CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

A Fast Radio Burst (FRB) Capture System for the Onsala Space Observatory

Master's thesis in Embedded Electronic System Design

Bocheng Jia
Xiao Chen

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

A Fast Radio Burst (FRB) Capture System for the Onsala Space Observatory

Bocheng Jia
Xiao Chen

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

A Fast Radio Burst (FRB) Capture System for the Onsala Space Observatory
Bocheng Jia, Xiao Chen

Supervisors:
Gary Hovey, Onsala Space Observatory
Per Larsson-Edefors, Department of Computer Science and Engineering

Examiner:
Lena Peterson, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

A Fast Radio Burst (FRB) Capture System for the Onsala Space Observatory
Bocheng Jia, Xiao Chen
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

The discovery of the fast radio burst (FRB) in 2007 was an important moment in astronomy due to FRB's unique property, which helps astronomers explore how the universe expanded and gives clues to study celestial phenomenons such as the appearance of neutron stars. The purpose of this thesis is to explore and develop a new digital backend system for Onsala Space Observatory to capture FRB events in real time and reduce the data rate. A trigger mechanism that can flag the FRB candidates is implemented. Only when potential FRB candidates are detected, signal data would be buffered and saved. We implemented polyphase filterbank channelization to generate the signal's spectrum in real time. Moreover, we implemented incoherent de-dispersion algorithms to compensate for the frequency-dependent delay. Filtering and averaging are introduced to improve an FRB's signal-to-noise ratio. We first built the system model in MATLAB and analyzed the simulation results. We also demonstrated the hardware implementation of the FRB capturing system and tested the system with a signal generator. The test results showed that our system is able to capture FRB events and reduce the data rate compared to the current system in Onsala Space Observatory.

Keywords: Fast Radio Burst, Radio astronomy, Incoherent dedispersion, Polyphase filterbank channelizer, FPGA, Data rate reduction.

Acknowledgements

We want to thank the Onsala Space Observatory for offering us this precious master project, and providing technical equipment. We also would like to give sincere thanks to our supervisors: Gary Hovey and Per Larsson-Edefors. They provide constructive help in academic area and technical areas, guiding us to complete this thesis. In addition, thanks to Lukas Sandström, Franz Kirsten, and Olof Forssén for introducing more insights about radio astronomy.

This thesis project is a challenging project, we want to thank all the people who have ever helped us, it is your support that make this project come true.

Bocheng Jia, Xiao Chen, Gothenburg, October 2021

Contents

1

Introduction

Fast radio bursts (FRBs) are transient signals caused by some high-energy astrophysical processes not yet understood. The first FRB was discovered in 2007 by Lorimer et al. [1], since then many FRBs have been observed and retrospectively identified in previously recorded data.

An FRB is an extremely strong transient pulse with a duration from one millisecond to a few milliseconds. The energy of an FRB is exceptionally high, and it has been estimated that the average FRB energy in a millisecond is as much as the sun emits in three days [2]. Additionally, an FRB pulse is dispersed over 100's of MHz, with low frequencies arriving up to a second behind the high frequencies. While the dispersion is due to electrons in interstellar space, the origin of FRBs is still under investigation and is of great scientific interest.

FRBs are an important subject in astronomy. This project aims to assist astronomers in capturing FRBs with higher efficiency and accuracy by improving the hardware and algorithms of existing systems. By measuring the frequency-time structure of a pulse (dispersion measure) and the properties of the interstellar space, researchers can infer information about the distance to a source and the interstellar medium (ISM). Such information contributes to our understanding of how the universe expanded over its history, as well as to our knowledge of other unknown phenomena, like the 'dark energy' and 'dark matter' in the universe [3].

1.1 Propagation characteristic: dispersion

Free electrons in cosmic space cause lower frequency electromagnetic waves to propagate more slowly than higher frequency waves. Hence, a frequency-dependent delay is added to FRBs, which is a function of distance and free electron density [4]. Some examples of dispersed FRB pulses and their frequency-time structure are shown in Fig. 1.1.

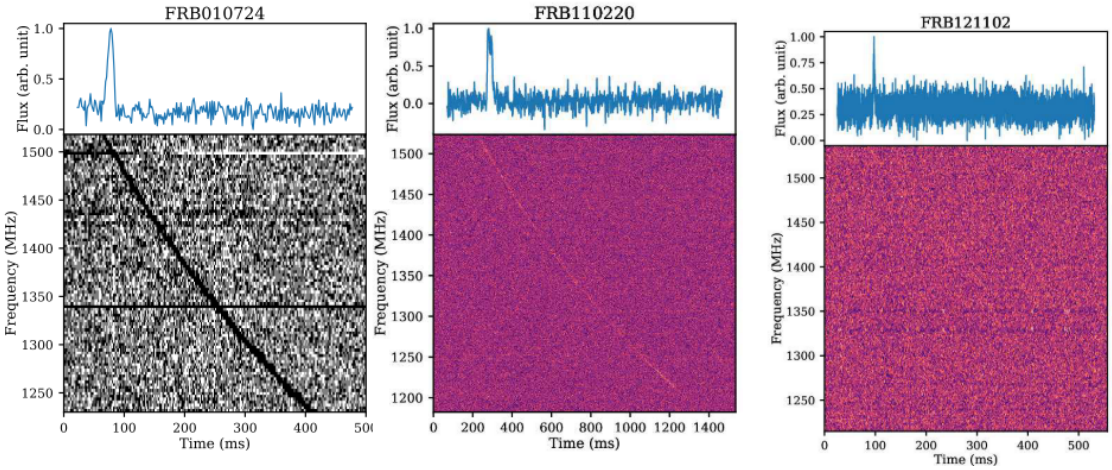

Figure 1.1: The pulse profile (top) and frequency-time spectrum (bottom) of the dispersed FRB010724, FRB110220, and FRB121102 [5]. Note that the prominent frequency-dependent delay in the leftmost plot (FRB010724) and the weak SNR (FRB121102) in the rightmost plot is improved by de-dispersion (i.e., the pulse profile)

.

In each plot, the lower (waterfall) sub-plot shows the frequency sweep across the time-frequency plane. In contrast, the upper sub-plot shows the total pulse intensity after removing the quadratic dispersion sweep (i.e., de-dispersion).

The dispersion measure, DM, is a common metric used to relate electron density and distance to the dispersed frequency-time structure of a pulse [6].

The extra delay, t, added at a frequency v is

$$t = k_{DM} \times (\frac{DM}{v^2}) \tag{1.1}$$

Here, DM is the dispersion measure and k_{DM} is the dispersion constant, which is given by

$$k_{DM} = \frac{e^2}{2\pi m_e c} \approx 4.149\,\mathrm{GHz^2\,pc^{-1}\,cm^3\,ms} \tag{1.2}$$

In this expression, m_e is the mass of the electrons, and c is the speed of light. The dispersion measure DM can be derived by:

$$DM = \int_0^d n_e dl \tag{1.3}$$

where n_e is the density of electrons, and d is the distance from the FRB source to the Earth. The unit of DM is $\mathrm{cm^{-3}}$ pc [1].

[1]The parsec (pc) is a unit of length used to measure the large distances to astronomical objects outside the Solar System, approximately equal to 3.26 light-years.

Note that for a pulsed signal like an FRB, the DM can be calculated using the time between the highest and lowest frequencies, v_{hi} and v_{lo}, as:

$$\Delta t = k_{DM} \times DM \times (\frac{1}{v_{lo}^2} - \frac{1}{v_{hi}^2}) \tag{1.4}$$

After rewriting the above equation, we can determine DM by measuring pulse arrival times at different frequencies. This is important because most celestial objects radiate continuously, and as such their DM cannot be computed this way. The strong transient nature of FRBs makes them extremely useful probes to measure both the character of the source, the distance to it, and the nature of the electron density in the intervening cosmic medium.

1.2 Current FRB capture system

The FRB capture system currently used in Onsala Space Observatory is a VLBI (very-long-baseline interferometry) recorder, which consists of a Digital Baseband Converter 2 (DBBC2) hardware and the Heimdall software [7][8].

The DBBC2 is a digital backend that has been endorsed by the European VLBI network and is used as a digital processing unit in radio telescopes. Its primary purpose is to downconvert radio signals to baseband and perform digital signal processing on the data before writing it to disks. Note that the DBBC2 assumes the SNR is very low so that the data may be re-quantized to 2 bits to save transmission bandwidth and disk space. A block diagram of DBBC2 is shown in Fig. 1.2. Table 1.1 summarizes some of the key specifications of DBBC2.

Figure 1.2: A block diagram of the DBBC2 processing flow.

Table 1.1: Specification of DBBC2.

Bandwidth	4× IF-512/1024 MHz
Record data rate	4.096/8.192 Gbps
Data width	2 bits

Heimdall is a fast transient search software that can use graphics processing units (GPUs) for FRB searching. It has been deployed on many different radio telescopes as a real-time transient detection system to discover FRBs. A data flow chart of how Heimdall works is shown in Fig. 1.3.

The software reads the data from the DBBC2 hardware, and Fourier transforms it to frequency space. Digital signal processing algorithms such as de-dispersion,

Figure 1.3: Flow chart of Heimdall.

whitening, and matched filtering are then applied to the received data. The FRB candidates are detected by examining the matched filter output. Finally, the software outputs and displays the FRB candidates.

However, the VLBI recorder has some shortcomings: Firstly, it re-quantizes the data to 2 bits to reduce the data rate and file size. This can only be done for SNR's $\ll$ 1, but when the SNR is > 1, non-linearities cause spurious signals and distortions. Secondly, FRB events cannot be identified in real time, and post-processing is required to detect FRB events. Lastly, it is difficult to maintain the DBBC2 as it uses custom hardware that is not well documented, and it is impractical to modify its firmware to better reject radio frequency interference (RFI), flag FRB events in real time, change data rate, and requantized data.

1.3 Project goal

The goal of this project is to develop a new backend based on commericial off-the-shelf DSP system whose firmware and software can be modified as needed to monitor FRBs as well as pulsars (which also emit transient pulses similar to FRBs). We break down this goal into several parts as follow:

- Investigate existing FRB capture systems, to understand the advantages and disadvantages of the existing FRB systems, and use them as references for designing our system.
- Investigate FRB detection algorithms, compare their pros and cons, and choose the appropriate algorithms to use in our system according to our needs.
- Build the system model based on the algorithms we chose in MATLAB, simulate each algorithm, and explore the influence of the parameters in the algorithm on our system.
- Implement the algorithms and functions as needed using Xilinx Vivado, and test the performance of the hardware FPGA system.

1.4 Thesis outline

Apart from this introduction, the thesis consists of five chapters. Chapter 2 mainly introduces the principles of the signal processing algorithms involved in the FRB capture system. Chapter 3 presents the design and simulation of the system, while chapter 4 introduces the hardware implementation of the system. Chapter 5 presents our test scheme and the test results. The final chapter is the conclusion.

2

Theory

As mentioned in chapter 1, a significant feature of FRBs is that the signal is highly dispersed. Thus, the most important part of the digital signal processing chain is to remove the distortion by moving all the FRB's frequencies to a reference frequency. Hence, before the de-dispersion algorithm can be applied, the broadband input signals must be divided into many narrow frequency bins (or channels). This is done using a channelizer. This chapter will give a brief introduction to the basic principles of the signal processing algorithms such as the de-dispersion and channelizer algorithms, as well as some other algorithms which will be used in the system.

2.1 Sampling

Many FRBs are detected around 1400 MHz, and a few are detected at lower frequencies in the range of 400-800 MHz[9]. According to the Nyquist-Shannon sampling theorem [10], to fully recover or reconstruct the original signal after sampling, the sampling frequency should be greater than twice the signal's highest frequency. Hence, to sample the FRB signals without aliasing and distortion, we need a sampling frequency at around 2.8 GHz ($2 \cdot 1400$ MHz). Although the current high-speed analog-to-digital converters (ADCs) can reach such a high sampling rate, it makes the computational load for the digital signal processing system extremely high and results in unnecessary resources waste.

Bandpass sampling theorem is then introduced to handle these problems. For a bandpass signal with a frequency range (f_L, f_H), the sampling rate must fulfill the following equation from Nyquist sampling theorem [11]:

$$fs \geq 2(f_H - f_L) \tag{2.1}$$

And the acceptable sampling rate is given by [11]:

$$\frac{2 \cdot f_H}{m+1} \leq fs \leq \frac{2 \cdot f_L}{m} \tag{2.2}$$

Here f_L and f_H stand for the lowest and highest frequency of the signal, respectively. m is the maximum integer that fulfills the requirement of the above equation. Therefore, it is more efficient to sample the FRB signal at twice its bandwidth than at twice its highest frequency.

2.2 Channelizer

A channelizer is used to divide a broadband signal into multiple narrow channels and downconvert each output channel to baseband, so one or more channels may be extracted.

The following subsections explain the function of a channelizer in more detail. The terms used are defined as follows: M is the total number of channels, k is the channel index range from 0 to $M-1$, D is the decimation ratio, and $\omega(k)$ is the center frequency of each channel.

2.2.1 Channel division

In general, there are two allocations for the channels: odd stacking and even stacking [12]. Fig. 2.1 illustrates the difference between these two different channel allocations.

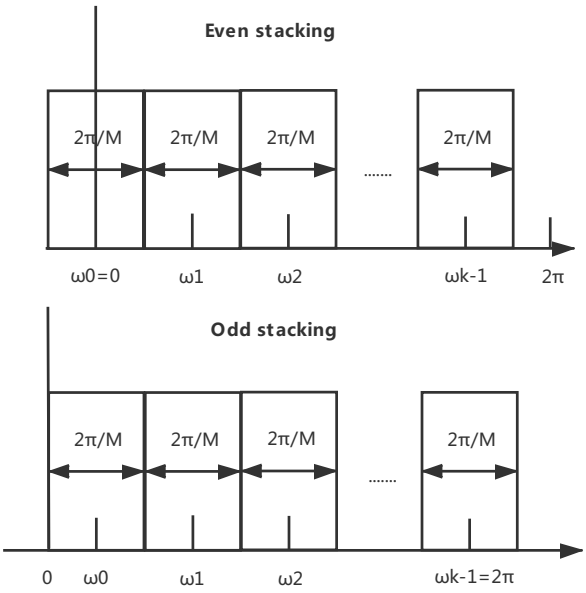

Figure 2.1: Channel allocation. The upper part shows an even-stacked channel allocation and the lower part shows an odd-stacked channel allocation. The bandwidth of each sub-channel is $2\pi/M$.

For even stacking, the center frequency of each channel can be calculated as:

$$\omega(k) = \frac{k\pi}{M} \tag{2.3}$$

In contrast, for odd stacking, the center frequency is calculated as:

$$\omega(k) = \frac{2k+\pi}{M} \tag{2.4}$$

2.2.2 Discrete Fourier transform (DFT) channelizer

Typically, a channelizer operates by shifting the desired band to baseband, attenuating out-of-band signals, and finally decimating the signal to a narrower bandwidth [12]. The channelizer structure based on this is commonly known as the discrete Fourier transform analyzer. Fig. 2.2 illustrates a channelizer with four channels. Here, different shapes represent different frequency components.

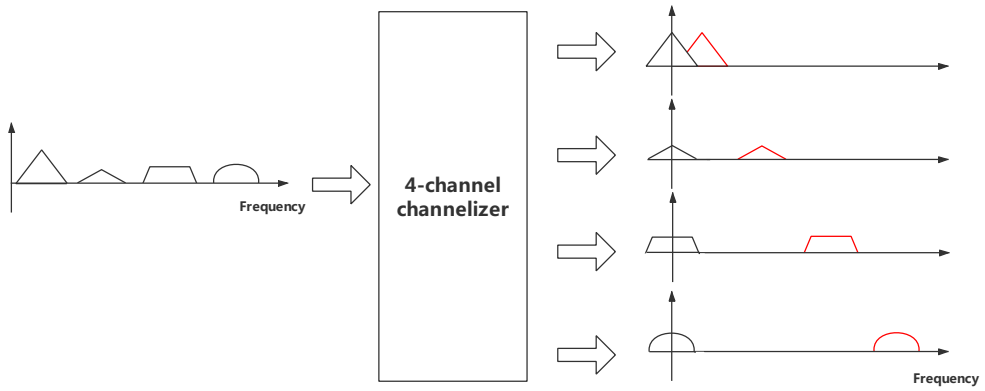

Figure 2.2: Illustration of a 4-channel channelizer.

The structure of a DFT channelizer is shown in Fig. 2.3. In this figure, n represents the undecimated samples and m represents the decimated sample. $\omega_k, k \in [0, M-1]$, is the Fourier frequency used to downconvert the signal to baseband, and $H(z)$ is the discrete time filter transfer function.

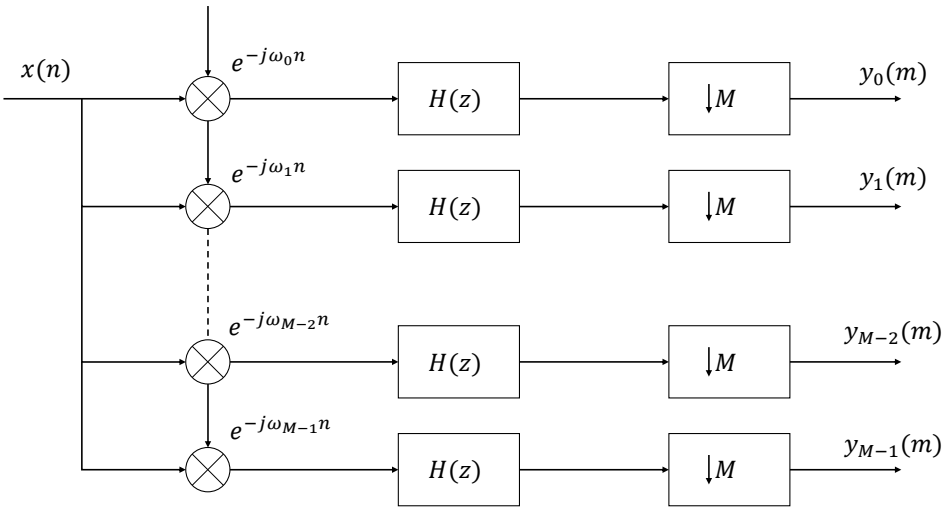

Figure 2.3: Conventional DFT channelizer structure.

In this approach, M independent 1-channel channelizers are working in parallel. Each separate channel extracts one desired band from the broadband input. The

advantage of using this straightforward approach is the high degree of flexibility in selecting each channel's center frequency and bandwidth. Meanwhile, the bandwidth of each channel are not required to be uniform. However, this approach is inefficient (more resources, power, cost) for high-sampling-rate applications, for the reason that every sample, n, is processed by M filters, $H(z)$, at the full sample rate, and down-converted by M shift frequencies $e^{-j\omega_k n}$.

2.2.3 Polyphase filterbank (PFB) channelizer

A more efficient method than the DFT channelizer is to decimate first and then process m samples by M filters and downconvert using the fast Fourier transform (FFT). This approach has order $n \log_2 n$ operations instead of n^2 (where n stands for data length) saving $\frac{\log_2 n}{n}$ operations over a DFT channelizer. Such a channelizer is known as a polyphase filterbank, or PFB, because it processes the decimated phases of the signal in separately, and typically downconverts and integrates using the FFT. The term filterbank refers to it having an identical channel bandwidths.

Since the PFB channelizer decimates the input samples first and then processes them, it is very suitable for application in FPGAs because it can handle the clock speed mismatch between the fast ADCs and slower FPGAs: the highest clock rate that most of the FPGAs could run at is typically hundreds of megahertz while modern ADCs can reach a very high sampling rate up to several Giga samples per second. In this project, for example, the ADC clock rate is 5 GHz while the FPGA clock rate is 1/16 of that, 312.5 MHz.

A typical PFB structure is shown in Fig. 2.4. This kind of PFB channelizer is also known as the critically sampled PFB channelizer because the channel complex output data rate is the same as the channel bandwidth. In other words critically sampled..

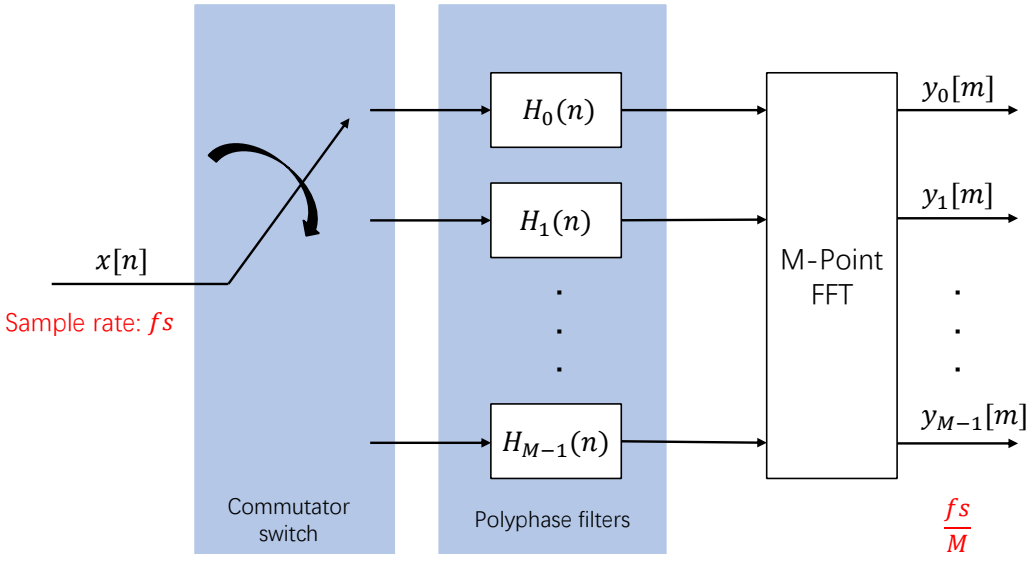

Figure 2.4: Critically sampled PFB channelizer structure.

The first stage of a PFB channelizer structure is a commutator switch where the decimation of factor M is applied. It starts on the first branch as shown in Fig. 2.4 and travels through these branches in the clockwise direction. The $H_k(n)$ in the second stage represents the polyphase filters which are formed by decomposing a prototype filter into M sub-filters. Here, the prototype filter is a low-pass filter with a bandwidth of $\frac{1}{M}$. An example of the polyphase filter coefficients are obtained for a prototype filter is shown in Table 2.1. This table assumes that the coefficients of prototype filter are: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] and the number of channels is $M = 4$.

Table 2.1: Arrangement of PFB.

Polyphase filters	Prototype filter coefficients
$H_0(n)$	0, 4, 8, 12
$H_1(n)$	1, 5, 9, 13
$H_2(n)$	2, 6, 10, 14
$H_3(n)$	3, 7, 11, 15

Each polyphase decomposed filter takes one coefficient from the prototype filter in order until all coefficients of the prototype filter are taken away.

2.2.4 Oversampled PFB channelizer

There are various types of channelizers, each with its own set of characteristics. Two commonly used channelizers are the critically sampled PFB channelizer and oversampled PFB channelizer. The former one (critically sampled channelizer) is the basic one where the bandwidth of each channel is exactly $\frac{1}{M}$ of the original signal spectrum, and the decimation ratio D is equal to the number of channels M. Therefore, the output sample rate is the same as the channel spacing. However, because of aliasing, frequencies outside the channel (from noise or strong signals) will fold back into the channel, placing constraints on the prototype filter pass, transition and stop band. To avoid this problem in critically sampled channelizers, one can create a wider guard band between signals on adjacent channels, and another way is to design a higher order prototype filter so that a sharper transition band can be achieved and hence reducing the frequency overlap between a filter and its images [13]. However, the guard band would reduce the useful bandwidth of the signal, and the higher-order filter would increase the resource usage.

The oversampled PFB channelizer can avoid reducing the useful signal bandwidth and minimizing the frequency aliasing efficiently by making the Nyquist frequency of the channel, $\frac{fs}{D}$, greater than the channel spacing $\frac{fs}{M}$ so that the channel filters are overlapping in terms of the input signal, but their images do not overlap, and therefore, do not cause aliasing after decimation [14].

To summarize, a critically sampled channelizer has $M = D$ while an oversampling channelizer has $M > D$. The ratio between M and D is known as oversampling ratio.

For an oversampled PFB channelizer, there are M outputs for every D inputs. To increase the number of samples at the output, the number of input samples to the polyphase filter should be increased accordingly. Therefore, there will be a $M - D$ overlapping input samples to the PFB. Assume, for example, that the number of channels M is 8 and the decimation ratio D is 4; the resulting arrangement of samples feeding into the PFB is shown in Table 2.2.

Table 2.2: Arrangement of input samples to oversampled PFB channelizer.

Third Group	Second Group	First Group	Polyphase Filters
S_{15}	S_{11}	S_7	$H_0(n)$
S_{14}	S_{10}	S_6	$H_1(n)$
S_{13}	S_9	S_5	$H_2(n)$
S_{12}	S_8	S_4	$H_3(n)$
S_{11}	S_7	S_3	$H_4(n)$
S_{10}	S_6	S_2	$H_5(n)$
S_9	S_5	S_1	$H_6(n)$
S_8	S_4	S_0	$H_7(n)$

In the table above, S_n represents input samples to the PFB, and $H_k(n)$ represents the polyphase filters for each channel. Each group of samples will be fed to the corresponding polyphase filters group by group. Note that the upper four samples in each group overlap with the lower four samples of its next group. Fig. 2.5 shows the structure of an oversampled PFB channelizer.

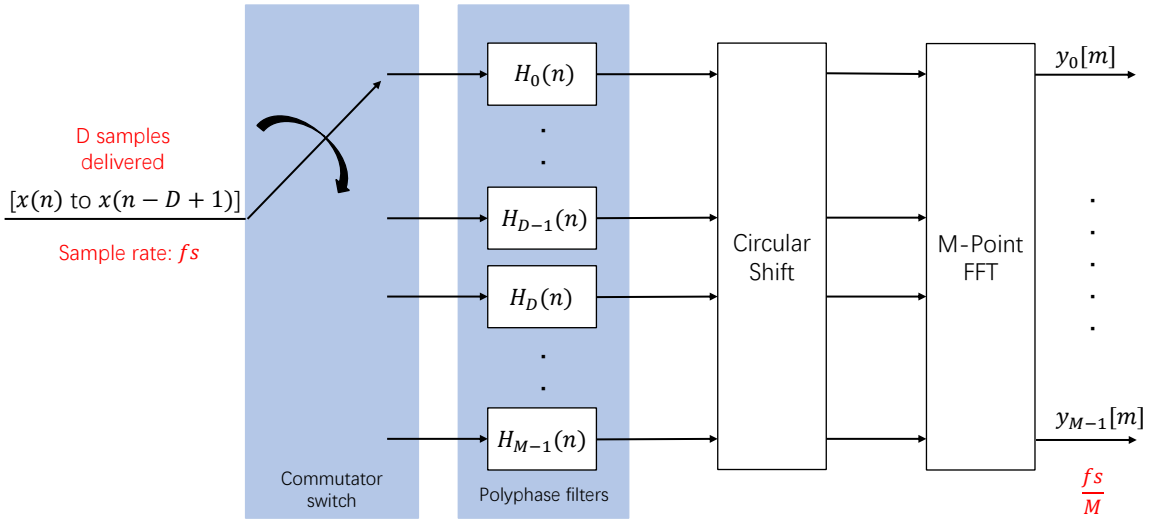

Figure 2.5: Oversampled channelizer structure.

A consequence of using an oversampled PFB channelizer is that a phase rotation is introduced at each of the output channel of the channelizer [15]. This phase rotation

can be corrected by implementing a circular shifting buffer between the FIR filters and the Fast Fourier Transform (FFT) unit as shown in Fig. 2.5.

For a special case when the oversampling ratio of the oversampled PFB channelizer is 2, this phase rotation can be corrected by a state machine with 2 states. To help explain, name these two states as state 1 and state 2 respectively. The state machine switches its state whenever the polyphase filters output a new set of data. In state 1, the outputs of the polyphase filters are delivered to the input of the FFT unit directly, while in state 2, the upper part and the lower part of polyphase filters' outputs are exchanged first, and then sent to the FFT unit.

2.2.5 Fast Fourier Transform (FFT)

The FFT is an efficient algorithm for implementing a DFT. It transforms a sequence of N complex numbers $x_n := x_0, x_1, ..., x_{N-1}$ into another sequence of complex numbers $X_K := X_0, X_1, ..., X_{N-1}$.

The calculation formula of DFT is:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, 0 \leq k \leq N-1 \tag{2.5}$$

where $W_N^{nk} = e^{-j\frac{2\pi k}{N}n}$.

By splitting the DFT formula into parity terms, the first $N/2$ points can be expressed as:

$$\begin{aligned} X[k] &= \sum_{r=0}^{\frac{N-1}{2}} X[2r] W_N^{2rk} + \sum_{r=0}^{\frac{N-1}{2}} X[2r+1] W_N^{(2r+1)k} \\ &= \sum_{r=0}^{\frac{N-1}{2}} X[2r] W_{\frac{N}{2}}^{rk} + W_N^k \sum_{r=0}^{\frac{N-1}{2}} X[2r+1] W_{\frac{N}{2}}^{rk} \\ &= A[k] + W_N^k B[k], k = 0, 1, ..., \frac{N}{2} - 1 \end{aligned} \tag{2.6}$$

Similarly, the last $N/2$ points can be expressed as:

$$X[k + \frac{N}{2}] = A[k] + W_N^k B[k], k = 0, 1, ..., \frac{N}{2} - 1 \tag{2.7}$$

It can be seen that the value of the last $N/2$ points can be determined entirely by calculating the intermediate process value when the first $N/2$ points are calculated. The parity decomposition is continued for $A[k]$ and $B[k]$ until it becomes a 2-point DFT, which avoids a large number of repeated calculations and realizes the FFT process.

Suppose the number of FFT operation points is N, and there are M levels of operations in total, then the relationship between N and M is:

$$M = \log_2 N \tag{2.8}$$

An 8-point FFT structure with 3 levels of operations is shown in Fig. 2.6.

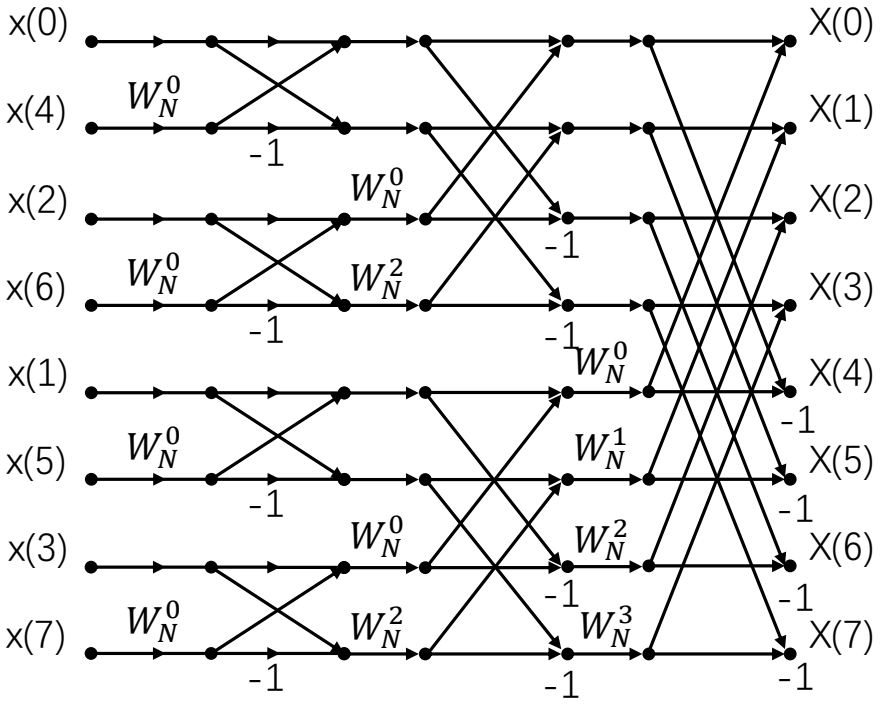

Figure 2.6: 8-point FFT structure.

From the figure above, it can be seen that the DFT can be completed by several multiplications and additions, instead of serial multiplication and accumulation of each sample. The basic operation unit of FFT is commonly known as butterfly unit. Fig. 2.7 shows a schematic diagram of the butterfly unit:

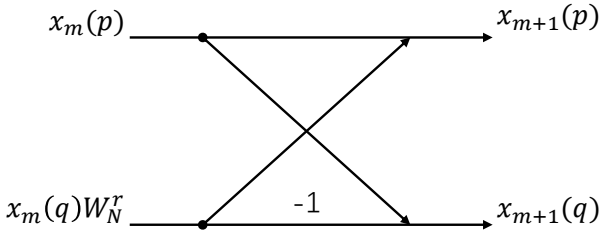

Figure 2.7: Schematic diagram of the butterfly unit.

The relationship between the input and output of a butterfly unit is:

$$x_{m+1}(p) = x_m(p) + x_m(q) \times W_N^r \tag{2.9}$$

$$x_{m+1}(q) = x_m(p) - x_m(q) \times W_N^r \tag{2.10}$$

There are one multiplication and two additions in a butterfly unit. In each level, there are $\frac{N}{2}$ butterfly units. Therefore, the total number of multiplications for one FFT operation is $\frac{N}{2}\log_2 N$ and additions is $N\log_2 N$.

The FFT we introduced previously is complex FFT because both the input and output of the FFT are complex value. However, in many cases the data transformed are not complex but real. In this case, if we still use a complex FFT to deal with sequence of real valued data, there will be a waste of approximately half of the resources.

There is an algorithm that can use an N-point complex FFT operation to obtain a $2N$-point real sequence FFT. Assume $x(n)$ is a real sequence of $2N$ points, and now $x(n)$ is artificially divided into an even sequence and an odd sequence:

$$x_1(n) = x(2n), n = 0, 1, ..., N-1$$

$$x_2(n) = x(2n+1), n = 0, 1, ..., N-1$$

Then combine $x_1(n)$ and $x_2(n)$ into a complex sequence:

$$y_(n) = x_1(n) + jx_2(n)$$

By using an N-point FFT we can get:

$$Y(k) = X_1(k) + jx_2(k)$$

According to the nature of DFT, we can get:

$$X_1(k) = \frac{1}{2}[Y(k) + Y*(N-k)]$$

$$X_2(k) = -\frac{j}{2}[Y(k) - Y*(N-k)]$$

In order to get $X(k)$, we need to find the relationship among $X(k)$, $X_1(k)$ and $X_2(k)$:

$$X[k] = \sum_{n=0}^{2N-1} x[n]W_{2N}^{nk} = \sum_{n=0}^{N-1} x(2n)W_{2N}^{2nk} + \sum_{n=0}^{N-1} x(2n+1)W_{2N}^{(2n+1)k}$$

$$\sum_{n=0}^{N-1} x(2n)W_N^{nk} + W_{2N}^k \sum_{n=0}^{N-1} x(2n+1)W_N^{nk}$$

and

$$X_1[k] = \sum_{n=0}^{N-1} x_1[n]W_N^{nk} = \sum_{n=0}^{N-1} x(2n)W_N^{nk} \tag{2.11}$$

$$X_2[k] = \sum_{n=0}^{N-1} x_2[n]W_N^{nk} = \sum_{n=0}^{N-1} x(2n+1)W_N^{nk} \tag{2.12}$$

Thus,

$$X(k) = X_1(k) + W_{2N}^k X_2(k) \tag{2.13}$$

The complex sequence composed of $x_1(n)$ and $x_2(n)$ is processed by FFT to get $Y(k)$. Then $X_1(k)$ and $X_2(k)$ can be calculated with the help of conjugate symmetry. Finally, using (2.13) yields $X(k)$. Using this method an N-point FFT can calculate a $2N$-point real sequence DFT.

2.3 De-dispersion

Cosmic radio signals are typically extremely weak and often weaker than the most sensitive amplifiers' noise. Because the energy of a dispersed pulse is spread in time and frequency, it is much weaker than the original undispersed version. Therefore, to recover the FRB signal, it is vital to de-disperse the received pulse and compensate for the time delay between different frequencies. A figure of three de-dispersed FRBs is shown in Fig. 2.8. It can be seen that the time delay between frequencies has been removed after de-dispersion. Currently, there are two methods to do this. The first is called coherent de-dispersion, and the other is incoherent de-dispersion. These methods are described separately in the following two sections.

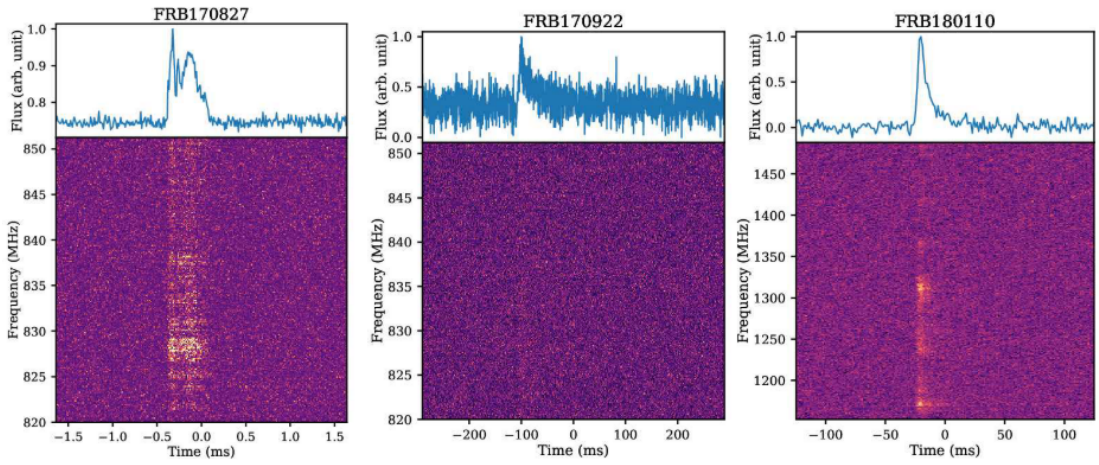

Figure 2.8: The pulse profile (top) and frequency-time spectrum (bottom) of the three FRBs: FRB170827, FRB170922, and FRB180110 [5].

2.3.1 Incoherent de-dispersion method

The incoherent de-dispersion method calculates the time delay across a frequency range with a known DM value, then subtracts this delay from each frequency channel. The premise of achieving incoherent de-dispersion is to know how long each channel needs to be delayed. From Eq. 3.3, the delay between channels can be calculated:

$$t_d = k_{DM} \cdot DM \cdot ({v_1}^{-2} - {v_2}^{-2}) \tag{2.14}$$

Here, the dispersion constant $k_{DM} = 4.15 \times 10^3$ MHz^{2}pc^{-1}cm^{3}s. v_1 and v_2 are the start frequency and stop frequency of the signal in MHz. DM is the dispersion measure in pc cm^{-3}.

There are three steps to achieve incoherent de-dispersion:

1. PFB channelization: Divide the broadband signal to several independent narrow channels.
2. Delay compensation: Calculate each channel's delay from Eq. 3.3, select a reference channel, and then move other channels to the time point where the reference channel's components appear.
3. Channel integration: Sum all channel's signal components together to generate the de-dispersed time series.

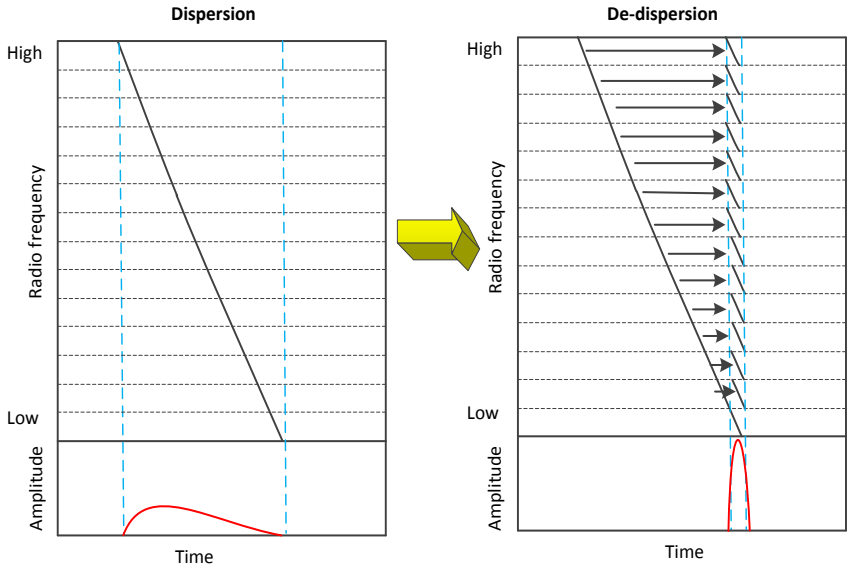

Figure 2.9: Incoherent de-dispersion demonstration.

Fig. 2.9 demonstrates the process of incoherent de-dispersion. The left figure is the dispersed signal, while the right figure is the de-dispersed signal. From the figure, it is clear that de-dispersion increases the dispersed signal's amplitude.

However, the ability of incoherent de-dispersion to compensate for the frequency-dependent delay is limited by the width of each channel. Incoherent de-dispersion moves the sub-channel to compensate for the delay, so delay inside sub-channels can not be compensated. When the width of each channel decreases, the delay inside each channel will decrease.

2.3.2 Coherent de-dispersion method

The coherent de-dispersion can compensate for the dispersion phase by convolving the signal with the inverse of the phase distortion resulting from the pulse passing

through the interstellar medium (ISM). The convolution can also be done by multiplication in the frequency domain. The computational cost of coherent de-dispersion depends on the observational bandwidth as well as the DM, so for FRBs, the coherent de-dispersion calculation can be computationally complex and slow [2][6]. The transfer function of ISM is defined in frequency domain [16]:

$$H(f + f_0) = \exp(\frac{2\pi i f^2 k_{DM} DM}{{f_0}^2 (f + f_0)}) \tag{2.15}$$

Here, f_0 is the center frequency of a particular channel, f is the offset to f_0 within that channel. Note that f_0 and f are both in MHz.

There are three steps to achieve coherent de-dispersion:

1. PFB channelization: Divide the broadband signal to several independent narrow channels in order to improve FFT efficiency.
2. Fourier transform : Transform time-domain data to frequency domain.
3. De-dispersion in frequency domain: Multiply the raw signal with the inversed ISM transfer function $H(-f)$.
4. Inverse Fourier transform: Transfer the frequency domain data back to time domain.
5. Channel integration: Sum all channel's signal in time to generate the de-dispersed time series.

2.4 Signal, noise and averaging

When the telescope observes a radio source, a randomly fluctuating noise signal with instantaneous amplitude $A(t)$ would be introduced at the receiver side. This noise is Gaussian and has a standard deviation σ and a mean $\mu = 0$. Thus, for N independent observing samples, the mean power of noise $A(t)$ is:

$$\bar{P} = \frac{\sum (A - \mu)^2}{N} = \frac{\sum A^2}{N} = \sigma^2 \text{ [17]} \tag{2.16}$$

The standard deviation σ is calculated by the following equation:

$$\sigma = \sqrt{\frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2}{N}} \tag{2.17}$$

Here $x_1, x_2 \cdots x_n$ are the power samples, $\bar{x}$ is the average power, N is the total number of samples.

The averaging filter technique is used to improve the SNR for a series of power samples. The averaging filter is a statistical technique to calculate averages of different subsets of the entire data set. Because the average power of a large subset (N»1) has a Guassian distribution (a consquence of the Central Limit Theorem), its variance (due to noise) decreases with the number of independent samples averaged. Thus the SNR, which is the ratio of average noise power, $\bar{P}$, to standard deviation, is improved by the square root of the number of power samples, N, averaged.

From the Nyquist sampling theorem, we know the relationship between the number of samples N, integration (averaging) time τ, and the equivalent noise bandwidth $\Delta\nu$ (Hz) is:

$$N = 2\Delta\nu\tau \tag{2.18}$$

Therefore, the improved new SNR is:

$$\mathrm{SNR}_{\mathrm{new}} = \mathrm{SNR}_{\mathrm{old}} N = \mathrm{SNR}_{\mathrm{old}}(2\Delta\nu\tau) \tag{2.19}$$

The above derivations show that averaging in time or bandwidth can improve the system SNR by a factor of N, and the factor can also be represented by $2\Delta\nu\tau$.

3

System design and simulation

This chapter will first give a brief overview of the system structure and then show how we simulate and verify the signal processing algorithm in MATLAB.

3.1 System overview

A block diagram of the system being developed is shown in Fig. 3.1.

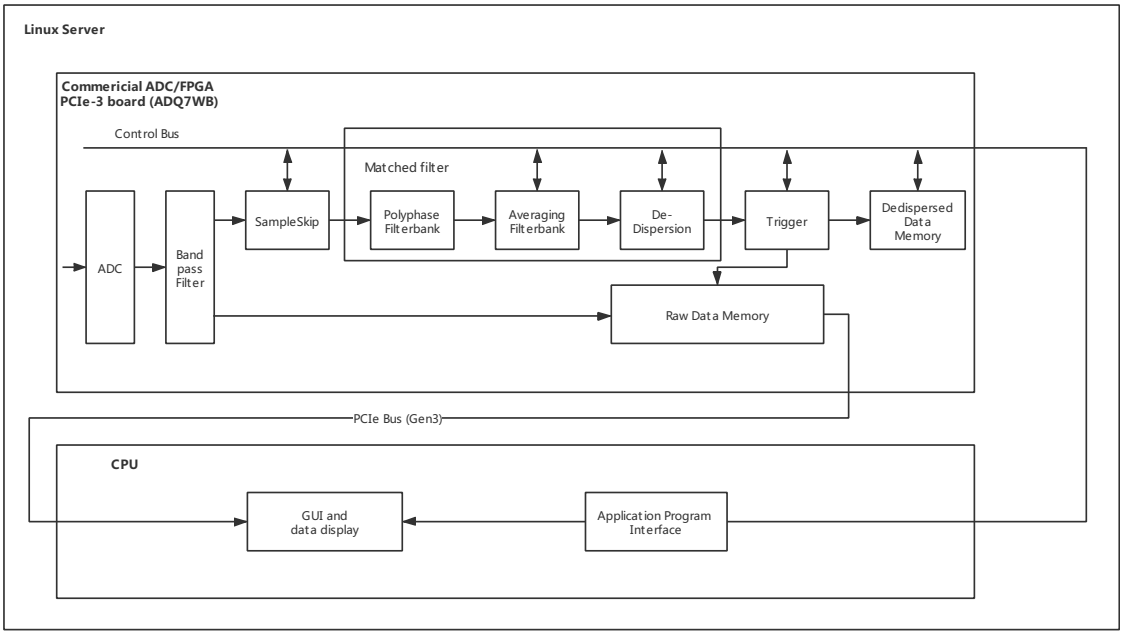

Figure 3.1: A block diagram of the FRB monitor system.

As shown in Fig. 3.1, the system is divided into two parts. The upper part is implemented in a ADQ7WB hardware platform [18], and the lower part is the software for data display and analysis. More detailed information about the hardware platform is given in Chapter 4.

The system starts with an analog-to-digital converter (ADC) block where the analog signal is sampled at 5 Giga samples per second (Gsps). In order to minimize the influence of RFI and increase the sensitivity of our system, there is a bandpass filter implemented at the very beginning of our DSP chain. The bandpass filter has a passband from 625MHz to 781.25MHz (BW = 156.25 MHz) since this frequency

range matches a folding frequency related to the clocks that we have (the sample clocks have to be a multiple, or a fraction of 5 Gsps). According to the bandpass sampling theorem (see section 2.1), for a signal with 156.25 MHz bandwidth, the sampling frequency only needs to be twice the signal bandwidth. Hence, there is no need to sample the signal at 5 Gsps. Therefore, a SampleSkip block is implemented after the bandpass filter to reduce the data rate from 5 Gsps to 312.5 Msps.

To detect target FRB signals, a de-dispersion algorithm is implemented to select the FRB with the DM we are looking for and suppress other transients. The basic idea of the de-dispersion algorithm is to compensate for the time delay for different frequency components which had been discussed in section 2.3. A PFB channelizer is needed to divide the broadband input signal into many narrow channels, providing various frequency bins for the incoherent de-dispersion algorithm.

Averaging is implemented between the PFB and the de-dispersion algorithm because of its ability to improve signal SNR and reduce the data rate.

The trigger block generates a trigger when the averaged matched filter output exceeds a programmable threshold and buffers the de-dispersed data in the de-dispersed data memory while the raw data is stored in the raw data memory. A data rate reduction block is placed before the raw data memory to store a longer period of raw data. Both the raw data and the de-dispersed data are then delivered to CPU for data display and analysis.

Table 3.1 shows some key parameters of the system. The reasons for choosing these parameters will be explained in subsequent sections.

Table 3.1: Key parameters of the system.

ADC	Sampling rate/channel	5 Gsps/Ch
Bandpass Filter	F_{stop1}	625 MHz
	F_{pass1}	639.125 MHz
	F_{center}	703.125 MHz
	F_{pass2}	767.125 MHz
	F_{stop2}	781.25 MHz
	attenuation at F_{stop1}	8.5 dB
	attenuation at F_{stop2}	8.5 dB
	Filter order	64
SampleSkip	Sampleskip factor	16
PFB	Number of channels	128
	Downsample factor	64
	Oversampling ratio	2
	Prototype filter order	511
	Prototype filter F_{pass}	0.61 MHz
	Prototype filter F_{stop}	1.83 MHz
	attenuation at F_{stop}	40 dB
Averaging Filterbank	Average length	256 points

3.2 System Design

3.2.1 PFB channelizer

An oversampled PFB channelizer is implemented to channelize the broadband signal into multiple narrower frequency channels. An M-channel oversampled PFB channelizer takes D input samples to compute M outputs. The structure of oversampled PFB is shown in Fig. 2.5.

In our system, the channelizer has 128 channels with a down-sampling factor of 64. Because the sampling frequency is 312.5 MHz, each channel would have a bandwidth of 2.44 MHz. And 50% of the channel width is overlapped with its adjacent channels because we set the oversampling ratio to 2. Our prototype filter is a low-pass finite impulse response (FIR) filter with a pass frequency of 0.61 MHz and a stop frequency of 1.83 MHz.

According to the Nyquist sampling theorem, the first half channels (channel 1 to channel $\frac{M}{2}$) are symmetrical to the second half channels (channel $\frac{M}{2}+1$ to channel M). So, we can only use the first half of channels' results to do further signal processing processes such as averaging and de-dispersion to save hardware resources.

3.2.2 Incoherent de-dispersion algorithm

When FRBs travel through the interstellar medium, they will be dispersed in time, and the higher frequencies will arrive earlier than lower frequencies. Similarly, the energy of FRBs would be dispersed over time, making FRB too weak to be detected. A solution to this problem is de-dispersion, which delays higher frequencies to lower frequencies, integrating signal energy, and giving FRB signals a unique signature, making them easier to be detected in an RFI environment despite their weakness. As discussed in chapter 2, compared with coherent de-dispersion, although its sensitivity is lower than coherent de-dispersion, incoherent de-dispersion is much simpler to realize. Considering our system aims to reduce the data rate and storage needed to detect an FRB, the lower sensitivity is not a problem as long as a transient pulse from a target FRB, which is unpredictable and rare, is not missed. So, to first get a working system and verify the system's feasibility, incoherent de-dispersion is used in this project.

As the final system will be running in real-time, signals that arrive first should be delayed. In a given observing band with a particular DM, the channel with the lowest frequency will be considered as the reference channel. The basic idea is to have variable delay times for each frequency channel. Our system calculates the delay between other channels and this reference channel and uses the system clock to express this delay. The delay precision is set by the averaging time. Less averaging means more delays and higher delay precision. The delays are implemented in variable-length shift registers, 64 shift registers would be generated for the first 64 channels, and each shift register's length equals the number of clock cycles that

specific channel should be delayed. During the system running, all shift registers will shift one unit at each clock cycle. The outputs of the shift registers are summed to implement the final integration stage of a matched filter.

Fig. 3.2 is a simple demonstration of de-dispersion operation where 'S' stands for FRB signal. At time t0, the FRB signal appears in channel 4, which is the channel with the highest frequency component, then at time t1, the FRB signal starts to appear at channel 3, finally, when the time goes to t3, FRB reaches channel 4. Because these four registers right shift one unit at each clock cycle, at time t3, all FRB signals would be aligned in time.

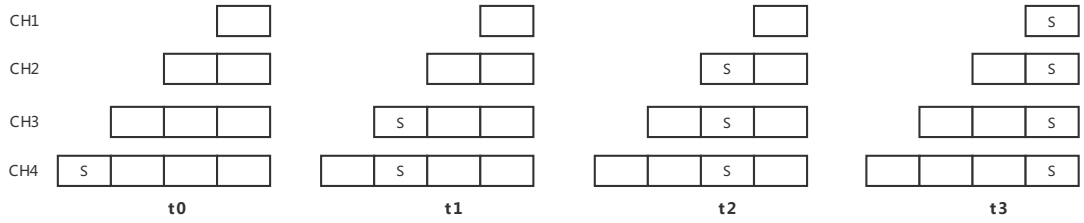

Figure 3.2: Shift registers for de-dispersion, 'S' represents FRB signal.

3.2.3 Triggering

After de-dispersion, the next step is to generate an integrated de-dispersed series to find the target signal. FRB signals in each channel are aligned in time to get the integrated de-dispersed series by summing them together. If the integrated de-dispersed series exceeds a user-defined threshold, then a potential FRB is found, so we generate a trigger to store a snapshot of the candidate.

When the control unit detects a trigger, the system takes a snapshot and transfers the data from the raw data memory to the host computer for astronomers to analyze FRB signals with high fidelity. The trigger time would be set as the center point of the snapshot, and to avoid missing any target signal, the snapshot would contain extra buffers at the start and end of the target signal. Users can set the length of the extra buffer.

3.3 Module simulation

The function of our system is to generate a trigger signal when a candidate FRB signal arrives and save the candidate signal in a "snapshot". By only saving candidates, the data rate of the system can be reduced. To verify the system, a simulated FRB signal is generated by creating a dispersed pulse with random Gaussian noise added. We will start with this simulated FRB signal and apply our algorithms to see if we can detect the pulse.

The signal processing chain starts with the sampling module, where the FRBs will be sampled and quantized to simulate the ADC. Then the PFB channelizer will divide the input into several frequency channels. Moving averaging is applied to the PFB results to improve SNR and reduce the data rate. Since the dispersion measure (DM) of the FRB is known a priori, a frequency-dependent delay is applied to each frequency channel to yield the de-dispersed FRB. When the SNR reaches the programmed threshold, a trigger is generated. In parallel with this, sample data is being stored in the raw data memory buffer so that when a trigger occurs, The sample data can be transferred to the CPU, where it is stored on disk and optionally displayed. The flow chart of the simulation system is shown in Fig. 3.3.

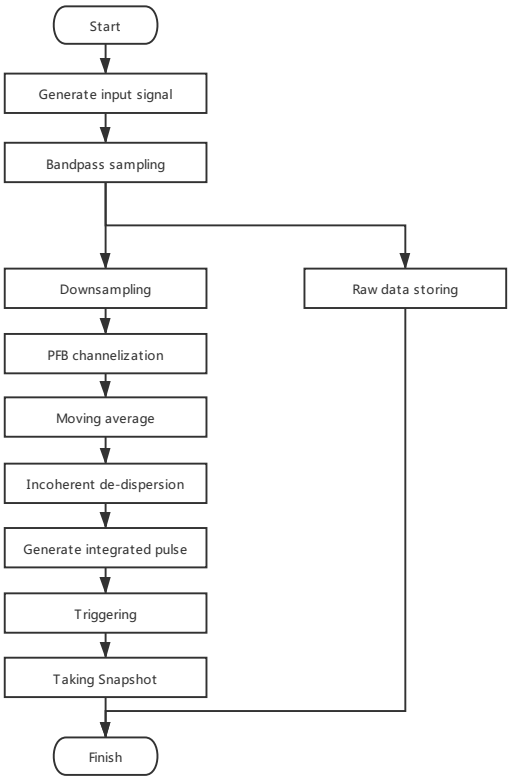

Figure 3.3: Simulation flow.

3.3.1 Simulated FRB and downsampling

A simulated FRB signal is used as the input signal. This signal is a quadratic chirp with DM equals to 10 pc cm^{-3}, whose spectrum and power spectrum are shown in Fig. 3.4. The frequency-time spectrum is shown in Fig. 3.5. Because the sampling rate of ADC is 5 GHz, the spectrum is 2.5 GHz wide in these two figures. The input simulated FRB sweeps from 781.25 MHz to 625 MHz, arrives at around 0.5 ms, and lasts for 0.5 ms. Due to the computer's performance limitation, the windowing length is not long enough compared to the 5 GHz sampling rate. So, among the whole spectrum, signal leakage would happen as an artifact of the plot. Artifact Gaussian noise is added to the original FRB to simulate the telescope's noise, which mainly comes from the low noise amplifier (LNA) and ground radiation picked up in

the sidelobes of the antenna. For the 25m telescope, the noise temperature is about 30Kelvin.

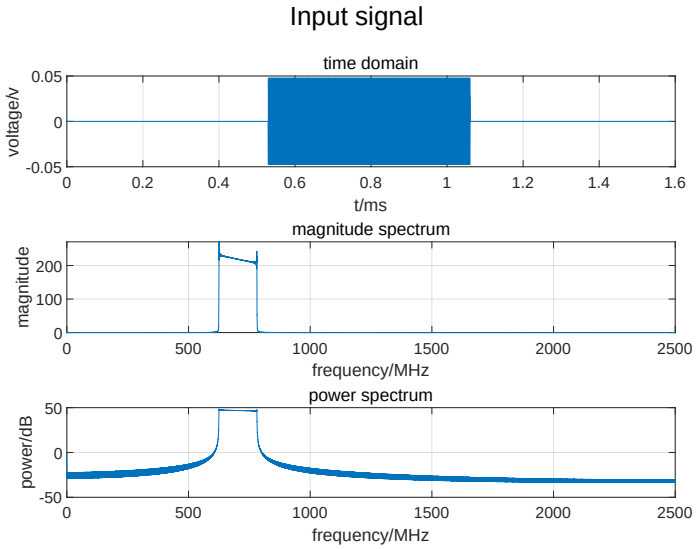

Figure 3.4: Input simulated FRB: (1) Top: Time series of the simulated FRB. (2) Middle: Magnitude spectrum of the simulated FRB. (3) Bottom: Power spectrum the simulated FRB.

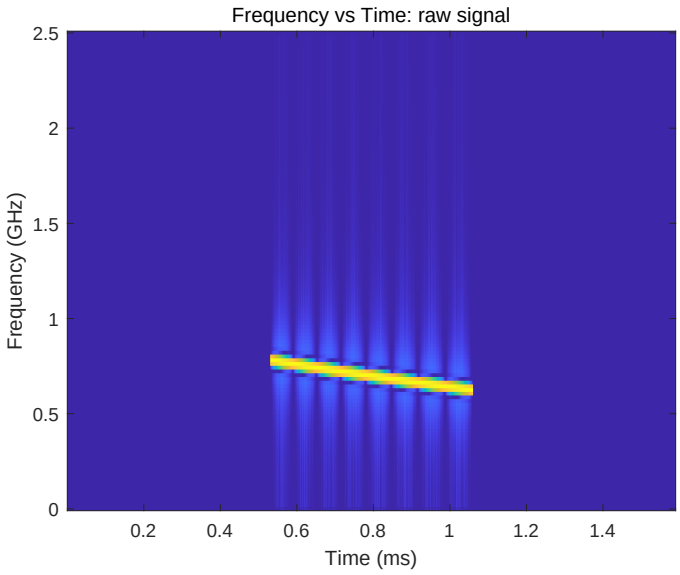

Figure 3.5: Simulated FRB: 781.25 MHz to 625 MHz.

After receiving the input signal, a bandpass filter would be added to attenuate the frequencies outside the 625 - 781.25 MHz. Then the signal is downsampled by a factor of 16, which results in a 312.5 MHz sampling rate. One reason for downsampling here is that this can reduce the data rate and save disk space. Another

reason is that the interested observing band is 625 - 781.25 MHz, so according to the bandpass sampling theorem, 312.5 MHz is minimal sampling frequency to get sampled signal without aliasing. Fig. 3.6 shows the signal after the downsampling process. Because the spectrum is 0 - 156.25 MHz now, the simulated FRB signal with 156.25 MHz wide crosses the entire band exactly.

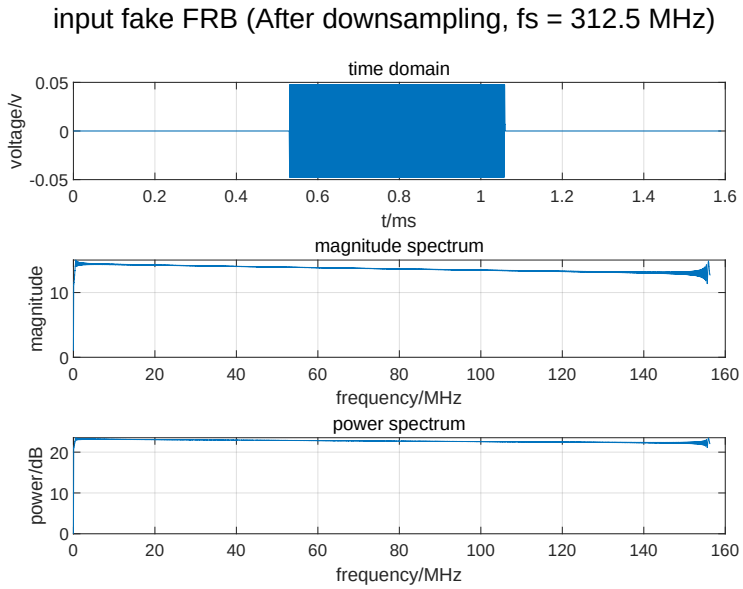

Figure 3.6: Down-sampled signal: (1) Top: Time series of the down-sampled signal. (2) Middle: Magnitude spectrum of the down-sampled signal. (3) Bottom: spectrum the down-sampled signal.

3.3.2 Simulation of PFB channelizer

3.3.2.1 Simulation behaviour

Since we would implement incoherent de-dispersion in our system, the resolution of de-dispersion depends on each channel's bandwidth. The more narrow the channel, the higher the resolution. In our design, the PFB channelizer has 128 channels, covering the whole 312.5 MHz sampling frequency band, and each channel would have a 2.44 MHz wide bandwidth. To better understand the detection limits of our system, a simulated FRB with -3 dB SNR was generated. The simulated FRB signal is from 625 MHz to 781.25 MHz, which locates between 0 MHz to 156.25 MHz after downsampling, crossing the whole 128 channels. The channelization results are shown in Fig. 3.7, where different colors represent signal components in different channels. Components from channel 1-128 are placed along the time axis, which matches our expectations.

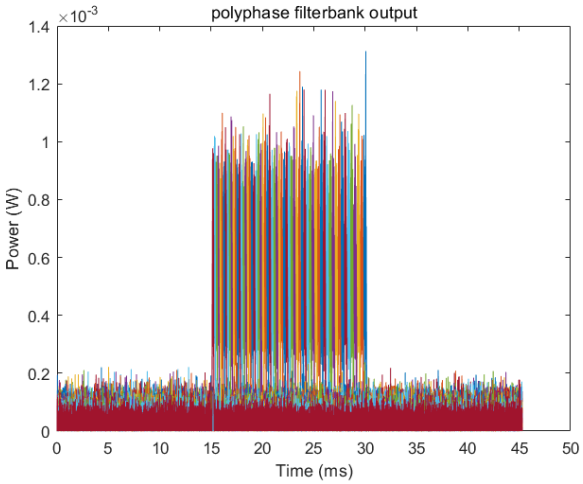

Figure 3.7: PFB channelizer output for a -3 dB input.

Furthermore, if we zoom in at several specific channels, we should see individual signal components. In 9 to 14 in Fig. 3.8, we can see that there are corresponding frequency components in each channel, and higher frequency components arrive earlier than lower frequency components, which exactly shows the FRB's dispersion property. Using these frequency components, we can de-disperse the FRB.

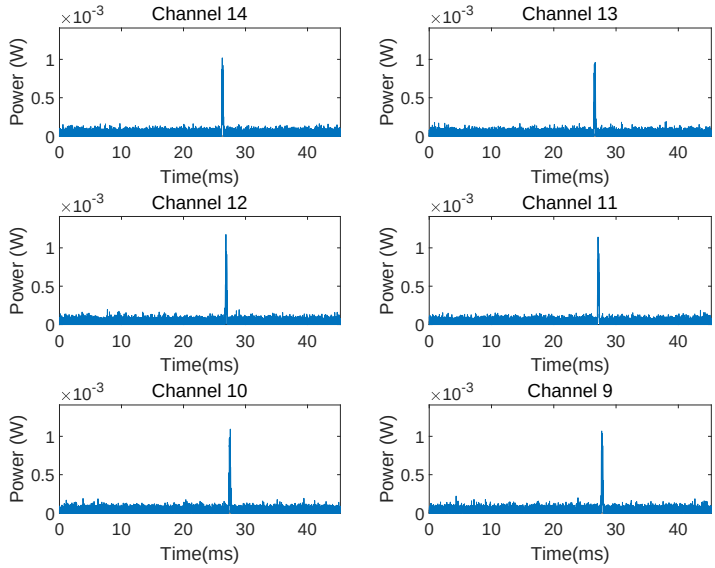

Figure 3.8: Channelization results in channel 9 to 14.

3.3.2.2 Signal-to-Noise Performance

A Gaussian noise vector is used as input to verify the PFB channelizer's function and performance, as Gaussian noise is distributed over the whole spectrum. The PFB channelization process divides the one channel input signal into 128 channel outputs, so the noise power should be distributed in 128 channels evenly. Assume

the total power of the Gaussian noise vector is 1 W, then the ideal power in each channel is:

$$\text{ideal power in each channel} = \frac{\text{Total Power}}{\text{Num of channel}} = \frac{1}{128} = 7.8\ mW \tag{3.1}$$

In MATLAB simulation, a Gaussian noise vector is generated with standard deviation equals to 1, and the resistance is also set to 1 Ω, so the total power of this input is $\frac{V^2}{R} = 1$ W. After PFB channelizer, the power in each channel is 6.8 mW, compared to the ideal value 7.8 mW, there is a 13% power attenuation. The results are shown in Table 3.2. One reason for this power attenuation is that in our PFB design, in each channel, 50% of the channel bandwidth is passband, 50% of the channel is transition band, which is overlapped with adjacent channels. The 50% long overlapped transition band can cause power attenuation. Another reason is that we use a 512 taps equal ripple filter as the prototype filter, and the prototype filter's order may not be high enough. Moreover, the frequency response of the equal ripple filter shown in Fig. 3.9 has a narrow passband, which can also cause attenuation.

Table 3.2: PFB channelizer results of Gaussian noise.

	Predicted result	Simulation result
Noise power in each channel	7.8 mW	6.8 mW
Total noise power	1 W	0.87 W

Table 3.3: PFB channelizer results of a single tone signal.

	Predicted result	Simulation result
Signal power in channel 3 and 127	0.5 W	0.48 W
Total signal power	0.5 W	0.48 W

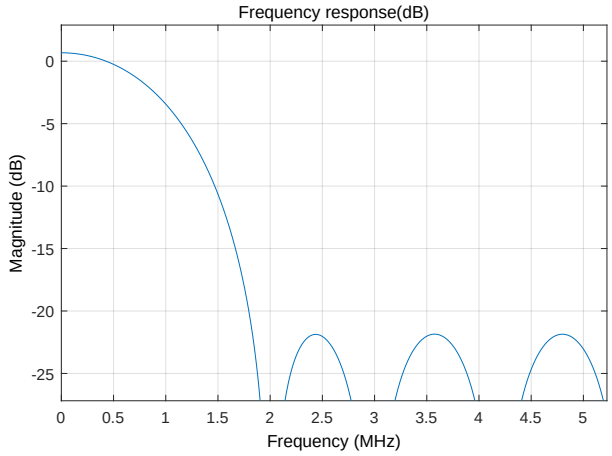

Figure 3.9: Frequency response of equal ripple prototype filter.

Apart from the Gaussian noise that crosses all channels, a single tone sine wave signal is also generated at 4.88 MHz. The power of this sine wave is 0.5 W. The sine wave should appear at the center of channel 3 and channel 127, and since these two channels are mirror channels, they should each have half of the total power of the sine wave. The simulation result shows that channels 3 and 127 each have 0.24 W power, which matches our predicted result. The results are shown in Table 3.3.

3.3.3 Simulation of averaging

3.3.3.1 Simulation behaviour

In later process steps, the de-dispersed signal will be accumulated to generate the integrated time series. This means that not only the signal would be integrated but also the noise. So as discussed in chapter 2, we average each channel in time to improve the SNR and reduce the data rate in our signal chain.

The results of the averaging filter are shown in Fig. 3.10 and Fig. 3.11. Compared to Fig. 3.7 and Fig. 3.8, the signal pattern remains the same, and the signal-to-noise ratio improves significantly. The power increases along the time axis in Fig. 3.10 because the averaging process can be seen as a low-pass filter, and the simulated FRB's frequency decreases along the time axis. Frequencies that come later have lower frequency result in higher amplitude.

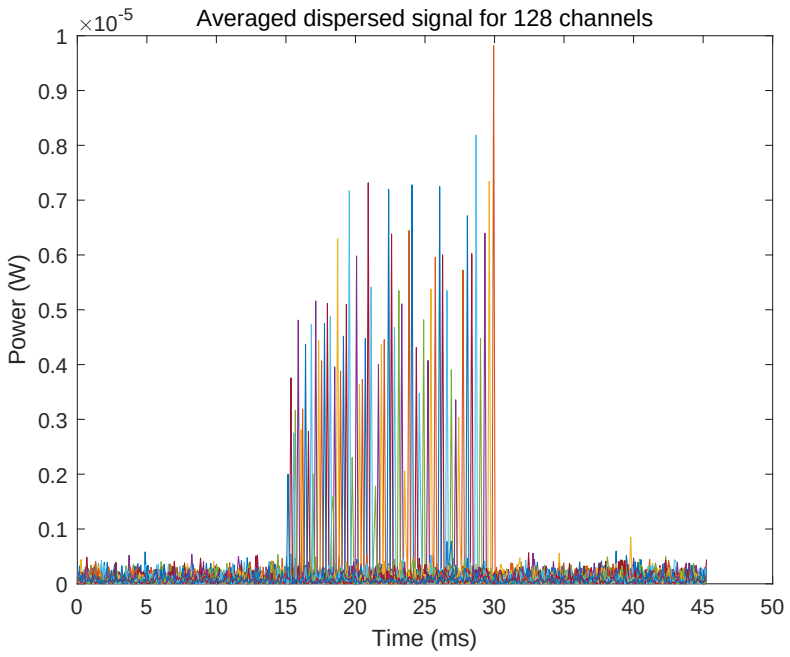

Figure 3.10: Averaged channelization result.

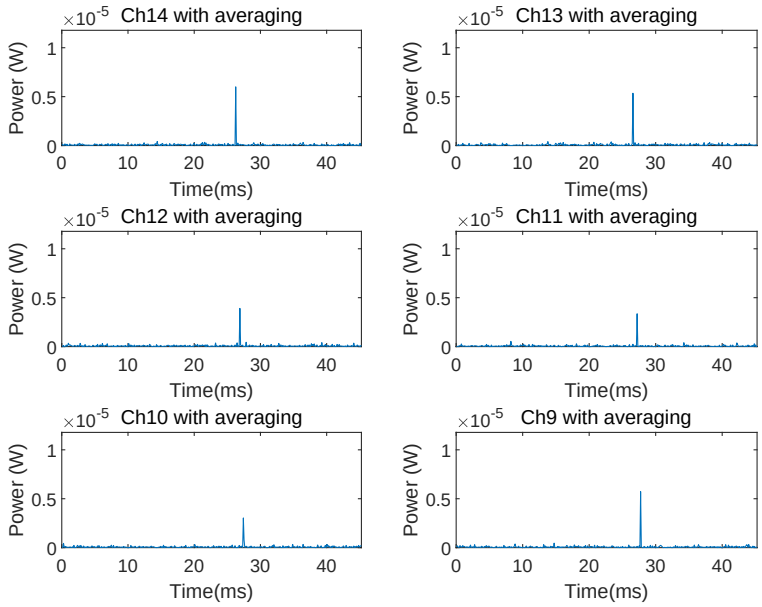

Figure 3.11: Averaged channelization results in channel 9 to 14.

If we zoom in on a specific channel: channel 14, the PFB result and averaging PFB result are shown in Fig. 3.13 and Fig. 3.12. We noticed that the baseline noise level is decreased by a factor of 256, and SNR improves significantly.

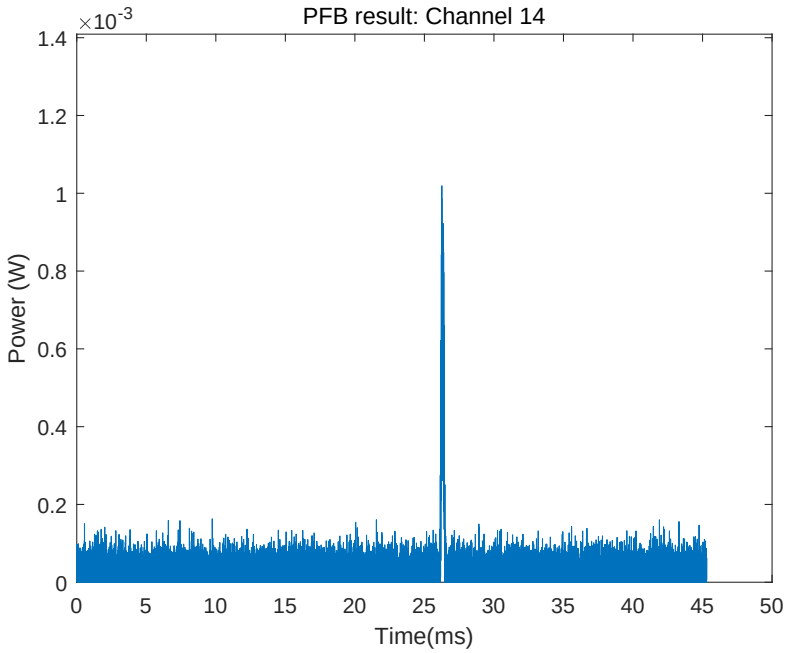

Figure 3.12: PFB channelization results in channel 14.

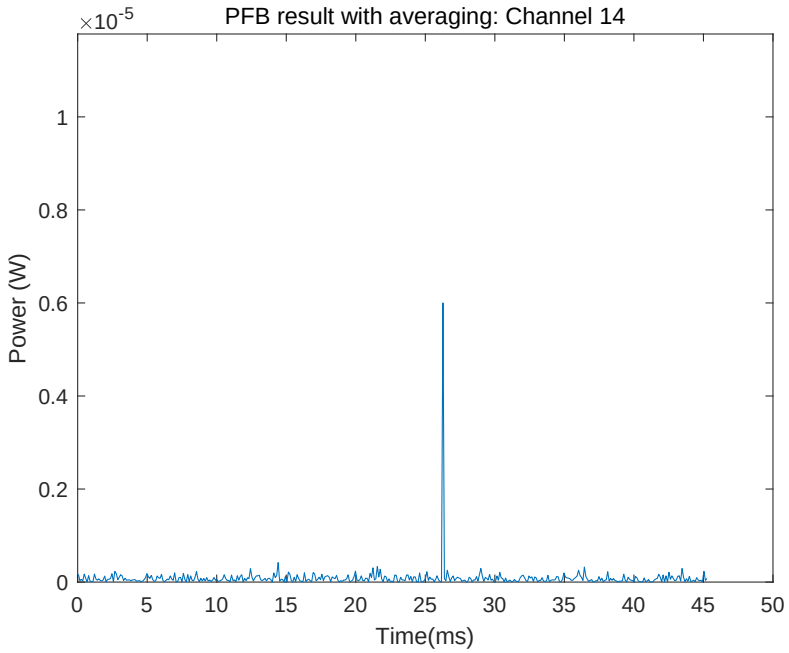

Figure 3.13: Averaged channelization results in channel 14.

3.3.3.2 Predicted results

For Gaussian noise, $power = (std)^2$, where std stands for the standard deviation. By applying N-point averaging, std would decrease by a factor of $\sqrt{N}$, so the noise power would decrease by a factor of N. And in our PFB design, since 50% of the channel's bandwidth is overlapped with its adjacent channel, that half of the signal is repetitive in each channel, so that the power would decrease by a factor of $\frac{N}{2}$, then the SNR would increase by a factor of $\frac{N}{2}$ consequently.

A Gaussian noise vector and a single tone sine wave are generated to verify that the simulation matches our predicted results. The power of the noise vector is 1 W, and a 256-point averaging filter is added in the simulation. So the total noise power after averaging should be:

$$\text{ideal total power} = \frac{\text{Total Power}}{\frac{N}{2}} = \frac{1}{256} = 0.0039\ W \tag{3.2}$$

If we also compare the averaging results for different settings, we can see an SNR improvement of factor $\frac{N}{2}$ in Table 3.5. The comparison between 512-point simulation and predicted results is shown in Table 3.4, the SNR improvement factor has some attenuation comparing to the ideal case, but still, the results are in close agreement with our predicted results.

Table 3.4: Averaging results for 512-point averaging.

	Predicted result	Simulation result
Noise power in each channel	3.0469*1e-5 W	3.0088*1e-5 W
Total power after averaging	0.0039 W	0.0039 W
SNR improvement factor	256	248

Table 3.5: Averaging results for 512-point averaging.

	128-point averaing	256-point averaging	512-point averaging	1024-point averaging
Total power after averaging	0.0156 W	0.0078 W	0.0040 W	0.0020 W
SNR improvement factor	64	128	248	495

3.3.4 Simulation of incoherent de-dispersion

The de-dispersion result is shown in Fig. 3.14, and compared to Fig. 3.10, all the signal components are aligned at one same time point.

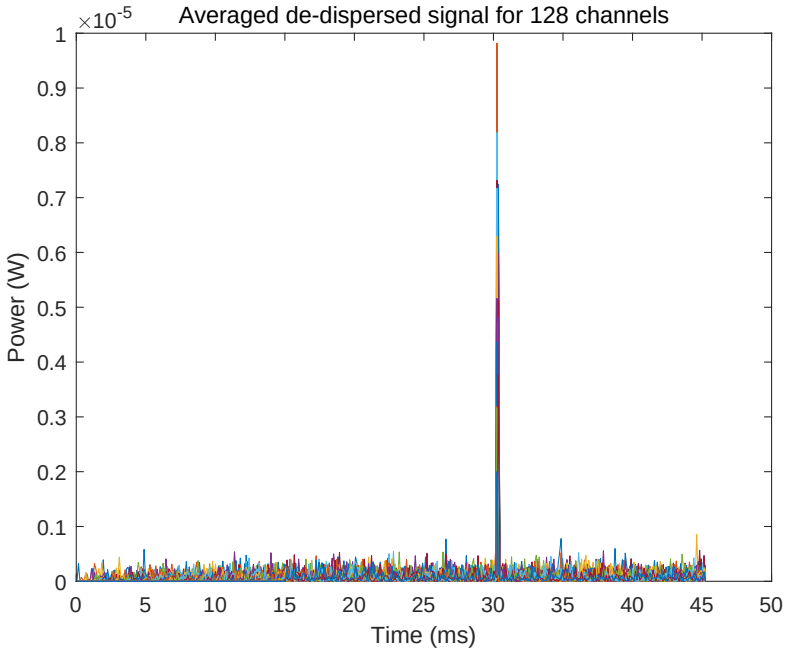

Figure 3.14: De-dispersion result.

3.3.5 Simulation of triggering

Once the incoherent de-dispersion is implemented, we can get de-dispersed time series by integrating all channels. The de-dispersed time series is shown in Fig. 3.15. A pulse appears at 30 ms, which is the endpoint of the target FRB. The trigger

would also be generated at 30 ms. Fig. 3.16 shows the comparison between dispersed signal and de-dispersed signal. We can see that the de-dispersed time series has higher SNR compared to dispersed time series.

The SNR can be calculated by

$$SNR = \frac{\text{signal power}}{std\_noise} = \frac{\text{pulse} - std\_noise}{std\_noise} \tag{3.3}$$

where the *std_noise* is the standard deviation of the noise baseline. When we only put pure noise as input to the system, then the result we get is the noise baseline. *pulse* is the power of the de-dispersed time series vector. In our simulation case, the SNR of the de-dispersed series is 8.11 dB, and for the dispersed series, it is almost impossible to distinguish any valid signal from it. Furthermore, in the bottom two plots in Fig. 3.16, the dispersed series's frequency components are spread out along the time axis, while the de-dispersed signal's frequency components are all gathered at one same time point.

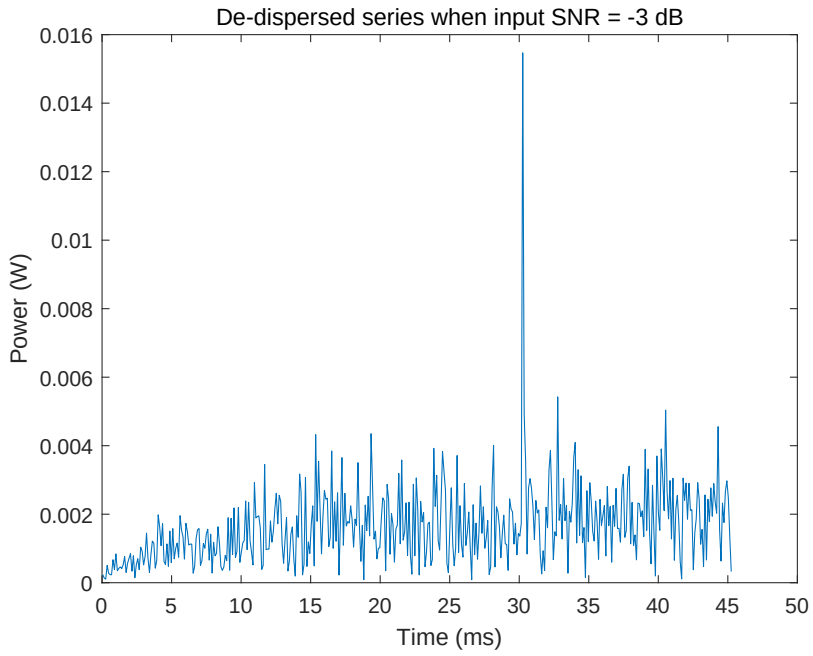

Figure 3.15: De-dispersed series.

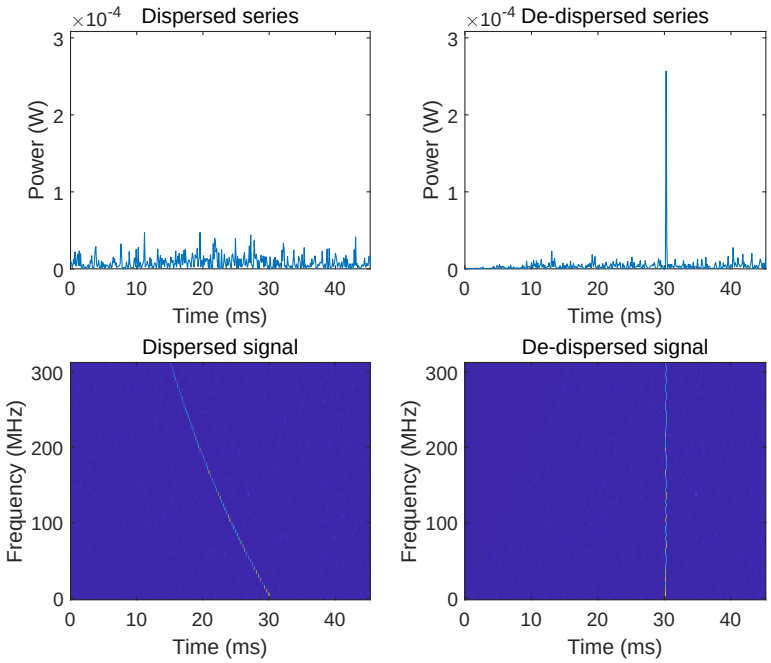

Figure 3.16: Comparison between dispersed and de-dispersed time series.

Furthermore, Fig. 3.17 shows the comparison between dispersed time series and de-dispersed time series. So, after de-dispersion, a clear pulse can be detected by setting a proper threshold. Secondly, compared to dispersed time series, the de-dispersed pulse's power has an obvious increase, making the FRB out of the noise baseline and increasing the SNR. Thirdly, it can be seen that the noise floor is reduced in the de-dispersed plot. The reason is that in dispersed time series, the noise baseline is the summation of noise power and dispersed FRB power. However, in the de-dispersed time series, the noise baseline only equals pure noise power since all FRB components are delayed to the same time point.

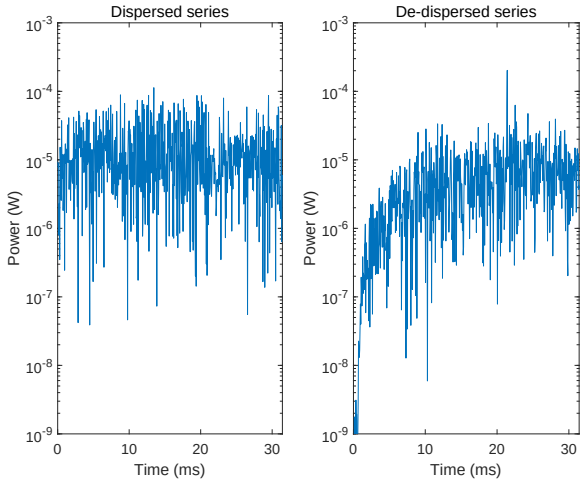

Figure 3.17: Dispersed signal and de-dispersed signal for -3 dB input.

If we set a trigger threshold that is just below the maximum magnitude of the de-dispersed time series, that is, the de-dispersed pulse at 30 ms in Fig. 3.17, then only target FRB will exceed this threshold. Ideally, the power of the noise will not be larger than this threshold, so every trigger is valid. However, when the input's SNR is extremely low, the final de-dispersed pulse's power can still be less than that of noise even with averaging and de-dispersion. Consequently, false triggering will happen.

3.4 Hardware system simulation

To simulate the hardware system's performance, we simulate our hardware system implementation in MATLAB. The hardware system implementation uses a linear chirp signal as the input signal, and the input signal is from 725 MHz to 625 MHz. To keep consistency with the hardware tests that will be presented in section 5.2, the root mean square (RMS) value is set to 100 mV, and Fig. 3.18 shows the input signal at -3 dB SNR. The input signal comes at 0 ms and lasts to 5 ms. Fig. 3.19 shows the linear chirp pattern of the input.

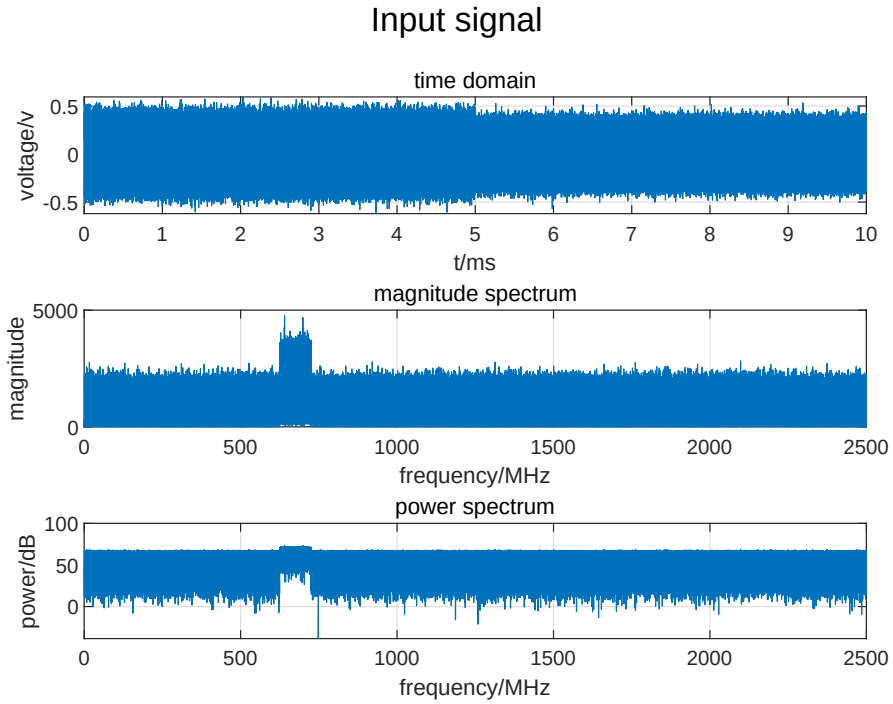

Figure 3.18: Input linear chirp with noise: (1) Top: Time series of the input linear chirp. (2) Middle: Magnitude spectrum of the input linear chirp. (3) Bottom: Power spectrum the input linear chirp.

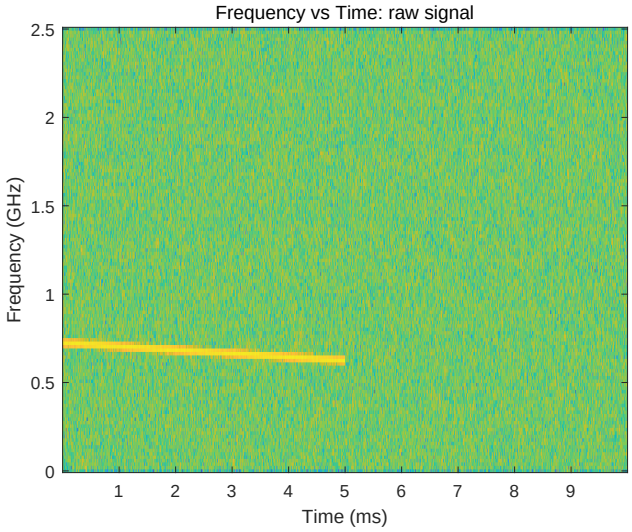

Figure 3.19: Input linear chirp: 725 MHz to 625 MHz.

A bandpass filter is implemented to select the signal band of interest, which is 625 MHz to 781.25 MHz. When passed through the bandpass filter, the noise's amplitude is decreased to 35 mV due to the filtering process. The filtering result is shown in Fig. 3.20, we see that the signal components outside the interested frequency band are attenuated. Furthermore, bandpass sampling is implemented to downsample the input. In Fig. 3.21, the signal sampling rate is decreased to 312.5 MHz. Comparing to Fig. 3.20, the signal pattern in the time domain remains the same, but in the frequency domain (middle and bottom plots of both figures) since the signal is downsampled, the spectrum width is decreased to 312.5 MHz from 5 GHz, fewer data would be recorded. Thus, we can save hardware resources.

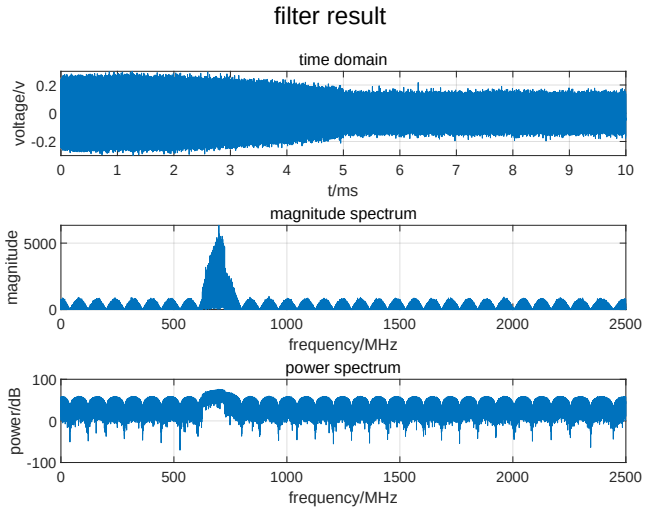

Figure 3.20: Input signal after bandpass filtering: (1) Top: Time series of the filtered input. (2) Middle: Magnitude spectrum of the filtered input. (3) Bottom: Power spectrum of the filtered input.

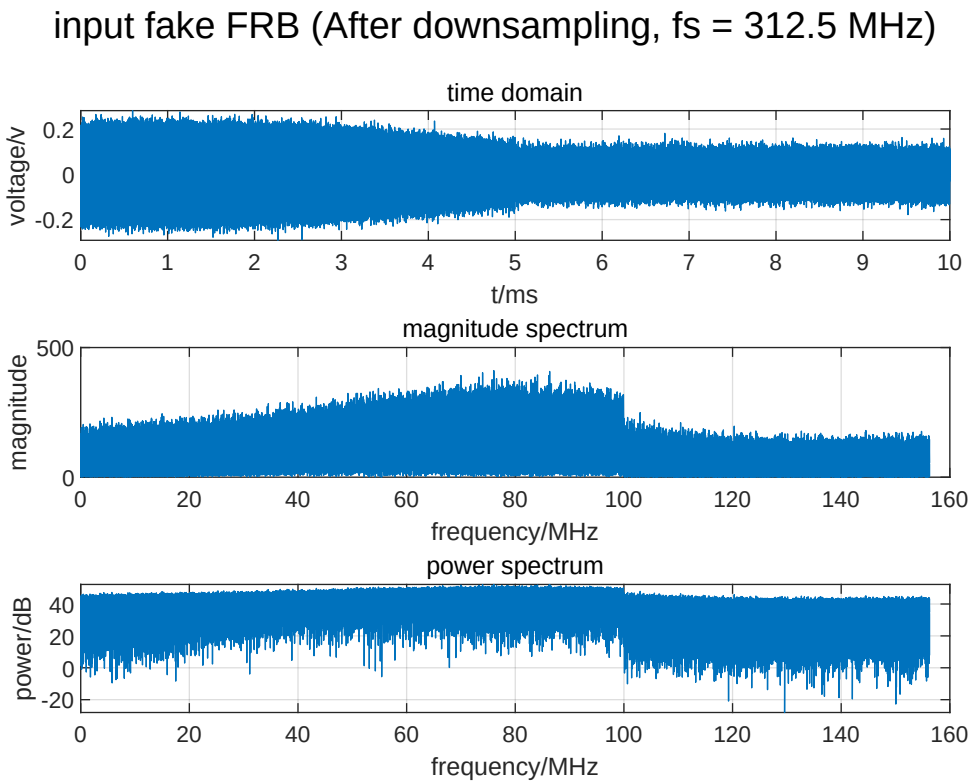

Figure 3.21: Input signal after downsampling: (1) Top: Time series of the downsampled input. (2) Middle: Magnitude spectrum of the downsampled input. (3) Bottom: Power spectrum of the downsampled input.

The PFB channelization result after averaging is shown in Fig. 3.22, it can be seen that signal components are dispersed along the time axis. Since the signal passes through the bandpass filtering process, due to the bandpass filter response, frequency components near the pass band edge would be attenuated, so that low frequency components' magnitude is lower than high frequency components. After de-dispersion, all frequency components are moved to a same time point (shown in Fig. 3.23). The integrated time series of both dispersed time series and de-dispersed time series are shown in Fig. 3.24. It is obvious that the de-dispersed time series have a strong pulse, indicating the occurrence of FRB. Fig. 3.25 also shows the same result, frequency components from 100 MHz to 0 MHz are gathered together at 5 ms, which represents the original 100 MHz wide input in 625 MHz to 725 MHz.

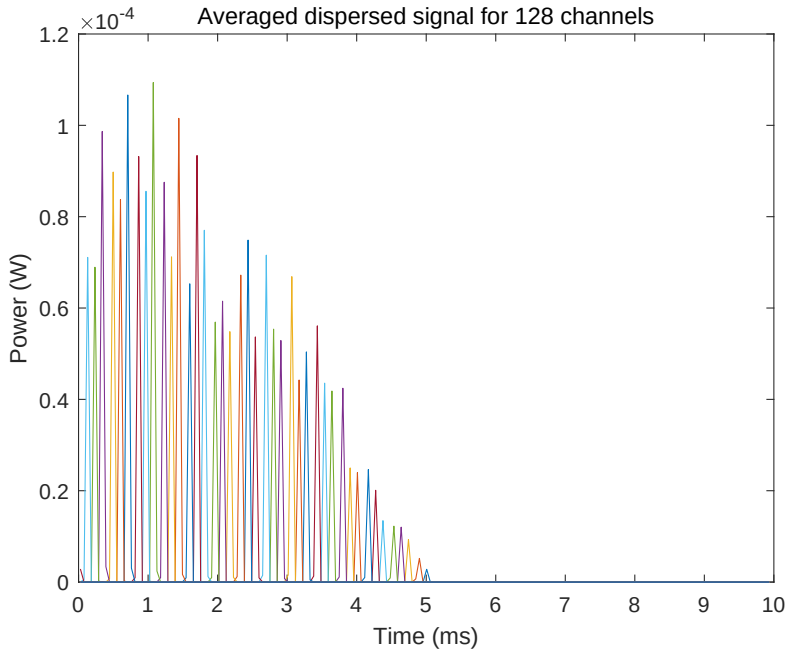

Figure 3.22: Averaged PFB channelization result: input signal is dispersed along time axis.

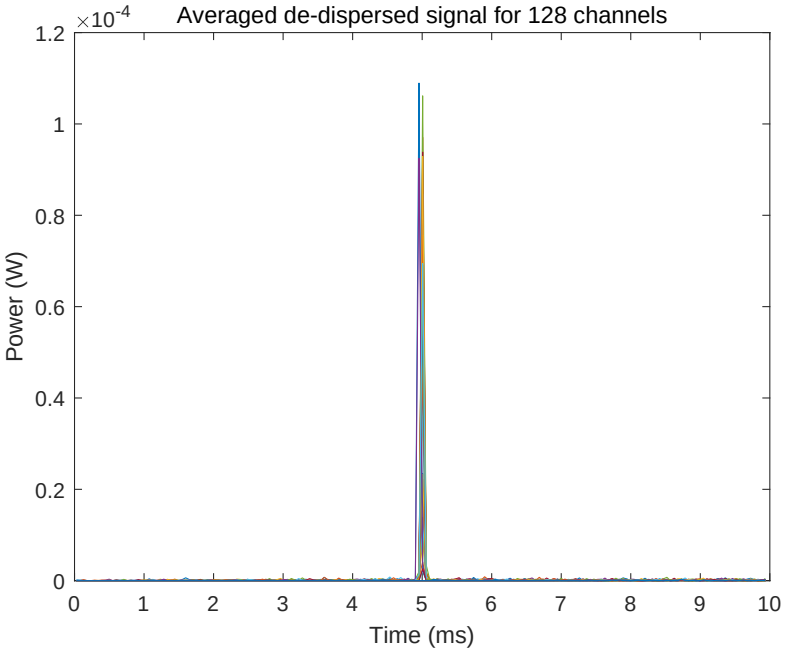

Figure 3.23: De-dispersed PFB channelization result.

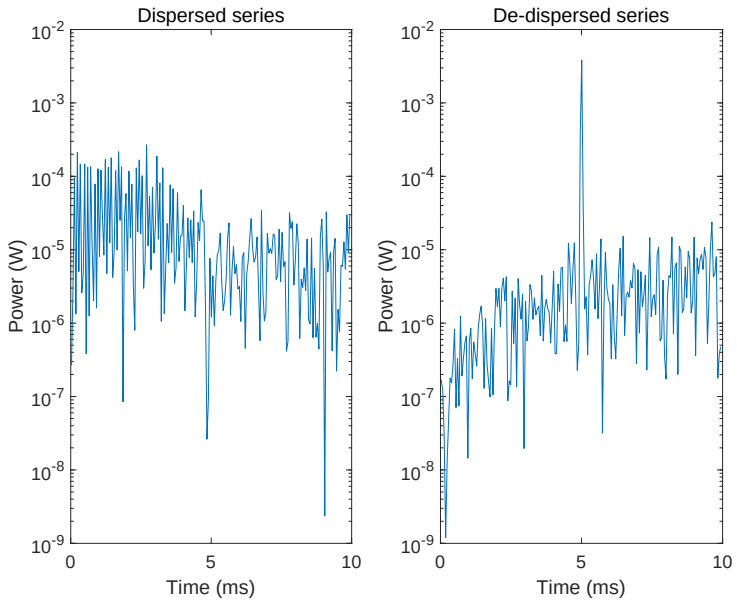

Figure 3.24: Comparison between dispersed and de-dispersed time series.

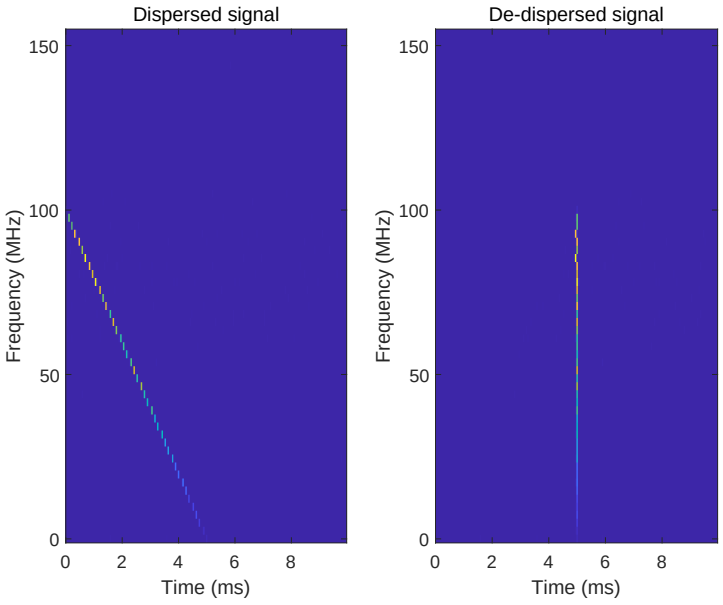

Figure 3.25: Dispersed linear chirp and de-dispersed time series.

To explore the performance of the hardware system, we use input signal within the SNR range between -24 dB to 3 dB, and the results are shown in Table 3.6. Moreover, comparisons between dispersed time series and de-dispersed time series are shown in Fig. 3.26 and Fig. 3.27, which represent 3 dB input and -20 dB input cases.

Table 3.6: De-dispersion performance for different inputs.

Input SNR (dB)	3	0	-4	-8	-12	-16	-20	-24
De-dispersed SNR (dB)	33.6	30.0	25.5	23.0	17.7	13.2	9.2	5.0
False trigger	No	No	No	No	No	No	No	Yes
SNR improvement (dB)	30.6	30.0	29.5	31.0	29.7	29.2	29.2	29.0
False trigger rate	-	-	-	-	-	-	-	42.3%

From Table 3.6, we find that when the input SNR decreases, the maximum pulse of the de-dispersed time series would have a lower magnitude. When the magnitude is lower than the noise, false triggering will happen. In our tests, if the input signal's SNR is stronger than -24 dB or less, the original FRB signal can be detected without false triggers. However, when the input has -24 dB SNR, a false trigger that is stronger than the original FRB would appear. And in general, we can see a constant SNR improvement of rough 30 dB for all cases, which shows that our signal processing system is able to identify an FRB even in a very noisy environment. Fig. 3.28 shows an example of false triggering, if we set a trigger threshold just below the de-dispersed pulse of FRB (the pulse at 5 ms), there are still two noise spikes higher than the threshold. So, there are two false triggers in this case, and the false trigger rate is 66.67%.

As a result of 50 independent simulations, the -24 dB case has an average false trigger rate of 42.25%, and the rest 57.75% of the triggers are valid, So even when the de-dispersed SNR is only 5 dB, we can still reduce the data rate by 57.75%, which is a significant improvement.

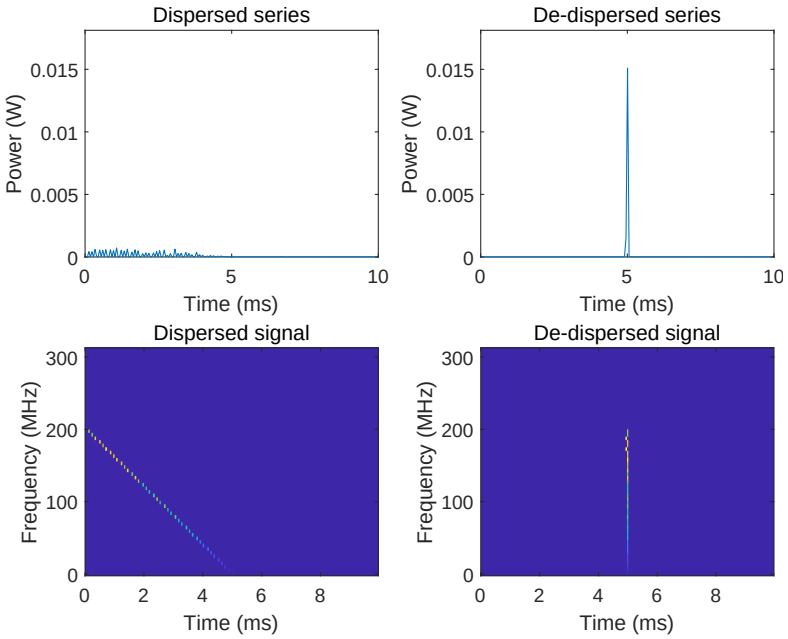

Figure 3.26: Dispersed signal and de-dispersed signal for 3 dB input.

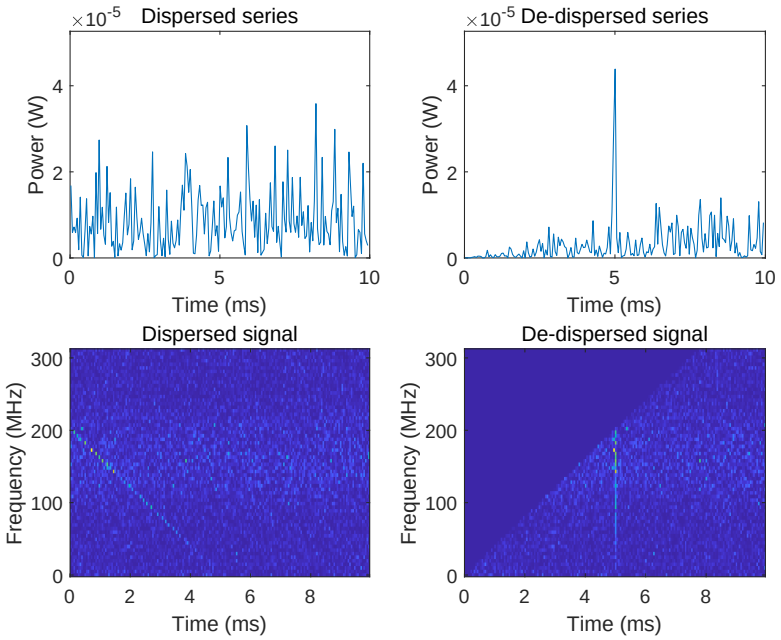

Figure 3.27: Dispersed signal and de-dispersed signal for -20 dB input.

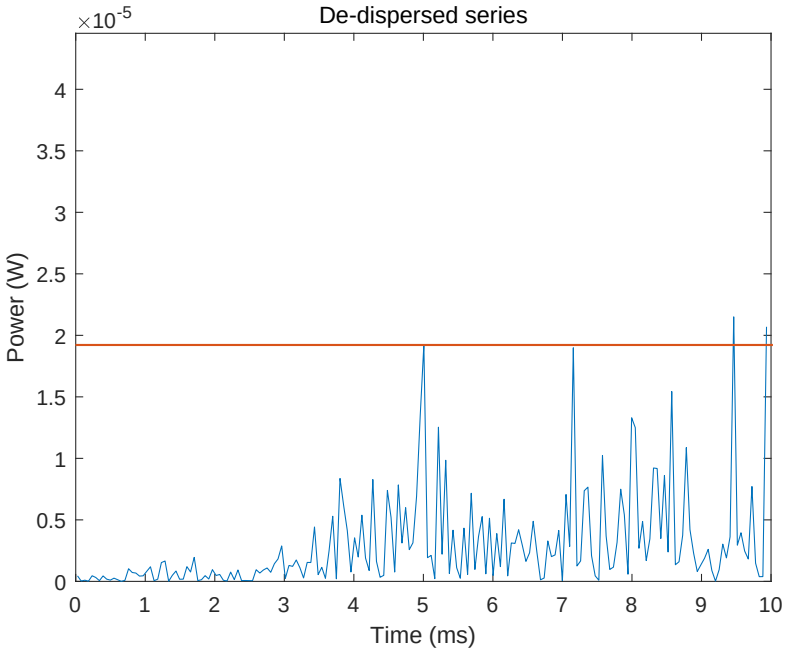

Figure 3.28: False triggering.

4

Hardware implementation

In this chapter, we will first briefly introduce the hardware platform we are using for the design and then describe the implementation of the functions and algorithms as needed using Xilinx Vivado.

4.1 The advantage of using Field-Programmable Gate Array (FPGA)

FRB signal processing is difficult to accomplish on a serial processor because of the high frequency, large DM, and broad bandwidth properties of the signal itself, resulting a large computational load. An ideal solution is to make use of an FPGA which is widely used in digital signal processing.

FPGAs are a type of high-performance programmable logic device based on CPLDs (Complex Programmable Logic Device). FPGAs have a high level of integration, allowing them to perform highly complex timing and combinational logic circuit functions. They are ideal for designing high-speed, high-density digital logic circuits.

The main advantage of using FPGAs to implement FRB signal processing applications is their high-speed parallel processing capability. This is mainly achieved through two technologies: concurrency and pipeline. Concurrency refers to the repeated allocation of computing resources so that multiple modules can perform independent calculations simultaneously. A simple example is that FPGA can perform multiple additions and multiplications at the same time. Pipeline refers to divide one task into segments and execute them simultaneously.

This parallel mechanism makes FPGA an excellent choice for performing repetitive digital signal processing tasks such as digital FIR filters. For high-speed parallel digital signal processing tasks, FPGA performance far outperforms the serial execution architecture of general-purpose digital signal processing (DSP) processors.

As outlined in the previous sections, 5×10^9 multiply accumulate operations are needed per second to keep up with the data rate. To do this using a CPU would require a clock rate of 5 GHz and a memory bandwidth of 40 Gbps which is far beyond the capacity of a single processor. Only an FPGA or custom hardware can be used to process the 40 Gbps data stream. To deal with the data rate the current system (DBBC2) decreases the dynamic range of the data to only 2 ADC bits and

can only process that at 1 Gbps rate. The FPGA based system we propose has 40 times the processing capability of the existing system.

4.2 ADQ7WB platform

The ADQ7WB is a high-end 12b data acquisition board with digital signal processing firmware [18]. It is designed for wide-band radio frequency applications. Table 4.1 lists the critical features for ADQ7WB.

Table 4.1: Features for ADQ7WB.

Channels	2
Sampling	5 Gsps
Resolution	12 bit
Analog BW[MHz](-1 dB)	4000
Analog BW[MHz](-3 dB)	6500
ENOB	8.8
SNR[dBc]	55
SFDR[dBc]	60
Data Transfer	Gen3×8
ADC	TI ADC12DJ2700
Transfer to PC	7 GBytes/s sustained
FPGA model	XCKU085
Buffer Memory	4 GByte

The ADQ7WB has two analog channels, each with a 5 Gsps sample rate and a 12-bit vertical resolution. The high-speed PCIe Gen3 × 8 enables efficient communication between the board and the host computer. There is a 4 GBytes data memory for data transfer, and the maximum transfer rate to PC is 7 GBytes/s.

Figure 4.1: ADQ7WB.

The FPGA chip integrated into ADQ7WB is Xilinx Kintex UltraScale XCKU085 [19], a high-performance, low total power consumption FPGA. The UltraScale architecture of this chip enables users to perform more calculations with fewer DSP resources comparing to other chips, improving both device utilization and performance. The available resources for XCKU085 are shown in Table 4.2.

Table 4.2: Available resource for XCKU085.

LUT	497520
LUTRAM	267840
FF	995040
BRAM	1620
DSP	4100
IO	624
GT	48
BUFG	1128
MMCM	22
PLL	44
PCIe	4

Fig. 4.2 illustrates the data flow path of ADQ7WB firmware. ADC data, external trigger, and timestamp information first enter the Data Trigger module. The module selects different trigger methods according to user configuration, including external trigger, internal trigger, software trigger, and level trigger. It decides which data is inserted into the trigger vector at the start of the data path. Then the data goes through User Logic 1, which has a built-in symmetrical 17 taps FIR filter for reducing the noise in unwanted signal bands. Users can change the filter coefficients via API. The Sampleskip next to User Logic 1 is used to downsampling the data and reduce the data rate. Subsequently, the Acquisition module handles triggers and controls the data flow. Next, the data enters User Logic 2, where most of our digital signal processing algorithms are deployed. In Package Generator, all the valid data is packaged and sent to memory for transferring to the host PC.

Each User Logic module has its register control bus, as shown in the upper part of Fig. 4.2. The bus can be used to interface block Random Access Memory (RAM), first in, first out (FIFO), and other custom blocks.

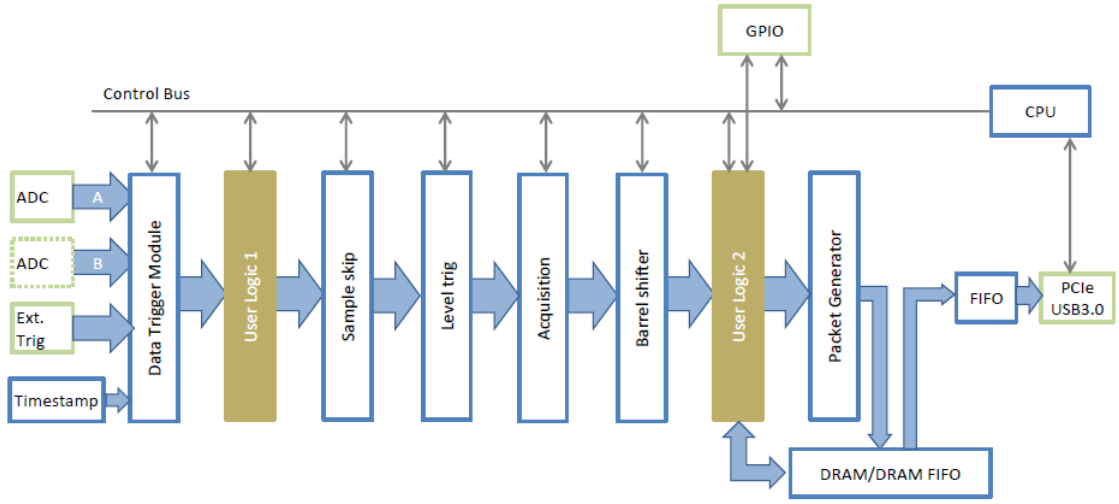

Figure 4.2: Block diagram of ADQ7WB firmware.

4.3 Hardware implementation of algorithms

The FPGA implementation is broken down into blocks, each implementing a separate portion of the algorithm. This method helps us to develop concurrent development and individual block testing process. Each block is implemented in Verilog and checked for proper functionality with Vivado Simulator. Fig. 4.3 shows a functional block diagram of the functions and algorithms implemented in the target FPGA board.

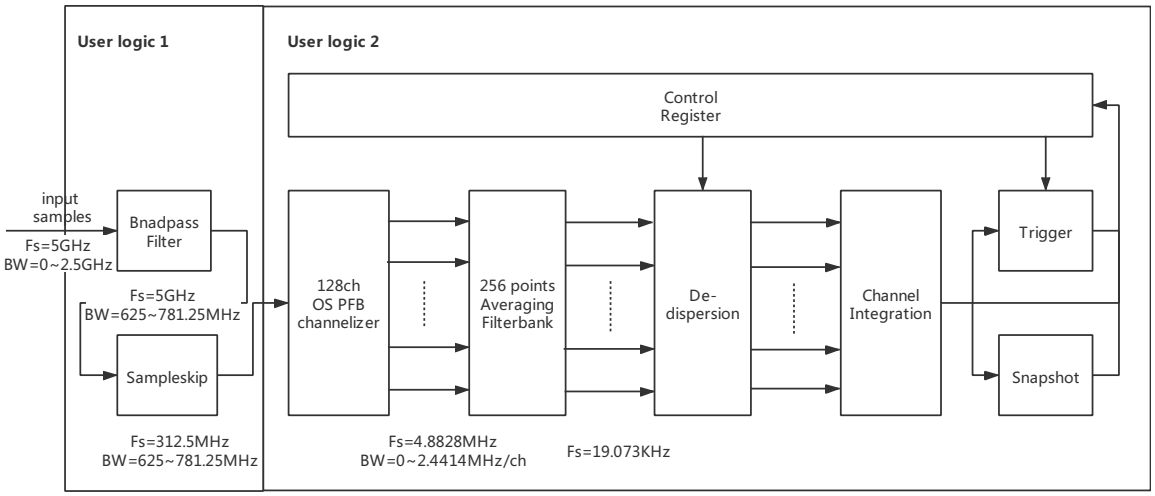

Figure 4.3: Block diagram of hardware design.

As can be seen from Fig. 4.3, the hardware system is built with eight different blocks. Each of these blocks will be illustrated in the following sections in detail. Note that the bandpass filter and sample skip functions are mapped to User Logic 1 while others are mapped to User Logic 2.

The overall resource usage, including the basic firmware design, is shown in Table 4.3.

Table 4.3: Resource utilization.

Resource	Used	Available	Utilization ratio
LUT	243129	497520	48.87%
LUTRAM	32345	267840	12.08%
FF	457110	995040	45.94%
BRAM	749.50	1620	46.27%
DSP48	3338	4100	81.41%

4.3.1 Parallel samples and data format

Before introducing the hardware implementation of the functions and algorithms needed in our system, it is necessary to know the concept of parallel design because the FPGA cannot clock at 5 Gsps (the ADC's sampling rate). Instead the ADQ7WB clock is set to the ADCs decimated clock of $1/16^{th}$ of 5 Gsps. This results in a clock rate of 312.5 MHz, which is lower than the maximum FPGA clock rate of 850 MHz and a good choice to avoid timing issues.

In ADQ7WB, although the ADC has 12 bits resolution, the format of the data inside the FPGA and out to the host PC is 16 bits. The 12 bits from the ADCs are MSB aligned to a 16 bits word. Initially the 4 LSBs are zeros. However, due to calibration and other computations in the FPGA, the entire full scale 16 bits range is output from ADQ7WB.

Since the user logic receives 16 parallel samples at each ADC clock, and each sample has 16 bits, the User Logic 1 will receive 256 bits at each clock. Table 4.4 is a detailed description of the 256 bits input and output data bus format. Note that sample 0 represents the oldest sample and sample 15 represents the newest sample.

Table 4.4: 256 bits input and output data bus format.

Bits	**Description**
Bit 0∼15	Sample 0
Bit 16∼31	Sample 1
Bit 32∼47	Sample 2
Bit 48∼63	Sample 3
Bit 64∼79	Sample 4
Bit 80∼95	Sample 5
Bit 96∼111	Sample 6
Bit 112∼127	Sample 7
Bit 128∼143	Sample 8
Bit 144∼159	Sample 9
Bit 160∼175	Sample 10
Bit 176∼191	Sample 11
Bit 192∼207	Sample 12
Bit 208∼223	Sample 13
Bit 224∼239	Sample 14
Bit 240∼255	Sample 15

4.3.2 SampleSkip

The SampleSkip module is a built-in module in ADQ7WB firmware. It takes every k^{th} set of 16 input samples. The SampleSkip module is placed between User Logic 1 and User Logic 2, as shown in Fig. 4.2. Thus, User Logic 1 runs at full sampling rate (5 Gsps) while User Logic 2 runs at decimated sampling rate.

Take sample skip by a factor of 1 (no smapleskip) and 2 as examples, Table 4.5 and Table 4.6 show how the sample skip module influence the data received in User Logic 2. Note that DC means *Don't Care*.

Table 4.5: Sampleskip when sampleskip factor = 1.

Clock cycle	**16 parallel samples**					**data_in_valid**
Cycle 0	S_{15}	S_{14}	.....	S_1	S_0	1
Cycle 1	S_{31}	S_{30}	.....	S_{17}	S_{16}	1
Cycle 2	S_{47}	S_{46}	.....	S_{33}	S_{32}	1
Cycle 3	S_{63}	S_{62}	.....	S_{49}	S_{48}	1
.....	.....	.....	.....	.....	.....	.....

Table 4.6: Sampleskip when sample skip factor = 2.

Clock cycle	**16 parallel samples**					**data_in_valid**
Cycle 0	S_{32}	S_{30}	.....	S_2	S_0	1
Cycle 1	DC	DC	.....	DC	DC	0
Cycle 2	S_{64}	S_{62}	.....	S_{36}	S_{34}	1
Cycle 3	DC	DC	.....	DC	DC	0
.....	.....	.....	.....	.....	.....	.....

In general, SampleSkip module packs up every k^{th} sample in the 256 bits vector and asserts data_valid == 1 every k^{th} clock cycle.

In our design, the sampleskip factor k is set to 16 by default, so the sampling frequency after SampleSkip module is 5 GHz/16 = 312.5 MHz.

4.3.3 Control and Data Register

The read and write of registers in each user module is accomplished through a Control Register module, specifically designed to configure the trigger threshold and the delay for each channel to adapt to different input signals. Furthermore, it is also used as a buffer for de-dispersed data transmission from the FPGA to the host PC. The signal ports for the Control Register module are listed in Table 4.7.

Table 4.7: Signal ports of Control Register module.

Signal	**Direction**	**Description**	
clk	input	1 bit	CPU clock
rst_i	input	1 bit	Active high reset
addr_i	input	14 bits	Read/Write address
wr_i	input	1 bit	Active high write strobe
wr_ack_o	output	1 bit	Active high write data qualifier
wr_data_i	input	32 bits	Write data
rd_i	input	1 bit	Active high read strobe
rd_ack_o	output	1 bit	Active high read data qualifier
rd_data_o	output	32 bits	Read data

The 14 bits address is allocated for different registers for different usages. Table 4.8 shows the allocation of these addresses.

Table 4.8: Address allocation.

Address	**Destination**	**Description**
0x10 $\sim$ 0x13	Null	Reserved for internal functions
0x14 $\sim$ 0x53	Dedispersion module	number of delay for each channel
0x54	Trigger module	Trigger threshold
0x55	Snapshot module	For reading out de-dispersed data

The first four addresses are reserved by the firmware developer for internal functions and cannot be used. Addresses 0x14 to 0x53 are used to set the number of delays for channel 0 to channel 63, respectively. Address 0x54 is used for configuring the trigger threshold, and address 0x55 is used as a buffer for transferring the de-dispersed data snapshot to the host PC. The registers in addresses 0x14 to 0x54 are both readable and writable, but 0x55 is only readable because this is where the dispersed data are stored.

4.3.4 Oversampled PFB channelizer

In our design, we used a 128-channel oversampled PFB channelizer with an oversampling ratio of 2 to divide the 312.5 MHz broadband signal into 128 narrower subchannels. Each subchannel has a bandwidth of 2.44 MHz. The advantage of using this type of channelizer has been discussed in section 2.2.4, and the reason for choosing 128 channels has been explained in chapter 3.

The signal ports for the oversampled PFB channelizer module are listed in Table 4.9.

Table 4.9: Signal ports of PFB module.

Signal	**Direction**	**width**	**Description**
clk	input	1 bit	Data clock
rst	input	1 bit	Active high reset
data_in	input	256 bits	input data
data_in_valid	input	1 bit	input data valid
data_out	output	256 bits	output data
data_out_valid	output	1 bit	output data valid

According to the structure of the oversampled PFB channelizer shown in Fig. 2.5, the FPGA implementation of the oversampled PFB channelizer is divided into four sub-blocks. Since the oversampling ratio of the PFB channelizer is 2, we can use a 2-state state machine to correct the phase rotation instead of using a circular shift buffer. Fig. 4.4 illustrates how these sub-blocks are connected to form the oversampled PFB channelizer.

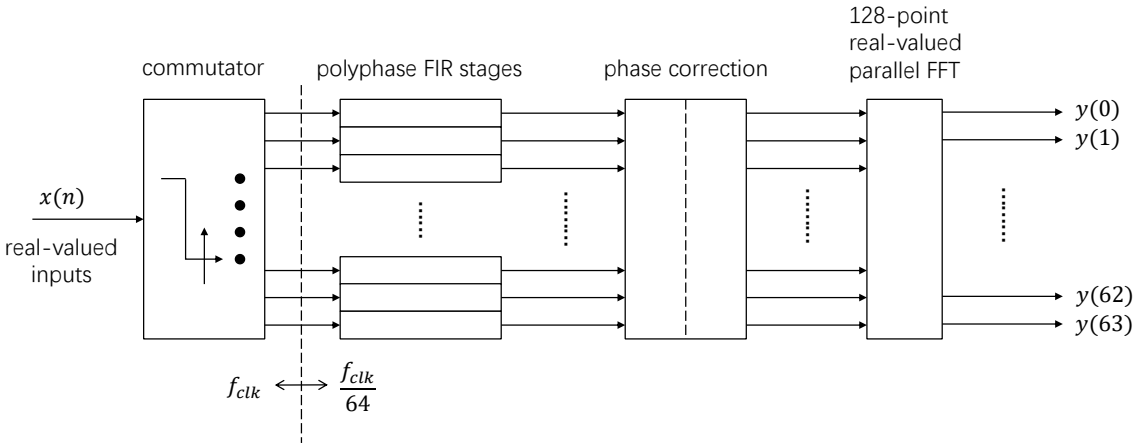

Figure 4.4: Block diagram of PFB module.

As can be seen from Fig. 4.4, the oversampled PFB channelizer is formed by a 128-path commutator switch, a 128-path polyphase FIR filterbank, and a phase correction unit, as well as a 128-point real-valued parallel FFT unit. Each of these sub-blocks will be described in the following sub-sections.

Sets of 16 data points sampled at 312.5 MHz are input to this module. A 128 point real FFT will yield 64 channels, each with a reduced bandwidth and sample rate further decimated by 64.

4.3.4.1 Commutator

The first stage of the oversampled PFB channelizer is the commutator. It distributes input samples to 128 lines and reduces clock speed by a factor of 64.

Fig. 4.5 illustrates how the commutator is implemented by two data arrays and one counter. To help explain, we name these two arrays $array1$ and $array2$ respectively, and the counter cnt.

From section 4.3.1 we know that our user logic will receive 16 parallel samples at each clock when data is valid. Since the 128-channel oversampled PFB channelizer will process 128 samples in one run, the data arrays used here will have a depth of 8 (128 samples/16 samples) and a width of 256 bits (16 samples $\times$ 16 bits per sample). Among them, $array1$ is used as a buffer for input samples, and $array2$ is the output of commutator. The data in $array2$ will be delivered to polyphase FIR filterbank for filtering. According to Table 2.2, the top half of the samples in each group are overlapping with the bottom half of samples of its next group. Therefore, the counter cnt is used to count how many sets of 16 parallel samples have been delivered to the array.

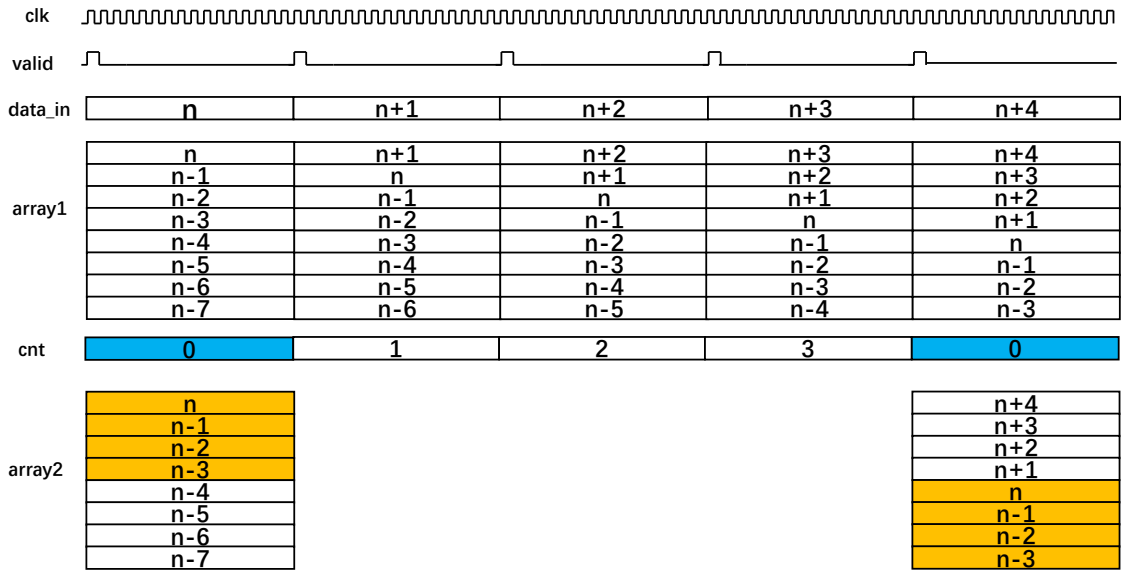

Figure 4.5: Timing diagram of commutator.

From Fig. 4.5, we can see that a new set of 16 parallel samples will be delivered to the top of $array1$ at every valid data clock. At the same time, the old sets of samples in $array1$ will be moved down by one unit, and the oldest set of 16 parallel samples will be removed from the array. Meanwhile, every time a new set of samples comes cnt increases by 1 (when the value of cnt exceeds 3, it will return to 0). When $cnt == 0$, the data in $array1$ will be buffered in $array2$ as shown in the bottom of Fig. 4.5 so that the upper part of the $array2$ is overlapping with the next $array2$. In this way, the inputs are rearranged in the form described in Table 2.2.

The clock rate is 312.5 MHz. Since data_valid signal will be only asserted every 16th clock cycle, the data rate for $array1$ is 312.5 MHz/16 = 19.53 MHz. The data in $array2$ will be only updated when $cnt == 0$, hence the data rate for $array2$ is 19.53/4 = 4.88 MHz. Thus, the input samples to the oversampled PFB channelizer have been decimated by a factor of 64 after the commutator.

4.3.4.2 Polyphase FIR filterbank

The polyphase FIR filterbank is composed of 128 FIR filter components as shown in Fig. 4.6.

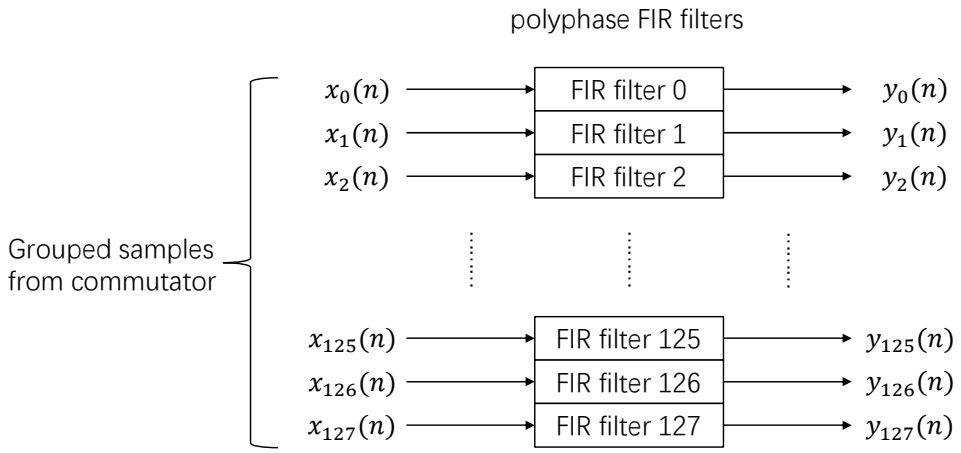

Figure 4.6: Block diagram of the polyphase FIR filterbank.

For the design of a filter, the filter order is of vital importance. We always want the order to be as high as possible to get a better filter response. However, this is unrealistic because higher order requires more resources. In this design, the prototype filter has an order of 512. Table 4.10 shows the characteristic of the prototype filter used in this design.

Table 4.10: Characteristic of the prototype filter.

Response type	Lowpass
Filter order	512
Sampling frequency	312.5 MHz
Passband frequency	0.61035 MHz
Stopband frequency	1.83105 MHz

The inputs to this module are the 128 parallel samples that come from the commutator. Each of the polyphase FIR filter components takes 4 of the 512 filter coefficients. Fig. 4.7 shows the hardware implementation of an FIR filter component.

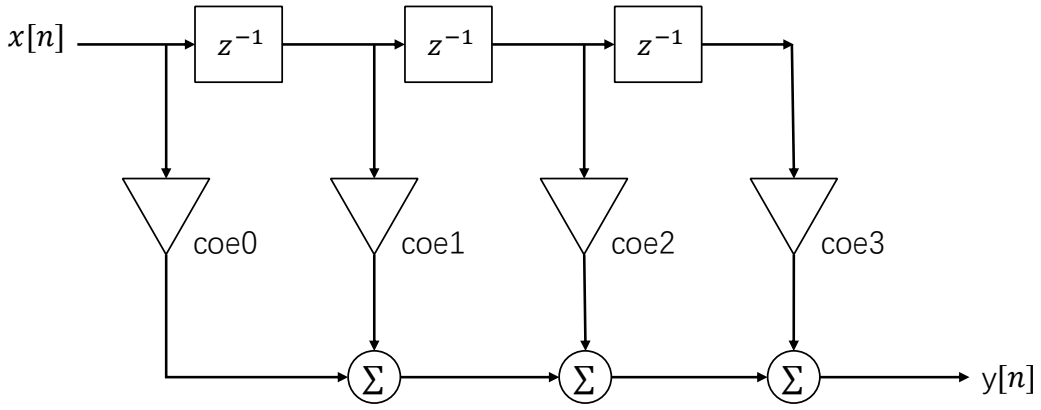

Figure 4.7: Hardware implementation of a FIR filter component.

Here $x[n]$ is the input sample, and $y[n]$ is the filter output that will be send to the circular shift buffer. The top part of Fig. 4.7 is a 4-stage delay line with four taps,

the delayed input samples will be multiplied with the filter coefficients, and its result will be accumulated to get the filter output.

4.3.4.3 Phase correction

The phase rotation at the outputs of the polyphase filterbank are corrected by a phase correction module before the FFT unit can process the samples. Its basic principles are already illustrated in section 2.2.4.

In the hardware implementation, we use a 1-bit register named flag to indicate the current state of the 2-state state machine. The value of flag toggles whenever there are valid outputs from the polyphase filterbank. When flag == 0, the outputs of polyphase filterbank are directly delivered to the FFT unit. However, when flag == 1, the upper and lower parts of the polyphase filterbank's outputs are first swapped and then sent to the FFT unit.

4.3.4.4 FFT

In section 2.2.5 we have shown that the FFT of a real-valued sequence of 2N points can be computed efficiently by employing an N-point complex FFT. Since the input to our system is a real signal, it will be more efficient to use a 128-point real-valued FFT which is formed by a 64-point complex valued FFT rather than a 128-point complex FFT. Fig. 4.8 shows a block diagram of the 128-point real valued FFT unit.

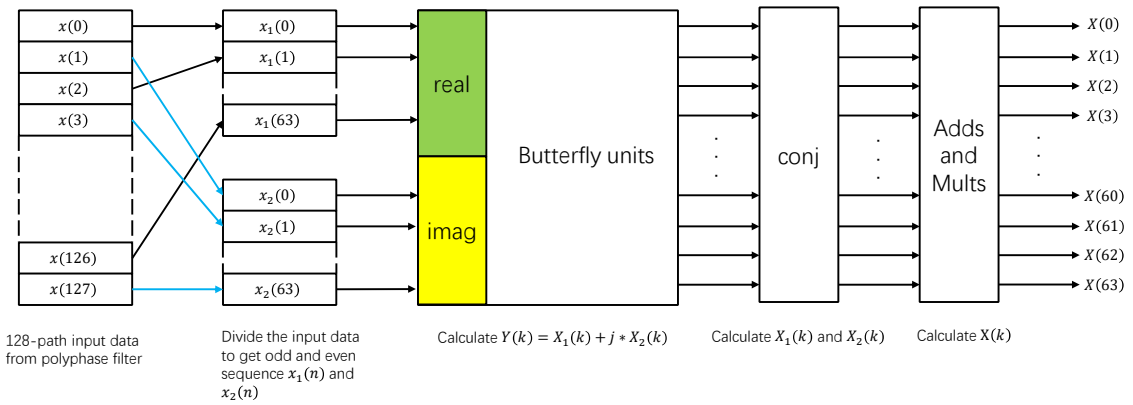

Figure 4.8: Block diagram of real valued FFT.

As can be seen from Fig. 4.8, the data from the circular shift buffer will be divided into an odd sequence $x_1(n)$ and an even sequence $x_2(n)$. Then, the odd and even sequence will be sent to the real and imaginary input of the 64-point complex FFT units, respectively, to compute the $Y(k)$. After that, $X_1(k)$ and $X_2(k)$ can be calculated from $Y(k)$ according to (2.11) and (2.12). Finally, the DFT of the input real sequence can be calculated by multiplying the twiddle factors with $X_2(k)$, and then adding $X_1(k)$ to it.

The 64-point complex FFT unit is composed of 6 stages, each with 32 butterfly units. We use MATLAB to generate the twiddle factor of each butterfly operation unit, and all are scaled with 8192 to convert it from floating-point to fix-point. The multiplications in each stage cause an increment in the number of bits and to avoid overflows we essentially divide the outputs of each stage by 8192 by truncating the LSB's. Table 4.11 shows the resource utilization of the PFB module.

Table 4.11: Resource utilization of PFB module.

Resource	Used	Available	Utilization ratio
LUT	60745	497520	12.20%
LUTRAM	17217	267840	6.42%
FF	126958	995040	12.75%
BRAM	0	1620	0%
DSP48	1664	4100	40.58%

4.3.5 Averaging Filterbank

The Averaging Filterbank module is used to increase the SNR and reduce the data rate. This permits the De-dispersion module to achieve longer time delays and the Snapshot module to store data for longer periods. The signal ports of the Averaging Filterbank module are listed in Table 4.12.

Table 4.12: Signal ports of Averaging Filterbank module.

Signal	Direction	Width	Description
clk	input	1 bit	Data clock
rst	input	1 bit	Active high reset
data_in	input	256 bits	input data
data_in_valid	input	1 bit	input data valid
data_out	output	256 bits	output data
data_out_valid	output	1 bit	output data valid
wr_en	input	1 bit	write register enable
rd_en	input	1 bit	read register enable

The Averaging Filterbank is composed of 64 block averaging filters. A block diagram of the Averaging Filterbank is shown in Fig. 4.9.

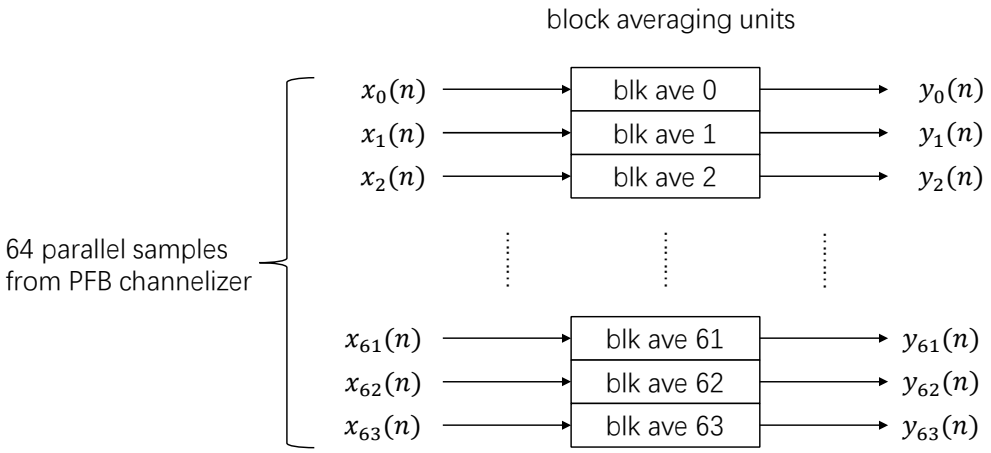

Figure 4.9: Block diagram of Averaging Filterbank.

The block averaging filter divides the input data into blocks, and each block is treated as an individual data point and calculates the average from that. The algorithm flowchart is shown in Fig. 4.10.

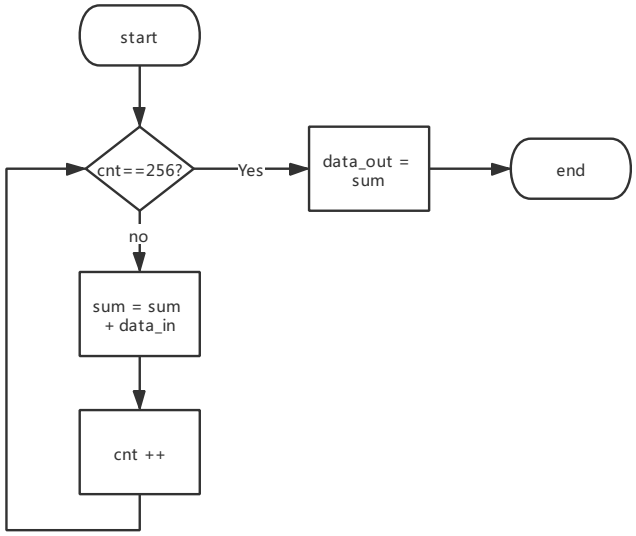

Figure 4.10: Block averaging flowchart.

There is a counter which counts the number of sums in the variable *cnt*. If *cnt* is less than 256, which is the integration length in our design, then the input data will be added to the variable *sum* where the cumulative sum of input data is stored. At the same time, *cnt* will be increased by 1. When *cnt* reaches 256, then the output of this block averaging algorithm is the cumulative sum of the 256 input data. Note that the integration length is fixed.

Since we are considering a block of 256 samples as an individual sample, the data frequency has been decimated by a factor of 256:

$$f_{datanew} = \frac{f_{data}}{ave\_length} = \frac{4.8828\,\text{MHz}}{256} \approx 19\,\text{kHz} \tag{4.1}$$

Thus, the time resolution of the system is now approximately 52 μs. Table 4.13 shows the resource utilization of the Average Filterbank module.

Table 4.13: Resource utilization of Averaging Filterbank module.

Resource	Used	Available	Utilization rate
LUT	468	497520	0%
LUTRAM	0	267840	0%
FF	3072	995040	0.31%
BRAM	0	1620	0%
DSP48	0	4100	0%

4.3.6 De-dispersion

The De-dispersion module compensates for the time delays of frequency components in different frequency channels. Different lengths of delay are added to each of the 64 frequency channels so that the target frequency component can arrive simultaneously. The signal ports of the De-dispersion module are listed in Table 4.14.

Table 4.14: Signal ports of De-dispersion module.

Signal	Direction	width	Description
clk	input	1 bit	Data clock
rst	input	1 bit	Active high reset
data_in	input	256 bits	input data
data_in_valid	input	1 bit	input data valid
data_out	output	256 bits	output data
data_out_valid	output	1 bit	output data valid
wr_en	input	1 bit	write register enable
rd_en	input	1 bit	read register enable
addr	input	14 bits	register address
cmd_ch_dly	input	2048 bits	delay per channel (32 bits)

The number of delays to each channel is pre-calculated in MATLAB according to (3.3), and it can be configured in real-time via the Control Register module. A block diagram of the De-dispersion module is shown in Fig. 4.11.

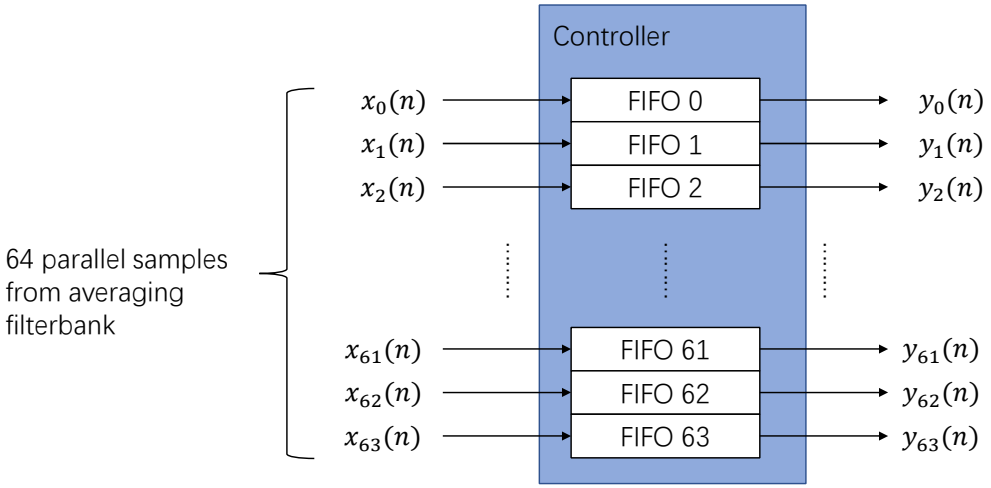

Figure 4.11: Block diagram of De-dispersion module.

As shown in Fig. 4.11, the delay is achieved by 64 FIFOs in parallel. Ideally, the length of each FIFO could be different since the highest frequency channel needs the longest delay while lower frequency channels do not. But because of implementation reasons, we make them all the same fixed length. The write and read of these FIFOs are controlled by a controller. The data are written into a FIFO, and the write enable signal of the FIFO is set to high at the very beginning while the read enable signal stays low. At the same time, a counter starts to count the number of clock cycles, and when the value of the counter is equal to the number of delays, the read enable signal will be set to high. After this, the delayed data will be read out from the FIFOs.

Because of the FPGA chip's limited BRAM resources, extremely large FIFOs are not possible in the design. For the time being, the FIFO used in the design has a depth of 8192, implying that the De-dispersion module can achieve a maximum of 8192 clock cycles of delay. In this case, the maximum delay in terms of time is $8192 \times 52\ \mu\text{s} = 425$ ms.

It is also worth noting that the FIFO and the counter will be reset whenever the number of delays changes.

Table 4.15 shows the resource utilization of the De-dispersion module.

Table 4.15: Resource utilization of De-dispersion module.

Resource	Used	Available	Utilization ratio
LUT	9496	497520	1.90%
LUTRAM	0	267840	0%
FF	14080	995040	1.41%
BRAM	256	1620	15.80%
DSP48	0	4100	0%

After the De-dispersion module, a channel integration module is built to integrate

all 64 channels together to form the de-dispersed time series or matched filter. The matched filter output indicates the strength of the de-dispersed pulse. When the output exceeds a pre-programmed level a trigger is generated.

4.3.7 Trigger and snapshot

The trigger module is used to generate a trigger based on the trigger threshold. The signal ports of the Trigger module are listed in Table 4.16.

Table 4.16: Signal ports of Trigger module.

Signal	Direction	width	Description
clk	input	1 bit	Data clock
rst	input	1 bit	Active high reset
data_in	input	22 bits	input data
data_in_valid	input	1 bit	input data valid
cmd_threshold	input	32 bits	trigger threshold
trigger	output	1 bit	trigger

If a sample from the Channel Integration module is greater than the threshold and the previous sample, then trigger.

4.3.8 Snapshot

The snapshot module is used to buffer the FRB snapshot. A great feature of the system is the ability to see what is going on before the trigger. Therefore, a 'Mid-display trigger' mechanism is implemented in the Trigger module to buffer the data before the trigger. The basic idea is that the Snapshot module continuously collects data, and the memory gets overwritten repeatedly. However, if a trigger occurs, the Trigger module will continue to collect for half more of its memory depth before stopping. It keeps half of its memory with what happened before the trigger and half of what happened after.

The implementation is achieved by a Finite State Machine (FSM), and it is illustrated in Fig. 4.12.

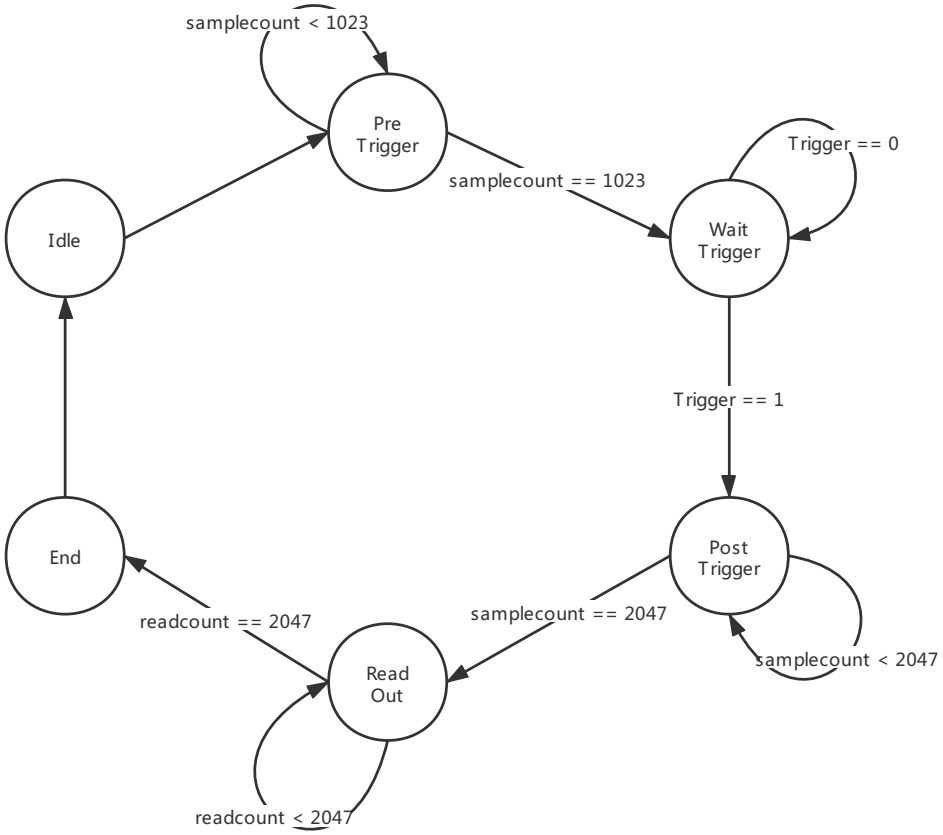

Figure 4.12: A state machine diagram of Snapshot module with 6 states.

The finite state machine consists of 6 states in total. The Idle state is where the state machine starts and where all the registers are initialized. In the PreTrigger state, data is kept written to the RAM, and when half of the RAM's memory is filled with data, the state machine goes to the next state. In WaitTrigger state, the counter stops counting, but the data is written to the RAM repeatedly. Moreover, when a trigger is detected, the state machine goes to PostTrigger state. In this state, the trigger position is recorded, and the counter starts counting again to write half more data and then stops. The state machine goes to ReadOut state, where the data in RAM is delivered to a FIFO. After all the data in RAM is delivered to the FIFO, the state machine ends and then starts over.

The de-dispersed data will be delivered to a register with an address of 0x55 in the Control Register module (See Table 4.8) so that the data can be transferred to the host computer for post signal processing. However, the clock domain between the Control Register module and other signal processing modules is different. So an independent clock FIFO is used here to deal with the cross time-domain issue. Table 4.15 shows the resource utilization of the Trigger module.

5

System test and results

In this chapter, we will mainly introduce the testing methods used to test our system and show the test results that we have.

Due to practical issues, we are not able to use the telescope to test our system. Thus, all the testing is carried out with the signal generator Rohde&Schwarz SMP02 [20]. The frequency range of this signal generator is from 10 MHz to 20 GHz, and the output level is from -20 dBm to 11.5 dBm, which is enough for us to emulate the high frequency range of an FRB signal.

The test is mainly divided into two parts. One is to test the performance of the oversampled PFB channelizer, and the other is to test the ability to detect FRBs. We will introduce these results separately in the following two sections.

5.1 Oversampled PFB channelizer

The purpose of testing the oversampled PFB channelizer is mainly to check if the frequencies can arrive in the correct channel and the resolution can match our expectation. The method is to use the signal generator to generate two sine signals with different frequencies, feed them into the system, and observe whether the frequencies can arrive in the correct channel and whether the PFB channelizer can distinguish these two different frequencies.

According to section 2.2, the bandwidth for each channel is $\frac{312.5}{128} = 2.44$ MHz, which means the channelizer would have a resolution of 2.44 MHz theoretically. Therefore, if we input two signals with a frequency difference of 2.44 MHz to the channelizer, then at the output of the channelizer, these two signals should appear in two adjacent channels.

In the actual test, we input two signals with frequencies of 650 MHz and 652.44 MHz into the channelizer, and the results are shown in Fig. 5.1.

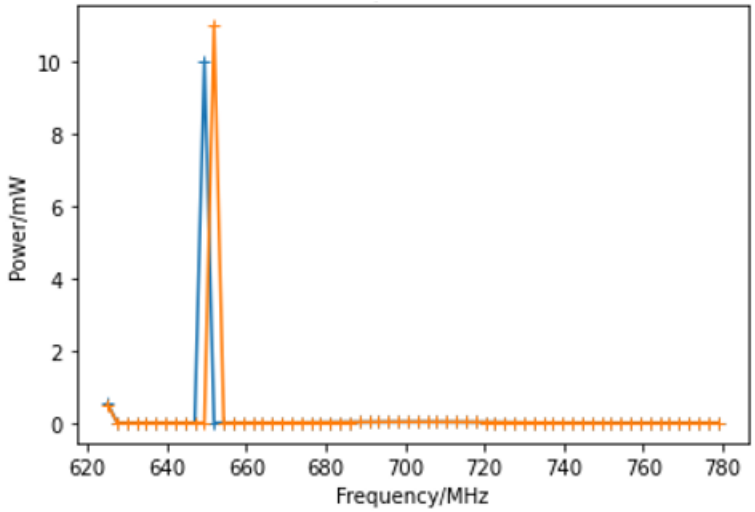

Figure 5.1: Channelizer outputs when f=650 MHz and 652.44 MHz.

In Fig. 5.1, the x-axis is the center frequency of each channel, which is obtained by equation 2.3 and the y-axis is the signal power in each channel. As can be seen from the figure, it is clear that these two frequencies appear in the correct channels and the channelizer is able to distinguish them.

For further verification, we randomly select another set of inputs: 708.56 MHz and 711 MHz. The result is shown in Fig. 5.2.

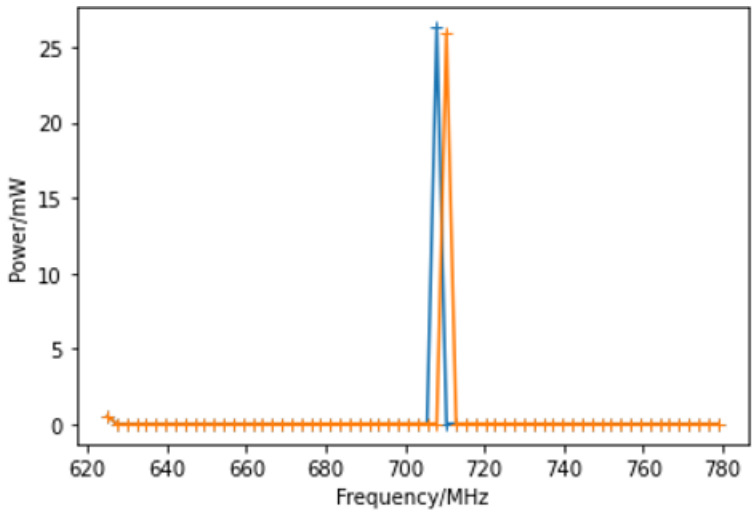

Figure 5.2: Channelizer outputs when f=708.56 MHz and 711 MHz.

Similarly, the frequencies arrive in the correct channels and the channelizer could distinguish these two input signals as shown in Fig. 5.2. Hence, according to the above tests, we can conclude that our oversampled PFB channelizer works and the resolution of it is 2.44 MHz which is the same as the theoretical resolution.

It would also be interesting to know the frequency response within one channel. A feasible method is to use a signal to sweep the entire bandwidth of a chan-

nel. From previous sections we know that the bandwidth BW for each channel is 312.5 MHz/128 = 2.44 MHz and the sampling frequency fs for each channel is 312.5 MHz/64 = 4.88 MHz. We normalized the frequency with Nyquist frequency ($fs/2 = 2.44$ MHz). Fig. 5.3 shows the frequency response for one channel.

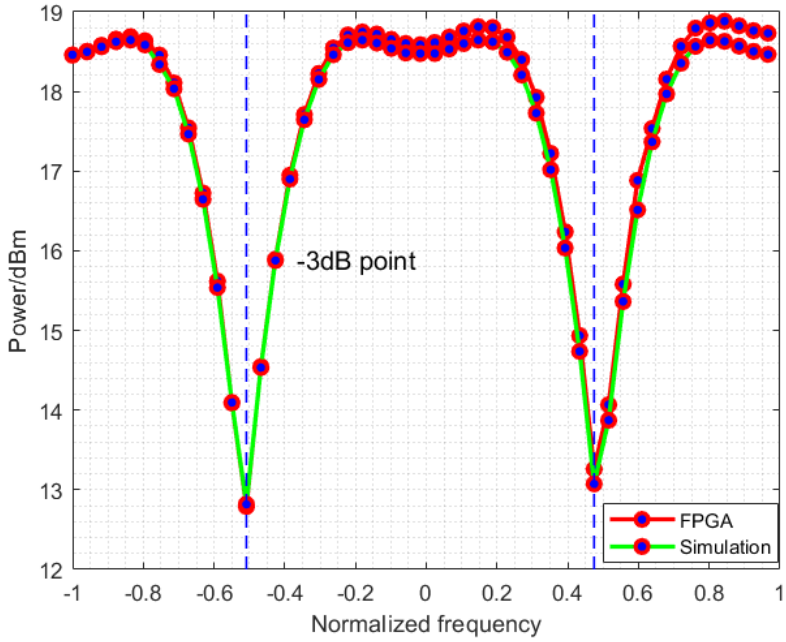

Figure 5.3: Frequency response within one channel.

In Fig. 5.3, the green and red lines are the results from MATLAB simulation and FPGA, respectively. It can be seen from the figure, except for the slight difference in magnitude, that the FPGA result is basically the same as the MATLAB simulation. The main reason for this difference is that all operations in the simulation use floating-point numbers while in FPGAs, fixed-point numbers are used. In addition, we can also get from the figure that the channel bandwidth is 1 ($1 \times 2.44 = 2.44$ MHz), the passband of each channel is 0.5 ($0.5 \times 2.44 = 1.22$ MHz), and the one-sided transition band is 0.25 ($0.25 \times 2.44 = 0.61$ MHz), which is consistent with our filter design as discussed in Table 4.10.

5.2 FRB detection

From section 1.1, we know that one of the most obvious feature for an FRB signal is that its frequency will change in time due to the effect of dispersion. Therefore, we can use the signal generator to generate a linear step sweep signal to emulate this feature of the FRB signal. Table 5.1 shows the characteristics of the linear step sweep signal that we used for testing.

Table 5.1: Characteristics of linear step sweep used for testing.

Start frequency	625 MHz
Stop frequency	725 MHz
Frequency span	100 MHz
Frequency step	10 MHz
dwell time	10 ms

The frequency of this test signal starts at 625 MHz, and the frequency of the signal increases by 10 MHz every 10 ms until the signal frequency reaches 725 MHz. The relationship between frequency and time is shown in Fig. 5.4.

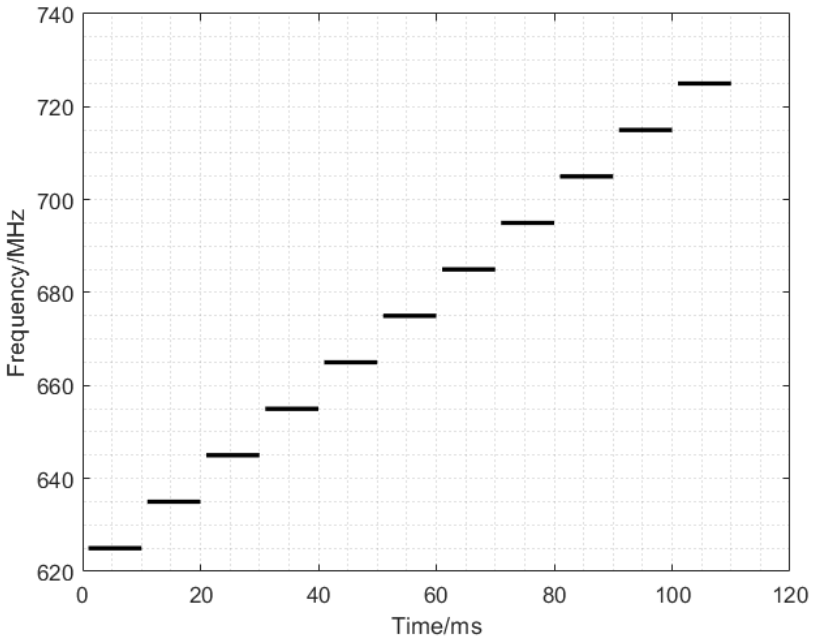

Figure 5.4: Time vs frequency for the test signal.

If the system can successfully de-disperse the test signal whose frequency changes with time, all its frequency components will appear simultaneously, as shown in Fig. 5.5.

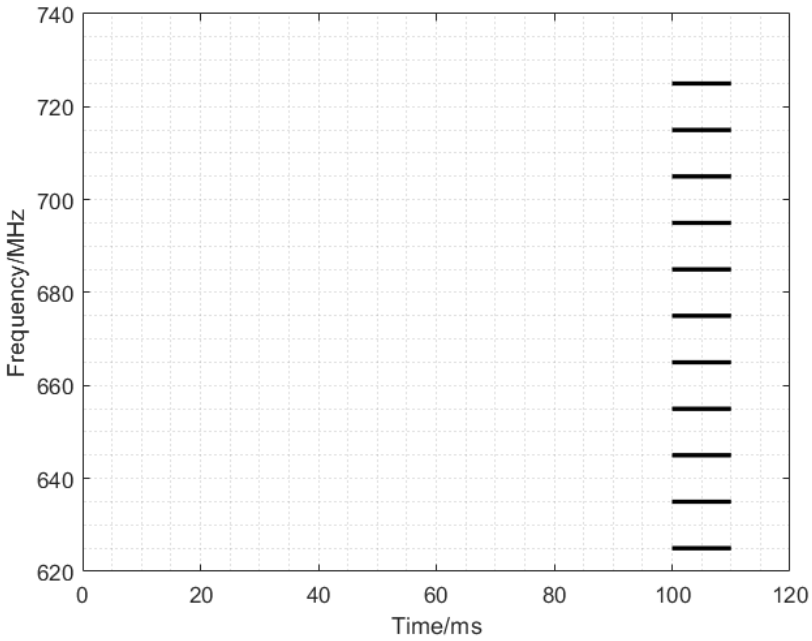

Figure 5.5: Time vs frequency for the test signal after de-dispersion.

At this point, if we add up all the frequency components, we will get an evident pulse. By comparing the magnitude of the pulse with the threshold we set, we can judge whether the target signal is captured. If the target signal is captured, then we can analyze the data at the trigger time and plot the power spectrum of it. Theoretically, its power spectrum should have eleven frequency components, as shown in Fig. 5.6.

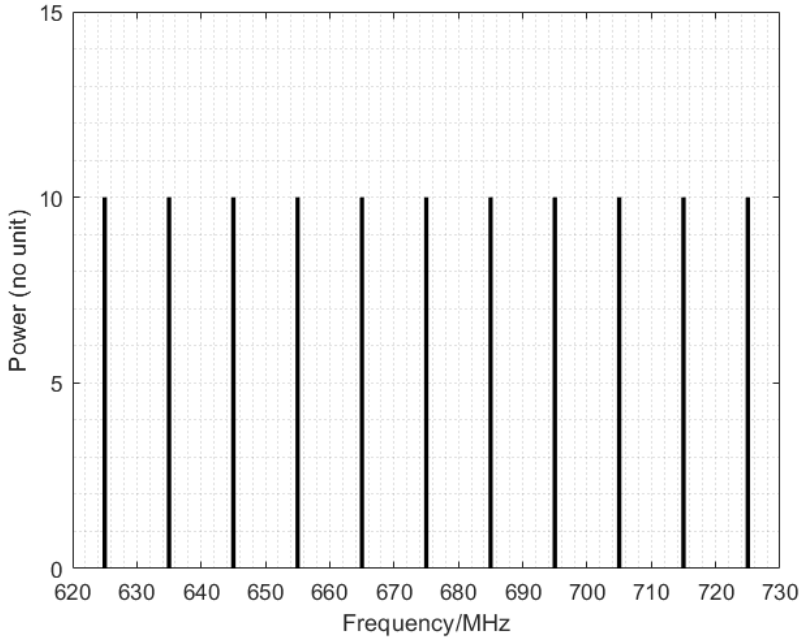

Figure 5.6: Power spectrum of the target signal.

Then we use this test signal together with noise as the input to verify the function of the hardware system. The SNR of the input signal is 0 dB. Fig. 5.7 shows the de-dispersed time series after de-dispersion, and Fig. 5.8 shows the power spectrum of the de-dispersed data snapshot.

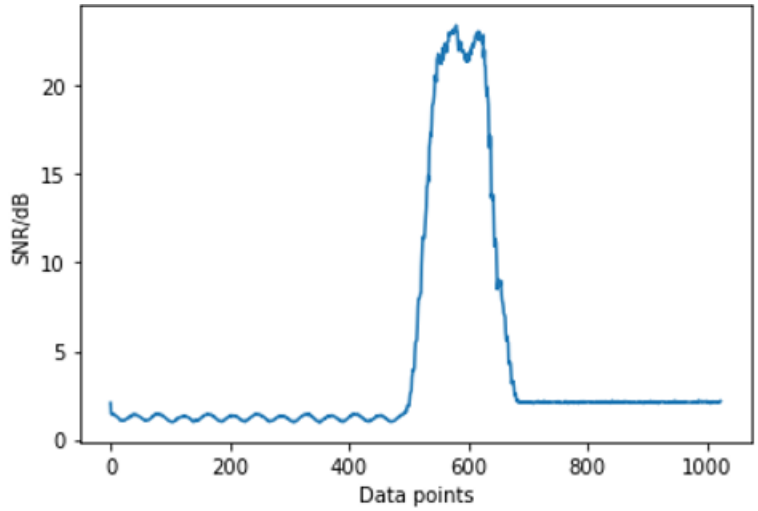

Figure 5.7: Dedispersed time series.

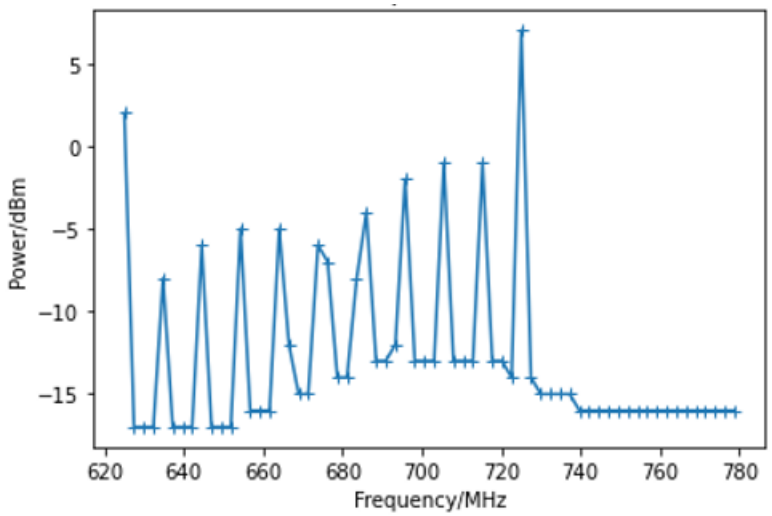

Figure 5.8: Power spectrum of the de-dispersed data snapshot.

As shown in Fig. 5.7, the dedispersed time series are the summation of the data over all frequency channels in a way that follows the dispersive sweep. From Fig. 5.7 we can clearly see that there is an SNR improvement of around 23 dB compared to the input signal's SNR.

As we introduced in the previous sections, this time series (pulse) will be used to indicate whether we have detected the target signal. If the value of the time series at a particular moment exceeds the trigger threshold we set, the system considers that

the target signal is detected at this moment, and a snapshot of this target signal will be sent to the PC for post-processing and analysis.

Fig. 5.8 is the spectrum analysis of the snapshot, we can see that there are 11 spikes in this figure and each of these spikes represents one frequency component. From left to right, they are 625MHz, 635MHz, 645 MHz, 655 MHz, 665 MHz, 675 MHz, 685 MHz, 695 MHz, 705 MHz, 715 MHz, and 725 MHz. This result is basically in line with our expectations (see Fig. 5.6), except that the magnitude of each frequency component is different. The reason is that these frequencies do not exactly fall on the center frequency of each channel. The farther away from the center frequency (see Fig. 5.3), the greater the attenuation, so the amplitude will be different.

Next, we adjust the SNR of the input signal to find the limitation of the hardware system for FRB detection. The test conditions and results are described in Table 5.2.

Table 5.2: System test.

Input SNR (dB)	0	-4	-8	-12	-16	-20	- 24
FPGA results							
Dedispersed SNR (dB)	23	17.4	11.8	8.2	4.2	2.0	1.2
SNR improvement (dB)	23	21.4	19.8	20.2	20.2	22.0	25.2
MATLAB simulation results							
Dedispersed SNR (dB)	30.0	25.5	23.0	17.7	13.2	9.2	5.0
SNR improvement (dB)	30.0	29.5	31.0	29.7	29.2	29.2	29.0

The upper part of Table 5.2 is the test result of the FPGA and the lower part is the simulation result from MATLAB which we had presented in Table 3.6. From the table, we know that our hardware system can significantly improve the SNR of the signal. Therefore, even when the SNR of the input signal is very low, we can still detect the FRB. Compared to the simulation, the actual test result and the simulation result as shown in Table 5.2 are roughly the same. The difference is that the SNR improvements in FPGA are lower than that in the simulation. One reason is that FPGA does not have the feature of floating point. So all the data processed in FPGA are fix point which means we have to discard the decimal part of the data. This result the hardware system to lose some accuracy compared to MATLAB simulation. Another reason is that when we process data in FPGA, in order to prevent the data bit width from expanding continuously, the data is truncated to 16 bits at each modules' output. This causes the output value of some modules to be smaller than the MATALB simulation which also affect the output SNR.

6
Conclusion

In this work, we have developed a new backend system for FRB capturing in the commercial FPGA platform ADQ7WB. The needed algorithms for detecting FRB events such as channelizer and de-dispersion algorithms have been reviewed and evaluated. A simulation model of the DSP chain has been built to explore the parameters of these algorithms. The hardware implementation of these algorithms has been optimized to reduce resource usage, but it still has much room for improvement. The performance of the hardware system has been tested via a signal generator.

The system is flexible, and most of the internal parameters of the modules can be programmed in real time via the application program interface (API). For example, the delay of each channel in the de-dispersion module and the trigger threshold in the triggering module are both programmable. This flexibility makes the system able to deal with different FRBs.

The system can detect target FRBs even when the input SNR is low. In the simulation, the dispersed target FRB can be detected without false triggers down to an input SNR of -20 dB. When the input SNR is -24 dB, 42.5% of the triggers are false triggers. Even so, it shows that 57.5% of the triggers are valid, which can reduce the system‘s data rate significantly. The hardware system test results show that it can detect the target FRB when the input SNR is as low as -24 dB, and it has an SNR improvement of 22 dB on average.

From a qualitative perspective, the system can effectively screen out FRB candidates in observations by giving a reasonable trigger threshold (lower than the highest de-dispersed pulse). For extreme noisy inputs, false triggers may happen. However, the data rate can still be reduced compared to the current system, which records and transfers all the data, including pure noise data in the observations.

The current system only processes one channel of ADC samples, but it has already taken 80% of the DSP resources in the target FPGA chip, which means that we might not have enough DSP resources for the second ADC channel. Due to time limitations, we cannot improve the efficiency of hardware implementation in this project. So, the future work will be to further optimize the algorithms implemented in hardware, especially the PFB, since it takes up the most resources. The currently used PFB does 128 channels which is a waste of resources since those channels are working at a fraction of the clock rate. A 16-channel PFB would be enough to decimate the data rate to the FPGA basic clock rate of 312.5 MHz. After the PFB,

a serial FFT can be implemented to generate more channels.

So far, the system can generate a trigger when it detects an FRB candidate but can not capture the raw samples and transfer them to the host PC. Thus, one can improve the system by implementing a data capture function that can transfer the raw data to the host PC once a trigger is generated.

Furthermore, the system could be improved by having variable bit rates. The system can send coarse quantization data continuously to the host PC, and when an FRB event is triggered, it switches to full resolution. In this way, the system can be used as a continuous recorder which does not lose any samples within the observation.

In addition, the system could have a graphical user interface (GUI) that can control the hardware system, display and analyze the collected data. An event notification function can also be added to flag FRB candidates in real time.

Bibliography

[1] D. Lorimer, M. Bailes, M. McLaughlin, D. Narkevic, and F. Crawford, "A bright millisecond radio burst of extragalactic origin," *Science*, vol. 318, no. 5851, pp. 777–780, 2007.

[2] E. Petroff, J. Hessels, and D. Lorimer, "Fast radio bursts," *The Astronomy and Astrophysics Review*, vol. 27, no. 1, pp. 1–75, 2019.

[3] D. H. MacMahon, D. C. Price, M. Lebofsky, A. P. Siemion, S. Croft, D. DeBoer, J. E. Enriquez, V. Gajjar, G. Hellbourg, H. Isaacson *et al.*, "The breakthrough listen search for intelligent life: A wideband data recorder system for the Robert C. Byrd Green Bank Telescope," *Publications of the Astronomical Society of the Pacific*, vol. 130, no. 986, p. 044502, 2018.

[4] J. T. Luo, L. Chen, J. L. Han, A. Esamdin, Y. J. Wu, Z.-X. Li, L.-F. Hao, and X.-Z. Zhang, "A digital pulsar backend based on FPGA," *Research in Astronomy and Astrophysics*, vol. 17, no. 1, p. 9, 2017.

[5] J. M. Cordes and S. Chatterjee, "Fast Radio Bursts: An Extragalactic Enigma," *Annual Review of Astronomy and Astrophysics*, vol. 57, pp. 417–465, 2019.

[6] A. Sclocco, J. van Leeuwen, H. E. Bal, and R. V. van Nieuwpoort, "Real-time dedispersion for fast radio transient surveys, using auto tuning on many-core accelerators," *Astronomy and computing*, vol. 14, pp. 1–7, 2016.

[7] G. Tuccari, W. Alef, A. Bertarini, S. Buttaccio, G. Comoretto, D. Graham, A. Neidhardt, P. R. Platania, A. Russo, A. Roy *et al.*, "DBBC2 backend: Status and development plan," in *IVS General Meeting Proceedings*, 2010, pp. 392–395.

[8] B. B. Andrew Jameson. Heimdall: Transient detection pipeline. (2021, June 9). [Online]. Available: https://sourceforge.net/projects/heimdall-astro/

[9] M. Wall, "Scientists find 13 mysterious deep-space flashes, including 2nd known 'repeater'," *SPACE*. [Online]. Available: https://www.space.com/42943-fast-radio-burst-repeater-new-discovery.html

[10] C. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, jan 1949. [Online]. Available: https://doi.org/10.1109/jrproc.1949.232969

[11] R. G. Vaughan, N. L. Scott, and D. R. White, "The theory of bandpass sampling," *IEEE Transactions on signal processing*, vol. 39, no. 9, pp. 1973–1984, 1991.

[12] F. Wu, "FPGA based uniform channelizer implementation," Ph.D. dissertation, National University of Ireland Maynooth, 2016.

[13] L. Arnaldi and H. Dellavale, "Oversampled filter bank channelizer for cryogenic detectors," *Review of Scientific Instruments*, vol. 92, no. 2, p. 023304, 2021.

[14] R. E. Crochiere and L. R. Rabiner, "Interpolation and decimation of digital signals—a tutorial review," *Proceedings of the IEEE*, vol. 69, no. 3, pp. 300–331, 1981.

[15] J. Tuthill, G. Hampson, J. D. Bunton, F. Harris, A. Brown, R. Ferris, and T. Bateman, "Compensating for oversampling effects in polyphase channelizers: A radio astronomy application," in *2015 IEEE Signal Processing and Signal Processing Education Workshop (SP/SPE)*. IEEE, 2015, pp. 255–260.

[16] D. R. Lorimer and M. Kramer, "Handbook of pulsar astronomy," *Handbook of Pulsar Astronomy*, 2012.

[17] H. Nyquist, "Certain topics in telegraph transmission theory," *Transactions of the American Institute of Electrical Engineers*, vol. 47, no. 2, pp. 617–644, 1928.

[18] *ADQ7WB Datasheet*, TELEDYNE SP DEVICES, 2021, 1(27). [Online]. Available: https://www.spdevices.com/documents/datasheets/92-19-2226-adq7wb-datasheet/file

[19] *Kintex UltraScale FPGAs Data Sheet DC and AC Switching Characteristics*, XILINX, 2020, v1.19. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds892-kintex-ultrascale-data-sheet.pdf

[20] *Microwave Signal Generator R&S SMP*, ROHDE&SCHWARZ, 2007, v05.01. [Online]. Available: https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_common_library/dl_brochures_and_datasheets/pdf_1/SMP_dat_en.pdf