Spectrum Access System: Design and Implementation of the Decision-Feedback Equalizer in Hardware

Ci Chen

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2017-83
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-83.html

May 12, 2017

Copyright © 2017, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

Professor Anant Sahai, University of California, Berkeley
Professor John Wawrzynek, University of California, Berkeley
Christopher Yarp, University of California, Berkeley
Colin de Vrieze, University of California, Berkeley
James Martin, University of California, Berkeley
Liheng Zhu, University of California, Berkeley
Kaidi Du, University of California, Berkeley

University of California, Berkeley College of Engineering

MASTER OF ENGINEERING - SPRING 2017

EECS

Signal Processing and Communication

Spectrum Access System: Design and Implementation of the Decision-Feedback Equalizer in Hardware

Ci Chen

This **Masters Project Paper** fulfills the Master of Engineering degree requirement.

Approved by:

1. Capstone Project Advisor:

Signature: ______________________ Date 5-5-17

Print Name/Department: ANANT SAHAI EECS

2. Faculty Committee Member #2:

Signature: ______________________ Date May 12, 2017

Print Name/Department:

JOHN WAWRZYNEK EECS

Abstract

The purpose of my capstone project is to design and implement a decision feedback equalizer (DFE) in ASIC technology for the Defense Advanced Research Projects Agency (DARPA) spectrum challenge. The equalizer is to reduce the intersymbol interference in the receiver side the communication system. To accomplish this task, testing signals were written in MATLAB according to the DARPA challenge. Then I implemented the DFE in MATLAB to test the functionality. The entire design comprises a preamble correlator, a decision device, and an adaptive LMS-feedback filter. Furthermore, I worked with my CS250 teammates to implement the design in hardware using the Chisel HDL language, and synthesized with Synopsis Design Compiler, and the final layout was generated using Synopsis IC Compiler. The design was evaluated for its use of chip area and its power consumption. Finally, our CS250 team optimized the DFE design and explored design spaces such as using a SRAM to replace the shift registers.

Spectrum Access System:

Design and Implementation of the Decision-Feedback Equalizer in Hardware

Master of Engineering Capstone Report

Ci Chen

with Heyi Sun and Zhuangyi Zhao

Friday, May 5th, 2017

Acknowledgements

Professor Anant Sahai, University of California, Berkeley

Professor John Wawrzynek, University of California, Berkeley

Christopher Yarp, University of California, Berkeley

Colin de Vrieze, University of California, Berkeley

James Martin, University of California, Berkeley

Liheng Zhu, University of California, Berkeley

Kaidi Du, University of California, Berkeley

Table of Contents

Chapter 1. Individual Technical Contribution **5**
1. Project and Chapter Introduction **5**
1.1. Background of the DARPA challenge 5
1.2. M.Eng Team Project 6
1.3. Purpose of chapter 1 7
1.4. Introduction of the project testing environment 8
2. Equalizer Design in Software and Verification **9**
2.1. Reasons to pick DFE 9
2.2. Introduction of the DFE algorithm 10
2.3. DFE design modification 11
2.4. Implementing a Golay correlator 12
2.5. Software design verification and conclusion 14
3. Hardware Implementation **16**
3.1. Theory of operation 16
3.1.1. Correlator 17
3.1.2. Decision Block 18
3.1.3. Feedback Filter 18
3.1.4. Control Unit 19
3.2. Testing and verification 20
3.3. Design space exploration 21
3.3.1. Fixed point representation 22
3.3.2. Feedback filter optimization 23
3.3.3. SRAM and register 25
3.4. Chip layouts 26
4. Conclusion and Future Work **28**
4.1. Project summary 28
4.2. Project reflection 28
4.3. Future work 29
Reference 1 ..... **31**
Chapter 2. Engineering Leadership Paper **33**
1. Market Analysis 33
2. Project Management 34
3. Ethics 36
Reference 2 ..... **38**
Appendix A: [United States Frequency Allocations, 2013] **39**
Appendix B: Golay 128 Sequence **40**
Appendix C: Software and Hardware Code **41**

Chapter 1. Individual Technical Contribution

1. Project and Chapter Introduction

1.1. Background of the DARPA challenge

The spectrum access system project is part of the Defense Advanced Research Projects Agency (DARPA) challenge project at University of California, Berkeley. The aim of the DARPA spectrum challenge is to achieve a cooperative communication system. Traditionally, signals are transmitted by using different channels or frequencies. According to the Federal Communications Commission, these channels are categorized in two types: licensed and unlicensed. Licensed channels are granted to specific signal providers or users such as AT&T, Verizon, and Union Pacific Railroad to use statically and exclusively (Accessing Spectrum, n.d.). The main benefit of having licensed channels is that they prevent interference from other users and guarantee the availability of bandwidth (Railroad Frequencies, n.d.). Alternatively, unlicensed channels are channels not assigned or reserved to specific users and can be used by the general public. However, if more than one user is utilizing the same channel at the same time frame, interference might occur. The "United States Frequency Allocations" chart presents a complete summary of the spectrum allocation in the US (see appendix A). According to the chart, this traditional method fixes the usage of different frequencies and thus, in a sense, wastes the frequency resources when the channel is not currently used.

The DARPA challenge proposed a cooperative communication system. Participants will create a new, efficient wireless paradigm in which radio networks autonomously collaborate to dynamically determine how the spectrum should be used moment to moment (Defense Advanced Research Projects Agency).

There are three main advantages of the new wireless system. Firstly, the system dynamically allocates the bandwidth resources according to the user's need. By doing this, it reduces the amount of the idle time in the licensed bandwidth. Secondly, by introducing the autonomous and collaborative algorithm, the communication system also reduces the possibility of collisions which is one of the main concerns from the unlicensed band. Thirdly, the ability to cooperate between differently designed systems will free the designers from the standard communicating protocols. Through this advantage, the new corporative system provides platforms for more innovative designs in the communication field.

1.2. M.Eng Team Project

The M.Eng project focuses on implementing an equalizer in the hardware for the DARPA challenge team. An equalizer, introduced to recover the effect of intersymbol interference (ISI), is very crucial in the real world communication system. ISI results from the channel model, which is depicted as an example from Figure 1 and can be represented as:

$$r[t] = u[t] * h_c[t] + n[t] \qquad [1]$$

where u[t] is the transmitted signal, $h_c[t]$ is the impulse responses of the channel, n[t] is AWGN noise, and * is the convolution operation. $r[t]$ is the receiving signal (Wang, n.d.).

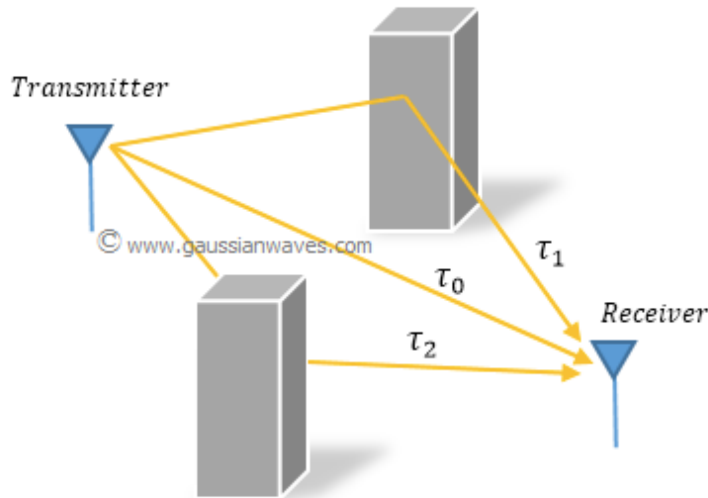


Figure 1. Multipath ISI demonstration (Mathuranathan, 2014).

The goal of our project is to explore the best algorithm to achieve the equalizer's functionality and implement the algorithm in hardware. Programming the equalizer in the software requires the hardware to transfer large amount of sampled data to the software in each second. For example, if the sample rate is 20MHz, then the hardware will transmit 20 million symbols every second to the software. This transmission requires a large amount of bandwidth. In addition, the hardware version will also run faster than the software. This fact will ultimately decrease the latency. Due to these two main reasons, it is necessary to implement an equalizer in the hardware.

1.3. Purpose of chapter 1

The purpose of Chapter 1 is to present my individual work and how it contributes to the team. My primary responsibility was to implement the equalizer in hardware. To accomplish this task, I first modified the equalizer algorithm according to the test environment and verified it in the software. Then I modified the software to fit into the hardware design. Finally, I optimized the hardware design, verified the design, and used tools to generate the hardware post synthesis reports and layouts.

1.4. Introduction of the project testing environment

Since the main purpose of the equalizer is to reduce the intersymbol interference, but not to decode the entire message, our testing signal is a simplified version of the real communicating sequence. Our transmitted sequence contains 256 bits binary phase shift keying (BPSK) preamble and 1000 bits quadrature phase shift keying (QPSK) modulated random signal as Figure 2 indicates.

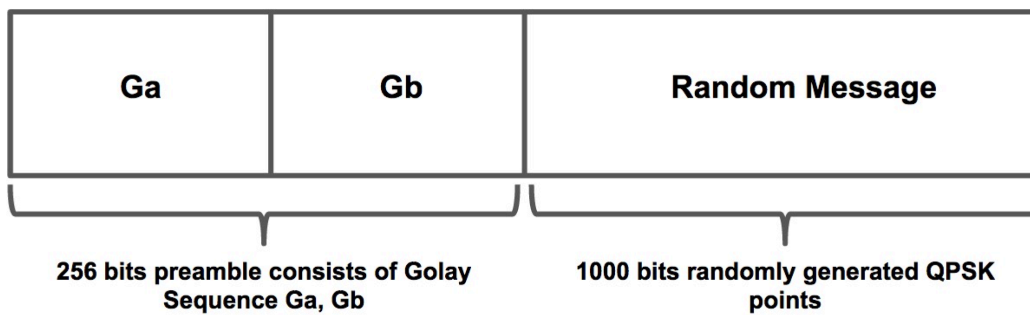


Figure 2. The randomly generated transmitted message with 256 bits fixed preamble. The reasons of using Ga, Gb sequences as preamble will be explained in the correlator section.

The DARPA challenge provides the channel model with four non-zero impulse responses within a time period of 512 samples. The radio system will oversample the received samples into symbols. Assume the system oversamples at a rate of four samples per symbol, the channel model will have a duration of 128 symbols. Equation 2 is a mathematical representation of the channel model, where δ represents the impulse response of the channel model, a_1 through a_4 represents the gain of each impulse response with the corresponding time delays τ_1 through τ_4. The transmitted message in Figure 2 will run through this channel model with some noise added to simulate the real world scenario. The entire test signal can be represented in Equation 3, where u[t] is the testing sequence presented in Figure 2 and n[t] is the AWGN noise.

$$h_c[t] = a_1 \bullet \delta[\tau_1] + a_2 \bullet \delta[\tau_2] + a_3 \bullet \delta[\tau_3] + a_4 \bullet \delta[\tau_4] \quad [2]$$

$$r[t] = a_1 \bullet u[t-\tau_1] + a_2 \bullet u[t-\tau_2] + a_3 \bullet u[t-\tau_3] + a_4 \bullet u[t-\tau_4] + n[t] \qquad [3]$$

In the DARPA challenge team's receiver design, there will be a correlator in front of the equalizer. A correlator uses the known preamble to detect the beginning of a transmitted message and provide an estimation of $h_c[t]$ in equation 2. Due to the noise and crosstalk, a correlator might not provide exact coefficients a_i in equation 2. However, calculating the τ_i should be fairly accurate.

2. Equalizer Design in Software and Verification

With the guidance of the advisors, the team picked a decision feedback equalizer (DFE) for the hardware implementation. Prior to starting coding in hardware, I first modified the algorithm to better fit into the test environment and the hardware specifications. Then I verified the algorithm in the software. The following sections will present my work in the algorithm and MATLAB parts.

2.1. Reasons to pick DFE

There are two different kinds of equalizers: linear and non-linear. The linear equalizer includes zero-forcing and minimal mean square equalizer. The non-linear equalizer includes DFE, maximum likelihood sequence equalizer, and maximum likelihood symbol detector (Aziz, 2007). Based on my teammates Zhuangyi's software comparison, the team picked Decision Feedback Equalizer (DFE) for hardware implementation. Detailed explanation of this choice can be found in the conclusion section of Zhuangyi's paper.

2.2. Introduction of the DFE algorithm

The DFE algorithm contains a feedforward filter, a feedback filter, and a decision device. Both filters' coefficients are updated with the least mean square (LMS) machine learning algorithm (Proakis, 2000). Figure 3 provides a block diagram for the DFE structure.

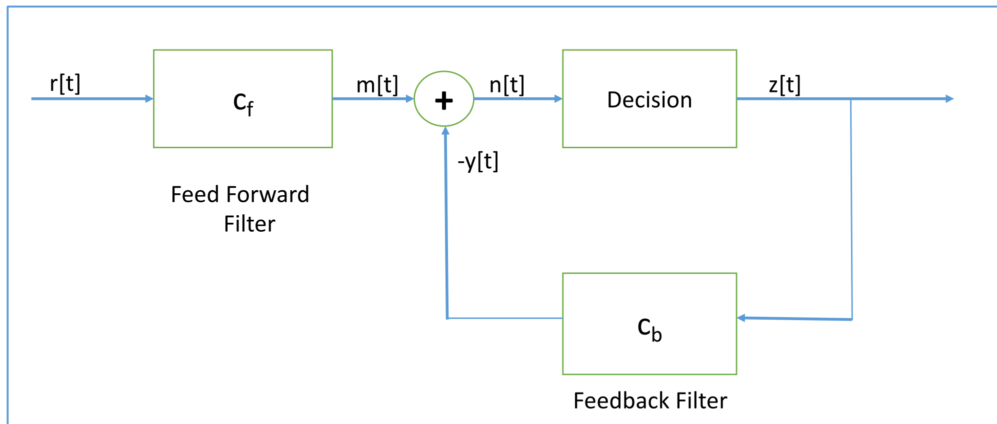


Figure 3. DFE block diagram (Aziz, 2007).

As shown in Figure 3, the signal first goes through the feedforward filter, then is subtracted by the output of the feedback filter, and finally it goes through a decision block.

The feedforward filter serves as the inverse of the channel model. The feedback filter will filter out the echo of the signal. The decision device is then used to correct the signal points in the constellation to the desired points and output the correct bits. For instance, for a QPSK modulated signal, the output of the decision device should be one of the four points in the QPSK constellation (Proakis, 2000).

Both the feedforward and the feedback filters' coefficients can be updated by the LMS algorithm. The LMS algorithm measures the difference between the input and the output of the decision device, called the error term and represented as equation 4. The loss function J is defined as

$$e[t] = z[t] - n[t] \qquad [4]$$

$J = E| e[t]^2 | - E| e[t] \bullet e^*[t] |$ [5]

where e*[t] is the complex conjugate of e[t]. E|f[t]| represents the expected value of f[t]. The feedback filter coefficients are updated by taking the derivative of the loss function over the previous filter coefficients. In the similar way we can take the derivative of the loss function over the previous feedforward filter coefficients to update the feedforward filter. The paper of Sajjad et al. provides a detailed math prove and the results are listed here:

$$\frac{\partial J}{\partial c_f[t]} = -y[t] \bullet e^*[t]$$

$$\frac{\partial J}{\partial c_b[t]} = z[t] \bullet e^*[t]$$

Referring to Figure 3, c_f represents a vector of the feedforward filter's coefficients and c_b represents the vector of the feedback filter's coefficients. y[t] is the input of the feedforward filter and z[t] is the input of the feedback filter, and μ is the step size defined by the user (Ghauri, Adee, Butt, & Arslan, 2013).

c_f (updated) = c_f (current) + μy[t] •e*[t]

c_b (updated) = c_b (current) - μz[t] •e*[t]

With these signal processing components, the equalizer is able to recover the signal from the ISI with a low bit error rate.

2.3. DFE design modification

The DFE with the LMS algorithm performs well even with an unknown channel model. However, using the original algorithm to recover 128 taps from 0 takes a large amount of time. To avoid the long equalizing time, I tried to make better estimations of the initial values of the coefficients.

The DARPA challenge's complete design includes a correlator which will detect the beginning of the signal and provide an estimation of the channel model, as mentioned in Section 1.4. The estimated channel is represented as Equation 6.

$$h_c'[t] = a_1' \bullet \delta[t-\tau_1] + a_2' \bullet \delta[t-\tau_2] + a_3' \bullet \delta[t-\tau_3] + a_4' \bullet \delta[t-\tau_4] \quad [6]$$

$$c_b[t] = a_2' \bullet \delta[t-\tau_2] + a_3' \bullet \delta[t-\tau_3] + a_4' \bullet \delta[t-\tau_4] \quad [7]$$

$$y[t] = a_1 \bullet h[t-\tau_1] + (a_2-a_2') \bullet h[t-\tau_2] + (a_3-a_3') \bullet h[t-\tau_3] + (a_4-a_4') \bullet h[t-\tau_4] + n[t] \quad [8]$$

In the design, the second to the last correlator's output serves as an initial value to the feedback filter coefficients ($c_b[t]$ in equation 7). Equation 8 indicates the receiving signal subtracted by the output of the feedback filter to reduce the ISI from timestamps τ_2, τ_3 and τ_4. Then the LMS algorithm attempts to correct $[a_2', a_3', a_4']$ to the original coefficient $[a_2, a_3, a_4]$. In this way, it will reduce the number of computational cycles during the LMS algorithm and thus save power for the entire design. Since the correlator will simplify the design, it is my intention to implement the correlator in the hardware as well.

In conclusion, the hardware design is optimized to implementing a DFE with filter coefficients initialized to the outputs of the correlator.

2.4. Implementing a Golay correlator

A correlator utilizes the known preamble to detect the beginning of a receiving signal. An ordinary correlator will take the received signal and multiply the complex conjugate of the preamble. This value compared against the expected value will give us the gain and the time delay.

The team uses the Golay sequences from the IEEE standard as the preamble of the transmitting signal as shown in Figure 2. Golay sequences consist of different combinations of

Golay A and Golay B sequences. The team uses 128 bits Golay A and 128 Golay B which can be found in appendix B. Both Golay A and Golay B sequences are BPSK modulated.

Choosing a reasonable preamble is crucial to the accurate computation of the gain and the time delay. The team chose Golay sequences because they minimize the false peaks and use less computations than the random sequences. Arithmetic design simplifications can be seen from Figure 4, which correlates 128 symbols with only 7 sets of computations instead of 128 sets (IEEE Standards, 2014).

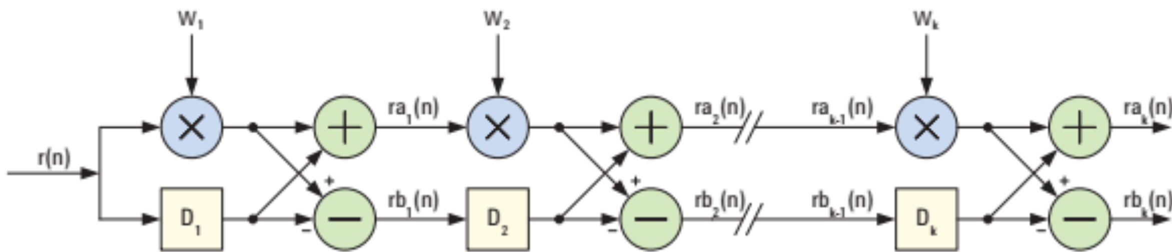


Figure 4. Arithmetic block diagram of the Golay correlator. For 128 bits Golay sequence, Dk = [1 8 2 4 16 32 64]. Wk = [-1 -1 -1 -1 +1 -1 -1] (Agilent Technologies, n.d.).

The minimizing of false peaks is due to the cancellations of Golay sequence A and Golay sequence B. In Figure 4, the input of the correlator is the received signal r(n). The output ra(n) and rb(n) are outputs which separately detecting Golay A and Golay B. Figure 5.a is the result from the correlator plotted in MATLAB when correlating Ga128, Gb128 followed by a random sequence. The blue line represents ra(n) and the red line represents rb(n). In order to cancel the false peaks, the algorithm delays ra(n) 128 bits and adds it onto rb(n), which is shown in Figure 5.b. Comparing Figure 5.a and Figure 5.b, the amplitude of false peaks in 5.b is much smaller than 5.a.

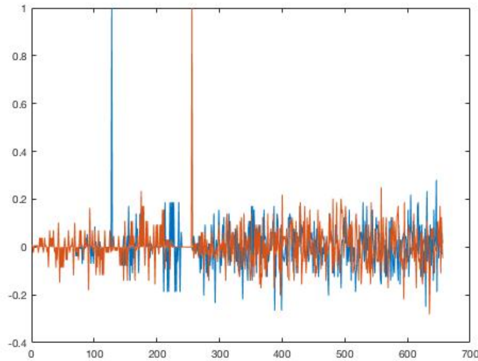


Figure 5.a. Correlator results of ra(n) and rb(n).

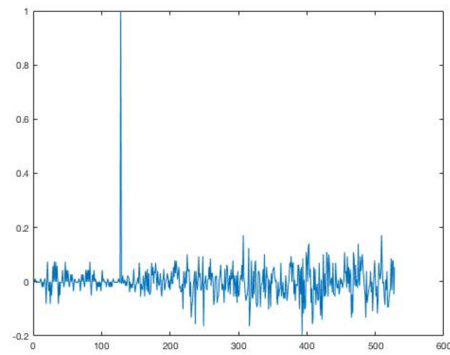


Figure 5.b. Correlator result of ra(n)+rb(n+128).

2.5. Software design verification and conclusion

I implemented the correlator and the DFE in MATLAB and verified them by testing the bit error rate in the specified testing environment. With the correct step size and iteration from the LMS, the algorithm can get zero bit error rate.

To verify the algorithm's resistance to noise, I plotted the bit error rate versus Eb/No curve as shown in Figure 6. Eb/No is the signal noise ratio per bit. Figure 6 test two different structures of the design. There are three lines in the plot. The red line represents the DFE with LMS algorithm. The blue line represents the DFE with a correlator to provide an estimation of the channel model. The yellow line represents the DFE with a correlator and using the LMS algorithm to adjust the coefficients. From Figure 6, Firstly I am able to confirm that with less noise and interference, there will be less bit error rate (Heegard & Wicker, 1999). Furthermore, the Figure indicates that the DFE with a correlator performs better than using only the DFE. Finally, since the difference between the yellow and the blue lines are fairly small, the correlator provides a decent estimation of the channel model.

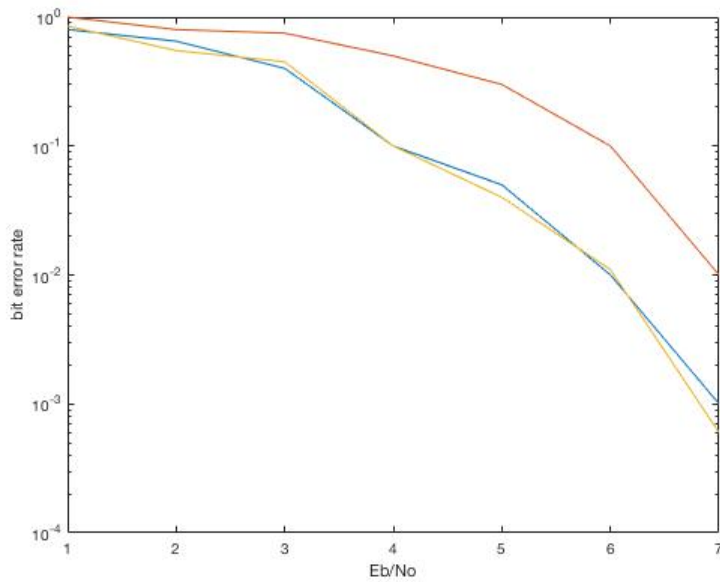


Figure 6. Bit error rate versus E_b/N_o.

Since the hardware symbols will be presented in a fixed point representation, which assigns certain bits to represent the integer and the fraction parts of a number. To test how many minimum fractional bits are needed to represent a symbol without sacrificing the bit-error-rate, I used a range of 1 to 15 bits to represent the fractional part of the symbol in the software. The testing signal has a signal noise ratio (SNR) of 10 and the result bit error rate is an average of running the code 100 times. The result is presented in Figure 7 and the MATLAB code is in appendix C section 1.1. From the MATLAB plot, we can conclude that we need at least 6 bits for the fractional part of the fixed point representation.

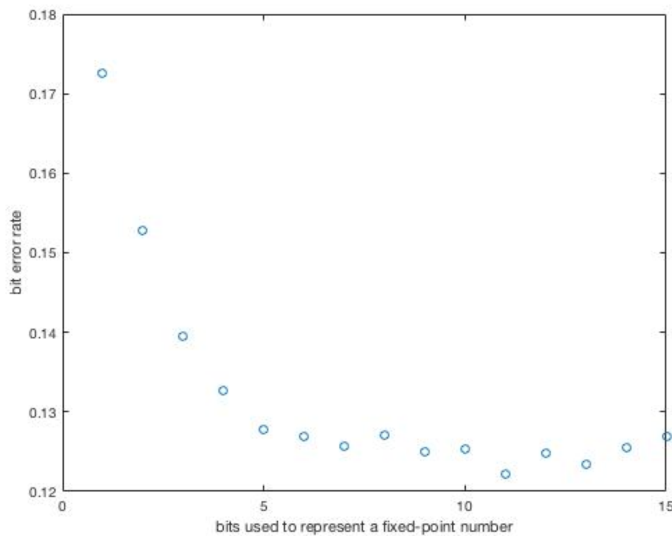


Figure 7. Plot of bits used to represent a fixed-point number versus average bit error rate.

3. Hardware Implementation

With the verified software model, the team was able to move forward and implement the design on the hardware. Berkeley developed a hardware tool called Constructing Hardware in a Scala Embedded Language (Chisel) which facilitates with hardware programming and synthesis. Chisel DSP is one of the Chisel libraries that enhances the functionality of Chisel in the digital signal processing(DSP) field. Chisel DSP contains tools that are frequently used in DSP algorithm such as the complex number operations (Wawrzynek, 2016). Without previous knowledge in Chisel and application-specific integrated circuit (ASIC) hardware design, I decided to take a very-large-scale integration (VLSI) class. Two classmates, Henry Zhu and Kate Du, were interested in the project and joined the DFE circuit level implementation.

3.1. Theory of operation

The basic DFE design is based on the MATLAB software as section 2 described in details. Figure 8 shows the general hardware design and detailed hardware code of each

components which can be found in appendix C section 2. Due to the time constrain, the feedforward filter has not been implemented in the hardware. The following sections will explain the hardware components and the design trade-offs.

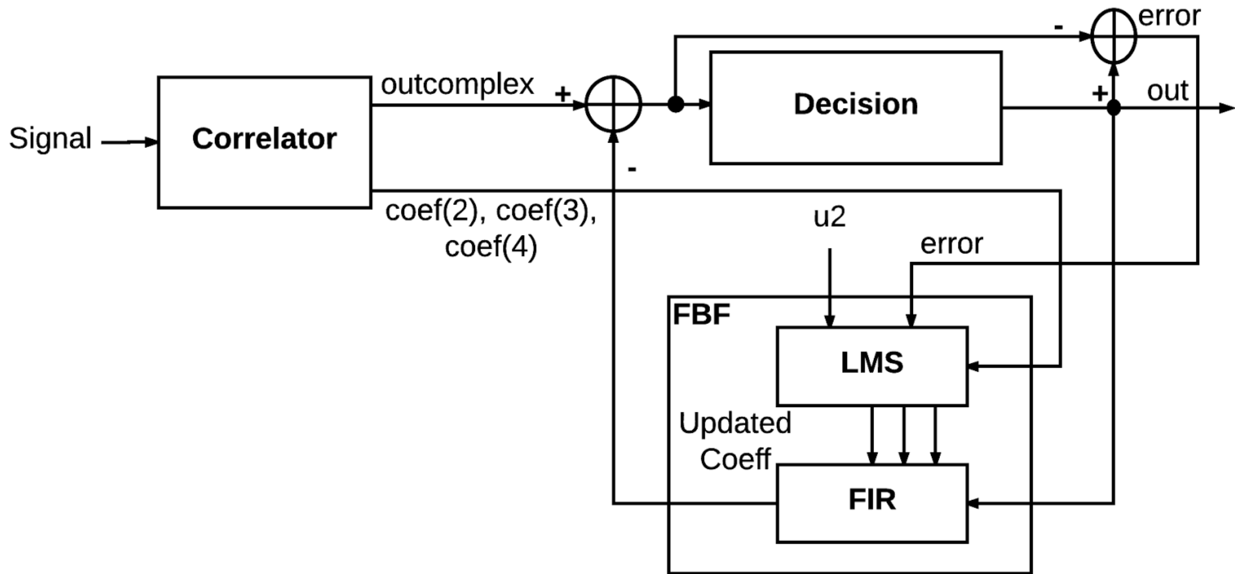


Figure 8. A block diagram of the DFE with a correlator.

3.1.1. Correlator

Section 2.4 described the correlator's algorithm and this section will explain the circuit level implementations in detail. Figure 9 shows the block diagram of the correlator designed by Kate Du. In this design, D1to D7 registers are used to calculate the peak detection of Golay B. The 128 registers after the ra(n) in Figure 9 are used to delay the output of Golay A. Then to synchronize the entire design, additional 256 registers are used to delay the original input signal so that the first detected peak from the correlator and the first sample in the preamble Golay A will come out in the same clock cycle. Since the maximum peak value is around 256, we use 9 bits to represent this peak value.

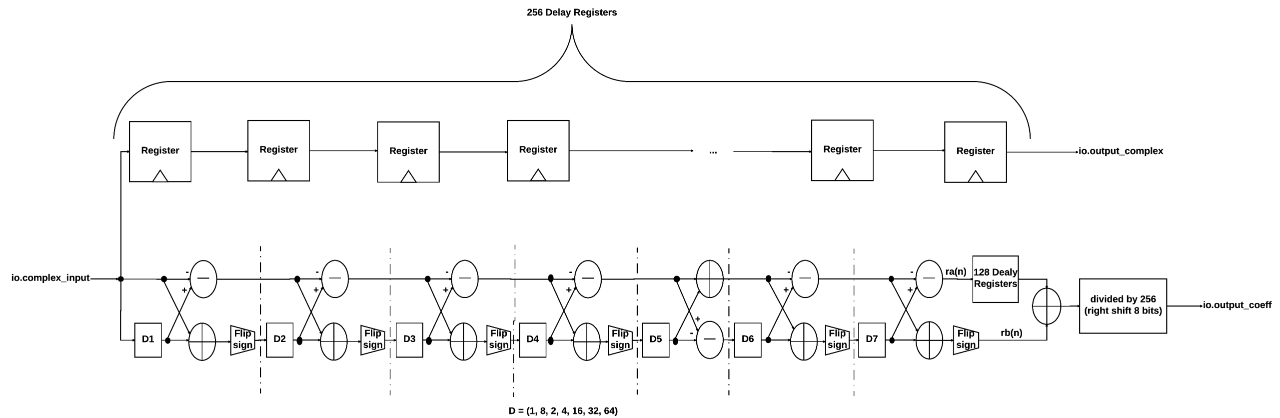


Figure 9. Block diagram design of the correlator. The 256 delay registers are used to delay input signals for 256 clock cycles at the output. The 128 delay registers after ra(n) are used to delay ra(n) for 128 clock cycles. D1 to D7 blocks represent groups of registers and delays. One register means one clock cycle delay. The number of registers in each block is shown in the set of D. D1 block has 1 register; D2 block has 8 registers; D3 block has 2 registers; D4 block has 4 registers; D5 block has 16 registers; D6 block has 32 registers; D7 block has 64 registers.

3.1.2. Decision Block

The decision block is a purely combinational logic circuit, which pushes the input data to the correct constellations. Since the signal contains two different kinds of modulation methods (BPSK for the preamble and QPSK for the message, refer to Figure 2), two different sets of mapping functions were designed. Choosing the correct mapping functions will be accomplished by the control unit in section 3.1.4.

3.1.3. Feedback Filter

The feedback filter is a finite impulse response (FIR) filter which only calculates the previous data of the signal (Barr, 2002). These previous data are stored in 128 registers. Figure 10 shows the block diagram of the feedback filter.

Section 2.3 explained the reason for three non-zero coefficient taps in the feedback filter. In the hardware design, three registers are used to store these three taps as well as the corresponded delays of the taps. The LMS algorithm is implemented directly in the feedback

filter block. When the LMS enable signal goes high, the algorithm will use the calculated error from the decision device to update the three coefficients.

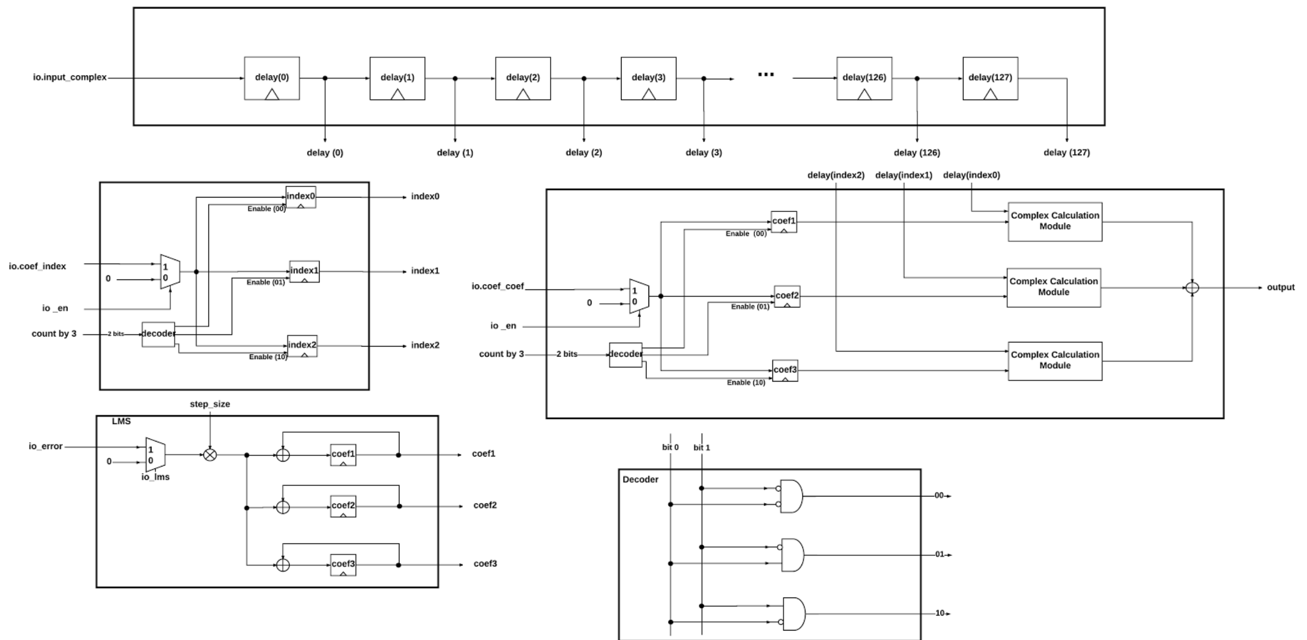


Figure 10. Block diagram of the feedback filter with 512 shift registers and the LMS algorithm.

3.1.4. Control Unit

A finite state machine (FSM) was designed to control different components of the hardware design. As shown in Figure 11, there are four different states in the FSM: s_idle, s_correlator, s_dfe_bpsk, and s_dfe_qpsk states. The s_idle state resets each component and initializes them back to default values. The s_correlator state is enabled by the enable signal and the correlator is turned on to find the preamble of the signal. The s_dfe_bpsk state is reached when the correlator finds the first peak. This s_dfe_bpsk state includes the correlator, the decision device, the feedback filter, and a counter to achieve the equalization functionality. The decision device will demodulate the signals based on the QPSK modulation scheme. The counter starts to count from 0 to 256. The s_dfe_qpsk state is activated when the counter reaches 256. At

this time, 256 samples have been passed to the decision device. The decision device will switch to QPSK modulation scheme. Finally, when counter reaches 1256, the entire message is processed and the state machine is reset back to s_idle stage. Any time during the s_correlator, s_dfe_bpsk, and s_dfe_qpsk states, the FSM can be reset back to the s_idle state via a reset signal.

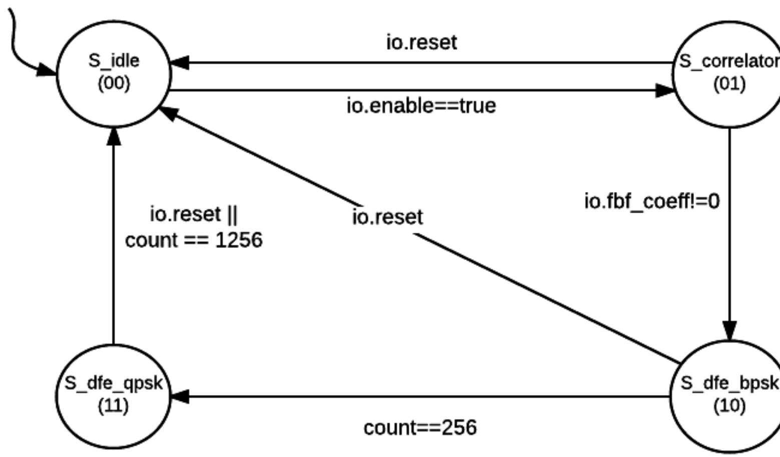


Figure 11. FSM control unit diagram for the DFE hardware.

3.2. Testing and verification

The hardware DFE design is implemented based on the Figure 8. As mentioned in section 3.1.1, since the correlator's peak calculation requires 9 integer bits, a total of 22 bits are required to represent a sample accurately (9 bits for the integer part, 12 bits for the fractional part, and 1 bit for the sign). This is the largest number appeared in the design. To ensure the accuracy of calculations, this longest fixed point bits is used throughout the design.

To verify the functionality of the design, the same software testing signals was passed to the hardware design module. Scala is a testing language for Chisel hardware design. The MATLAB code will generate the testing input signals and expected output results as text files. Then the hardware Scala testing code will read in these testing files and test through multiple

cycles. Finally, Scala will collect the outputs of the hardware design and compare with the expected values.

The entire design passed the Scala test. The longest hardware test ran more than 5000 clock cycles and decoded three different sequences of received messages. After the Chisel code passed the test, we pushed this original design to the post synthesis gate level simulation and used the IC Compiler (ICC) to performs place-and-route (PAR) operations. The clock frequency used for these simulations is 500MHz. Table 1 shows parts of the report file. The combinational area is the area consumption in μm^2 for the combinational logic, while the non-combinational area is for the sequential logic, such as the flip flops, registers, and SRAMs. As indicated in the Table 1, this original design requires large amount of area and power. The next section will talk about design optimizations and trade-offs.

Table 1. ICC report summary for the original design.

		Original Design
Area	Combinational Area (μm^2)	135110.32
	Noncombinational Area (μm^2)	158056.98
	Total Area (μm^2)	293167.31
Power	Switch Power (μW)	$1.95*10^3$
	Int Power (μW)	$7.94*10^4$
	Leak Power (pW)	$5.60*10^{10}$
	Total Power (μW)	$1.37*10^5$
Throughput	Clock Cycle(ns)	2.0
	Longest critical depth(ns)	1.9412

3.3. Design space exploration

In the hardware design, different optimization ideas and design trade-offs were created due to concerns such as area, time, and energy consumption. In the following sections, three main design optimizations and trade-offs are discussed: 1) fixed point representation 2) the

optimization of the feedback filter 3) the exploration of design trade-offs between the static random-access memory (SRAM) and the shift registers.

3.3.1. Fixed point representation

As mentioned in section 3.2, the design uses 22 bits to represent any symbols in the hardware. However, the software verification proves that with more than 6 fractional bits, different fixed point representations will not make a noticeable effect on the bit error rate. With this consideration, the team experimented to use 10 bits (3 bits for the integer part, 6 bits for the fractional part, and 1 bit for the sign) for most of the DFE design except the correlator's peak calculations. To declare different fixed point representations in Chisel DSP, we imported the `breeze.math.Complex` library and declared the shift registers as :

```
val output = Reg(Vec(512, DspComplex(FixedPoint(10.W, 6.BP),FixedPoint(10.W, 6.BP) )))
```

The ICC reports from Table 2 shows the area difference after the fixed point optimization.

Table 2. ICC report summary of the fixed point optimized design (10 bits for some shift registers in the correlator and the feedback filter) and comparison to the original design (20 bits for the entire design).

		Fixed point	Original	Improvements
Area	Combinational Area (μm^2)	96580.82	135110.32	28.52%
	Noncombinational Area (μm^2)	126797.02	158056.98	19.78%
	Total Area (μm^2)	223377.84	293167.31	23.81%
Power	Switch Power (μW)	$1.67*10^3$	$1.95*10^3$	14.36%
	Int Power (μW)	$6.4*10^4$	$7.94*10^4$	19.40%
	Leak Power (pW)	$3.3*10^{10}$	$5.60*10^{10}$	41.07%
	Total Power (μW)	$9.88*10^4$	$1.37*10^5$	27.88%
Throughput	Clock Cycle (ns)	2.0	2.0	
	Longest critical depth (ns)	1.9335	1.9412	0.40%

According to the Table 2, with the fixed point optimization, the design area is 23.8% smaller than the original design. Our design contains a total of 640 shift registers (128 registers for the feedback filter, 256 for the correlator's Ga, Gb calculation, and 256 for the correlator's output delays). Only 60% of these shift registers can be optimized. This fixed point optimization saves 50% of bits represented in each symbol, which is around 30% of the register area (calculation: 50% × 60%=30%). With less area used in the design, less power is used for the chip functionally as well. This report result meets the expectation.

3.3.2. Feedback filter optimization

Based on fixed point optimized design, the team moved forward to the feedback filter and explore more optimization method. The functionality of the feedback filter is to multiply three filter taps with the corresponding delayed signals from the 512 shift registers. There were two parts that could be optimized.

Firstly, MATLAB generates QPSK modulated signals [$\sqrt{2}+\sqrt{2}j, \sqrt{2}-\sqrt{2}$j, $-\sqrt{2}+\sqrt{2}$j, $-\sqrt{2}-\sqrt{2}$ j], which are under zero-phase shifting. To store these fractional numbers accurately, we need to use lots of bits. However, in the decision device, if these four points are represented as [1+j, 1-j, -1+j, -1-j], only two bits are needed to represent each number. Since the testing message contains both QPSK and BPSK modulated signals, we used three bits to represent these six numbers. The first three columns of Table 2 illustrates the method mapping these six complex numbers to the three bits symbol representations.

Secondly, multiplication is hardware intensive due to the amount of calculations, area consumption, energy consumption, and large delays in the combinational logic (Azarmehr,2008). These three bits symbol representations stored in the shift registers are one of the 6 options: [1, -1, 1+j, 1-j, -1+j, -1-j]. Consequently, multiplying these signals with the complex taps only

requires changing signs and additions. Table 3 summarizes the complex multiplication mapping method.

Figure 12 illustrates the block diagram of the optimized multiplication design. Using the simplified version of complex multiplication which was designed by Kate, we are able to save 18.25% of total area and 19.03% of the power. Table 4 shows the result of this feedback filter optimized version.

Table 3. Results of the six numbers multiplied by a+bj.

Modulation Scheme	Stored Signal	3 bits Representation	Multiplication Results
BPSK	1	000	a+bj
BPSK	-1	010	-a-bj
QPSK	1+j	100	(a-b) + (a+b)j
QPSK	1-j	101	(a+b) + (-a+b)j
QPSK	-1+j	110	-(a+b) + (a-b)j
QPSK	-1-j	111	(-a+b) - (a+b)j

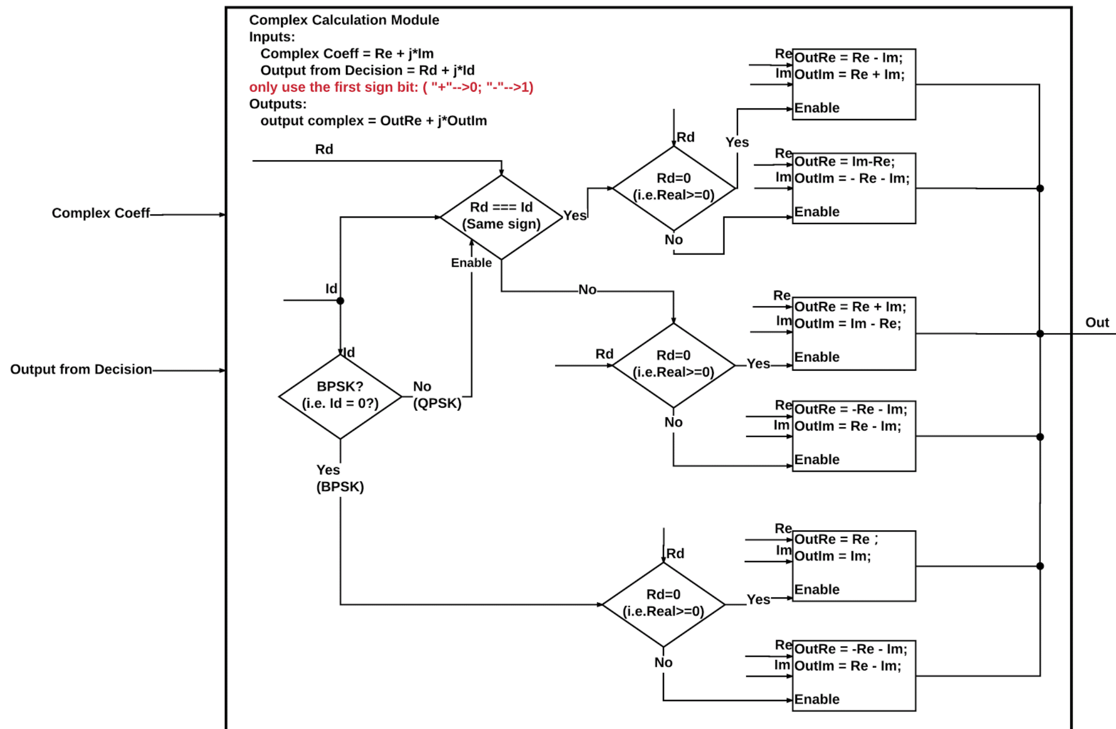


Figure 12. Design block diagram of using multiplexers to replace the complex multiplication operation.

Table 4. ICC reports summary of the feedback filter optimized design (using 3 bits and replaced complex multiplication with multiplexers in the feedback filter) and comparison to the fixed point optimized design (10 bits for part of the shift registers in the correlator and the feedback filter).

		3 bits for the feedback filter	Fixed point (10 bits)	Improvements
Area	Combinational Area (μm^2)	71215.98	96580.82	26.26%
	Noncombinational Area (μm^2)	111396.91	126797.02	12.14%
	Feedback Filter (μm^2)	21023.3	62314.5845	66.26%
	Correlator (μm^2)	160566.6596	159979.332	-0.36%
	Total Area (μm^2)	182612.89	223377.84	18.24%
Power	Switch Power (μW)	$1.67*10^3$	$1.67*10^3$	0%
	Int Power (μW)	$5.71*10^4$	$6.4*10^4$	10.78%
	Leak Power (pW)	$2.42*10^{10}$	$3.30*10^{10}$	26.66%
	Total Power (μW)	$8.3*10^4$	$9.88*10^4$	15.99%
Throughput	Clock Cycle (ns)	2	2	
	Longest critical depth (ns)	1.8812	1.9335	2.70%

Since the feedback filter optimization is implemented based on the fixed point optimization, Table 4 compares the reports between these two design. With the 128 10 bits registers optimized to 3 bits, we expected to save 70% of the feedback filter space. Replacing complex multiplication to multiplexers saves some design area and power as well. Thus results in Table 4 meet our expectation.

3.3.3. SRAM and register

Both the feedback filter and the correlator designs include a long shift register that storages the useful symbols. Besides shift registers, memories are also used to storage data (Preston, 2001). SRAM is one of the memory types that has high performance. Moving from registers to the SRAM, the speed and the cost per bit decreases, while the data storage capacity increases (Singh et al, 2012). Using SRAM, this large amount of storage has the potential to be

more efficient. Since the team could not find direct evidence to confirm the idea, we replaced the shift registers to the SRAMs. The SRAM section of the CS250 DFE team's final report discussed the implementation details. Table 5 is the ICC reports and the comparison. From Table 5, SRAM saves area and power of the entire design, however, the throughput increases. This is due to the read and write latency from the SRAM.

Table 5. ICC report summary of the SRAM version of the correlator design and comparison to the correlator without the SRAM.

		sram	w/o sram	Improvements
Area	Combinational Area (μm^2)	65898.26978	71215.98	7.47%
	Noncombinational Area (μm^2)	82621.70895	111396.91	25.83%
	sram area	5047.3002	NA	NA
	Total Area (μm^2)	148519.9787	182612.89	18.66%
Power	Switch Power (μW)	$9.05*10^3$	$1.67*10^3$	-441.91%
	Int Power (μW)	$4.13*10^4$	$5.71*10^4$	27.67%
	Leak Power (pW)	$3.02*10^{10}$	$2.42*10^{10}$	-2.479%
	Total Power (μW)	$8.050*10^3$	$8.30*10^4$	3.01%
Throughput	Clock Cycle (ns)	2.0	2.0	
	Longest throughput (GHz)	2.0092	1.8812	-6.81%

3.4. Chip layouts

With the above optimizations and recommendations, the design was passed through Chisel tools to generate chip layouts. Figure 13 shows a screenshot of the optimized design without SRAM while Figure 14 is the optimized design with a SRAM.

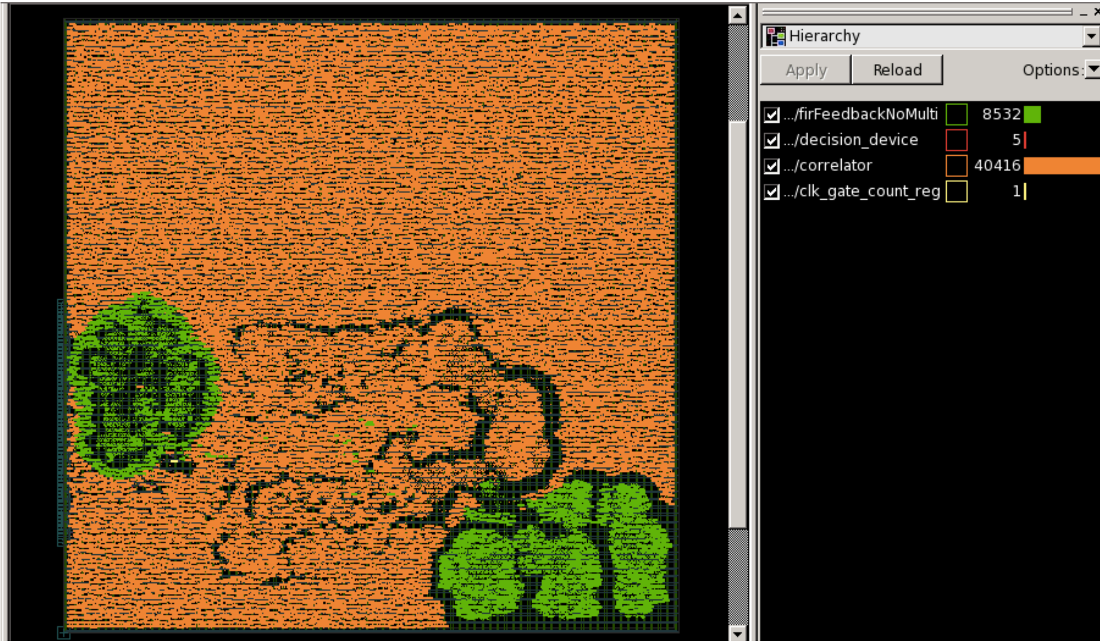


Figure 13. Chip layout with optimized fixed point representation and optimized feedback filter without SRAM. Green area is the feedback filter, orange area is for the correlator, red area (too small to see) is for the decision device, and the yellow area (too small to see) is for the control unit.

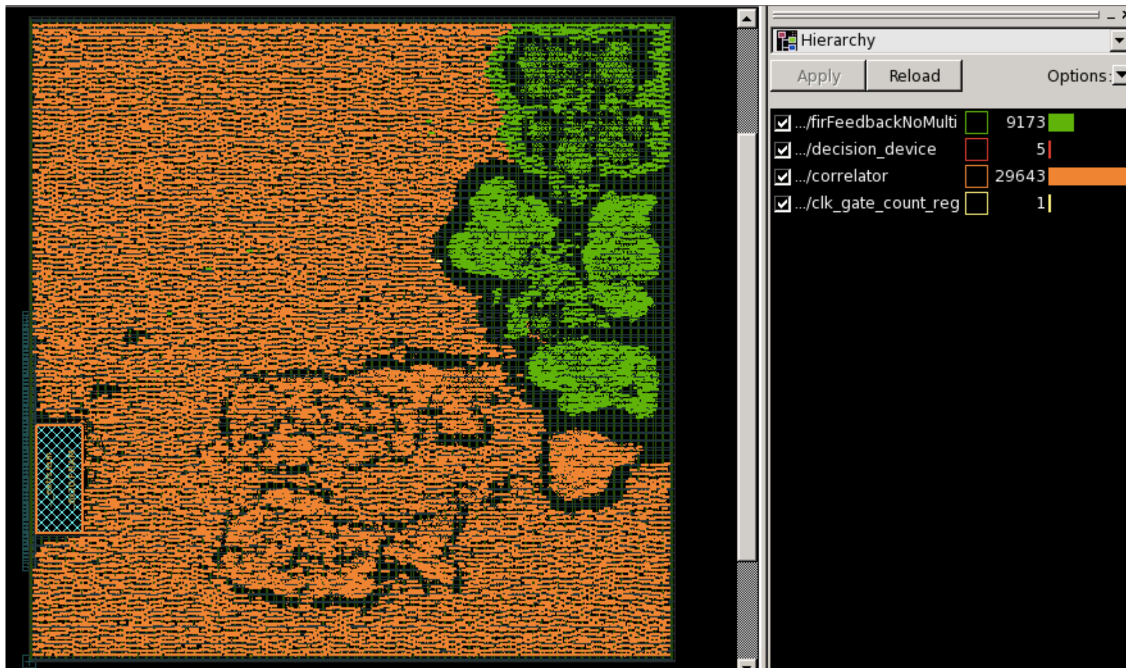


Figure 14. Chip layout with optimized fixed point representation and optimized feedback filter without SRAM. Green area is the feedback filter, orange area is for the correlator, red area (too small to see) is for the decision device, and the yellow area (too small to see) is for the control unit. The bottom left rectangular area is the SRAM.

4. Conclusion and Future Work

4.1. Project summary

In summary, the primary objective of my work on this capstone project was to design a hardware equalizer and implement it in ASIC technology. After implementing software models of equalizers using MATLAB, and evaluating the alternatives, the team decided to use a decision feedback equalizer (DFE). The entire equalizer comprises a decision block, an adaptive LMS-based feedback filter, and a preamble correlator. The design was implemented and tested using the Chisel HDL, and synthesized with Synopsis Design Compiler, and the final layout was generated using Synopsis IC Compiler. The design was evaluated for its use of chip area and its power consumption. Several optimizations were performed to reduce chip area and power. First, it was determined that the entire equalizer design could operate with only 6 fractional bits and 4 integer bits, with no loss in accuracy over a larger number representation. In the case of the feedback filter, which processes the output of the decision block and therefore operates on 2-bit values, its area and power was further reduced by eliminating the need for multiplications. Also, the correlator design was optimized by replacing the flip-flops used to implement the 256 stage shift register with SRAM blocks. While, this optimization resulted in reduced area and total power, throughput was decreased.

4.2. Project reflection

Through this implementation progress, I gained a deep understanding about filters, equalizers, channel models, and gradient descent algorithm. For software programming, using MATLAB to process streaming data was a new experience. The streaming structure maintains a useful section of the message in the time frame in a first-in-first-out buffer, or a programmer can index certain sections of the message according to the time frames. Hardware programming is a

brand new field to me. Taking CS250 VLSI class helped me get familiar with the hardware design and programming tools.

From this project, there are a few improvements that I could have made. Firstly, for hardware Chisel programming, I should draw functioning block diagrams before implementing the hardware code. For this project, I only drew brief design blocks but not detailed components such as registers and multiplexers. This created some communication issues. Other teammates and professors could not easily understand my block diagrams and had to search into my code to follow my design ideas. Secondly, I should push a working version through the Chisel tools as soon as possible. We had a working version of the design a month before the deadline, but we put our focus on optimizing it instead of generating post synthesis gate level reports, par operations, and chip layouts. Since none of the teammates were familiar with the Chisel tools, after we optimized our design, figuring out how to use the tools to get desired results in time was a challenge for the team.

4.3. Future work

The simulation results of the hardware design show that there are still some design spaces to be explored. Firstly, the feedforward filter is not implemented in the DFE hardware design. The team implemented the feedforward filter's Chisel code but did not have time to integrate it in the datapath and the control unit. It would be interesting to see the feedforward filter being implemented in the hardware and compare the performance with our current results. Secondly, from the final optimized versions of the DFE, as Figure 13 and Figure 14 indicate, the correlator is taking too large of the entire design area. The bit-width in the correlator can be further optimized, especially the long shift register used during the calculation of the correlator's peaks.

Finally, section 3.3.2 implemented a simplified version of the feedback filter by having three bits to represent six complex numbers. Discussed this idea with the CS 250 professor and the Graduate Student Instructor, the results could be simplified with only two bits and a control signal indicating which constellation scheme the signal belonged to. The detailed implementation plan can be found in the future work section of the CS250 final report.

Reference 1

Accessing Spectrum. (n.d.). Retrieved October 15, 2016, from https://www.fcc.gov/general/accessing-spectrum

Agilent Technologies: Wireless LAN at 60 GHz - IEEE 802.11ad Explained. (n.d.). Retrieved April 13, 2017, from http://cp.literature.agilent.com/litweb/pdf/5990-9697EN.pdf

Azarmehr,M.(2008) [PDF document]. Retrieved from Multipliers, Algorithms, and Hardware Designs, http://www.vlsi.uwindsor.ca/presentations/2008/8-Multipliers.pdf

Aziz, A. (2007 July). Decision Feedback Equalizer for StarCore –Based DSPs.*Freescale Semiconductor, AN2072(Rev. 2)*. Retrieved from http://www.nxp.com/assets/documents/data/en/application-notes/AN2072.pdf

Defense Advanced Research Projects Agency. (n.d.). Retrieved March 11, 2017, from https://spectrumcollaborationchallenge.com/about/

Ghauri, S. A., Adee, H., Butt, M. S., & Arslan, M. (2013 August). Adaptive Decision Feedback Equalizer(ADFE). *International Journal of Computer and Electronics Research, Volume 2, Issue 4.* Retrived from http://ijcer.org/index.php/ojs/article/viewFile/177/162

Heegard, C., & Wicker, S.B. (1999). *Turbo Coding.*Kluwer Boston/Dordrecht/LondonAcademic Publishers.

IEEE Standards. (2014, March) 8802-11:2012/Amd.3:-2014 - ISO/IEC/IEEE International Standard for Information technology--Telecommunications and information exchange between systems--Local and metropolitan area networks--Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band (adoption of IEEE Std 802.11ad-2012).*IEEE Explore, Page 490.* DOI: 10.1109/IEEESTD.2014.6774849

Mathuranathan. (2014, July). Retrieved March 11, 2017, from http://www.gaussianwaves.com/2014/07/statistical-characteristics-of-multipath-channels-scattering-function/

Barr, M. "Introduction to Finite Impulse Response Filters for DSP." *BARR Group,* 01 Dec. 2002, https://barrgroup.com/Embedded-Systems/How-To/Digital-Filters-FIR-IIR. Accessed 13 Apr. 2017.

Proakis, J.(2000) *Digital Communications, 4th ed.*, McGraw-Hill.

Railroad Frequencies: Original AAR Plan. (n.d.). Retrieved October 15, 2016, from http://www.dpdproductions.com/page_rrfreqs.html

Singh, J., Mohanty, S.P.,Pradhan, D.K. *Introduction to SRAM*. Springer New York, 2006, pp. 4.

United States Frequency Allocations: The Radio Spectrum [Chart]. (2003, October). Retrieved October 16, 2016, from https://www.ntia.doc.gov/files/ntia/publications/2003-allochrt.pdf

Wang, T. *Theory of Digital Communications* [power point slide]. Retrieved from Lecture Notes Online Web site: http://wireless.ece.ufl.edu/twong/Notes/Comm/ch4.pdf

Wawrzynek, J.(2016) [PDF document]. Retrieved from CS250: VLSI System Design https://inst.eecs.berkeley.edu/~cs250/sp17/lectures/lec02-chisel-sp17.pdf

Chapter 2. Engineering Leadership Paper

The purpose of this engineering leadership paper is to apply our technical capstone project into the industry world. This chapter was written by all the M.Eng team members. The team chose to focus on three different aspects: current market analysis, team project management experience, and ethics concerns. These three aspects help shaping the project into more specific design and demonstrate the team's ability to situate the project in a broader context.

1. Market Analysis

Wireless Communication is the industry with big market that now experiencing rapid growth. Wireless telecommunication industry provides people with cellular mobile phone service (ex. Talk, text message), wireless internet access and wireless video services through radio-based cellular networks. The total revenue of the industry is expected to grow to $255.9 billion in the 2016 with 1.5% growth compared to last year. The average annual growth rate is expected to reach 3.2% from 2016 to 2022 (IBIS,2016).

The target clients of our solution are company in telecommunication industry and the government. Our solution can help the company in the industry run down their operation cost. The government can utilize the signal bandwidth more efficiently by adopting our solution. The stakeholders of our solution are the customers of telecommunication industry, our solution can provide them faster access to the wireless connection.

Our solution will provide government and companies a robust, reliable and efficient spectrum management system. The software defined radio(SDR) is employed to analyze and optimize the spectrum sharing method. The system will analyze the condition of spectrum usage and allocate the redundant bandwidth resource to the channel that is overloaded or need extra bandwidth. The dynamic allocation will make spectrum resource utilization more efficient and

balance the workload of each channel. Furthermore, the function of each channel is now more versatile and no longer confined to certain use. This enable the government to distribute and utilize spectrum more efficiently. This will also greatly reduce the company's expenditure on spectrum management. This turn out to speed up the transmission of data and information which will enhance the user experience of customers.

Our solution includes both product and service. The service includes maintenance and technical support. We will attract our customer by setting up a booth on the technical shows and trade conferences of telecommunication industry. This allows us to meet with technicians in the companies and government and introduce our product to them. Since the need and consideration tend to be different from customer to customer. The product will be distributed directly, so we can reach out our customer to better understand their need. The product will be customized to better address the need of the customer and to compact with the system and infrastructure current deployed by the customers. The customer will be charged on product and maintenance and technical support is available for free. We will adopt vertical strategy for the development of the business, which mean we will start from telecommunication industry and then expand it to more and more industries.

2. Project Management

Apart from the technique part, we used the project management knowledge we learnt from class to coordinate with each other and promote our project.

The primary challenge of project management is to achieve all of the project goals within the given constraints. The primary constraints for us is time and quality since we have coursework, projects and exams apart from this project.

At first, we use the traditional phased approach, which identifies a sequence of steps to be completed. In the traditional approach, five developmental components of a project can be distinguished

1. Initiation
2. Planning and design
3. Execution and construction
4. monitoring and controlling systems
5. completion and finish point

However, the biggest challenge for us under this project management method is that the work scope may change. Due to our schedule and the progress of the whole semester, it's not uncommon that we have to postpone our project some time. Then we changed to another project management method-PRINCE2.

PRINCE2 provides a method for managing projects within a clearly defined framework, it focuses on the definition and delivery of products, in particular their quality requirements. As such, it defines a successful project as being output-oriented (not activity- or task-oriented) through creating an agreed set of aims that define the scope of the project and provides the basis for planning and control.

So we basically divided our task to several pieces and then determine our separate task for the week and each person is responsible of his/her own part. And the last thing we do each week is try to combine our parts together. In this way, we boosted our efficiency a lot.

3. Ethics

Universal communication protocols with shared bandwidth also rises ethics considerations. Two main ethical concerns are related to information theft and untruthful demand of usage.

The first concern, information theft, reflects the section 1 from *the Code of Ethics for Engineers* about the private information [National society of professional engineers, 2007]. This communication system is not supposed to expose any private information or facilitate other malicious parties to obtain user's information. From this project, in order to achieve a shared bandwidth, public standard information such as the universal receipt address and sender address is required. However, the system should not acquire any personal information or data under the communication service. To solve this ethics concern, both the users and the service platform have the responsibility to protect the private information. The users of this communication protocol are recommended to encrypt their message in a more sophisticated method than the public communication addresses. A two-way authentication should be established between the receiver and the sender. Users should also be aware and report to the organization immediately if personal information is acquired during the communication process [Giampaolo et al., 2013]. From the system provider's side, they could either enhance their security tools or adopt a third party to audit the process. In Shah's paper, a three steps initialization, audit, and extraction method could be used to monitor the privacy process [Shah et al., 2008].

Secondly, since this communication method is based on shared bandwidth and demand, some users might request more bandwidth or usage time than they actually needed. This action could cause other users to delay their traffics and not achieve their goal in time. The capstone team comes up with a charged system to solve this concern. This system will charge each user by

the amount of usage they claim. This method could potentially decrease the amount of untruthful demands of usage and also encourage users to come up with optimal way to transmit the message with minimal bandwidth. To follow the ethic code section 5, the charge system should be fair to every users and not be influenced by any interests [National society of professional engineers, 2007].

In this paper, we talked about how to apply our technical capstone project into the industry world in three different categories, the current market analysis, team project management experience, and ethics concerns, which showed our ability as a master of engineering student to apply our skill into industry.

Reference 2

Blau, G. (2016, October 14). IBISWorld Industry Report Wireless Telecommunications Carrier in US. Retrieved October 16, 2016, from http://clients1.ibisworld.com/reports/us/industry/default.aspx?entid=1267

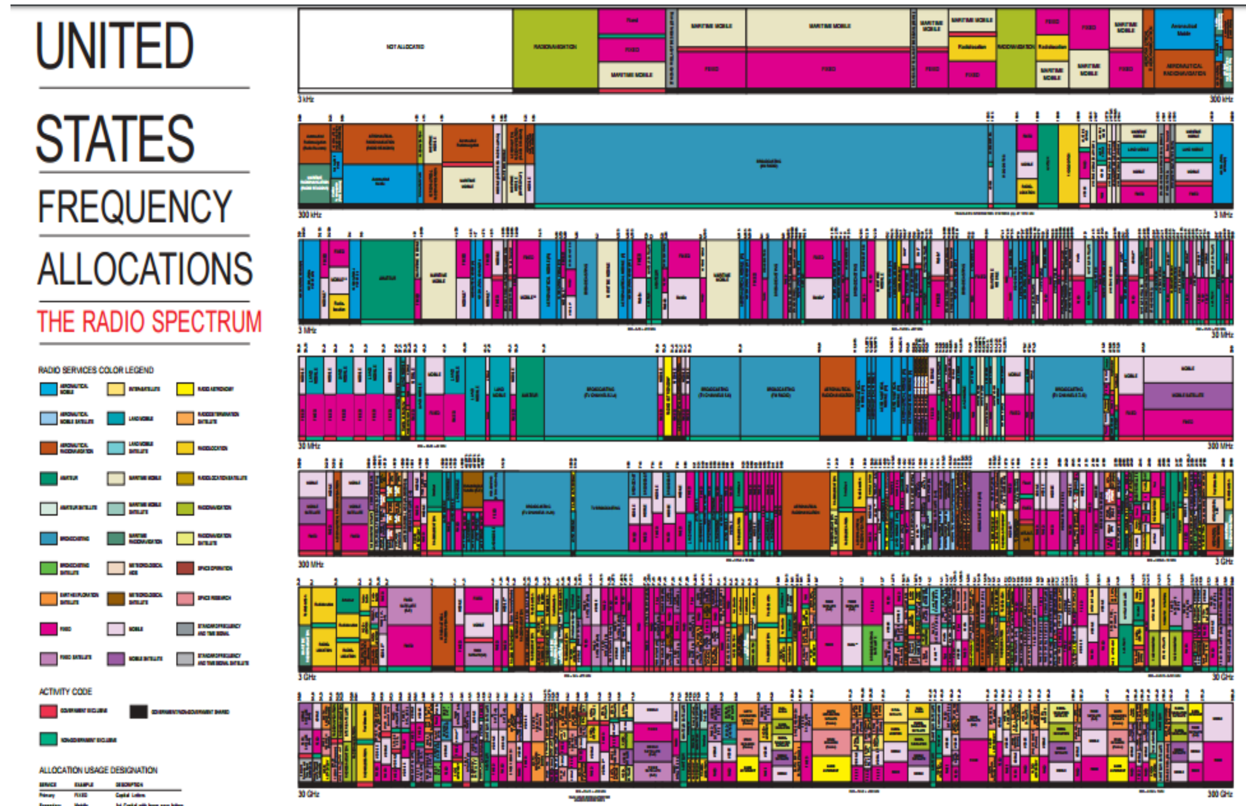
National society of professional engineers. (2007). *Code of Ethics for Engineers.* Retrieved from https://www.nspe.org/sites/default/files/resources/pdfs/Ethics/EthicsReferenceGuide.pdf

Giampaolo, B., Rosario, G., Gabriele, L., & Curzon., P. (2014), *A Socio-technical formal analysis of TLS certificate validation in modern browsers*. Retrieved February 5, 2017 from https://pdfs.semanticscholar.org/2eb2/e2ddf7c7dd2ba2887279800385115b60f45e.pdf

Shah, M., Swaminathan, R., & Baker, M. (2008, April 30). *HP Laboratories.Privacy-Preserving Audit and Extraction of Digital Contents* Retrieved from http://shiftleft.com/mirrors/www.hpl.hp.com/techreports/2008/HPL-2008-32R1.pdf

Sherman, J. (2013, March 25). YOUR WIRELESS CARRIER IS GOUGING YOU AND WE HAVE THE NUMBERS TO PROVE IT. Retrieved October 16, 2016, from http://www.digitaltrends.com/mobile/your-wireless-carrier-is-gouging-you-and-we-have-the-numbers-to-prove-it/

Appendix A: [United States Frequency Allocations, 2013]



Appendix B: Golay 128 Sequence

The Sequence $Ga_{128}(n)$, to be transmitted from left to right, up to down
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 +1 +1 -1 -1 +1 +1 +1 +1 -1 +1 -1 +1 -1 +1 +1 -1
-1 -1 +1 +1 +1 +1 +1 +1 +1 -1 +1 -1 -1 +1 +1 -1 +1 +1 -1 -1 +1 +1 +1 +1 -1 +1 -1 +1 -1 +1 +1 -1
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 +1 +1 -1 -1 +1 +1 +1 +1 -1 +1 -1 +1 -1 +1 +1 -1
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 -1 -1 +1 +1 -1 -1 -1 -1 +1 -1 +1 -1 +1 -1 -1 +1

The Sequence $Gb_{128}(n)$, to be transmitted from left to right, up to down
-1 -1 +1 +1 +1 +1 +1 +1 +1 -1 +1 -1 -1 +1 +1 -1 -1 -1 +1 +1 -1 -1 -1 -1 +1 -1 +1 -1 +1 -1 -1 +1
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 -1 -1 +1 +1 -1 -1 -1 -1 +1 -1 +1 -1 +1 -1 -1 +1
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 +1 +1 -1 -1 +1 +1 +1 +1 -1 +1 -1 +1 -1 +1 +1 -1
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 -1 -1 +1 +1 -1 -1 -1 -1 +1 -1 +1 -1 +1 -1 -1 +1

Appendix C: Software and Hardware Code

Section1. Software MATLAB code

1.1. Fixed point bits vs bit error rate

```
k = zeros([1,15]);
for i = 1:100
    counter = 1;
    for n= linspace(1,15,15)
[y,msg, preamble] = testBench(n);
input = y;
fbf_coef = correlator(input, preamble);
delta = 2^(-10);
iteration = 1;
[output,updated_fbf,ber] = olddfe_lms(msg, fbf_coef, delta, input, iteration);
k(n) = ber+k(n);
    end
    n = linspace(1,15,15);
    plot(n,k/100,'x')
end

function [y,modmsg, preamble] = testBench(n)
snr = 10;
preamble_bitb = [3 0 0 0 1 1 2 1 3 0 3 3 1 1 1 2 0 3 3 3 2 2 1 2 2 0 3 3 1 1 1 2 3 0 0 
0 1 1 2 1 3 0 3 3 1 1 1 2 3 0 0 0 1 1 2 1 0 3 0 0 2 2 2 1];
preamble_bita = [0 3 3 3 2 2 1 2 0 3 0 0 2 2 2 1 3 0 0 0 1 1 2 1 0 3 0 0 2 2 2 1 3 0 0 
0 1 1 2 1 3 0 3 3 1 1 1 2 3 0 0 0 1 1 2 1 0 3 0 0 2 2 2 1];
M = 4; % Alphabet size for modulation
len = 4096+128;
msg = randi([0 M-1],len,1); % Random message
msg = cat(1,preamble_bitb',preamble_bita', msg);
hMod = comm.QPSKModulator();
modmsg = step(hMod,msg);  % QPSK modulated signal
preamble = modmsg(1:128);
fs = 20e6;
fd = 0;
chan = stdchan(1/fs, fd, 'cost207RAx4');
y = filter(chan,modmsg);
y = fi(awgn(y,snr),1,4,n);
end

function [ output_c, output_b] = olddecision_device(input)
if real(input)>= 0
    if imag(input)>=0
        output_c = sqrt(2)/2 + sqrt(2)/2i;
        output_b = 0;
    else
        output_c = sqrt(2)/2 - sqrt(2)/2i;
        output_b = 2;
    end
else
    if imag(input)>=0
        output_c = -sqrt(2)/2 + sqrt(2)/2i;
        output_b = 1;
    else
        output_c = -sqrt(2)/2 - sqrt(2)/2i;
        output_b = 3;
    end
end
```

```
function [ output,fbf_coef, ber ] = olddfe_lms(modmsg, fbf_coef, delta, input ,
iteration)
len = length(modmsg);
coef_len = length(fbf_coef);
output = zeros([1,len]);
output_b = zeros([1,len]);
output_arr = zeros([1, coef_len]);
coef_needed = [fbf_coef(6);fbf_coef(28);fbf_coef(42)];
for i= 1:len %6:len
    if i<=coef_len
        temp = input(i) - fliplr(output(1:i))*fbf_coef(1:i)';
        [output(i),output_b(i)] = olddecision_device(temp);
    else
        temp = input(i) - fliplr(output(i-coef_len+1:i))*fbf_coef';
        [output(i),output_b(i)] = olddecision_device(temp);
    end
    error = output(i) - temp;
    if(i<iteration)
        output_arr (1,2:coef_len) = output_arr (1,1:coef_len-1);
        output_arr (1) = output(i);
        output_needed = [output_arr(6);output_arr(28); output_arr(42)];
        b_update = output_needed * conj(error);
        coef_needed = coef_needed - delta*b_update;
        fbf_coef(1) = 0;
        fbf_coef(6) = coef_needed(1);
        fbf_coef(28) = coef_needed(2);
        fbf_coef(42) = coef_needed(3);
    end
end
ber = bit_error_rate(modmsg, output_b);
end

function [fbf_coef] = correlator(input, preamble)
pre_len = length(preamble);
fbf_coef = zeros([1,512]);
for i = 1:512
    tmp = reshape(input(i:i+pre_len-1), [1,pre_len]) * conj(preamble) / pre_len;
    if tmp>0.3
        fbf_coef(i) = tmp;
        disp(tmp)
    end
end
end
```

Section 1.2 testing signal

```
close all;
ga128 = [+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 +1 +1 -1 -1 +1 +1 +1 +1 -1 +1
-1 +1 -1 +1 +1 -1 ...
-1 -1 +1 +1 +1 +1 +1 +1 +1 -1 +1 -1 -1 +1 +1 -1 +1 +1 -1 -1 +1 +1 +1 +1 -1 +1 -1 +1 -1
+1 +1 -1 ...
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 +1 +1 -1 -1 +1 +1 +1 +1 -1 +1 -1 +1 -1
+1 +1 -1 ...
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 -1 -1 +1 +1 -1 -1 -1 -1 +1 -1 +1 -1 +1
-1 -1 +1];
gb128 = [-1 -1 +1 +1 +1 +1 +1 +1 +1 -1 +1 -1 -1 +1 +1 -1 -1 -1 +1 +1 -1 -1 -1 -1 +1 -1
+1 -1 +1 -1 -1 +1 ...
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 -1 -1 +1 +1 -1 -1 -1 -1 +1 -1 +1 -1 +1
-1 -1 +1 ...
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 +1 +1 -1 -1 +1 +1 +1 +1 -1 +1 -1 +1 -1
+1 +1 -1 ...
+1 +1 -1 -1 -1 -1 -1 -1 -1 +1 -1 +1 +1 -1 -1 +1 -1 -1 +1 +1 -1 -1 -1 -1 +1 -1 +1 -1 +1
-1 -1 +1];
```

```
ts = 20e6;
fd = 0;
M = 4; % Alphabet size for modulation
len = 1000;
msg = randi([0 M-1],len,1); % Random message
hMod = comm.QPSKModulator();
modmsg = step(hMod,msg);  % QPSK modulated signal
message =
cat(1,reshape(ga128,128,1),reshape(gb128,128,1),reshape(ga128,128,1),reshape(gb128,128
,1),modmsg);
rng('shuffle');
chan = stdchan(ts, fd, 'cost207HTx6');
filteredMsg = filter(chan, message);
filteredMsg = awgn(filteredMsg, 10);

fileID1 = fopen('filter_real_correlator1.txt', 'w');
fprintf(fileID1,'%4.6f\n',real(filteredMsg));
fclose(fileID1);

fileID1 = fopen('filter_imag_correlator1.txt', 'w');
fprintf(fileID1,'%4.6f\n',imag(filteredMsg));
fclose(fileID1);

fileID1 = fopen('testing_real_correlator1.txt', 'w');
fprintf(fileID1,'%4.6f\n',real(message));
fclose(fileID1);

fileID1 = fopen('testing_imag_correlator1.txt', 'w');
fprintf(fileID1,'%4.6f\n',imag(message));
fclose(fileID1);
```

Section2. Hardware Chisel code

2.1 Feedback filter:

```
package dfe3
import chisel3._
import chisel3._
import chisel3.experimental.FixedPoint
import chisel3.iotesters.{Backend}
import chisel3.{Bundle, Module}
import dsptools.{DspContext, DspTester}
import dsptools.numbers.{FixedPointRing, DspComplexRing, DspComplex}
import dsptools.numbers.implicits._
import org.scalatest.{Matchers, FlatSpec}
import spire.algebra.Ring
import dsptools.numbers.{RealBits}
import breeze.math.Complex


//tap_coeff_complex only allows three non-zero inputs
class fir_feedbackIo[T <: Data:RealBits](gen: T) extends Bundle {
  val input_complex = Input(DspComplex(FixedPoint(16.W, 12.BP),FixedPoint(16.W,
12.BP) ))
  val tap_coeff_complex = Input(DspComplex(FixedPoint(22.W, 12.BP),FixedPoint(22.W,
12.BP) ))
  val error = Input(DspComplex(FixedPoint(16.W, 12.BP),FixedPoint(16.W, 12.BP) ))
  val tap_index = Input(UInt(12.W))
  val coef_en = Input(Bool())
  val lms_en = Input(Bool())
```

```
  val output_complex = Output(DspComplex(FixedPoint(16.W, 12.BP),FixedPoint(16.W,
12.BP) ))
  val rst = Input(Bool())

  override def cloneType: this.type = new fir_feedbackIo(gen).asInstanceOf[this.type]
}

//step_size: int indicate how much left shift the user want to input, min:0
class fir_feedback[T <: Data:RealBits](gen: T,var window_size: Int, var step_size:
Int) extends Module {
  val io = IO(new fir_feedbackIo(gen))

  val delays = Reg(Vec(window_size, DspComplex(FixedPoint(16.W,
12.BP),FixedPoint(16.W, 12.BP) )))
  val index_count = Reg(init = 0.U(2.W))
  val buffer_complex = Reg(Vec(3, DspComplex(FixedPoint(22.W, 12.BP),FixedPoint(22.W,
12.BP) ))) //vector of reg
  val index = Reg(Vec(3,0.U(12.W)))

  when(io.rst){
   index_count := 0.U
   index(0) := 0.U
   index(1) := 0.U
   index(2) := 0.U
   buffer_complex(0) := DspComplex(0.0.F(22.W,12.BP), 0.0.F(22.W,12.BP))
    buffer_complex(1) := DspComplex(0.0.F(22.W,12.BP), 0.0.F(22.W,12.BP))
    buffer_complex(2) := DspComplex(0.0.F(22.W,12.BP), 0.0.F(22.W,12.BP))
    for (i <- 0 until window_size) {
      delays(i) := DspComplex(0.0.F(16.W,12.BP), 0.0.F(16.W,12.BP))
    }
  }
  .otherwise{
//input in a shift register
  delays(0) := io.input_complex
  for (i <- 1 until window_size) {
    delays(i) := delays(i-1)
  }

//update non-zero coef while count the index
  when (io.coef_en && (index_count < 3.U )) {
    when(io.tap_coeff_complex.imag > 0 || io.tap_coeff_complex.real > 0 ||
          io.tap_coeff_complex.imag < 0 || io.tap_coeff_complex.real < 0) {
      index(index_count) := io.tap_index -1.U
      buffer_complex(index_count) := io.tap_coeff_complex
      index_count := index_count + 1.U
    }
  }
  }
//update lms
  when (io.lms_en) {
    // io.error needs to be conjugated
   buffer_complex(0).real := buffer_complex(0).real - (delays(index(0)).real *
io.error.real +(-delays(index(0)).imag) * io.error.imag)>> step_size
   buffer_complex(0).imag := buffer_complex(0).imag - ((-delays(index(0)).imag) *
io.error.real -delays(index(0)).real * io.error.imag)>> step_size
   buffer_complex(1).real := buffer_complex(1).real - (delays(index(1)).real *
io.error.real +(-delays(index(1)).imag) * io.error.imag)>> step_size
   buffer_complex(1).imag := buffer_complex(1).imag - ((-delays(index(1)).imag) *
io.error.imag -delays(index(1)).real * io.error.imag)>> step_size
   buffer_complex(2).real := buffer_complex(2).real - (delays(index(2)).real *
io.error.real +(-delays(index(2)).imag) * io.error.imag)>> step_size
   buffer_complex(2).imag := buffer_complex(2).imag - (-delays(index(2)).imag *
io.error.imag -delays(index(2)).real * io.error.imag)>> step_size
}
```

```
  when (index_count === 0.U) {
  io.output_complex := DspComplex(0.0.F(16.W,12.BP), 0.0.F(16.W,12.BP))
  } .elsewhen (index_count === 1.U) {
  io.output_complex := delays(index(0))* buffer_complex(0)
  } .elsewhen (index_count === 2.U) {
  io.output_complex := delays(index(0))* buffer_complex(0) +
                        delays(index(1))* buffer_complex(1)
  } .otherwise {
  io.output_complex := delays(index(0))* buffer_complex(0) +
                        delays(index(1))* buffer_complex(1) +
                         delays(index(2))* buffer_complex(2)
  }
}
```

2.1.2 SRAM version of the feedback filter: [written by Henry Zhu]

```
package dfe3

import chisel3._
import chisel3.util._
import chisel3.experimental.FixedPoint
import chisel3.iotesters.{Backend}
import chisel3.{Bundle, Module}
import dsptools.{DspContext, DspTester}
import dsptools.numbers.{FixedPointRing, DspComplexRing, DspComplex}
import dsptools.numbers.implicits._
import org.scalatest.{Matchers, FlatSpec}
import spire.algebra.Ring
import dsptools.numbers.{RealBits}
import breeze.math.Complex

class fir_Io[T <: Data:RealBits](gen: T) extends Bundle {
  val input_complex = Input(DspComplex(gen.cloneType, gen.cloneType))
  val tap_coeff_complex = Input(DspComplex(gen.cloneType, gen.cloneType))
  val tap_index = Input(UInt(10.W))
  val coef_en = Input(Bool())
  val rst = Input(Bool())
  val output_complex = Output(DspComplex(gen.cloneType, gen.cloneType))
  val counter = Input(UInt(10.W))
  override def cloneType: this.type = new fir_Io(gen).asInstanceOf[this.type]
}

// data width 32, 12
class fir[T <: Data:RealBits](gen: => T,var window_size: Int, var step_size: Int)
extends Module {
  val io  = IO(new fir_Io(gen))

  // Instantiated 1st SRAM
  val sram_depth = 512
  val buffer_mem1 = SyncReadMem(DspComplex(gen.cloneType, gen.cloneType), 512)
  val buffer_wen = Wire(Bool()); buffer_wen := true.B //Default value
  val buffer_raddr1 = Wire(UInt(log2Ceil(sram_depth).W)); buffer_raddr1 := 0.U
  val buffer_raddr2 = Wire(UInt(log2Ceil(sram_depth).W)); buffer_raddr2 := 0.U
  val buffer_waddr = Wire(UInt(log2Ceil(sram_depth).W)); buffer_waddr := 0.U
  val buffer_wdata = Wire(DspComplex(gen.cloneType, gen.cloneType));
  val buffer_rdata1 = Wire(DspComplex(gen.cloneType, gen.cloneType));
  val buffer_rdata2 = Wire(DspComplex(gen.cloneType, gen.cloneType));

  // Instantiated 3rd SRAM
  val buffer_mem2 = SyncReadMem(DspComplex(gen.cloneType, gen.cloneType), 512)
  val buffer_raddr3 = Wire(UInt(log2Ceil(sram_depth).W)); buffer_raddr3 := 0.U
  val buffer_rdata3 = Wire(DspComplex(gen.cloneType, gen.cloneType));
```

```
// read and write for all of the srams
when(buffer_wen) {
  buffer_mem1.write(buffer_waddr, buffer_wdata)
}

when(buffer_wen) {
  buffer_mem2.write(buffer_waddr, buffer_wdata)
}


val index_count = Reg(init = 0.U(2.W))
val index = Reg(init = Vec.fill(3){0.U(10.W)})
val buffer_complex = Reg(Vec(3, DspComplex(gen, gen)))
val buffer_index = Reg(DspComplex(gen,gen))
val buffer_index1 = Reg(DspComplex(gen,gen))

// coefficient update block
when (io.rst){
     buffer_complex(0) := DspComplex[T](Complex(0.0, 0.0))
     buffer_complex(1) := DspComplex[T](Complex(0.0, 0.0))
     buffer_complex(2) := DspComplex[T](Complex(0.0, 0.0))
     buffer_index := DspComplex[T](Complex(0.0, 0.0))
     buffer_index := DspComplex[T](Complex(0.0, 0.0))
}
.elsewhen (io.coef_en) {
  when(io.tap_coeff_complex.imag > 0 || io.tap_coeff_complex.real > 0 ||
        io.tap_coeff_complex.imag < 0 || io.tap_coeff_complex.real < 0) {
    index(index_count) := io.tap_index-1.U
    buffer_complex(index_count) := io.tap_coeff_complex
    index_count := index_count + 1.U
  }
}
when(io.counter===0.U)
{
     buffer_index := io.input_complex
}
when(io.counter===1.U)
{
     buffer_index1 := io.input_complex
}

// read/write address update
buffer_waddr := io.counter
buffer_wdata := io.input_complex

buffer_raddr1 := io.counter-index(0)
buffer_raddr2 := io.counter-index(1)
buffer_raddr3 := io.counter-index(2)

when(buffer_raddr1===io.counter){
  buffer_rdata1 := io.input_complex
}
.elsewhen(buffer_raddr1===0.U){
     buffer_rdata1 := buffer_index
}
.otherwise{
  buffer_rdata1 := buffer_mem1(buffer_raddr1)
}

when(buffer_raddr2===io.counter){
  buffer_rdata2 := io.input_complex
}
.elsewhen(buffer_raddr2===0.U){
     buffer_rdata2 := buffer_index
}
```

```
 .otherwise{
    buffer_rdata2 := buffer_mem1(buffer_raddr2)
   }

   when(buffer_raddr3===io.counter){
    buffer_rdata3 := io.input_complex
   }
   .elsewhen(buffer_raddr3===0.U){
        buffer_rdata3 := buffer_index
   }
 .otherwise{
    buffer_rdata3 := buffer_mem2(buffer_raddr3)
   }

 when(index_count===0.U){
       io.output_complex := DspComplex[T](Complex(0.0, 0.0))
   }
   .elsewhen(index_count===1.U){
       io.output_complex := buffer_complex(0)*buffer_rdata1
   }
   .elsewhen(index_count===2.U){
       io.output_complex := buffer_complex(0)*buffer_rdata1 +
buffer_complex(1)*buffer_rdata2
   }
   .elsewhen(index_count===3.U){
       io.output_complex := buffer_complex(0)*buffer_rdata1 +
buffer_complex(1)*buffer_rdata2 + buffer_complex(2)*buffer_rdata3
   }
   //io.output_complex := buffer_complex(0)*buffer_rdata1 +
buffer_complex(1)*buffer_rdata2 + buffer_complex(2)*buffer_rdata3
}
```

2.2 Decision Device:

```
package dfe3

import chisel3._
import chisel3.experimental.FixedPoint
import dsptools.numbers.{RealBits}
import dsptools.numbers.implicits._
import dsptools.DspContext
import dsptools.{DspTester, DspTesterOptionsManager, DspTesterOptions}
import iotesters.TesterOptions
import org.scalatest.{FlatSpec, Matchers}
import math._
import dsptools.numbers._
import breeze.math.Complex


class decision_deviceIo[T <: Data:RealBits](gen: T) extends Bundle {
  val input_complex = Input(DspComplex(gen.cloneType, gen.cloneType))
  val qpsk_en = Input(Bool())
  val output_complex = Output(DspComplex(gen.cloneType, gen.cloneType))
  val error_complex = Output(DspComplex(gen.cloneType, gen.cloneType))
  override def cloneType: this.type = new
decision_deviceIo(gen).asInstanceOf[this.type]
}

class decision_device[T <: Data:RealBits](gen: T) extends Module {
  val io = IO(new decision_deviceIo(gen))
```

```
  when (io.qpsk_en) {
    val positive1 = DspComplex[T](Complex(sqrt(0.5), sqrt(0.5)))
    val positive2 = DspComplex[T](Complex(-sqrt(0.5), sqrt(0.5)))
    val positive3 = DspComplex[T](Complex(sqrt(0.5), -sqrt(0.5)))
    val positive4 = DspComplex[T](Complex(-sqrt(0.5), -sqrt(0.5)))

  when(io.input_complex.real<0){
   when(io.input_complex.imag<0){
      io.output_complex := positive4
    }
    .otherwise{
      io.output_complex := positive2
    }
  }.otherwise {
    when(io.input_complex.imag<0){
      io.output_complex := positive3
    }
    .otherwise{
      io.output_complex := positive1
    }
  }
 }.otherwise{
 val positive = DspComplex[T](Complex(1.0, 0.0))
 val negative = DspComplex[T](Complex(-1.0, 0.0))
   when(io.input_complex.real<0){
        io.output_complex := negative
   }.otherwise {
   io.output_complex := positive
   }
  }
 io.error_complex := io.output_complex - io.input_complex
}
```

2.3.1 Correlator without SRAM: [written by Kate Du and Cindy Chen]

```
package dfe3

import chisel3._
import chisel3.util._
import chisel3.experimental.FixedPoint
import dsptools.numbers.{RealBits}
import dsptools.numbers.implicits._
import dsptools.DspContext
import dsptools.{DspTester, DspTesterOptionsManager, DspTesterOptions}
import iotesters.TesterOptions
import org.scalatest.{FlatSpec, Matchers}
import math._
import breeze.math.Complex
import dsptools.numbers._

class correlatorIo[T <: Data:RealBits](gen: T, var S_w: Int, var C_w: Int, var bp:
Int) extends Bundle {
  val input_complex = Input(DspComplex(FixedPoint(S_w, bp),FixedPoint(S_w, bp) ))
  val output_complex = Output(DspComplex(FixedPoint(S_w, bp),FixedPoint(S_w, bp) ))
  val output_coefficient = Output(DspComplex(FixedPoint(C_w, bp),FixedPoint(S_w,
bp) ))
val rst = Input(Bool())
  override def cloneType: this.type = new correlatorIo(gen, S_w, C_w,
bp).asInstanceOf[this.type]
}
```

```
class correlator[T <: Data:RealBits](gen: T,var S_w: Int, var C_w: Int, var bp: Int)
extends Module {
val io = IO(new correlatorIo(gen, S_w, C_w, bp))
//Set up constant
val delay_size = 128
val n = 7
val W = Array(-1, -1, -1, -1, 1, -1, -1)
val Dk = Array(1, 8, 2, 4, 16, 32, 64)
val output = Reg(Vec(128+128, DspComplex(FixedPoint(S_w, bp),FixedPoint(S_w, bp) )))
val D1 = Reg(Vec(Dk(0), DspComplex(FixedPoint(C_w, bp),FixedPoint(C_w, bp) )))
val D2 = Reg(Vec(Dk(1), DspComplex(FixedPoint(C_w, bp),FixedPoint(C_w, bp) )))
val D3 = Reg(Vec(Dk(2), DspComplex(FixedPoint(C_w, bp),FixedPoint(C_w, bp) )))
val D4 = Reg(Vec(Dk(3), DspComplex(FixedPoint(C_w, bp),FixedPoint(C_w, bp) )))
val D5 = Reg(Vec(Dk(4), DspComplex(FixedPoint(C_w, bp),FixedPoint(C_w, bp) )))
val D6 = Reg(Vec(Dk(5), DspComplex(FixedPoint(C_w, bp),FixedPoint(C_w, bp) )))
val D7 = Reg(Vec(Dk(6), DspComplex(FixedPoint(C_w, bp),FixedPoint(C_w, bp) )))
val DW = Wire(Vec(n, DspComplex(FixedPoint(C_w, bp) ,FixedPoint(C_w, bp) )))
val ra  = Wire(Vec(n, DspComplex(FixedPoint(C_w, bp),FixedPoint(C_w, bp) )))
val rb  = Wire(Vec(n, DspComplex(FixedPoint(C_w, bp),FixedPoint(C_w, bp) )))
val delays = Reg(Vec(delay_size, DspComplex(FixedPoint(C_w, bp) ,FixedPoint(C_w,
bp))))


when(io.rst){
  for (i <-0 until 127) {
    output(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until Dk(0)){
    D1(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until Dk(1)){
    D2(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until Dk(2)){
    D3(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until Dk(3)){
    D4(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until Dk(4)){
    D5(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until Dk(5)){
    D6(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until Dk(6)){
    D7(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until n){
    DW(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until n){
    ra(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until n){
    rb(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }
  for (i <-0 until delay_size){
    delays(i) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
  }


}
.otherwise {
```

```
//set up ShiftRegister for output Complex
output(0) := io.input_complex
for (i<-1 until 128+128){
  output(i) := output(i-1)
}
io.output_complex := output(255)

//delay modules
delays(0) := ra(6)
for (i <- 1 until delay_size) {
    delays(i) := delays(i-1)
}

val temp1 = (delays(127)+rb(6)).real>>8
val temp2 = (delays(127)+rb(6)).imag>>8

when (((temp1*temp1+temp2*temp2)>>18) >0) {
//could not compare in dsp. so I right shift from 12 bits to only 2 bits left.
  io.output_coefficient.real := temp1
  io.output_coefficient.imag := temp2
}
.otherwise {
  io.output_coefficient := DspComplex(0.0.F(C_w.W,bp.BP), 0.0.F(C_w.W,bp.BP))
}
// Set up ShiftRegister for delay
//D1
D1(0) := io.input_complex
DW(0) := D1(0)
//D2
for (i<- 0 until Dk(1)) {
  if(i == 0){
    D2(0) := rb(0)
  }else{
    D2(i) := D2(i-1)
  }
}
DW(1) := D2(Dk(1)-1)
//D3
for (i<- 0 until Dk(2)) {
  if(i == 0){
    D3(0) := rb(1)
  }else{
    D3(i) := D3(i-1)
  }
}
DW(2) := D3(Dk(2)-1)
//D4
for (i<- 0 until Dk(3)) {
  if(i == 0){
    D4(0) := rb(2)
  }else{
    D4(i) := D4(i-1)
  }
}
DW(3) := D4(Dk(3)-1)
//D5
for (i<- 0 until Dk(4)) {
  if(i == 0){
    D5(0) := rb(3)
  }else{
    D5(i) := D5(i-1)
  }
}
DW(4) := D5(Dk(4)-1)
//D6
```

```
for (i<- 0 until Dk(5)) {
  if(i == 0){
    D6(0) := rb(4)
  }else{
    D6(i) := D6(i-1)
  }
}
DW(5) := D6(Dk(5)-1)
//D7
for (i<- 0 until Dk(6)) {
  if(i == 0){
    D7(0) := rb(5)
  }else{
    D7(i) := D7(i-1)
  }
}
DW(6) := D7(Dk(6)-1)
// Calculate ra
for (i <- 0 until n){
  if (i == 0){
    ra(i) := -io.input_complex+DW(i)
  }else if (i == 4){
    ra(i) := ra(i-1)+DW(i)
  }else{
    ra(i) := -ra(i-1)+DW(i)
  }
}
// Calculate rb
for (i <- 0 until n){
  if (i == 0){
    rb(i) := -io.input_complex-DW(i)
  }else if (i == 4){
    rb(i) := ra(i-1)-DW(i)
  }else{
    rb(i) := -ra(i-1)-DW(i)
  }
}
}
}
```

2.3.2 Correlator with SRAM: [written by Henry Zhu]

```
// instantiate the sram
val sram_depth = 256
val buffer_mem = SyncReadMem(DspComplex(FixedPoint(10.W, 6.BP),FixedPoint(10.W,
6.BP)), 256)
val buffer_wen = Wire(Bool()); buffer_wen := true.B //Default value
val buffer_raddr = Wire(UInt(log2Ceil(sram_depth).W)); buffer_raddr := 0.U
val buffer_waddr = Wire(UInt(log2Ceil(sram_depth).W)); buffer_waddr := 0.U
val buffer_wdata = Wire(DspComplex(FixedPoint(10.W, 6.BP),FixedPoint(10.W, 6.BP)));
val buffer_rdata = Wire(DspComplex(FixedPoint(10.W, 6.BP),FixedPoint(10.W, 6.BP)));
val counter = Reg(UInt(12.W))


when(buffer_wen) {
  buffer_mem.write(buffer_waddr, buffer_wdata)
}

counter := counter +1.U
buffer_waddr := (counter-1.U)%256.U  //io.counter_debug % 512.U
buffer_wdata := io.input_complex

when(counter-1.U < 256.U){
```

```
   buffer_rdata := DspComplex(0.0.F(10.W, 6.BP),0.0.F(10.W, 6.BP))
}
.otherwise{
   buffer_rdata := buffer_mem((counter-256.U)%256.U)  //buffer_mem((io.counter_debug-
255.U)%512.U)
}
io.output_complex := buffer_rdata
```

2.4 Datapath:

```
package dfe3

import chisel3._
import chisel3.experimental.FixedPoint
import chisel3.iotesters.{Backend}
import chisel3.{Bundle, Module}
import dsptools.{DspContext, DspTester}
import dsptools.numbers.{FixedPointRing, DspComplexRing, DspComplex}
import dsptools.numbers.implicits._
import org.scalatest.{Matchers, FlatSpec}
import spire.algebra.Ring
import dsptools.numbers.{RealBits}

class dpathtotalIo[T <: Data:RealBits](gen: T, var S_w: Int, var C_w: Int, var bp:
Int) extends Bundle {
  val signal_in = Input(DspComplex(FixedPoint(S_w, bp) ))
  val signal_out = Output(DspComplex(FixedPoint(S_w, bp) ))
  val coeff_in = Input(DspComplex(FixedPoint(C_w, bp) ))
  val coeff_out = Output(DspComplex(FixedPoint(C_w, bp) ))
  val stage = Input(UInt(2.W))
  val count = Input(UInt(12.W))
  val lms_en = Input(Bool())
  val tap_en = Input(Bool())

  override def cloneType: this.type = new dpathtotalIo(gen, S_w, C_w,
bp).asInstanceOf[this.type]
}


class dpathtotal[T <: Data:RealBits](gen: T,var S_w: Int, var C_w: Int, var bp: Int)
extends Module {
 val io = IO(new dpathtotalIo(gen, S_w, C_w, bp))
 val window_size = 128
 val step_size = 5

 //val corr = Module(new correlator(gen, S_w, C_w, bp)).io  //without sram
 val corr = Module(new correlator(gen)).io   //with SRAM
 val dec = Module(new decision_device(FixedPoint(S_w, bp))).io
// val fbf = Module(new fir_feedback(gen,window_size,step_size)).io //fir_feedback
 val fbf = Module(new firFeedbackNoMulti(gen,window_size,step_size, S_w, C_w, bp)).io
//filter without Multiplier


 when (io.stage === 0.U) {
    fbf.rst := true.B
    corr.rst := true.B
 }
//only correlator is working
 when (io.stage === 1.U) {
    fbf.rst := false.B
    corr.rst := false.B
    dec.qpsk_en := false.B
    corr.input_complex := io.signal_in
    io.signal_out := corr.output_complex
```

```
 }

 //dfe is working
 when (io.stage === 2.U) {
  fbf.rst := false.B
  corr.rst := false.B
  corr.input_complex := io.signal_in
  dec.input_complex := corr.output_complex - fbf.output_complex
  dec.output_complex <> fbf.input_complex
  dec.error_complex <> fbf.error
  fbf.tap_coeff_complex := io.coeff_in  //corr.output_coefficient
  fbf.tap_index := io.count
  fbf.lms_en := io.lms_en
  fbf.coef_en := io.tap_en
  io.signal_out := dec.output_complex
  dec.qpsk_en := false.B
 }

  when (io.stage === 3.U) {
  fbf.rst := false.B
  corr.rst := false.B
  corr.input_complex := io.signal_in
  dec.input_complex := corr.output_complex - fbf.output_complex
  dec.output_complex <> fbf.input_complex
  dec.error_complex <> fbf.error
  fbf.tap_coeff_complex := io.coeff_in  //corr.output_coefficient
  fbf.tap_index := io.count
  fbf.lms_en := io.lms_en
  fbf.coef_en := io.tap_en
  io.signal_out := dec.output_complex
  dec.qpsk_en := true.B
 }
 io.coeff_out := corr.output_coefficient

}
```

2.4 Control Unit:

```
package dfe3
import chisel3._
import chisel3.experimental.FixedPoint
import dsptools.numbers.{RealBits}
import dsptools.numbers.implicits._
import dsptools.DspContext
import dsptools.{DspTester, DspTesterOptionsManager, DspTesterOptions}
import iotesters.TesterOptions
import org.scalatest.{FlatSpec, Matchers}
import math._
import dsptools.numbers._
import scala.collection.mutable.HashMap
import scala.collection.mutable.ArrayBuffer
import spire.algebra.Ring
import chisel3.util._
import breeze.math.Complex

class ctrlIo[T <: Data:RealBits](gen: T, var C_w: Int, var bp: Int) extends Bundle {
  val enable = Input(Bool())
  val reset = Input(Bool())
  val stage = Output(UInt(2.W))
  val count = Output(UInt(12.W))
  val fbf_coeff = Input(DspComplex(FixedPoint(C_w, bp) ))
  //val ga_coeff = Input(Bool()) //might needed
  val coeff_output = Output(DspComplex(FixedPoint(C_w, bp) ))
  val tap_en = Output(Bool())
```

```
  val lms_en = Output(Bool())
}

class ctrl[T <: Data:RealBits](gen: T, var C_w: Int, var bp: Int) extends Module {
 val io = IO(new ctrlIo(gen, C_w, bp))

 //import submodule
 val count = Reg(init = 0.U(12.W))
 val s_idle :: s_correlator :: s_dfe_bpsk :: s_dfe_qpsk :: Nil = Enum(4)
 val stage = Reg(init = s_idle)

 io.lms_en := false.B
 io.tap_en := true.B

 switch (stage) {
  is (s_idle) {
    count := 0.U
    when (io.enable) {
        stage := s_correlator
    }
  }

  is (s_correlator) {
      when (io.reset) {
      stage := s_idle
    }
    .otherwise {
    when (io.fbf_coeff.real > 0 || io.fbf_coeff.real < 0 || io.fbf_coeff.imag > 0 ||
io.fbf_coeff.imag < 0) {
      count := count + 1.U
      stage := s_dfe_bpsk
      io.tap_en := true.B
      io.coeff_output := DspComplex(0.F(C_w.W,bp.BP),0.F(C_w.W,bp.BP))
    }
   }
  }

  is (s_dfe_bpsk) {
    when (io.reset) {
      stage := s_idle
    }
    .otherwise {
    count := count +1.U
    io.coeff_output := io.fbf_coeff //NOT SURE
    when (count === 256.U) {
      stage := s_dfe_qpsk
      io.tap_en := false.B
    }
  }
}

  is (s_dfe_qpsk) {
    when (io.reset) {
        stage := s_idle
    }
    .otherwise {
    count := count +1.U
    io.coeff_output := io.fbf_coeff
    when (count === 513.U) {
      io.tap_en := false.B
    }
    when (count === 1255.U) {
      stage:= s_idle
    }
  }
```

```
}
 } //end switch
 io.stage := stage
 io.count := count
}
```

2.5 Top Module:

```
package dfe3

import chisel3._
import chisel3.experimental.FixedPoint
import chisel3.iotesters.{Backend}
import chisel3.{Bundle, Module}
import dsptools.{DspContext, DspTester}
import dsptools.numbers.{FixedPointRing, DspComplexRing, DspComplex}
import dsptools.numbers.implicits._
import org.scalatest.{Matchers, FlatSpec}
import spire.algebra.Ring
import dsptools.numbers.{RealBits}


class dfe3Io[T <: Data:RealBits](gen: T, var S_w: Int, var C_w: Int, var bp: Int)
extends Bundle {
  val signal_in = Input(DspComplex(FixedPoint(S_w, bp) ))
  val signal_out = Output(DspComplex(FixedPoint(S_w, bp) ))
  val enable = Input(Bool())
  val reset = Input(Bool())
}

class dfe3Main[T <: Data:RealBits](gen: T, var S_w: Int, var C_w: Int, var bp: Int)
extends Module {
 val io = IO(new dfe3Io(gen, S_w, C_w, bp))

 val dpath = Module(new dpathtotal(gen, S_w, C_w, bp)).io
 val ctrl = Module(new ctrl(gen, C_w, bp)).io

 ctrl.enable := io.enable
 ctrl.reset := io.reset

 dpath.signal_in := io.signal_in
 io.signal_out := dpath.signal_out

 ctrl.stage <> dpath.stage
 ctrl.count <> dpath.count
 ctrl.fbf_coeff <> dpath.coeff_out
 ctrl.coeff_output <> dpath.coeff_in
 ctrl.tap_en <> dpath.tap_en
 ctrl.lms_en <> dpath.lms_en
}



object dfe3MainTest extends App {
  var S_w = 16 //22
  var C_w = 22
  var bp = 12

  Driver.execute(args.drop(3), () => new dfe3Main(FixedPoint(22, 12), S_w, C_w, bp))

}
```

2.6 Simplified version of the multiplication:

```
package dfe3

import chisel3._
import chisel3.experimental.FixedPoint
import chisel3.iotesters.{Backend}
import chisel3.{Bundle, Module}
import dsptools.{DspContext, DspTester}
import dsptools.numbers.{FixedPointRing, DspComplexRing, DspComplex}
import dsptools.numbers.implicits._
import org.scalatest.{Matchers, FlatSpec}
import spire.algebra.Ring
import dsptools.numbers.{RealBits}
import breeze.math.Complex
import spire.math.{ConvertableTo}


class SimpMultiIo[T <: Data:RealBits](gen: T, var S_w: Int, var bp: Int) extends
Bundle {
  val input_complex = Input(DspComplex(FixedPoint(S_w, bp),FixedPoint(S_w, bp) ))
  val sign = Input(UInt(3.W))
  val output_complex = Output(DspComplex(FixedPoint(S_w, bp),FixedPoint(S_w, bp) ))
  override def cloneType: this.type = new SimpMultiIo(gen, S_w,
bp).asInstanceOf[this.type]
}

class SimpMulti[T <: Data:RealBits](gen: T, var S_w: Int, var bp: Int) extends Module
{
  val io = IO(new SimpMultiIo(gen, S_w, bp))

  //BPSK
  when (io.sign(2) === 0.U){
    when (io.sign(1) === 0.U){
      io.output_complex.real :=  io.input_complex.real
      io.output_complex.imag :=  io.input_complex.imag
    } .otherwise{
      io.output_complex.real :=  -io.input_complex.real
      io.output_complex.imag :=  -io.input_complex.imag
    }
  } .otherwise{
    //QPSK
    when(io.sign(1) === io.sign(0)){
      when (io.sign(1)=== 0.U){
        io.output_complex.real :=  (io.input_complex.real - io.input_complex.imag) *
{ ConvertableTo[FixedPoint].fromDouble(0.7071067811865475244) }
        io.output_complex.imag :=  (io.input_complex.real + io.input_complex.imag) *
{ ConvertableTo[FixedPoint].fromDouble(0.7071067811865475244) }
      } .otherwise{
        io.output_complex.real :=  (-io.input_complex.real + io.input_complex.imag) *
{ ConvertableTo[FixedPoint].fromDouble(0.7071067811865475244) }
        io.output_complex.imag :=  (-io.input_complex.real - io.input_complex.imag) *
{ ConvertableTo[FixedPoint].fromDouble(0.7071067811865475244) }
      }
    } .otherwise{
      when (io.sign(1) === 0.U){
        io.output_complex.real :=  (io.input_complex.real + io.input_complex.imag) *
{ ConvertableTo[FixedPoint].fromDouble(0.7071067811865475244) }
        io.output_complex.imag :=  (-io.input_complex.real + io.input_complex.imag) *
{ ConvertableTo[FixedPoint].fromDouble(0.7071067811865475244) }
      } .otherwise{
        io.output_complex.real :=  (-io.input_complex.real - io.input_complex.imag) *
{ ConvertableTo[FixedPoint].fromDouble(0.7071067811865475244) }
        io.output_complex.imag :=  (io.input_complex.real - io.input_complex.imag) *
{ ConvertableTo[FixedPoint].fromDouble(0.7071067811865475244) }
      }
    }
```

```
  }
}
```

2.7 Apply the multiplication to filter:

```
package dfe3

import chisel3._
//import chisel3.core._
import chisel3.experimental.FixedPoint
import chisel3.iotesters.{Backend}
import chisel3.{Bundle, Module}
import dsptools.{DspContext, DspTester}
import dsptools.numbers.{FixedPointRing, DspComplexRing, DspComplex}
import dsptools.numbers.implicits._
import org.scalatest.{Matchers, FlatSpec}
import spire.algebra.Ring
import dsptools.numbers.{RealBits}
import breeze.math.Complex
import math._

class firFeedbackNoMultiIo[T <: Data:RealBits](gen: T,var S_w: Int, var C_w: Int, var
bp: Int) extends Bundle {
  val input_complex = Input(DspComplex(FixedPoint(S_w, bp),FixedPoint(S_w, bp) ))
  val tap_coeff_complex = Input(DspComplex(FixedPoint(C_w, bp),FixedPoint(C_w, bp) ))
  val error = Input(DspComplex(FixedPoint(S_w, bp),FixedPoint(S_w, bp) ))
  val tap_index = Input(UInt(12.W))
  val coef_en = Input(Bool())
  val lms_en = Input(Bool())
  val output_complex = Output(DspComplex(FixedPoint(S_w, bp),FixedPoint(S_w, bp) ))
  val rst = Input(Bool())

  override def cloneType: this.type = new firFeedbackNoMultiIo(gen, S_w, C_w,
bp).asInstanceOf[this.type]
}


//step_size: int indicate how much left shift the user want to input, min:0
class firFeedbackNoMulti[T <: Data:RealBits](gen: T,var window_size: Int, var
step_size: Int, var S_w: Int, var C_w: Int, var bp: Int) extends Module {
  val io = IO(new firFeedbackNoMultiIo(gen, S_w, C_w, bp))

  val Multi_0 = Module(new SimpMulti(gen, S_w,bp)).io
  val Multi_1 = Module(new SimpMulti(gen, S_w,bp)).io
  val Multi_2 = Module(new SimpMulti(gen, S_w,bp)).io

  val delays = Reg(Vec(window_size, UInt(3.W)))
  val index_count = Reg(init = 0.U(2.W))
  val buffer_complex = Reg(Vec(3, DspComplex(FixedPoint(S_w, bp),FixedPoint(S_w,
bp) )))
  val index = Reg(Vec(3,0.U(12.W)))
  val sign = Wire(UInt (3.W))
  when (io.input_complex.imag > 0){
    when (io.input_complex.real >= 0){
      sign := 4.U
    }.otherwise{
      sign := 6.U
    }
  } .elsewhen(io.input_complex.imag < 0){
    when (io.input_complex.real >= 0){
      sign := 5.U
    }.otherwise{
```

```
      sign := 7.U
    }
  }.otherwise{
    when(io.input_complex.real >= 0){
      sign := 0.U
    } .otherwise{
      sign := 2.U
    }
  }

  when(io.rst){
    buffer_complex(0) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
    buffer_complex(1) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
    buffer_complex(2) := DspComplex(0.0.F(S_w.W,bp.BP), 0.0.F(S_w.W,bp.BP))
    index_count := 0.U
    index(0) := 0.U
    index(1) := 0.U
    index(2) := 0.U
    for (i <- 0 until window_size) {
      delays(i) := 0.U
    }
  }
  .otherwise{
  delays(0) := sign
  for (i <- 1 until window_size) {
    delays(i) := delays(i-1)
  }
//update non-zero coef while count the index
  when ((io.coef_en) && (index_count < 3.U )) {
    when(io.tap_coeff_complex.imag > 0 || io.tap_coeff_complex.real > 0 ||
          io.tap_coeff_complex.imag < 0 || io.tap_coeff_complex.real < 0) {
      index(index_count) := io.tap_index -1.U
      buffer_complex(index_count) := io.tap_coeff_complex
      index_count := index_count + 1.U
    }
  }
  }
//update lms

  when (io.lms_en) {
    // io.error needs to be conjugated
    val error = Reg(DspComplex(gen,gen))
    error.real := io.error.real >> step_size
    error.imag := io.error.imag >> step_size

    val Multi_3 = Module(new SimpMulti(gen, S_w,bp)).io
    val Multi_4 = Module(new SimpMulti(gen, S_w,bp)).io
    val Multi_5 = Module(new SimpMulti(gen, S_w,bp)).io

    Multi_3.input_complex := error
    Multi_3.sign := delays(index(0))

    Multi_4.input_complex := error
    Multi_4.sign := delays(index(1))

    Multi_5.input_complex := error
    Multi_5.sign := delays(index(2))

    buffer_complex(0) := buffer_complex(0) - Multi_3.output_complex
    buffer_complex(1) := buffer_complex(1) - Multi_4.output_complex
    buffer_complex(2) := buffer_complex(2) - Multi_5.output_complex

}

Multi_0.input_complex := buffer_complex(0)
```

```
Multi_0.sign := delays(index(0))
Multi_1.input_complex := buffer_complex(1)
Multi_1.sign := delays(index(1))
Multi_2.input_complex := buffer_complex(2)
Multi_2.sign := delays(index(2))
io.output_complex := Multi_0.output_complex + Multi_1.output_complex +
Multi_2.output_complex

}
```