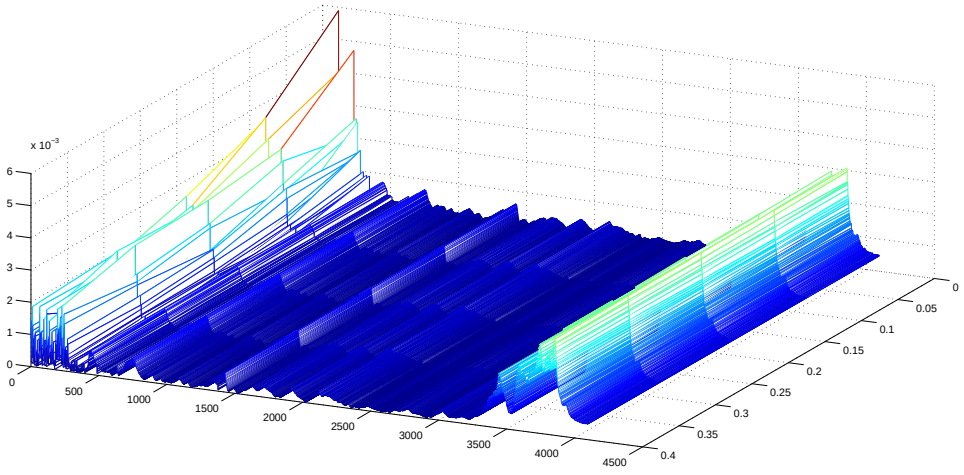
CHALMERS



Hardware Platform For Active Acoustic Spectroscopy Sensors

Master of Science Thesis in the Programme Integrated Electronic System Design

LIXUN XIA
BIN LIAO

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, September 2010

Hardware Platform for Active Acoustic Spectroscopy Sensors

LIXUN XIA
BIN LIAO

Examiner: Lars Bengtsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone +46(0)31-772 1000

Cover:
Frequency response functions with different number of windows for averaging, on page 12.

Department of Computer Science and Engineering
Göteborg, Sweden 2010

MASTER'S THESIS 2010:09

Hardware Platform for Active Acoustic Spectroscopy Sensors

Master's Thesis in Integrated Electronic System Design

LIXUN XIA
BIN LIAO

Department of Computer Science and Engineering
Division of Computer Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2010

Hardware Platform for Active Acoustic Spectroscopy Sensors
Master's Thesis in the Master's Program in Integrated Electronic System Design
LIXUN XIA
BIN LIAO
Department of Computer Science and Engineering
Division of Computer Engineering
Chalmers University of Technology

Abstract

Real-time estimation of the frequency response function of fluids is one of the critical building blocks for cost-effective application of active acoustic spectroscopy technology in process control industry. The motivation was to study and develop real-time algorithm for non-parametric estimation of frequency response function on embedded platforms, according to the specific requirements and application scenario involved in active acoustic spectroscopy sensors. In addition to that, implementations of the algorithm on fixed-point and floating-point digital signal processors have been made in order to benchmark the performance and evaluate different platforms.

The up-to-date ADSP-BF561 and ADSP-21489 processors, designed for signal processing tasks around audio frequency range, served as the implementation and evaluation platforms. The embedded programs for each have been developed based on the distinct processor architectures and memory hierarchies. It has been observed that the floating-point processor which runs at much slower core frequency, delivers better performance over the other. By the benchmark results, it could be identified that the less efficient memory substructure of the fixed-point processor constitutes the bottleneck of its total performance.

Keywords: Active Acoustic Spectroscopy, Frequency Response Function, Digital Signal Processing

Contents

Acknowledgements

We would like to express our gratitude to the examiner of the thesis Lars Bengtsson from Chalmers University of Technology, and our supervisor David Brohall and Felix Törner from Acosense AB.

1. Introduction

As a company developing active acoustic spectroscopy sensors for process control industry, Acosense AB needs real-time estimation of the frequency response function(FRF) of fluids flowing through pipes based on acoustic noise-stimulus-and-response method. The system to be estimated could be modeled as a single-input single output(SISO) linear system, but with delay due to the physical propagation time of the sound wave. The estimation algorithm is expected to provide decent estimate precisions and meanwhile to keep a small profile so that the program could run on commercial digital signal processors. The hardware have three input signals to process during estimation: one stimulus and two responses. Therefore a single such device would be able to calculate two FRFs. Figure 1.1 has shown a pipe with two sensors clamped around both edges.

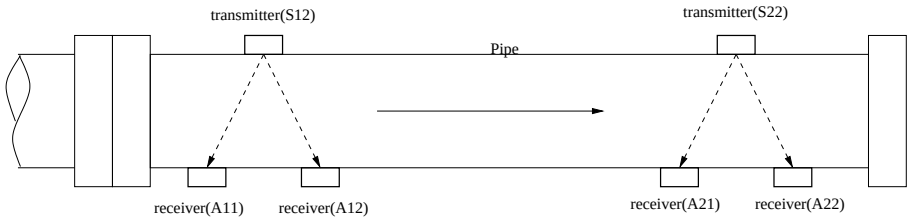


Figure 1.1.: Sensor Installation

In chapter 2, we will take a close look at the methods for FRF estimation. Different estimators will be examined and evaluated according to the design trade-offs decided by the hardware resource available. H1 estimator turns out to be the best method for our application. In chapter 3, algorithm design and implementation based on the fixed-point digital signal processor is presented. Due to the insufficient capacity of the fast on-chip memory on this platform, slower SDRAM must be used to fulfill the computation. Meanwhile the program has to be modified because of the restricted on-chip memory space. The counterpart implementation on the floating-point processor will be given in chapter 4. Thanks to the larger internal memory and the floating point unit, algorithm could be realized straight forward on this platform. The error analysis and performance benchmark between both platforms will be discussed in the final chapter(chapter 5). The precision and computation speed for both have been measured and compared in several tables.

2. Theories for Estimation of Frequency Response Function

2.1. System Model

The system to be estimated $H(\omega)$ can be modeled as a linear single-input single-output(SISO) system, with propagation delay τ of the sound. The measurements of the stimulus and response would also be contaminated by additive measurement noises. Other physical effects such as Doppler frequency shift are not modeled and will not be discussed in the thesis. Thus the model for the test system can be visualized as in the block diagram of figure 2.1.

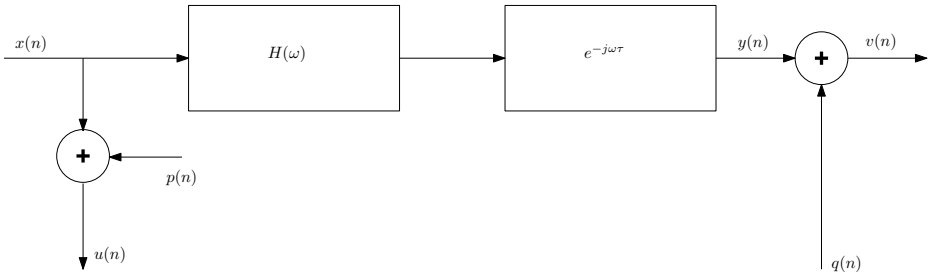


Figure 2.1.: System Under Estimate

$x(n)$ and $y(n)$ are the stimulus and response signal respectively, whose measurements – $u(n)$ and $v(n)$ contain the observation noises $p(n)$ and $q(n)$. We assume these noises, which come from AD conversion or any other sources, are uncorrelated with each other and with the stimulus and response. The stimulus time series we are using are white noises with user defined distributions. Therefore the stimulus belongs to wide sense stationary stochastic process. This process, filtered by the linear system $H(\omega)$, gives the output as a process that is also wide sense stationary. Accordingly our future discussions will be based on the presumption that both stimulus and response are wide sense stationary stochastic process. We will derive all equations that are going to be realized in the prototype from the definition of the cross correlation function at the beginning. Each step of the derivation and analysis is helpful to analyze and verify the algorithms for implementation at early phase.

2.2. Estimate Spectral Densities

Estimation of spectral densities has been investigated for a long time and it has a strong connection with FRF estimation. We will focus on cross correlation and cross spectral density estimation in this section because auto correlation and power spectral density are just the special cases. For discrete time series $x(n)$ and $y(n)$, the cross correlation between them is defined as

$$r_{xy}(k) = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} x^*(n)y(n+k) \tag{2.1}$$

We usually work with a finite number or "window" of samples in engineering due to the limited resource of the hardware, and this "window" of the true time series often results in frequency leakages [Whw 09]. One solution to this problem is to add weights to the samples within such that the credibility of the samples reduces from the middle of the window to both far ends, besides the weights outside the window must be zero. So any "window" of the true time series is identical to the multiplication of the true time series which are infinitely long with a window function. So the cross-correlation function of rectangularly windowed time series $x(0), x(1), \ldots x(N-1), x(k) = 0$ when $k < 0$ or $k > N-1$ and $y(0), y(1), \ldots y(N-1), y(k) = 0$ when $k < 0$ or $k > N-1$ could be derived in equation 2.2, according to definition 2.1.

$$\hat{r}_{xy}(k) = \frac{1}{N} \sum_{n=0}^{N-1} x^*(n)y(n+k) \tag{2.2}$$

$\hat{r}_{xy}(k)$ could also be regarded as an estimate of the authenticate cross correlation function. The cross spectral density is the discrete-time fourier transform(DTFT) of the cross-correlation function of both wide sense stationary processes, given as the Wiener–Vkhinchin theorem and shown in equation 2.3.

$$S_{xy}(\omega) = DTFT\{r_{xy}(k)\} = \sum_{k=-\infty}^{\infty} r_{xy}(k)e^{-j\omega k} \tag{2.3}$$

So the cross spectral density of the rectangularly windowed samples would be

$$\hat{S}_{xy}(\omega) = DTFT\{\hat{r}_{xy}(k)\} = \sum_{k=-\infty}^{\infty} \hat{r}_{xy}(k)e^{-j\omega k} = \sum_{k=1-N}^{N-1} \hat{r}_{xy}(k)e^{-j\omega k} \tag{2.4}$$

Meanwhile the cross spectral density in equation 2.4 would alternatively be obtained from multiplication of DTFT of the rectangularly windowed time series directly.

$$\begin{aligned} \hat{S}_{xy}(\omega) &= \frac{1}{N} X^*(\omega)Y(\omega) \\ &= \frac{1}{N} \left[\sum_{n=0}^{N-1} x(n)e^{-j\omega n}\right]^* \left[\sum_{m=0}^{N-1} y(m)e^{-j\omega m}\right] \end{aligned} \tag{2.5}$$

We can show that equation 2.4 and 2.5 are the same, because they are just different ways of summing up the entries in equation 2.6 and 2.7. The cross correlation matrix can be derived by the multiplication of column vector $x(n)$ and $y(n)$ as in equation 2.6.

$$x(n)y(n)' = \begin{pmatrix} x_0y_0 & x_0y_1 & x_0y_2 & x_0y_3 & \cdots & x_0y_{N-1} \\ x_1y_0 & x_1y_1 & x_1y_2 & x_1y_3 & \cdots & x_1y_{N-1} \\ x_2y_0 & x_2y_1 & x_2y_2 & x_2y_3 & \cdots & x_2y_{N-1} \\ x_3y_0 & x_3y_1 & x_3y_2 & x_3y_3 & \cdots & x_3y_{N-1} \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots \\ x_{N-1}y_0 & x_{N-1}y_1 & x_{N-1}y_2 & x_{N-1}y_3 & \cdots & x_{N-1}y_{N-1} \end{pmatrix} \tag{2.6}$$

The phasor matrix can be written as in equation 2.7.

$$W^{\phi}(\omega) = \begin{pmatrix} e^{-j\omega 0} & e^{-j\omega 1} & e^{-j\omega 2} & e^{-j\omega 3} & \cdots & e^{-j\omega(N-1)} \\ e^{-j\omega(-1)} & e^{-j\omega 0} & e^{-j\omega 1} & e^{-j\omega 2} & \cdots & e^{-j\omega(N-2)} \\ e^{-j\omega(-2)} & e^{-j\omega(-1)} & e^{-j\omega 0} & e^{-j\omega 1} & \cdots & e^{-j\omega(N-3)} \\ e^{-j\omega(-3)} & e^{-j\omega(-2)} & e^{-j\omega(-1)} & e^{-j\omega 0} & \cdots & e^{-j\omega(N-4)} \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots \\ e^{-j\omega(1-N)} & e^{-j\omega(2-N)} & e^{-j\omega(3-N)} & e^{-j\omega(4-N)} & \cdots & e^{-j\omega 0} \end{pmatrix} \tag{2.7}$$

Then the cross spectral density $\hat{S}_{xy}(\omega)$ is the "dot product" of the cross correlation matrix and the phasor matrix, given in equation 2.8. The "dot product" operator multiplies corresponding entries between two matrices and accumulates the productions, similar as dot product of two vectors. Equation 2.5 implies that the entries of matrix in equation 2.8 are summed row by row. This can be verified by expanding the polynomial multiplications in equation 2.5. While in equation 2.4, the entries are summed up diagonally, as marked by the red line for the zero lag.

$$\hat{S}_{xy}(\omega) = \frac{1}{N} x(n)y(n)' \cdot W^{\phi}(\omega) \tag{2.8}$$

Because equation 2.4 and 2.5 are mathematically the same, we will develop the mathematical expectation with both methods for further comparison and verification, which will be described in the following subsections.

2.2.1. Transform of Correlation Functions

From equation 2.4 we have: when $k = 0, 1, \cdots, N-1$,

$$\begin{aligned} E\left\{\hat{r}_{xy}(k)\right\} &= \frac{1}{N}\sum_{n=0}^{N-1} E\left\{x^*(n)y(n+k)\right\} \\ &= \frac{1}{N}\sum_{n=0}^{N-1-k} E\left\{x^*(n)y(n+k)\right\} \\ &= \frac{1}{N}\sum_{n=0}^{N-1-k} r_{xy}(k) \\ &= \frac{N-k}{N} r_{xy}(k) \end{aligned} \tag{2.9}$$

when $k = 1-N, 2-N, \cdots, -1$,

$$\begin{aligned} E\left\{\hat{r}_{xy}(k)\right\} &= \frac{1}{N}\sum_{n=0}^{N-1} E\left\{x^*(n)y(n+k)\right\} \\ &= \frac{1}{N}\sum_{n=-k}^{N-1} E\left\{x^*(n)y(n+k)\right\} \\ &= \frac{1}{N}\sum_{n=-k}^{N-1} r_{xy}(k) \\ &= \frac{N+k}{N} r_{xy}(k) \end{aligned} \tag{2.10}$$

when $|k| \geq N$, $\hat{r}_{xy}(k)$ gives only zeros, and the expectation will also be zero. To sum up, we have

$$E\left\{\hat{r}_{xy}(k)\right\} = w_B(k) r_{xy}(k) \tag{2.11}$$

where

$$w_B(k) = \begin{cases} \frac{N-|k|}{N} & |k| < N \\ \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

The triangular shaped $w_B(k)$ function was also named Bartlett window, in figure 2.2. As the existence of the window, $\hat{r}_{xy}$ is a biased estimate of the true cross correlation function. Nonetheless, we can always extend the number of N so that the window converges to constant unity in the time domain, which in turn gives an unbiased estimate of the cross correlation. Therefore $\hat{r}_{xy}$ is an asymptotic unbiased estimate of the true cross correlation [Hys 96].

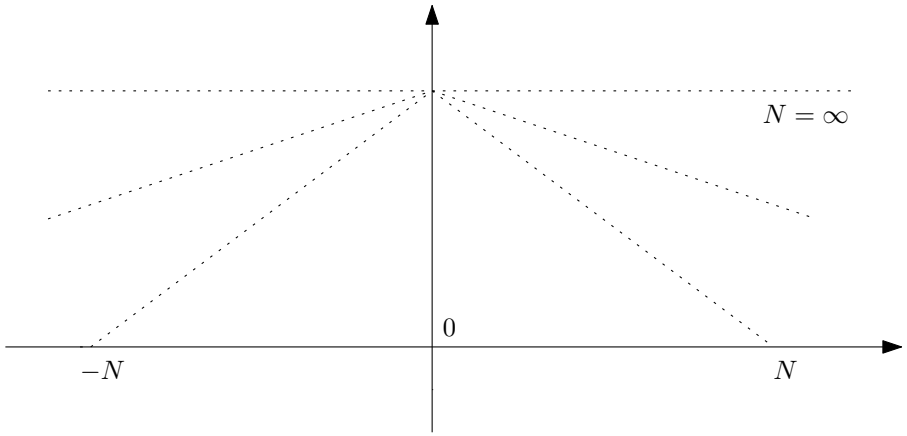


Figure 2.2.: Bartlett Window

The mathematical expectation of the cross spectral density $\hat{S}_{xy}(\omega)$ will be

$$\begin{aligned} E\left\{\hat{S}_{xy}(\omega)\right\} &= E\left\{\sum_{k=-\infty}^{\infty} \hat{r}_{xy}(k)e^{-j\omega k}\right\} \\ &= \sum_{k=-\infty}^{\infty} E\left\{\hat{r}_{xy}(k)\right\} e^{-j\omega k} \\ &= \sum_{k=-\infty}^{\infty} r_{xy}(k)w_B(k)e^{-j\omega k} \\ &= \frac{1}{2\pi} S_{xy}(\omega) * W_B(\omega) \end{aligned} \tag{2.13}$$

Where "$*$" indicates convolution operation, besides $W_B(\omega)$ is the DTFT of $w_B(k)$. As $N \to \infty$ we have $w_B(k) \to 1$. It follows that $W_B(\omega) \to 2\pi\delta(\omega)$. Hence

$$\begin{aligned} \lim_{N\to\infty} E\left\{\hat{S}_{xy}(\omega)\right\} &= \lim_{N\to\infty} \frac{1}{2\pi} S_{xy}(\omega) * W_B(\omega) \\ &= \frac{1}{2\pi} S_{xy}(\omega) * 2\pi\delta(\omega) \\ &= S_{xy}(\omega) * \delta(\omega) = S_{xy}(\omega) \end{aligned} \tag{2.14}$$

2.2.2. Periodogram's Perspective

From equation 2.5, given $w_R(k)$ as the rectangular window, i.e. $w_R(k) = 1$ when $k = 0 \cdots N-1$ otherwise 0, we have:

$$\begin{aligned}
E\left\{\hat{S}_{xy}(\omega)\right\} &= \frac{1}{N} E\left\{ \left[\sum_{n=0}^{N-1} x(n) w_R(n) e^{-j\omega n}\right]^* \left[\sum_{m=0}^{N-1} y(m) w_R(m) e^{-j\omega m}\right] \right\} \\
&= \frac{1}{N} E\left\{ \left[\sum_{n=0}^{N-1} x^*(n) w_R(n) e^{j\omega n}\right] \left[\sum_{m=0}^{N-1} y(m) w_R(m) e^{-j\omega m}\right] \right\} \\
&= \frac{1}{N} E\left\{ \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x^*(n) w_R(n) y(m) w_R(m) e^{-j\omega(m-n)} \right\} \\
&= \frac{1}{N} \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} E\left\{x^*(n) y(m)\right\} w_R(n) w_R(m) e^{-j\omega(m-n)} \\
&= \frac{1}{N} \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} r_{xy}(m-n) w_R(n) w_R(m) e^{-j\omega(m-n)}
\end{aligned} \tag{2.15}$$

Note that the rectangular and Bartlett window are all real functions, whereas $x(n)$ and $y(n)$ are not necessary the case. By variable substitution as $k = m - n$, we have:

$$\begin{aligned}
E\left\{\hat{S}_{xy}(\omega)\right\} &= \frac{1}{N} \sum_{k=1-N}^{N-1}\sum_{m=0}^{N-1} r_{xy}(k) w_R(m-k) w_R(m) e^{-j\omega k} \\
&= \sum_{k=1-N}^{N-1} r_{xy}(k) \left[\frac{1}{N}\sum_{m=0}^{N-1} w_R(m-k) w_R(m)\right] e^{-j\omega k} \\
&= \sum_{k=1-N}^{N-1} r_{xy}(k) w_B(k) e^{-j\omega k} \\
&= \frac{1}{2\pi} S_{xy}(\omega) * W_B(\omega)
\end{aligned} \tag{2.16}$$

The result is consistent with what we've obtained before. Besides we have got a new relationship from equation 2.16:

$$w_B(k) = \frac{1}{N}\sum_{m=-\infty}^{\infty} w_R(m-k) w_R(m) = r_{RR}(k) = \frac{1}{N} w_R(k) * w_R(-k) \tag{2.17}$$

Therefore the DTFT of the Bartlett window can be represented by DTFT of rectangular window shown in equation 2.18.

$$\begin{aligned}
W_B(\omega) &= \sum_{k=-\infty}^{\infty} w_B(k) e^{-j\omega k} \\
&= \sum_{k=-\infty}^{\infty} \left[\frac{1}{N} w_R(k) * w_R(-k)\right] e^{-j\omega k} \\
&= \frac{1}{N} W_R(\omega) W_R^*(\omega) = \frac{1}{N} \left|W_R(\omega)\right|^2
\end{aligned} \tag{2.18}$$

Consequently, we have

$$E\left\{\hat{S}_{xy}(\omega)\right\} = \frac{1}{2\pi N} S_{xy}(\omega) * |W_R(\omega)|^2 \tag{2.19}$$

2.2.3. A More General Result

Although we have expected the same results from both methods previously described. The run-time and space complexities are different from each other. For the cross-correlation method, we have to handle data array of length up to $2N-1$. The periodogram method on the other hand has a constant data array length of N. For large number of N, it is more feasible to use the periodogram based method. Therefore we prefer the periodogram based method to be implemented for the prototype.

In a more general case, the discrete time series are windowed with arbitrary window function, marked as $w(n)$. Based on the conclusion of section 2.2.2, we could derive the result quickly as well. For a window function other than rectangular, the mathematical expectation of the estimate might be biased, thus we need to insert a scalar to compensate this bias. Assume we need U for an asymptotically unbiased estimate of the CSD.

$$\hat{S}_{xy}(\omega) = \frac{1}{NU}\left[\sum_{n=-\infty}^{\infty} x(n)w(n)e^{-j\omega n}\right]^* \left[\sum_{m=-\infty}^{\infty} y(m)w(m)e^{-j\omega m}\right] \tag{2.20}$$

So we have the mathematical expectation by equation 2.19:

$$\begin{aligned} E\left\{\hat{S}_{xy}(\omega)\right\} &= \frac{1}{2\pi NU} S_{xy}(\omega) * |W(\omega)|^2 \\ &= S_{xy}(\omega) * \frac{1}{2\pi NU} |W(\omega)|^2 \end{aligned} \tag{2.21}$$

We need the area of the function that $S_{xy}(\omega)$ convolutes with to be an unity, so that when $N \to \infty$, the $\delta(\omega)$ function would be unit area(no bias).

$$\int_{-\pi}^{\pi} \frac{1}{2\pi NU} |W(\omega)|^2 \, d\omega = 1 \tag{2.22}$$

Now considering Parseval theorem(proof in the appendix), we have

$$\sum_{n=0}^{N-1} |w(n)|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |W(\omega)|^2 \, d\omega \tag{2.23}$$

With equation 2.22 and 2.23 we have determined the value for U as

$$U = \frac{1}{N} \sum_{n=0}^{N-1} |w(n)|^2 \tag{2.24}$$

The equation 2.20 can even more be averaged over several overlapped or non-overlapped data windows to reduce the variance [Pwh 67]. Given K data windows, it can be described as

$$\hat{S}_{xy}(\omega) = \frac{1}{NUK} \sum_{i=0}^{K-1} \left[\sum_{n=-\infty}^{\infty} x_i(n) w(n) e^{-j\omega n} \right]^* \left[\sum_{m=-\infty}^{\infty} y_i(m) w(m) e^{-j\omega m} \right] \tag{2.25}$$

The reduce of variance is crucial since only when both (1) the sample mean of the estimate is unbiased or asymptotically unbiased, (2) variance converges to zero, can we say that the estimate is consistent. The proof that guarantees the convergence of the variance can be found in literature [Hys 96].

2.3. Estimate FRF

Now that equation 2.25 give us a consistent and implementation efficient way to estimate spectral densities, we can step further to find out the relationship between different spectra. Assume the impulse response of our system under estimate is $h(n)$, as in figure 2.1, the output of the system will be

$$y(n) = \sum_{m=0}^{\infty} h(m) x(n-m) \tag{2.26}$$

The cross correlation of the input and output will be

$$\begin{aligned} r_{xy}(k) &= E\left\{x^*(n) y(n+k)\right\} \\ &= E\left\{ x^*(n) \sum_{m=0}^{\infty} h(m) x(n+k-m) \right\} \\ &= E\left\{ \sum_{m=0}^{\infty} h(m) x^*(n) x(n+k-m) \right\} \\ &= \sum_{m=0}^{\infty} h(m) E\left\{x^*(n) x(n+k-m)\right\} \\ &= \sum_{m=0}^{\infty} h(m) r_{xx}(k-m) \\ &= h(k) * r_{xx}(k) \end{aligned} \tag{2.27}$$

Alternatively,

$$\begin{aligned} r_{yy}(k) &= E\left\{y^*(n)y(n+k)\right\} \\ &= E\left\{y^*(n)\sum_{m=0}^{\infty}h(m)x(n+k-m)\right\} \\ &= E\left\{\sum_{m=0}^{\infty}h(m)y^*(n)x(n+k-m)\right\} \\ &= \sum_{m=0}^{\infty}h(m)E\left\{y^*(n)x(n+k-m)\right\} \\ &= \sum_{m=0}^{\infty}h(m)r_{yx}(k-m) \\ &= h(k)*r_{yx}(k) \end{aligned} \tag{2.28}$$

It follows the relationship in frequency domain as

$$S_{xy}(\omega) = H(\omega)S_{xx}(\omega) \tag{2.29}$$

$$S_{yy}(\omega) = H(\omega)S_{yx}(\omega) \tag{2.30}$$

Hence, as long as we know relevant spectral densities of input and output signals, the estimate of the frequency response function would be

$$F\hat{R}F = \hat{H}(\omega) = \frac{\hat{S}_{xy}(\omega)}{\hat{S}_{xx}(\omega)} = \frac{\hat{S}_{yy}(\omega)}{\hat{S}_{yx}(\omega)} \tag{2.31}$$

Now we insert equation 2.20 into 2.31 and we have

$$\begin{aligned} \hat{H}(\omega) &= \frac{\frac{1}{NU}\left[\sum_{n=-\infty}^{\infty}x(n)w(n)e^{-j\omega n}\right]^*\left[\sum_{m=-\infty}^{\infty}y(m)w(m)e^{-j\omega m}\right]}{\frac{1}{NU}\left[\sum_{n=-\infty}^{\infty}x(n)w(n)e^{-j\omega n}\right]^*\left[\sum_{m=-\infty}^{\infty}x(m)w(m)e^{-j\omega m}\right]} \\ &= \frac{\sum_{m=-\infty}^{\infty}y(m)w(m)e^{-j\omega m}}{\sum_{m=-\infty}^{\infty}x(m)w(m)e^{-j\omega m}} \end{aligned} \tag{2.32}$$

We can see that the estimation based on single data window degenerates to the direct division of output and input in frequency domain, which has much higher variance

compared to averaged version using several data windows, described below

$$\begin{aligned}\hat{H}(\omega) &= \frac{\frac{1}{NUK}\sum_{i=0}^{K-1}\left[\sum_{n=-\infty}^{\infty} x_i(n)w(n)e^{-j\omega n}\right]^*\left[\sum_{m=-\infty}^{\infty} y_i(m)w(m)e^{-j\omega m}\right]}{\frac{1}{NUK}\sum_{i=0}^{K-1}\left[\sum_{n=-\infty}^{\infty} x_i(n)w(n)e^{-j\omega n}\right]^*\left[\sum_{m=-\infty}^{\infty} x_i(m)w(m)e^{-j\omega m}\right]} \\ &= \frac{\sum_{i=0}^{K-1}\left[\sum_{n=-\infty}^{\infty} x_i(n)w(n)e^{-j\omega n}\right]^*\left[\sum_{m=-\infty}^{\infty} y_i(m)w(m)e^{-j\omega m}\right]}{\sum_{i=0}^{K-1}\left[\sum_{n=-\infty}^{\infty} x_i(n)w(n)e^{-j\omega n}\right]^*\left[\sum_{m=-\infty}^{\infty} x_i(m)w(m)e^{-j\omega m}\right]}\end{aligned} \tag{2.33}$$

As shown in the cover figure, increase of total number of windows results in the increase of the FRF magnitude in low frequencies, showing better and better approximations to the real value. The Fourier transform in equation 2.33 is the DTFT of windowed sample series, which could be replaced by DFT when discrete frequency bins or frequency samples [Mkv 09], are preferred for digital implementations, as given in equation 2.34.

$$\hat{H}(k) = \frac{\sum_{i=0}^{K-1}\left[\sum_{n=-\infty}^{\infty} x_i(n)w(n)e^{-j\frac{2\pi}{N}kn}\right]^*\left[\sum_{m=-\infty}^{\infty} y_i(m)w(m)e^{-j\frac{2\pi}{N}km}\right]}{\sum_{i=0}^{K-1}\left[\sum_{n=-\infty}^{\infty} x_i(n)w(n)e^{-j\frac{2\pi}{N}kn}\right]^*\left[\sum_{m=-\infty}^{\infty} x_i(m)w(m)e^{-j\frac{2\pi}{N}km}\right]} \tag{2.34}$$

Nevertheless the measured signals contain observation noise. We should see what equation 2.31 becomes when measurement noise cannot be ignored. To simplify the analysis we can assume the additive noises are uncorrelated with each other and with the stimulus and response.

$$\begin{aligned}H_1(\omega) &= \frac{\hat{S_{uv}}}{\hat{S_{uu}}} \\ &= \frac{\hat{S_{xy}} + \hat{S_{py}} + \hat{S_{xq}} + \hat{S_{pq}}}{\hat{S_{xx}} + \hat{S_{px}} + \hat{S_{xp}} + \hat{S_{pp}}} \\ &= \frac{\hat{S_{xy}}}{\hat{S_{xx}} + \hat{S_{pp}}} \\ &= \frac{\hat{S_{xy}}}{\hat{S_{xx}}}\frac{\hat{S_{xx}}}{\hat{S_{xx}} + \hat{S_{pp}}} \\ &= \hat{H}(\omega)\frac{\hat{S_{xx}}}{\hat{S_{xx}} + \hat{S_{pp}}}\end{aligned} \tag{2.35}$$

Since $|S_{xx}/(S_{xx} + S_{pp})|$ will always less than 1, thus the input observation noise has introduced an error, which makes $|H1|$ estimate less than the perfect estimate $|\hat{H}(\omega)|$.

Following the same logic, we can get the so called $H2$ estimator, which has a positive systematic error for the output observation noise, shown in equation below.

$$\begin{aligned} H_2(\omega) &= \frac{\hat{S_{vv}}}{\hat{S_{vu}}} \\ &= \frac{\hat{S_{yy}} + \hat{S_{qy}} + \hat{S_{yq}} + \hat{S_{qq}}}{\hat{S_{yx}} + \hat{S_{qx}} + \hat{S_{yp}} + \hat{S_{qp}}} \\ &= \frac{\hat{S_{yy}} + \hat{S_{qq}}}{\hat{S_{yx}}} \\ &= \frac{\hat{S_{yy}}}{\hat{S_{yx}}} \frac{\hat{S_{yy}} + \hat{S_{qq}}}{\hat{S_{yy}}} \\ &= \hat{H}(\omega) \frac{\hat{S_{yy}} + \hat{S_{qq}}}{\hat{S_{yy}}} \end{aligned} \tag{2.36}$$

We can also notice that equation 2.36 can not be evaluated at zeros of the linear system due to the response power spectrum as the denominator. The third kind of FRF estimator is the geometric mean of H_1 and H_2, named with H_r. Although H_r provides a better estimate than H_1 and H_2 in terms of systematic error, still H_r could not handle systems with zeros. There are also many other sophisticated estimators [Apv 06] that optimize certain performance but the drawbacks are also obvious—they are not suitable for embedded computation. Taken these all into account, we have chosen the $H1$ estimator for implementation. The form of $H1$ can be referred to as in equation 2.34 but with $x(n)$ and $y(n)$ replaced by measured quantities $u(n)$ and $v(n)$.

The last factor that may affect our estimate quality would be the propagation delay of the sound wave. Mathematically this delay stage only introduces a phase shift for the FRF, which could be observed from the fourier transform of the delayed system. But we are working with limited window length(samples). The delay reduces the number of samples that are correlated with each other between stimulus and response in a given window. Therefore the propagation delay affects the magnitude response as well. Consequently it is a common practice to estimate or measure the delay first so that the stimulus and response can be properly aligned digitally in the memory space [Ayt 77]. The impulse response of the delayed system could be used for such detection.

2.4. Estimate Impulse Response

The impulse response function can be estimated statistically by time-domain correlation analysis method. Again we take a retrospect into equation 2.27. If the input signal

is white noise with unit variance($\sigma_x^2 = 1$), then we have the new relationship as

$$\begin{aligned} r_{xy}(k) &= h(k) * r_{xx}(k) \\ &= h(k) * \sigma_x^2 \delta(k) \\ &= \sigma_x^2 h(k) \\ &= h(k) \end{aligned} \tag{2.37}$$

Therefore the problem has been converted to how to precondition the input and output signals so that white noise can be achieved. We use whitening filters $F(\omega)$ added to known linear system, as in figure 2.3. The whitening filter is designed such that the fil-

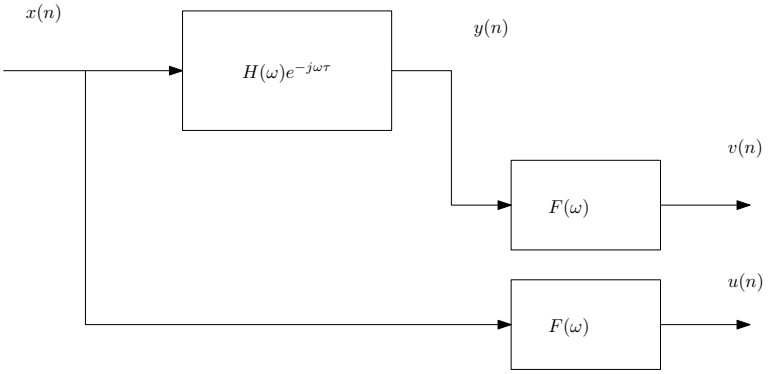


Figure 2.3.: Estimate of Impulse Response

tered $x(n)$, which is $u(n)$, will be white noise with variance σ^2. For the newly generated signals $u(n)$ and $v(n)$, still we have

$$\begin{aligned} r_{uv}(k) &= h(k) * r_{uu}(k) \\ &= h(k) * \sigma^2 \delta(k) \\ &= \sigma^2 h(k) \end{aligned} \tag{2.38}$$

$$\rightarrow h(k) = \frac{r_{uv}(k)}{\sigma^2}$$

The design of the whitening filter may start with auto-regressive estimation of the power spectrum of signal $x(n)$, and then compensate the spectrum to form a constant power spectrum, indicating uncorrelated time series or white noise. Efforts can be saved however, when $x(n)$ is already uncorrelated, thus whitening filters might not be needed.

In this chapter, we have rebuilt the entire calculation model for FRF estimation based from some basic definitions. This process helps analyze the conversion from mathematical equations to algorithms for machines. Even more, connections of different methods

have been clarified. Consequently this chapters serves as the foundation for the rest of the contents.

3. Fixed-Point Implementation

In this chapter, we will discuss how to use fixed-point DSP, ADSP-BF561 in the project, to implement FRF algorithm in real time system. First, we cover briefly description of hardware system. Then, dependent on property of hardware, we will focus on the topic that how to design suitable software system on it.

3.1. Hardware Description

The hardware system mainly consists of three parts, Analog/Digital Converter(ADC), Digital Signal Processor(DSP) and Computer(PC). In this project, ADSP-BF561 EZ-KIT lit(EZ-KIT), which is evaluation system for Blackfin processor, is used. It contains ADSP-BF561 Blackfin processor and AD1836 multichannel audio codec on the board. Connection between PC and EZ-KIT lit is showed in figure 3.1.

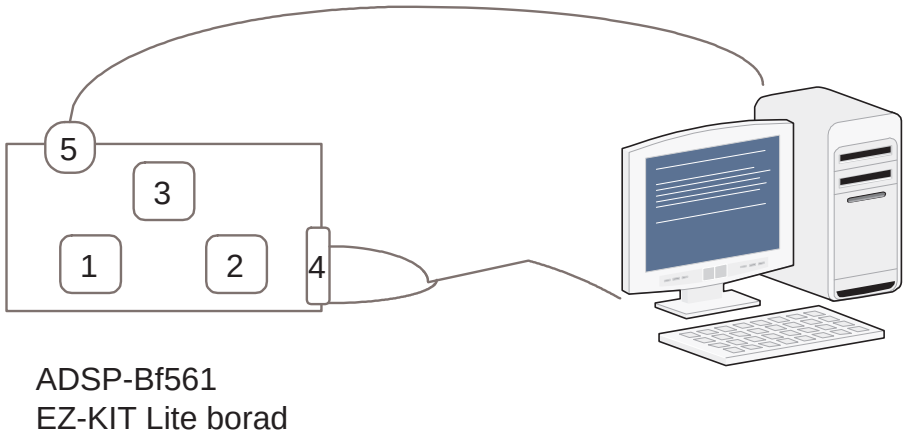


Figure 3.1.: Construction Of Hardware System: 1.ADSP-BF561 Blackfin processor, 2.AD1836 multichannel 96kHz audio codec, 3.64M SDRAM, 4.RCA jacks for stereo audio input, 5.USB port

One function of PC is used to send audio signals, which are imaged as analog input signals in the working environment, from sound card to RCA jack on EZ-KIT. Another function is used to receive and display results after processing. In DSP the final results are dumped out through USB port, then showed on PC screen by using Matlab

software. The following sections will briefly introduce the properties of each part of hardware.

3.1.1. Digital Signal Processor: ADSP-BF561

The ADSP-BF561[Bds 09] is a high performance member of Blackfin family of products targeting a variety of multimedia, industrial and telecommunications applications. At the heart of this device are two independent Analog Devices Blackfin processors. Each of these Blackfin processors combine a dual-MAC state-of-the-art engine, the advantage of a clean, orthogonal RISC-like microprocessor instruction set, and signal instruction, multiple data(SIMD) multimedia capabilities in a signal instruction set architecture.

The ADSP-BF561 processor has dual symmetric 600MHz high performance cores and each core has include:

- two 16-bit MACs
- two 40-bit ALUs
- four 8-bit video ALUs
- 40-bit shifter

And 328K bytes of on-chip memory. Each of core includes:

- 16K bytes of instruction SRAM/cache
- 16K bytes of instruction SRAM
- 32K bytes of data SRAM/cache
- 32K bytes of data SRAM
- 4K bytes of scratchpad SRAM

Additional on-chip memory peripherals include:

- 128K bytes of low latency on chip L2 SRAM
- Four-channel internal memory DMA controller
- External memory controller with glueless for SDRAM, mobile SDRAM, SRAM and flash.

Blackfin processors also provide advanced power management and performance, which is world-class level, but we are not supposed to use that property in this project. So we will not mention the detail of portable low power architecture here.

3.1.2. Analog-To-Digital Converter: AD 1836A

The AD 1836A is high performance, single-chip codec which provides three independent stereo DACs and two independent stereo ADCs. The ADC section can be operated at both 96KHz and 48KHz. And it accepts 16-/18-/20-/24-bit data. In this project, one of ADC channels is used for transforming analog input signals to digital input signals. It is supposed to work at 24-bit data length and 96KHz sampling rate in order to achieve high performance.

3.2. Software System

3.2.1. Dual Cores Programming

ADSP-BF561 processor[Bhr 10] has dual-symmetric blankfin cores on chip, each of which is running independent with each other. Each core has its own Level one(L1) cache, and a shared Level two(L2) cache. One of the biggest advantage of having a dual cores processor is for ability to perform thread level parallelism(TLP). A thread is a task that computer puts resources to run. One processor has two cores. It means that, on a single clock cycle, a dual-cores processor is able to operate more data and cost less energy relative to single-core processor with the same clock speed and the same architecture. Just as two head are better than one.

While the advantages far outweigh the disadvantage, we should be aware of a few negatives. One of these is that software need to be redesigned to work specifically with dual-cores processor. It is important to note that the most reliable design is to have an individual processor on a chip and due to the fact that dual-core processor always works faster and harder, which puts constraints on system buses and other part of hardware, such as memory. About memory, programmers have to carefully consider the fact, keeping memory consistence between two cores.

In this project, FRF algorithm is supposed to only be implemented on one core(core A), meanwhile another core might be remained to process works such as flow control in the future, which will not be covered in this report. Two motivations encourage us to do that. The first advantage is that memory consistence, which would definitely increase the cost of program development, do not need be considered in this stage. The second one is that there is no need add chip, if extra functions, such as system control, are supposed to be implemented on system in the future. Therefore programmer should have less space requirement of final products. As a result, we develop program on BF561 processor as single core processor programming during this stage, meanwhile core B is in idle condition all the time.

3.2.2. Programming Language

DSPs are programed in the same languages as other scientific or engineering applications, usually assembly or C. Assembly is able to easily control and operate hardware resource, such as MACs, ALUs and shifters to exploit parallel execution as much as possible, therefore the program written in assembly usually execute faster. However, it is difficult to program in assembly for beginner, since it will need roughly six months training before that.

While the program written in C is much easier relative to assembly. A key advantage of using C is that the programmer dose not need to totally understand architecture of processor being used; knowledge of architecture is left to compiler to deal with. So the programs written in C are usually easier to develop and maintain. However, if the programmer, who does not care about any knowledge about architecture and assembly instruction set, try to develop software system in C, the compiler probably can not translate them efficient to machine language. C code usually requires a large number of memory than assembly, resulting in more expensive hardware. Therefore, even if programmer plan to program only in C, we will probably need to write compiler-friendly sentences in C.

What is best language for our application? During this project developing, we do not have enough time to learn assembly and implement FRF algorithm in it. On the other hand, FRF is so complex algorithm which is inefficient and not easy to implement it in assembly from the beginning to the end. In comparison, the program written in C even can obtain good enough performance in this project. And we also can balance between to world: written program in C, but use assembly for critical sections which must execute quickly.

3.2.3. Fractional Number Computation

Fixed point DSPs, such as BF561, usually represents each number with minimum 16 bits. There are four common way that $2^{16} = 65536$ possible bit patterns can be represented as a number.In unsigned integer, the store number value can be any integer from 0 to 65536 and scaling factor is equal to $2^0 = 1$. Similarly, signed integer, use two' complement, in which the most significant bit(MSB) stands for the sign symbol, to make range from -32768 to 32767, and scaling factor is still equal to $2^0 = 1$. With unsigned fraction notation and signed fraction notation, the 65536 levels are uniformed spread between 0 and 1 and between -1 and 1 separately, while both of their scaling factors are equal to 2^{-15}. In the project, since the value of input data, which are sampled by ADC from analog environment, is always limit between -1 and 1, any numbers in DSP computation can be used as signed fraction pattern.[Api 09]

The purpose of this project is supposed to calculate FRF with a large number of points

each time, which is equal to 4096 or 8192. One motivation to do that is because we could obtain perfect accuracy on frequency spectrum of channel, no matter in frequency domain or amplitude domain. High resolution of result, of course, will benefit engineers for analysis in the future works. Another reason is that there always exist some time delay, which is estimated below 50 sampling periods, between sending side and receiving side. It will cause serious damage on phase frequency response, but little influence on amplitude frequency response. Without any delay detection, if the more samples is used for FRF calculation, the influence of delay will be decreased. If the more samples be used, the more relative points are used for calculation between the two sides. As a result, without any delay compensation, the results of frequency response are able to estimated more accuracy with a large number of samples relative a small one.

The coins have two sides. There is a big problem following with these benefits: the dynamic range of value of results. Since fixed point numbers are always with finite possible values, it is very important to programmer that adjusting this range to proper position to avoid overflow and underflow occur if the result of operation is larger or smaller than the numbers in that range. Taking 4096 points FRF calculation as an example, FFT on input data is a necessary step during FRF calculation. To avoid overflow, we at least scaling down by 2 in each stage of FFT. As 4096 points FFT, the results of FFT on inputs have to be scaled down by 2^{13}. And during cross-correlation and auto-correlation step, in which two results after FFT are multiplied together, the result is scaled down by 2^{26}. The problem is coming up. If the numbers in DSP are represented as 16 bits, it is out of ability of 16 bit numbers to handle overflow and underflow together. In other word, the dynamic range of results are much larger than 65536 possibles which 16 bits can be represented. While we try to prevent overflow occur, which will serious damage big value results, the small values are always equal to zero, where the underflow occur. The condition would turn to worst, when the 8192 points FRF is calculated. Therefore, it has to be considered that using 32 bits to represent each number.

As we mention before, each core of BF561 processor has two 16 bits MACs. It means that it working on 16 bits operations, such as multiplication and addition, are very efficient, which only need one clock cycle to finish that. However, it impossible to do the same thing efficient to 32 bits operation, especially 32 bits multiplication. Although BF561 assembler provides some intrinsic(built-in) functions[Clm 05], that enable efficient to use hardware resources, to do 32 bits operations, these functions still cost several times time relative to 16 bits operations. In this project, speed performance of FRF has to be scarified in order to guarantee overflow and underflow not happening under a lager dynamic range of results.

3.2.4. Software Design

The memory portions of BF561 are arranged in a hierarchical structure to provide good performance balance of very fast, low latency memory as cache or SRAM close to processor and larger, low cost and performance memory systems further from the processor. The level one(L1) on-chip memory systems in each core have the highest performance with limited memory size. The level two(L2) on-chip memory systems provide additional memory capacity with lower memory performance. Lastly, off-chip memory systems have much more memory capacities complied with high data latency.

According to table 3.1, programming on BF561 processor, which is under memory-restricted(on-chip memory) environment, is not easy. In this application, each signal on three channels, required to be processed simultaneously, use the window size of 8192, however the capacity of L1 data memory is only 68KB, where less than three of 8192-elements arrays are possibly stored. Moreover, beside input signals, the program also need extra spaces for interval arrays and look-up tables. Hence, we have to move out most of data to off-chip memory(L3), although a gap between processor speed and SDRAM speed is large.

Memory Level	Access Latency	Size	Data
L1 code memory	1 CCLK	32KB	Codes
L1 data memory	1 CCLK	68KB	Temporary data
L2 shared memory	9 CCLKs	128KB	DMA scripts
L3 SDRAM	8 to 10 SCLKs	4*16MB	All data

Table 3.1.: The Memory System

Fortunately, the Blackfin family processors, including ADSP-BF561, provide Direct Memory Access(DMA)[Bhr 10] to transfer data within memory spaces and between a memory space to a peripheral without the attention of the processor. With DMA we set the DMA engine off and running and at the end we just setup an interrupt to be triggered when the specified data stream transmission is accomplished, meanwhile we have 100% of processor time to allocated to main process. In this application, three DMAs are implemented and to be assigned the different jobs, listed in table 3.2. DMA0 and DMA1 cooperate to move samples from ADC port to circle buffer which is a space-saving method to placing all of samples. In addition, DMA2 is important part, because the processor is supposed to calculate data only in L1, where it works at core clock speed(600M Hz). DMA2 help the processor transfer the necessary data into L1 and move the results back to L3 memory.

Despite DMAs work independent with processor, they are still not able to process at the same time if they operation the same object. For example, if there is only one group of an array to store data in L1, after processing, the processor has to wait for DMA to

Name	ISR No.	Type	Mode	Function
DMA0	ISR0	Peripheral DMA	Autobuffer mode	Transmit samples: ADC→Receive buffer
DMA1	ISR1	Memory DMA	Descriptor list (small mode)	Transmit data: Receive buffer→Circular buffer
DMA2	ISR2	Memory DMA	Stop mode	Rest transmission work: L1↔L3

Table 3.2.: Three DMAs In Program

move out the result to SDRAM and replace the new data into the array. Also, when the data is calculated by processor, the DMAs are forbidden to intervene. Both of processor and DMAs have to stall waiting for other finish. In order to decrease the bad influence by structure, the double buffers are initialized. As illustrated in figure 3.2, one buffer is processed by processor, while the data in another buffer can be transferred from/to L3 memory. And vice versa. Consequently, the processor is free to run as fast as possible without wasting time of data transferring, meanwhile the most of transmission time can be covered by processor.

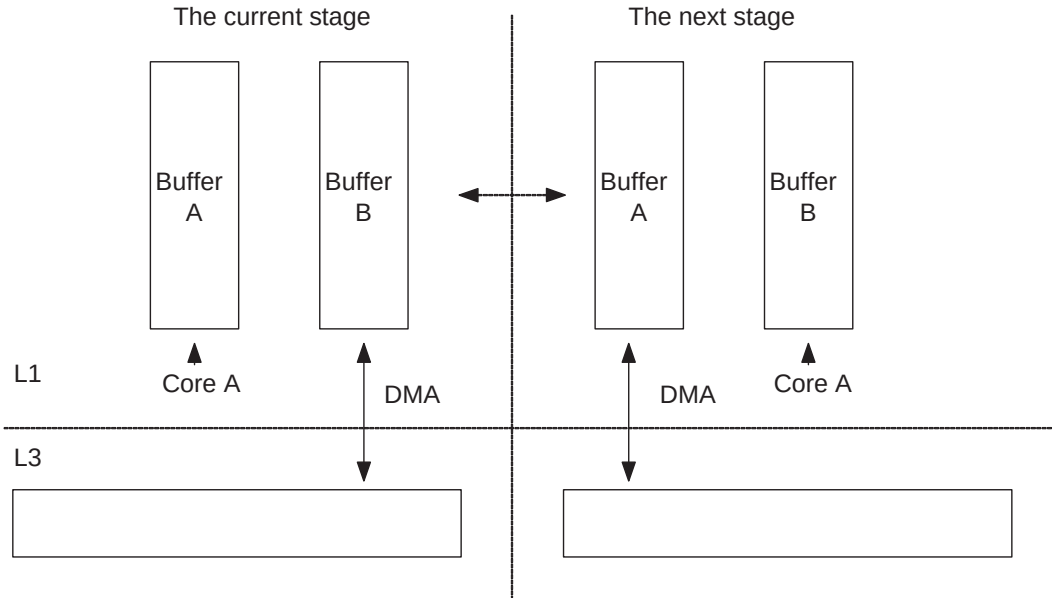


Figure 3.2.: Ping-Pong Buffer

Due to limited spaces in L1, six 1024-points arrays, working as buffer, have been created. But smaller buffer bring us a problem, one operation has to be separated into many small operations to finish the same work. That definitely add more data commu-

nication and increase the complexity of algorithm to program.

Because many data communications in codes, a number of interrupt events are generated by DMA. If any of two interrupt event occur at the same time, it is probably to cause program sequence disorder. Therefore, finite state machine(FSM), which defines many states to isolate every interrupt, is used here. Through changing state and value of flag signals, the handshake is made between processor and DMA. Then they can make sure to safely move to the next state.

The main algorithm of FRF consist of five main parts, windowing, FFT, spectral calculation, window accumulation and final division of FRF. The program flow diagram is showed in figure 3.3.

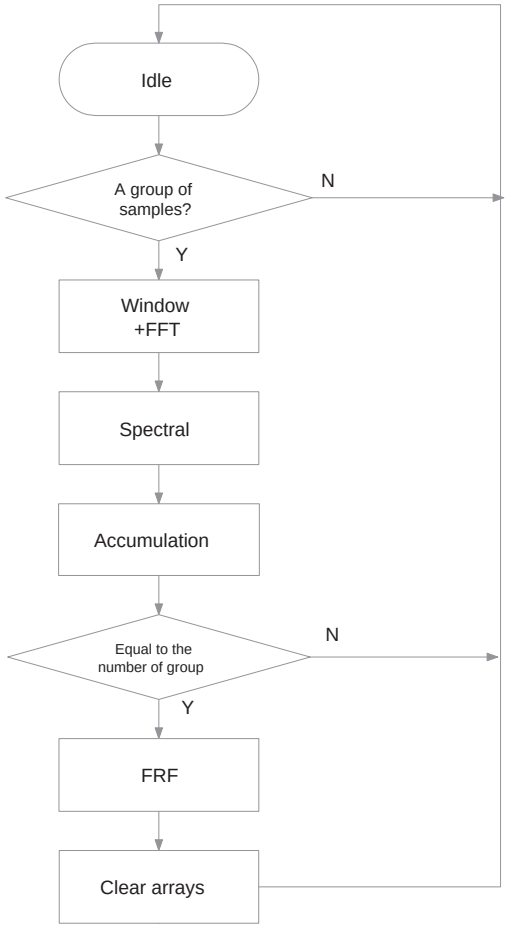


Figure 3.3.: Program Flow Diagram

In transmission stage, DMA0 and DMA1, employed to carry samples from ADC to circular buffer in L3, work invisibly and independently to processor. Both work modes of DMA0, in Autobuffer mode, and DMA1, in Script mode, are only need to set up

once when the whole system initializing and then following setup works automatically and repeatedly all the time. There is no necessity for processor to be noticed until the completed window size of samples are collected in circular buffer. Then the processor start to move out data from circular buffer and prepare for FRF calculation of a new group samples. The flow chart is show in figure 3.4.

The various of FFT algorithms which have been developed for its evaluation use the complex number and complex arithmetic. In this application, all input data are pure real sequences, therefore, only half of works should be needed.[Hvs 87] By exploiting symmetric property of real transform, half memory spaces and calculations could be saved. However, because of limited spaces, where it is impossible to directly calculate 8192 point FFT on L1 memory, the algorithm need to be change to adapt this situation. The 8192 points FFT needs to be separated into two part: eight 1024 points FFTs and FFT merge which uses specified way to combine small FFT results into a bigger one. The first part adopt 2-radix in place FFT algorithm, where the complexity is $O(nlogn)$, whereas complexity of FFT merge, which is inefficient but unavoidable, is $O(n^2)$.

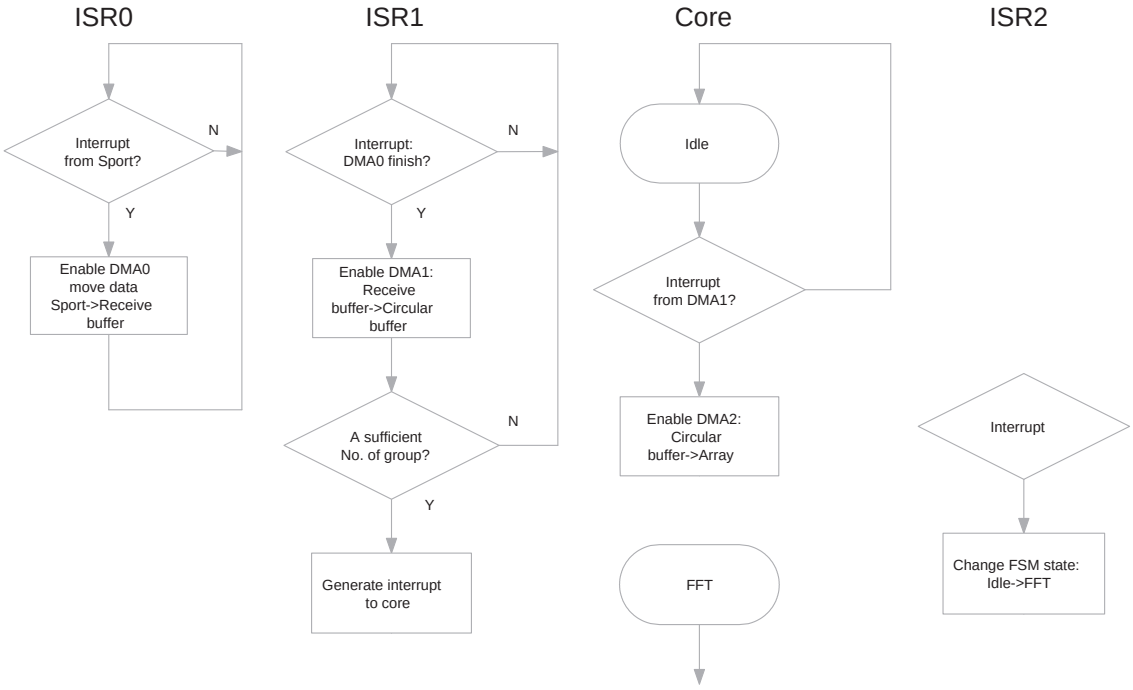


Figure 3.4.: Transmission Flow Chart

In turn, the spectral densities are computed as multiplied the result of FFT of stimulus signal and response signal. And in accumulation part, corresponding results of spectral are added together until the sufficient the number of windows arrive. Both of complexities of this two parts are linear to $O(n)$.

Real-time calculation of FRF involves thousands of division operations because the estimate algorithm given is based upon the division between cross-spectral density(CSD) and power-spectral density(PSD). To accomplish such divisions as fast as possible, we need high performance division algorithms. Among them Goldschmidt's method turns out to be a reasonable choice. In order to accelerate the convergence, approximation method finds its way based on series expansion. In this method, an initial estimation of the quotient is gradually refined until desired precision is achieved. The operations are then converted to additions and multiplications, which are fast and easily available in present hardware platforms. Approximation method can reach a quadratic convergence rate. The complexity of this part is $O(n)$.

4. Floating-Point Implementation

4.1. Hardware Setup

The experiment environment consists of three parts, as plotted in figure 4.1. The lap-

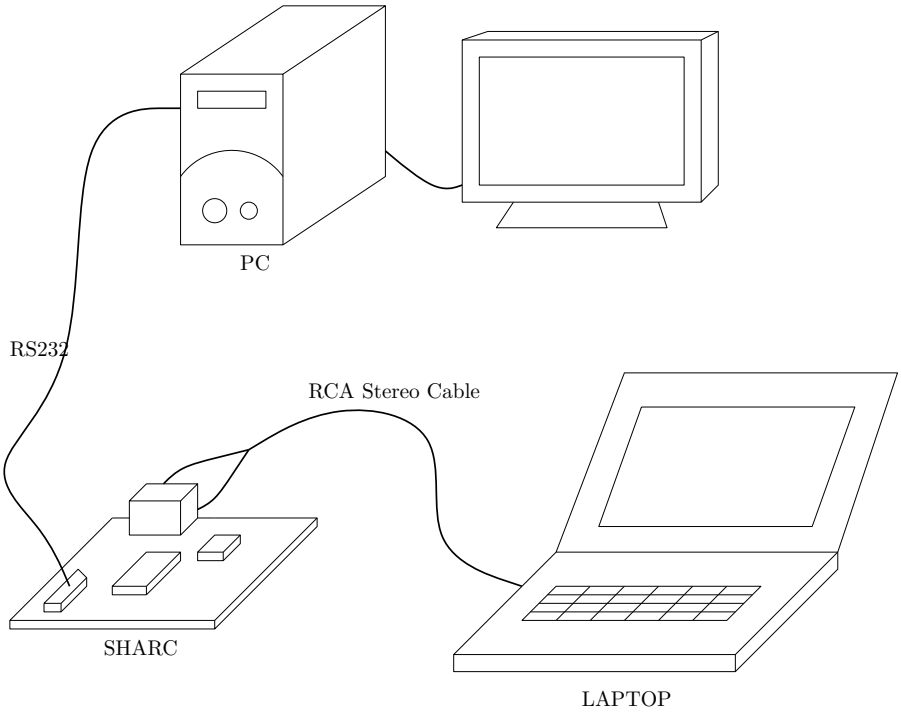


Figure 4.1.: Experiment Setup

top functions as the signal generator using soundcard driven by Matlab audioplayer functions. The analog output of the soundcard is fed through the stereo cable into the conditioning and analog-to-digital converter(ADC) circuit of the ADSP-21489 SHARC development board. The processor then upload the results to the monitor PC using a RS232 cable. Meanwhile the users are expected to use the monitor software on the PC for calibration and observation.

The ADC on SHARC development board has 24-bit word length and 192kHz sampling rate, which are of the same configuration as the DAC in the soundcard of the laptop. The ADSP-21489 digital signal processor is a single-instruction multiple-data(SIMD)

harvard floating point processor [Shc 09] for audio applications. The processor's floating point unit completes a single-precision multiplication in one core cycle(2.5ns); a single-precision division in three core cycles. Besides that it features with a hardware fast fourier transform(FFT) accelerator. The on-chip 5M bits memory space allows to keep large window of samples internally on-chip without overhead of swap between the SDRAM memory. This advantage enables a straight forward algorithm design and maintains a fast execution speed. Moreover, quite a lot of peripheral direct memory access(DMA) channels result in an easy and efficient timing control of the hardware.

4.2. Software Design

The program is written in ANSI C programming language. Moreover each function component has been designed for maximum portability and reusability. The control structure has maintained a simple design—it only contains a single ADC interrupt service routine, which writes the samples into circular buffers. Outside of the ADC ISR, the main function fulfills the main calculation tasks, as shown in the flow diagram 4.2 below. The ADC ISR of the program is invoked whenever the event of the AD conversions of the three channels occurs at every sampling instant. The background loop in the main function detects whether a full window of samples is formed or not, if so the program starts to perform the FFT and spectral density calculations, and averaging spectral densities over several windows to reduce the estimation variance. In the final window the FRF will be derive by divisions of the averaged spectra.

One of the best data structures for holding the overlapping and delayed samples from the ADC would be circular buffers. They are implemented in arrays with modulo indexing. The samples in the circular buffers will then be copied to other arrays for FFT and spectral calculations, because during the calculation process the circular buffers are continuously being written by new samples from ADC.

Although the SHARC processor features with a hardware FFT accelerator, unfortunately this pipelined structure, working with both of a source and sink array, is memory hungry. What we need is an in-place algorithm minimizing the memory costs so that larger window of samples can be calculated within internal memories. Based on this requirement, it is more reasonable to run the in-place FFT routine using the processor core instead of the accelerator to halve the memory usage.

Since the ADC samples are all real numbers, and the fourier transform of real numbers yields complex conjugate frequency results. So only single sided spectra and FRF are needed for computation. The single sided fourier transform for real numbers is called real valued fourier transform. We have adopted the real split radix FFT to minimize the number of operations[Pdh 83, Hvs 86, Hvs 87, Pdm 90, Sgj 07, Mfs 05]. The twiddle factors are stored as a sine table without calculating from scratch. The com-

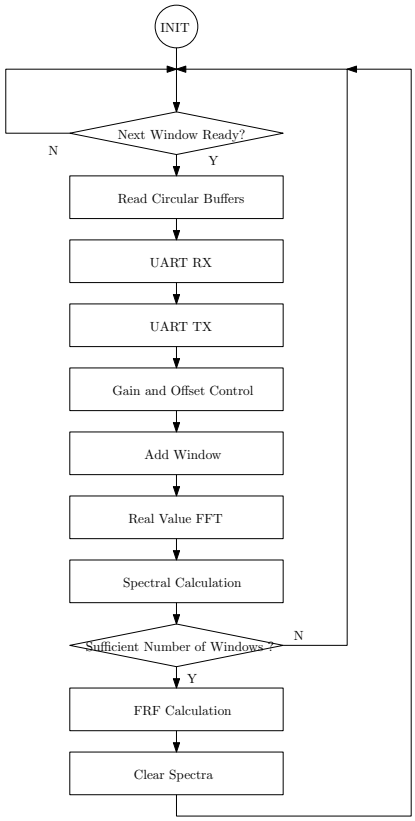


Figure 4.2.: Program Flow Diagram

plexity for this part is $O(nlogn)$.

The spectral densities are computed as multiplications of the fourier transform of windowed stimulus and response, as described in chapter 2. The complexity is linear $O(n)$. The final division of the spectra is also of $O(n)$ complexity. To boost the performance, these loop-based non-dependency code lines could be optimized by the compiler according to the SIMD structure [Spr 09]. In SIMD executions the loop will be vectorized by a factor of two as the core fetches two words at a time [Srl 09].

The results will be transmitted through the UART to a PC using RS232 cables. The command to determine the runtime state is also downloaded to the SHARC board by the same duplex transmission line. The UART DMA improves the concurrent executions of the processor core and peripherals, thus gives a simpler and better timing control, which is illustrated in figure 4.3. In this example diagram, the number of samples for one window has been scaled down to 4, and the FRF is averaged over 2 windows.

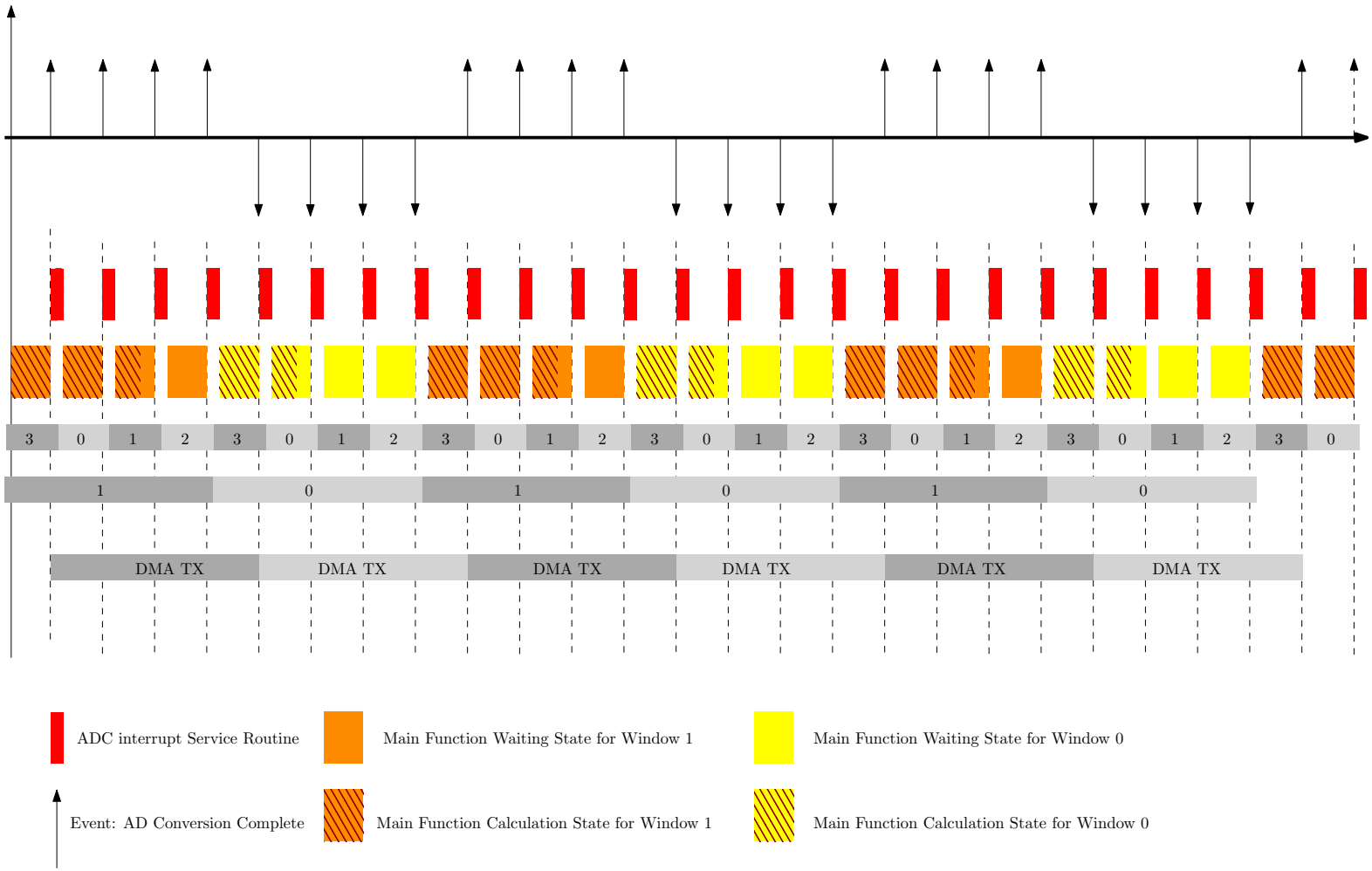


Figure 4.3.: Timing Diagram

4.3. Calibration

Calibration is an important procedure before any instruments starting their measurements. The underlying principle is to compare the instrument with standards to validate the coming measurements. In our case, the only means of gain control of the analog paths is the analog mixer circuit of the laptop soundcard, which would never reach a precise match of the stereo channels. Our solution to this problem is thus the additional digital bias and gain compensations in the DSP program: the ADC samples of each channel are added with biases and multiplied by coefficients adjustable from the monitor software on the PC. Aside from calibration purpose, the digital gains would help scale the data into the numeric range of better numeric precisions as well. So to

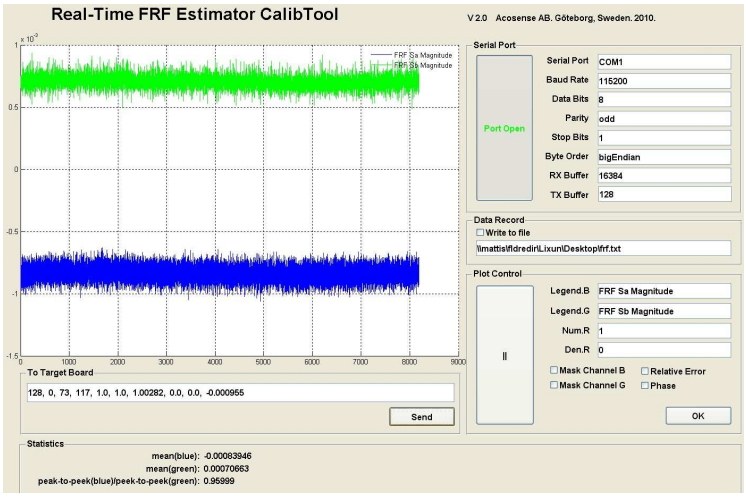


Figure 4.4.: Stimulus and Response Samples Before Calibration

calibrate the estimator, we should set proper bias and gain values when given common standard analog signals for stimulus and response in order to balance the total gains between them. We use a common zero volt reference and a sinusoidal signal to help adjust the bias and the gain respectively. There are three steps for calibration. Firstly the samples from the ADC should be biased to zero(which could be observed from the monitor software) when given zero volt analog signals. The samples of the stimulus and response before any calibrations are depicted in figure 4.4. We can see that they are not reside on zero level but contain small DC biases. Hence we add the negated values to stimulus and response digitally in the hope of shifting them to zero level. The result after proper bias compensation is shown in figure 4.5. The statistical values have been provided by the monitor software in its lower portion. Secondly, the stimulus and response samples should have the same gain when given a common sine wave analog input. This sine wave meanwhile helps identify the safe range of the input analog

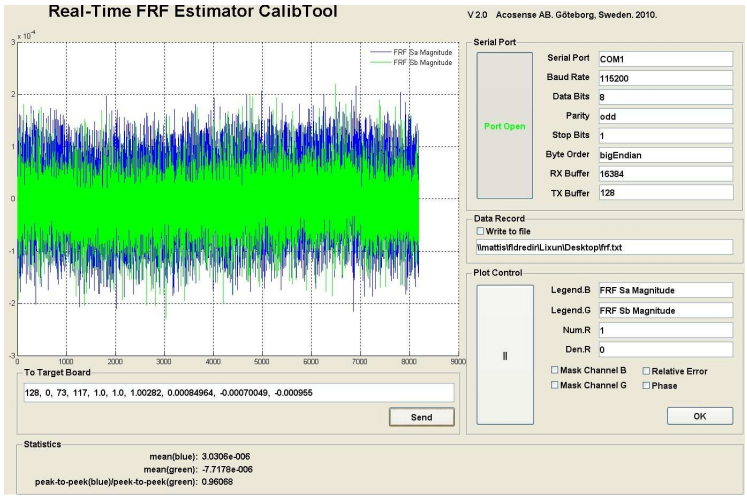


Figure 4.5.: Samples After Proper Bias

signals, preventing from cases of saturation failure. The pre and afterward gain adjustments have been shown in figure 4.6 and 4.7. We could read the peek-to-peek ratios under the display canvas of the calibration tool. The statistical values including mean

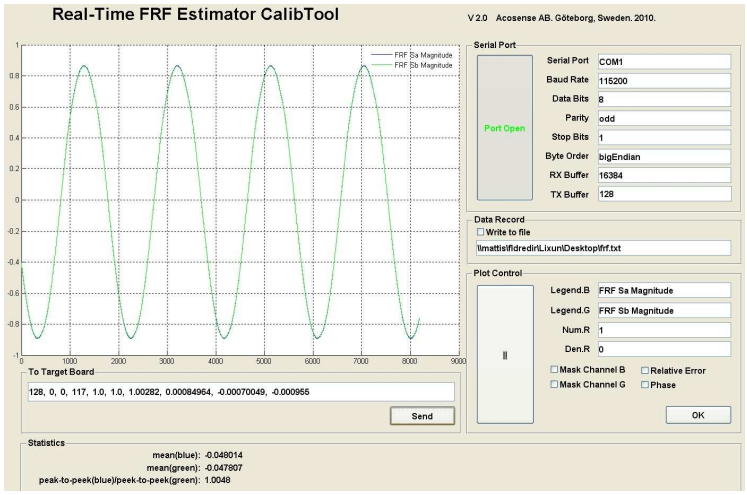


Figure 4.6.: Stimulus and Response Samples Before Gain Adjustment

and peak-to-peak values, given by the monitor software are derived from each window of sampled data and averaged over several windows to reduce the variances. The complexity of these statistical functions are approximately $O(n)$, which is acceptable for embedded real-time calculations. In other words, these functions could be incorporated into the DSP program in the future to realize the auto-calibration features. The

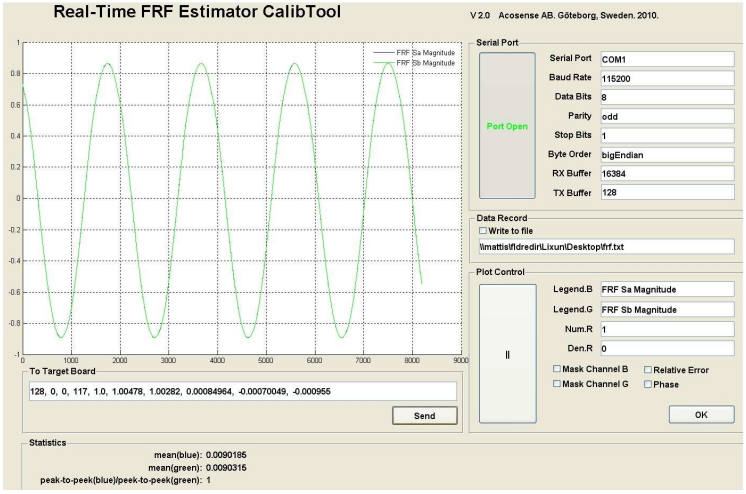


Figure 4.7.: Samples After Proper Gain Compensation

final step is to make time domain correlation analysis for impulse response of the system using noise test signals. The necessity of this step has been described in chapter 2. Propagation delay of the sound wave could be estimated based on the characteristics of the system's impulse response in time domain. Accordingly the delay values mea-

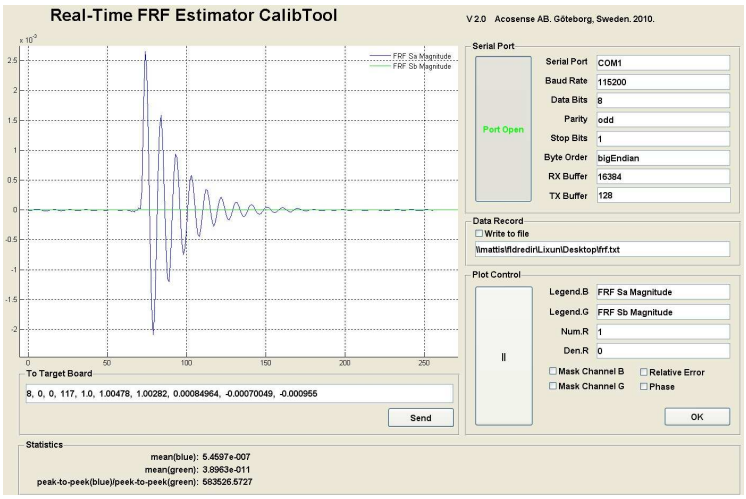


Figure 4.8.: Impulse Response Estimate

sured in number of samples, would be used to realign the windows of the stimulus and response inside the DSP program. The impulse response is calculated as the cross correlation of the stimulus and response from lag 0 to lag 256 within each window and averaged over several number of windows, shown in figure 4.8. The prerequisite is that

the stimulus should be white noise. Although the stimulus signal is approximate white time series, for applications of delay estimation there is no need for extra whitening filters. The correlation algorithm has been carried out in time domain, which has a run-time complexity of nearly $O(n^2)$. Therefore we could not calculate for a large number of lags for lengthy windows. So far the estimator is ready for FRF estimation, demon-

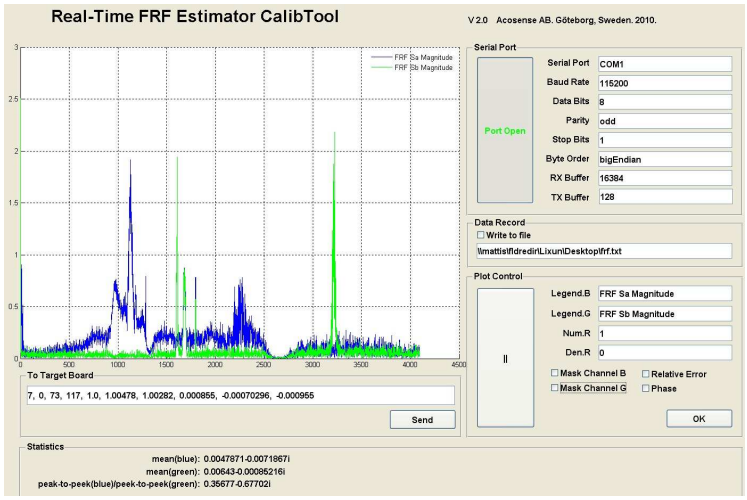


Figure 4.9.: Ready To Estimate FRF

strated in figure 4.9. With optimization of speed turned on for the SIMD structure, the floating point platform is able to calculate two FRFs with 8192 point window. The duty ratio is 35.1% at 400MHz core speed, and the internal memory usage is over 90%. The duty ratio without optimization is around 70%. The improvement of the execution speed is contributed by the vectorization of the loops for SIMD structure.

4.4. Verification

Verification would be the last step of calibration since this step justifies the correctness of both analog-and-ADC subsystem and the digital calculations. To verify the estimate precisions for certain standard LTI systems, we use second order high-pass and low-pass filters as the reference systems, whose accurate FRFs could be acquired by "freqz" function provided in Matlab. The stimulus and response samples are generated by Gaussian white noise and the reference system output. There are totally four test items: (0)The lowpass reference system:

$$H_{lp}(z) = \frac{1}{1 - 1.5z^{-1} + 0.9z^{-2}}$$

(1)the highpass reference system:

$$H_{hp}(z) = \frac{1}{1 + 1.5z^{-1} + 0.9z^{-2}}$$

(2)the delayed version of H_{lp}:

$$z^{-117} H_{lp}(z)$$

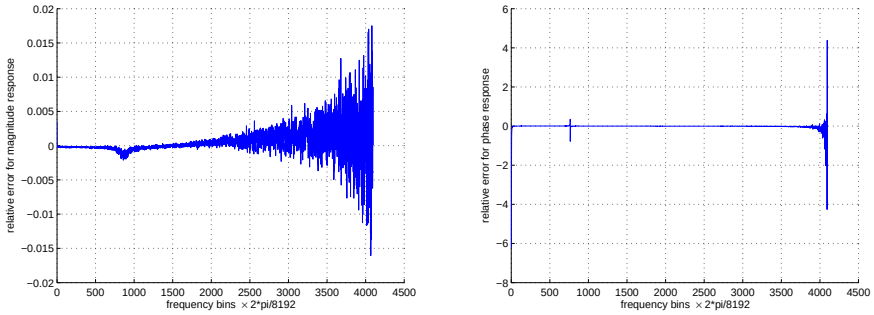
(3)the delayed version of H_{hp}:

$$z^{-73} H_{hp}(z)$$

All these tests are exercised after proper calibrations described in previous section. The window size is 8192 point; no overlapping samples; number of windows is 200; window function is hann function. In order to test the ability of calculating two FRFs within the same window period, the low-pass filter was tested by one FRF channel and the high-pass filter by the other simultaneously.

4.4.1. Lowpass Filter Without Delay

Estimate $H_{lp}(z) = \frac{1}{1-1.5z^{-1}+0.9z^{-2}}$. Since there is no delay for this model, we don't have to calculate the impulse response to find out number of delayed samples. The relative errors for magnitude and phase response are plotted in figure 4.10. We can see that the error increases as the frequency increases. We have observed a $\pm 2\%$ error for the



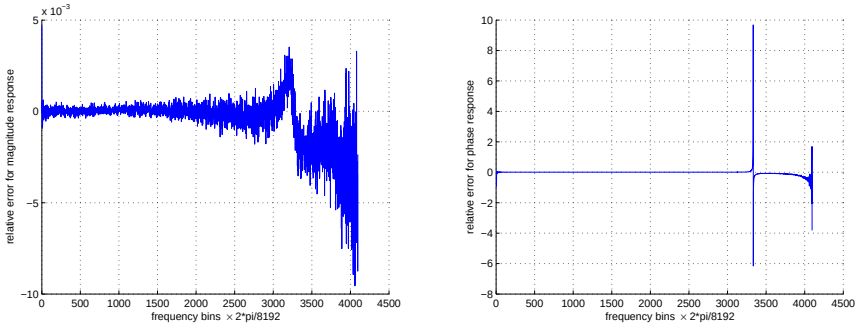
(a) Magnitude (b) Phase

Figure 4.10.: Relative Errors for the Low Pass Reference Filter

worst case around π. These data could help identify which frequency band has the best accuracy. After all in real application situations, users would only expect credible FRF at frequency bins much lower than the sampling rate. From figure 4.10, the most reliable estimate lies in frequency bins from zero to $\pi/2$. There are three locations where the largest phase errors emerge: the zero frequency, also known as the DC component; the frequency of the peak of magnitude; and π. The locations have correspondence with the magnitude errors.

4.4.2. Highpass Filter Without Delay

Estimate $H_{hp}(z) = \frac{1}{1+1.5z^{-1}+0.9z^{-2}}$. The relative error for magnitude and phase response have been shown in figure 4.11. Again the most credible frequency bins are from DC



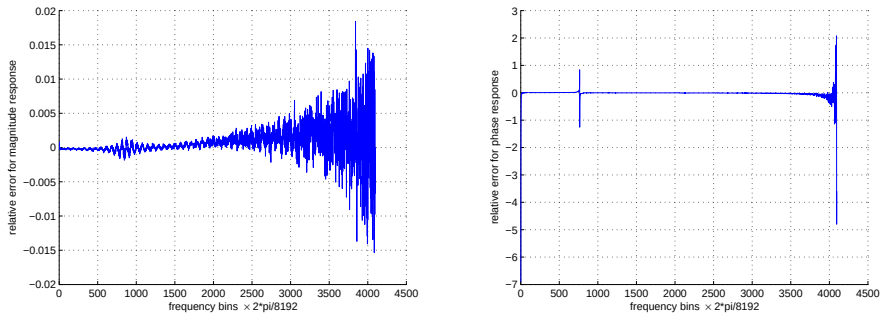
(a) Magnitude (b) Phase

Figure 4.11.: Relative Errors for the High Pass Reference Filter

component to $\pi/2$, where an average of $\pm 0.03\%$ error exists. The large error for the DC components may originate from the bias error in calibration phase. Besides the accuracy drops as the frequency increases.

4.4.3. Lowpass Filter With Delay

Estimate H_{lp} with 117-sample delay stage. To determine the number of samples of delay, we should estimate its impulse response before the estimate of the FRF. We could read the delay value from the result, given in figure 4.13. The estimated delay value will



(a) Magnitude (b) Phase

Figure 4.12.: Relative Errors for Delayed Low Pass Reference Filter

be used as the shift values to choose proper window of samples for prospect calculation.

The relative errors of magnitude and phase response have displayed in figure 4.12. We can see that the magnitude errors have a $\pm 2\%$ worst case as well, the same as the worst case in non-delayed test. Without delay compensation on the other hand will give

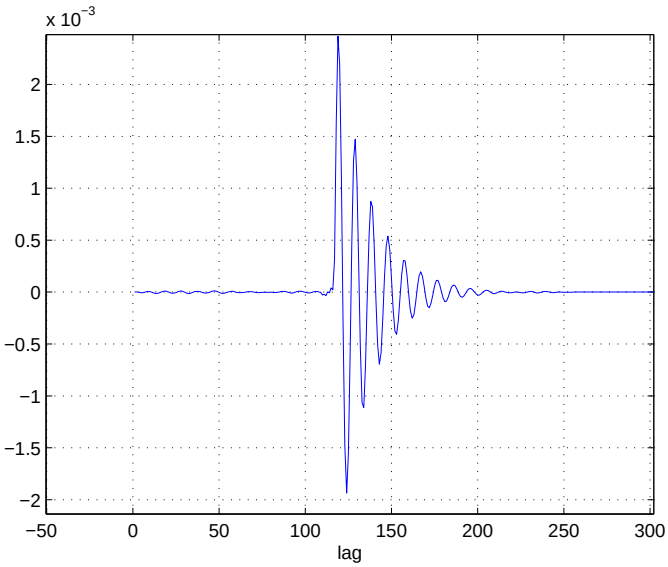
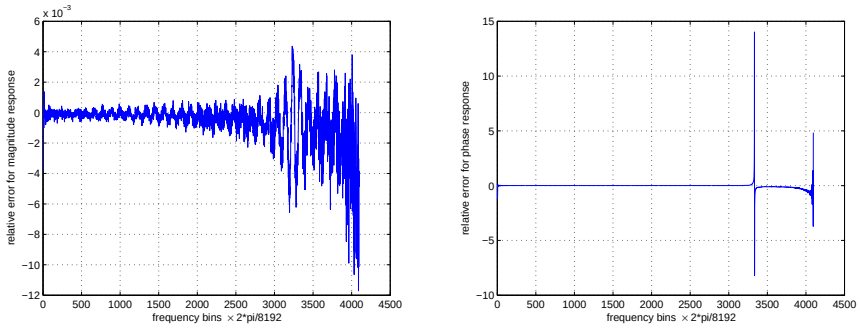


Figure 4.13.: Impulse Response for Delayed Low Pass Reference Filter

worse estimates.

4.4.4. Highpass Filter With Delay



(a) Magnitude (b) Phase

Figure 4.14.: Relative Errors for the Delayed High Pass Reference Filter

Estimate H_{hp} with 73-sample delay stage. The impulse response is again been estimated and we could find out the exact delay from the graph 4.15. The separate delay

control for both channels requires more memory allocations than a common delay value for both channels. This is also the reason why we eliminate the redundant estimation methods like H_2 and H_r to trade for space. The magnitude and phase relative errors are given in figure 4.14. We could expect an equal estimate quality compared with the

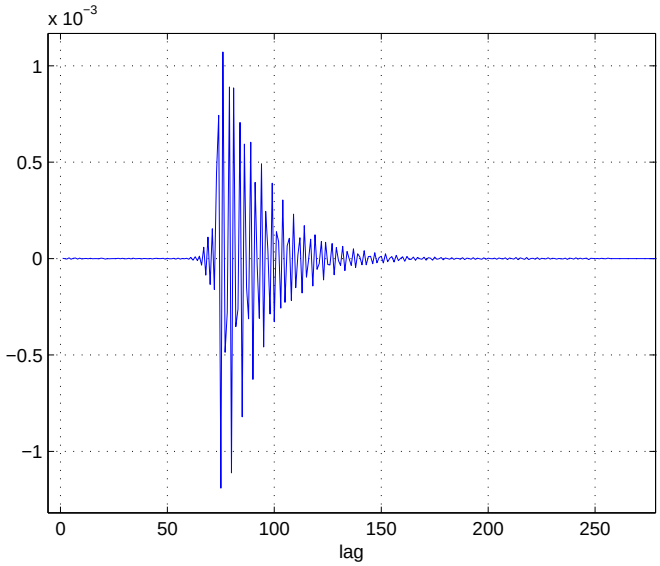


Figure 4.15.: Impulse Response for the Delayed High Pass Reference Filter

non-delay version.

In this chapter we have introduced the hardware platform for the prototype, as well as the structure of the software. Many crucial design considerations and trade-offs have been elaborated on. In addition to those, the calibration and verification process have also been covered in detail. Several test cases are demonstrated to justify our design. The performance of the SHARC platform will be benchmarked against the Blackfin for speed and precisions in the final chapter.

5. Benchmarks and Conclusion

We have noticed that the analog-to-digital converters of the Blackfin and the SHARC platform differ in sampling rates. To make a better benchmark between those two, we decided to bypass the analog paths and the AD conversion and calculate the FRF with the user-case stimulus and response data pre-stored in off-chip SDRAM. In other words, our benchmarks only compare the digital implementations for precision and speed. The automated benchmark procedure is depicted in figure 5.1. When doing the

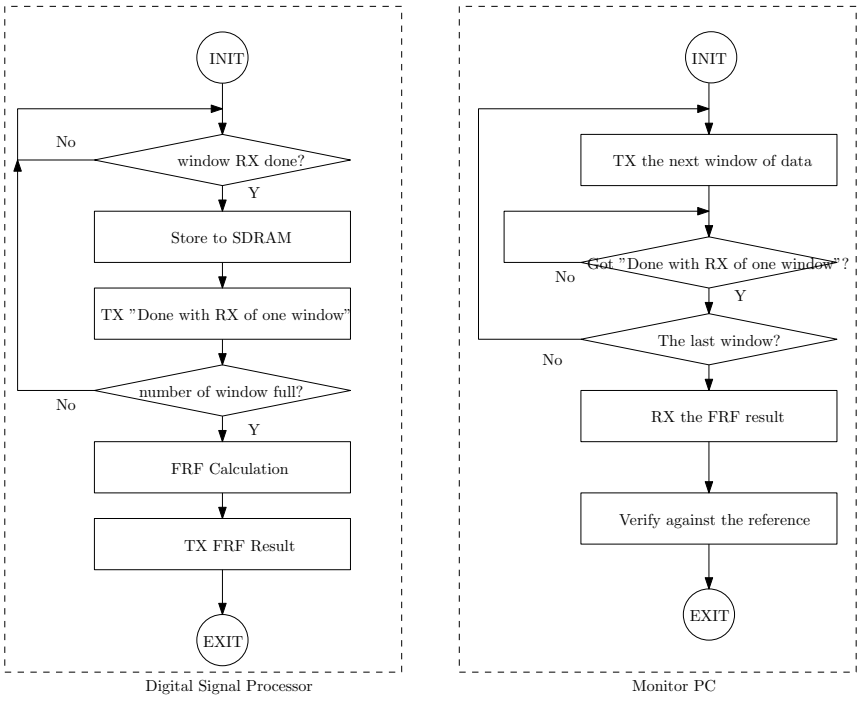


Figure 5.1.: Automated Benchmark Process

benchmark, the monitor software on the PC sends predefined number of window of reference stimulus and response data to the DSP processors through the asynchronous serial transmission cable. The window length is 8192, and a total of 8, 20 and 100 windows of reference data have been tested. The data set is so large that they should be stored in the SDRAM of the embedded system before to be calculated. UART RX and TX DMAs are used to simplify the communications. When calculations have com-

pleted, the processors will transmit the FRF data back to the monitor software, where the result will be processed and compared against the correct result. Time costs of each function block of the DSP program will also be measured by core timers. The cycles are averaged over the number of windows used.

5.1. Timing

Both of the implementations have met their timing requirements, as shown in table 5.1 and 5.2. It should be pointed out that we were using non-overlapping windows for

SHARC Platform		
Functions	Cycles(CCLK:400Mhz)	Timing Average
Moving 8192-point stimulus and response from SDRAM to internal memory	average:375925 worst:376004	average:939.8125us worst:940.01us
Add windows to stimulus and response	32790	81.975us
FFT of 8192-point stimulus and response	1295337	3238.3425us
Derive S_{xx}, S_{xy} and accumulate	49233	123.0825us
Division	22531	56.3275
Total		
FRF Calculation		4439.515us
For $f_s = 192$kHz Max allowed time for FRF is 42666.6667us		Timing Satisfied

Table 5.1.: Timing of the SHARC platform

the benchmark calculations. The duty ratios on the other hand, determines the maximum number of overlapping samples between subsequent windows. The bottleneck of the Blackfin platform mainly involves the block data moving from SDRAM to internal memory, and the merge step, which has $O(n^2)$ complexity. The bottleneck of the SHARC platform involves moving data from off-chip memory to internal and FFT, which are less intensive than its Blackfin counterpart. Since the SDRAMs require extra timing control for refreshment, the access delay deviates over time, thus averaged values are preferred. There are several reasons that could explain the larger duty ratio of the Blackfin platform.

Firstly, the memory subsystem structure of the Blackfin platform forced us to adapt the algorithm into memory distributed solution, which slows down the total speed.

Blackfin Platform		
Functions	Cycles(CCLK:600Mhz)	Timing Average
Moving 8192-point stimulus and response from SDRAM to internal memory	319396	532.32664us
FFT 1024×8 and window stimulus and response	11823335	197705.558us
Merge eight 1024-point FFT into one 8192-point FFT	4166359	6943.9318us
Spectra S_{xx}, S_{xy}	367978	613.2967us
Accumulation($\sum S_{xx}, \sum S_{xy}$)	375003	625.00499us
Spectra and Accumulate	742981	1238.30167us
Scaling	72569	120.94833us
Division and Reset	1167718	1946.1967us
Total		
FRF Calculation		30486.129us
For $f_s = 96$kHz Max allowed time for FRF is 85333.3333us		Timing Satisfied

Table 5.2.: Timing of the Blackfin platform

This could be justified by the large amount of time in the merge step of the algorithm. On the other hand however, we could focus on optimization of merge algorithm for better performance.

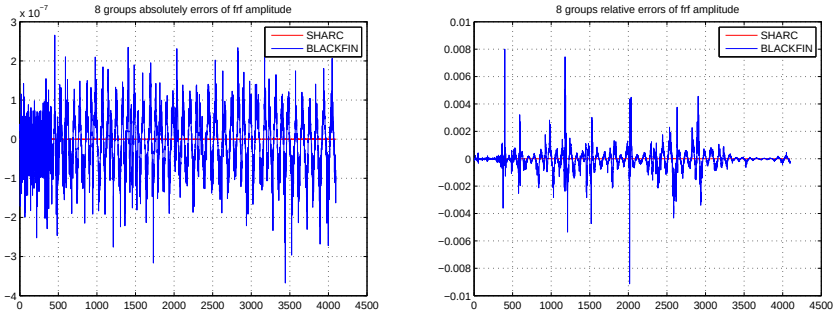
Secondly the dual-core quad-ALU structure of the Blackfin was not fully utilized since only single core was assigned to the computation. We believe that the total latency could be halved if the other core is used for the block algorithm. Parallelism might be increased because of the parallel nature of the block algorithm.

Lastly the hardware data path for the Blackfin processor is 16 bit, whereas the application requires 32×32 bit fixed-point multiplication for decent numeric accuracy. In spite of the dual 16×16 multipliers for each processor core, the extension of the width of the data path leads to an overhead in merge the outputs of the two multipliers when doing the 32×32 bit multiplications. The accumulation of the small overhead of this most frequently used operation compromised the 600MHz clock advantage.

5.2. Precision

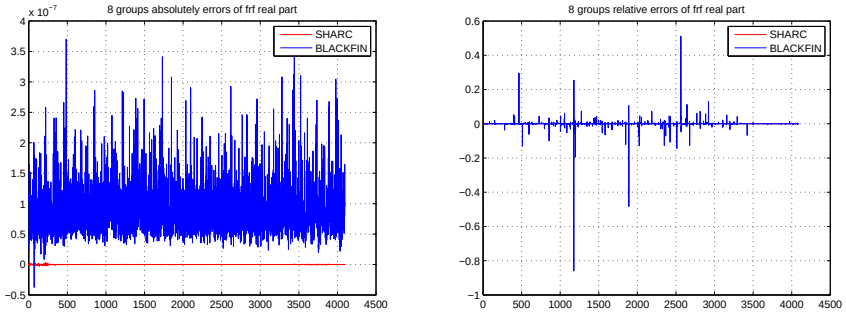
The numeric precisions of the FRF have been tested and results are plotted in figures below. The basic data type for the Blackfin processor is 32-bit fixed point number, and

for the SHARC 32-bit single precision floating point number. To maximize the speed, the Blackfin processor did not use emulated floating point number system, but rather the ordinary means by keeping track of the binary point. The SHARC processor, whose single precision floating point number covers a larger numeric range with regard to the fixed point processor, still has to be scaled the sampled data in order to find regions of better accuracies. In the benchmark tests of different number of windows, the error decreases as the number of windows increases. For 8 window case, the values of the accumulated spectra are small, which in turn results in a considerable numeric error in the final divisions. The increase of the windows augments the values of the spectra, thus decreases the division errors for the derivation of FRF. So in short, no matter fixed point numbers or floating point numbers, scaling is a crucial part for data processing.



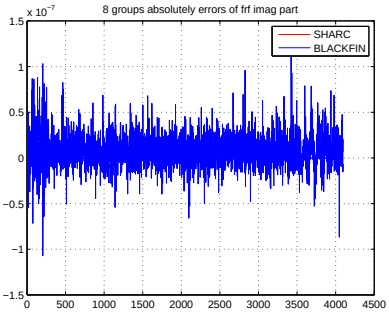
(a) Absolute Error (b) Relative Error

Figure 5.2.: Magnitude Errors, $N_{window} = 8$

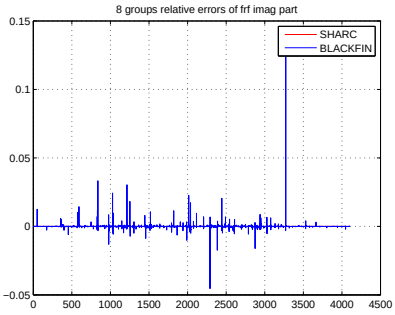


(a) Absolute Error (b) Relative Error

Figure 5.3.: Errors of Real Parts, $N_{window} = 8$

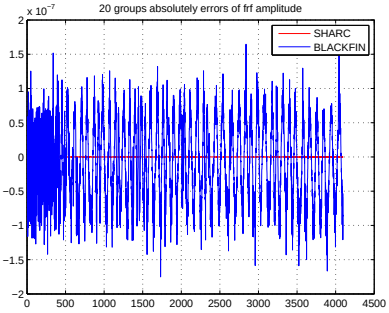


(a) Absolute Error

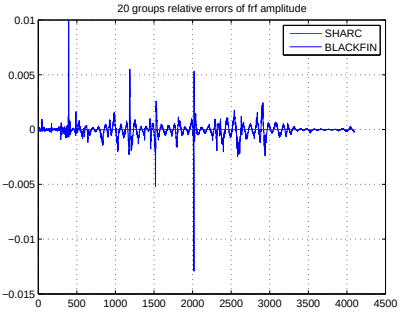


(b) Relative Error

Figure 5.4.: Errors of Imaginary Parts, $N_{window} = 8$

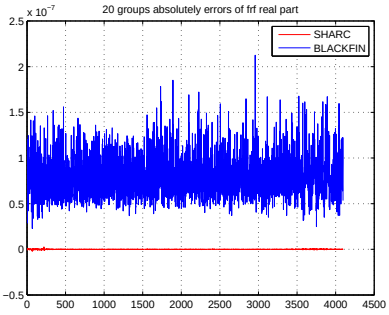


(a) Absolute Error

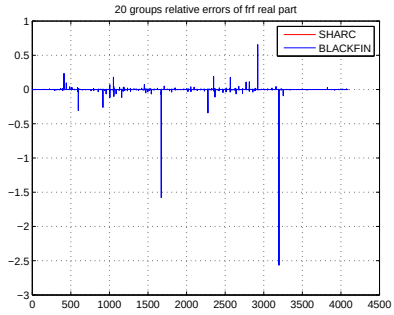


(b) Relative Error

Figure 5.5.: Magnitude Errors, $N_{window} = 20$

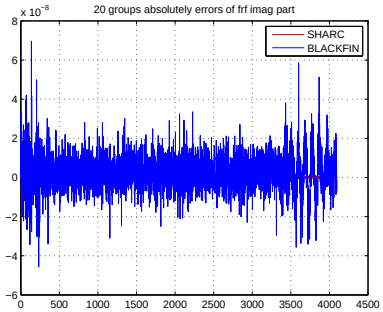


(a) Absolute Error

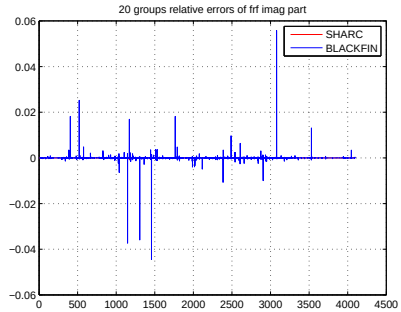


(b) Relative Error

Figure 5.6.: Errors of Real Parts, $N_{window} = 20$

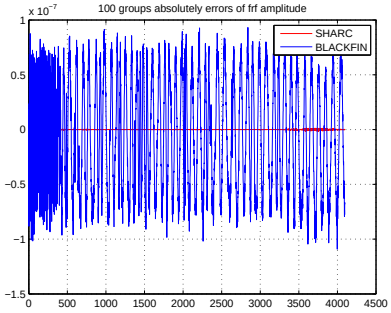


(a) Absolute Error

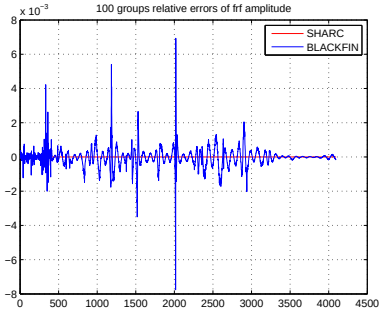


(b) Relative Error

Figure 5.7.: Errors of Imaginary Parts, $N_{window} = 20$

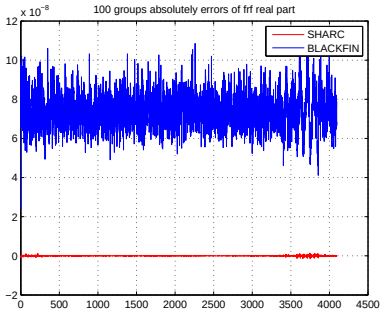


(a) Absolute Error

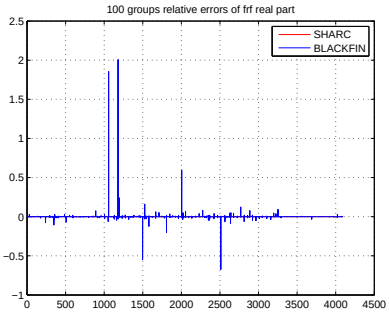


(b) Relative Error

Figure 5.8.: Magnitude Errors, $N_{window} = 100$

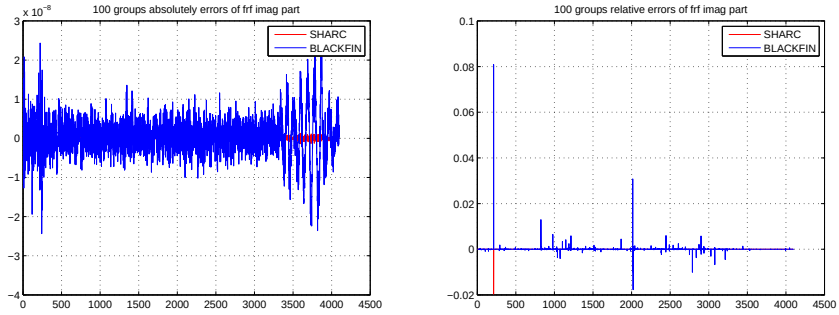


(a) Absolute Error



(b) Relative Error

Figure 5.9.: Errors of Real Parts, $N_{window} = 100$



(a) Absolute Error (b) Relative Error

Figure 5.10.: Errors of Imaginary Parts, $N_{window} = 100$

5.3. Summary

The goal of the thesis has been fulfilled since we managed to find out a suitable customized FRF estimation algorithm for active acoustic spectroscopy sensors, as well as to make successful implementations and evaluations. The benchmark results imply that for signal processing tasks of large windows and wide dynamic range, large tightly coupled memory and floating point unit might play a more crucial role than other factors. Due to the limited amount of time allocated for the project, the development and optimization of the algorithm and hardware is not complete and sufficient, thus the benchmark itself can only be regarded as certain reflections or suggestions of the "golden" solution. There will be a long way to converge to the perfect solution as several factors can further be explored in the future: (a)the system to be estimated is in fact nonlinear; (b) other spectral density/FRF estimation approaches that lead to faster and more accurate spectral/FRF estimations; (c) implementations other than digital signal processors.

References

[Hys 96] Monson H. Hayes: *Statistical Signal Processing and Modeling*, Wiley, 1996

[Mkv 09] Tomas McKelvey: *SSY130 Applied Signal Processing Course Notes*, Chalmers University of Technology, 2009

[Apv 06] Alexander Potchinkov: *Measurement of frequency responses of nonlinearly distorted SISO systems in noisy environments with generalized parameter frequency response estimators*, Signal Processing, vol.86(8), pp2094-2114, 2006.

[Ayt 77] A. F. Seybert: *Time delay bias errors in estimating frequency response and coherence functions*, Journal of Soundand Vibrations, vol 60(1), pp1-9, 1978

[Whw 09] Wang Hongwei: *FFT Basics and Case Study using Multi-Instrument*, Application notes, Virtins Technology, 2009

[Pwh 67] Peter D. Welch: *The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short modified periodograms*, IEEE trans. Audio and Electroacoust. vol AU-15, pp.70-73, 1967

[Shc 09] Analog Devices: ADSP—2146x SHARC Processor Hardware Reference, Analog Devices Inc. Rev0.2, 2009

[Spr 09] Analog Devices: SHARC Processor Programming Reference , Analog Devices Inc. Rev2.0, 2009

[Srl 09] Analog Devices: Run-Time Library Manual for SHARC Processors, Analog Devices Inc. Rev1.3, 2009

[Pdh 83] P. Duhamel and H. Hollmann: *'Split Radix' FFT Algorithm*, Electron. Lett. Volume 20, Issue 1, p.14V16, 1984

[Hvs 86] Henrik V. Sorensen: *On Computing the Split-Radix FFT*, IEEE transactions on acoustic, speech and signal processing, Vol.ASSP-34, No. 1, 1986

[Hvs 87] Henrik V. Sorensen: *Real-Valued Fast Fourier Transform Algorithms*, IEEE transactions on acoustic, speech and signal processing, Vol. ASSP-35, No. 6, 1987

[Pdm 90] P. Duhamel and M. Vetterli: *Fast Fourier Transforms: A Tutorial Review and A State of the Art*, Signal Processing, Vol.19, pp.259-299, 1990

[Sgj 07] Steven G. Johnson and Matteo Frigo: *A modified split-radix FFT with fewer arithmetic operations*, IEEE Trans. Signal Processing, Vol.55 (1), pp.111V119, 2007

[Mfs 05] Matteo Frigo and Steven G. Johnson: *The Design and Implementation of FFTW3*, Proc. IEEE, vol. 93, no. 2, pp. 216V231, 2005

[Bds 09] Analog Devices: Data Sheet Final: ADSP-BF561 Blackfin Embedded Symmetric Multiprocessor, Rev D, Analog Devices Inc. 2009

[Bhr 10] Analog Devices: ADSP-BF561 Blackfin Processor Hardware Reference, Analog Device Inc. Rev1.2, 2010

[Api 09] Randy Yates: *Fixed-Point Arithmetic: Introduction*, Digital Signal Labs, 2009

[Clm 05] Analog Devices: VisualDSP++ 4.0 C/c++ Compiler and Library Manual for Blackfin Processor, Analog Device Inc. Rev 3.0, 2005

A. Proofs

1. Fourier transform of unity.The Fourier transform of the delta function is one.

$$\int_{-\infty}^{\infty} \delta(t) e^{-j\omega t} dt = 1$$

So the inverse transform should give a delta function back:

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} e^{j\omega t} d\omega = \delta(t)$$

As a result we have

$$\int_{-\infty}^{\infty} e^{j\omega t} d\omega = 2\pi\delta(t)$$

To calculate the Fourier transform of unity,

$$\int_{-\infty}^{\infty} e^{-j\omega t} dt$$

we change variable as $p = -t$, so the equation becomes

$$-\int_{\infty}^{-\infty} e^{j\omega p} dp = \int_{-\infty}^{\infty} e^{j\omega p} dp = 2\pi\delta(\omega)$$

2. Proof of the Parseval equation for previous equation.

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} W(\omega) W^*(\omega) d\omega$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} \left[\sum_{m=-\infty}^{\infty} w(m) e^{-j\omega m} \right] \left[\sum_{n=-\infty}^{\infty} w(n) e^{-j\omega n} \right]^* d\omega$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} \left[\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} w(m) w^*(n) e^{j\omega(n-m)} \right] d\omega$$

given $n - m = k$, we have

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} \left[\sum_{k=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} w(n-k) w^*(n) e^{j\omega k} \right] d\omega$$

$$= \sum_{k=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} w(n-k)w^*(n) \left[\frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\omega k} d\omega \right]$$

$$= \sum_{k=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} w(n-k)w^*(n)\delta(k)$$

$$= \sum_{n=-\infty}^{\infty} w(n)w^*(n)$$

$$= \sum_{n=-\infty}^{\infty} |w(n)|^2$$

B. Matlab Scripts

FRF estimation function:

```
% FRF estimate
% Note that the 'stimulus' and 'response' are column vectors
function [frf] = myfrf(stimulus,response,win,N,K,V);

%V = 0.5; % overlap coefficient
%N = 8192;  % window length
%K = 10;  % number of windows

if win == 'hamming'
    W = hamming(N);
elseif win == 'hanning'
    W = hann(N);
elseif win == 'bartlett'
    W = bartlett(N);
elseif win == 'blackman'
    W = blackman(N);
else
    W = ones(N,1);
end;

for k = 0:(K-1)
    x(:,k+1) = stimulus( (k*N*(1-V)+1) : (k*N*(1-V)+N) ).*W;
    y(:,k+1) = response( (k*N*(1-V)+1) : (k*N*(1-V)+N) ).*W;
end

xomega = fft(x);
Pxx = xomega.*conj(xomega);
Pxx = sum(Pxx.');

yomega = fft(y);
Pxy = conj(xomega).*yomega;
Pxy = sum(Pxy.');
```

```
frf = (Pxy./Pxx).';  % two sided spectrum
frf = frf(1:N/2); % convert to single-sided normalized spectrum
```

Fixed point Goldschimdt division:

```
%------------------------------------------------------------------------
% Fixed-point Goldschimdt division --- reciprocal function
%
% Goldschimdt algorithm provides a high throughput divison
% based on series expansion and pipeline, for more info
% please refer to document "pipelined division"
%
% The divisor must be scaled to [0.5 1) before division's taken place,
% so the Q format would be Q(0.WORDLENGTH-1)
% The reciprocal will be (1 2], which is of Q(2.WORDLENGTH-3)
%
% April 26, 2010
%------------------------------------------------------------------------
% x : input scalar in format Q(0,WORDL-1)
% y : reciprocal output scalar in format Q(2.WORDL-3)
% LUT : precalculated M-bit-address lookup vector for initial guess
% WORDL : word length scalar
% M : address width of the lookup table, scalar

function [y] = mydiv(x, LUT, M, WORDL)

% since WORDL-LSB-2 = M bit for the lookup table
% the map function would be
map = floor(x/(2^(WORDL-M-2)))-2^M+1;

% nominator is 1
n(1) = 2^(WORDL-2);    % Q(1,WORDL-2)

% initial guess of the reciprocal value
n(1) = n(1)*LUT(map);  % Q(1,WORDL-2)*Q(1.WORDL-2)=Q(3,2*WORDL-4)
n(1) = floor(n(1)/(2^(WORDL-1)));  % Q(3,2*WORDL-4) to Q(2, WORDL-3)
if n(1) >= 2^(WORDL-1) ||  n(1) <= 0
    disp('Overflow Occurs');
```

```
end

% initial calculation of the divisor
d(1) = x*LUT(map);  % Q(0,WORDL-1)*Q(1.WORDL-2)=Q(2,2*WORDL-3)
d(1) = floor(d(1)/(2^(WORDL-1))); % convert Q(2,2*WORDL-3) to Q(1,WORDL-2)
if d(1) >= 2^(WORDL-1) || d(1) <= 0
    disp('Overflow or underflow');
end

%converge within N steps
N = 3;
for k = 1:N
    d(k+1) = d(k)*(-d(k)-(-2*(2^(WORDL-2))));  % Q(1,WORDL-2)*Q(1.WORDL-2)=Q(3,2*WORDL-4)
    d(k+1) = floor(d(k+1)/(2^(WORDL-2)));    % convert Q(3,2*WORDL-4) to Q(1,WORDL-2)
    if d(k+1) >= 2^(WORDL-1) || d(k+1) <= 0
        disp('Overflow or underflow');
    end

    n(k+1) = n(k)*(-d(k)-(-2*(2^(WORDL-2))));  % Q(2,WORDL-3)*Q(1.WORDL-2) = Q(4,2*WORDL-5)
    n(k+1) = floor(n(k+1)/(2^(WORDL-2)));  % Q(4,2*WORDL-5) to Q(2, WORDL-3)
    if n(k+1) >= 2^(WORDL-1) || n(k+1) <= 0
        disp('Overflow or underflow');
    end

end

y = n(N+1);

%------------------------------------------------------------------------
% for test only
% y = [n; d];
%------------------------------------------------------------------------




%------------------------------------------------------------------------
% Fixed-point Goldschimdt division --- testbench
%
```

```
% Goldschimdt algorithm provides a high throughput divison
% based on series expansion and pipeline, for more info
% please refer to document "high throughput division"
%
% The divisor must be scaled to [0.5 1) before division's taken place,
% so the Q format would be Q(0.WORDLENGTH-1)
% The reciprocal will be (1 2], which is of Q(2.WORDLENGTH-3)
%
% The most important design parameters are WL and MSB for lookup table.
% Because convergence loses precision due to the truncated procuts,
% to increase the lookup table means better initial guess and less
% precision loss.
%
% April 26, 2010
%--------------------------------------------------------------------------


clear all;
close all;
clc

% reciprocal investigation
% n/d, for denominator d, we pre scale it into the range of [0.5 1)
% in q0.15 format
% i.e. "01yy-yyxx-xxxx-xxxx..."
WL = 18;
MSB = 6;
LSB = WL-2-MSB;
RES = 2^(1-WL);

% use cases
d = 0.5:RES:(1-RES);  % note that 1 is open range
dQ = d*(2^(WL-1));    % in Q(0,WL-1) format
ref = 1./d;           % reference reciprocal values in floating format
refQ = floor(ref*(2^(WL-3))); % in Q(2,WL-3) format

% initial demoninator relies on LUT
% 10-bit LSB look up table gives increment 1024
% 4-bit lookup table
dLUT = [ ];
```

```
for k = (2^LSB):(2^LSB):length(d)
    dLUT = [dLUT 1/d(k)*(2^(WL-2))];  % in Q(1.WL-2)
end

% find out the errors
 for k = 1:length(d)
     result(k) = mydiv(dQ(k), dLUT, MSB, WL);
 end
plot(refQ-result);
title('Error analysis of reciprocal in Q(2.13) format for all Q(0.15) divisors in [0.5  1)');
xlabel('divisor in Q(0.15) scaled into [0.5  1)');
ylabel('error');
grid;

%-------------------------------------------------------------------------
% test only
%-------------------------------------------------------------------------
% test = mydiv(dQ(1398), dLUT, MSB, WL);
```