Digital Radio Baseband and Testbed for Next Generation Wireless System

Xiaohui Li

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-73
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-73.html

May 13, 2016

Digital Radio Baseband and Testbed

for

Next Generation Wireless System

Capstone Project Final Report

Xiaohui Li

MEng EECS

May 2016

Table of Contents

Chapter 1

Technical Contributions

Introduction

With the name “Digital Radio Baseband and Testbed for Next Generation Wireless Systems”, our project aims to build several components of the receiver in an open source Software defined radio (SDR) block generator with Chisel, which is a hardware construction language developed at UC Berkeley.

In our daily lives, radios are used everywhere, Bluetooth headphones, Wi-Fi connection, cellphone LTE network and so on, all of these are radios but with different standards. Moreover, standards are defined differently in each country/region and new standards are being designed and implemented from time to time as well. In each radio, digital signal processing (DSP) is needed to process the signal in order to get the original data from the transmission waves. Even though SDR, in which the processing part can be modified by software, is popular nowadays, designing DSP circuit for different standards is still time and cost consuming as there are so many different standards and each of them requires a specific design.

This is the main reason why our team, which is part of a bigger team that has been working on this project for a while, is trying to improve the design process by creating a generator for the hardware design. It is a solution to save the troubles of designing from scratch for each standard by building a generic code which can generate the design for customized standard. With our generic code, the users only need to define the parameters and set them in the code, then the design, described in hardware language, will be generated so the user can implement or add the functions they need to the SDR design for experiment without going through the design of the fundamental SDR. With the ease of designing SDR using the generator, start-up companies

which want to build new products based on certain standards can have the design in a short time with minimum cost.

Our team is responsible for several blocks used in the receiver of the radio. At the beginning, we built Cordic, which is a useful and simple algorithm implementation used in the receiver, to learn the new tool Chisel. After we got more familiar with the tool and DSP knowledge, we then worked on demodulator, automatic gain controller (AGC) and channel estimation with equalizer. Naing Ye Aung will have more detail for the demodulator as he was the one who completed and improved it. Gilbert Lityo and I worked on AGC and channel estimation together and he will talk more about the Cordic and AGC. In this paper, I'll explain the work we have done for the channel estimation with equalizer.

Background

Software defined radio is getting more popular because of several reasons. Compared to the traditional radio design, in which most of the processes are implemented for analog signals and each process is accomplished by a specifically designed circuit. Anytime there is a modification, a new design is needed and the whole circuit will need to be re-built. SDR is named because it started using digital technology in the signal processing units around 1980s and then most of the main functions in the whole process have been controlled by software (Rohde & Schwarz, 2014). As there are various of standards and new standards are being used over time, SDR gradually replaced the traditional analog radio because of its versatility and ease of modification.

With the versatility of SDR, changing standards can be as easy as changing a parameter of a software. But as there is tradeoff between the versatility of the SDR and some other factors like the area and power of the circuit. Special designs are still needed for SDR with different requirements or new standard. The idea of hardware generator is getting more attentions

nowadays as it saves time and cost compared to traditional hardware design. An ideal generator should be able to generate SDR blocks which can process the signals with the given parameters of any standards. This generator will be very helpful because the basic structure of the SDR is shared but special design is needed as there are countless number of standards in wireless communication. Using a generator will significantly reduce the designing time and cost compared to designing each standard from scratch.

Before designing a component in SDR, it is crucial to have a whole picture about how an SDR works. As shown in Figure 1, the top chain is the transmitter, signal start from left to the right. The bottom chain is the receiver, the signal goes through from right to left, which is the inverse of the transmitter.

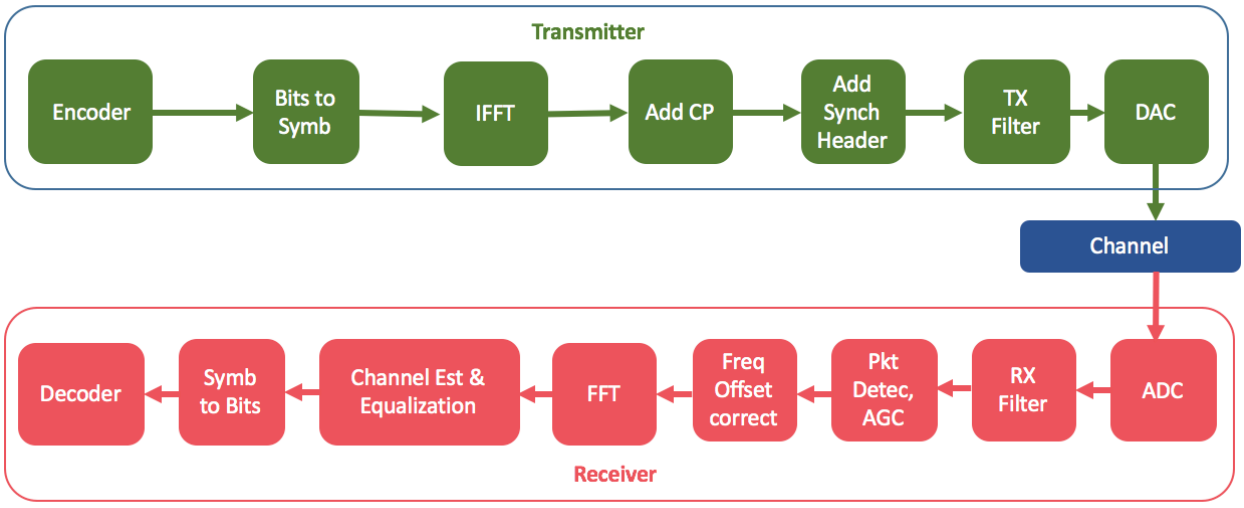

Figure 1: Block diagram showing the transmission of signal.

At the start of transmitter process, the data that needs to be transmitted first get encoded with different methods. For example, the low density parity check (LDPC). LDPC adds some parity bits to the data and they will be used later in the receiver to determine whether the received data is the same as the original data. Then the data in bits get converted to symbols which means the

data, which contains 0s and 1s, is converted into waveforms in frequency domain. After going through the IFFT (Inverse Fast Fourier Transform), the frequency domain waveforms which contains the data becomes time domain waveforms that can be transmitted. After the conversion from frequency domain to time domain, cyclic prefix (CP) is needed to indicate the begin and the end of each cycle, the synch header is similar as it contains some information about the data which will be used in the receiver to resume the signal. After all these processes, the signal now becomes waveform and it will go through a transmitter filter before it gets converted to analog signal in digital analog convertor (DAC) and get transmitted.

The receiver in the bottom is the inverse of the transmitter. Starting from the right, the analog signal gets converted to digital signal, then it goes through the filter to remove some noise before determining the signal range and position using the header, cycle prefix. Before we convert the time domain waveform to frequency domain waveform with FFT (Fast Fourier Transform), the frequency offset of the carrier wave caused by the transmission channel and interference needs to be detected and fixed. In the frequency offset corrector, Cordic is used to detect the angle between the shifted and original signal. After the corrected signal has gone go through the FFT, the frequency domain signal is then converted to data bits by the demodulator. The result is called soft bits as they need to be checked with the parity bits to determine the right value.

After the signal gets transmitted from the antenna of the transmitter, it needs to travel to the antenna of the receiver. In between, the signal will encounter interference which could affect the data that is stored in it, we call this journey in between the “channel”. To gain better accuracy of the transmitted signal, we want to estimate the channel and eliminate the impacts from the channel before we convert the signal to bits. This is why we add a channel estimation block after FFT. After we know the effect from the channel, we will inverse the effect in order to recover the

original signal, this process is done in the equalizer. For example, a signal's amplitude in a certain frequency get reduced by half during the transmission (in the channel) due to noise, reflection or other effects, the channel estimation block will estimate that the channel is "divided by 2" and the equalizer will "multiply 2" to the received signal as to recover the original signal.

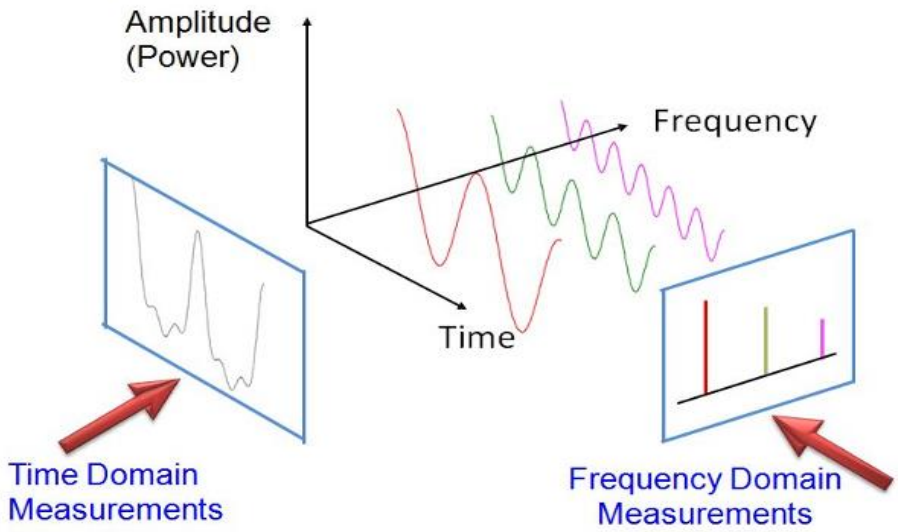

Figure 2: Signal Presented in Time Domain and Frequency Domain. (COMM Tech)

Design Principles

The primary purpose of our hardware generator is used to design the receiver for LTE standard, which uses orthogonal frequency division multiplexing (OFDM) as signal carriers and access schemes. Instead of using a single carrier with wide bandwidth, OFDM uses multiple narrow-band subcarriers and each subcarrier is "orthogonal" to any others which means that they are mutually exclusive and won't interfere with any other subcarriers. As shown in figure 2, a signal in time domain can be represented as the sum of several *sin* waves in different frequencies with different amplitude and phase shift. From the outputs of the FFT in the receiver, we can know the amplitude and the phase shift of each subcarrier from the signal and when summing them together, we should be able to get the original signal transmitted.

This is true only if the channel doesn't have any impact on the signal. In reality, the channel will distort the signal, even worse, it distorts signal differently in each frequency. This is the reason why we need to find a way to estimate the channel and reverse the distortion. One way is to add pilot tone. The pilot tone is a signal known by both the transmitter and the receiver. There are different ways to place pilot tones in order to estimate the channel accurately while not occupying too much space. The most basic types are block type and comb type as shown in the following Figure 3.

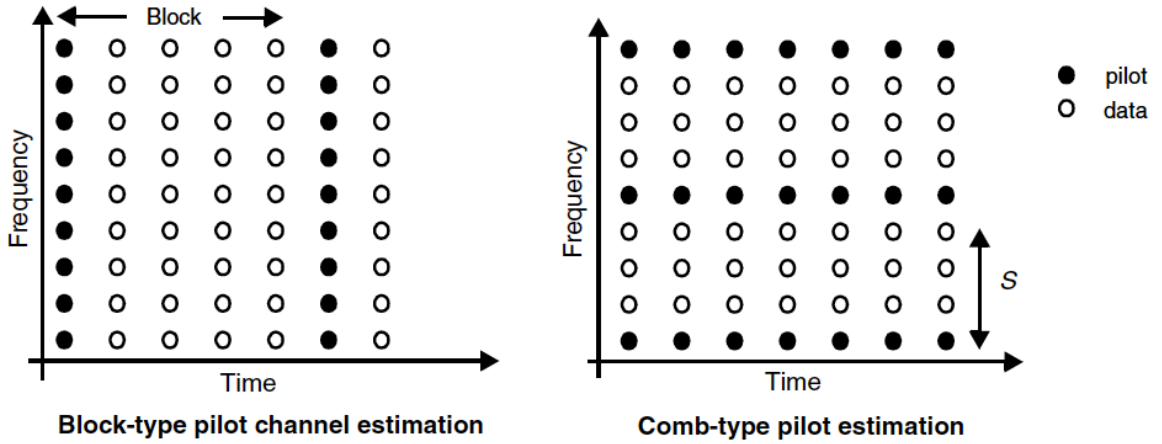

Figure 3: Pilot tone position arrangement examples.

In the block type pilot channel estimation block, the pilots are placed in every frequency, and sent once in a while in time domain. This arrangement can predict the channel well if the channel doesn't change much in a short time, or it's good when the transmission time is short but robust. While in the Comb type pilot channel estimation block, only certain frequency has pilots but they are sending the pilot tone continuously.

For either type, the channel estimation block will compare the pilot received with the known pilot, then find the difference between them and determine the function to use in order to make the received pilot the same as the set pilot. As it's connected to the equalizer, channel estimation

will output the function so the equalizer will apply the function to all signals assuming all signals get the same distortion as the pilot signal.

In our design, the inputs come from the FFT. The FFT is designed to stream one output each time. As shown in figure 4, the *f(k)* axis represent the subcarrier frequencies the signal has, *t(n)* axis represents the signal in the time duration. Even though a signal that's sent in an instant of time *t(n)* contains *k* data, the FFT only sent one data at a time, for example $X_2(2)$, followed by $X_3(2)$. $X_3(2)$ means this is the value of the third subcarrier that's sent at time 2. In this example, the column in green are set to be the pilot tones while the column in white represent the subcarriers that hold the data that are being transmitted.

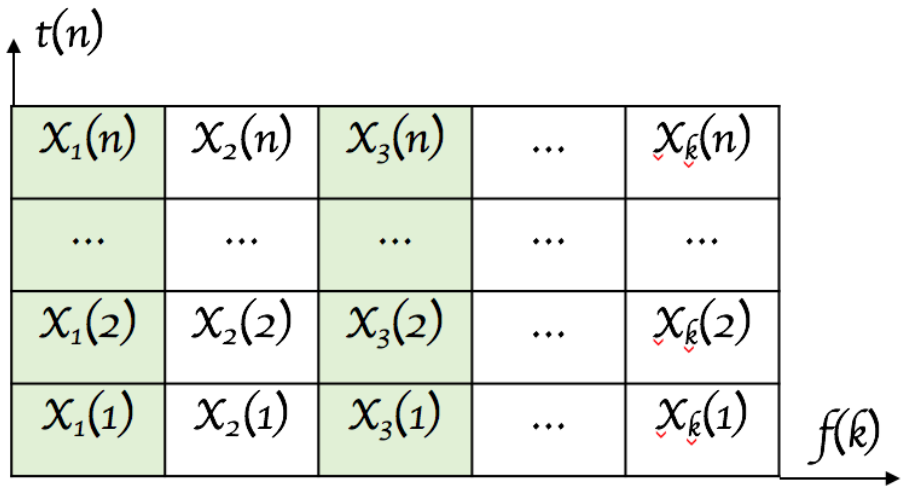

Figure 4: Streaming Inputs Order.

Several algorithms for an adaptive filter can be used. The two options we have are normal least mean squares (LMS) and error sign LMS. As shown in figure 5, the linear filter is an adaptive filter as the filter coefficient, also called as weight, depends on the result of the adaptive algorithm. Weight is applied to the input signal in order to match it with the desire signal. The output of the adaptive algorithm, which is used to determine the function of the filter, is updated using the error signal, *e(n)*. The error signal represents the difference between the desired signal

d(n), which is the original pilot signal, and the filtered received pilot signal *y(n)*. The less the error is, the better the filter is to reverse the channel as it's output *y(n)* has a small difference with the desired signal. Even though there are usually more than one pilot tones and subcarriers, only one LMS calculation block and equalizer is needed as the input is streaming only one each time. However, we need to include a memory to store the previous weight for each pilot tone so with every new pilot tone, we can update the weight according to the output that used previous weight.

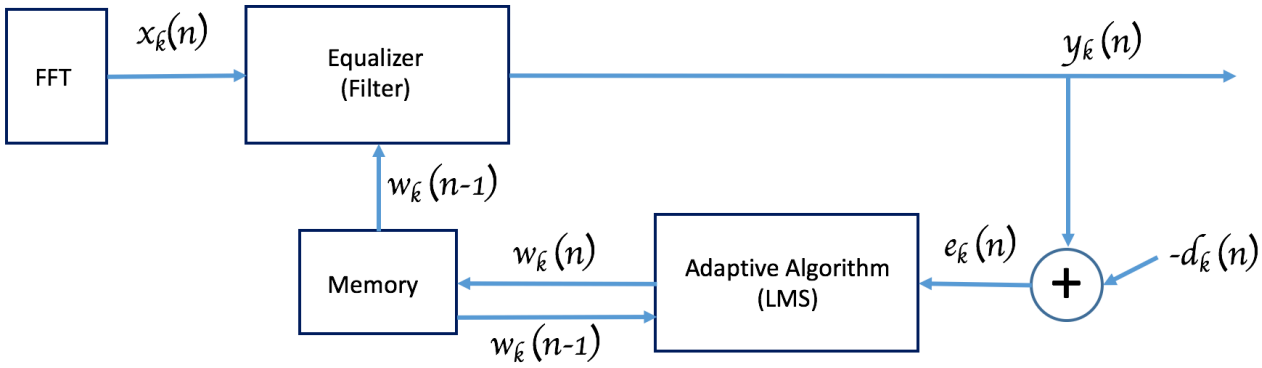

Figure 5: Block diagram for channel estimation and equalization using LMS Algorithm.

Implementation

We implemented the LMS algorithms first in Matlab as a reference and to better understand the algorithms. In Matlab, there are built-in functions that can be used as a LMS filter directly, with those functions as a reference, we used the input and output to verify our implementation.

After verifying the algorithm, we implemented it in Chisel. As a basic version, we used the comb type arrangement of pilot tone signal, in this way, we can determine whether the signal is a pilot tone by checking its position with a counter. The position of the pilot tone is defined at the run time as it is a parameter. Also, the value of the original/desired pilot tone, which is used as a reference *d(n)*, is a set value. As LMS is an adaptive filter, it's trying to minimize the error

between the desired signal and the output signal by updating its filter coefficient every time it gets a known signal or pilot tone. As time goes by, it gradually has more and more pilot tones, which leads to better estimation of the channel, assuming the channel doesn't change much in this period. As shown in the following figure 6, the blue wave is the original signal, the red wave is the signal with noise, the green wave is the output signal that processed through the channel estimation block. We can see that the recovered signal is getting closer to the original signal with more pilot tones received.

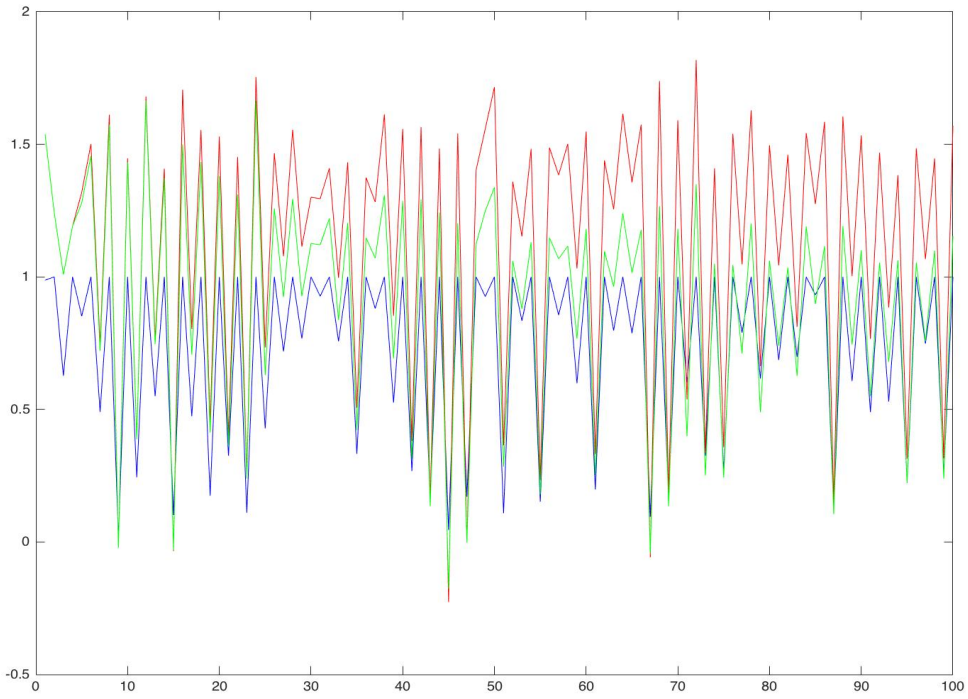

Figure 6: Signal waveforms of clean, noisy and recovered signals.

There are several important parameters to set in the channel estimation block, one of them is the step size. The step size in the LMS algorithm is used to determine how much the reversing function should change each time to make the output get smaller error compared to the desired value. With a big step size, the signal can adapt to the desire signal faster if the error is big. However, after a few adjustments, when the error between the desired signal and the output

signal is close to the step size, it might not be able to improve it further. In another case, if the step size is small, LMS can get a reverse function with smaller error, but it will take much more steps if the error was big in the beginning. With the generator, the designer should be able to set how precise the reverse function can be or how fast it will get to small error to recover the original signal by setting the LMS filter step size.

There are several LMS calculations methods we can choose from as well. The main difference is the equation to calculate how much each time the weight *w(n)*, is changed. We implement two of them and this can be selected by changing the parameter *algorithm*. When *algorithm* = 0, it will generate the LMS block using the normal LMS algorithm in which the weight is updated by: *mu * error * in.* If *algorithm* = 1, it will generate the LMS block using the Error Sign algorithm in which the weight is updated by: *mu * sign(error) * in*. With normal LMS algorithm, when *error* is bigger than 1, it has a bigger step size so it will get closer to the original signal faster. When error is getting smaller, the change becomes smaller as well so it can adjust more precisely to get to the original signal. With error sign LMS algorithm, the step size only depends on the input signal as it's only using the sign of the error, which is either 1 or -1.

Other than the LMS algorithm which is used to mimic the reverse function of the channel for the pilot tones, the algorithm used to determine the reverse function for the data will affect the error rate a lot as well. For example, in the comb type pilot shown on the left in figure 3, pilot 1 and pilot 2 might be affected differently in the channel, so the functions to reverse them back might be different. For the basic step version, shown on the left of figure 7, the weight (red), which represent the reverse function, for the first pilot tone (green) will be applied to all the data (blue) between the first and second pilot tone, also the reverse function for the second pilot tone will be applied to the data between the second and third pilot tone and so on. This is called step

interpolation which means the reverse function is determined by only one pilot tone. For a better estimation, two pilot tones can be used to determine the weight for the data in between. For example, as shown ion the right in figure 7, if there are 2 data signals in between pilot tone 1 and pilot tone 2, the first data will add 2/3 of the weight from pilot tone 1 and 1/3 of the weight from pilot tone 2 to get the final weight for itself. And the second data will add 1/3 of the weight from pilot tone 1 and 2/3 of the weight from pilot tone 2 to get the final weight as it's closer to pilot tone 2. That's why there is another parameter, *interp*, that the user can set to indicate what impairment algorithm they want to implement in the channel estimation block. When *interp* = 0, it will use the step interpolation while *interp* = 1 will use the linear interpolation.

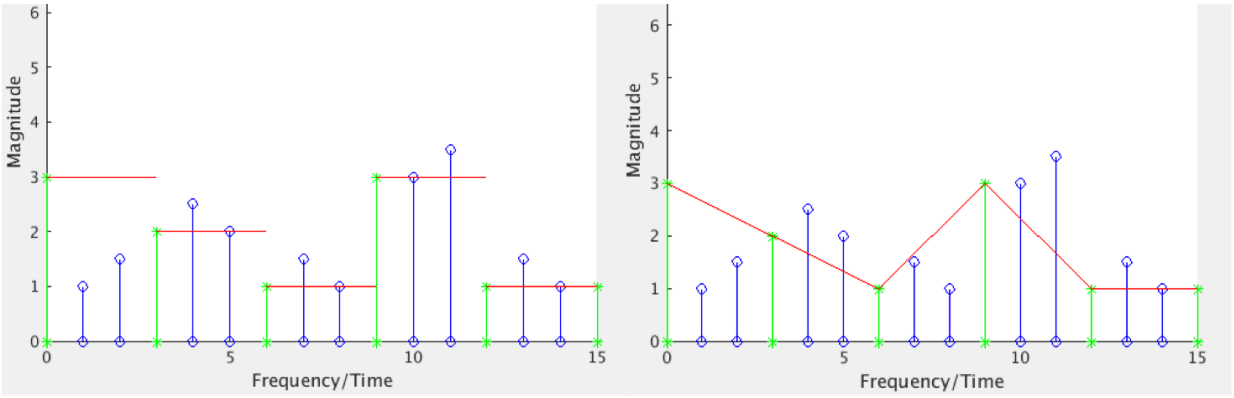

Figure 7: Step and linear interpolation methods.

Testing Method and Results

It will be ideal if we can have the received signals go through the whole process from FFT to channel estimation and then demodulation so we can compare the recovered signal to the original ones and get the bit error rate (BER). In this paper, different combination of the parameters is set and the output of the channel estimation and equalization block is then compared to the original signal directly and check the difference by plotting the squared value of the difference between the recovered signal and the original signal.

Following are the parameters that can be modified to obtain different designs of channel estimation block and some output examples.

interp: Interpolation method. Choose from step, or linear interpolation.

algorithm: LMS algorithm. Error sign or normal LMS.

pipeline: Choose from including 1, 2 or 3 pipeline stages.

mu: Step size, determine how much the weight is updated each time

Matlab is used to generate the noisy signal which will be the input of the Channel estimation block. At first, a simple version of noise is generated by scaling all the signals by the same value, for example, 2, and then add white Gaussian noise with EbNo=10, which means the noise has 1/10 energy of the original signal. The noisy signal is compared to the original signal in the bottom of Figure 8.

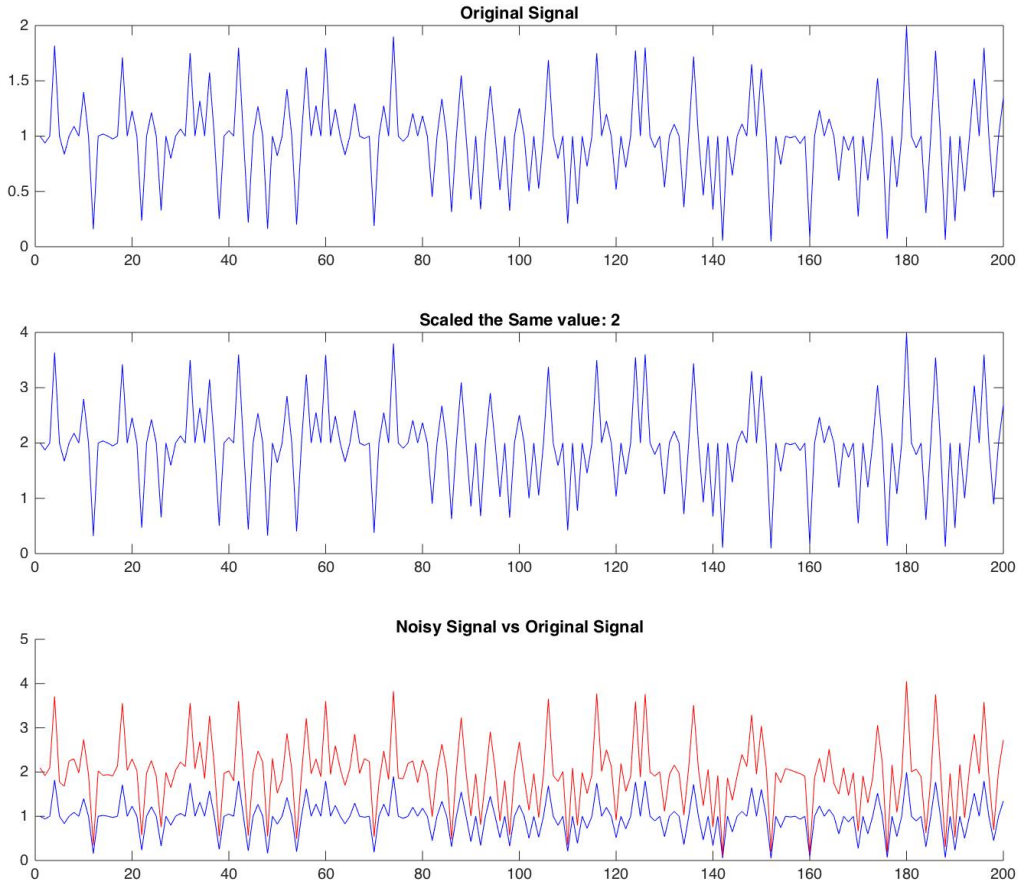

Figure 8: Generate the simple version of noisy signal.

Given the noisy signal as the input, following are the results using different algorithm and interpolation methods.

The signal in blue is the original signal, the signal in red is the noisy signal which is the input of the channel estimation block. The signal in green is the recovered signal that we get from the channel estimation.

For the following examples, we set the pt_position =2, which means there is a pilot tone every other input so the process can go through more iterations in 200 cycles. It is shown in the plots that the recovered signal (in green) is getting closer to the original signal after few iterations.

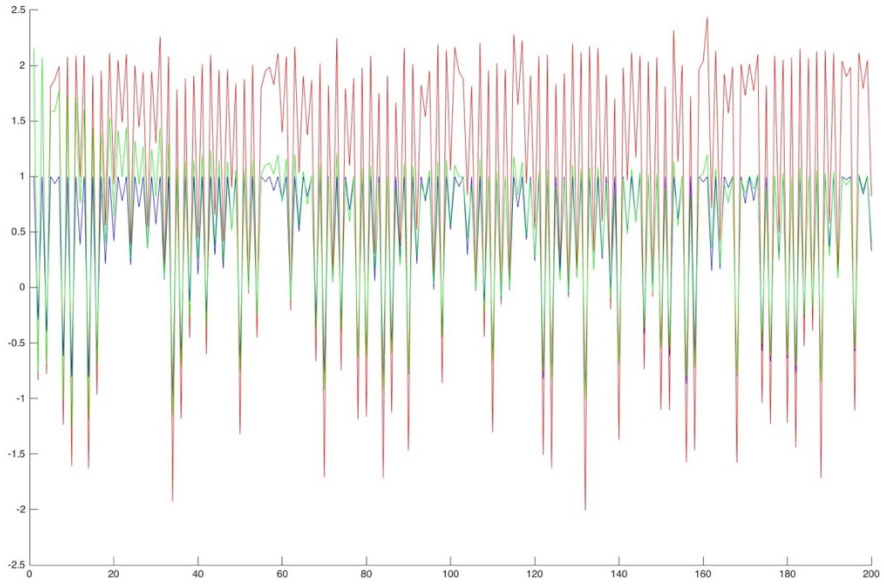

Figure 9: Signal comparison with linear interpolation and LMS algorithm.

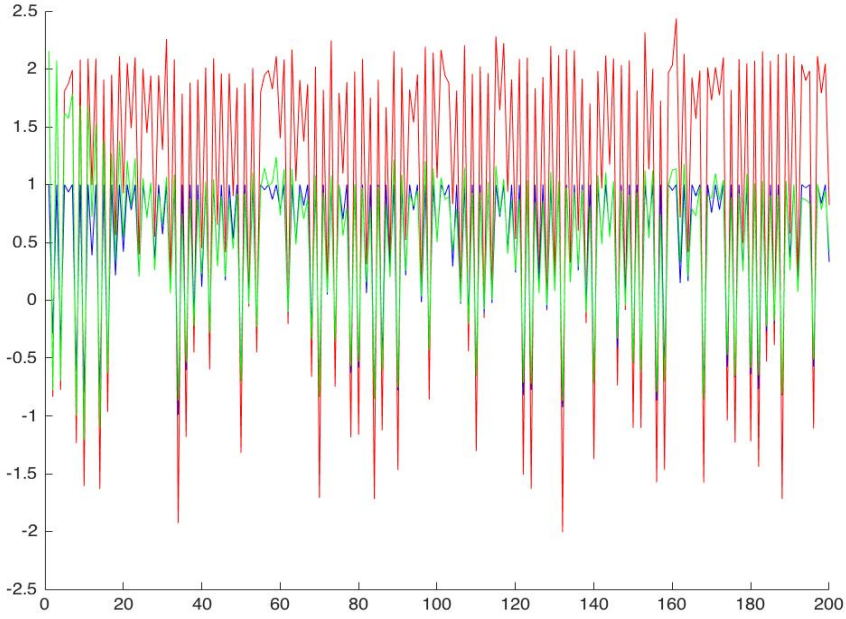

Figure 10: Signal comparison with linear interpolation and error sign algorithm.

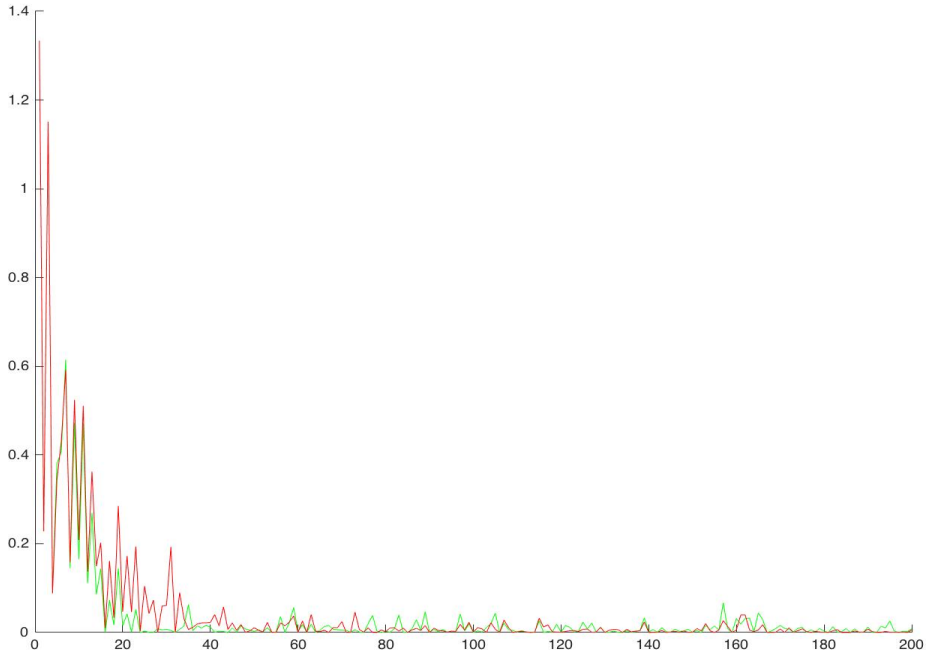

Figure 11: Error comparison of error sign algorithm and LMS algorithm.

To compare these two algorithm, the the square of the difference is plotted in Figure 11, both of them used Linear interpolation, the red is with Error Sign algorithm while the green is with LMS algorithm. We can see that both of the them recover the signal pretty well and LMS (in green) gets to small error faster compared to Error Sign algorithm. With multiple tests and summing up the difference between the original signal and the recovered signal for each test, the sum of the difference using LMS or Error Sign is similar to each other. So there is no obvious advantage of using LMS algorithm instead of Error Sign algorithm.

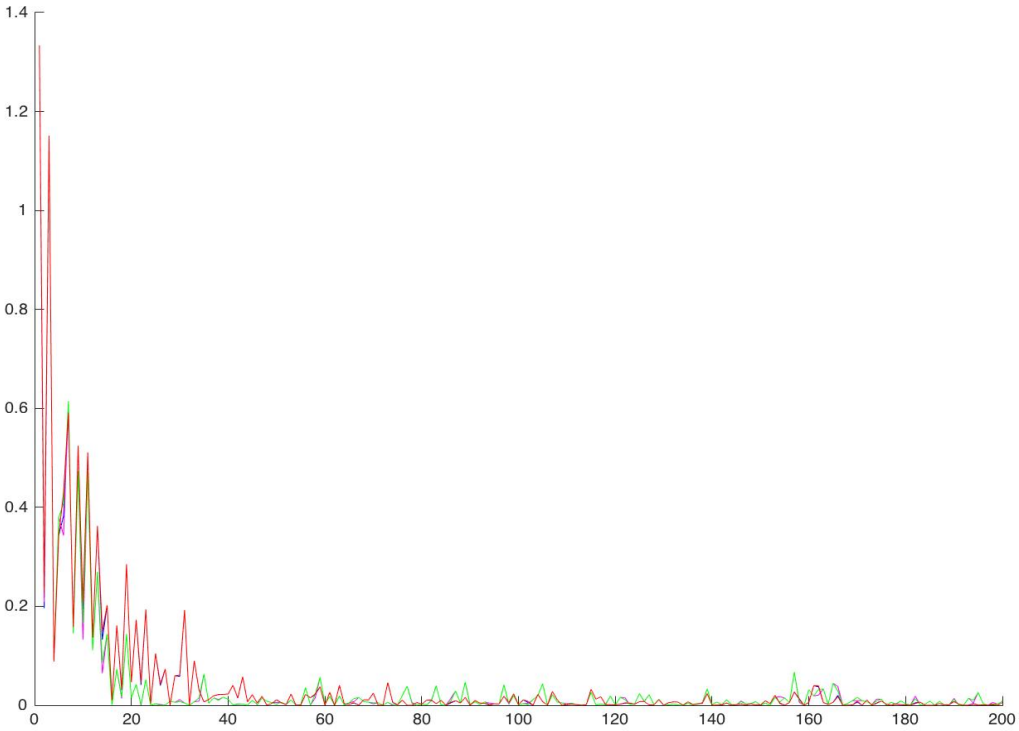

Figure 12: Error comparison of 4 combinations

Figure 12 shows the error square of the 4 combination. Green is linear interpolation with LMS algorithm, blue is step interpolation with Error Sign algorithm, magenta is step interpolation with LMS algorithm, red is linear interpolation with Error Sign algorithm. We can see that all of them recover the signal pretty well. This simple noisy input doesn't show how linear interpolation can achieve better result because all signals are scaled at the same level, so to show the difference, a better way to generate noisy signal is used.

Instead of multiplying the original signal with the same number, it's multiplying M which linearly increase from 0.4 to 3.4 for the 16 signals in a frame, 4 of them are pilot tones, after adding Gaussian noise to it, we multiplied the original and M and add white Gaussian noise again. The noisy signal is compared to the original signal in the bottom of the Figure 13.

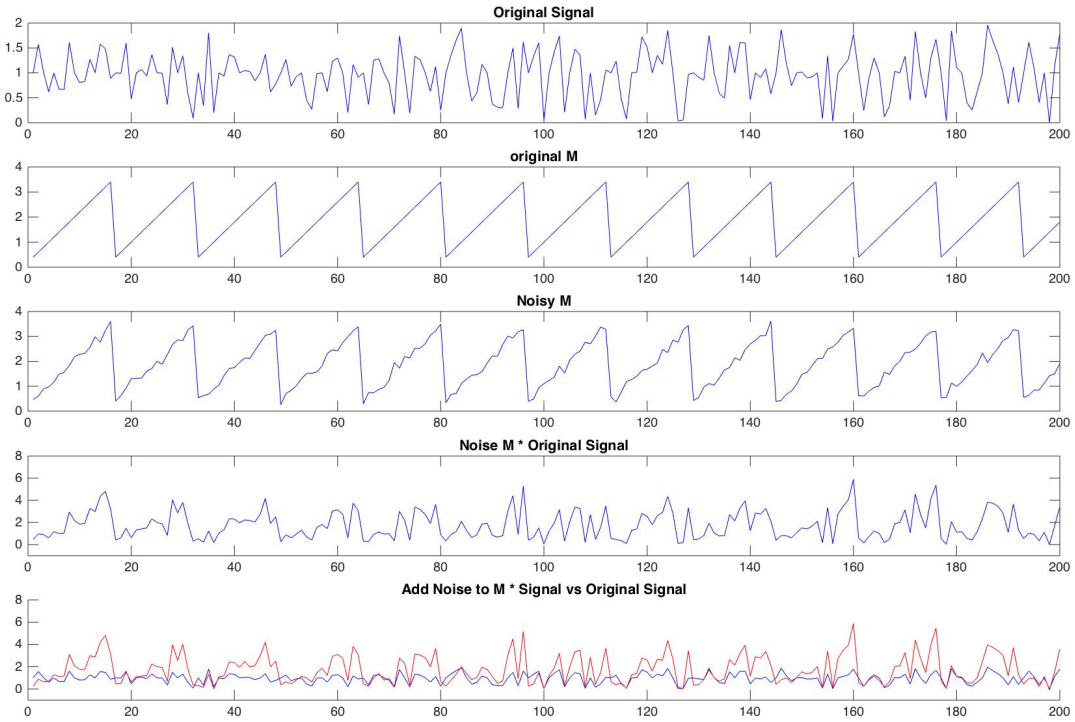

Figure 13: Generate better noisy signal to test linear interpolation.

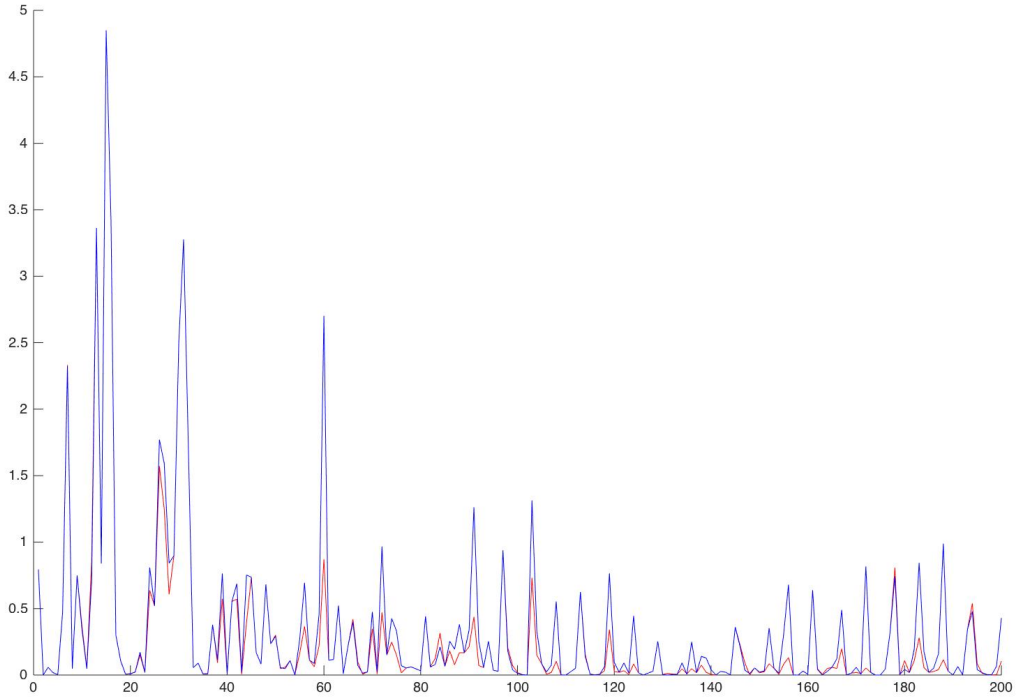

Figure 14: Signal comparison with linear interpolation and step Interpolation.

Figure 14 shows the error square of the recovered signal from the channel estimation and equalization block. Both of them are using LMS algorithm. Blue is the one using step interpolation and the red is using linear interpolation. It's shown that red is generally lower than blue, and the sum of the error adds up to 50.19 for step interpolation and only 36.61 for linear interpolation. After several tests, similar differences between these two interpolation methods proves that the linear interpolation recover the signal better.

As the outputs using different algorithm combinations are close, the design was pushed through FPGA flow to see the LUTs needed after synthesis. As shown in the following table, LUTs increases with more advanced algorithm.

Algorithm	Step / Error Sign	Step / LMS	Linear / Error Sign	Linear / LMS
LUT number	177	334	994	1153

With same interpolation method, Error Sign use about 150 less LUT than LMS algorithm. As shown previously there is no obvious difference between the performance, Error Sign algorithm will be a better choice for most of the design. While linear interpolation has a better performance, it requires a lot more LUTs/Area as it needs extra multiplication circuits.

After pushed the design through the ASIC design flow, the power and area of each combination (normal LMS, Sign LMS, step interpolation and linear interpolation) are documented as below. The LMS block includes the LMS calculation so it's power and area change depend on which algorithm (normal or Sign LMS) is used while the equalizer is applying the results from the LMS block to the input, so the power and area have big difference between using linear and step interpolation. Compare the power and area of each sub-block, the results are similar to the FPGA results as shown:

Pipeline stages	power (uW)			area (um^2)			clock (ns)	energy
	Total	Equalizer	LMS	Total	Equalizer	LMS		
Error Sign LMS with Step Interpolation								
1	5.19E+04	3.19E+04	9.28E+03	2.27E+04	1.32E+04	5.56E+03	1.272	6.60E-05
2	4.49E+04	2.59E+04	9.53E+03	2.33E+04	1.27E+04	6.52E+03	0.919	4.13E-05
3	3.99E+04	2.63E+04	5.20E+03	2.20E+04	1.27E+04	5.16E+03	0.941	3.75E-05
Error Sign LMS with Linear Interpolation								
1	1.33E+05	1.12E+05	8.80E+03	6.16E+04	5.16E+04	5.52E+03	1.982	2.64E-04
2	1.15E+05	1.02E+05	3.58E+03	5.98E+04	5.06E+04	4.08E+03	1.707	1.96E-04
3	9.13E+04	8.02E+04	3.46E+03	5.91E+04	4.97E+04	4.70E+03	1.703	1.55E-04
Normal LMS with Step Interpolation								
1	1.15E+05	3.21E+04	7.13E+04	4.05E+04	1.32E+04	2.31E+04	1.966	2.26E-04
2	8.60E+04	2.41E+04	5.24E+04	4.16E+04	1.14E+04	2.59E+04	1.417	1.22E-04
3	9.08E+04	2.72E+04	5.39E+04	4.41E+04	1.16E+04	2.81E+04	1.145	1.04E-04
Normal LMS with Linear Interpolation								
1	2.00E+05	1.13E+05	7.46E+04	8.03E+04	5.17E+04	2.40E+04	2.70	5.40E-04
2	1.47E+05	9.83E+04	3.75E+04	7.35E+04	4.87E+04	1.98E+04	1.654	2.43E-04
3	1.43E+05	1.03E+05	3.02E+04	7.53E+04	5.12E+04	1.91E+04	1.658	2.37E-04

In summary, this channel estimation and equalization generator is now capable of generating a channel estimation and equalization block with options to use normal LMS or error sign LMS algorithm, linear or step interpolation method and the number of pipeline stages included.

Algorithm	0: Normal LMS	1: Error sign LMS
Interpolation method	0: Step interpolation	1: Linear interpolation
Pipeline stage numbers	Choose from 0, 1, 2	

Chapter 2

Engineering Leadership

Team-written by Naing Ye Aung, Xiaohui (Karon) Li, Gilbert Lityo

Introduction

The goal of our Capstone Project is to develop open source hardware generators for a Software Defined Radio(SDR), specifically the baseband processing components. SDR is a radio in which the operating physical layer of the system is replaced by software. Baseband processing components serve to correct any errors the environment might have created in the message received during transmission, and decode it after correction. Hardware generator is a hardware design methodology in which the design is created on demand based on the users' specified parameters.

Industry Analysis

To understand the prospect of profitability in the industry our product fits in the best, the Software Defined Radio Industry, we adopt Porter's Five Force Analysis, where the five forces are substitutes, buyers, new entrant, rivalry, and suppliers. We will analyze how these five forces influence profitability in this industry (Porter, 2008).

The force of substitutes for our product is weak. Our primary substitute is the traditional hardware based radio. SDR poses very attractive advantages that substitutes cannot provide. Its unique ability to implement multiple current communication standards on a radio is non-existent in traditional hardware based radio devices. Secondly, SDR customers will have a sustainable system with less recurring cost every time a new communication technology update emerges. As

a result, SDR can create a reconfigurable radio system that has multimode and multiband to support different communications standards with lower overall cost.

Buyers have a lot of power in our industry. The two largest customer sectors in the current SDR industry are the military and the telecommunication infrastructure equipment companies. SDR is being deployed in tactical radios because it enables joint operations between separate troops from national and international operations even though the network communications in each country are different. For the telecommunication infrastructure, they avoid creating new infrastructure for each update. According to Mobile Experts and the Wireless Innovation Forum, tactical radio manufacturers sold approximately 200,000 SDR embedded tactical radios in 2012 and the amount has been increasing annually (Pucker and Renaudeau, 2012). The buyers possess significant power because of low switching costs between different SDR vendors. Additionally, there are a limited number of buyers in a huge market, which strengthens their negotiation power. Since SDRs support a variety of standard communication standards, the buyers can just switch to another SDR vendor that support the standards if the product of current vendor, for example, becomes too expensive.

The force of new entrants is also strong because of the low barrier to entry. As there are several small SDR design/manufacturing companies, very few of which are public, it indicates that the SDR industry does not require large financial investment. From a technical perspective, the core technology dates back to a few decades ago so the information is widely available.

Although entering the market is relatively easy, there is strong rivalry based on features such as power consumption, bandwidth, and efficient architecture. Since there are already many SDR manufacturers such as Northrop Grumman Corporation, L-3 Communication Holdings Inc., and Raytheon Co. (Marketsandmarkets.com, 2014), most of the features are already covered by one

or more companies' product. However, if our product, which is a design tool for SDR, can be used to build a new product that has a unique feature, the rivalry force would be weak.

However, the force of suppliers is weak. The major suppliers for SDR are similar to the suppliers for traditional radio, which includes component manufacturers, for example, antenna and other basic analog devices like analog digital converter (ADC), digital analog converter (DAC). Since these parts are standardized, the suppliers have weak bargaining power.

With weak forces in suppliers and substitute, but strong forces in new entrants, rivalry based on feature, and buyer, SDR may not seem to be an attractive industry to enter at the first glance. However, our marketing strategy explained in the following section will create a more favorable situation for our product.

Marketing

The SDR market has been constantly growing in the last few years. The SDR market size is predicted to reach $27.29 billion by 2020 with a Compound Annual Growth Rate(CAGR) of 12.5% from 2014 to 2020 (Marketsandmarkets.com, 2014). The market demand for SDR still remains strong and has yet to saturate more than 30 years after its initial inauguration (Clarke and Kreitzer, 2015). The huge growth is understandable because new communication technologies have been emerging year after year. According to IBIS industry reports, the revenues of both the Communication Equipment Manufacturing (Ulama, 2015) and Wireless Telecommunications Carriers (Blau, 2015) industries in the US currently stand at $33.8 billion and $248.7 billions respectively. This positive market trend and huge market size on the communication industry provides worthy reasoning to explore SDR development.

There is another great future potential market for SDR. Part of current SDR users are those who use small consumer electronic devices. Tremendous growth in the Internet of Things(IoT) market has led to various applications, for instance Smart Grid, home automation systems, and intelligent industrial system (Bushehri, 2013). IoT acts as a smart gateway to connect between multiple low-power, low-cost devices and with each other and the internet (Bushehri, 2013). All of these devices have different ranges of communications standards and interfaces, such as Bluetooth, Wi-Fi, and ZigBee. These multiple wireless standards are implemented more efficiently using SDR technology as SDR can re-configure those devices and create a gateway that can connect everything into a whole system.

Apart from the market size, we will also discuss the "4P's" of marketing: product, price, place, and promotion. Though our project is currently set for research purposes, for this analysis, we will be assuming the scenario where we apply our technology in a business setting.

Our product is an open-sourced Software Defined Radio(SDR) platform which can be customized to user's needs. It can provide SDR designers with instant hardware designs to expedite their product development by only focusing on developing the value-added components.

As an open-sourced project, our product is available for anyone who is trying to develop SDR from scratch. Users can integrate our product on their systems without paying any licensing fee. On the other hand, as the main developer of the product, we can adapt a service business model similar to that of Linux/Red Hat. We can provide services to help potential customers transition to our framework, implement, and integrate customized features not yet implemented. The major platform for support will be available through online forum, chat, or video-

conferencing. Thus, we will be able to create a direct channel without face-to-face encounter (Wenkart, 2014). Field Engineers will also be available if requested.

We recognize the lack of trust potential users may have in adopting our platform. Therefore, we aim to promote our reputation, which will be achieved through journal publications and conferences. Sponsorship deals will also be provided for renowned corporates as their subscription for our support service will accelerate the level of credibility.

Intellectual Property (IP)

The two options considered for open source licensing are Berkeley Software Distribution(BSD) and General Public License(GPL). The main difference between BSD and GPL is that any developer who modifies source code licensed under GPL is also required to license his/her work under GPL if he/she wants to distribute the product. In contrast, developers who enhance or modify a product licensed using BSD are not under such restriction, opening up business opportunities. As a result, BSD will encourage more people to improve and use this tool.

Open source is chosen due to several reasons. We are currently using publicly available design architectures to create these generators. Thus, we do not have any new algorithm/design for the SDR blocks, making them unsuitable for patents. Instead, our generator aims to help others who want to build new products to use the existing design. As we are using/implementing the existing designs, making this project open source will protect us from lawsuit. It is also beneficial to the design community as donating our idea to the public will encourage innovation by saving time/effort to design the fundamental blocks.

By building a support model around this, we can create win-win situation; by creating a lower barrier to adopt this framework, the number of SDR designers will increase which not only encourages innovation, but also generates more revenue for our business.

Conclusion

A hardware generator will be a very useful tool when designing Software Defined Radio and other hardware that is based on SDR. As for most of the projects, there is not much addition value in the baseband, however, the additional value or function need to be operated on a specific standard SDR. With the tool, Chisel, which includes the advantages of the software languages in order to make hardware design easier, using a design generator is easier than using the traditional hardware description languages. At first, the goal of our team was to finish the design of the generator for the whole receiver. After some time spent on learning the new tool Chisel and understanding the fundamentals of Digital Signal Processing, we set a more realistic goal as to finish several important blocks in the receiver including the channel estimation with equalization block.

This channel estimation and equalization generator is now capable of generating a channel estimation and equalization block with options to use LMS or Error Sign algorithm, linear or step interpolation method. From the testing results, there is not significant improvement using LMS algorithm instead of Error Sign algorithm, but linear interpolation does recover the signal better with the cost of much more LUTs / bigger area needed.

It is very important to keep in mind that our goal is not just to design a certain block with specific standard, but a generic generator which has numbers of parameters that can be set to accommodate various of given parameters. Which means that there are still more options can be added to this generator, like the position of the pilot tones. In the current design the pilot tones are placed evenly, which is comb type. Any team that is interested in the idea of hardware generator can add more options to this block or even complete the receiver generator and test the receiver as a whole.

References

Haykin, S. S., & Widrow, B. (2003). Least-mean-square adaptive filters. Hoboken, NJ: Wiley-Interscience.

COMM Tech Knowledge:
http://www.rf-mw.org/the_spectrum_analyzer_introduction_introduction.html

Chiueh, T., & Tsai, P. (2007). OFDM baseband receiver design for wireless communications. Singapore: John Wiley and Sons (Asia).

Blau, G. (2015). IBISWorld Industry Report 51721: Wireless Telecommunications Carriers in the US. Retrieved October 18, 2015 from IBISWorld database.

Bushehri, E. (2013). Future Wireless Networks Will Rely On Programmable SDRs. Electronic Design.

Clarke, B. & Kreitzer, K. (2015). How to Maximize Your Software-Defined Radio's Dynamic Range. Microwave Product Digest.

Marketsandmarkets.com. (2014). Marketsandmarkets: Software Defined Radio (SDR) Market for Communication by Component (FPGA, DSP, GPP, PSOC, Amplifier, and Software), Application (Military, Telecommunication, Short Range, Positioning, Transportation, and Public Safety), & Geography - Analysis & Forecast to 2014 – 2020. Retrieved from http://www.marketsandmarkets.com/Market-Reports/software-defined-radio-communication-market-11265833.html.

Porter, M (2008). The Five Competitive Forces That Shape Strategy. Harvard Business Review.

Pucker, L & Renaudeau, D. (2012). Conquering SDR tactical radio market challenges. Military Embedded Systems.

Ulama, D. (2015). IBISWorld Industry Report 33422: Communication Equipment Manufacturing in the US. Retrieved October 18, 2015 from IBISWorld database.

Wenkart, M (2014). The Marketing Bible. Norderstedt: on Demand.