we have test our algorithm for different phase delay. Similarly, other simulations simulation parameters are chosen following: SNR is 100 dB, time delay is 0 ms, frequency shift is 0 Hz. We have repeated the same simulations at least 100 frames or 50 frame errors. As it can be seen from the Figure 33, for all phase delays our setup can find the phase delay correctly. As a natural consequence of that, we will end up with 0 BER and FER as it can be seen from Figures 34 and 35.

5 Algorithmic Optimisations

In the previous, we have implemented the algorithm discussed in the Section 1.3.3. In this algorithm, we apply an interpolation algorithm by using the certain threshold τ. To summarize *threshold aided linear interpolation*, we first sort the dft coefficients as $m_1, m_2, m_3, \ldots, m_{128}$, and their indices as $k_1, k_2, k_3, \ldots, k_{128}$. Then we choose dft coefficients and their indices ($\mathcal{L}$) larger than $\frac{m_1}{\tau}$. Then we apply interpolation by applying the following formula:

$$\Delta F = 125 \times \frac{\sum_{l \in \mathcal{L}} m_l k_l}{\sum_{l \in \mathcal{L}} m_l} Hz. \quad (1)$$

The main benefit of this algorithm is that frequency shift is exact multiples of 125 Hz, it only uses 1 indices to detect it. This helps to find more accurate solution in these cases.

To improve our estimation, we have add a new interpolation method namely *2 points linear optimization*. With this algorithm, we always use first two peaks to detect frequency shift. In other words, we will detect interpolated index k

$$\Delta F = 125 \times \frac{m_1 k_1 + m_2 k_2}{m_1 + m_2} Hz. \quad (2)$$

With simulation results in Figures 36 and 37, we have compared both algorithms. While *threshold aided linear interpolation* can find frequency delay exactly for multiple of 62.5 Hz, its error increases dramatically on other frequencies. On the other hand, *2 points linear optimization* never find exactly, as a nature consequence of the DFT, its error is not too much for most of frequencies. Any can select the proper algorithm according to system requirement.

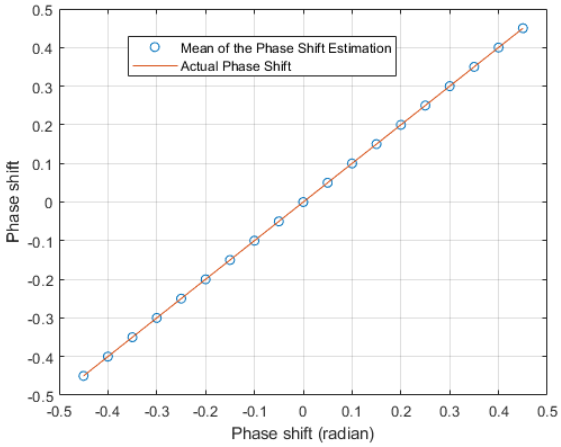


Figure 33: Estimated mean of the phase estimation and actual phase delay when SNR is 100 dB, time delay is 0 ms, frequency shift is 0 Hz.

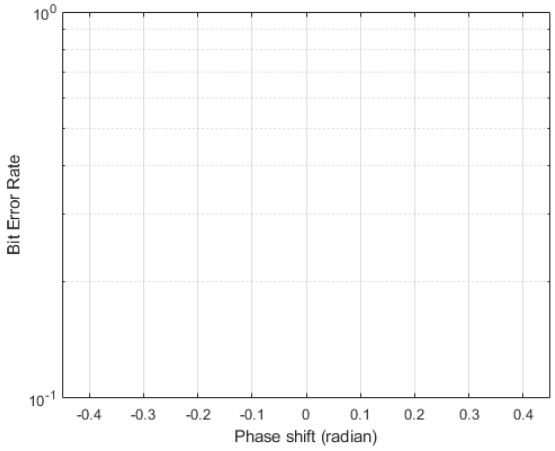


Figure 34: BER vs phase delay when SNR is 100 dB, time delay is 0 ms, frequency shift is 0 Hz.

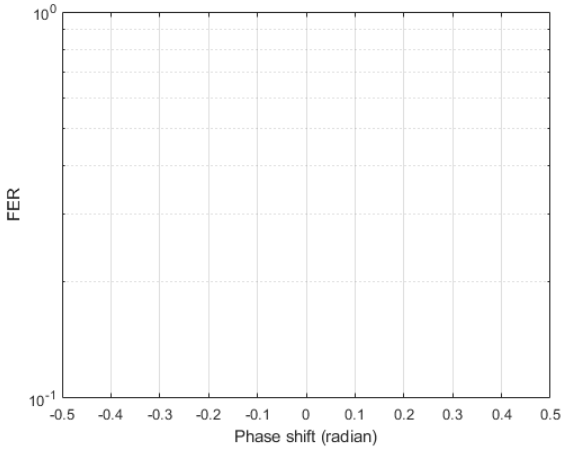


Figure 35: FER vs phase delay when SNR is 100 dB, time delay is 0 ms, frequency shift is 0 Hz.

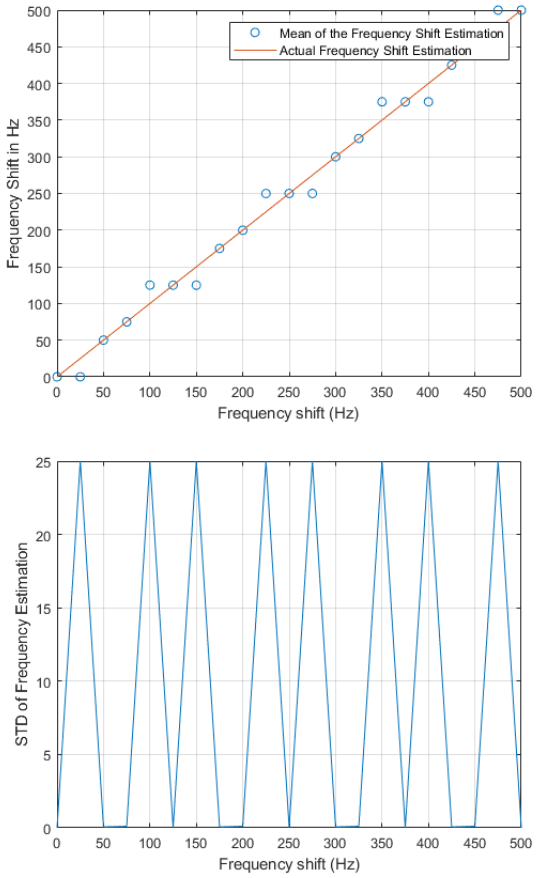


Figure 36: Threshold aided interpolation results a) mean b) standart deviation.

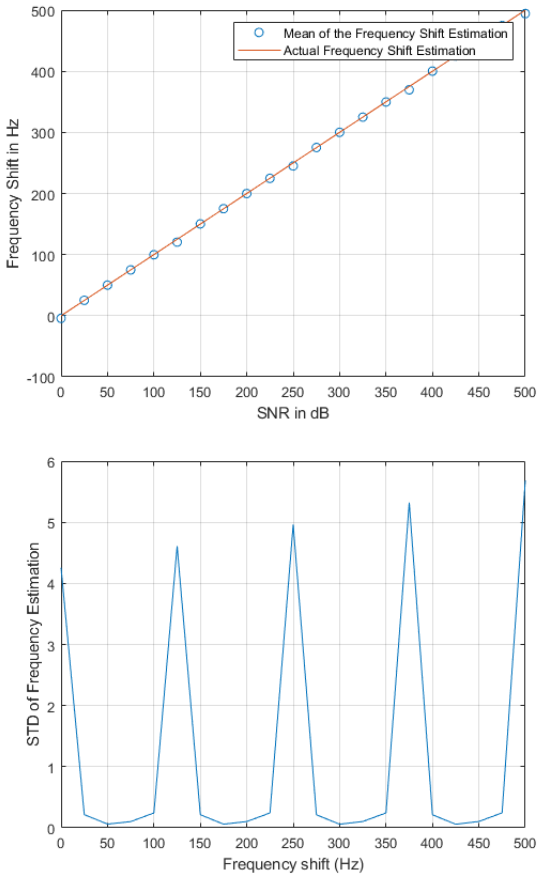


Figure 37: Threshold aided interpolation results a) mean b) standart deviation.

6 Fixed Point Design

6.1 Fixed point Implementation

After all the algorithmic optimisations we made to make the floating point system perform as well as it could, we moved on to making the TX/RX system FPGA ready. This part of the project has our receiver working completely on 16 and 32 bit registers. Everything after the channel of the system has been implemented using fixed point arithmetic and logical operations.

6.2 System Design

6.2.1 Analog to Digital Converter

The ADC is the most important part of the receiver side since it is in essence responsible for correctly transforming our analog received output from the channel into a suitable fixed point expression that can be further manipulated. After the channel, we receive a frame of bit size (80+800)*16 due to the upsampling before the channel and the time delay introduced by the channel. This data now needs to be converted into an appropriate fixed point representation so that the receiver side can be optimised. This is a rather simple calculation and the output of the ADC can be expressed as follows:

$$R_{fixed}(i) = A * R_{float}(i)/MAX_{Norm}$$

where $MAX_{Norm} = max(|R_{float}|^2)$

The choice of the variable A here is a bit tricky though. Although for a 16 bit represenation, each interval should be multiplied by a factor of $2^{15} - 1$ due to the inherent range of a 16 bit integer, this value of the constant was resulting in a lot of overflow issues with our other operations. This is because the values that come out of the channel are quite small and the above choice of A was resulting in them being very close to 2^{15} themselves. Any further operations such as conjugating, adding and so on would result in a nonsensical answer to the overflow of the register. This was therefore a *hyperparamater* that we had to choose carefully so that we were doing our meaningful operations like convolution and DFT carefully.

After struggling with this problem for a while, we found $A = 2^3 - 1$ to be a meaningful truncation that we could work with. While this might seem like a very low number, the values that we found with this parameter for the fixed point were rather disjoint and could be easily told apart. We also tried A to be $2^5 - 1$ but this led to the same issue as the above mentioned problem. A similar approach was made for tuning MAX_{Norm}. A choice of 2 was sufficient to prevent overflow in low SNR cases. The code works perfectly well for the normal expression of MAX_{Norm} but the system then doesnt work very well in SNR regions less than 6 dB.

Since we had already designed a raised cosine filter in Section 1.1.3, we passed the designed parameters through the ADC as well to help us have everything in the desired format on the receiver side for quick operation.

6.2.2 Designing Helper functions

Through the course of the receiver, we designed all of our fixed point functions ourselves. Obviously we designed our own arithmetic operations such as SUM16, MUL16, SUB16 for 16 bit integer operation and SUM32, MUL32, SUB32 for 32 bit operations. However we went ahead and designed even more helper functions that were necessary to implement the project. We designed our own convolution that worked directly on complex fixed point integers, complex integer arithmetic versions of the basic math operators as enumerated before and even MATLAB functions that were not supported for complex arithmetic such as ABS, our very own sum() for compelex ints, CONJ and a really nice fixed point matrix-vector multiplier. However there were also some cool little tricks that we lost when trying to implement the fixed point receiver. These will be discussed in the end after we go through our experiment setups and observations.

6.3 Experiment Setup

6.3.1 Setup 1

Specifications:

- SNR = 100 dB *Clean channel*
- F0 = 0 Hz
- Time offset = -2.5 ms to 2.5 ms, Step = 0.0625 ms
- Number of simulations = 100 per point

6.3.2 Setup 2

Specifications:

- SNR = 100 dB *Clean channel*
- F0 = -1500 Hz to 1500 Hz, Step= 125 Hz
- Time offset = 0 ms
- Number of simulations = 100 per point

6.3.3 Setup 3

Specifications:

- SNR = -3 to 15 dB, Step= 0.5 dB
- F0 = A) 0 Hz, B) 625 Hz, C) 62.5 Hz
- Time offset = A) 0 ms, B) 0.25 ms, C) $\frac{2.5}{80}$ ms
- Number of simulations = >1000
- Frame Error Limit = >50

6.4 Evaluating Performance

6.4.1 Optimisations

- Fixed Point Receiver Every operation on the receiver side is done using 16 and 32 bit int registers with self designed tools and operators.
- Floating Point Phase estimation We have added additional experiments to show how well are our system performs wioth channel induced phase noise.
- Threshold Interpolator We implement and propose a threshold aided linear interpolator that allows more accurate and efficient detection of the frequency shift.
- Efficient DFT with Index truncation We have implemented the DFT algorithm as a table, thereby getting rid of the need to use the fft() function. We have only computed the DFT coefficients that lie in our own range of frequencies i.e signal frequency $\frac{f_s}{8} \pm 1500$ Hz.
- Parseval's Theorem We have exploited Parsseval's relation to bypass the need to compute the frequency interpretation for analysing the best possible downsampling time.
- Fixed point Interpolation Unfortunately due to complexity constraints, we were not able to convert our interpolator into an equivalent fixed point representation. But we have tried to alleviate this issue by implementing an extra interpolator in the floating point system. To summarize our new algorithm, it finds first two peaks of the DFT output. If these peaks are close each other (we control that with shift and comparator), we take average of these indices of two peaks to calculate frequency shift.
- Use of MAX We use the inbuilt max function for efficient estimation of the largest value in an array. This is not a huge setback though because the MAX can be easily replaced by a loop iterating and searching for the maximum fixed point value.

6.4.2 Simulation Results and Discussion

As it can be seen from Figures 38 and 39, our time delay and frequency shift algorithm work as well in the fixed point. Because we can find time delay and frequency shift, and our simulations are done in 100 dB, we get 0 BER and 0 FER.

For Setup 3, our system performs have achieved to reach the previous performances in the most cases. We can estimate time delay and frequency shifts almost exactly in part 3A and 3B, thus we reach very close to the theoretical bounds. The only drawback of this part is that when actual time delay and frequency shift are extreme intermediate values as 3C.

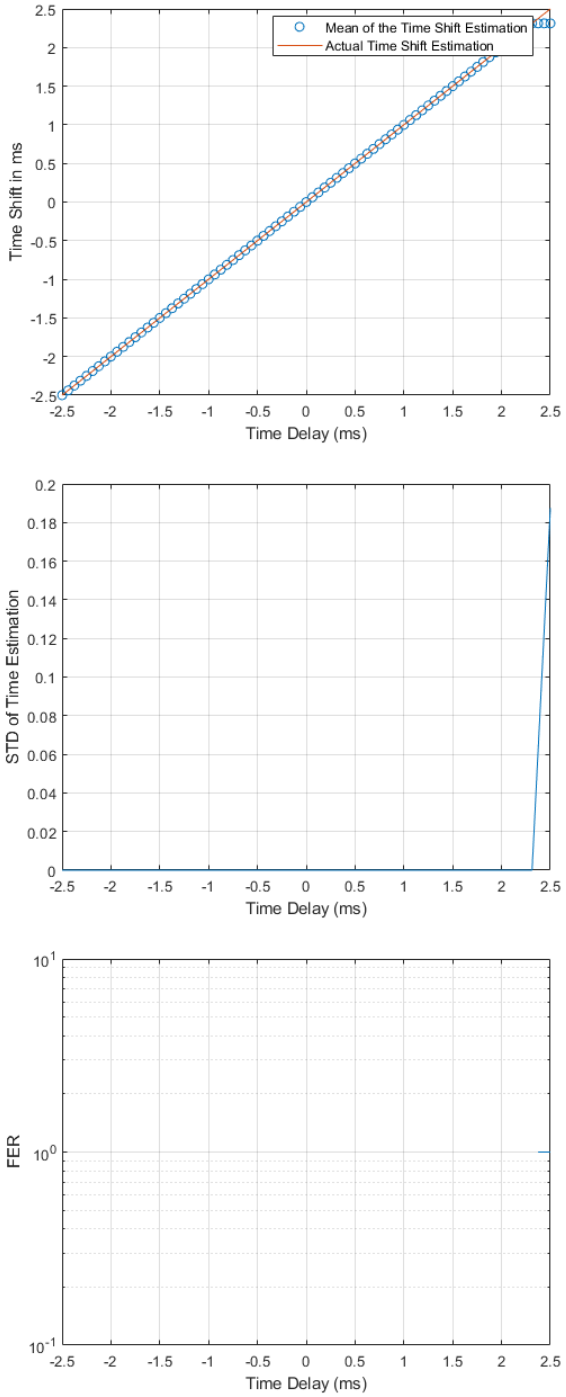


Figure 38: Setup 1 a) Mean of Time Delay Estimation v/s Actual Time Delay a) Mean of Time Delay Estimation v/s Actual Time Delay b) STD of Time Delay Estimation v/s Actual Time Delay c) Frame Error Rate v/s Actual Time Delay

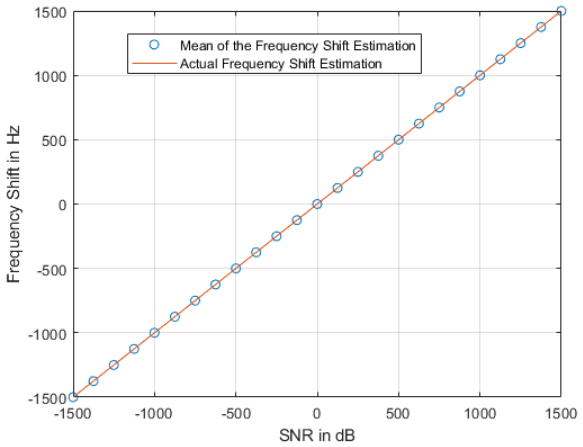
In this setup, we still can estimate time delay and frequency shifts very similar to actual one, this synchronization problem causes performance degradation on BER and FER performances.

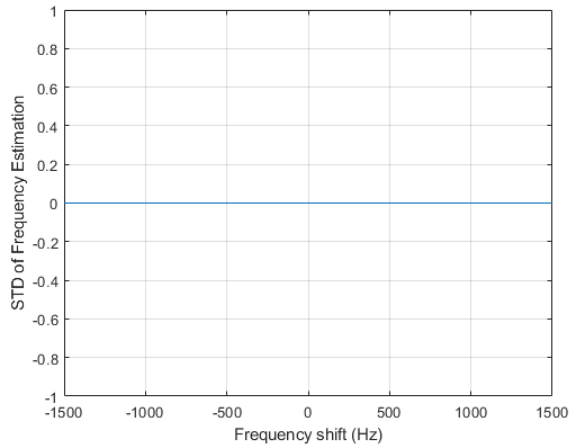
Table 2: Cost of Receiver Table

Operations	Lowpass Filter	Downsampling Filter	DFT	Time Estimation	Frequency Estimation	Demodulation
Cost	97^2 x 2 x 14080	16 x 4 x 880	128 x 128 +128 x 30	2 x 128 x 128^2	1+1+1+2	664 x 2

Table 3: Cost of Helper Functions

Operations	ADD32	MUL16	SHIFT	CONV32	SUMALL32	MATRIXVECMUL
Cost	4	1	1	$2L_2^2L_1$ *where* L_2= *filter length* L= *data length*	$4L$ *where* L= *vector length*	$2L_2^2L_1$ *where* $Matrix = L_2$ x L_1 $Vector = L_1$ x 1





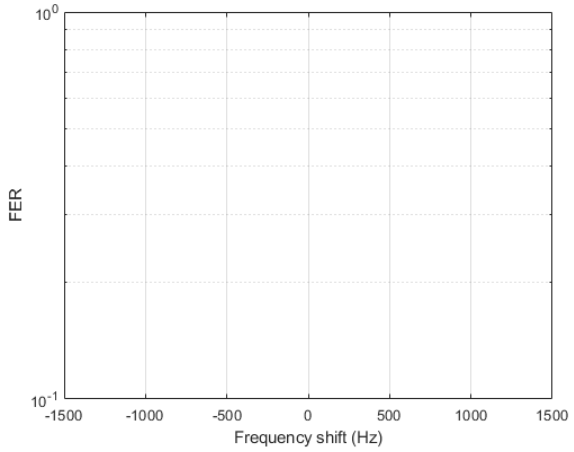


Figure 39: Setup 2 a) Mean of Frequency Shift Estimation v/s Actual Frequency Shift a) Mean of Frequency Shift Estimation v/s Actual Frequency Shift b) STD of Frequency Shift v/s Actual Frequency Shift c) Frame Error Rate v/s Actual Frequency Shift

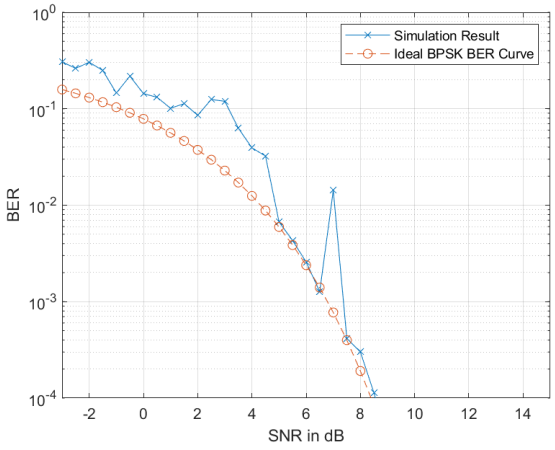


Figure 40: Setup 3A BER v/s SNR

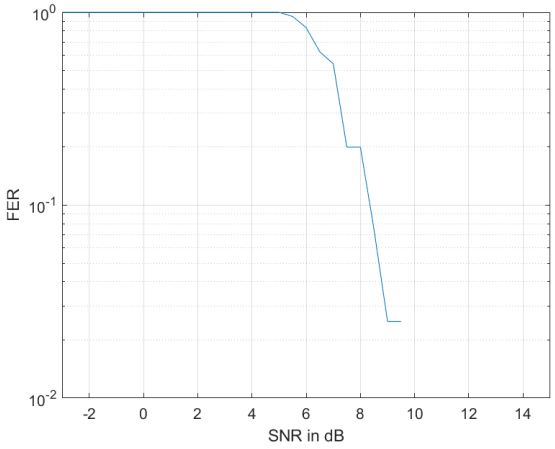


Figure 41: Setup 3A FER v/s SNR

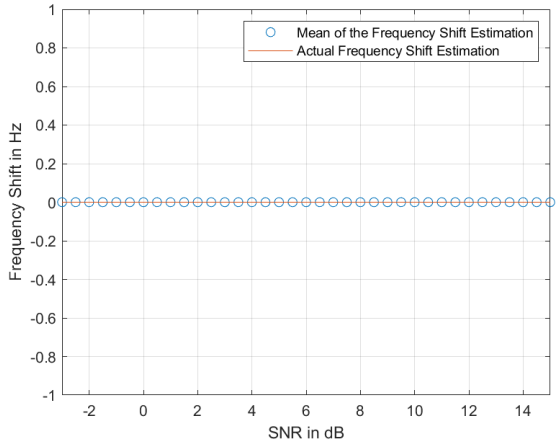


Figure 42: Setup 3A Mean Freq Estimate

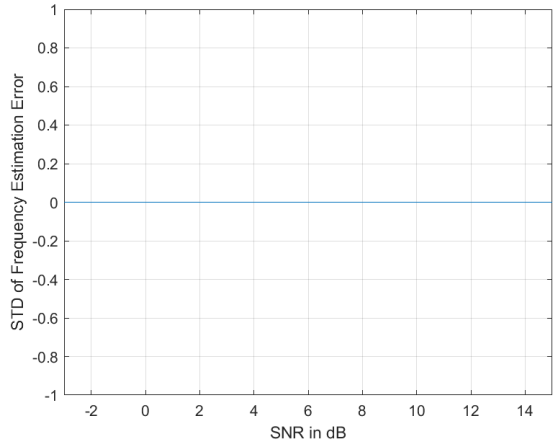


Figure 43: Setup 3A Std Deviation of Freq estimate

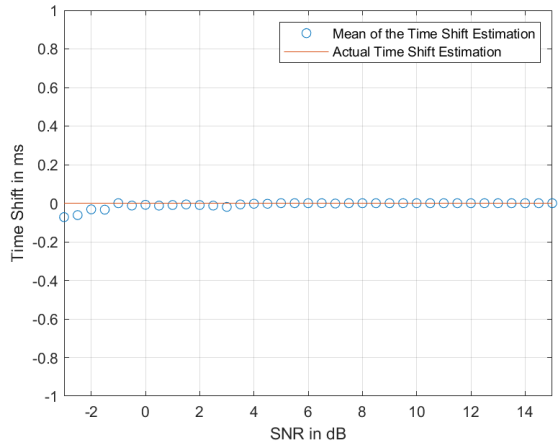


Figure 44: Setup 3A Mean Time Estimate

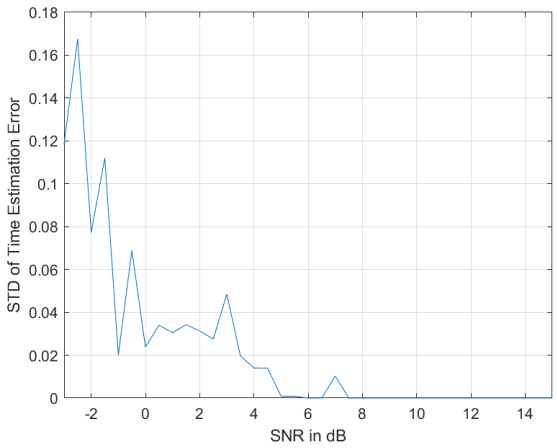


Figure 45: Part 3A Std Deviation of Time estimate

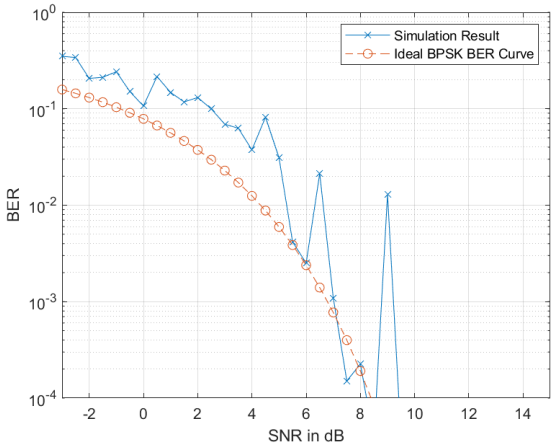


Figure 46: Part 3B BER v/s SNR

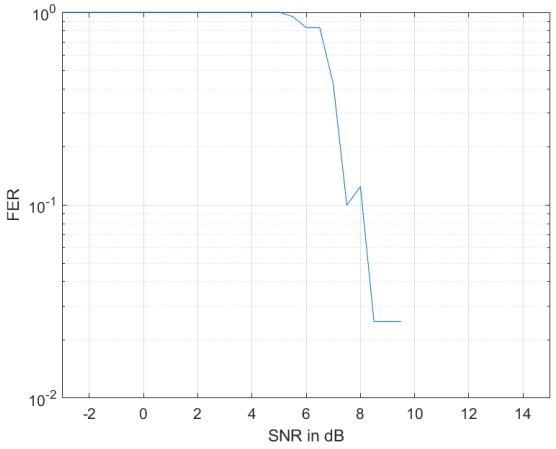


Figure 47: Part 3B FER v/s SNR

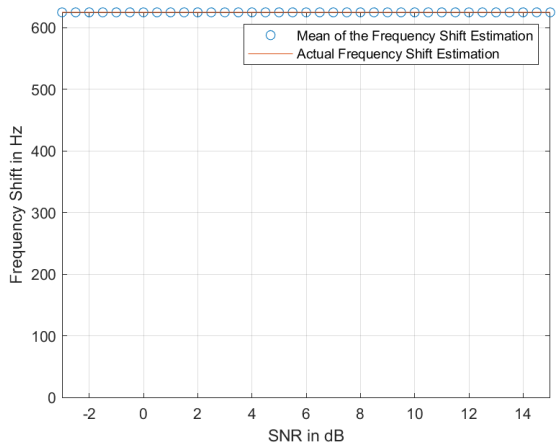


Figure 48: Part 3B Mean Freq Estimate

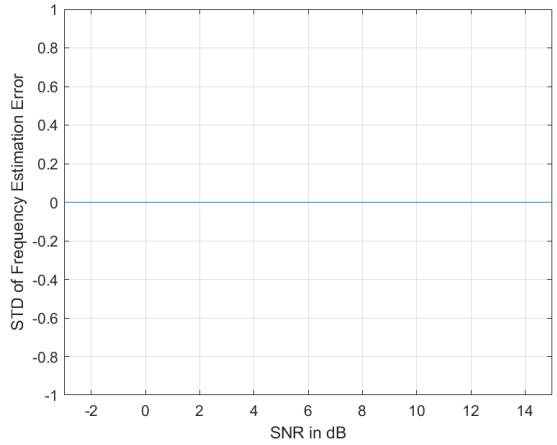


Figure 49: Part 3B Std Deviation of Freq estimate

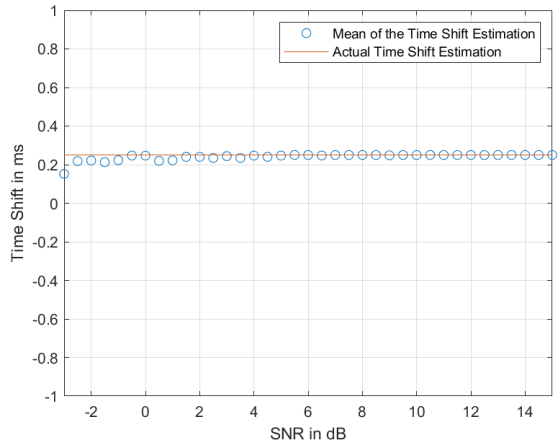


Figure 50: Part 3B Mean Time Estimate

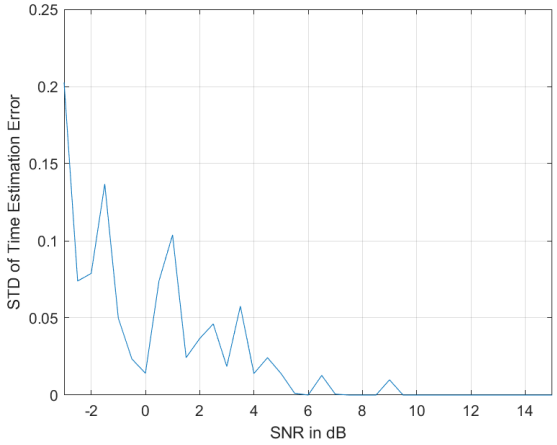


Figure 51: Part 3B Std Deviation of Time estimate

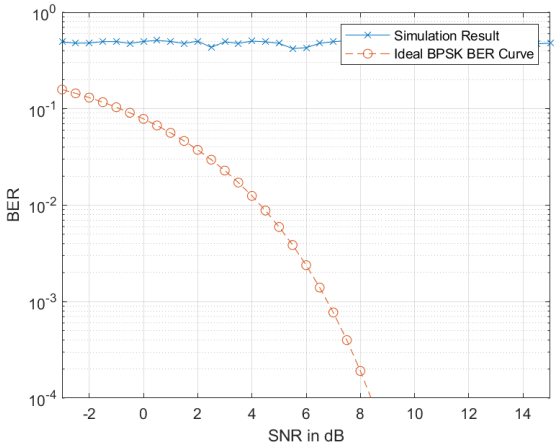


Figure 52: Part 3C BER v/s SNR

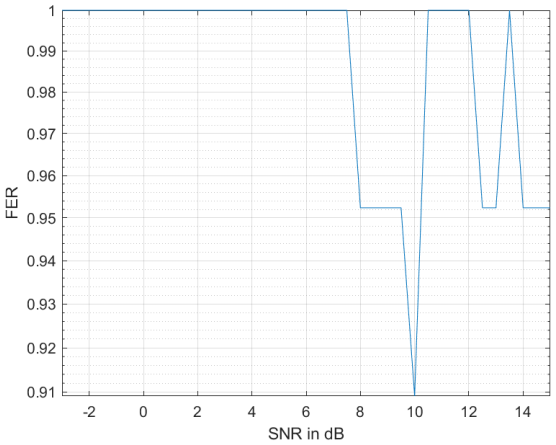


Figure 53: Part 3C FER v/s SNR

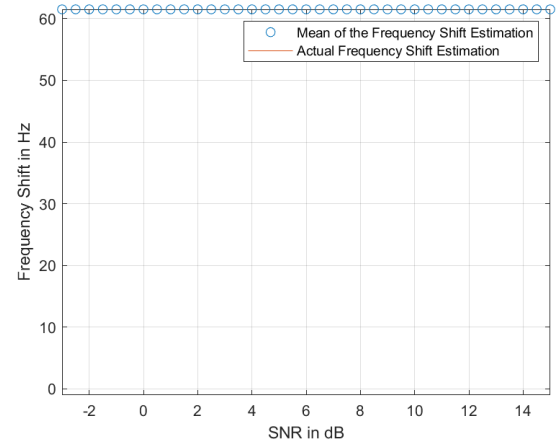


Figure 54: Part 3C Mean Freq Estimate

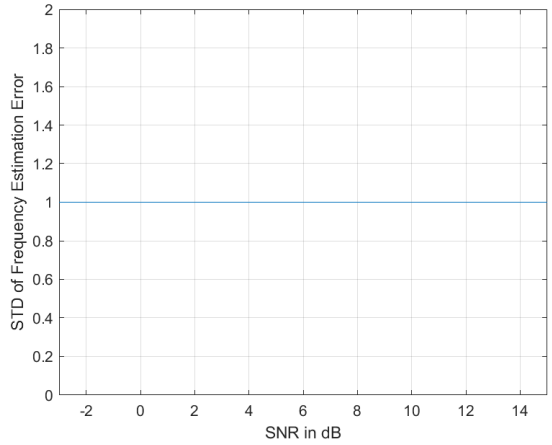


Figure 55: Part 3C Std Deviation of Freq estimate

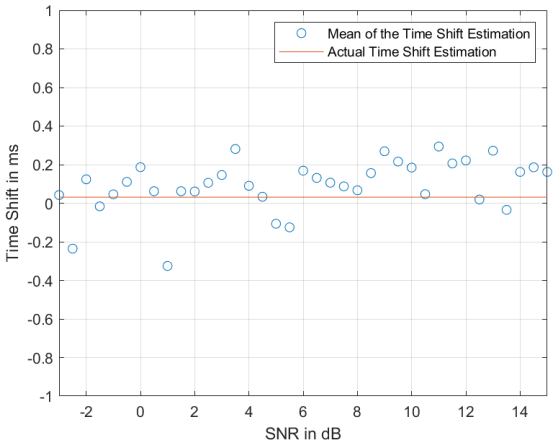


Figure 56: Part 3C Mean Time Estimate

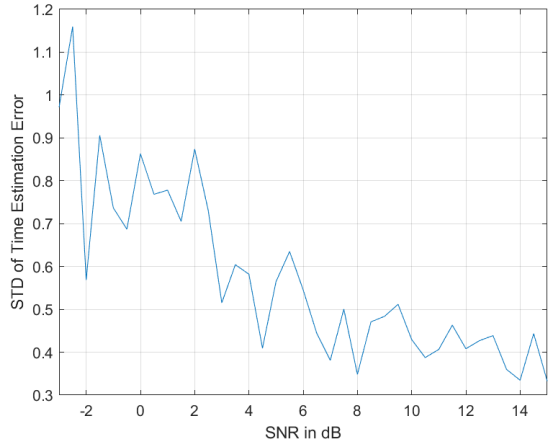


Figure 57: Part 3C Std Deviation of Time estimate