GPU Implementation of Data-Aided Equalizers

Jeffrey T. Ravert

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Michael D. Rice, Chair
Brian D. Jeffs
Brian A. Mazzeo

Department of Electrical and Computer Engineering

Brigham Young University

April 2017

ABSTRACT

GPU Implementation of Data-Aided Equalizers

Jeffrey T. Ravert

Department of Electrical and Computer Engineering

Master of Science

Multipath is one of the dominant causes for link loss in aeronautical telemetry. Equalizers have been studied to combat multipath interference in aeronautical telemetry. Blind Constant Modulus Algorithm (CMA) equalizers are currently being used on SOQPSK-TG. The Preamble Assisted Equalization (PAQ) has been funded by the Air Force to study data-aided equalizers on SOQPSK-TG. PAQ compares side by side no equalization, data-aided zero forcing equalization, data-aided MMSE equalization, data-aided initialized CMA equalization, data-aided frequency domain equalization, and blind CMA equalization. An real time experimental test setup has been assembled including an RF receiver for data acquisition, FPGA for hardware interfacing and buffering, GPUs for signal processing, spectrum analyzer for viewing multipath events, and an 8 channel bit error rate tester to compare equalization performance. Lab tests were done with channel and noise emulators. Flight tests were conducted in March 2016 and June 2016 at Edwards Air Force Base to test the equalizers on live signals. The test setup achieved a 10Mbps throughput with a 6 second delay. Counter intuitive to the simulation results, the flight tests at Edwards AFB in March and June showed blind equalization is superior to data-aided equalization. Lab tests revealed some types of multipath caused timing loops in the RF receiver to produce garbage samples. Data-aided equalizers based on data-aided channel estimation leads to high bit error rates. A new experimental setup is been proposed, replacing the RF receiver with a RF data acquisition card. The data acquisition card will always provide good samples because the card has no timing loops, regardless of severe multipath.

Keywords: MISSING

ACKNOWLEDGMENTS

Students may use the acknowledgments page to express appreciation for the committee members, friends, or family who provided assistance in research, writing, or technical aspects of the dissertation, thesis, or selected project. Acknowledgments should be simple and in good taste.

Table of Contents

List of Tables

List of Figures

Chapter 1

System Overview

1.1 Overview

This thesis gives and overview of the the GPU implementation and performance of data aided equalizes. but the focus of this thesis is on the computation and application of the equalizers. Before data-aided equalizers can be computed and applied: the preambles must found, the signal packetized, the signal ”de-rotated” then the channel and noise variance estimated from the de-rotated signal. The equalizers are then computed and applied using the de-rotated signal, estimated channel and noise variance. After the equalizers have been applied a OQPSK detector is applied to the output of each equalizer. A simple block Diagram is shown in Figure 1.1.

This chapter will proceed as follows, section 1.1.1 will explain the GPU implementation of the finding the preambles and packetizing the received signal, section 1.1.2 will explain the GPU implementation of estimating the frequency offset and de-rotating the signal, section 1.1.3 will explain the GPU implementation of estimating the channel, section 1.1.4 will explain the GPU implementation of estimating the noise variance, section 1.1.5 will explain the GPU implementation of the OQPSK detector. The explanation of the GPU implantation of the equalizers will be explained in much detail in Chapter 2.

1.1.1 Preamble Detection

The received samples in this project has the iNET packet structure shown in Figure 1.2. The iNET packet consists of a preamble and ASM periodically inserted into the data stream. The iNET preamble and ASM bits are inserted every 6144 data bits. The received signal is sampled at 2 samples/bit, making a iNET packet $\mathbf{L}_{\text{pkt}}$ long or 12672 samples. The iNET preamble comprises

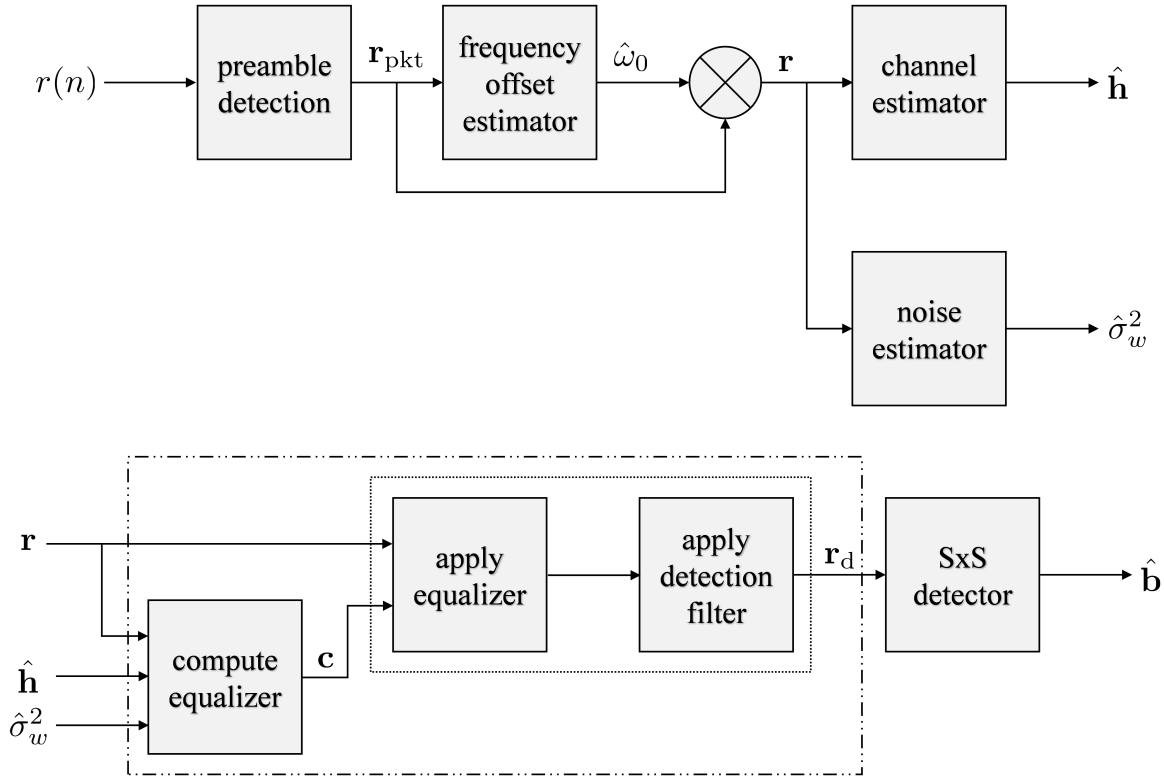

Figure 1.1: This a simple block diagram of what the GPU does.

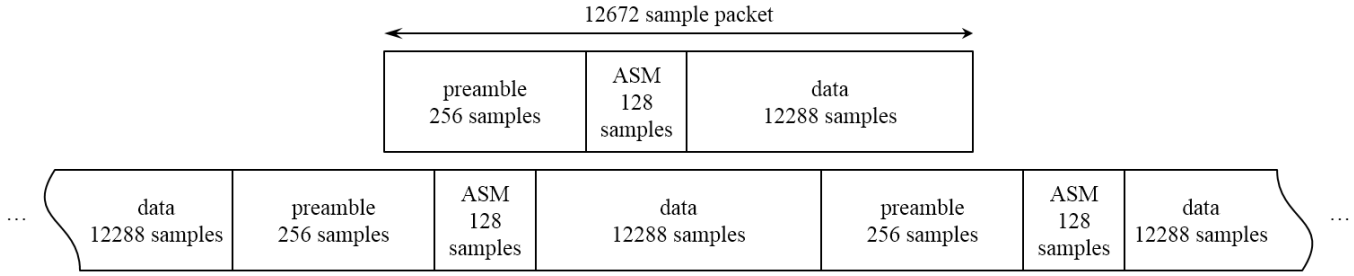

Figure 1.2: The iNET packet structure.

eight repetitions of the 16-bit sequence CD98_{hex} and the ASM field

$$034776C72728950B0_{\text{hex}} \tag{1.1}$$

The iNET packet and received sample structure is shown in Figure1.2.

To compute data-aided preamble assisted equalizers, preambles in the received signal are found to estimate various parameters. The goal of the preamble detection step is to "packetize" the received samples into vectors with the packet structure shown in Figure 1.2. Before the received samples can be packetized, the preambles are found using a preamble detector to locate the iNET preambles. The preamble detector used in this project is the algorithm explained in [1]. Peaks in the preamble detector output indicate the start a preamble. Using the index of these peaks the received samples can be packetized.

When the received samples are packetized, $\mathbf{r}_{\text{pkt}}$ shown in Figure 1.1 contains one packet worth of samples with the packet structure shown in Figure 1.2. The received samples are packetized by running the preamble detector and searching the output for peaks. Starting at each peak index a vector $\mathbf{r}_{\text{pkt}}$ is built for each packet.

The Preamble Detector

To find the preambles in the batch, the preamble detector computes the sample correlation function between the received samples and a stored local copy of the known samples of the iNET preamble. A peak in the correlation function indicates the start of a preamble in the received samples. The preamble detector output peaks are found be searching for local peaks that are about $\mathbf{L}_{\text{pkt}}$ samples apart.

The preamble detector detailed in [1] is a highly optimized correlator used to compute to how a signal correlates with the iNET preamble eight repetitions of the 16-bit sequence CD98_{hex}. The preamble detector outer summation is

$$L(u) = \sum_{m=0}^{7} \left[I^2(n,m) + Q^2(n,m) \right] \tag{1.2}$$

where the inner summations are

$$I(n,m) \approx \sum_{\ell \in \mathcal{L}_1} r_R(\ell+32m+n) - \sum_{\ell \in \mathcal{L}_2} r_R(\ell+32m+n) + \sum_{\ell \in \mathcal{L}_3} r_I(\ell+32m+n) - \sum_{\ell \in \mathcal{L}_4} r_I(\ell+32m+n)$$
$$+ 0.7071 \left[\sum_{\ell \in \mathcal{L}_5} r_R(\ell+32m+n) - \sum_{\ell \in \mathcal{L}_6} r_R(\ell+32m+n) \right.$$

$$+\sum_{\ell\in\mathcal{L}_7} r_I(\ell+32m+n)-\sum_{\ell\in\mathcal{L}_8} r_I(\ell+32m+n)\Bigg], \quad (1.3)$$

and

$$\begin{aligned} Q(n,m) \approx & \sum_{\ell\in\mathcal{L}_1} r_I(\ell+32m+n)-\sum_{\ell\in\mathcal{L}_2} r_I(\ell+32m+n) \\ & -\sum_{\ell\in\mathcal{L}_3} r_R(\ell+32m+n)+\sum_{\ell\in\mathcal{L}_4} r_R(\ell+32m+n) \\ & +0.7071\Bigg[\sum_{\ell\in\mathcal{L}_5} r_I(\ell+32m+n)-\sum_{\ell\in\mathcal{L}_6} r_I(\ell+32m+n) \\ & -\sum_{\ell\in\mathcal{L}_7} r_R(\ell+32m+n)+\sum_{\ell\in\mathcal{L}_8} r_R(\ell+32m+n)\Bigg] \end{aligned} \quad (1.4)$$

with

$$\begin{aligned} \mathcal{L}_1 &= \{0,8,16,24\} \\ \mathcal{L}_2 &= \{4,20\} \\ \mathcal{L}_3 &= \{2,10,14,22\} \\ \mathcal{L}_4 &= \{6,18,26,30\} \\ \mathcal{L}_5 &= \{1,7,9,15,17,23,25,31\} \\ \mathcal{L}_6 &= \{3,5,11,12,13,19,21,27,28,29\} \\ \mathcal{L}_7 &= \{1,3,9,11,12,13,15,21,23\} \\ \mathcal{L}_8 &= \{5,7,17,19,25,27,28,29,31\}. \end{aligned} \quad (1.5)$$

The preamble detector in Equations (1.2)-(1.5) are easily implemented into a GPU. Two GPU kernels compute the preamble detector output . The first kernel computes the inner summations and the second computes the outer summation. In both kernels, one thread to received signal is launched.

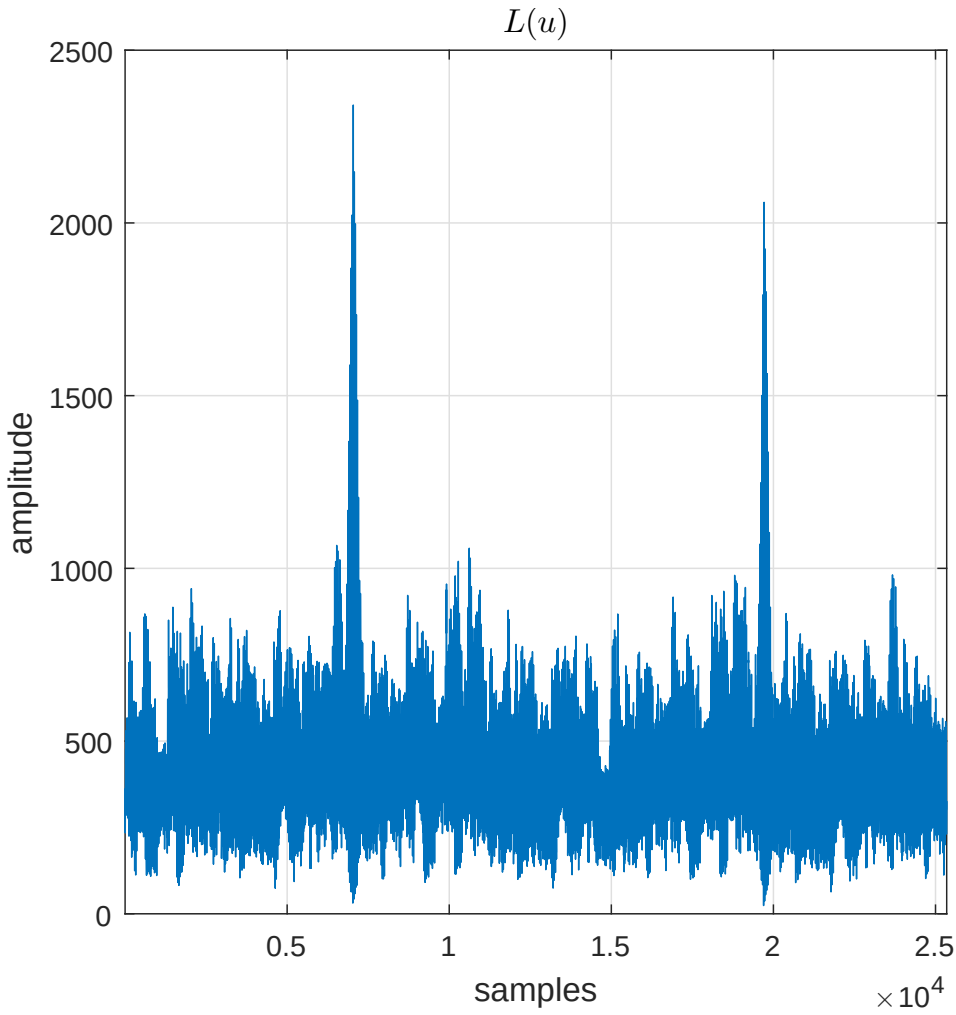

Figure 1.3: The output of the Preamble Detector $L(u)$.

Searching for the Starting Indices

Equipped with the preamble detector output, $L(u)$ is searched for local maximums or peaks to find the starting index for each preamble. Figure 1.3 shows $2 \times L_{pkt}$ samples of $L(u)$. The local peaks of $L(u)$ in indicate a preamble starts at the peak's sample index. A packet starts at sample index 7040 and 19712. The difference between these sample indices is $\mathbf{L}_{\text{pkt}}$, this difference agrees with the packet length shown in Figure 1.2.

Because of the repetitive structure of the preamble, the preamble detector output $L(u)$ has some unique properties. The Figures in 1.4 show the correlation function around a correlation peak. The correlation functions have peaks every 32 samples because the repeated CD98_{hex} pattern.

With ideal received samples, 512 samples of $L(u)$ looks like Figure 1.4(a). The structure of the correlation peaks still occur when the signal to noise ratio is low, as shown in Figure 1.4(b). But when the signal to noise ratio is low and major multipath distortion happen, the correlation peaks look like Figures 1.4(c) and (d). The structure of the correlation peaks can cause a local maximum 32 samples off of the start of a preamble.

The preamble detector output is searched for the peak indices using $\mathbf{L}_{\text{pkt}}$ sample windows to ensure only one correlation peak is searched. The peak indices are checked to ensure each peak index is $\mathbf{L}_{\text{pkt}}$ away from adjacent peak indices. One GPU kernel searches $L(u)$ for peaks to find the the starting index of each packet. A thread per $\mathbf{L}_{\text{pkt}}$ received samples is launched to search a window for a local maximum or correlation peak.

In the worst case scenario, a simple algorithm might find an incorrect preamble index by searching a poorly placed search window. A poorly placed window might search the large side correlation peaks from Figure 1.4(a) and the small main correlation peaks from Figures 1.4(c) or (d). Because the first three side peaks in Figure 1.4(a) are much taller than the main peak in Figures 1.4(c) and (d), an incorrect preamble starting indices will result if search windows are not defined safely.

To prevent searching multiple sets of preamble correlation peaks, search windows should only search the correlation peak of one preamble. Windows are safely defined by centering them on an initial search is done on $2\mathbf{L}_{\text{pkt}}$ received samples to find the estimated index of the first preamble $\hat{i}_0$. Figure 1.5 shows an example of safe search windows centered on expected preamble starting locations. The vector

$$\hat{\mathbf{i}} = \begin{bmatrix} \hat{i}_0 \\ \hat{i}_1 \\ \vdots \\ \hat{i}_{3103} \\ \hat{i}_{3104} \end{bmatrix} \tag{1.6}$$

is built by the GPU search kernel. The vector $\hat{\mathbf{i}}$ is a vector of rough estimates of where the preambles should be located. The windows in the GPU search kernel are centered on elements of $\hat{\mathbf{i}}$ as shown in Figure 1.5. Adjustments are made to the peak index vector $\hat{\mathbf{i}}$ is checked for $\mathbf{L}_{\text{pkt}}$ spacing.

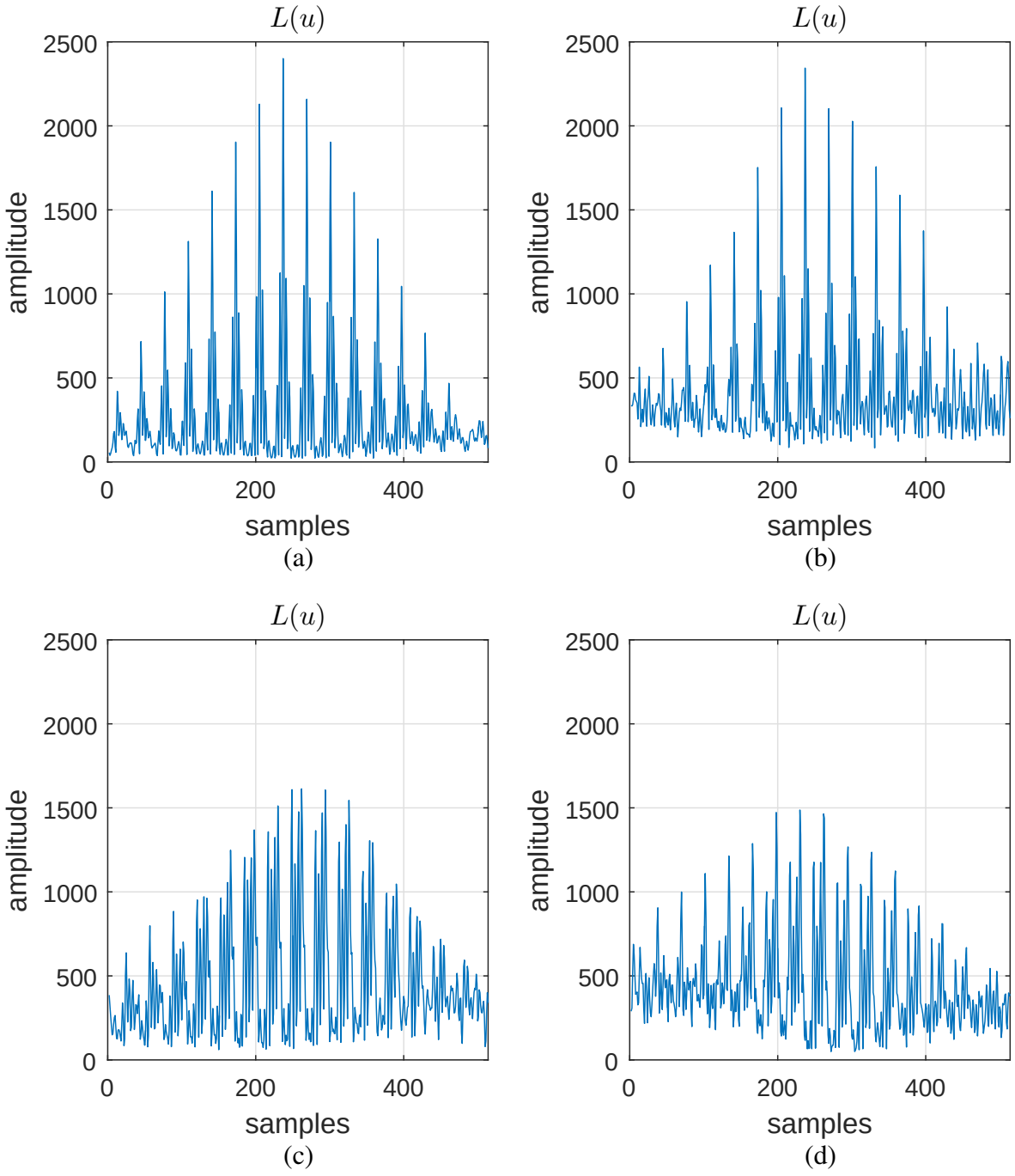

Figure 1.4: Detailed view of $L(u)$. (a): correlation peaks of a distortion free and noiseless signal; (b): correlation peaks of a distortion free but noisy signal with $E_b/N_0 = 0$dB; (c): correlation peaks of a distorted and noisy signal with $E_b/N_0 = 0$dB; (d): correlation peaks of a distorted and noisy signal with $E_b/N_0 = 0$dB

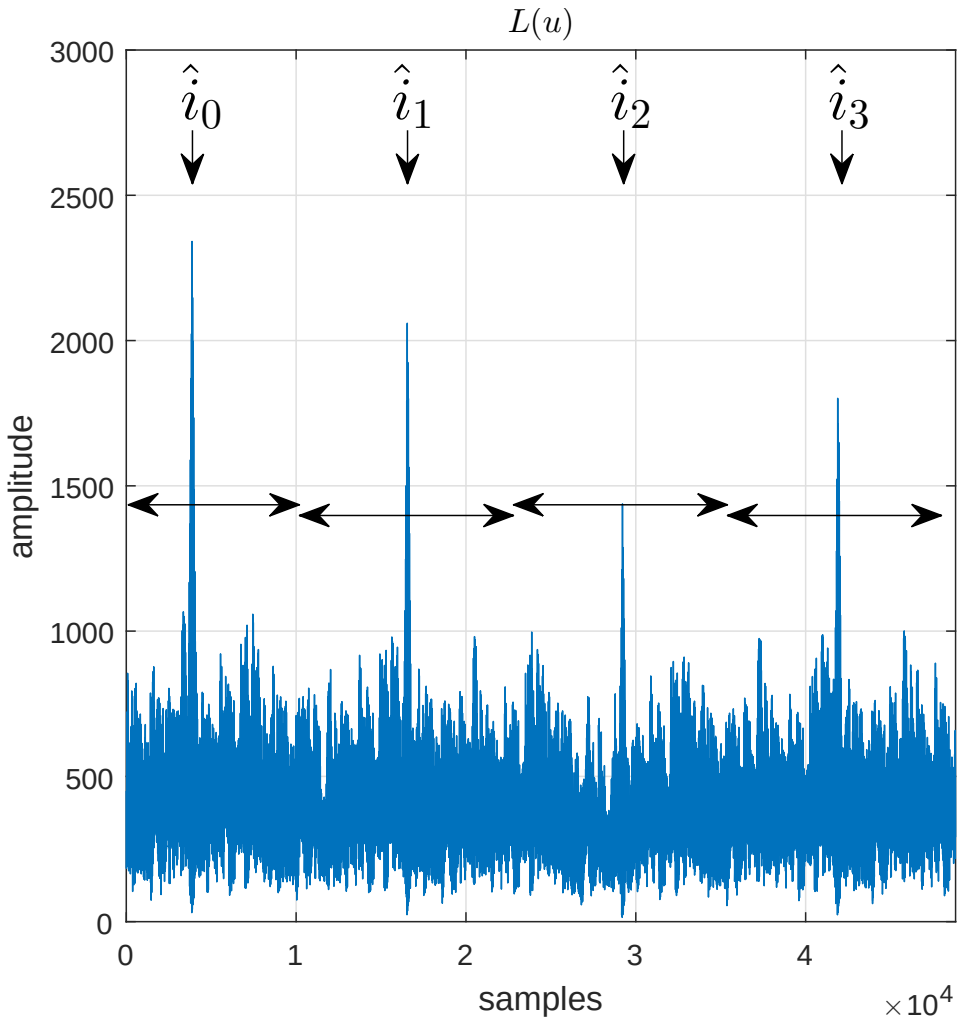

Figure 1.5: Safe search windows defined to search only one preamble correlation peak.

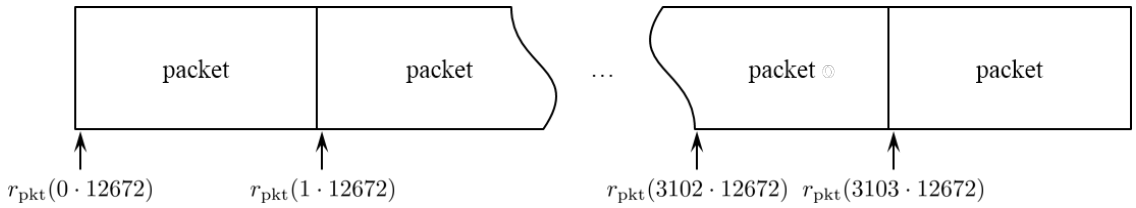

Figure 1.6: The packetized structure of the received signals after the frame synchronization step.

Packetizing the Samples

Finally, with the best estimate of the preamble starting locations, the received samples can be packetized. Using $\hat{\mathbf{i}}$, the GPU launches one thread per received sample to packetize $r(n)$ into $\mathbf{r}_{\text{pkt}}$ as shown in Figure 1.6. The structure of $\mathbf{r}_{\text{pkt}}$ majorly simplifies indexing and removes the need for $\hat{\mathbf{i}}$.

$$\mathbf{r}_{\text{pkt}} = \begin{bmatrix} r(\hat{i}_0) \\ r(\hat{i}_0 + 1) \\ \vdots \\ r(\hat{i}_0 + 12670) \\ r(\hat{i}_0 + 12671) \\ r(\hat{i}_1) \\ r(\hat{i}_1 + 1) \\ \vdots \\ r(\hat{i}_{3103} + 12670) \\ r(\hat{i}_{3103} + 12671) \end{bmatrix} \tag{1.7}$$

1.1.2 Frequency Offset Estimation and Compensation

With the vector $\mathbf{r}_{\text{pkt}}$ built, estimators can now easily use the preamble to estimate various parameters.

1.1.3 Channel Estimation

1.1.4 Noise Variance Estimation

1.1.5 OQPSK Detector

Chapter 2

Equalizer Equations

2.1 Overview

There are 3 different kinds of equalizers I run 1. the solving ones!!! They are equations like Ax=b where I have A and b but I need x 2. the initialized then iterative ones. CMA is initialized with MMSE then runs as many times a possible 3. the multiply ones! the FDEs are a simple multiply in the frequency domain

2.2 Equations

2.2.1 The Solving Equalizers

The Zero-Forcing Equalizer

The ZF equalizer was studied in the PAQ Phase 1 Final Report in equation 324

$$\mathbf{c}_{\text{ZF}} = (\mathbf{H}^{\dagger}\mathbf{H})^{-1}\mathbf{H}^{\dagger}\mathbf{u}_{n_0} \tag{2.1}$$

where $\mathbf{c}_{\text{ZF}}$ is a $L_{eq} \times 1$ vector of equalizer coefficients computed to invert the channel estimate $\mathbf{h}$. The channel estimate is used to build the $L_{eq} + N_1 + N_2 \times L_{eq}$ convolution matrix

$$\mathbf{H} = \begin{bmatrix} h(-N_1) & & & \\ h(-N_1+1) & h(-N_1) & & \\ \vdots & \vdots & \ddots & \\ h(N_2) & h(N_2-1) & & h(-N_1) \\ & h(N_2) & & h(-N_1+1) \\ & & & \vdots \\ & & & h(N_2) \end{bmatrix}. \tag{2.2}$$

and $\mathbf{u}_{n_0}$ is the desired channel impulse response centered on $n0 = N_1 + L_1 + 1$

$$\mathbf{u}_{n_0} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} n_0 - 1 \text{ zeros} \\ \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} N_1 + N_2 + L_1 + L_2 - n_0 + 1 \text{ zeros} \end{matrix} . \tag{2.3}$$

The computation of the coefficients in Equation (2.1) can be simplified in a couple of ways: First the matrix multiplication of $\mathbf{H}^\dagger$ and $\mathbf{H}$ is the autocorrelation matrix of the channel

$$\mathbf{R}_h = \mathbf{H}^\dagger\mathbf{H} = \begin{bmatrix} r_h(0) & r_h^*(1) & \cdots & r_h^*(L_{eq}-1) \\ r_h(1) & r_h(0) & \cdots & r_h^*(L_{eq}-2) \\ \vdots & \vdots & \ddots & \\ r_h(L_{eq}-1) & r_h(L_{eq}-2) & \cdots & r_h(0) \end{bmatrix} \tag{2.4}$$

where

$$r_h(k) = \sum_{n=-N_1}^{N_2} h(n)h^*(n-k). \tag{2.5}$$

Second the matrix vector multiplication of $\mathbf{H}^\dagger$ and $\mathbf{u}_{n_0}$ is simply the n_0th row of $\mathbf{H}^\dagger$ or the conjugated n_0th column of $\mathbf{H}$. A new vector $\mathbf{h}_{n_0}$ is defined by

$$\mathbf{h}_{n_0} = \mathbf{H}^\dagger\mathbf{u}_{n_0} = \begin{bmatrix} h(L_1) \\ \vdots \\ h(0) \\ \vdots \\ h(-L_2) \end{bmatrix} . \tag{2.6}$$

Replacing the matrix multiplication $\mathbf{H}^\dagger\mathbf{H}$ and $\mathbf{H}^\dagger\mathbf{u}_{n_0}$ simplifies Equation (2.1) to

$$\mathbf{c}_{\text{ZF}} = \mathbf{R}_h^{-1}\mathbf{h}_{n_0}. \tag{2.7}$$

Computing the inverse of $\mathbf{R}_h$ is computationally heavy because an inverse is an N^3 operation. To avoid an inverse, $\mathbf{R}_h$ is moved to the left side and $\mathbf{c}_{\text{ZF}}$ is found by solving a system of linear equations. Note that $r_h(k)$ only has support on $-L_{ch} \leq k \leq L_{ch}$ making $\mathbf{R}_h$ sparse or $\%63$ zeros. The sparseness of $\mathbf{R}_h$ can be leveraged to reduce computation drastically. The Zero-Forcing Equalizer coefficients are computed by solving

$$\mathbf{R}_h\mathbf{c}_{\text{ZF}} = \mathbf{h}_{n_0}. \tag{2.8}$$

MMSE Equalizer

The MMSE equalizer was studied in the PAQ Phase 1 Final Report in equation 330.

$$\mathbf{c}_{MMSE} = \left[\mathbf{G}\mathbf{G}^\dagger + \frac{\sigma_w^2}{\sigma_s^2}\mathbf{I}_{L_1+L_2+1}\right]\mathbf{g}^\dagger \tag{2.9}$$

where

$$\mathbf{G} = \begin{bmatrix} h(N_2) & \cdots & h(-N_1) & & & \\ & h(N_2) & \cdots & h(-N_1) & & \\ & & \ddots & & \ddots & \\ & & & h(N_2) & \cdots & h(-N_1) \end{bmatrix} \tag{2.10}$$

and

$$\mathbf{g} = \begin{bmatrix} h(L_1) \cdots h(-L_2) \end{bmatrix}. \tag{2.11}$$

The matrix multiplication $\mathbf{G}\mathbf{G}^\dagger$ is the same autocorrelation matrix $\mathbf{R}_h$ as Equation (2.4). The vector $\mathbf{g}^\dagger$ is also the same vector as $\mathbf{h}_{n_0}$. The signal-to-noise ratio estimate $\frac{1}{2\hat{\sigma_w^2}}$ is substituted in for the fraction $\frac{\sigma_w^2}{\sigma_s^2}$ using Equation 333 Rice's report. Equation (2.9) can be reformulated to

$$\left[\mathbf{R}_h + \frac{1}{2\hat{\sigma_w^2}}\mathbf{I}_{L_1+L_2+1}\right]\mathbf{c}_{\text{MMSE}} = \mathbf{h}_{n_0}. \tag{2.12}$$

The MMSE Equalizer coefficients are solved for in a similar fashion to Zero-Forcing is in Equation (2.8). The only difference between Equation (2.12) and (2.8) is the noise variance is added down the diagonal of $\mathbf{R}_h$. The matrix $\mathbf{R}_{hw}$ is defined to make the computation of the MMSE Equalizer coefficients the same as the Zero-Forcing Equalizer coefficients by adding the noise variance to the diagonal of $\mathbf{R}_h$

$$\mathbf{R}_{hw} = \mathbf{H}^{\dagger}\mathbf{H} = \begin{bmatrix} r_h(0) + \frac{1}{2\hat{\sigma_w^2}} & r_h^*(1) & \cdots & r_h^*(L_{eq}-1) \\ r_h(1) & r_h(0) + \frac{1}{2\hat{\sigma_w^2}} & \cdots & r_h^*(L_{eq}-2) \\ \vdots & \vdots & \ddots & \\ r_h(L_{eq}-1) & r_h(L_{eq}-2) & \cdots & r_h(0) + \frac{1}{2\hat{\sigma_w^2}} \end{bmatrix}. \tag{2.13}$$

The MMSE Equalizer coefficients are computed by solving

$$\mathbf{R}_{hw}\mathbf{c}_{\text{MMSE}} = \mathbf{h}_{n_0}. \tag{2.14}$$

2.2.2 The Iterative Equalizer

The Constant Modulus Algorithm

CMA uses a steepest decent algorithm.

$$\mathbf{c}_{b+1} = \mathbf{c}_b - \mu \nabla \mathbf{J} \tag{2.15}$$

The vector $\mathbf{J}$ is the cost function and ∇J is the cost function gradient defined in the PAQ report 352 by

$$\nabla J = \frac{2}{L_{pkt}} \sum_{n=0}^{L_{pkt}-1} \Big[y(n)y^*(n) - R_2 \Big] \, y(n)\mathbf{r}^*(n). \tag{2.16}$$

where

$$\mathbf{r}(n) = \begin{bmatrix} r(n+L_1) \\ \vdots \\ r(n) \\ \vdots \\ r(n-L_2) \end{bmatrix}. \tag{2.17}$$

This means ∇J is of the form

$$\nabla J = \begin{bmatrix} \nabla J(-L_1) \\ \vdots \\ \nabla J(0) \\ \vdots \\ \nabla J(L_2) \end{bmatrix}. \tag{2.18}$$

To Leverage computational efficiency of FFT, re-express the elements of ∇J as a convolution.

To begin define

$$z(n) = 2 \Big[y(n)y^*(n) - R_2 \Big] \, y(n) \tag{2.19}$$

so that ∇J may be expressed as

$$\nabla J = \frac{z(0)}{L_{pkt}} \begin{bmatrix} r^*(L_1) \\ \vdots \\ r^*(0) \\ \vdots \\ r^*(L_2) \end{bmatrix} + \frac{z(1)}{L_{pkt}} \begin{bmatrix} r^*(1+L_1) \\ \vdots \\ r^*(1) \\ \vdots \\ r^*(1-L_2) \end{bmatrix} + \cdots \frac{z(L_{pkt}-1)}{L_{pkt}} \begin{bmatrix} r^*(L_{pkt}-1+L_1) \\ \vdots \\ r^*(L_{pkt}-1) \\ \vdots \\ r^*(L_{pkt}-1-L_2) \end{bmatrix}. \quad (2.20)$$

The kth value of ∇J is

$$\nabla J(k) = \frac{1}{L_{pkt}} \sum_{m=0}^{L_{pkt}-1} z(m) r^*(m-k), \quad -L_1 \leq k \leq L_2. \quad (2.21)$$

The summation almost looks like a convolution. To put the summation in convolution form, define

$$\rho(n) = r^*(n). \quad (2.22)$$

Now

$$\nabla J(k) = \frac{1}{L_{pkt}} \sum_{m=0}^{L_{pkt}-1} z(m) \rho(k-m). \quad (2.23)$$

Because $z(n)$ has support on $0 \leq n \leq L_{pkt}-1$ and $\rho(n)$ has support on $-L_{pkt}+1 \leq n \leq 0$, the result of the convolution sum $b(n)$ has support on $-L_{pkt}+1 \leq n \leq L_{pkt}-1$. Putting all the pieces together, we have

$$\begin{aligned} b(n) &= \sum_{m=0}^{L_{pkt}-1} z(m) \rho(n-m) \\ &= \sum_{m=0}^{L_{pkt}-1} z(m r^*(m-n) \end{aligned} \quad (2.24)$$

Comparing Equation (2.23) and (2.24) shows that

$$\nabla J(k) = \frac{1}{L_{pkt}} b(k), \quad -L_1 \leq k \leq L_2. \quad (2.25)$$

The values of interest are shown in Figure Foo!!!!(c)

This suggest the following algorithm for computing the gradient vector ∇J Matlab Code!!!

2.2.3 The Multiply Equalizers

The Frequency Domain Equalizer One

The Frequency Domain Equalizer One (FDE1) is the MMSE or wiener filter applied in the frequency domain. Ian E. Williams and M. Saquib derived FDE1 for this project in a paper called Linear Frequency Domain Equalization of SOQPSK-TG for Wideband Aeronautical Telemetry. The FDE1 equalizer is defined in Equation (11) as

$$C_{\text{FDE1}}(\omega) = \frac{\hat{H}^*(\omega)}{|\hat{H}(\omega)|^2 + \frac{1}{\hat{\sigma}^2}} \tag{2.26}$$

The term $C_{\text{FDE1}}(\omega)$ is the Frequency Domain Equalizer One frequency response at ω. The term $\hat{H}(\omega)$ is the channel estimate frequency response at ω. The term $\hat{\sigma}^2$ is the noise variance estimate, this term is completely independent of frequency because the noise is assumed to be white or spectrally flat.

FDE1 needs no massaging because Equation (2.26) is easily implemented in the GPU and it is computationally efficient.

The Frequency Domain Equalizer One

The Frequency Domain Equalizer Twe (FDE2) is the MMSE or wiener filter applied in the frequency domain. Ian E. Williams and M. Saquib derived FDE1 for this project in a paper called Linear Frequency Domain Equalization of SOQPSK-TG for Wideband Aeronautical Telemetry. The FDE2 equalizer is defined in Equation (12) as

$$C_{\text{FDE2}}(\omega) = \frac{\hat{H}^*(\omega)}{|\hat{H}(\omega)|^2 + \frac{\Psi(\omega)}{\hat{\sigma}^2}} \tag{2.27}$$

FDE2 almost identical to FDE1. The only difference is term $\Psi(\omega)$ in the denominator. The term $\Psi(\omega)$ is the average spectrum of SPQOSK-TG shown in Figure 2.1. FDE2 needs no massaging because Equation (2.27) is easily implemented in the GPU and is computationally efficient.

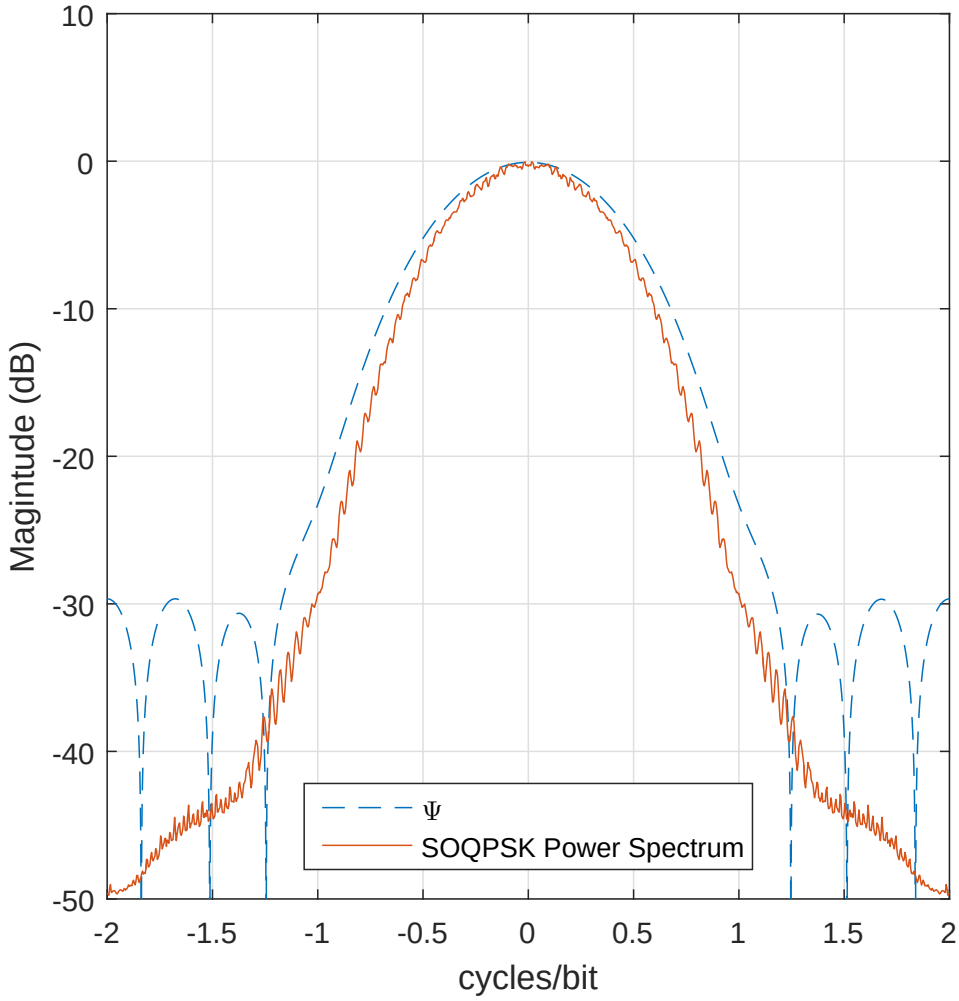

Figure 2.1: A block diagram illustrating organization of the algorithms in the GPU.

Bibliography

[1] M. Rice and A. Mcmurdie, "On frame synchronization in aeronautical telemetry," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 5, pp. 2263–2280, October 2016. 3