12-2015

Novel Software Defined Radio Architecture with Graphics Processor Acceleration

Lalith Narasimhan
Western Michigan University, lalith.narasimhan@gmail.com

NOVEL SOFTWARE DEFINED RADIO ARCHITECTURE WITH GRAPHICS PROCESSOR ACCELERATION

by

Lalith Narasimhan

A dissertation submitted to the Graduate College
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
Electrical and Computer Engineering
Western Michigan University
December 2015

Doctoral Committee:

Bradley J. Bazuin, Ph.D., Chair
Janos L. Grantner, Ph.D.
John Kapenga, Ph.D.

NOVEL SOFTWARE DEFINED RADIO ARCHITECTURE WITH GRAPHICS PROCESSOR ACCELERATION

Lalith Narasimhan, Ph.D.

Western Michigan University, 2015

Wireless has become one of the most pervasive core technologies in the modern world. Demand for faster data rates, improved spectrum efficiency, higher system access capacity, seamless protocol integration, improved security and robustness under varying channel environments has led to the resurgence of programmable software defined radio (SDR) as an alternative to traditional ASIC based radios. Future SDR implementations will need support for multiple standards on platforms with multi-Gb/s connectivity, parallel processing and spectrum sensing capabilities. This dissertation implemented key technologies of importance in addressing these issues namely development of cost effective multi-mode reconfigurable SDR and providing a framework to map sequential wireless communication algorithms to the parallel domain.

Initially, a novel software defined radio platform using commercial off-the-shelf components was successfully developed. This hybrid platform consists of an USRP N210 device performing the role of an RF front end, an NVIDIA Quadro 600 GPU functioning as the parallel computing node, and a commodity PC with PCIe backplane as the high-speed interconnect. Validation of the architectural concepts was demonstrated through real-world applications on the GNURadio software. Performance analysis and benefits of the proposed architecture over other custom solutions was also demonstrated.

In the second project, we demonstrate an important application of GPU technology to SDR systems, namely the polyphase channelizer. The proposed channelizer architecture exploits block and thread level processing in the GPU and delivers high throughput, arbitrary resampling of multiple channels. These characteristics make it attractive for a variety of communication receiver algorithms.

Finally the third project will deal with critical high data rate dataflow between radio peripheral devices and parallel processing resources. Software routines for this project were written in C++ and are based on the UHD code from Ettus Research. In addition to enabling transfer of data, the software is also responsible for configuring the USRP devices. Analysis of performance metrics and dataflow bottlenecks, show the proposed architecture is capable of meeting the demanding requirements of current wireless standards.

ACKNOWLEDGEMENTS

I would first like to express my sincere gratitude to my advisor and committee chair Dr. Bradley J. Bazuin for the continuous support of my Ph.D. Study and related research. His patience, motivation and immense knowledge guided me though the highs and lows of my research and in the completion of this dissertation. I could not have imagined having a better advisor and mentor for my Ph.D. study. I would also like to thank Dr. Janos Grantner and Dr. John Kapenga for agreeing to be on my committee and providing me their insightful comments and encouragement in all aspects related to this work.

I would like to express my appreciation to the faculty and staff of the Electrical and Computer Engineering department for all the assistance that they have provided me throughout my years at Western Michigan University. I am also particularly grateful to Dr. Steven M. Durbin for his confidence in my abilities and continued support during the final stages of my dissertation.

I thank my fellow research colleagues for the stimulating discussions and their professional assistance in writing this dissertation. I am deeply indebted to Ravishankar Thangavelu, Kartik Ramakrishnan, Sarawana Vijayakumar, Guruprasath Narayanan, Vijay Balakrishnan, Balamurugan Chandrasekharan, Rathna Kalianan, Shankaar Thiagarajan and Vijay Meghanathan for their support and friendship throughout my graduate studies.

Acknowledgments—Continued

I would like to thank my parents and my sister for all the sacrifices they have made over the years in order to provide me with the finest education possible. Special thanks to my wonderful nieces, Sanjana and Shruthi Maheve for the support and motivation you have provided me to reach this milestone. This dissertation would not have been possible without the unwavering support and encouragement of my family.

Lalith Narasimhan

TABLE OF CONTENTS

Table of Contents—continued

Table of Contents—continued

Table of Contents—continued

LIST OF TABLES

LIST OF FIGURES

List of Figures—continued

List of Figures—continued

CHAPTER I

INTRODUCTION

The twentieth century has seen significant scientific and technological developments, of which wireless communication is in many ways unique and one of the most important. From the first transatlantic transmission by Marconi in 1901 [1, 2] to the current smartphone revolution, wireless has become one of the most pervasive technology with a variety of applications ranging from third/fourth generation (3G/4G) cellular devices, WiFi, vehicle-to-vehicle (V2V) systems, sensor networks, and near-field communications (NFC) [3, 4]. Although initially developed to provide only voice based services, mobile devices in recent years have seen a massive proliferation of data and internet services, enabling a wide range of computing and multimedia applications ranging from navigation and search to mobile video streaming. According to industry reports [5, 6] traffic from mobile and wireless devices will exceed that from personal computers (PCs) by the year 2016. In the future, delivery of internet based services will not be limited to cellular devices. In fact various estimates [7-9] predict that by 2020, more than 30 billion devices will be wirelessly connected to the internet of things (IoT); a network of physical entities or things embedded with electronics, software and sensors capable of providing automation, enhanced connectivity and services.

This paradigm shift towards mobile platforms has serious implications for radio hardware platforms, physical layer technologies, wireless protocols and standards, as well as delivery of internet based applications on fixed devices like PCs and televisions. Delivery of internet based services wirelessly will require faster data rates, improved spectrum efficiency, higher system access capacity, seamless protocol integration, improved security, robustness under varying channel environments, and many others. Progressive demand for higher data rate is best illustrated by the Edholm's law of bandwidth (see Figure 1).

According to Edholm's law [10], telecommunication data rates follow an exponential rate of growth in a manner reminiscent of Moore's law. Figure 1 shows the increase in access speeds over the past 20 years across three telecommunications category; wireline, nomadic and wireless. It can be seen that three categories march almost in lock step; the slower rates trailing the faster ones by a predictable time lag. For example, it can be seen that in the last decade, cellular link speeds have increased from 2 Mb/s with 3G systems to about 100 Mb/s with 4G. In the same period, WiFi speeds have increased from 54 Mb/s (802.11g) to about 1.3 Gb/s (802.11ac) and wired Ethernet from 10 Gb/s to about 100 Gb/s.

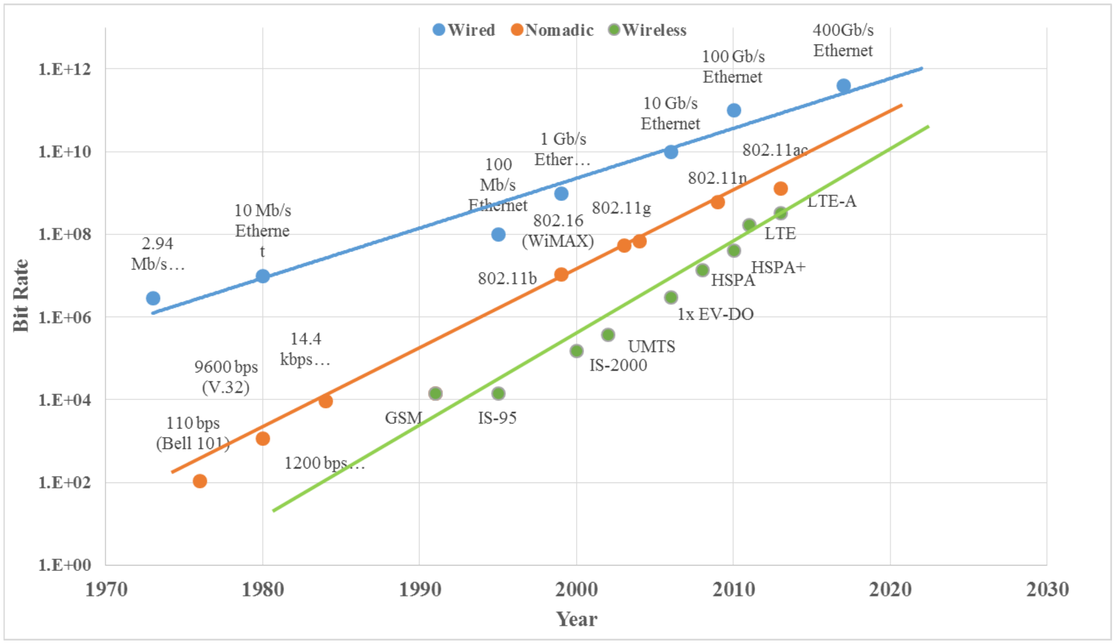


Figure 1. Edholm's law of bandwidth

It is evident that we are witnessing a historic increase in wireless bit rate and system capacity to levels needed to support internet and media rich applications. Not surprisingly, much of the research and development in the past decade is aimed at addressing these issues. Of the various radio technology ideas and innovations that have been proposed in the last 20 years (see Figure 2), key areas of strategic importance are:

- Wireless Communication Physical Layer Formats and Algorithms
- Wireless Networking Services and Protocols
- Radio Technologies Platforms

Wireless Communication Physical Layer Formats and Algorithms: Historically, there have been a wide range of physical layer wireless signal formats requiring a range of signal processing algorithms and techniques to transmit and receive computer and voice data. While many such legacy systems currently exist and will continue to exist, advanced systems are consistently moving toward higher bandwidth and capacity formats requiring significant digital signal processing. Current high-speed cellular and wireless local area network (WLAN) standards implement wideband orthogonal frequency division multiplexing (OFDM), which offers higher spectral efficiency and performance [11]. In the future, it is expected that both cellular and WiFi standards will continue along the OFDM path for physical layer with several enhancements like dynamic spectrum access (DSA), noncontiguous OFDM (NC-OFDM) [12], spatial beamforming with multiple-input-multiple-output (MIMO) antennas [13], co-operative communication and many other wireless algorithms [4, 14].

Wireless Networking Services and Protocols: Both wired and wireless communication of voice and computer data have evolved to contain well defined structural elements that support broader network communication and specific user services that are not required in a simple point-to-point link. Typical service features include; authentication, link encryption, user mobility and seamless intersystem handover (3G, 4G, Wi-Fi) [15, 16]. The convergence of cellular and internet applications has pushed the implementation of these services through enhanced

Internet Protocols (IP) rather than traditional telephony signaling protocols [15]. Wireless networking research in this field has therefore been geared towards designs and techniques like multihoming, infrastructureless (or “ad hoc”) [17] and multihop mesh networks [18], delay tolerant routing, quality of service, and mobile content caching [19] aimed at integration of a multiplicity of systems, services and protocols [20].

Radio Technologies Platforms: The digital revolution has ushered in a variety of radio technology standards to meet the needs of a diverse range of applications from personal area networks (PAN) to wide area networks (WAN). The physical layer (PHY) and the media access control (MAC) layers of these standards involve considerable signal processing complexity requiring custom Application Specific Integrated Circuit (ASICs) to meet these demands. However due to increasing chip development costs and long development cycles, there is a demand for flexible, low cost and dynamically reprogrammable platform capable of supporting a wide range of standards. Software defined radios (SDR) having many of the features desired above is often viewed as the most appropriate solution for future radio technology platforms. Current SDR platforms still use field-programmable gate arrays (FPGA) and are extremely power hungry; suitable for only base station type implementations. Further work is required to develop software radios on low-cost chipsets (CPU/GPU) so as to have a greater impact on the wireless industry

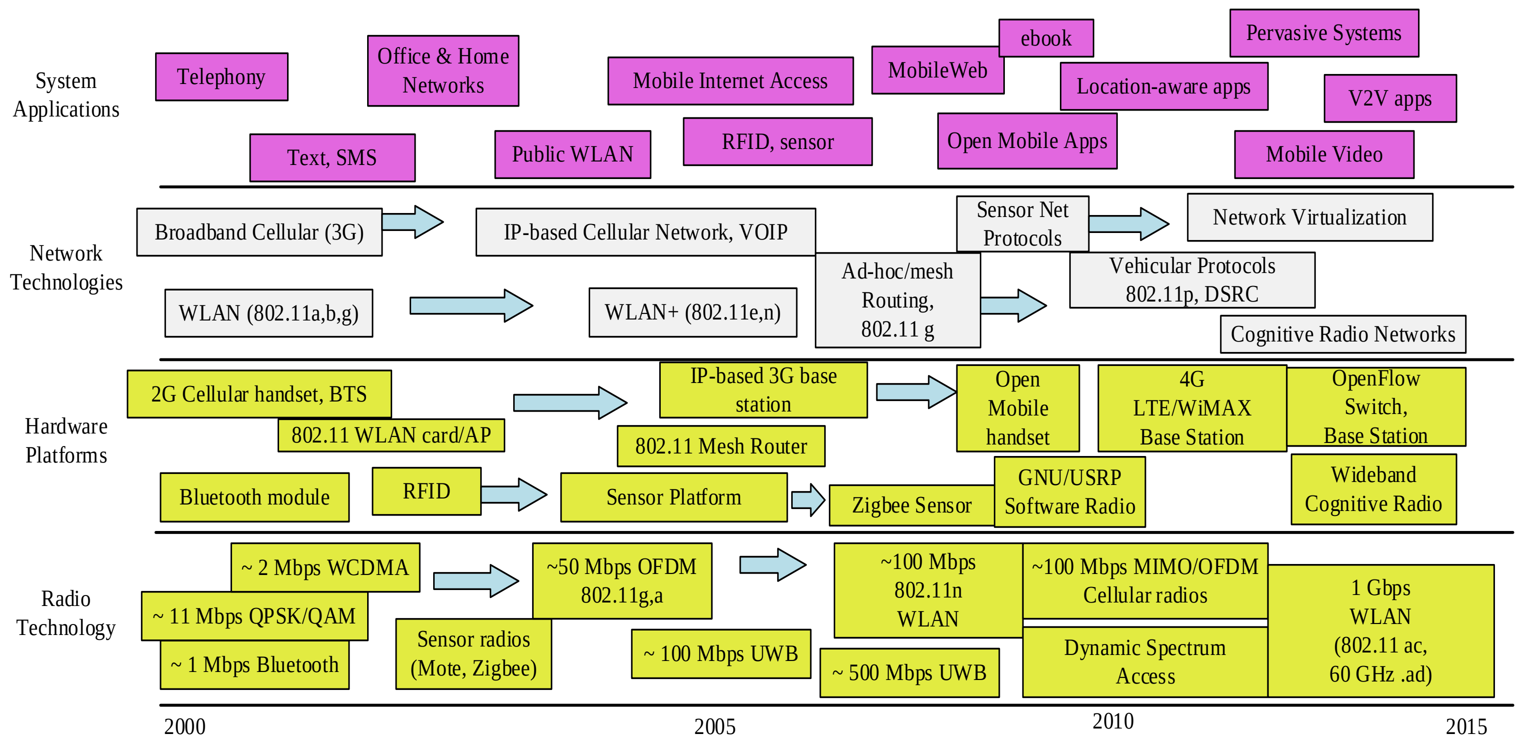


Figure 2. Wireless technology roadmap [6]

1.1 Contribution

In the foreseeable future, it is likely that many communication devices are going to be supported by or even primarily based on SDR technologies. This thesis presents a novel SDR architecture that combines advances in radio peripherals, personal computer bus and network wired interconnections, and PC based super-computer processing elements. The prototype system, based on commodity SDR and PC components, will provide a low-cost, scalable architecture that can readily support the widest range of frequencies and bandwidths while potentially providing base station level processing capability for the widest range of physical layer wireless signal formats. As envisioned, it could function as a universal wireless base station, replacing numerous custom platforms, or as a new generation of universal wireless access points providing services to any and all local wireless devices simultaneous. To achieve the required wireless capacity, digitized frequency signal data rates, and computation capability, the prototype architecture uses; one or more universal software radio peripherals (USRP) from Ettus Research, a PC with a multilane PCI Express (PCIe) backplane, a graphical processing units (GPU), and a multicore, multithreaded CPU.

The proposed architecture is a hybrid system involving commercial-off-the-shelf (COTS) components that are familiar to a range of users and developers in super-computer, parallel computing, high-end computer gaming, computer architecture, and cellular telephone disciplines. As a result, this dissertation and the research activities performed span a broad range of topics in computer science, electrical engineering and computer engineering all focused on realizing a scalable, universal base station or universal access point capable SDR. The physical configuration for a prototype system might appear as shown in Figure 3.

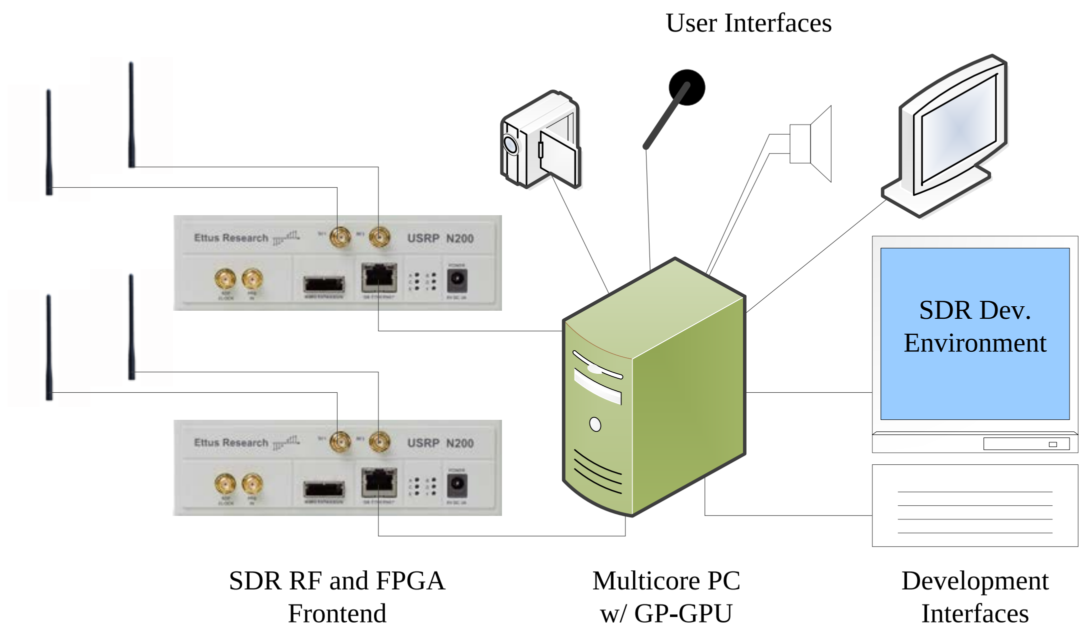


Figure 3. Hybrid SDR system prototype architecture hardware components

In the first project we describe one possible embodiment of the architecture and described a flexible demonstrator platform that can be used to validate concepts and can be used to conduct research on the physical layer of wireless communication. The hardware demonstration platform consists of three components: USRP with multiple daughtercards (SBX, Basic RX/TX), NVIDIA Quadro 600 GPU, and a commodity personal computer (PC) with a general purpose processor. The entire system is highly modular and built around the PCI Express backplane of the PC. The high speed PCIe interconnect provides the flexibility to scale the processing capabilities of the system with minimum system modification. The processor on the PC will be used to load the radio modem firmware on the GPU. This will make the GPU function as a SDR modem. The processor will also be responsible for exchanging source data with the GPU to modulate/demodulate, and perform other signal processing tasks.

A parallel signal processing software environment for the SDR and enumeration of candidate algorithms in a typical wireless communication system will be the focus of the second project. Modem code to be executed on the GPU is written in the C-language using common unified device architecture (CUDA) environment provided by NVIDIA. Initialization and control code will be also be written in C and executed on the PC. This code is compiled using a general purpose compiler such as Microsoft Visual Studio and the open source GNU C compiler (GCC). A number of difficulties exist in employing GPU for SDR applications; including architectural complexity, new programming language, and stylized parallelism. Therefore to assist the future SDR developer in prototyping parallel processed communications algorithms on the GPU, an efficient polyphase filter bank channelizer implementation on the GPU is demonstrated. By examining performance metrics like floating point operations, peak performance, occupancy, execution time and other parameters, we intend to show the suitability of the GPU for SDR applications. The polyphase implementation will also expose the programmability and performance requirements for the proposed system architecture.

Finally the third project will deal with critical high data rate dataflow between radio peripheral devices and parallel processing resources. For the demonstration system, the USRP exchanges digital base-band signals with the GPU consisting of the digitized, complex receive/transmit RF signal from/to the air. This is accomplished through a Gigabit Ethernet (GbE) interface on the USRP to the PC and the multilane PCI Express backplane in the PC, along with software routines in C and low level APIs. The SBX daughtercard in the USRP is capable of 40MHz analog bandwidth, and provides 16-bit samples. Hence the useable bandwidth is about 20MHz. All the software routines and real-time data interfaces should be capable of handling about 800Mb/s including overhead.

1.2 Outline

The need to support many increasingly complex wireless algorithms and standards will make use of programmable systems for these protocols inevitable. The techniques proposed in this thesis are relevant in designing viable solutions for software defined radio systems. The remainder of this dissertation is organized as follows. Chapter 2 will provide a detailed overview of software defined radios. In addition, a review of the current multicore SDR systems and platforms will be provided. The architecture of the proposed SDR, its goals and specifications are discussed in chapter 3. Incorporation of real-time parallel signal processing for communication on the GPU is the most important component of this research. Chapter 4 discusses internal architecture of CUDA capable device, programming model and tools required to develop software targeting GPU. Implementation details, trade-offs and results of polyphase filter implementation are also provided in this chapter. Dataflow between primarily hardware and software based processing components in the SDR architecture are detailed in chapter 5. Chapter 6 summarizes the contributions of this work and recommends extensions and directions for future research.

CHAPTER II

SOFTWARE DEFINED RADIO BACKGROUND

2.1 History of Radio – How it all began?

Pre-industrial age, wireless transmission/reception was done over line-of-sight distances (later extended using telescopes) using smoke signals, torch signaling, flashing mirrors, flares and semaphore flags [21].Complex messages were generally relayed using an elaborate set of the above rudimentary signals. The development of what we currently consider wireless communications stems from the works of Oersted, Faraday, Gauss, Maxwell, and Hertz [22]. In 1820, Oersted discovered that a current carrying conductor produces an electromagnetic field [23, 24]. Michael Faraday discovered electromagnetic induction in 1831 when he showed that a time varying magnetic field produces an electric current across the conductor [23]. In 1864, James Maxwell formulated the classical theory of electromagnetic radiation, bringing together for the first time electricity, magnetism and light as manifestations of the same phenomenon. Maxwell was able to show in theoretical and mathematical form that electromagnetic waves could propagate through free space [23-25]. Between 1886 and 1889 Heinrich Hertz conducted a series of experiments that proved conclusively Maxwell's theory of electromagnetic radiation [23, 24, and 26]. This discovery eventually led to the development of commercial Hertzian wave based wireless telegraphy, radio and television. Italian engineer Guglielmo Marconi is often credited as the inventor of radio; having first transmitted and received a coded message at a distance of 1.75 miles near his home in Italy [3, 23]. The Indian physicist Jagadish Chandra Bose also independently generated and detected wireless signals of 6mm wavelength [24, 27, and 28]. These two achievements ushered in the era of modern wireless communications. Soon commercial radio and television broadcasting stations were setup in the US and the rest of the world. However, it was the WWII years that saw rapid and forced developments in nearly all areas of engineering and technology including communications. The twentieth century saw other significant advancements in the field of communication but none greater than the works of Claude Shannon. In a landmark paper written at Bell Labs in 1948 [29], Shannon defined in mathematical terms what information is, how it can be transmitted over a noisy channel and what are the capacity limits of a channel. Today all modes communication rely on Shannon's contribution and it is for this reason he is called "the father of information theory".

2.2 Evolution of Radio Systems

The basic functionality of any communication system is to convey information from a source to one or more destinations. At its heart, the architecture of a communication system consists of three basic parts; transmitter, channel and the receiver as shown in Figure 4. The information from an input source, either voice, picture, or plain text, is applied to an input transducer that converts it into electrical signals suitable for transmission.

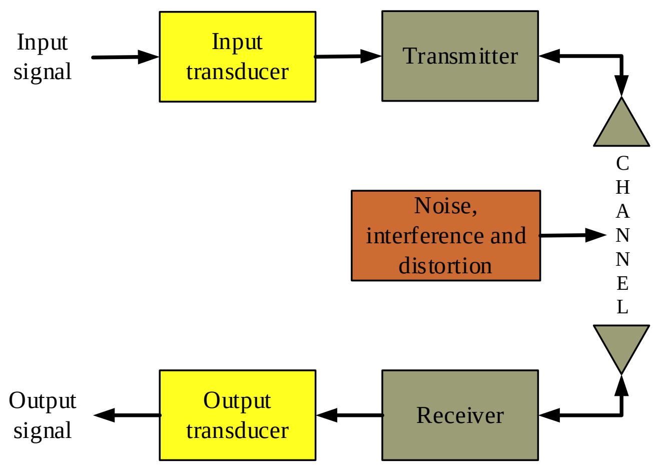


Figure 4. Functional block diagram of a communication system

At the transmitter the electrical signals are converted to a form suitable for transmission at the appropriate frequency, by a process called modulation. Modulation normally involves using the message (modulating) signal to alter the characteristics of a sinusoidal carrier signal. There are two different types of communication; analog and digital, depending on the type of the modulating signal. Amplitude, frequency, phase modulation (AM, FM, PM) are examples of analog modulation and amplitude, frequency, phase shift keying (ASK, FSK, PSK) are examples of digital modulation.

The communication channel is the physical medium through which signals from the transmitter are sent to the receiver. In the case of wireless systems, this is usually the atmosphere. The channel acts as filter degrading the fidelity of the transmitted signal by introducing random noise and distortion. The job of the system designer is to use statistical models and empirical data to overcome the effects of the channel. The objective of the receiver is to recover the original message signal by a process called demodulation. Depending on the type of modulation employed the receiver may have to perform additional tasks like synchronization, error correction, amplification and noise suppression.

2.2.1 Analog Communication System

Early communication systems were mostly analog in nature. This was hardly surprising given the analog nature of message signals (voice, music, etc.) and the communication medium, and the non-availability of advanced electronics. In analog systems, a modulator systematically alters the carrier wave in correspondence with the variations of the modulating message signal. Three different types of analog modulation systems exist; amplitude modulation (AM) where the amplitude of the carrier is altered, frequency modulation (FM) where the frequency of the carrier is modified and phase modulation (PM) where the phase of the carrier is varied. AM and FM broadcasting grew rapidly across the United States and around the world in the early part of the twentieth century [22, 30].

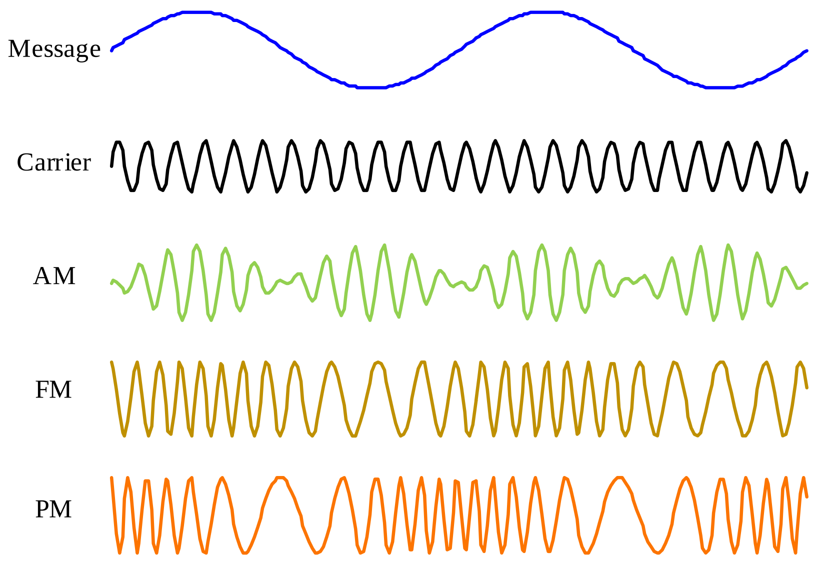


Figure 5. Analog modulation waveforms

The transistor radio is a prime example of an evolutionary analog device that was revolutionary in providing to a mass market a small, portable, hand held devices for receiving transmitted AM and FM radio waves. With billions produced, they are the most popular wireless communication device to date [30].

Television and even the first generation (1G) cellular telephone systems were based on analog modulation techniques. While analog communication was a natural option, it is significantly less prevalent and may in time become obsolete. Cellular phone systems from second generation (2G) onwards are digital, high power TV stations across the world have also migrated or are in the process of migrating to digital format [31]. Economic and political factors prevent radio and television broadcasting to migrate towards current generation technologies, but transition to digital formats is inevitable. The presence of noise in certain analog systems, coupled with complex detection/demodulation requirements and current device technology has meant that even the early analog broadcasts systems, such as AM and FM, have mostly transitioned to digital techniques in their system chain.

2.2.2 All-Digital "Analog" Communication System

Up to this point, we have described a system wherein both the message signal and the modulated waveform is a continuous time-varying analog signal. There are a number of reasons as to why modern communication systems are going digital. The principal advantage being signal fidelity and the ease with which digital signals can be regenerated compared to analog signals. Digital circuits operate in one of two states—on or off. Thus any interference or distortion acting on the signal must be large enough to change the operating point from one to another. If not, the signal can be detected easily by simple amplification. In contrast, analog systems slowly degrade the signal of interest as noise increases at every step. When digital detection with proper error correction and encoding is employed, the original signal is restored at each step prior to the next stage. With advances in digital signal processing and digital computers, it is now possible to implement complex algorithms more economically in the digital domain than in analog.

While the goal is to transmit bits in digital communication, it is essential to note that the physical layer over which these bits are sent is still analog. Thus, mixed signal design (analog/digital) still plays a crucial role in modern wireless communication systems. Design of mixers, amplifiers, filters, antennas, analog to digital (ADC) and digital to analog (DAC) in the analog domain is vital for successful development of any communication system. Although the classical definition of analog and digital communication [22, 32] is based on the type of source signal and the modulation format, there exist systems that span this boundary. For example, early computer modems [33] transmitted digital information in the form of text, computer data over telephone lines using analog modulation. As mentioned earlier, modern commercial FM broadcasts use digital audio and processing before the signals are converted to analog using DAC and then frequency modulated.

Limitations in the ADC/DAC technology meant that first generation systems, as depicted in Figure 3, sampled at baseband symbol rates. So all rate conversion (baseband to IF and then to RF), filtering and amplification is performed in the analog domain. However as converter technology advanced, the ADC/DAC blocks moved further up the RF chain, as shown in Figure 4. This allowed the receiver to acquire a large segment of the input bandwidth and facilitated signal processing tasks to be performed using DSP algorithms. The high sampling rates also allowed multirate techniques to be employed, relaxing the analog filtering requirements and incorporating digital filtering techniques. Digitization also allowed filters and processing steps to become identical and repeatable, with little or no effect due to analog component variations.

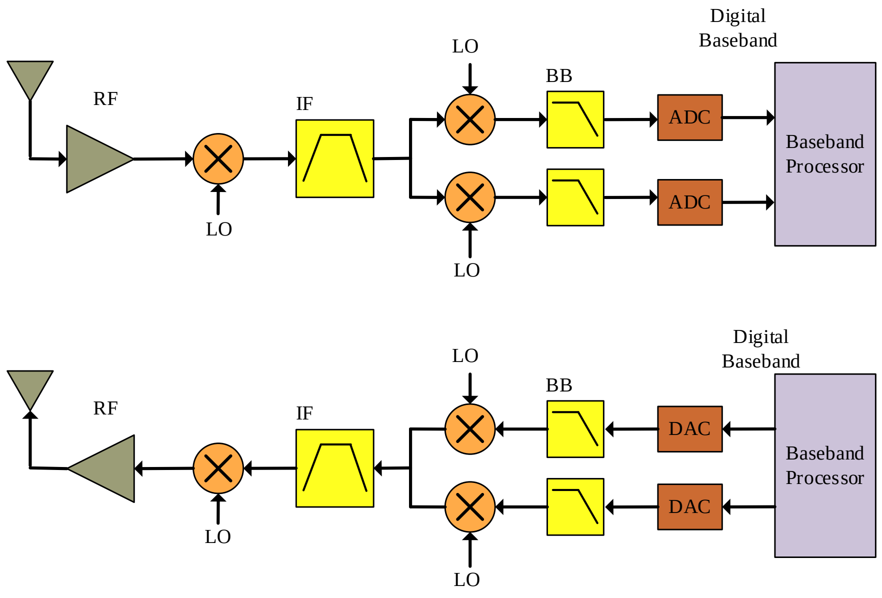


Figure 6. First generation systems sampling at baseband [34]

The IF sampling system also supports the digital processing of multiple simultaneous frequency signals from the input bandwidth. This is performed by replicating the digital processing electronics, distributing the ADC input to as many

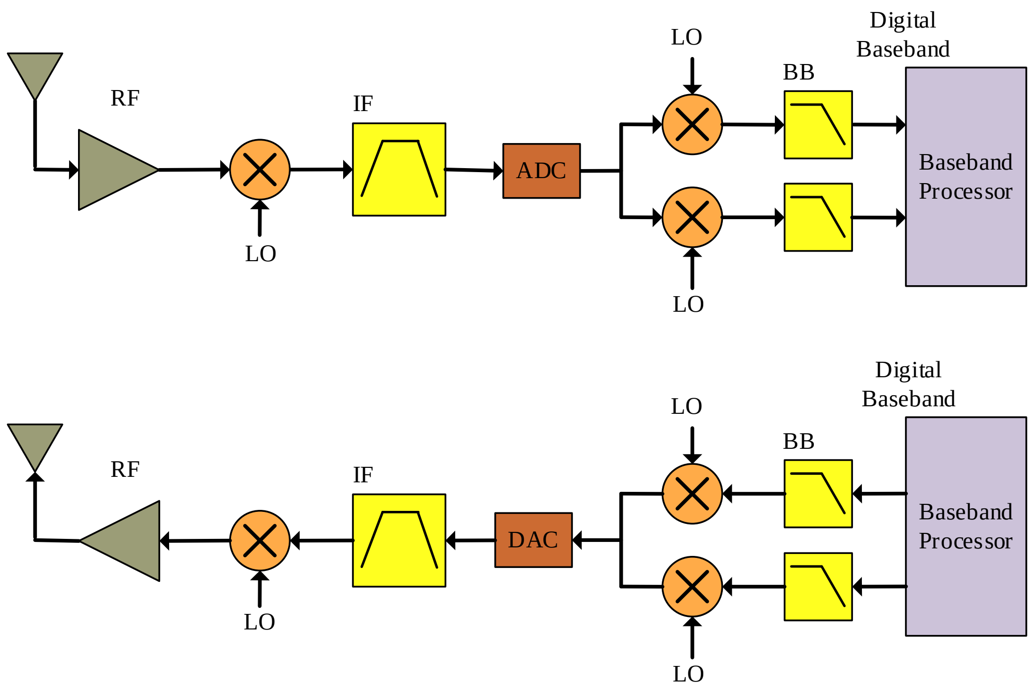


Figure 7. Second generation systems sampled at IF [34]

digital baseband receiver processing chains as desired and summing the outputs of multiple digital baseband transmitting chains prior to the DAC. With this architecture, one set of analog elements supports multiple sets of digital elements.

2.2.3 Digital Communication Systems and Techniques

In the past two decades, digital communication systems (DCS) performing ever more complex digital signal processing operations have become increasingly attractive because of the phenomenal growth of the cellular telephone market and wireless services. These systems offer processing options and flexibilities not available in single channel, narrowband analog transmissions. The distinguishing characteristic of a digital communication system is that in a finite interval, it sends a waveform from a finite set of possible waveforms, in contrast to an analog system that sends one possibly continuously varying waveform from an infinite set of waveforms with infinite resolution in the presence of noise with similar characteristics.

Figure 8 illustrates the functional diagram and signal flow of an advanced digital communication system [32]. The information source can be either an analog signal, such as audio or video signal, or a digital signal, such as output from a computer. This input is then converted to binary digits (bits); the bits are then grouped to form message symbols ($m_i, where\ i = 1,2, \ldots, M$). If $M = 2$, the message symbol m_i is binary. Even though definition of M-ary includes binary, it generally refers to those cases where $M > 2$. Although in the simplest form, source encoding is a direct conversion of the message to binary digits, the process may involve additional steps to perform this conversion efficiently.

The purpose of the channel encoder is to introduce some redundancy in the binary information sequence so as to allow the receiver to mitigate the effects of noise and interference introduced by the channel. The primary purpose of the pulse

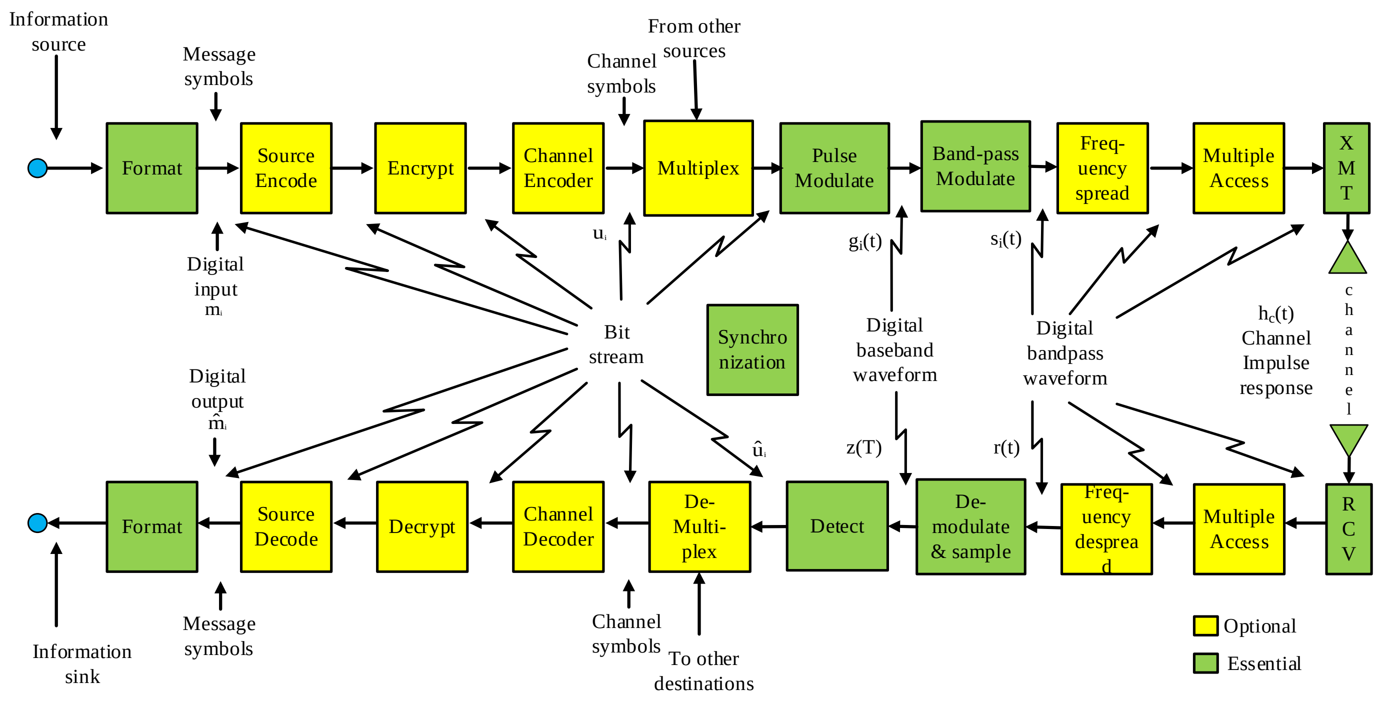


Figure 8. Block diagram of a typical digital communication system [32]

modulation block (also called digital modulator) is to transform each symbol from a binary representation to a baseband waveform. Suppose coded information is to be transmitted one bit at a time, the pulse modulator block may simply map the binary digit 0 into a waveform $g_0(t)$ and the binary digit 1 into $g_1(t)$. Alternatively, b coded information bits may be represented by using $M = 2^b$ distinct waveforms $g_i(t), where\ i = 0,1, \dots M - 1$, one for each of the 2^b possible b-bit sequences. The pulse modulation block also includes filters to contain the spectral content of the pulses within the transmission bandwidth. The transmission medium does not support propagation of pulse-like waveforms, so the next important step is bandpass modulation. This process involves translation of the baseband waveform $g_i(t)$ to a bandpass waveform $s_i(t)$, using a high frequency carrier wave. In some cases, this frequency translation is performed in stages through the use of intermediate frequencies (IF). The transmitter portion (XMT) consists of an antenna and a high-power amplifier.

The receiver portion consists of an antenna and a low-noise amplifier. The front end and/or the demodulator provides frequency down-conversion of the bandpass waveform $r(t)$. Unwanted high-frequency terms resulting from the down conversion process are removed by filters along the receive chain. Other filters like equalizers are also used to reverse any degrading effects of the channel on the signal. The job of the detector is to make an estimate of the channel symbol $\hat{u}_i$ or an estimate of the message symbol $\hat{m}_i$ in the absence of channel coding. As a final step, when an analog output is required the source decoder/formatting block performs the necessary conversion. It should be noted that the blocks shown in Figure 5 represent one possible arrangement; however not all the blocks may be present or the blocks shown may be implemented in a different order, depending on the wireless application.

2.3 Need for Software Radio

The pace of advance in the digital cellular market has been a major catalyst to the emergence of new standards, protocols and technologies in field of wireless communications. Early cellular systems offered support for only limited number of concurrent users, services and standards. With the explosion in demand for cellular voice and data services [35], numerous frequency bands have become available [36], various signal formats have evolved, and even more content like streaming audio, streaming video are being offered to smart phone users. User demand for seamless global coverage and consistent quality of service (QOS) has pushed the development of multi-mode radios that are capable of switching between multiple frequency bands and standards like CDMA, GSM, LTE and Wi-Fi. Current techniques of using dedicated hardware and air interfaces for each of these standards is a challenge for cellular network service providers and their existing base station infrastructure. The present day practice of periodic, expensive infrastructure upgrade is likely not sustainable in the long run. In addition, both network service providers and cellular phone equipment manufacturers are constantly looking at next generation technologies with higher capacity and complexity, in order to provide greater bandwidth and services to the user. Soaring chip development cost and shorter product development cycle have heralded software defined radio as a potential solution,-providing reconfigurable transceivers with the ability to adapt to different regional radio formats and interfaces [37]. In fact, current smart phones have incorporated numerous software defined radio processing techniques [38], leading the way for other market segments; including, advancements in research and development prototypes, base stations and potential smart pico-cells for both business and home users.

2.4 What is Software Radio?

The term ‘Software Radio’ was coined in the early nineties by Joseph Mitola III in a paper on radio architectures at the National Telesystems Conference, New York, in May 1992 [39]. There is no consensus on the definition, scope and requirements of a software defined radio system but it can considered as one in which all or part of the physical layer components are implemented on a programmable or reconfigurable platform.

Both modulation and demodulation is performed in software and thus the radio is able to support a broad range of frequencies and functions, either concurrently or based on unique software downloads. The idea of implementing radio functions in software rather than hardware is not something new. The origins of software radio lies in the realm of military communications. In the 1970s the US Air Force wanted to improve its avionics by integrating functions into common programming modules, allowing the addition of new features and capabilities [40]. Lacking necessary processing capabilities, this technology was limited to only few military applications.

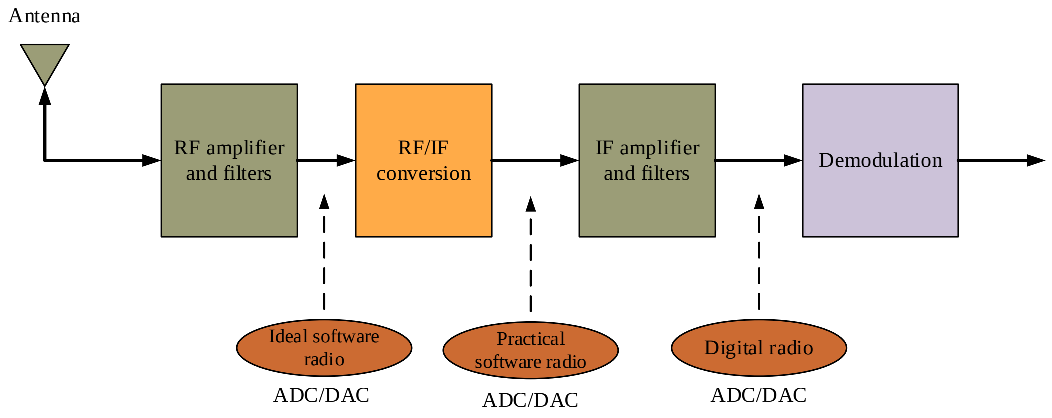


Figure 9. Simple software defined radio block diagram with possible ADC/DAC locations

Throughout the 90’s and the early part of the new millennium, progress and deployment of SDR based systems was minimal owing to the limited processing capabilities of FPGAs, CPUs and DSP devices existing at that time. The prohibitive cost of high-end processing solutions also prevented software defined radios from being viewed as commercially viable. However more recently, interest in SDR has been driven by three main factors; soaring cost in the development of chips used for radio systems, diversified market demand for different or even multiple wireless standards, and a faster development cycle due to the rapid evolution of wireless standards [37]. Technological advancements have also supported interest; including, increased density and lower power consumption of CPUs and DSP devices, high volume demand for WiFi and cellular telephony devices supporting IoT, increased clock rates and bit precision of ADC/DAC, and advancements in hybrid, mixed signal devices. All these factors combine to make SDR solution very attractive.

2.5 SDR Architecture

The functional blocks of a software defined radio system are identical to the ones found in any digital communication system [41, 42]. However, SDR places new requirements on these components in order to support multiple frequency bands, multiple services and rapidly reconfigurable operation needed for supporting different wireless standards. A model for an advanced software radio is shown in Figure 10. At the receiver, a smart antenna provides gain and directional characteristics to minimize interference, and noise. Antenna’s here may refer to antenna arrays (adaptive and/or

MIMO) with signal processing capabilities used to identify and exploit spatial signal characteristics and signatures. On the transmitter side, the smart antennas may have capability to perform beamforming, providing spatial diversity for greater system capacity. For maximum flexibility, the boundary of digital processing should be moved as close as possible to the antenna. An ideal software radio is one in which the analog-to-digital (ADC) and digital-to-analog conversion (DAC) is performed at the antenna. This would allow all operations to be performed in software. However, limitations in the bandwidth and the sampling rate specifications of the converters suggest that such an architecture is not feasible in the present and foreseeable future.

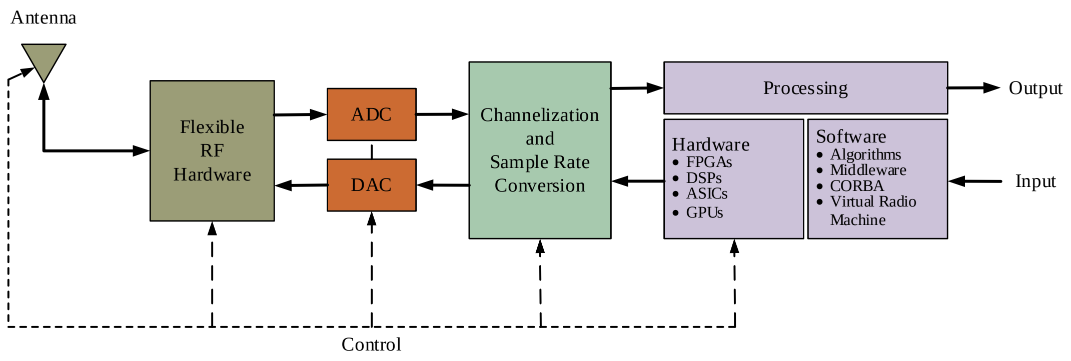


Figure 10. Model of a software radio [42]

In current SDR architectures digitization is often performed in the intermediate frequency (IF) domain. RF hardware contains low noise amplifiers (LNA) and local oscillators to mix the desired signal to IF through either direct-conversion or superheterodyne principles. The type of RF mixer and the number of stages depends on the type of applications being implemented. The multi-frequency band, multi-mode operation of an SDR affects the design of the RF hardware and the selection of A/D and D/A converters. The RF front end should be adjustable for different frequencies and bandwidths required by the different standards that the SDR supports. Meanwhile, the ADC's and DAC's must have sufficient sampling rates to support the maximum desired frequency bandwidth. Since the sampling requirements varies according to the type of standards supported, software radios often sample at very high rates, greater than the required bandwidth, and use a digital front end to further filter and lower the sampling rate to the desired values. The digital front end consists of digital filters, mixers, decimators and interpolators implemented either on FPGA's, ASIC's, or DSP's. The core digital signal processing hardware can be performed through DSP's, FPGA's, multi-core CPU's, ASIC's and/or GPU's. The choice of core computational platform depends on the required signal processing algorithms and their computational and throughput requirements. In practice, software radios often employ a combination of core computing elements, dividing the signal processing workload between the various elements. Communication oriented DSP algorithms quite often work best on a particular processor architecture, so multiple core elements provide the most flexible and efficient method of implementing any standard. Both algorithm processing and control routines may use a variety of software methodologies, such as high level programming languages (MATLAB, C/C++, CUDA C), middleware, or dedicated software applications like GNURadio, LabVIEW or OpenBTS.

2.6 SDR Signal Processing Stages

As described, the SDR has particular structures and attributes required for operation. These include the radio frequency front end, the ADC and DAC, the real-time digital processing using dedicated or configurable hardware components, software programmable digital signal processing, and general purpose software processing.

2.6.1 Radio Frequency Front-End

An ideal software radio is one that would have minimal analog RF front end, consisting of an ADC at or near the antenna. However, any practical SDR implementation still needs an RF front end whose design challenges are only exacerbated in software radio [41-43]. The main functions of radio frequency front end are up/down conversion, interference rejection, filtering, pre-amplification and to minimize distortion while achieving required dynamic range.

On the transmitter side, the RF front end takes the digital representation of the analog signal from the DAC, up-converts the signal to the desired RF center frequency, amplifies the signal to the appropriate level, avoids transmitting harmonics or interference by limiting the signal bandwidth with filtering and feeds the signal to the antenna. The operation of the receiver is more complex than the transmitter as the desired signal must be separated from the noisy background RF environment. At the receiver, the signal from the antenna is fed through an RF filter, amplified by a low noise amplifier (LNA), down-converted to a frequency that is compatible with the ADC, and then converted into digital samples at the ADC. These digital samples may undergo further filtering, translation and processing in the digital front end stages. The list below shows some of the parameters that influence the design and selection of the RF front end:

- ***Sensitivity*** is the weakest signal level that the receiver can detect.
- ***Selectivity*** refers to the ability of the receiver to detect the desired signal while rejecting all others.
- ***Dynamic Range*** is the power difference between the weakest and strongest signal that the receiver can detect.
- ***Stability*** is the lack of change in the gain and frequency response of the receiver with changes in temperature, time, etc.

One of the most common RF front end architectures is the well-known super-heterodyne receiver. It was initially designed by Edwin Armstrong in 1918 [44] as a means to overcome deficiencies in early vacuum tube triodes used as amplifiers in radio direction finding equipment. In this design, the received signal is converted or shifted in frequency by a mixer to a fixed intermediate frequency (IF) that is lower than the center frequency of the desired RF signal but greater than the bandwidth of the desired signal. The conversion is usually done in two stages to take advantage of the lower filtering requirements and availability of narrowband RF and IF components. The architecture of a superheterodyne receiver employing I/Q modulation at the second stage is shown in Figure 8. The received signal is first filter by the pre-selection band pass filter, then amplified by the low-noise amplifier and filtered by another bandpass filter to remove the image frequency effects before being mixed at the first stage. After this, the signal is again filtered, I/Q demodulated to baseband and then fed to the input of the anti-aliasing filter and then the ADC. The superheterodyne receiver has good selectivity and sensitivity but the bandwidth and the center frequency of the filters are narrow, fixed and not flexible. This makes the superheterodyne architecture unsuitable for wideband or simultaneous narrowband signal SDR applications. Multiple front ends or adjustable filter components mitigate these problems but add to complexity, cost and weight. Another option is to process the IF signal digitally. In this case the second mixing and filtering stages are done digitally.

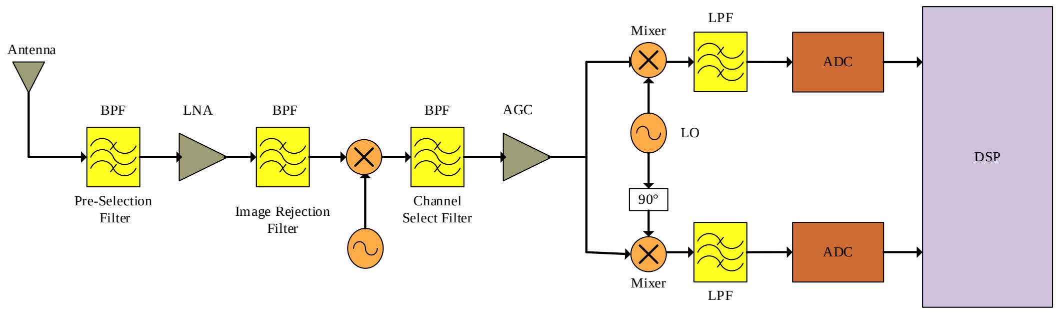


Figure 11. Superheterodyne receiver

In a direct conversion receiver architecture also known as the homodyne receiver or the zero-IF receiver, the received RF signal is directly down converted to baseband using a single mixing stage. The received signal is first bandpass filtered, amplified by the LNA and then mixed using an I/Q demodulator. I/Q demodulators are required when detecting phase or frequency modulated signals. As before the down-converted signal is filtered by the anti-aliasing filter before being fed to the ADC. Figure 9 illustrates one such direct conversion receiver that uses quadrature sampling.

Direct conversion receivers are conceptually attractive due to their simplicity and ability to switch between specific bands and modes relatively easily. However there are some air interface standards that are unsuitable for direct conversion receivers.

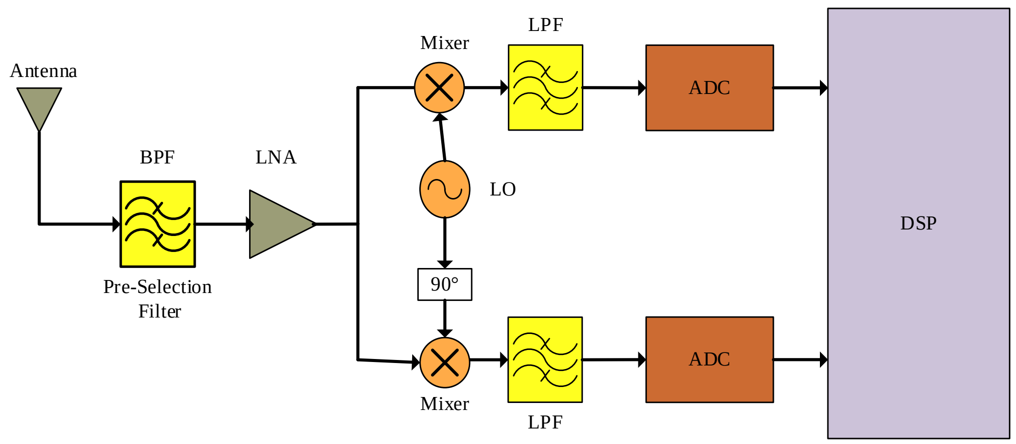


Figure 12. Direct conversion or zero-IF receiver

Moreover due to the fact that the local oscillator is at the desired signal frequencies, this architecture requires an extremely stable local oscillator and mixers with no local oscillator feed through to avoid unauthorized emissions and interferences. Some of the problems can however be compensated with digital signal processing which has prompted some sources [43] to suggest that the direct conversion architecture is the most promising RF front end architecture for software defined radios. As an alternative to a zero-IF design, this architecture may also be used to provide complex signal mixing to an IF frequency where digital signal processing may correct analog I/Q signal offsets and thereby eliminate signal images. It is similar to an IF sampling quadrature superheterodyne technique but can support both zero-IF and complex-IF processing.

A tuned radio frequency (TRF) receiver consists of only an antenna connected to a tunable, or switch selectable, RF bandpass filter and a low noise amplifier (LNA) with automatic gain control (AGC). An illustration of this type of architecture is shown in Figure 13. The BPF selects the signal and the LNA combined with the AGC amplifies the signal to levels acceptable by the ADC. The wide bandwidth and roll-off limitations of the RF filter requires an ADC with very high sampling rate or acceptable subsampling frequencies and bandwidths. Trade-offs between dynamic range and RF filter roll-offs also need to be made. Again as before digital filtering may need to be performed to select desired channel. However, compared to superheterodyne receivers, problems related to the local oscillator are eliminated.

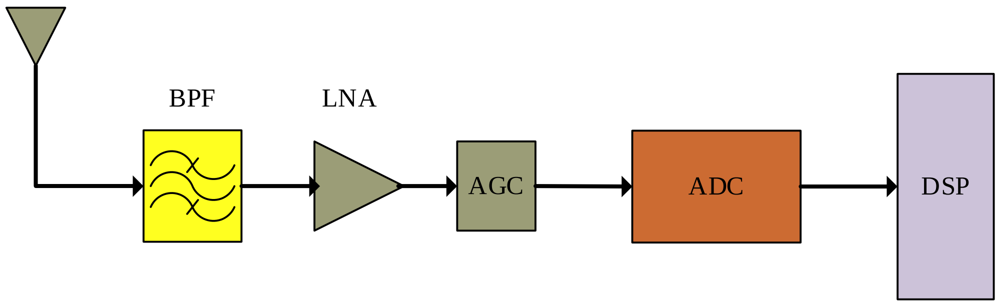


Figure 13. Tuned RF receiver

2.6.2 Analog to Digital and Digital to Analog Conversion

The analog-to-digital and digital-to-analog converters are amongst the most difficult to select and the one that is likely to drive the overall architecture of the software defined radio. In many cases, apart from cost and performance, their placement in the receiver chain will define the bandwidth, dynamic range and power consumption of the radio. An ideal software radio would use data converters at RF, resulting in conflicting needs:

- Very high sampling rate to support large bandwidths
- Bandwidths running up to several GHz
- High dynamic range for recovery of weak signals
- High resolution (quantization bits) to support the high dynamic range
- Low power consumption and reasonably low cost

However, advances in ADC and DAC technology is slower compared to other areas related to software defined radio and therefore these demands exceed the specifications of currently available converters.

The Nyquist sampling theorem determines that the rate at which a bandlimited signal must be sampled should be at least twice the signal bandwidth to ensure that the original signal can be reconstructed from the samples.

$$\frac{f_s}{2} \geq BW$$

Higher frequencies cause aliasing and therefore the ADCs are often preceded by an anti-aliasing low pass filter to band limit the spectral content of the original signal. SDR places additional constraints on the selection of these anti-alias filters. Ideal SDR's require frequency and bandwidth independent RF front ends, but even with tunable anti-alias filters, the flexibility of the software radio will be severely constrained. Oversampling reduces stringent requirements for the anti-aliasing filter but on the other hand more is the data that needs processing. Another technique is to use quadrature sampling, where the input signal is split into in-phase and quadrature components. Bandwidth is half of the bandwidth of the original signal; downside being the need for synchronized converters.

The number of bits in the ADC defines the maximum achievable dynamic range. Higher dynamic range also requires higher stop band attenuation in the filters. Current state of the art ADCs for software defined radio applications have 2GHz bandwidth, with 1GSPS sampling rate and approximately 85dB spurious free dynamic range and 14-bit resolution [45]. SDFR denotes the difference between minimum detectable input and he point at which third order distortion becomes stronger than that. Large SFDR is needed to allow recovery of small scale signals in the presence of large interference [42].

Power consumption is another key parameter to consider when selecting ADC; especially when it involves low power mobile devices. The ADC specified in the previous paragraph has a power consumption of 1.65W which may be too high for mobile radios. Although ADC performance is widely discussed and often the limiting factor in SDR architecture, the transmit path with the DAC is also a significant design problem. The requirements for DACs include high linearity, filtering and isolation of clock from output in order to prevent distortion and out-of-band emissions.

2.6.3 Digital Signal and Software Processing Choices

Compared to traditional wireless hardware design, the signal processing and other capabilities performed in digital components make the choice of processing element a more complicated affair. There are four key interrelated and often conflicting factors that must be considered when tasked with selecting appropriate processing hardware for any software defined radio application. These are:

- ***Flexibility*** to handle a multitude of standards and protocols; some yet to be defined.
- ***Modularity*** allows for both software and hardware to be reconfigured or modified quickly as and when needed.
- ***Scalability*** is the ability to expand current implementation by increasing number of channels that the system can handle.

- ***Performance*** may relate to computational capabilities, cost and other metrics that the system designers may use to aid in selecting the appropriate hardware.

There are five main categories of digital hardware available for implementing real-time software radio systems: ASICs, FPGAs, DSPs, multicore general purpose processors (GPPs) and GPUs [46]. All these hardware elements offer differing levels of programmability, reconfiguration time, processing power, power consumption, cost, etc. So at one end, there may be parameterized components that are application specific integrated circuits (ASICs) and at the other end the hardware may be totally configurable, e.g. GPPs. ASICs, provide highly optimized implementation but can only be utilized in the applications and modes it was designed for. DSPs are architecturally similar to microprocessors and offer support for programming in high-level languages. However, instructions in the DSP are executed sequentially and so applications that can be parallelized may not map efficiently. There are DSP platforms that provide multicore support but their size, number and power may not be optimal for a particular task [47]. On the other hand, the signal processing performance of FPGA is normally high, but time required for reprogramming is slower compared to other options. The ubiquity of the PC coupled with advances in the development of software tools and languages makes programming GPP the easiest compared to all other components. However, care has to be taken so as to not reduce the speed and functionality of other tasks traditionally assigned to the GPP in the computer. GPUs can process huge amounts of data, can be programmed using high-level languages and are found in most PCs. However, there is a steep learning curve to programming GPU's and not all algorithms map efficiently to the fixed architecture of the GPUs. The lines separating these three categories are constantly changing, with new products incorporating features from the other categories in an attempt to provide an optimum level of flexibility, high speed processing and power conservation. Another possibility currently being explored is to incorporate two or more processing elements from different categories thereby creating a system that offers the best features of the individual elements while at the same time masking their drawbacks.

2.7 Software Architecture, Reconfiguration and Management

Although software architecture and reconfigurability are essential parts of the SDR concept, it is a complicated and a wide issue. Software architecture ensures maintainability, expandability, compatibility, and scalability of the software radio system. Ideally, the software architecture should have a middleware layer that serves as the interface between the underlying hardware and the application oriented software layers. This will allow software application developers from worrying about the complexities of the hardware layer.

Software reconfigurability can be done at multiple levels. Early software radios were reconfigurable only at the air interface. Newer radios are reconfigurable even with respect to functionality. Service providers or users can perform reconfiguration by software download over the air (OTA) or by using wired connections. Reconfiguration at the lower levels allows roaming or other physical layer modifications. At higher levels, reconfigurability allows possibility of context aware services and applications. In the simplest case, reprogamming is performed based on user requirements. In the more complex case, configuration can be based on environment, queried services, and capabilities of users and network. Currently there exists no standardization with regards to the software architecture, frameworks, implementation, and deployment. Therefore, one of the challenges of future software based radios is technical and regulatory work on flexible, dynamic and efficient means of reconfiguration.

2.8 Summary

Software defined radio has emerged from obscurity to be heralded as the most promising solution to many of the problems plaguing the wireless communication industry. In this chapter we attempt to provide a historical background to the origins of wireless communication, the different types of analog and digital radio architectures and the issues of both current and future wireless radios. Then we set about answering ‘What is Software Radio’, the ‘Need for Software Radio’, and its potential benefits. One of the challenges in creating software radio is the broad scope of knowledge required, from RF circuits, to signal processing, software development and digital circuits. We therefore provide an introduction to the various architectural elements of software radio, various RF front ends, ADC, DAC, digital front ends, and finally the different types of processing elements. We also discussed how the multi-mode, and reconfigurability requirements affect the various parts of the radio from the processing hardware to the RF front end.

CHAPTER III

SDR SYSTEM ARCHITECTURE AND PROTOTYPE

The generalized SDR platform architecture and initial laboratory demonstration prototype developed for this work are based on a review of available SDR systems, consideration of the digital signal processing required for current and emerging communication standards, and a focus on commercially available, cost sensitive components and subsystems that can be incorporated, based on signal requirements, into a generalized SDR platform. The architectural components are cost sensitive as they are all commercially available and, except for radio front-end elements, are being marketed or developed for non-SDR computer, personal computer or even supercomputer applications. As such, continued rapid advancement in capability and performance may be expected along with commercial demand that will reduce costs, size, weight, and power requirements.

Before defining the architecture and prototype, an analysis of numerous SDR and high speed computer prototypes, commercial component designs and constructed devices will be presented. This will include; SDR RF electronics, ADC and DAC, and dedicated hardware processing elements; high rate data communication elements; parallel computation capability; and finally PC platform.

3.1 Current SDR Platforms

Some of the most popular SDR research platforms are currently available from companies like Pentek, Lyratech, 4DSP, Analog Devices and other. Regrettably, most of these platforms use proprietary software tools that are not particularly suitable for academic research or, in general, detailed publication and dissemination. An overview of other commercial and more accessible platforms follows.

3.1.1 Sora

The Microsoft research software radio (Sora) is a hybrid SDR platform that attempts to leverage the processing performance of an FPGA with programmability and flexibility of a general-purpose processor. Thus with a multi-core PC running a normal operating system (Microsoft Windows), Sora drivers, software development kit (SDK) and the Sora hardware kit installed, developers are able to realize modern high-speed wireless protocols like 802.11 and LTE [48]. The Sora hardware consists of PCIe based radio control board (RCB) with an interchangeable radio front end for transmission and reception.

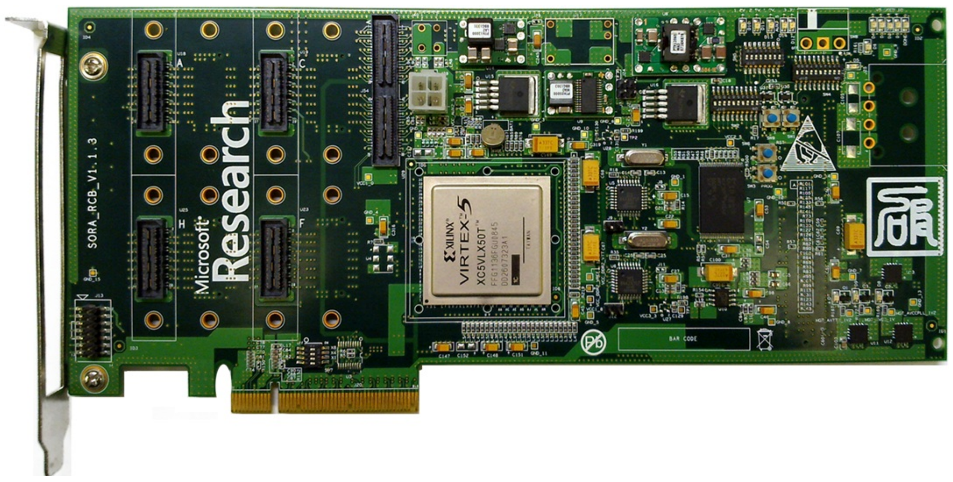


Figure 14. Sora radio control board (RCB) [49]

The RCB is designed and developed by Microsoft Research whereas the RF front end is third-party. Currently, only the boards listed in the table below are supported. The capability of the RF front end varies with manufacturer but overall it contains A/D and D/A converter and RF circuitry to perform frequency translation, amplification and filtering.

The RCB is designed around a Xilinx Virtex-5 FPGA with PCIe-x8 interface, providing high-throughput, low latency path for transferring digital signals between the RF front end and the PC memory. The x8 PCIe lanes are capable of providing transfer rates of up to 16.7 Gbps, satisfying throughput and timing requirements of modern wireless protocols [50].

Table 1. Sora compatible RF front-end daughtercards

RF Front End	**Frequency Band**
V3 SDR	2.4 GHz and 5 GHz
Rice University – WARP radio board	2.4 GHz and 5 GHz
/Ettus Research USRP - XCVR 2450c	2.4 GHz and 5 GHz

Figure 15 illustrates the logical components of the Sora hardware platform. The direct memory access (DMA) and the PCIe controller interfaces the RF front end with the PC. The various registers, FIFOs and the DDR memory enable smooth transfer of data between the synchronous RCB and the asynchronous host memory. The RCB interfaces to the RF front end through a standard LVDS interface and has the capability to support multiple daughtercards.

Sora software is written in C and makes use of lookup tables (LUT's) to take advantage of large caches on modern GPPs and single instruction multiple data (SIMD) extensions to accelerate data processing. The Sora software architecture also supports stream processing of signal components across multiple cores. A new kernel service called 'core dedication' manages processor core allocation exclusively for SDR tasks. Finally, the Windows platform provides developers with a rich set of tools for application development and debugging.

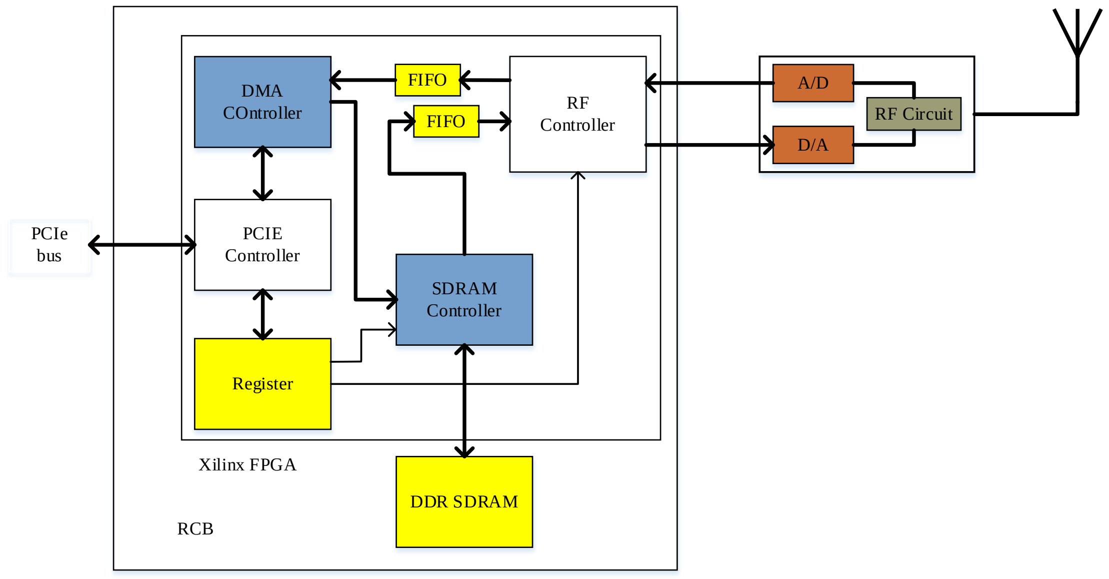


Figure 15. Hardware architecture of Sora

The hardware and software optimizations mentioned above allow Sora to fully support high speed wireless implementations like IEEE 802.11 a/b/g PHY and MAC layers entirely in software.

3.1.2 WARP

The “Wireless open Access Research Platform” (WARP) is a custom SDR platform developed at Rice University and Mango Communications for conducting research in advanced wireless algorithms and applications. The WARP platform was built from the ground up and consists of four key components [51]:

1. ***Custom hardware*** capable of implementing high speed DSP algorithms for communications, scalable interconnects that can handle hundreds of Mbps data rate, and modular architecture that supports future expansion.
2. ***Platform support packages*** including tools and drivers that provide seamless interface between the application layers and the underlying hardware layer.
3. ***Open access repository*** is an online archive that provides a community based support and discussion forum for users and developers alike.
4. ***Research applications*** with the goal of providing building blocks that allow for rapid prototyping of other novel and complex algorithms.

There are three iterations of the WARP hardware platform (v1, v2, and v3). All the versions use a FPGA as the primary processor. The computational capability of the FPGA in the three generations has steadily increased keeping in touch with the increasing demands of the evolving wireless standards. Version 1 and 2 platforms used the Xilinx Virtex II Pro and Virtex-4 FPGAs respectively with a PowerPC processor core providing MAC and network layer support. The latest v3 boards contain a Virtex-6 FPGA with a Xilinx MicroBlaze processor replacing the PowerPC core. The WARP v3 board has two programmable RF interfaces built around the Analog Devices AD9963 MxFE that integrates two 100 MSps, 12-bit ADCs and two

170 MSps, 12-bit DACs. Digital interpolation and decimation filters, programmable analog gain and offset adjustments are also provided in the RF interface section. There are two gigabit Ethernet ports that can be used to transfer samples over user datagram protocol (UDP) between the PC and the WARP board. The board also implements a high-pin count FMC connector that can be used to interface additional RF daughtercards and other high speed peripherals to the FPGA. A DDR3 SO-DIMM slot is also provided on the board. The Ethernet and DDR memory controller logic are designed using Xilinx cores which may be a limiting factor in certain cases.

There are two different methods by which physical layer designs can be incorporated into the WARP ecosystem; the first is using the Xilinx System Generator to modify the FPGA logic and the second is to use WARPLab. The first option is generally used when the SDR is required to function as a standalone entity. In the second option, data processing is done offline in MATLAB and therefore this method is more suited for prototyping/testing the designs.

3.1.3 The BEE2 Platform

The Berkeley Emulation Engine (BEE2) is a high-end reconfigurable computer (HERC), designed to research computationally intensive applications like

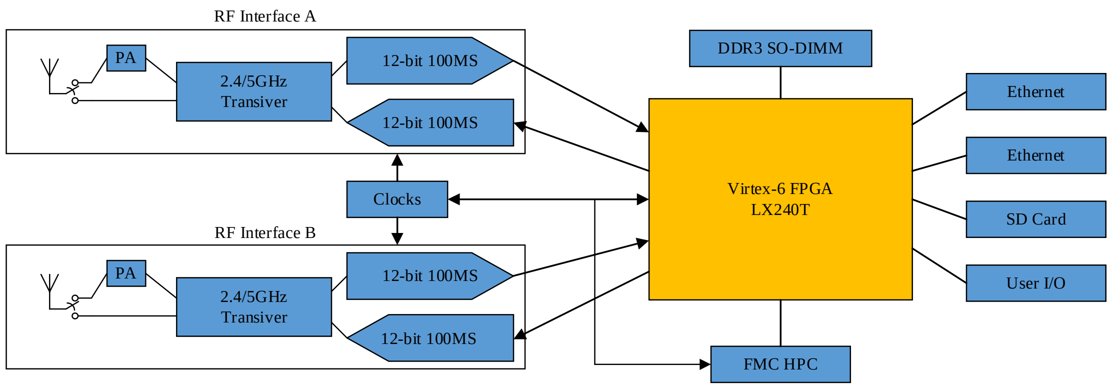


Figure 16. WARP v3 hardware architecture

wireless communication systems, real-time digital signal processing, scientific computation and acceleration of CAD tools [52]. The BEE2 has three primary components: processing elements, memory elements, and interconnects. The primary and only processing element is the Xilinx Virtex-2 Pro 70 FPGAs. Each FPGA embeds a PowerPC 405 core thereby maximizing throughput and minimizing latency between the reconfigurable logic and the processor. Five Virtex-2 FPGAs together form a single compute module with the center FPGA responsible for management and the remaining four for computation as shown in Figure 17. The FPGAs can interface a maximum of 4GB DDR400 memory each, thereby providing a peak memory bandwidth of 12.8GBps. Adjacent FPGAs are connected via onboard low-voltage 40Gbps (LV-CMOS) parallel interfaces. All computation FPGAs are connected to the control FPGA via 20Gbps links. These high bandwidth, low latency links allow the five FPGA to form a virtual FPGA of five times the capacity.

Off board interconnect is through multigigabit serial (MGT) interfaces. Each individual MGT is capable of running at up to 3.125Gbps. Four MGTs are channel bonded to form into a physical Infiniband 4X (IB4X) electrical connector, creating a 10 Gbps full duplex interface. There are a total of 18 IB4X connectors per board—two from two control FPGAs and four from each of the four compute FPGAs. An Infiniband switch can connect the compute modules to provide a high throughput link to communicate and exchange data. The board also has one 100 Base-T Ethernet port available on the control FPGA. This 100 Base-T interface allows for remote management and control. The Power PC of the control FPGA can run Linux and a full IP protocol stack. The board also contains USB and JTAG interfaces along with provision for a flash card.

Programming environment is unlike other parallel computing systems. Instead of traditional sequential high-level languages, the BEE2 is programmed using Matlab/Simulink from Mathworks coupled with the Xilinx system generator. This tool chain is augmented with Berkeley Wireless Research Center (BWRC) automation tools that allow for mapping high level block diagrams and state machine specifications into FPGA configurations.

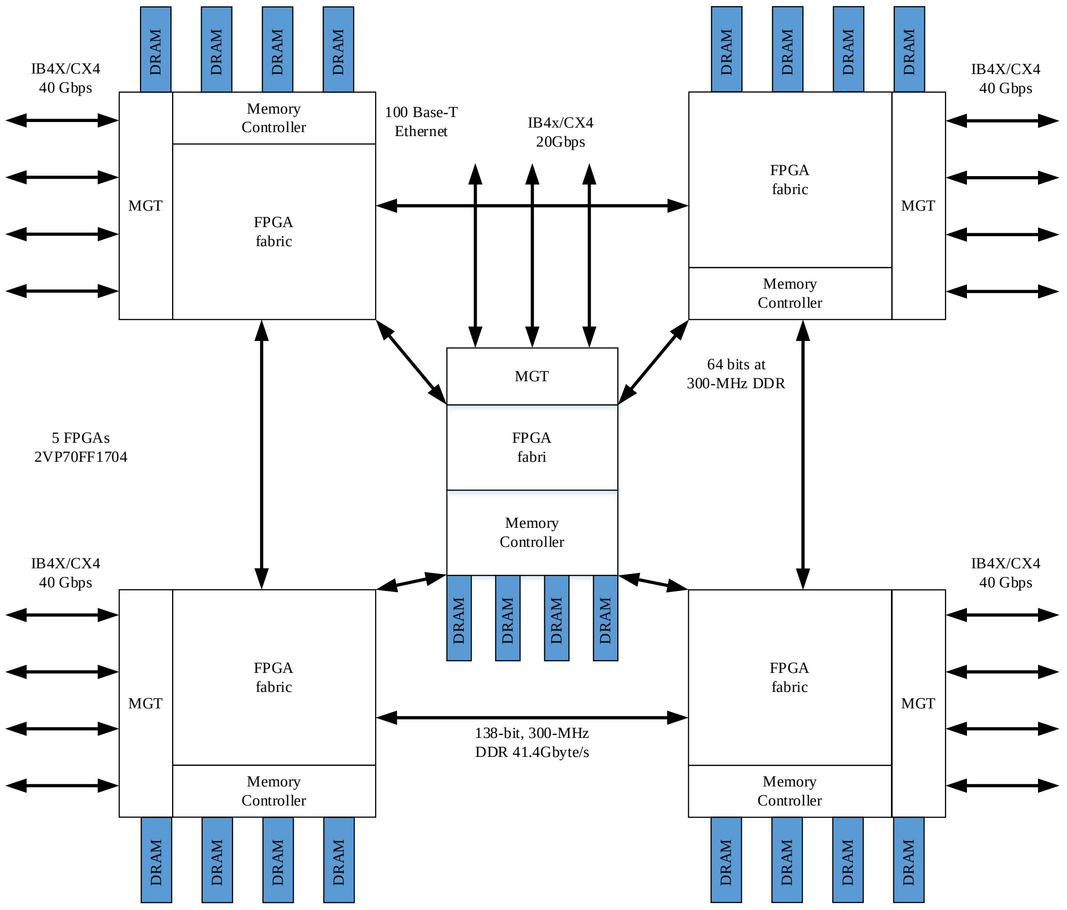


Figure 17. BEE2 compute module

A number of radio testbeds have been developed using the BEE2 [53]. One such system is used for cognitive radio experiments at UC, Berkeley [54]. In this system, the RF front end board contains filters, ADC/DAC chips and a Xilinx Vertex-II Pro FPGA. The ADC is 12-bit sampling at 64MHz and the DAC is 14-bit and 128MHz. The FPGA performs control, calibration and provides real time access to the programmable registers. It also supports 4 optical 1.25 Gbps links for transmitting and receiving data to/from BEE2. The optical link provides good analog signal isolation from digital noise sources and allows the frontend to be moved up to a third of a mile from BEE2 for wide range wireless experimentation. A separate RF modem module connects to the baseband board. The current RF modem module is capable of up/down converting 20MHz RF bandwidth at 2.4 GHz. The RF frequency is fully programmable in the entire 80MHz ISM band. A block diagram of a single RF modem is shown in Figure 18

3.2 Evolving the SDR Platform

In the above section, we discussed three architectures that followed different design philosophies when it comes to the implementation of software radio. The Sora platform uses commodity GPP for processing and is the most cost effective when compared to the other solutions.

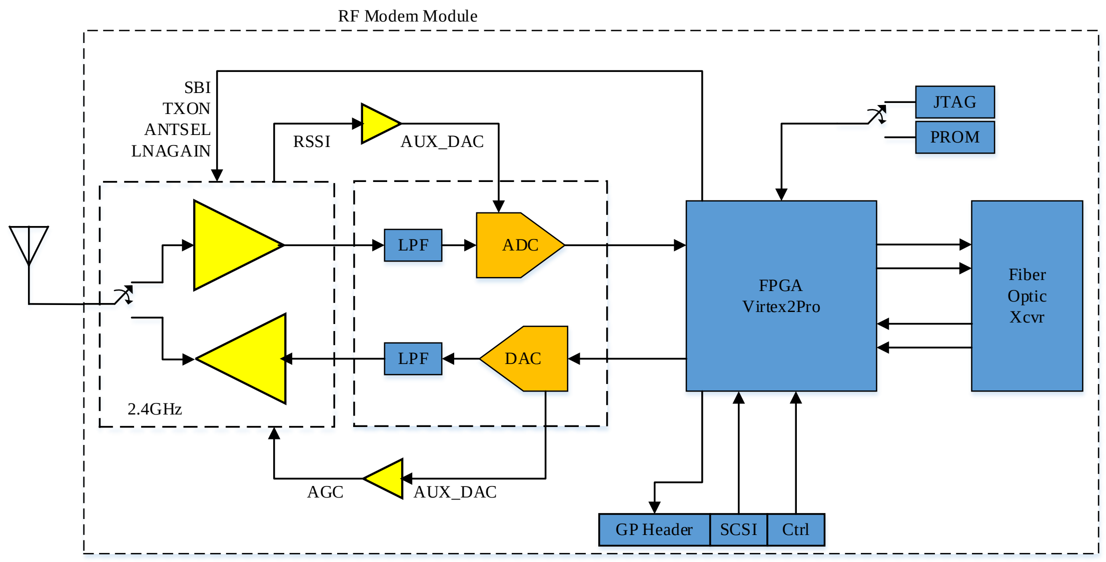


Figure 18. BEE2 RF modem module

It is however unsuited for base station applications where scalability is an important criteria. The GPP as a real-time processing element is also a performance bottleneck because, on a PC platform, the GPP still has to run regular operating system tasks. The WARP platform has the specifications to match the requirements of even the most demanding wireless protocols, but at approximately $10,000 per unit, it is very expensive. It is based on a FPGA with built-in high performance DSP capabilities. But without inherent parallel processing structures, it is left to the designer to build these structures. It also takes significantly longer to reconfigure. The BEE2 platform is the most expensive of the three but has very high computational capabilities. It is also a purely research platform with very little commercial viability.

One thing that unites all these architectures is their use of custom hardware. Though custom hardware allow greatest control over design trade-offs and performance, they also drive up costs. Another key aspect of these designs is their failure to take into account the invariable advances in both software and hardware technologies. When these advances occur it is tempting to define a new platform solution for each small incremental advance. This will create a plethora of incompatible solutions. The solution is to define and create a generic radio platform.

3.2.1 Reference Design

A well-conceived reference is the key to a sustainable and effective SDR architecture. The goal is to design a generic platform that concentrates on high degree of flexibility rather than meeting hardware specifications and requirements. Over-specifying the reference design may also result in platform that is implemented around a single design approach. The platform should also be valid through the incremental changes in technology. This section describes the guiding principles and approaches that were followed in designing a generic software radio platform.

The generic architecture proposed in this section uses commercial-off-the-shelf (COTS) components, is highly scalable and offers good performance. The use of COTS components keeps costs low and allows us to select devices on either end of the performance spectrum, as demanded by the application. Although cost was an essential factor, the design was also driven by the motivation to build a multifunction, multimode platform that supports research activity in multiple generations of narrowband and wideband applications.

3.2.2 Critical Application Areas

Software radios operate on any RF band that is within the capabilities of the underlying hardware. However the subsystem requirements or the amount of processing performed depends on the frequency and the bandwidth that the SDR is operating on. So when building a generic SDR architecture it is essential to identify and classify critical application areas based on two criteria: the number of frequency channels (single channel or multichannel), the baseband bandwidth of the type of service (voice, data, etc.). The critical application areas are:

1. ***Narrowband signals:*** Although wideband systems like Wi-Fi, 4G cellular are the most talked communication systems, there exists and will continue to exist numerous narrowband applications like two-way radios, wireless power meters, police and safety radios, maritime communication systems, etc. Narrowband systems are defined as those having a bandwidth of 25 KHz or less. They are the de facto standard for long-range, low data rate systems. They can further be classified as:

Single channel systems like two-way radios, wireless power meters, police and safety radios, maritime communication systems, wireless sensors.

Multichannel systems like disaster relief base stations, fire and emergency dispatch centers, smart home solutions

2. ***Wideband signals:*** Demand for wideband applications and services has soared in the past decade as shown in Chapter 1. Wideband signals require increased RF front end and ADC/DAC requirements. In addition, wider bandwidth also means more processing in the processing elements. Again they can be classified as:

   Single channel systems like wireless monitors, telemetry systems, etc.

   Multichannel systems like Wi-Fi, Bluetooth, cellular base stations, etc.

3. ***Universal base stations:*** These are systems that provide support to narrowband as well as wideband communications. They also may operate on a single frequency or on multiple frequencies. Cellular base stations that provide 4G and legacy communication support fall under this category.

3.2.3 RF Front-End

One of the primary requirements of the RF front end in software defined radio is to provide flexibility in selecting gain, center frequency, bandwidth, and sensitivity. This multiband/multimode requirement is not limited to software radios. In fact, the push for such radios came from three sources—military, public safety and cellular telephones. For example to support each of the five major US cellular carriers, the RF circuits must be able to tune to a wide range of frequencies with wide bandwidth capability. In addition, spectrum reallocation is constantly making room for more broadband wireless access. Fortunately, manufacturers like Ettus Research have released a wide range of motherboard, and RF daughtercard combinations to serve a wide variety of application areas. A list of the currently available products from Ettus is given in Table 2.

3.2.4 Dedicated DSP or Xilinx Elements

The RF requirements for frequency translation and usable bandwidth are closely tied to the selection of the ADC and the DAC. The wider the bandwidth, the higher the sampling rate of the ADC and the DAC. For example, a 100MSps ADC would provide about 40MHz bandwidth. These high rate ADCs allow current generation SDRs to digitize the IF rather than the baseband. Naturally additional digital signal processing is required to reduce the sample rate to that of the baseband. These sample rate changes may be performed by dedicated filter-decimate and interpolation filters. In addition to this, depending on the architecture of the RF front end, digital up/down conversion may also be required. Coordinate Rotation DIgital Computer (CORDIC) algorithm may be used to perform the frequency translation. Both these tasks are ideally suited for implementation on FPGAs. Additional processing like channel estimation, equalization and forward error correction may also be performed on the FPGA. Although the USRP motherboards contain an FPGA, the amount of logic resources available for user defined logic varies among the different types. Hence the generic architecture includes PCIe based FPGA card plugged into a PC. For example, the Xilinx SP605 shown in the below is an excellent candidate for this role.

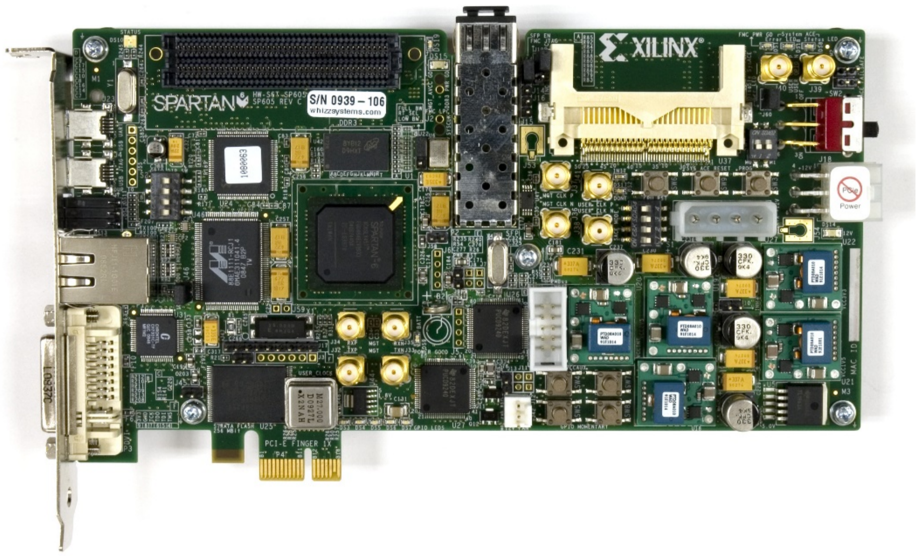


Figure 19. Xilinx SP605 PCIe board as DSP element

3.2.5 High Speed Interconnect to PC

Digital samples from the RF front end needs to be routed to the PC for IF and baseband processing. This link should have high data throughput to support the wide bandwidth capabilities of the RF front end. For example if the RF front end has a 16-bit ADC that samples at 100MSps complex data, the throughput required will be about 800Mbps. So, gigabit Ethernet or USB 3.0 will work as the high-speed interconnect whereas USB 2.0 with 480Mbps will be unsuitable. For even higher bandwidths, only a computer backplane like PCIe or 10 gigabit Ethernet will be able to support the high data rates.

3.2.6 High Speed Backplane Communication PC Motherboard

Peripheral Component Interconnect Express (PCIe) is a high-speed computer interconnect bus that offers high throughput and low latency. The PCIe standard defines slots and connectors with multiple widths (x1, x4, x8, x16 and x32). Depending on the PCIe version, each lane can transfer data anywhere between 2Gbps to 15.75Gbps. Such high transfer rates coupled with its ubiquitous existence make it a perfect candidate for transferring data amongst the various components of software defined radio. It also supports multiple link widths external cabling over short distances.

Table 2. Ettus research product list [55]

Name	**Type**	**Bandwidth**	**Frequency range**	**Interfaces**	**Processor**
USRP X series	SDR	120 MHz	DC-6 GHz	PCIe, dual 10 GigE, dual 1GigE	Kintex-7
USRP N series	SDR	40 MHz	–	GigE, custom expansion interface	Spartan 3A DSP
USRP B series	SDR	32 MHz	–	USB 2.0	Spartan 3A
UBX	Daughtercard	140 – 160 MHz	10 – 6000 MHz	LVDS	–
CBX	Daughtercard	40 – 120 MHz	1200 – 6000 MHz	LVDS	–
SBX	Daughtercard	40 – 120 MHz	400 – 4400 MHz	LVDS	–
WBX	Daughtercard	40 – 120 MHz	50 – 2200 MHz	LVDS	–
Basic TX/RX	Daughtercard	–	1 – 250 MHz	LVDS	–
LF TX/RX	Daughtercard	–	0 – 30 MHz	LVDS	–

The generic SDR will use the PCIe as an expansion slot to support multiple GPU compute cards and/or multiple SP605 FPGA cards. Such a requirement is envisioned for research on multiple-input multiple-output (MIMO) systems, simulating cellular base stations and beamforming.

3.2.7 GPU

Commercial deployment of SDR has always suffered due to the lack of low cost, high computing power, processing solutions. Conventional SDRs use GPP, FPGA and DSPs for their signal processing needs. However, in order to perform parallel processing, the number of devices needs to be scaled up adding to the cost of the overall system. Recent advancements in GPU computing whether it be architectural changes or programming environments, has made it more attractive as a viable alternative to FPGAs and DSPs. The single instruction multithread aspect of the GPU architecture is well suited for certain structurally parallel communication algorithms. Since majority of signal processing algorithms are iterative and mathematically computation intensive, the thousands of cores on a GPU will allow designers to accelerate these algorithms.

3.2.8 Multicore Multithreaded GPP

Although the GPU and the FPGA will be the main processing element in the reference SDR platform, a multicore multithreaded GPP is also included. The GPP is more of a co-processor in this aspect and will take care of initial configuration, control and re-configuration. Certain application layer processing may also be performed on this component. The GPP will run on a Linux environment owing to the fact that the OS provides more control over system resources like the USB, Ethernet and PCIe buses.

3.2.9 Generic System Architecture Block Diagram

The block diagram of a Generic SDR architecture based on the discussions of the previous section is shown Figure 20. The arrangement of the blocks is based on a typical signal processing flow as discussed in Chapter II. Due to the flexibility that this generic architecture provides, the structural elements can be removed and/or arranged differently depending on the requirements of the type of radio being supported.

Interconnects used in this generic architecture are PCIe and gigabit Ethernet, as they are the most widely available high speed interfaces as of this writing. It is possible that future PC platforms will support high speed interfaces like InfiniBand and Thunderbolt standards that are currently relegated to a few high-end systems.

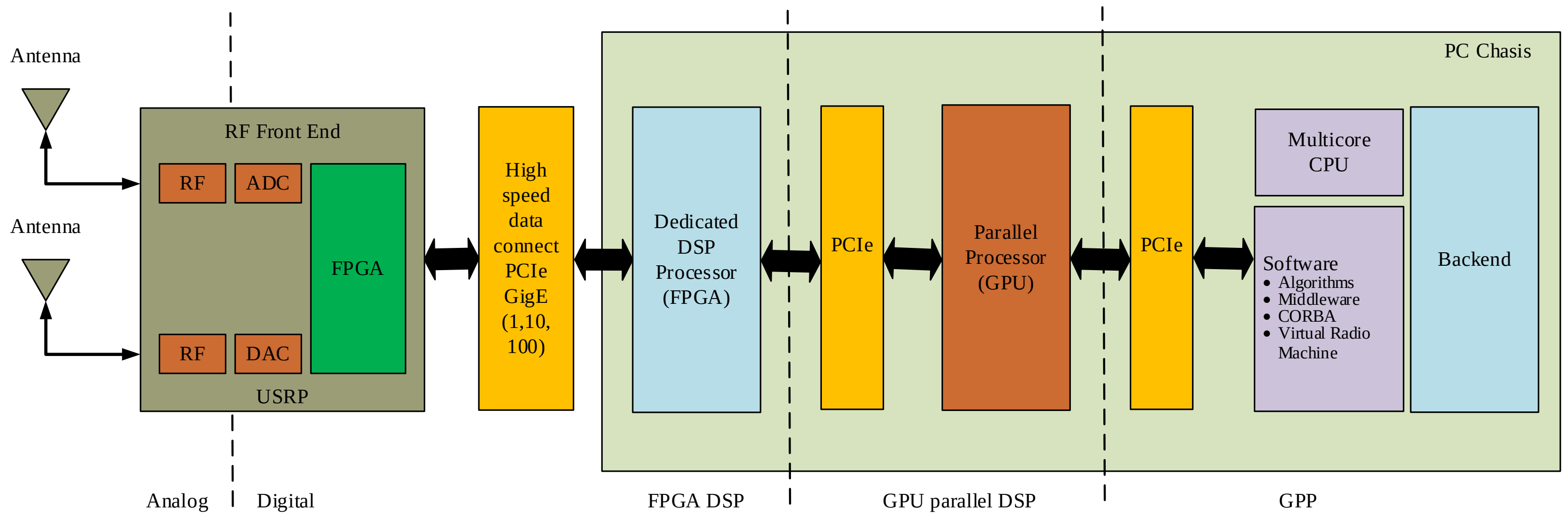


Figure 20. Generic SDR architecture block diagram

3.3 Demonstration System Architecture

Now that the generic architecture has been defined, we must validate the system by building a prototype. Not all the architectural elements of the reference design have been incorporated in the prototype system described in this dissertation.

3.3.1 Hardware Overview

In this section the initial laboratory porotype system architecture for a GPU based SDR platform is presented. A block diagram of this architecture is shown in Figure 21. It is based on a commodity workstation motherboard (Dell T1500) with a PCI Express backplane. PCI Express is a high performance IO interconnect for peripherals in computing/communication platforms. It is a serial packet based point-to-point interconnect with transfer rates of up to 2.5 Gb/s per lane (x1). Lanes can be aggregated to obtain higher throughput (x4, x8, x16, and x32). The choice of a standard bus like PCI Express enables usage of boards that are produced in high volume (and therefore at comparatively lower cost) and makes the system modular and upgradeable in the future. Two USRP devices (N210 and B100) interface the RF front end to the PC. A single Nvidia Quadro 600 graphics card functions as the main GPU based parallel signal processing module.

3.3.2 Dell T1500

The mainboard used in the testbed is a Dell 0XC7MM motherboard packages as T1500 workstation computer. It contains Intel Core i5-750 quad-core CPU running at 2.67 GHz. There is one x16 (16 lanes, 32 Gb/s raw data rate) and one x1 (1 lane, 2 Gb/s raw data rate) PCI Express slot. The extra PCIe slot is useful for connecting an extra FPGA board in case we need more logic resources. A Linux based distribution will be used as the main operating system. This should provide us more control over all the peripherals on the motherboard.

3.3.3 Nvidia Quadro 600

The gaming markets demand for real-time, high definition 3D graphics, has pushed graphics chip makers to modify programmable graphics processor unit into a highly parallel, multicore multithreaded processor with enormous computing power and high memory bandwidth. The computing power of GPUs, measured in terms of single precision floating point operations per second (FLOP/s) is well beyond the capability of modern CPUs. Combined with high memory bandwidth, double precision support and new software ecosystem for simplified programming, the GPU is even more appealing for general purpose computing.

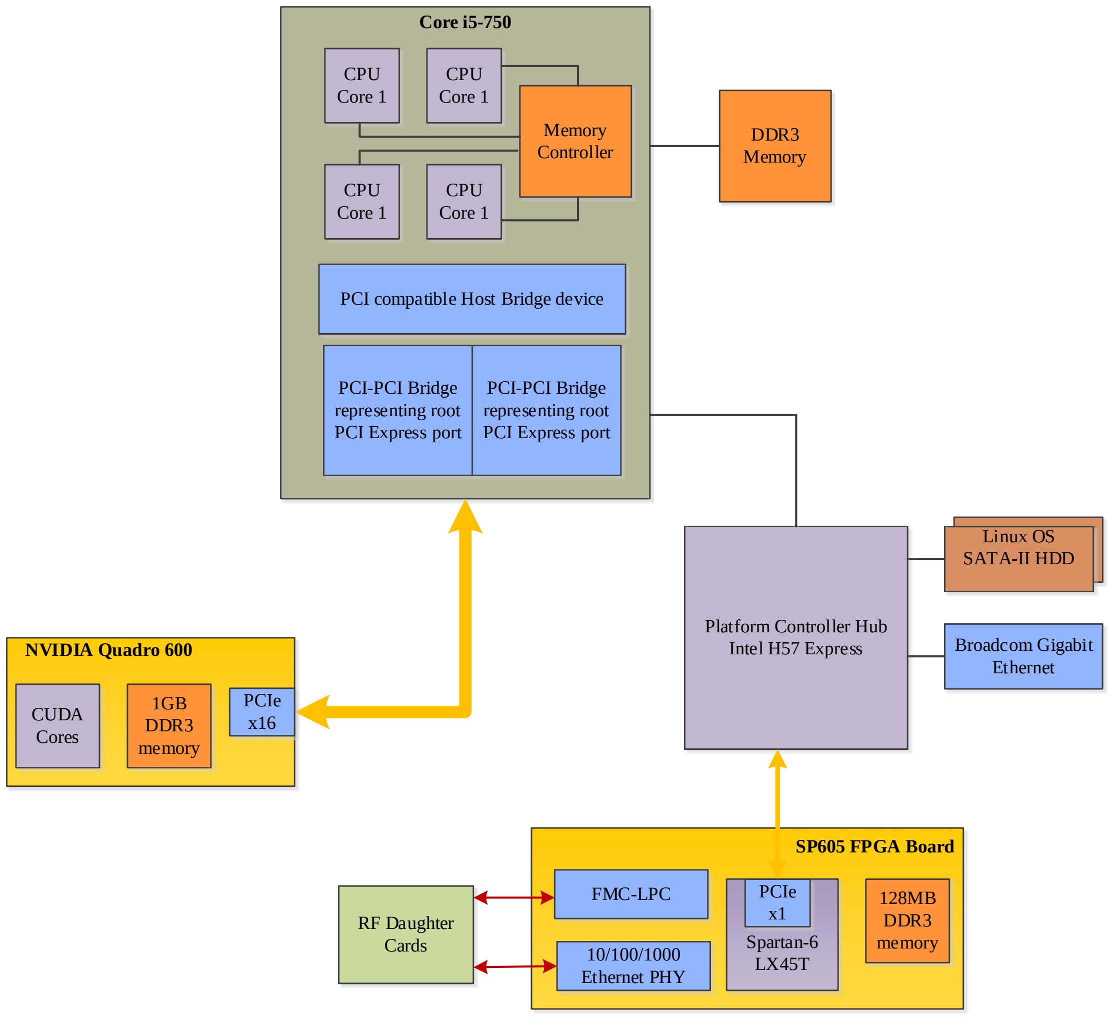


Figure 21. Proposed SDR Architecture

The massive discrepancy in floating point capability between CPU and GPU is due to the fact that the GPU is specialized for smaller size, compute intensive, massively parallel computation and therefore has more transistors devoted to processing rather than data caching and flow control. This is schematically illustrated in the Figure 22 below.

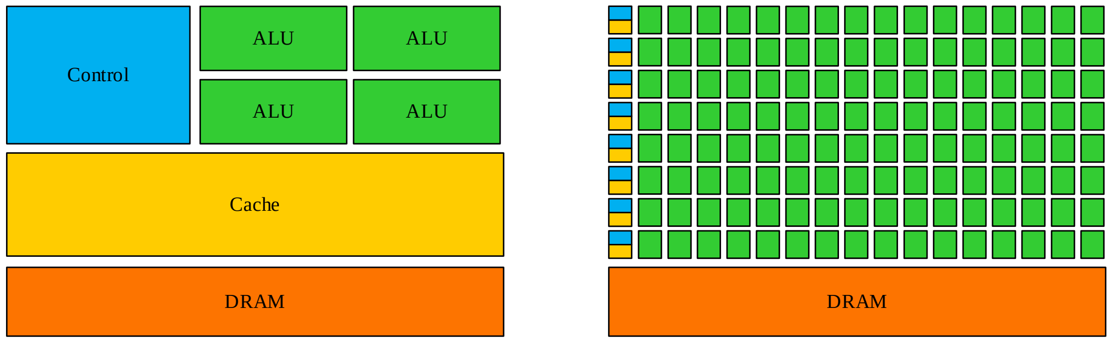


Figure 22. CPU vs GPU

Current generation of general purpose graphics processing units (GRGPU) are extremely flexible and powerful processors that implement single instruction multiple thread (SIMT) computational cores. The memory hierarchy of graphics architectures contains a number of memory spaces (global, constant, texture, shared and thread level) each with different features. This makes programming GPU's more difficult than devices with single unified memory model. GPU's also require rewriting algorithms to map to the small processing cores of the GPU architecture.

NVIDIA Compute Unified Device Architecture (CUDA)

The NVIDIA Compute Unified Device Architecture (CUDA) is both a family of hardware designs and a set of extensions to the C/C++ programming languages to interface to an NVIDIA graphics processing unit (GPU) enabling the execution of general purpose computing programs on a GPU device (GPGPU) as a coprocessor to a host CPU. The CUDA API provides functions to configure devices, allocate and copy data between host and device, and executing Single Processor Multiple Data

(SPMD) functions, called kernels, on the GPU device, as well as other functionality. The CUDA-supported architectures provide a thread hierarchy for massively parallel execution of kernel code, as well as a memory hierarchy for providing threads access to necessary data. The thread hierarchy is designed to allow for potentially tens of thousands of threads to be launched concurrently on a single device. This level of concurrency can provide significant increases in performance over traditional serial code. In particular, GPGPU programming using CUDA has been shown to yield significant performance improvements in many scientific computing applications.

The thread execution and memory hierarchy of the NVIDIA CUDA programming model can be seen in Figure 8. The execution threads are grouped into one or two dimensional thread blocks, and those thread blocks are further organized into one or two dimensional groups of thread blocks, called grids. This thread execution organization corresponds to the various types of memories that are shared among thread blocks and block grids. The CUDA memory hierarchy provides threads three types of read/write memory spaces. Each thread has access to its own local registers, each thread block has access to shared memory, and all threads within all thread blocks and grids have access to global memory. There are also two types of read-only memory spaces, constant and texture memory, both of which act as high-speed cached memories accessible by all threads.

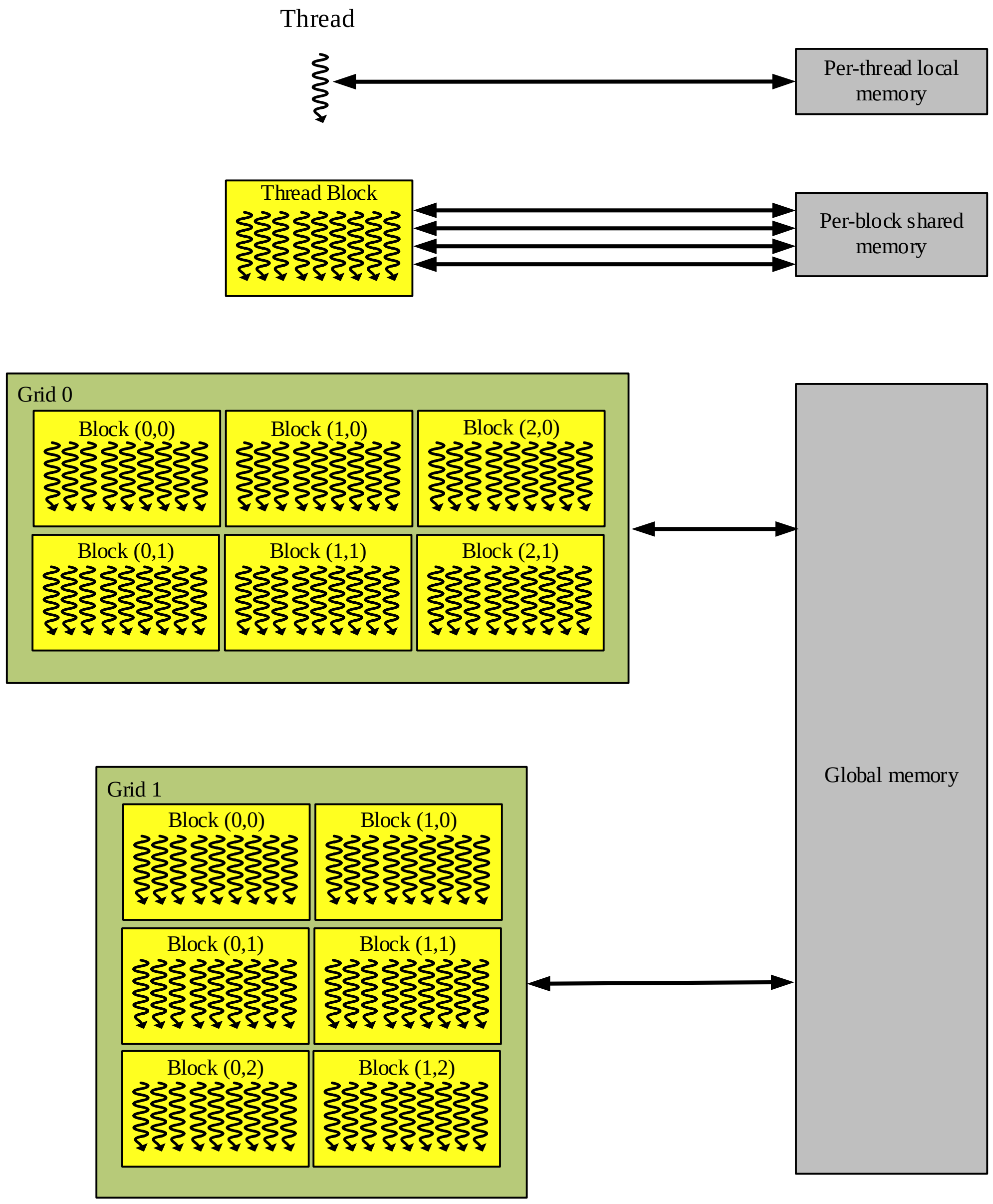


Figure 23. CUDA thread execution model and memory hierarchy

Each of the various GPU memory types have different access times and are best used for different types of computation. Properly managing the memory accesses and data flow through one’s algorithm is crucial to achieving good performance in a CUDA application. CUDA provides both synchronous and asynchronous API functions for launching GPU kernels and performing memory operations, such as allocation, deallocation, and copying data between the CPU host and the GPU device. Synchronous functions execute just as a C/C++ function would, returning to the calling function only after it has completed its execution. Asynchronous functions return to the calling function immediately, launching their operations concurrently. In order to query and synchronize code with asynchronous functions, the CUDA programming model provides events that can monitor asynchronous function execution status.

The functionality in the CUDA API is very closely tied to the proprietary NVIDIA GPU thread and memory hardware architectures. Execution features such as floating point support, floating point standard compliance, atomic functions, concurrent data copy and kernel execution, and others may vary depending on which generation of hardware is being used. All of these differences between the CUDA programming and memory models and the traditional unified shared memory model of general purpose processors makes developing applications for GPUs in a way that is portable and adaptable difficult. This is the primary reason for needing an intuitive and efficient abstraction of the interface between programmer and NVIDIA CUDA API that works with other heterogeneous computing architectures well.

3.4 Results and Analysis

Validation of the prototype SDR architecture was performed at multiple points in the signal flow chain. In this chapter, the RF section of the SDR was tested by implementing real-world scenarios in which the SDR may be employed. For this experiment, GNURadio tools were used to build models of transmitters and receivers. Examples were chosen for both narrowband and wideband applications. Other system elements of the SDR architecture will be tested separately in the coming chapters.

3.4.1 FM-RDBS Receiver

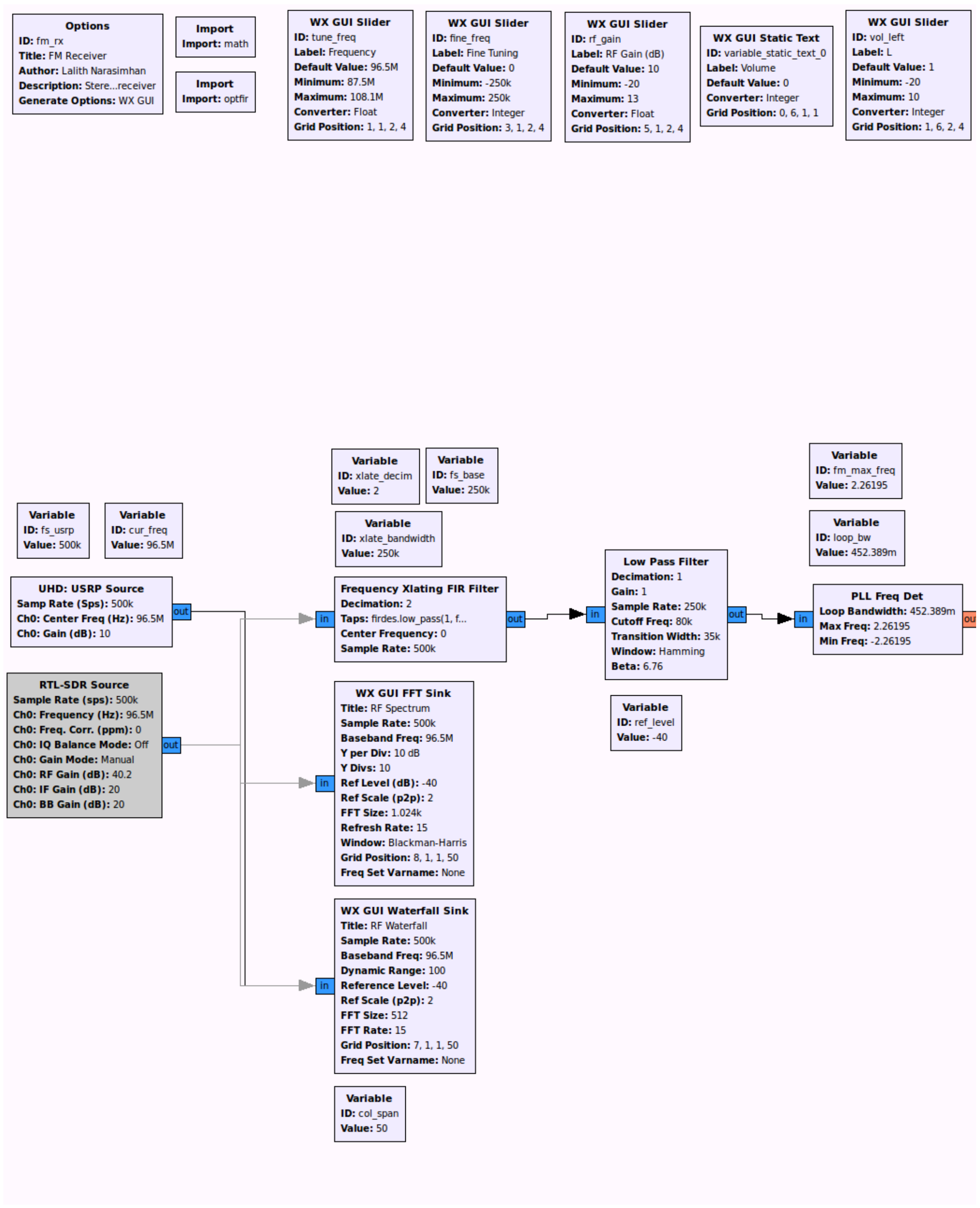
The frequency modulation (FM) receiver was built to demonstrate the capabilities of the BasicRx daughter card and the USRP B100 series devices. Broadcast FM radio is transmitted in the 88 MHz to 108 MHz frequency range and falls into the very high frequency (VHF) part of the radio spectrum. It works by modulating the instantaneous frequency of RF carrier wave in accordance with the input signal. FM was chosen for demonstration because it is still employed in numerous two-way radio systems. Broadcast FM will also be the perfect candidate in future to test polyphase channelizers described in Chapter IV of this dissertation. Broadcast FM is mostly stereo, transmitting the (L+R) signal on baseband, a pilot at 19 KHz, (L-R) centered at 38 KHz. RDBS stands for radio data broadcast system. It is a way to embed digital information into the FM broadcast. Typical information transmitted includes time, station identification, traffic updates and program information. RDBS is on a 54 KHz subcarrier carrying 1187.5 bps. The RDBS data is encoded using differential binary phase shift keying (BPSK).

The block diagram of the receiver as implemented in GNURadio is shown in Figure 24 - Figure 26. The BasicRX board of the USRP does not contain any tuners, instead it relies on aliasing or subsampling to tune to frequencies greater than the Nyquist rate of the ADC (32 MHz). The received signal is filtered, down sampled and demodulated using a phase locked loop (PLL) detector.

The received spectrum of the FM signal has a very distinct shape. As mentioned earlier, the L+R channel is at baseband audio, followed by the 19 KHz pilot. The L-R channel is between 23-53 KHz and the RDBS data is at 57 KHz. Each FM station occupies 200 KHz bandwidth. The deciphered RDBS output can also be seen. The constellation plot of the RDBS shows BPSK nature of the signal.

3.4.2 ADS – B Receiver

Automatic dependent surveillance – broadcast (ADS – B) is a surveillance system in which the aircraft automatically broadcasts its position information to the ground. ADS – B is a subsystem of the aims to replace secondary surveillance radar at airports worldwide. Commercial aircraft equipped with ADS – B transmit identification, position, heading information on 1090 MHz in an unencrypted fashion.



Options
ID: fm_rx
Title: FM Receiver
Author: Lalith Narasimhan
Description: Stere...receiver
Generate Options: WX GUI

Import
Import: math

Import
Import: optfir

WX GUI Slider
ID: tune_freq
Label: Frequency
Default Value: 96.5M
Minimum: 87.5M
Maximum: 108.1M
Converter: Float
Grid Position: 1, 1, 2, 4

WX GUI Slider
ID: fine_freq
Label: Fine Tuning
Default Value: 0
Minimum: -250k
Maximum: 250k
Converter: Integer
Grid Position: 3, 1, 2, 4

WX GUI Slider
ID: rf_gain
Label: RF Gain (dB)
Default Value: 10
Minimum: -20
Maximum: 13
Converter: Float
Grid Position: 5, 1, 2, 4

WX GUI Static Text
ID: variable_static_text_0
Label: Volume
Default Value: 0
Converter: Integer
Grid Position: 0, 6, 1, 1

WX GUI Slider
ID: vol_left
Label: L
Default Value: 1
Minimum: -20
Maximum: 10
Converter: Integer
Grid Position: 1, 6, 2, 4

Variable
ID: xlate_decim
Value: 2

Variable
ID: fs_base
Value: 250k

Variable
ID: fm_max_freq
Value: 2.26195

Variable
ID: fs_usrp
Value: 500k

Variable
ID: cur_freq
Value: 96.5M

Variable
ID: xlate_bandwidth
Value: 250k

Variable
ID: loop_bw
Value: 452.389m

UHD: USRP Source
Samp Rate (Sps): 500k
Ch0: Center Freq (Hz): 96.5M
Ch0: Gain (dB): 10

Frequency Xlating FIR Filter
Decimation: 2
Taps: firdes.low_pass(1, f...
Center Frequency: 0
Sample Rate: 500k

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 250k
Cutoff Freq: 80k
Transition Width: 35k
Window: Hamming
Beta: 6.76

PLL Freq Det
Loop Bandwidth: 452.389m
Max Freq: 2.26195
Min Freq: -2.26195

RTL-SDR Source
Sample Rate (sps): 500k
Ch0: Frequency (Hz): 96.5M
Ch0: Freq. Corr. (ppm): 0
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 40.2
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

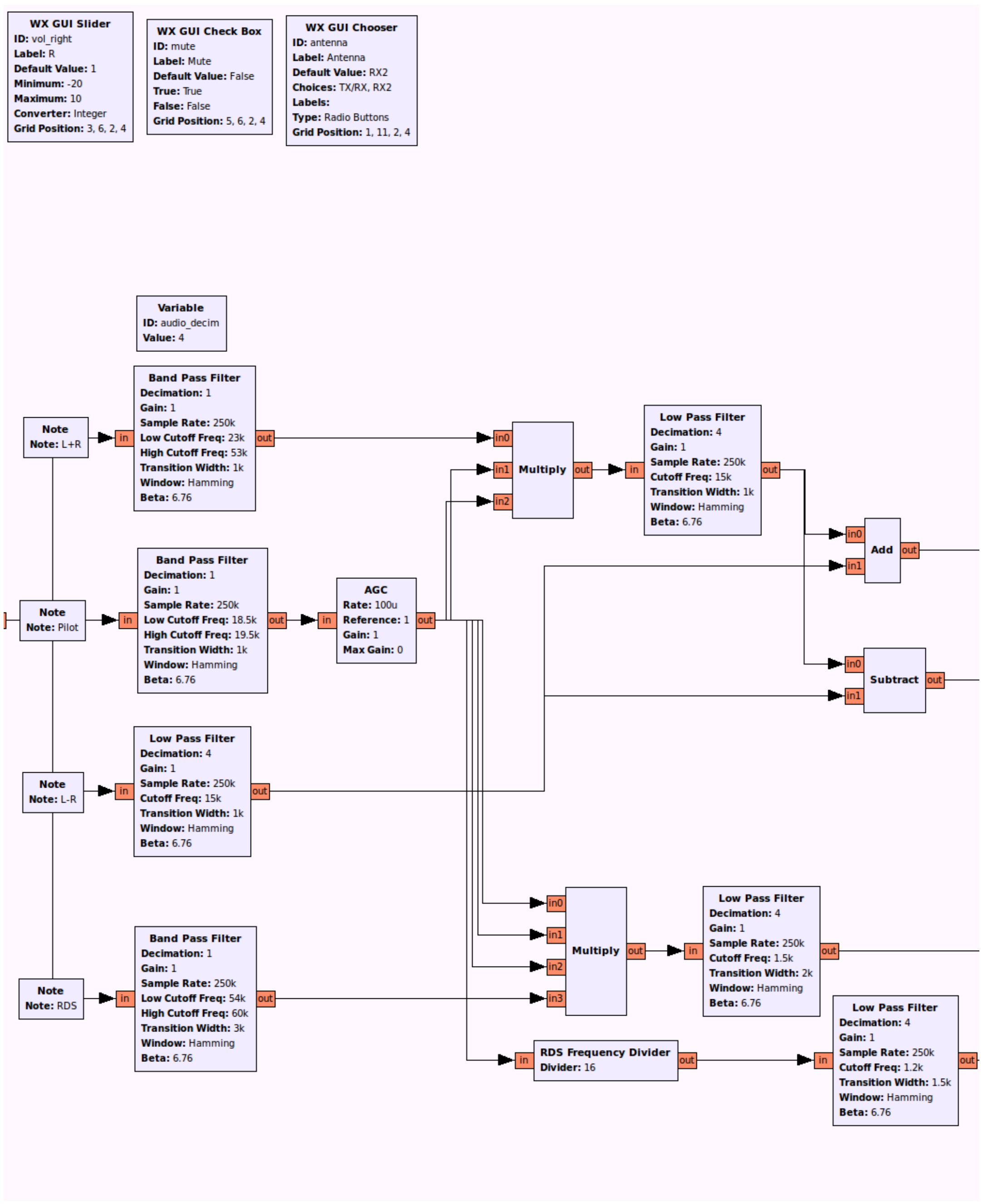
WX GUI FFT Sink
Title: RF Spectrum
Sample Rate: 500k
Baseband Freq: 96.5M
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): -40
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Window: Blackman-Harris
Grid Position: 8, 1, 1, 50
Freq Set Varname: None

Variable
ID: ref_level
Value: -40

WX GUI Waterfall Sink
Title: RF Waterfall
Sample Rate: 500k
Baseband Freq: 96.5M
Dynamic Range: 100
Reference Level: -40
Ref Scale (p2p): 2
FFT Size: 512
FFT Rate: 15
Grid Position: 7, 1, 1, 50
Freq Set Varname: None

Variable
ID: col_span
Value: 50

Figure 24. FM-RDBS receiver section 1


WX GUI Slider
ID: vol_right
Label: R
Default Value: 1
Minimum: -20
Maximum: 10
Converter: Integer
Grid Position: 3, 6, 2, 4

WX GUI Check Box
ID: mute
Label: Mute
Default Value: False
True: True
False: False
Grid Position: 5, 6, 2, 4

WX GUI Chooser
ID: antenna
Label: Antenna
Default Value: RX2
Choices: TX/RX, RX2
Labels:
Type: Radio Buttons
Grid Position: 1, 11, 2, 4

Variable
ID: audio_decim
Value: 4

Note
Note: L+R

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 250k
Low Cutoff Freq: 23k
High Cutoff Freq: 53k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Multiply

Low Pass Filter
Decimation: 4
Gain: 1
Sample Rate: 250k
Cutoff Freq: 15k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Add

Note
Note: Pilot

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 250k
Low Cutoff Freq: 18.5k
High Cutoff Freq: 19.5k
Transition Width: 1k
Window: Hamming
Beta: 6.76

AGC
Rate: 100u
Reference: 1
Gain: 1
Max Gain: 0

Subtract

Note
Note: L-R

Low Pass Filter
Decimation: 4
Gain: 1
Sample Rate: 250k
Cutoff Freq: 15k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Multiply

Low Pass Filter
Decimation: 4
Gain: 1
Sample Rate: 250k
Cutoff Freq: 1.5k
Transition Width: 2k
Window: Hamming
Beta: 6.76

Note
Note: RDS

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 250k
Low Cutoff Freq: 54k
High Cutoff Freq: 60k
Transition Width: 3k
Window: Hamming
Beta: 6.76

RDS Frequency Divider
Divider: 16

Low Pass Filter
Decimation: 4
Gain: 1
Sample Rate: 250k
Cutoff Freq: 1.2k
Transition Width: 1.5k
Window: Hamming
Beta: 6.76

Figure 25. FM-RDBS receiver section 2

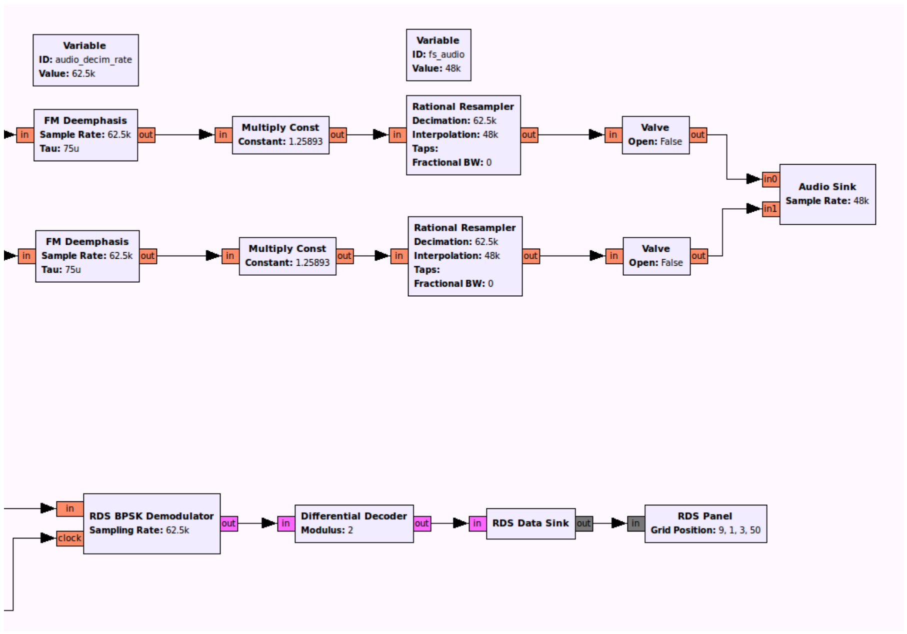
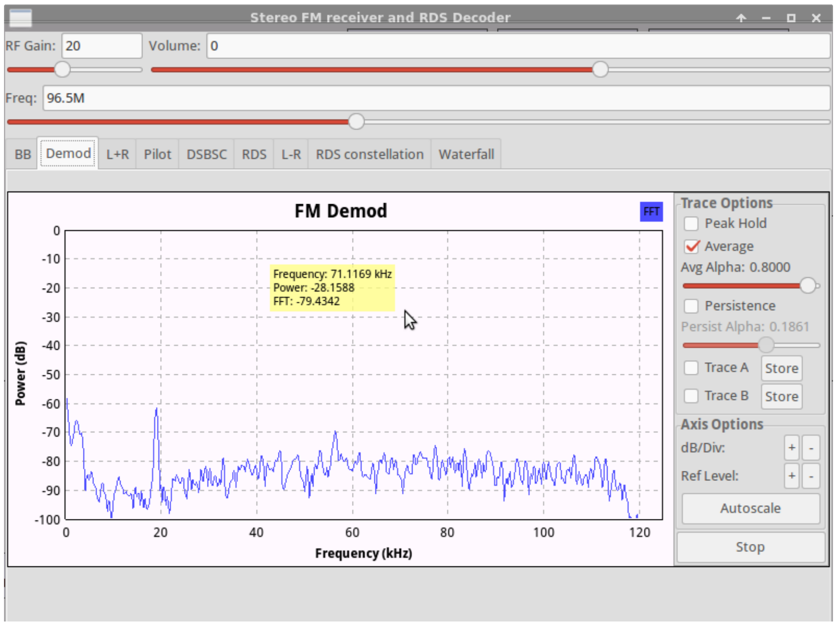


Figure 26. FM-RDBS receiver section 3

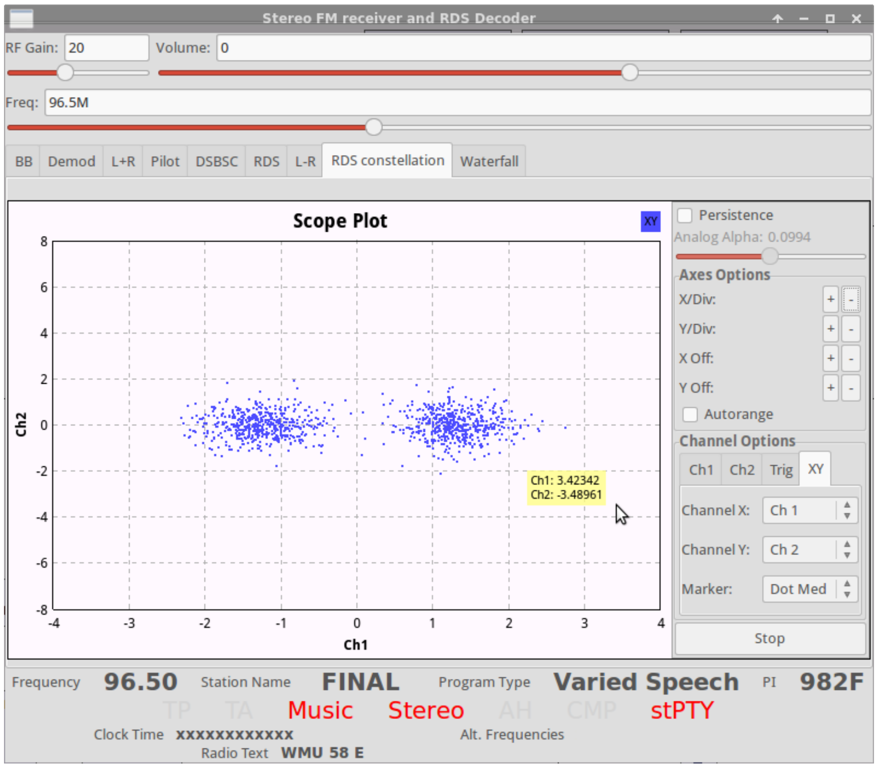


RF Gain: 20
Volume: 0
Freq: 96.5M

Frequency: 71.1169 kHz
Power: -28.1588
FFT: -79.4342

Avg Alpha: 0.8000
Persist Alpha: 0.1861

Figure 27. FM demodulated spectrum



RF Gain: 20 Volume: 0

Freq: 96.5M

Persistence
Analog Alpha: 0.0994
Autorange

Channel X: Ch 1
Channel Y: Ch 2
Marker: Dot Med

Ch1: 3.42342
Ch2: -3.48961

Frequency 96.50 Station Name FINAL Program Type Varied Speech PI 982F

TP TA Music Stereo AH CMP stPTY

Clock Time XXXXXXXXXXX Alt. Frequencies

Radio Text WMU 58 E

Figure 28. RDBS constellation plot

There are three components to an ADS – B system: ground station, airborne component and GPS satellite for position fixing. A representation of the ADS –B system is shown in

Figure 29. The ADS – B data frame consists of a 8μs long preamble with four pulses of 500 ns. Encoding is of Manchester type. Pulse position modulation (PPM) is used to transmit the data on a 1090 MHz carrier.

The frame structure for ADS – B is shown in Table 3. The USRP N210 device along with SBX daughtercard was used for receiving ADS – B signals. The system was only receiving the ADS – B broadcasts on 1090 MHz and was not interfering with the Mode – S transponder on 1030 MHz. The coding was done using Python and not GNURadio companion. The messages received were printed on the terminal. The geographic information was mapped to a mapping application using keyhole markup language (KML).

Table 3. ADS – B frame structure

DF11 format Total 56-bit	01011 5-bit	Capability 3-bit	ICAO address 24-bit	Parity 24-bit	
DF11 format Total 112-bit	01011 5-bit	Capability 3-bit	ICAO address 24-bit	ADS – B data 56-bit	Parity 24-bit

3.4.3 Simple Video Streamer

The objective for this demonstration was to create a simple video broadcasting setup using the USRP N210 and the SBX daughtercard. The experimental setup is as shown in Figure 32 and consists of a ThinkPad W510 laptop, PC desktop and two USRP N210s with SBX cards. Gstreamer was used to capture the webcam images to a Linux pipe which was then imported into GNURadio and modulated using Gaussian minimum shift keying. GMSK was chosen as this is the modulation type used in GSM. GMSK parameters were set to 2 samples/symbol with a sampling rate of 1 MSps.

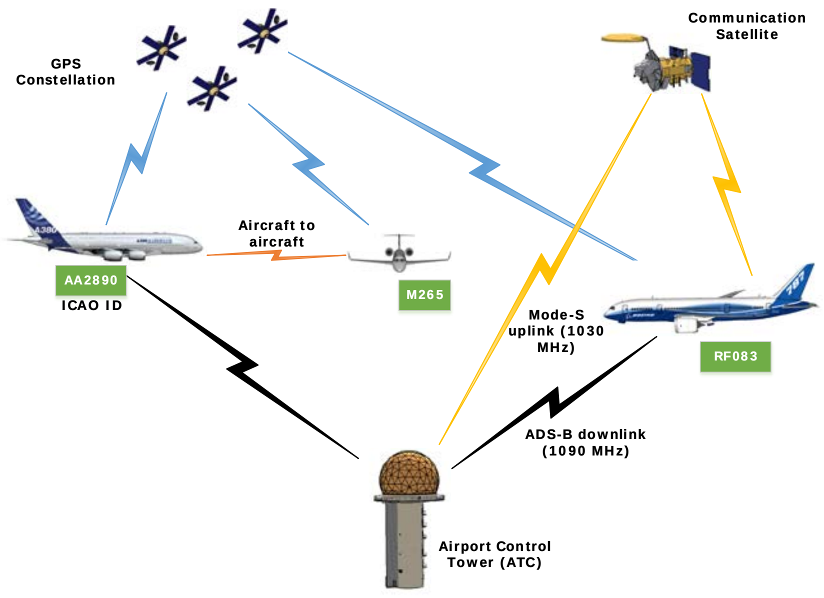
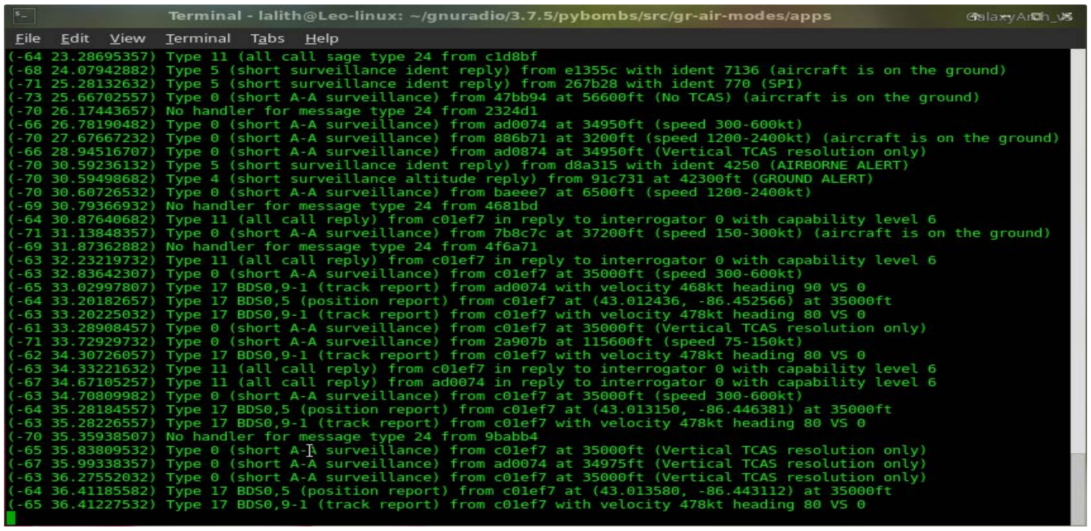


Figure 29. ADS – B system model


Terminal - lalith@Leo-linux: ~/gnuradio/3.7.5/pybombs/src/gr-air-modes/apps
File Edit View Terminal Tabs Help

```
(-64 23.28695357) Type 11 (all call sage type 24 from c1d8bf
(-68 24.07942882) Type 5 (short surveillance ident reply) from e1355c with ident 7136 (aircraft is on the ground)
(-71 25.28132632) Type 5 (short surveillance ident reply) from 267b28 with ident 770 (SPI)
(-73 25.66702557) Type 0 (short A-A surveillance) from 47bb94 at 56600ft (No TCAS) (aircraft is on the ground)
(-70 26.17443657) No handler for message type 24 from 2324d1
(-66 26.78190482) Type 0 (short A-A surveillance) from ad0074 at 34950ft (speed 300-600kt)
(-70 27.67667232) Type 0 (short A-A surveillance) from 886b71 at 3200ft (speed 1200-2400kt) (aircraft is on the ground)
(-66 28.94516707) Type 0 (short A-A surveillance) from ad0874 at 34950ft (Vertical TCAS resolution only)
(-70 30.59236132) Type 5 (short surveillance ident reply) from d8a315 with ident 4250 (AIRBORNE ALERT)
(-70 30.59498682) Type 4 (short surveillance altitude reply) from 91c731 at 42300ft (GROUND ALERT)
(-70 30.60726532) Type 0 (short A-A surveillance) from baeee7 at 6500ft (speed 1200-2400kt)
(-69 30.79366932) No handler for message type 24 from 4681bd
(-64 30.87640682) Type 11 (all call reply) from c0lef7 in reply to interrogator 0 with capability level 6
(-71 31.13848357) Type 0 (short A-A surveillance) from 7b8c7c at 37200ft (speed 150-300kt) (aircraft is on the ground)
(-69 31.87362882) No handler for message type 24 from 4f6a71
(-63 32.23219732) Type 11 (all call reply) from c0lef7 in reply to interrogator 0 with capability level 6
(-63 32.83642307) Type 0 (short A-A surveillance) from c0lef7 at 35000ft (speed 300-600kt)
(-65 33.02997807) Type 17 BDS0,9-1 (track report) from ad0074 with velocity 468kt heading 90 VS 0
(-64 33.20182657) Type 17 BDS0,5 (position report) from c0lef7 at (43.012436, -86.452566) at 35000ft
(-63 33.20225032) Type 17 BDS0,9-1 (track report) from c0lef7 with velocity 478kt heading 80 VS 0
(-61 33.28908457) Type 0 (short A-A surveillance) from c0lef7 at 35000ft (Vertical TCAS resolution only)
(-71 33.72929732) Type 0 (short A-A surveillance) from 2a907b at 115600ft (speed 75-150kt)
(-62 34.30726057) Type 17 BDS0,9-1 (track report) from c0lef7 with velocity 478kt heading 80 VS 0
(-63 34.33221632) Type 11 (all call reply) from c0lef7 in reply to interrogator 0 with capability level 6
(-67 34.67105257) Type 11 (all call reply) from ad0074 in reply to interrogator 0 with capability level 6
(-63 34.70809982) Type 0 (short A-A surveillance) from c0lef7 at 35000ft (speed 300-600kt)
(-64 35.28184557) Type 17 BDS0,5 (position report) from c0lef7 at (43.013150, -86.446381) at 35000ft
(-63 35.28226557) Type 17 BDS0,9-1 (track report) from c0lef7 with velocity 478kt heading 80 VS 0
(-70 35.35938507) No handler for message type 24 from 9babb4
(-65 35.83809532) Type 0 (short A-A surveillance) from c0lef7 at 35000ft (Vertical TCAS resolution only)
(-67 35.99338357) Type 0 (short A-A surveillance) from ad0074 at 34975ft (Vertical TCAS resolution only)
(-63 36.27552032) Type 0 (short A-A surveillance) from c0lef7 at 35000ft (Vertical TCAS resolution only)
(-64 36.41185582) Type 17 BDS0,5 (position report) from c0lef7 at (43.013580, -86.443112) at 35000ft
(-65 36.41227532) Type 17 BDS0,9-1 (track report) from c0lef7 with velocity 478kt heading 80 VS 0
```

Figure 30. ADS – B broadcast messages

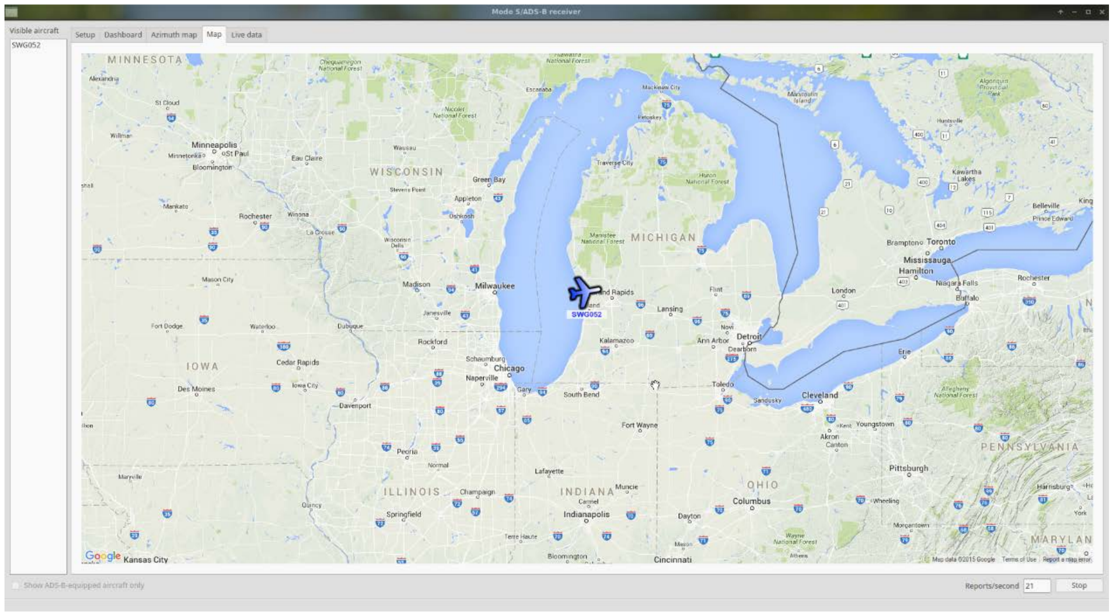


Figure 31. ADS – B KML data on map

The packet encoder block on the GNURadio had a payload size of 4000. It adds 12 Bytes as overhead resulting in a bit rate of 498.5 Kbps for the video. Gstreamer options were set so that the framerate is 30fps with a video size of 640x480. Video was encoded to H.264/MPEG-TS. Transmission was done on 915 MHz industrial, scientific and medical band (ISM). The experimental setup shows the video image that was received on the desktop. The spectrum analyzer plot showing the waterfall, the spectrum and the constellation plot is shown in Figure 35.

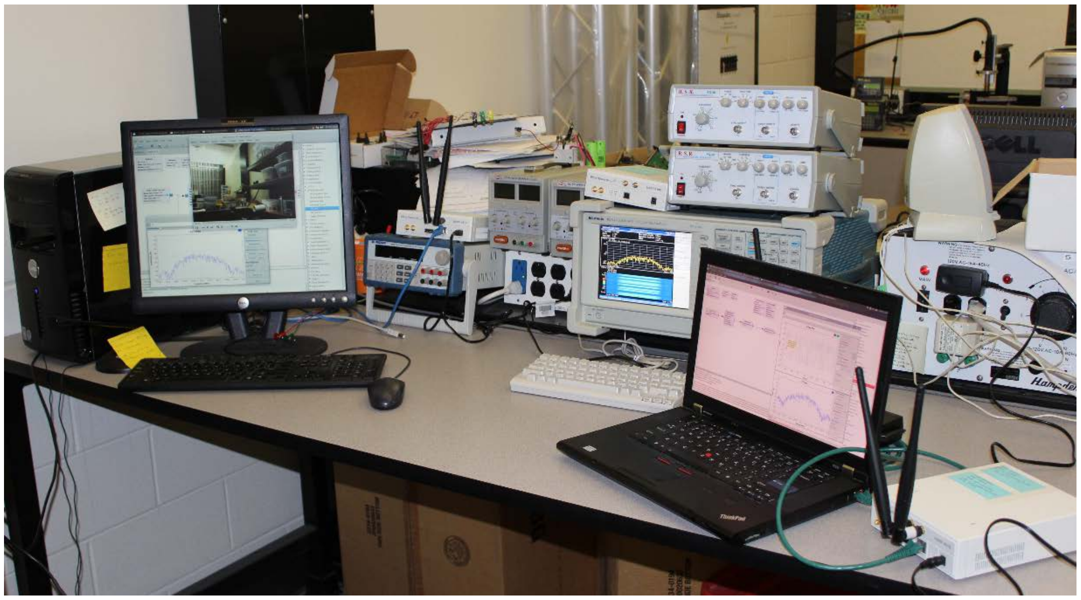
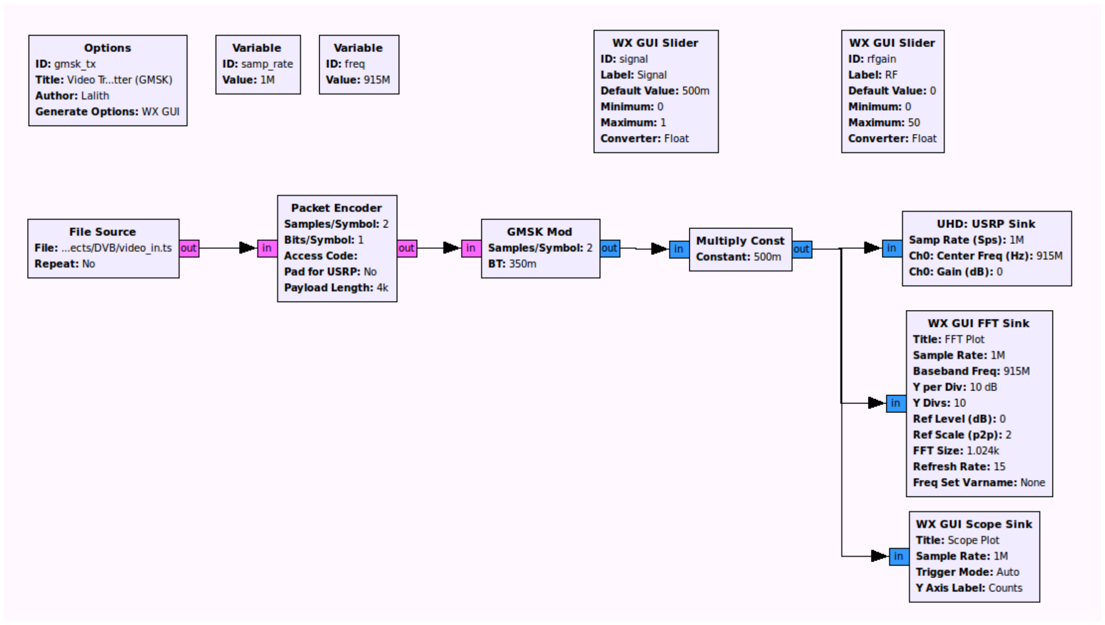


Figure 32. Video streaming experimental setup



Options
ID: gmsk_tx
Title: Video Tr...tter (GMSK)
Author: Lalith
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 1M

Variable
ID: freq
Value: 915M

WX GUI Slider
ID: signal
Label: Signal
Default Value: 500m
Minimum: 0
Maximum: 1
Converter: Float

WX GUI Slider
ID: rfgain
Label: RF
Default Value: 0
Minimum: 0
Maximum: 50
Converter: Float

File Source
File: ...ects/DVB/video_in.ts
Repeat: No

Packet Encoder
Samples/Symbol: 2
Bits/Symbol: 1
Access Code:
Pad for USRP: No
Payload Length: 4k

GMSK Mod
Samples/Symbol: 2
BT: 350m

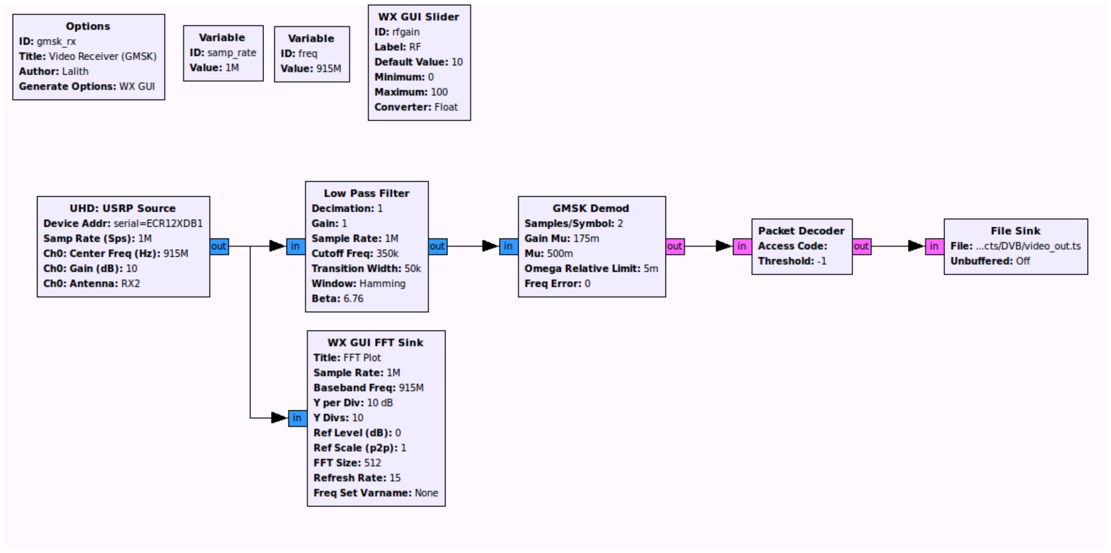
Multiply Const
Constant: 500m

UHD: USRP Sink
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 915M
Ch0: Gain (dB): 0

WX GUI FFT Sink
Title: FFT Plot
Sample Rate: 1M
Baseband Freq: 915M
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

WX GUI Scope Sink
Title: Scope Plot
Sample Rate: 1M
Trigger Mode: Auto
Y Axis Label: Counts

Figure 33. Video streaming transmitter


Options
ID: gmsk_rx
Title: Video Receiver (GMSK)
Author: Lalith
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 1M

Variable
ID: freq
Value: 915M

WX GUI Slider
ID: rfgain
Label: RF
Default Value: 10
Minimum: 0
Maximum: 100
Converter: Float

UHD: USRP Source
Device Addr: serial=ECR12XDB1
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 915M
Ch0: Gain (dB): 10
Ch0: Antenna: RX2

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Cutoff Freq: 350k
Transition Width: 50k
Window: Hamming
Beta: 6.76

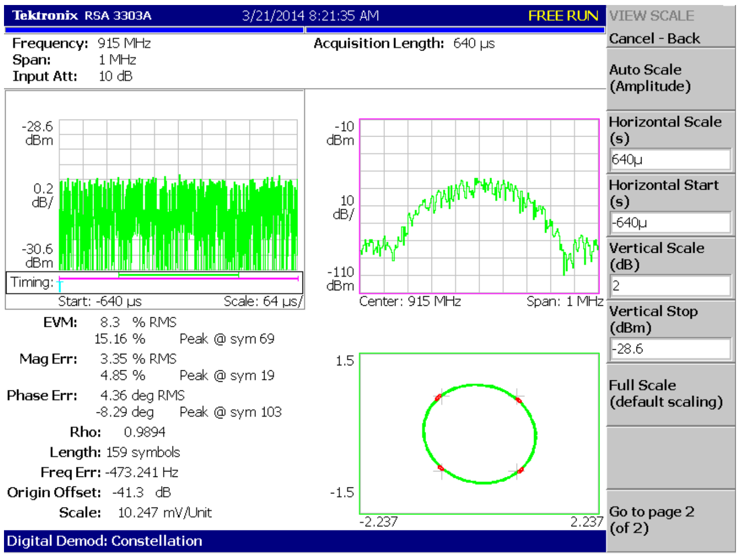
GMSK Demod
Samples/Symbol: 2
Gain Mu: 175m
Mu: 500m
Omega Relative Limit: 5m
Freq Error: 0

Packet Decoder
Access Code:
Threshold: -1

File Sink
File: ...cts/DVB/video_out.ts
Unbuffered: Off

WX GUI FFT Sink
Title: FFT Plot
Sample Rate: 1M
Baseband Freq: 915M
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 1
FFT Size: 512
Refresh Rate: 15
Freq Set Varname: None

Figure 34. Video streaming receiver


Tektronix RSA 3303A 3/21/2014 8:21:35 AM FREE RUN

Frequency: 915 MHz
Span: 1 MHz
Input Att: 10 dB
Acquisition Length: 640 µs

Timing:
Start: -640 µs Scale: 64 µs/
Center: 915 MHz Span: 1 MHz

EVM: 8.3 % RMS
15.16 % Peak @ sym 69
Mag Err: 3.35 % RMS
4.85 % Peak @ sym 19
Phase Err: 4.36 deg RMS
-8.29 deg Peak @ sym 103
Rho: 0.9894
Length: 159 symbols
Freq Err: -473.241 Hz
Origin Offset: -41.3 dB
Scale: 10.247 mV/Unit

Digital Demod: Constellation

VIEW SCALE
Cancel - Back
Auto Scale (Amplitude)
Horizontal Scale (s): 640µ
Horizontal Start (s): -640µ
Vertical Scale (dB): 2
Vertical Stop (dBm): -28.6
Full Scale (default scaling)
Go to page 2 (of 2)

Figure 35. Spectrum analyzer plot for video streaming demonstration

3.5 Summary

This chapter provided an overview of the three existing software defined radio platforms, Sora, WARP and BEE2 that are used for research. The architectures of the platforms in regards to their RF front ends, processing elements and software components were also discussed. A brief description of their advantages and disadvantages was also discussed. Incremental advances in hardware and software technologies are unavoidable and every effort must be taken at the design stage to create a modular base architecture that can absorb these advances. In this effect a generic reference design was defined, identifying key sub elements and their design issues. To help validate the reference design, a prototype SDR platform was built. This chapter focused on validating the performance of the RF front end using real world applications. FM, ADS – B and video streaming designs were implemented in GNURadio and their result verified.

CHAPTER IV

CUDA BASED PARALLEL PROCESSING FOR COMMUNICATIONS

While FPGA based real-time signal processing can provide a degree of reconfiguration or programmability in support of a generalized SDR system or architecture, they are typically used for dedicated high data and clock rate signal processing where GPP are not able to meet the system requirements. While multi-core GPP have emerged to provide greater computation capability, the most promising increase in computation capability that has been recognized in the super-computing field is GPUs. With the availability and continued advancements in GPU based parallel processing, they may prove to be useful in SDR application as software reconfigurable and rapidly reassigned computation elements. While GPUs used as coprocessors have been demonstrated to reduce GPP workload and increase throughput in various applications, for SDR applications they may also be able to support computations or algorithms typically performed in ASICs or FPGAs. When considering possible candidate algorithms a polyphase filter bank channelizer can be readily identified as a candidate.

4.1 Introduction

Base stations for cellular mobile communication systems require simultaneous processing of data streams from multiple users. The base station receiver must be able to down convert, and demodulate multiple evenly spaced frequency division multiplexed (FDM) channels as shown in Figure 36[56]. The DSP based technique called channelization is used to separate these signals from the single wideband communication stream. The channelizer performs three essential tasks: signal mixing or down conversion, filtering and sample rate reduction or decimation. The simplest form of the channelizer consists of a bank of N dual conversion sub-receivers, one for each channel. This approach incurs significant costs in area, power and complexity. Moreover most of the sub-receivers may be idle at any given time resulting in poor utilization of resources.

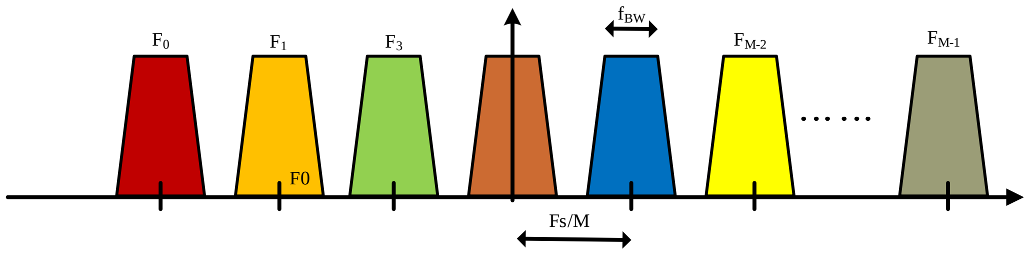


Figure 36. Input FDM spectrum to base station receiver

An alternate implementation introduced by [57, 58] performs the channelization through a process called polyphase N-path filter bank (PFB). The PFB performs the filtering operation in the channelizer and a discrete Fourier transform (DFT) is used for mixing the signals up or down. The polyphase filter bank offers significant advantages to the conventional channelizer in terms of cost and power consumption, due to overall reduction in system resources required for the channelization operation.

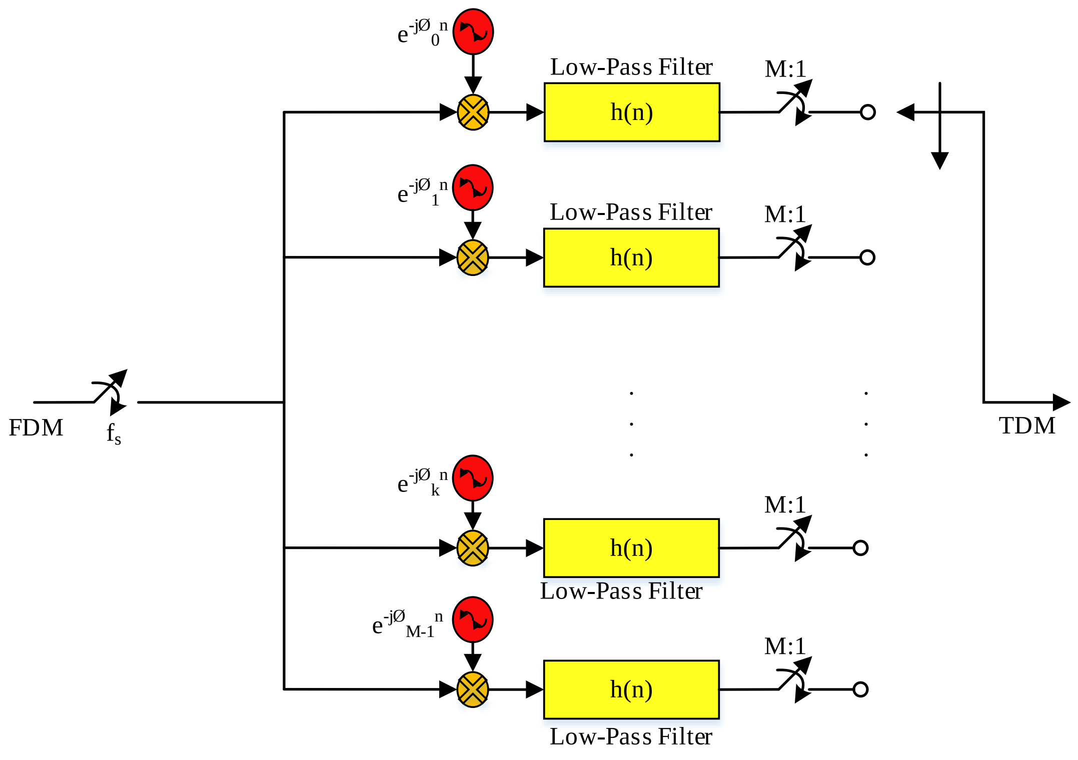


Figure 37. Conventional channelizer

In this chapter we develop a novel GPU-based polyphase channelizer to demonstrate the application of GPU technology to SDR systems. We designed a high throughput, reduced complexity PFB, capable of performing parallel processing of many channels.

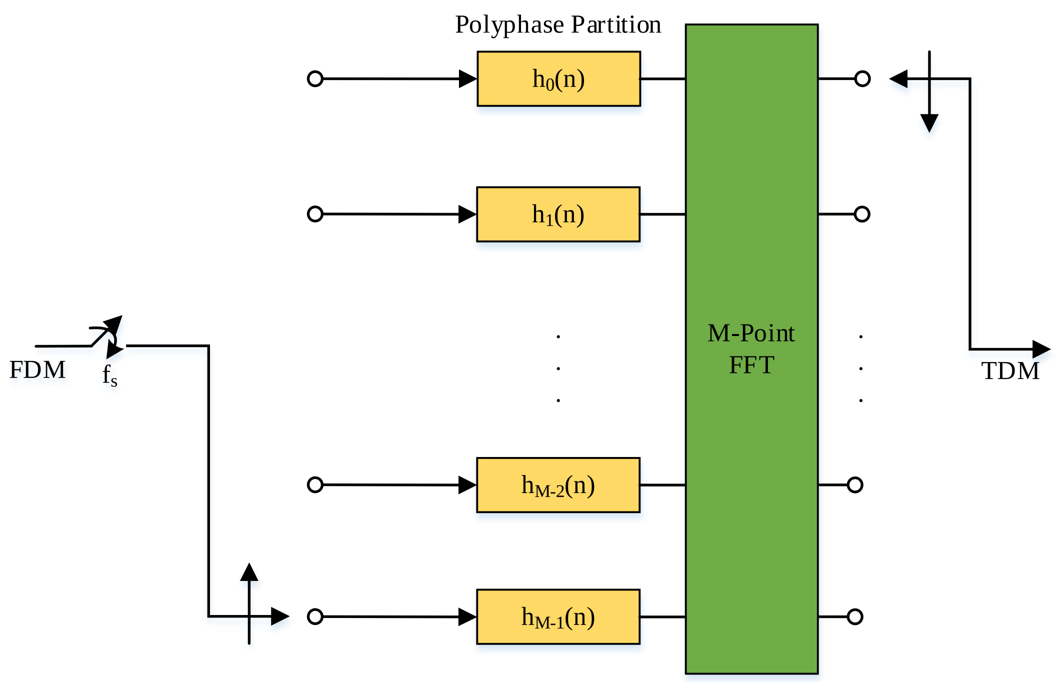


Figure 38. Polyphase filter bank (PFB) channelizer

4.2 Polyphase Channelizers - Theory and Operation

In order to map the polyphase channelizer algorithm onto a GPU and exploit available parallelism, there is a need to fully understand the mathematical theory and operation. In this section, the system of equations governing the polyphase channelizer will be derived before focusing on its design and implementation on a GPU. The block diagram of a single channel conventional receiver is shown in Figure

39. This structure performs the standard operations of down conversion or complex mixing of the channel of interest to a zero IF or baseband, low pass filtering to reject images from the mixing process and downsampling to reduce the sampling rate.

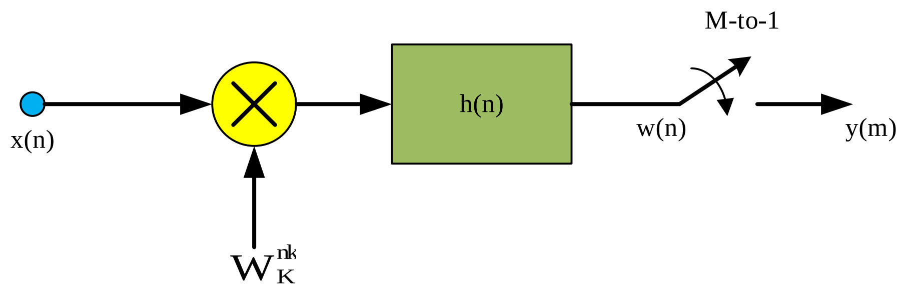


Figure 39. Single channel conventional channelizer

Let us assume k represents equally spaced frequency channels that exist in the digitized spectrum. Then the expression for the time series output $w_k(n)$ from the k-th channel, prior to resampling consists of a complex mixing and a simple convolution and is given by

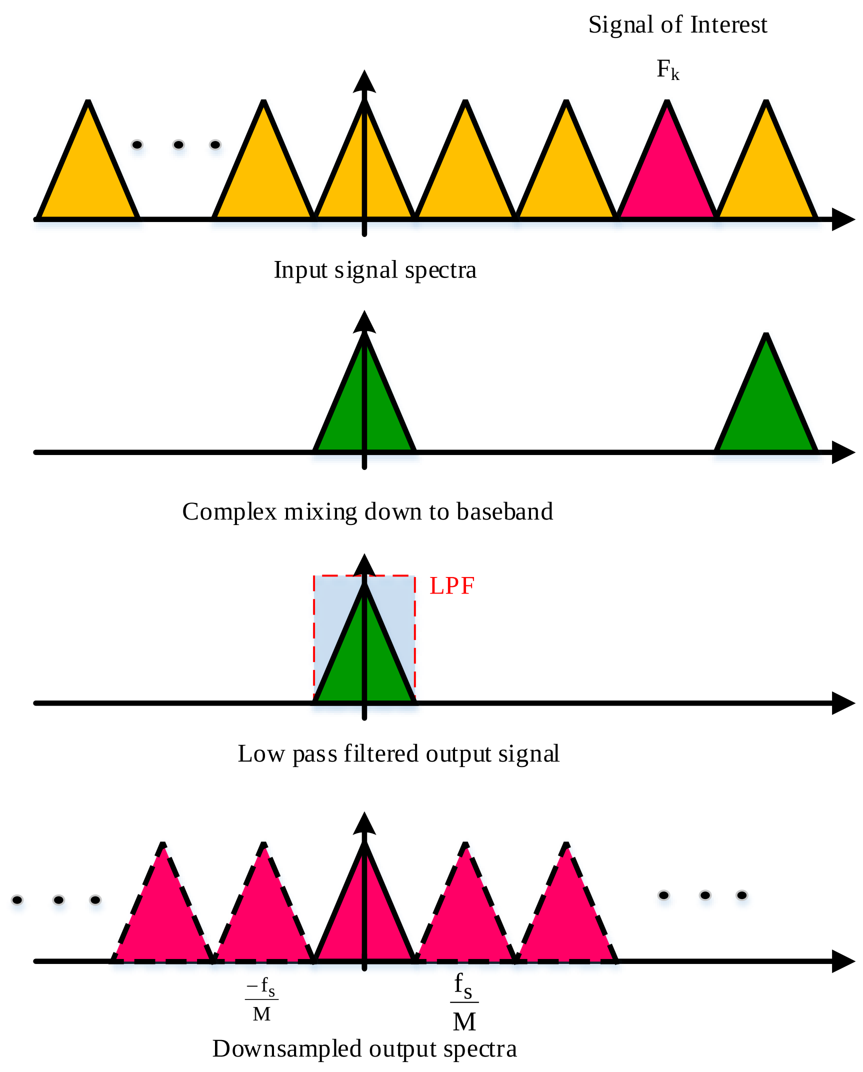
$$w_k(n) = \left[x(n)W_K^{nk}\right] * h(p) \tag{1}$$

where, $W_K^{nk} = e^{-\frac{j2\pi nk}{K}}$. Eq. (1) is a convolution operation that can be rewritten as

$$w_k(n) = \sum_{p=-\infty}^{\infty} h(p)x(n-p)e^{-\frac{j2\pi(n-p)k}{K}} \tag{2}$$

Only one sample out of M samples is retained at the output of the resampler. Hence when decimation is incorporated, its output $y(m)$ can be written as

$$y_k(m) = w_k(m.M) = \sum_{p=-\infty}^{\infty} h(p)x(m.M-p)e^{-\frac{j2\pi(m.M-p)k}{K}} \qquad (3)$$



Signal of Interest
F_k
Input signal spectra
Complex mixing down to baseband
LPF
Low pass filtered output signal
$\frac{-f_s}{M}$
$\frac{f_s}{M}$
Downsampled output spectra

Figure 40. Spectral description of the single channel channelizer

Let us assume that we want to implement this single channel conventional channelizer using digital hardware. We know that the output from the complex mixer is also complex. Let them be represented by two time series; in-phase $I(n)$ and quadrature $Q(n)$. Since the filter $h(n)$ is real, the filtering operation can be performed by two identical set of filters; one operating on the real time series $I(n)$ and the other on $Q(n)$. The filtering process can be performed by multiply-add operation between the data samples and filter coefficients stored in two registers. A conceptual digital filter is shown in

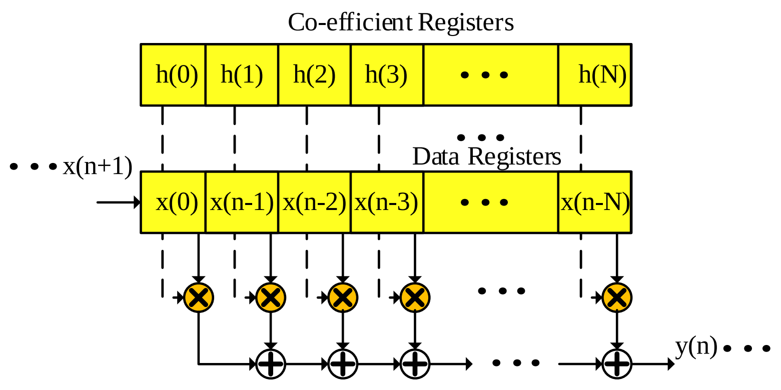


Figure 41. Conceptual digital filter

The output down conversion can also be performed by simply discarding the unwanted output samples from memory. There is still the task of implementing the complex heterodyne. This can be accomplished by direct digital synthesizer and hardware multipliers or by using CORDIC. However when there are k channels, the amount of resources required is quite significant. There is a simpler and more efficient technique of implementing Eq. (3). Let us start first by re-writing this equation,

$$y_k(m) = \sum_{p=-\infty}^{\infty} h(p)x(m.M - p)e^{-\frac{j2\pi(m.M-p)k}{K}}$$

For this equation, define a causal filter of length $k\lambda$ where

$$h(p) = \begin{cases} 0 & p < 0 \\ h(p) & 0 \le p < k\lambda \\ 0 & k\lambda \le p \end{cases} \tag{4}$$

If we introduce a change of variable, $p = rK + \rho$, where r goes from 0 to $\lambda - 1$ and ρ goes from 0 to $K - 1$, the filter becomes a two-dimensional structure in terms of K rows and λ columns.

$$h(rK + \rho) \Rightarrow \begin{bmatrix} h(0) & h(K) & \cdots & h\big((\lambda-1)K\big) \\ h(1) & h(K+1) & \cdots & h\big((\lambda-1)K+1\big) \\ \vdots & \vdots & \ddots & \vdots \\ h(K-1) & h(2K+1) & \cdots & h(\lambda K - 1) \end{bmatrix} \tag{5}$$

And the summation becomes a double summation in r and ρ. Substituting $p \rightarrow\ rK + \rho$, $y_k(m)$ can be written as

$$y_k(m) = \sum_{\rho=0}^{K-1}\sum_{r=0}^{\lambda-1} h(rK+\rho)x(m.M - rK - \rho)e^{-\frac{j2\pi(m.M-rK-\rho)k}{K}} \tag{6}$$

$$= \sum_{\rho=0}^{K-1}\sum_{r=0}^{\lambda-1} h(rK+\rho)x(m.M - rK - \rho)e^{-\frac{j2\pi(m.M-\rho)k}{K}}e^{\frac{j2\pi rKk}{K}} \tag{7}$$

but $e^{\frac{j2\pi rKk}{K}} = 1$. Therefore, identifying separate summation elements

$$y_k(m) = \sum_{\rho=0}^{K-1} e^{-\frac{j2\pi(m.M-\rho)k}{K}} \sum_{r=0}^{\lambda-1} h(rK+\rho)x(m.M-rK-\rho) \qquad (8)$$

If we define $W_K^{n-k} = e^{-\frac{j2\pi nk}{K}}$ as the twiddle factor, and $h_{\rho}(r) = h(rK+\rho)$, then

$$x_\rho(mM-rK) = x(mM-rK-\rho)$$

$$= \begin{bmatrix} x(mM) & x(mM-K) & \cdots & x(mM-(\lambda-1)K) \\ x(mM-1) & x(mM-K-1) & \cdots & x(mM-(\lambda-1)K-1) \\ \vdots & \vdots & \ddots & \vdots \\ x(mM-K+1) & x(mM-2K+1) & \cdots & x(mM-\lambda K+1) \end{bmatrix} \qquad (9)$$

This is because the input signal is also causal and hence can be described in a two-dimensional structure similar to the filter coefficients. The generalized form of the equation can now be written as

$$y_k(m) = W_K^{mkM} \sum_{\rho=0}^{K-1} W_K^{-\rho k} \sum_{r=0}^{\lambda-1} h_\rho(r) x_\rho(m.M-rK) \qquad (10)$$

In Eq.(10), it should be noticed that the term $\sum_{\rho=0}^{K-1} W_K^{-\rho k}$ represents the inverse discrete Fourier transform. So the processing elements consists of a polyphase filter with dimensions $K \times \lambda$, an inverse discrete Fourier transform, and a post multiplication factor.

If on the other hand we assume, $p = rK - \rho$, then

$$y_k(m) = \sum_{\rho=1}^{K} \sum_{r=1}^{\lambda} h(rK-\rho)x(m.M-rK+\rho)e^{-\frac{j2\pi(m.M-rK+\rho)k}{K}} \qquad (11)$$

$$y_k(m) = \sum_{\rho=1}^{K} e^{-\frac{j2\pi(m.M+\rho)k}{K}} \sum_{r=1}^{\lambda} h(rK-\rho)x(m.M-rK+\rho) \tag{12}$$

$$y_k(m) = W_K^{mkM} \sum_{\rho=1}^{K} W_K^{\rho k} \sum_{r=1}^{\lambda} h_\rho(r)x_\rho(m.M-rK) \tag{13}$$

where

$$h_\rho(r) = h(rK-\rho) \tag{14}$$

$$x_\rho(mM-rK) = x(mM-rK+\rho) \tag{15}$$

In Eq. 13, note that the term $\sum_{\rho=1}^{K} W_K^{\rho k}$ is in the form of discrete Fourier transform. To make implementation easier we can write $\sum_{\rho=1}^{K} W_K^{\rho k} = W_K^k \sum_{\rho=0}^{K-1} W_K^{\rho k}$. Eq. 13 then becomes

$$y_k(m) = W_K^{mkM} W_K^k \sum_{\rho=0}^{K-1} W_K^{\rho k} \sum_{r=1}^{\lambda} h_\rho(r)x_\rho(m.M-rK) \tag{16}$$

4.2.1 Special Cases

For Nyquist sampling considerations, the number of frequencies K and decimation rate M are related. This can be described in a ratio as

$$a = \frac{K}{M}$$

In interpretation, the frequencies for the demodulation are considered frequency bins based on the input sample rate of the data. If the data is complex, the sample rate and frequency content may be equivalent based on complex Nyquist sampling. If the original frequency band is evenly divided into K segments, the decimation rate and filter bandwidths must be appropriately defined to avoid aliasing in the resulting frequency bins. For example, if $M = K$, the bin width and decimated signal bandwidth are the same, whether the filter allows aliasing or not. If the bins frequency content (passband bandwidth) is to be equal to the bin width, then the output sample rate would want to allow a bandwidth greater than the bin width. As an example $2M = K$ would provide a complex sampling rate of $\frac{f_s}{M} = 2\left(\frac{f_s}{K}\right)$ or twice the $\frac{f_s}{K}$ bin spacing. This would facilitate an $\frac{f_s}{K}$ bandwidth filter passband, transition bands and stop bands within the $\frac{2f_s}{K}$ output signal frequency band. This concept readily supports the processing of closely spaced RF channels such as radio or television.

If the $\frac{K}{M}$ substitution is performed, Eq. (10) and Eq. (16) reduce to

$$y_k(m) = W_a^{mM} \sum_{\rho=0}^{K-1} W_K^{-\rho k} \sum_{r=0}^{\lambda-1} h_\rho(r) x_\rho(m.M - rK)$$

and

$$y_k(m) = W_K^{mkM} W_K^{k} \sum_{\rho=0}^{K-1} W_K^{\rho k} \sum_{r=1}^{\lambda} h_\rho(r) x_\rho(m.M - rK)$$

When $\boldsymbol{K = M}$

Maximal rate decimation for an input signal would define $a = 1$ or $K = M$. This will make the initial phase multiplication term $W_K^{mkM} = 1$. If we also ignore the constant phase term W_K^k in the $p = rM - \rho$ derivation, we end up with the following set of equations

Table 4. Summary of the equations for polyphase filter

$p = rM + \rho$	$p = rM - \rho$
$y_k(m) = \sum_{\rho=0}^{K-1} W_K^{-\rho k} \sum_{r=0}^{\lambda-1} h_\rho(r) x_\rho(m.M - rK)$ Where $h_\rho(r) = h(rK + \rho)$ $x_\rho(mM - rK) = x(mM - rK - \rho)$	$y_k(m) = \sum_{\rho=0}^{K-1} W_K^{\rho k} \sum_{r=1}^{\lambda} h_\rho(r) x_\rho(m.M - rK)$ Where $h_\rho(r) = h(rK - \rho)$ $x_\rho(mM - rK) = x(mM - rK + \rho)$

Block diagrams for the polyphase filter with IDFT implementation for $p = rM + \rho$ and DFT implementation for $r = rM - \rho$ are shown below (see Figure 42 and Figure 43). Notice the change in direction of the commutator in the two diagrams.

When $K \neq M$

Another special case is when K and M are arbitrary values and are not equal. In this case the polyphase filter structure still remains unchanged, whereas the input matrix $x(mM - rK - \rho)$ from Eq. (9) has to be re-written as shown below.

$$h(rK + \rho) \Rightarrow \begin{bmatrix} h(0) & h(K) & \cdots & h\big((\lambda - 1)K\big) \\ h(1) & h(K + 1) & \cdots & h\big((\lambda - 1)K + 1\big) \\ \vdots & \vdots & \ddots & \vdots \\ h(K - 1) & h(2K + 1) & \cdots & h(\lambda K - 1) \end{bmatrix}$$

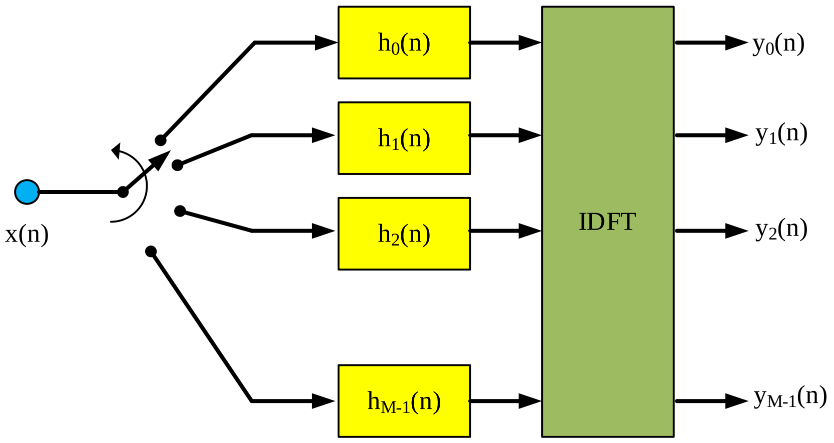


Figure 42. Polyphase filter with IDFT implementation

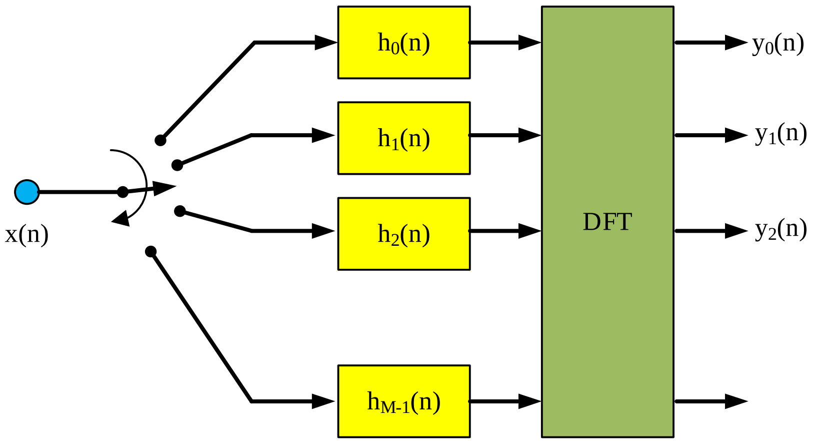


Figure 43. Polyphase filter with DFT implementation

$$x(mM - rK - \rho)$$

$$= \begin{bmatrix} x(mM-0) & x(mM-K-0) & \cdots & x(mM-(\lambda-1)K-0) \\ x(mM-1) & x(mM-K-1) & \cdots & x(mM-(\lambda-1)K-1) \\ \vdots & \vdots & \ddots & \vdots \\ x(mM-K+1) & x(mM-2K+1) & \cdots & x(mM-\lambda K+1) \end{bmatrix} \quad (17)$$

The output time series $y_k(m)$ can then be written as

$$y_k(m) = W_K^{mkM} \sum_{\rho=0}^{K-1} W_K^{-\rho k} \sum_{r=0}^{\lambda-1} h(rK+\rho)x(m.M - rK - \rho)W_K^{-rKk} \quad (18)$$

We can define the polyphase elements $PP_\rho(m)$ as

$$PP_\rho(m) = \sum_{r=0}^{\lambda-1} h(rK+\rho)x(mM - rK - \rho)\,, \quad for\ k = 0{:}\,K-1 \quad (19)$$

If we pay attention to the input data matrix it can be seen that it needs to be re-formed for each iteration, the oldest M samples are removed and then the newest M samples added. This can be accomplished by using a circular buffer in which the starting point of the data is identified by a pointer and offsets of K from the pointer are computed by “modulo-$K\lambda$” addition at each iteration.

$$y_{k(m)} = \sum_{\rho=0}^{K-1} W_K^{-\rho k} W_K^{mMk} \left[PP_\rho(m)\right] \quad (20)$$

The twiddle factor matrix for the Fourier transform also has to be rewritten as

$$W_K^{-nk} \Longrightarrow \begin{bmatrix} W_K^{-0} & W_K^{-0} & W_K^{-0} & \cdots & W_K^{-0} \\ W_K^{-0} & W_K^{-1} & W_K^{-2} & \cdots & W_K^{-(K-1)} \\ W_K^{-0} & W_K^{-2} & W_K^{-3} & \cdots & W_K^{-2(K-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W_K^{-0} & W_K^{-(K-1)} & W_K^{-2(K-1)} & \cdots & W_K^{-(K-1)(K-1)} \end{bmatrix} \tag{21}$$

Where each column will provide a different "k-based" multiplication or shifting from the previous column. An alternate method of achieving this transformation is to combine the twiddle factor with a complex shift.

$$y_{k(m)} = \sum_{\rho=0}^{K-1} W_K^{-(\rho-mM)k}\left[PP_\rho(m)\right] \tag{22}$$

$$y_{k(m)} = \sum_{r=-mM}^{K-1-mM} W_K^{-rk}[PP_{r+mM}(m)] \tag{23}$$

Using the forms of Eq. (22), it can be seen that the new summation is a $-mM$ circularly shifted version of the previous Fourier matrix. As the order of summation in a Fourier transform is arbitrary and the "twiddle factors" form a circular symmetry ($-mM{:}-1$ is the same as $K-mM{:}K-\ 1$), this equation could also be solved as

$$y_{k(m)} = \sum_{r=0}^{K-1} W_K^{-rk}\left[PP_{(r+mM)_K}(m)\right] \tag{24}$$

This implies that the data is circular shifted by mM prior to performing the transformation. If this is done, than no "post-multiplication" is required.

A block diagram of the filter bank implementation for the two special cases is shown in Figure 44 and Figure 45.with output buffer to change the final sample rate.

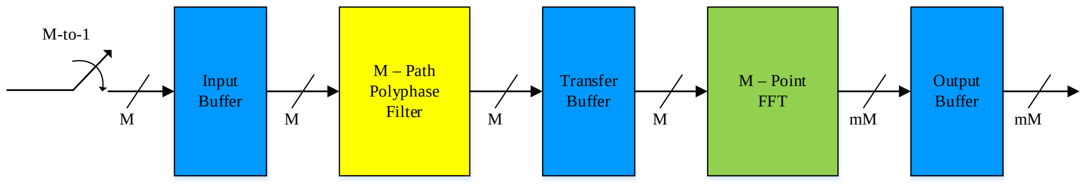


Figure 44. Maximally decimated filter bank structure

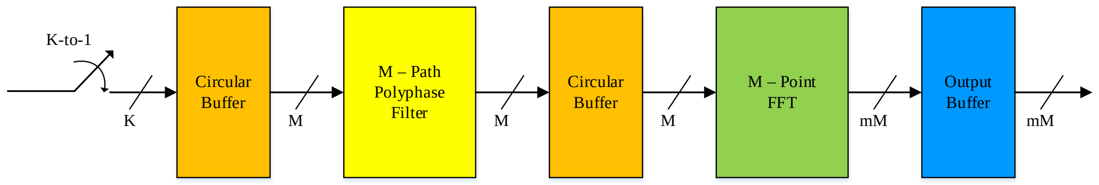


Figure 45. Partially decimated filter bank structure

4.3 CUDA Best Practices

In order to achieve good performance on the CUDA GPU, it is essential to understand the architecture but also perform code optimization. We have already discussed in detail the CUDA architecture in chapter 2, so this section will rather focus on practices that were followed to achieve optimum performance.

4.3.1 Memory Optimizations

Memory is the most important area of optimization when developing applications with CUDA. The goal here is to maximize use of fast types of memory and minimize usage of slow access memories. The host (CPU) and the device (GPU) have separate memories. Data read/written on the device must be copied to/from the host over PCIe. This data transfer is expensive because of overheads associated with each transfer. So it is essential to minimize data transfer between the host and the device and to group small transfers into one large transfer. The device to device PCIe transfer has much higher theoretical memory bandwidth than the host to device transfer. Therefore, intermediate results between kernel calls should be stored on the device memory, operated on and destroyed without being mapped to the host memory.

If the data sizes are small, it is also advisable to use page-locked or pinned memory instead of non-pageable memory. Pinned memory allocated using *cudaHostAlloc()* allows data transfers between host and device to achieve highest bandwidth. When possible, overlap data transfers with computation. This can be achieved using *cudaMemcpyAsync()* and streams instead of the regular *cudaMemcpy()*. When using integrated GPUs, zero copy feature will provided performance gains by avoiding the unnecessary host to device transfers (integrated GPUs share memory with the host).

Finally shared memory has higher bandwidth and lower latency than global memory and should be used where possible. When using global memory, accesses should be coalesced so as to minimize the number of transactions. When using constant memory space, make sure the threads of a warp access the same location as this will make constant memory as fast as register memory.

4.3.2 Configuration Optimizations

When dividing the work amongst the CUDA multiprocessors, it is essential to balance the workload and keep the GPU busy. This is the best way to hide memory latencies that will result in performance degradation. The metric to gauge hardware utilization is occupancy, defined as the ratio of number of active warps per multiprocessor to the maximum number of possible active warps. It should be mentioned here there is a point above which increasing occupancy does not necessarily lead to increase in performance.

On devices that support concurrent kernel execution, streams can be used to launch multiple kernels simultaneously. Threads on the GPU are executed in groups of 32, called a warp. For computing efficiency and coalescing, number of threads per block should be a multiple of 32.

4.3.3 Optimizing Instructions

Single precision floats offer the best performance and should be used for all arithmetic operations. Both division and modulo operations are expensive and should be replace by bitwise operations where possible. Threads in a warp execute an instruction in lock step albeit on different data. So it is essential to avoid branching statements as this would lead to diverging (different execution paths). If branching is unavoidable, then ensure that there is no intra-warp branching. For example, using thread index for branching will cause diverging warps. Instead, dividing the thread index by warp size will cause the controlling condition to be aligned with the warps.

4.4 GPU Based Maximally Decimated Polyphase Channelizer

This section will describe how the polyphase channelizer was mapped onto the GPU. Two different version of the polyphase channelizer were implemented. The first algorithm was for the maximally decimated case when $K = M$. This algorithm will be called PFB1. The algorithm will exploit the inherent parallelism found in the polyphase structure and use the GPU to accelerate the commutator and filter operations. The final FFT structure is implemented using CUDAs own FFT library called cuFFT. cuFFT is part of NVIDIA’s signal processing libraries and is highly optimized for use in CUDA. It is based on the popular fastest Fourier transform in the West (FFTW) implementation.

Traditional implementation of the polyphase channelizer on both single threaded microprocessors and CUDA GPUs mimicked the commutator operation by loading M input samples at a time into the circular input buffer and then performing the filter inner product in parallel. This would require a for-loop to cycle through all of the input data. Such an implementation will yield very low occupancy on the CUDA and will be highly inefficient. The latency caused by the for-loop serialization will dominate any speed gains achieved by the parallel inner product operation. So the PFB1 algorithm focuses on eliminating any serial control structures in the CUDA kernel. Recollect the structure of the PFB of Figure 42, each row of the polyphase filter structure operates on specific set of input samples. So, if we store all possible input samples that each row of the polyphase will operate on, in memory, then it is possible to assign separate CUDA threads to perform the filtering operation. In other words, each thread is now responsible for filtering a group (according to the dimension of the PFB) of samples. This resembles the single instruction multi-thread processing that the CUDA was designed for. With this structure it is now possible to spawn a large number of threads and blocks and improve the performance and efficiency of the PFB implementation. An illustration of the polyphase filter bank implementation is shown below (see Figure 46).

Initially the input array is transferred from the host CPU to the device global memory (GM). To maximize the bandwidth of the host to device PCIe transfer, page-locked or pinned memory on the host is utilized through the *cudaMallocHost* function. The input buffer with dimensions $M \times \lambda$ performs a serpentine shift every iteration and is loaded with M new input samples. The PFB1 algorithm eliminates this buffer instead relying on simply indexing to pull all the necessary input samples. Though the input samples are loaded column-wise, the polyphase structure operates row-wise. So when performing the polyphase inner product operation, global memory access will be non-coalesced and will lead to performance degradation. So a separate shifting kernel was employed to pre-sort the data before further processing.

It should be noted that in the shifting kernel, reads from the GM are linear and coalesced, whereas the writes are according to the polyphase structure. The C code for the shifting kernel is shown below:

Table 5. Data shuffling algorithm

Algorithm 1 Pseudocode for data shuffling

```
in_idx = blockIdx.x × blockDim.x + threadIdx.x
idx_rem = in_idx/SAMPLES_PER_ROW;
out_idx = (in_idx - (SAMPLES_PER_ROW * idx_rem)) * N_CHANNELS +
idx_rem
if idx < INPUT_LENGTH then
    out[out_idx] = in[idx]
end if
```

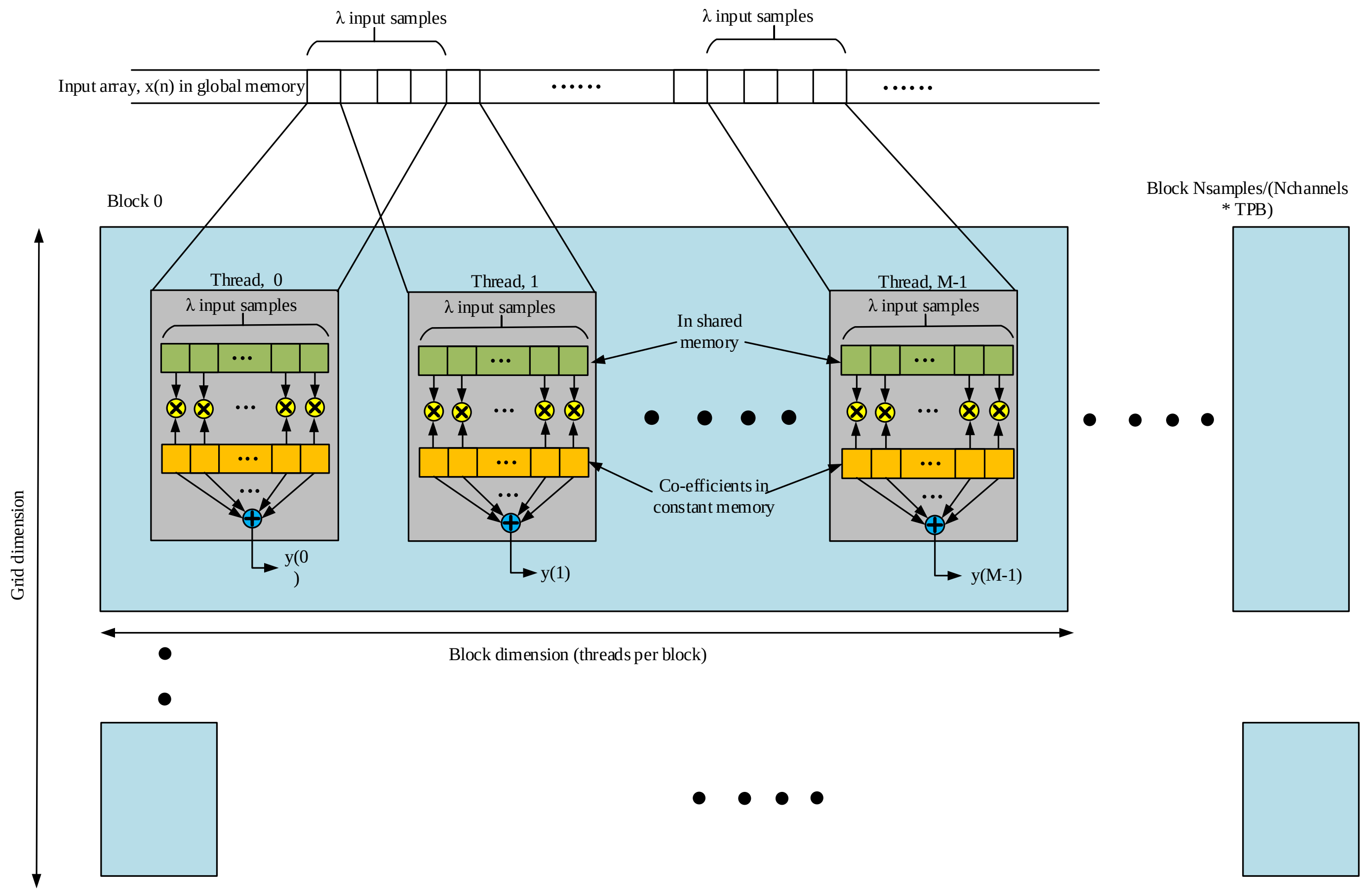


Figure 46. Illustration of polyphase filter on CUDA

Once the data has been re-arranged to be compatible with the row-wise operation of the PFB, the filtering kernel can be called. All the intermediate data structures between the shifting kernel and the filtering kernel are still resident on the device memory and are not mapped to the host memory. This will eliminate unnecessary overheads associated with host to device data transfers. The filtering kernel performs a lot of read and write operations from the global memory, so the data is moved to the shared memory for more efficient access. Each thread in the filtering kernel is responsible for filtering λ samples i.e., λ multiplies and $\lambda - 1$ adds. All the filter coefficients are stored in the constant memory. The C code for the filtering kernel is given below in Table 6

Single precision floating point datatype is used for all the data including filter coefficients. In an SDR environment the digital front end will provide I and Q samples to the GPU for processing, so two independent but identical kernels perform the filtering and shifting operations (one for I and one for Q). This is possible because the filter coefficients are real. The independent kernels are executed as streams so that their computation and data transfer overlap. Once the polyphase filtering is complete, the data is reshuffled back to its original format. At the same time, I and Q float values are converted into complex by simply interleaving them together. Then DFT of the polyphase outputs is performed using cuFFT. The type of cuFFT performed is complex to complex. Once the FFT is complete the channelization process is complete.

Table 6. Polyphase channelizer algorithm

Algorithm 2 Pseudocode for polyphase channelizer

```
idx = blockIdx.x × blockDim.x + threadIdx.x
in_idx = blockIdx.y × SAMPLES_PER_ROW + idx
out_dx = idx × L + blockIdx.y
**if** idx < SAMPLES_PER_ROW **then**
    SM_REG[ix +L − 1] = in[in_idx]
    **if** threadIdx.x < L − 1 **then**
        SM_REG[threadIdx.x] = in[in_idx −L + 1]
    **end if**
    **for** ii = 0 to L − 1 **do**
        SM_MAC[threadIdx.x]+=CM_COEF[(L−1−ii)×M+ blockIdx.y] ×
        SM_REG[threadIdx.x +L − 1 − ii]
    **end for**
    out[out_idx] = SM_MAC[threadIdx.x]
**end if**
```

4.5 Results and Analysis

To perform experimental analysis of the PFB1 algorithm we target verification of the PFB1 algorithm was performed using the 3GPP wideband code division multiple access (WCDMA) standard. WCDMA has an allotted set of bands and each band has a center frequency and an associated bandwidth. The bandwidth of the channels are typically 60 MHz, although they can be as much as 80 MHz. Each frame has a duration of 10 ms and there are 15 time slots per frame. The data rate is 3.84 MHz and occupies approximately 5 MHz of bandwidth. If we assume 60 MHz total bandwidth, then there are 12 channels of 5 MHz each.

The first step of the implementation is to design the prototype filter that will be polyphase decomposed. The filter specifications for WCDMA are shown in Table 7. The filter is designed in MATLAB using the ‘firpm’ algorithm. The number of taps was determined to be N = 192. The time and frequency response of the prototype filter is shown in Figure 47 below. There are 12 channels to process, so the PFB has 12 rows, with each row having a 16 sub-filter coefficients. Therefore $M = 12$ and $\lambda = 16$. Since the input sampling rate $f_{sin} = 60$ MHz and $M = 12$, it shows that the output sampling rate is $f_{sout} = 5$ MHz which is the channel spacing. This tells us that the PFB is maximally decimated.

Table 7. Prototype filter specifications

f_{pass}	3.84 MHz
f_{stop}	5 MHz
A_{ripple}	0.05 dB
A_{pass}	70 dB

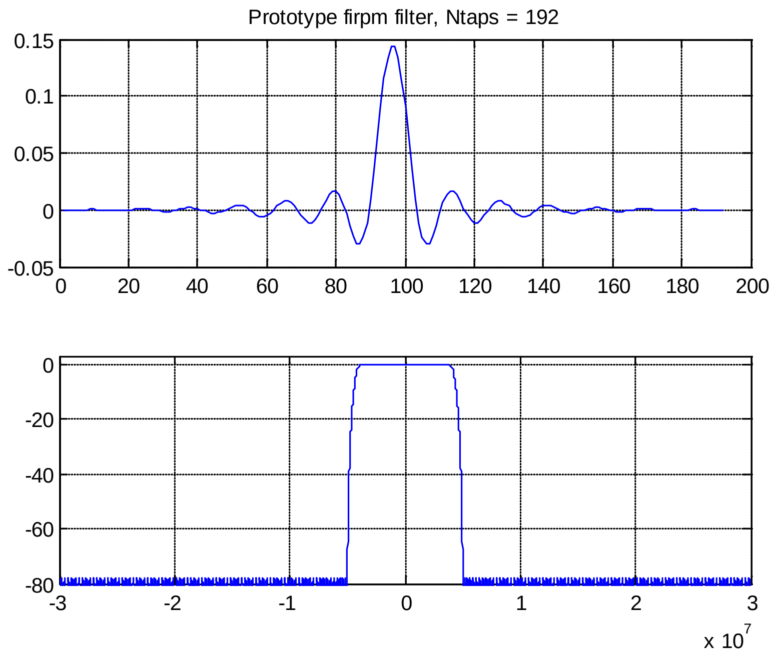


Figure 47. Prototype filter designed using MATLAB

It should be pointed out that the filter coefficients were designed in MATLAB to be single-precision as the computations that will be performed in CUDA will be single precision float. The coefficients are saved as a constant array in *.h header file that will be loaded into the constant memory of the GPU.

The input frame is 10 ms in duration, and with 60 MHz bandwidth there will be 600,000 samples. First step is to rearrange the input samples so that the filter kernel has coalesced access. For the shifting kernel, the number of threads per block was chosen to be 512. Hence the number of blocks would have to be $\frac{600{,}000}{512} = 1{,}172$. Since the shifting kernel only rearranges the data, it was implemented as a 1D grid of

1D blocks resulting in each thread processing one sample. The filter kernel was structured as a 2D grid of 1D blocks. This was because with 600,000 samples each row of the PFB will be processing 50,000 samples. So with a 2D grid the y-direction corresponds to the channel number and the x-direction the individual samples. Each thread is responsible for one sub-filter operation. Each sample along with $\lambda - 1$ previous samples are read into the shared memory for consumption by the thread. Since the input samples are complex there were two identical kernels one operating on real data and the other on imaginary data. Since the two sets of data are independent, concurrent kernel execution was performed. The output from each thread is still arranged based on the polyphase structure and has to be rearranged back before invoking the cuFFT. It should be noted that the cuFFT has to be performed for blocks of $M = 12$. So *cuFFTPlanMany()* was used to define the FFT sizes.

All the experiments were conducted on a NVIDIA Quadro 600 GPU resident on a Dell T1500 PC chassis. The Quadro 600 has 96 cores and is based on the Fermi architecture. It has 1 GB of DDR3 memory with a bandwidth of 25.6 GB/s. All of the CUDA programs were written using CUDA version 6.5 toolkit. The results for the PFB implementation are summarized in the Table 8 below.

Table 8. Performance results for the PFB1 algorithm

	With transfer	Without transfer
Shifting kernel	1.0474 ms	0.6234 ms
Filtering kernel	13.33 ms	12.911 ms
Overall	14.3774 ms	13.5344 ms

Table 9. Reference implementation [59]

	With transfer	Without transfer
Filtering kernel	142.4 ms	118.6 ms
Overall	142.4 ms	118.6 ms

4.6 Summary

The chapter started with a discussion on the issues faced by a cellular base station in processing a defined band in order to down convert, filter and down sample multiple streams of data from the users. Then an introduction to the concept of polyphase channelizer was made. This followed a detailed mathematical review of the polyphase filter bank; both the maximally decimated and rational fraction decimated. We have then discussed implementation of polyphase channelizer on the CUDA. This implementation has optimizations to minimize the rate of data transfers, increase utilization of shared memory, and enhanced GPU utilization by reducing thread granularity (operation complexity). The result is a simpler architecture with reduced bottlenecks and elimination of a dominant sequential loop. Collectively, these optimizations result in significant further improvement in throughput and latency.

CHAPTER V

FRAMEWORK FOR DATA TRANSFER BETWEEN USRP AND GPU

In defining a generalized SDR architecture, not only the required algorithmic digital signal processing requirements must be satisfied but the data transfer rates and the overhead of required protocols between each component must be supported. The number of ADCs and DACs and their sample rates establish the number of frequency bands and bandwidths of received and transmitted signals. The signal formats define the number and types of algorithms required and the appropriate type of processing to be performed, real-time FPGA, GPU, DSP or GPP. The data transfer rates and protocols may also establish fundamental limits on SDR system capability and performance. While real-time FPGAs or ASICs may be directly connected to the ADCs or DACs, all other components will require high data rate interconnects including structured protocols; between ASICs or FPGAs and GPUs, ASICs or FPGAs and DSPs or GPPs, and GPUs and DSPs or GPPs. As may be expected, the interface and interconnection method used to connect ASICs or FPGAs to the rest of the system is likely to be the most critical. In the prototype architecture being used, this involves the USRP with its FPGA based operation and a PC with a GPU.

5.1 Prototype SDR System Interface

The Universal software radio peripheral (USRP) is a family of radio platforms developed by Ettus Research. All the devices in the USRP family consist of an FPGA with various interfaces, ADC, DAC and an array of daughtercards. The flexibility these devices provide combined with their relative low cost is why the USRP is widely used as a research platform for wireless communications. The USRP architecture is designed in such a way that most of the resources like signal processing blocks, filters as well as resources on its motherboard can be configured, controlled and reprogrammed through software. As with all hardware platforms, the mechanisms of command, control and data uplink/downlink in current generation devices is completely different from that of the previous generation. Due to lack of documentation, this information can only be ascertained by reverse engineering the firmware that is provided by Ettus. In this chapter we will analyze the firmware of the USRP and describe a suitable framework to transfer data between the USRP and the GPU on the PC.

5.2 Types of USRP Drivers

Access to the functionality of the USRP devices from software distributions varies depending on the family of the USRP used and the type of interface that the device supports. In this section we discuss two distinct driver generations, the old driver that supported the USRP1 and USRP2 devices and the newer USRP hardware driver (UHD) that supports the current family of devices.

5.2.1 USRP1 and USRP2 Drivers

The USRP1 and USRP2 devices can be interfaced with two separate drivers provided by Ettus. The USRP1 was a USB based device and the USRP2 an Ethernet based device. Two separate C++ libraries (libusrp.so and libusrp2.so) were used to communicate with each SDR [55]. Both the libraries were distributed with GNURadio and provided only Linux support. The drivers provided support for real and complex samples, the ability to set operating parameters such as frequencies and amplification, and access to device buffers for transmission and reception over the separate wired communication interfaces. The libusrp2.so used raw sockets to communicate between the USRP2 and the host PC. They supported a three layer network stack (physical, data link and application). The absence of transport layer support meant the packets were not routable. Raw sockets also require root level access under Linux.

5.2.2 Next Generation USRP’s

To support various improvements of the new USRP generation, Ettus provided a new driver, the USRP hardware driver (UHD). UHD is a C++ based application programming interface (API) for all USRP devices irrespective of the type of interface. It provides all the functionality of the previous USRP1 driver along with added functionality providing resources to:

- Switch between oscillation and timing reference sources;
- Configure time tagged transmission;
- Read out time tags from received samples;
- Configure channel and antenna assignment in MIMO configuration;
- Transmit and receive a specific number of samples by start and end of burst flags; and
- Support the detection and notification of stream interruptions

The UHD can also access USRP1 devices where the features from above are emulated in the UHD driver on the PC, except for the start and end of burst capability which is ignored by the driver if a USRP1 device is accessed. This way all USRP devices can be accessed by the UHD with the maximum possible functionality for each device.

5.3 UHD

The UHD software is the hardware driver for all USRP devices. It is cross platform and supports Linux, windows and Macintosh Operating systems. It also provides support for GCC, Clang and Microsoft Visual Studio compilers. The goal of the UHD software is to provide a host driver and API for current and future Ettus research products. It can be used standalone or with third-party applications like

GNURadio, LabVIEW, Simulink, OpenBTS, Iris, Redhawk, Amarisoft LTE eNodeB. UHD functions can be split into two parts: (1) functions related to control and configuration and (2) API's related to streaming data samples to and from the devices. A top level architecture of the UHD driver is given below (see Figure 48).

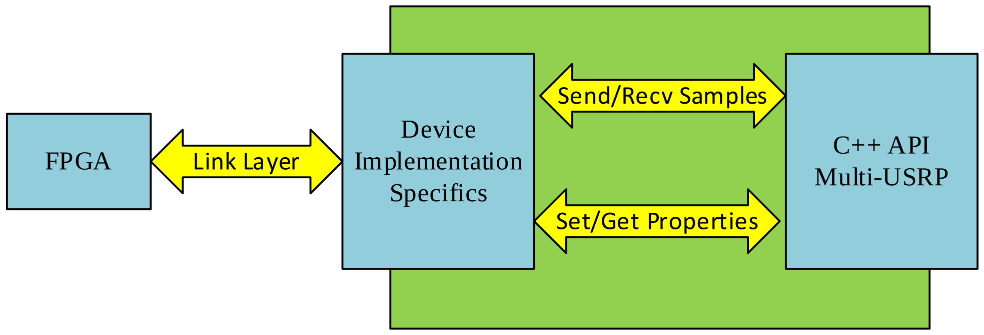


Figure 48. UHD architecture

Device specific functions can be used to find devices on the system, set/get device and daughtercard specific properties like gain, center frequency, and sample rate, antenna and frontend selection, and read back sensor data. Functions to stream data samples (device→send(..) and service→recv(..)), and messages (overflow, underflow, sequence error, and others) are also provided. Link layer support for USB, UDP/IPv4 protocols is such that user is not tasked with knowing the specifics of these standards. UHD employs libusb libraries for USB 2.0 support. UDP/IPv4 support is through Berkley sockets (send( ) and recv( )). As of this writing, Ethernet based USRP devices offer throughput of 1 Gbps (theoretical). Since performance parameters of UDP/IPv4 protocols vary with OS and network drivers, it is left up to the user to adjust buffer, packet sizes and latency settings.

5.3.1 VITA-49 Radio Transport

The third-generation Ettus research USRP has been developed with a unique routing architecture based on VITA-49. The VITA-49 radio transport (VRT) standard was established to provide a consistent way to format sampled IF or I/Q data in software defined radio system. Prior to VRT, vendors developed proprietary protocols to transport samples across processing elements of an SDR. Both samples and control messages are transported to and from the host PC via USB 2.0, Ethernet, or external memory interfaces. In the case of the N-series devices, the VITA-49 frames are encapsulated with UDP. On the PC side the UHD driver takes care of encoding and decoding VRT packets. However if the user wants to integrate custom signal processing blocks on the FPGA, configuration and control messages still have to implemented using VITA-49. The VRT packet consists of 4 bytes of header information and a payload of D bytes. There are two kinds of packets implemented on the USRP, context packet and data packet. These two types are differentiated using the first four bits of the header. Both the transmitted and the received streams have a unique stream identifier and a timestamp. The stream identifier is 4 bytes long and the timestamp is 12 bytes. The timestamp field can include both fractional and integer values.

IF data packet

The IF data packet header has a packet length of 32 bits. Again the first 4 bits of the header indicate packet type, the next two bits indicate class identifier and trailer bit. 24th and 25th bit are reserved for future use. The next four bits are for integer and fractional timestamp. Next come packet count and packet size.

Extension context packet

The extension context packet has a similar structure to the ID data packet in terms of header information. One difference though is that it does not have a trailer and so 26th and 25th bit are reserved. The time stamp mode (TSM) bit indicates whether the timestamp are coarse or fine grained.

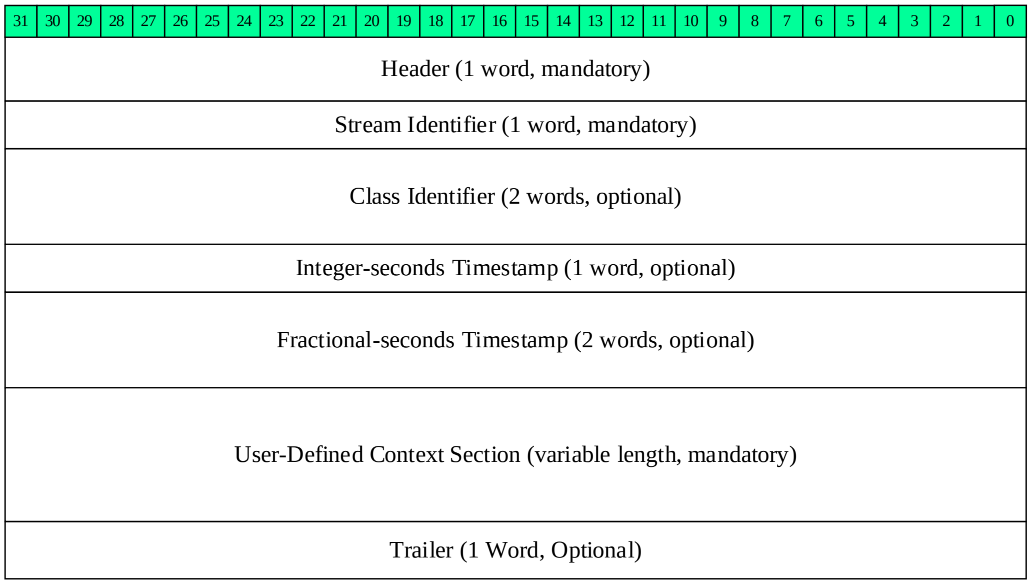
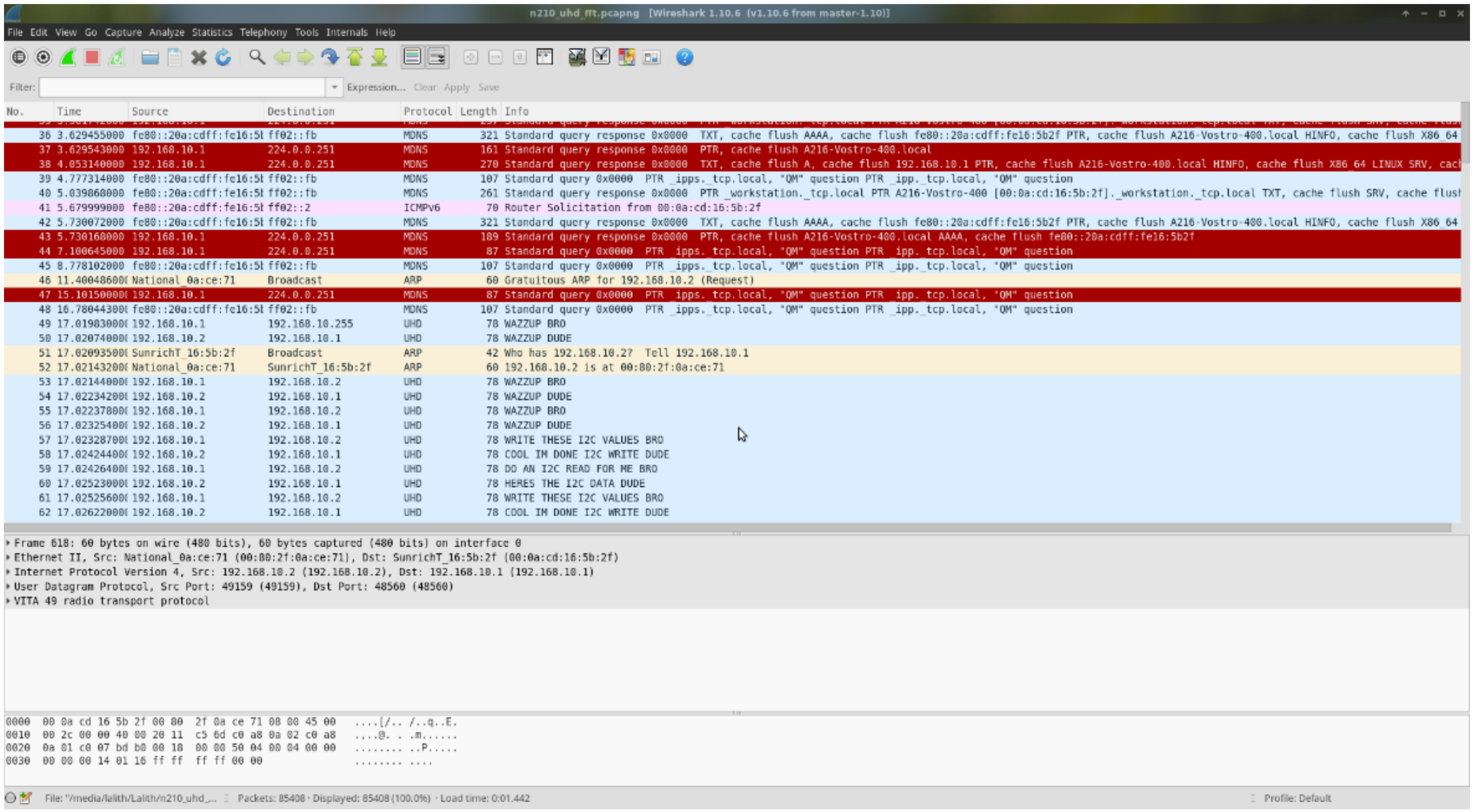

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Header (1 word, mandatory)
Stream Identifier (1 word, mandatory)
Class Identifier (2 words, optional)
Integer-seconds Timestamp (1 word, optional)
Fractional-seconds Timestamp (2 words, optional)
User-Defined Context Section (variable length, mandatory)
Trailer (1 Word, Optional)

Figure 49. VRT IF data packet


n210_uhd_fft.pcapng [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
Profile: Default

```
No.  Time         Source                    Destination     Protocol Length Info
 36 3.629455000  fe80::20a:cdff:fe16:5t ff02::fb         MDNS     321 Standard query response 0x0000  TXT, cache flush AAAA, cache flush fe80::20a:cdff:fe16:5b2f PTR, cache flush A216-Vostro-400.local HINFO, cache flush X86_64
 37 3.629543000  192.168.10.1             224.0.0.251      MDNS     161 Standard query response 0x0000  PTR, cache flush A216-Vostro-400.local
 38 4.053140000  192.168.10.1             224.0.0.251      MDNS     270 Standard query response 0x0000  TXT, cache flush A, cache flush 192.168.10.1 PTR, cache flush A216-Vostro-400.local HINFO, cache flush X86_64 LINUX SRV, cach
 39 4.777314000  fe80::20a:cdff:fe16:5t ff02::fb         MDNS     107 Standard query 0x0000  PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
 40 5.039868000  fe80::20a:cdff:fe16:5t ff02::fb         MDNS     261 Standard query response 0x0000  PTR _workstation._tcp.local PTR A216-Vostro-400 [00:0a:cd:16:5b:2f]._workstation._tcp.local TXT, cache flush SRV, cache flush
 41 5.679999000  fe80::20a:cdff:fe16:5t ff02::2          ICMPv6    70 Router Solicitation from 00:0a:cd:16:5b:2f
 42 5.730072000  fe80::20a:cdff:fe16:5t ff02::fb         MDNS     321 Standard query response 0x0000  TXT, cache flush AAAA, cache flush fe80::20a:cdff:fe16:5b2f PTR, cache flush A216-Vostro-400.local HINFO, cache flush X86_64
 43 5.730168000  192.168.10.1             224.0.0.251      MDNS     189 Standard query response 0x0000  PTR, cache flush A216-Vostro-400.local AAAA, cache flush fe80::20a:cdff:fe16:5b2f
 44 7.100645000  192.168.10.1             224.0.0.251      MDNS      87 Standard query 0x0000  PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
 45 8.778102000  fe80::20a:cdff:fe16:5t ff02::fb         MDNS     107 Standard query 0x0000  PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
 46 11.40048600( National_0a:ce:71       Broadcast        ARP       60 Gratuitous ARP for 192.168.10.2 (Request)
 47 15.10150000( 192.168.10.1             224.0.0.251      MDNS      87 Standard query 0x0000  PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
 48 16.78044300( fe80::20a:cdff:fe16:5t ff02::fb         MDNS     107 Standard query 0x0000  PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
 49 17.01983000( 192.168.10.1             192.168.10.255   UHD       78 WAZZUP BRO
 50 17.02074000( 192.168.10.2             192.168.10.1     UHD       78 WAZZUP DUDE
 51 17.02093500( SunrichT_16:5b:2f        Broadcast        ARP       42 Who has 192.168.10.2?  Tell 192.168.10.1
 52 17.02143200( National_0a:ce:71       SunrichT_16:5b:2f ARP       60 192.168.10.2 is at 00:80:2f:0a:ce:71
 53 17.02144000( 192.168.10.1             192.168.10.2     UHD       78 WAZZUP BRO
 54 17.02234200( 192.168.10.2             192.168.10.1     UHD       78 WAZZUP DUDE
 55 17.02237800( 192.168.10.1             192.168.10.2     UHD       78 WAZZUP BRO
 56 17.02325400( 192.168.10.2             192.168.10.1     UHD       78 WAZZUP DUDE
 57 17.02328700( 192.168.10.1             192.168.10.2     UHD       78 WRITE THESE I2C VALUES BRO
 58 17.02424400( 192.168.10.2             192.168.10.1     UHD       78 COOL IM DONE I2C WRITE DUDE
 59 17.02426400( 192.168.10.1             192.168.10.2     UHD       78 DO AN I2C READ FOR ME BRO
 60 17.02523000( 192.168.10.2             192.168.10.1     UHD       78 HERES THE I2C DATA DUDE
 61 17.02525600( 192.168.10.1             192.168.10.2     UHD       78 WRITE THESE I2C VALUES BRO
 62 17.02622000( 192.168.10.2             192.168.10.1     UHD       78 COOL IM DONE I2C WRITE DUDE
```

```
▸ Frame 618: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: National_0a:ce:71 (00:80:2f:0a:ce:71), Dst: SunrichT_16:5b:2f (00:0a:cd:16:5b:2f)
▸ Internet Protocol Version 4, Src: 192.168.10.2 (192.168.10.2), Dst: 192.168.10.1 (192.168.10.1)
▸ User Datagram Protocol, Src Port: 49159 (49159), Dst Port: 48560 (48560)
▸ VITA 49 radio transport protocol
```

```
0000  00 0a cd 16 5b 2f 00 80  2f 0a ce 71 08 00 45 00   ....[/.. /..q..E.
0010  00 2c 00 00 40 00 20 11  c5 6d c0 a8 0a 02 c0 a8   .,..@. . .m......
0020  0a 01 c0 07 bd b0 00 18  00 00 50 04 00 04 00 00   ........ ..P.....
0030  00 00 00 14 01 16 ff ff  ff ff 00 00               ........ ....
```

File: "/media/lalith/Lalith/n210_uhd_... Packets: 85408 · Displayed: 85408 (100.0%) · Load time: 0:01.442

Figure 50. Wireshark IF data packet decoding

5.3.2 USRP-Host Communication

This section will help us understand how the USRP communicates with the Host-PC, such that it is possible to transfer data to and from the FPGA. The communication between Host-PC and USRP can be incorporated by higher level software using the UHD library, which is a C++ library that holds different calls, abstracting the user from the lower layers such as the Ethernet protocol and the VRT, used to transfer the data between USRP and host.

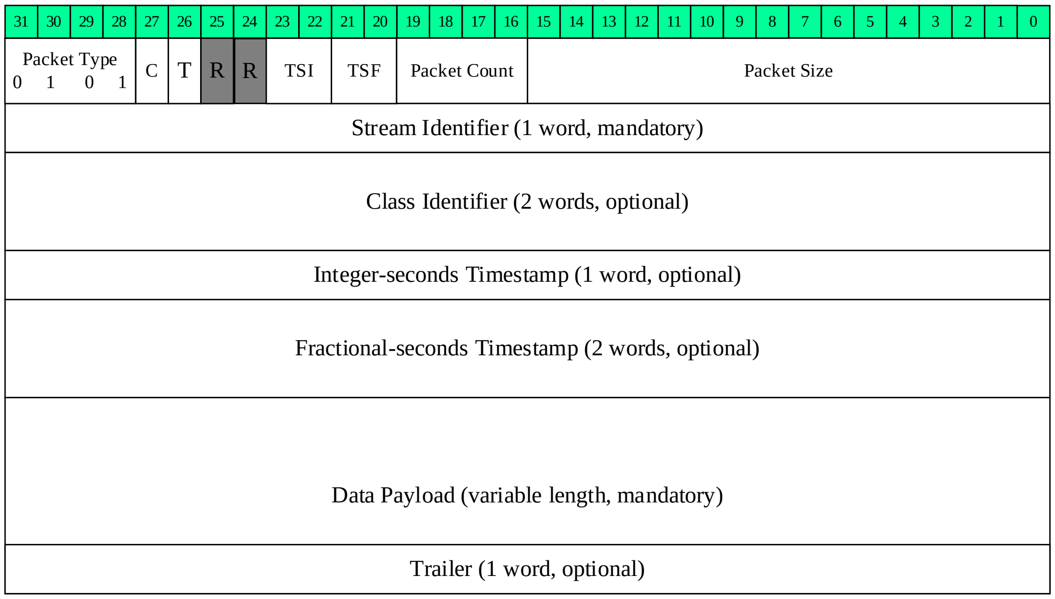
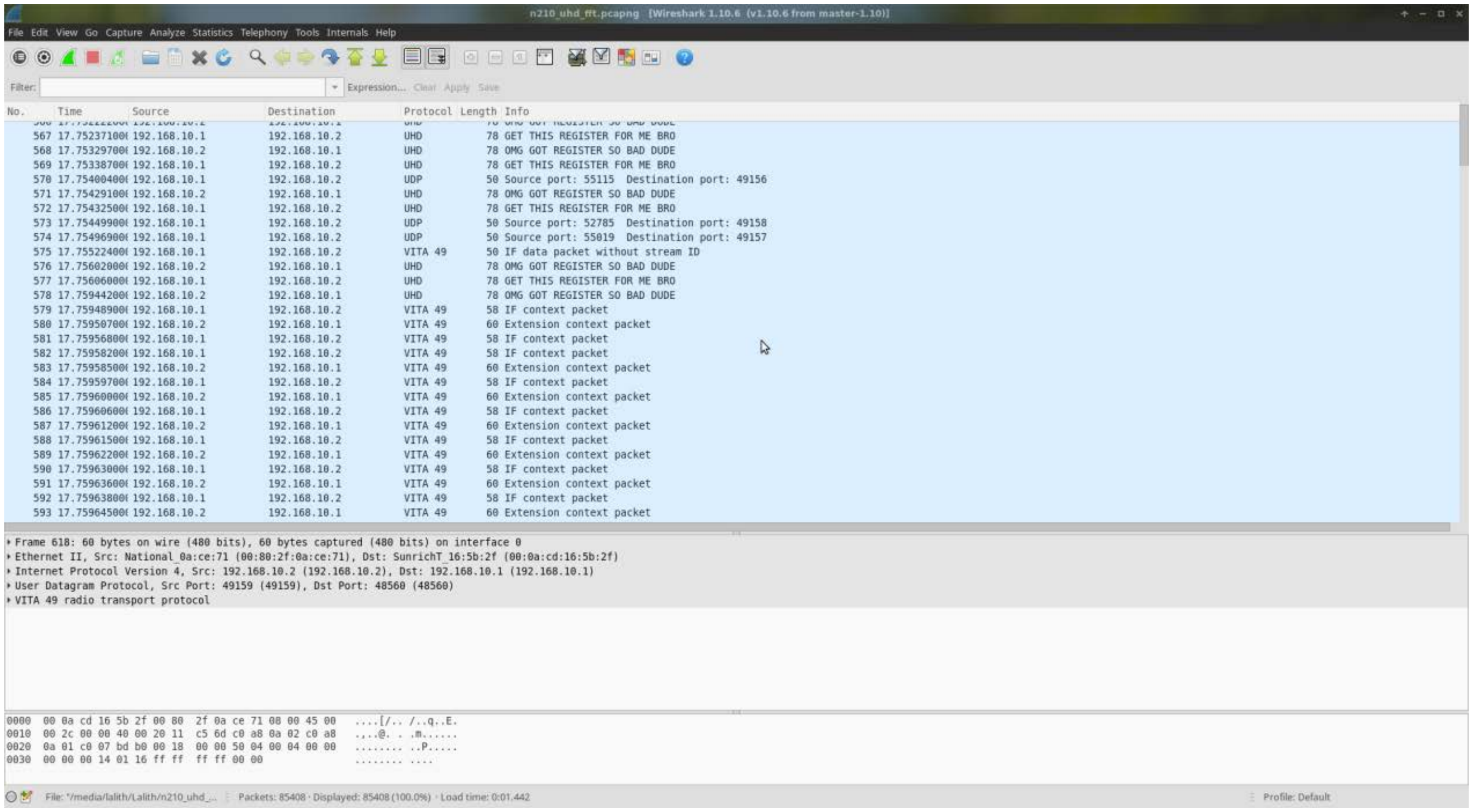

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Packet Type
0 1 0 1
C
T
R
R
TSI
TSF
Packet Count
Packet Size
Stream Identifier (1 word, mandatory)
Class Identifier (2 words, optional)
Integer-seconds Timestamp (1 word, optional)
Fractional-seconds Timestamp (2 words, optional)
Data Payload (variable length, mandatory)
Trailer (1 word, optional)

Figure 51. VRT extension context packet


n210_uhd_fft.pcapng [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.   Time        Source          Destination     Protocol Length Info
  567 17.75237100( 192.168.10.1   192.168.10.2    UHD       78 GET THIS REGISTER FOR ME BRO
  568 17.75329700( 192.168.10.2   192.168.10.1    UHD       78 OMG GOT REGISTER SO BAD DUDE
  569 17.75338700( 192.168.10.1   192.168.10.2    UHD       78 GET THIS REGISTER FOR ME BRO
  570 17.75400400( 192.168.10.1   192.168.10.2    UDP       50 Source port: 55115  Destination port: 49156
  571 17.75429100( 192.168.10.2   192.168.10.1    UHD       78 OMG GOT REGISTER SO BAD DUDE
  572 17.75432500( 192.168.10.1   192.168.10.2    UHD       78 GET THIS REGISTER FOR ME BRO
  573 17.75449900( 192.168.10.1   192.168.10.2    UDP       50 Source port: 52785  Destination port: 49158
  574 17.75496900( 192.168.10.1   192.168.10.2    UDP       50 Source port: 55019  Destination port: 49157
  575 17.75522400( 192.168.10.1   192.168.10.2    VITA 49   50 IF data packet without stream ID
  576 17.75602000( 192.168.10.2   192.168.10.1    UHD       78 OMG GOT REGISTER SO BAD DUDE
  577 17.75606000( 192.168.10.1   192.168.10.2    UHD       78 GET THIS REGISTER FOR ME BRO
  578 17.75944200( 192.168.10.2   192.168.10.1    UHD       78 OMG GOT REGISTER SO BAD DUDE
  579 17.75948900( 192.168.10.1   192.168.10.2    VITA 49   58 IF context packet
  580 17.75950700( 192.168.10.2   192.168.10.1    VITA 49   60 Extension context packet
  581 17.75956800( 192.168.10.1   192.168.10.2    VITA 49   58 IF context packet
  582 17.75958200( 192.168.10.1   192.168.10.2    VITA 49   58 IF context packet
  583 17.75958500( 192.168.10.2   192.168.10.1    VITA 49   60 Extension context packet
  584 17.75959700( 192.168.10.1   192.168.10.2    VITA 49   58 IF context packet
  585 17.75960000( 192.168.10.2   192.168.10.1    VITA 49   60 Extension context packet
  586 17.75960600( 192.168.10.1   192.168.10.2    VITA 49   58 IF context packet
  587 17.75961200( 192.168.10.2   192.168.10.1    VITA 49   60 Extension context packet
  588 17.75961500( 192.168.10.1   192.168.10.2    VITA 49   58 IF context packet
  589 17.75962200( 192.168.10.2   192.168.10.1    VITA 49   60 Extension context packet
  590 17.75963000( 192.168.10.1   192.168.10.2    VITA 49   58 IF context packet
  591 17.75963600( 192.168.10.2   192.168.10.1    VITA 49   60 Extension context packet
  592 17.75963800( 192.168.10.1   192.168.10.2    VITA 49   58 IF context packet
  593 17.75964500( 192.168.10.2   192.168.10.1    VITA 49   60 Extension context packet
```

```
▸ Frame 618: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: National_0a:ce:71 (00:80:2f:0a:ce:71), Dst: SunrichT_16:5b:2f (00:0a:cd:16:5b:2f)
▸ Internet Protocol Version 4, Src: 192.168.10.2 (192.168.10.2), Dst: 192.168.10.1 (192.168.10.1)
▸ User Datagram Protocol, Src Port: 49159 (49159), Dst Port: 48560 (48560)
▸ VITA 49 radio transport protocol
```

```
0000  00 0a cd 16 5b 2f 00 80  2f 0a ce 71 08 00 45 00   ....[/.. /..q..E.
0010  00 2c 00 00 40 00 20 11  c5 6d c0 a8 0a 02 c0 a8   .,..@. . .m......
0020  0a 01 c0 07 bd b0 00 18  00 00 50 04 00 04 00 00   ........ ..P.....
0030  00 00 00 14 01 16 ff ff  ff ff 00 00               ........ ....
```

File: "/media/lalith/Lalith/n210_uhd_...
Packets: 85408 · Displayed: 85408 (100.0%) · Load time: 0:01.442
Profile: Default

Figure 52. Wireshark extension packet decoding

The data from the Host-PC to the USRP2 is streamed through the Ethernet cable using the before mentioned VRT protocol, wrapped in a standard Ethernet protocol. The flow of the transmit stream and receive stream can be seen in Figure 53 and Figure 54, respectively. The ZPU is a soft-core processing unit, which is implemented on the FPGA and handles setting registers as will be described later in this section.

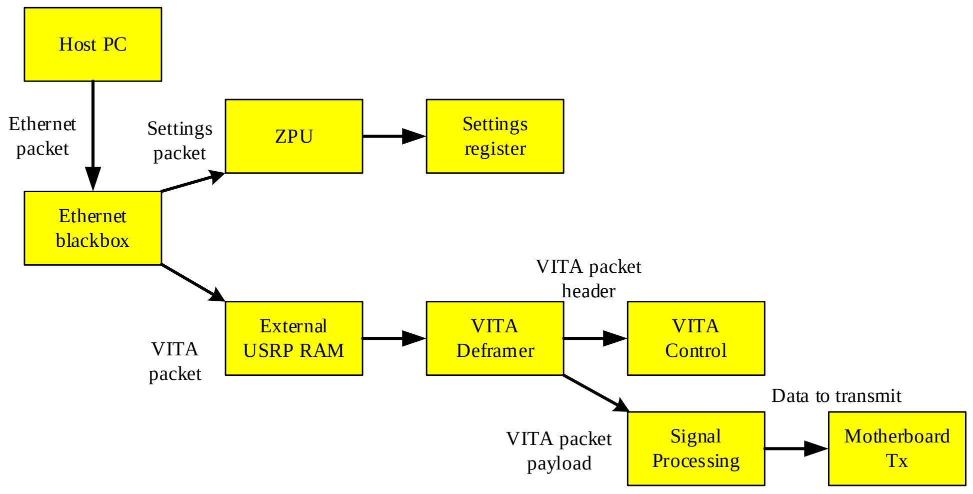


Figure 53. Data flow from host-PC to the transmit front-end

There are two main functionalities in the UHD; initialization and data transmission. As seen in Figure 53, it is the Ethernet black box which checks if the Ethernet packet received from the Host-PC holds setup information or transmission data. Similarly the Ethernet black box chooses if the data transmitted to the Host-PC from the USRP2 is information from the setting registers or data from the receiver. The initialization calls are used to setup, or read from, the USRP's control registers which determines variables such as the daughterboard sample rate, choice of filter or buffer size.

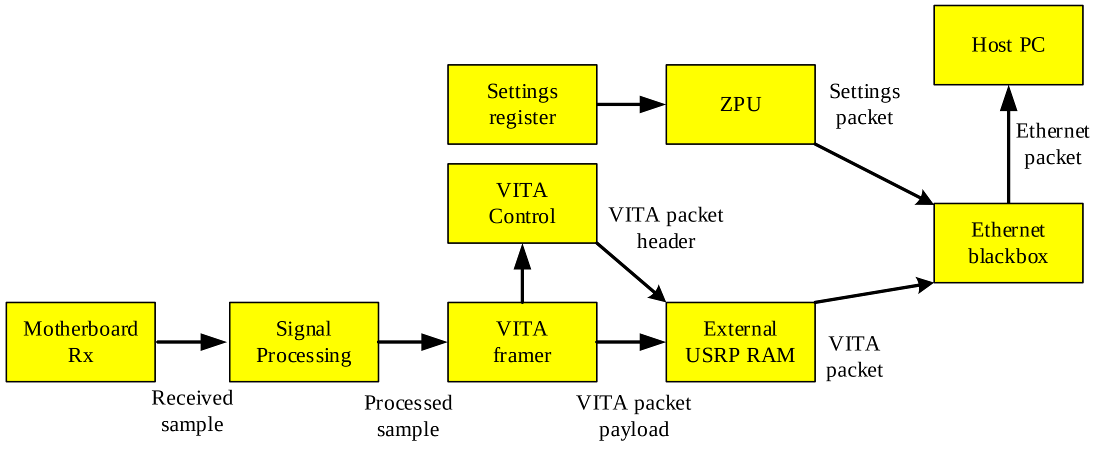


Figure 54. Data flow from receive front-end to host-PC

If the Ethernet black box determines that the received packet controls the USRP setup it is routed to the ZPU, which then handles the control registers. The control signals are stored in setting registers at different addresses, using a shared bus named set data. Figure 55 illustrates the principle of the setting registers usage in the USRP's FPGA image. When the setting register data strobe *set_stb* goes high, the output of the setting register which has the address defined by the address bus *set_addr* will take the value currently held by the *set_data* bus. If the Host-PC asks for information from the setting registers, the ZPU reads the given register and gives the data to the Ethernet black box which transmits it to the Host-PC.

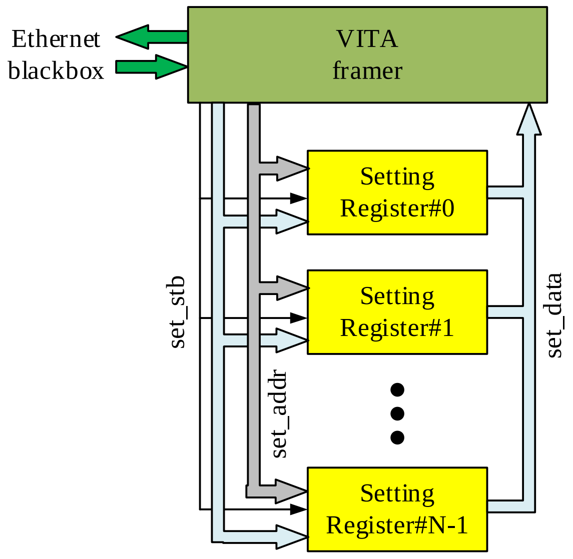


Figure 55. Simplified illustration of the settings register

The UHD "transmit" and "receive" commands are blocking calls, meaning that it is not possible to setup the USRP2 and transmit at the same time. When a transmission begins, the USRP2s setting register which controls the transmission is set, and the Ethernet blackbox reroutes the data to the VITA deframer, which ensures that the data is correctly transmitted. Similarly, if a reception is to begin, the setting register controlling this is set and the Ethernet black box uses the data input from the DSP core.

5.4 Data Transfer Framework

Now that the operation of UHD, VRT and the internal logic of the USRP FPGA is well understood, it is time to implement an interface function to collect data from the USRP. The UHD is implemented in object-oriented C++ and uses the boost C++ libraries. It is open source and can be compiled either using Microsoft Visual Studio Compiler (MVSC).or using GNU GCC toolkit. The UHD driver provides various classes, struts, and interfaces to identify, configure, control and stream data between the PC and the USRP. The most important classes are shown in Table 9 below.

Table 10. UHD list of important classes

uhd_static_fixture	Helper for static block, constructor calls function
uhd_range_t	Range of floating-point values
uhd_stream_args_t	A struct of parameters to construct a stream
uhd_stream_cmd_t	Define how device streams to host
uhd_subdev_spec_pair_t	Subdevice specification
uhd_tune_request_t	Instructs implementation how to tune the RF chain
uhd_tune_result_t	Stores RF and DSP tuned frequencies
uhd_usrp_register_info_t	Register info
uhd_usrp_rx_info_t	USRP RX info
uhd_usrp_tx_info_t	USRP TX info

A complete list of all the classes can be found in the USRP manual, and will not be listed in this document. Instead, the minimum number of steps required to create, configure and stream data is given below.

1. Create a USRP device using uhd::usrp::multi_usrp::sptr usrp = uhd::usrp::multi_usrp::make(args);
2. Lock any available onboard clocks usrp→set_rx_rate(value)

3. Similarly set gain, bandwidth, center frequency and antenna
4. Check if the local oscillator has achieved lock by reading sensor info
5. Create a receive streamer using uhd::rx_streamer::sptr rx_stream = usrp->get_rx_stream(stream_args);
6. Setup stream mode to continuous stream_cmd.stream_mode = uhd::stream_cmd_t::STREAM_MODE_STOP_CONTINUOUS;
7. Start the stream. When a transmit is required, replace the receive commands with the transmit commands.

Two C++ programs one for transmit and one for receive were created. Currently these files will stream data to a buffer on the PC hard-disk. Post processing of the data can then be performed on either MATLAB or GPU.

5.5 Results and Analysis

Testing of the two applications to collect data from the USRP was performed by first compiling the programs on Ubuntu 14.04 machine using GCC-4.8. A USRP N210 device was used with the BasicRX card. Two different FM spectrums were collected, one setup to receive only one station and the other setup to receive multiple stations. Both the spectrums were centered on 96.5 MHz. the collected samples were then imported to MATLAB to verify proper operation.

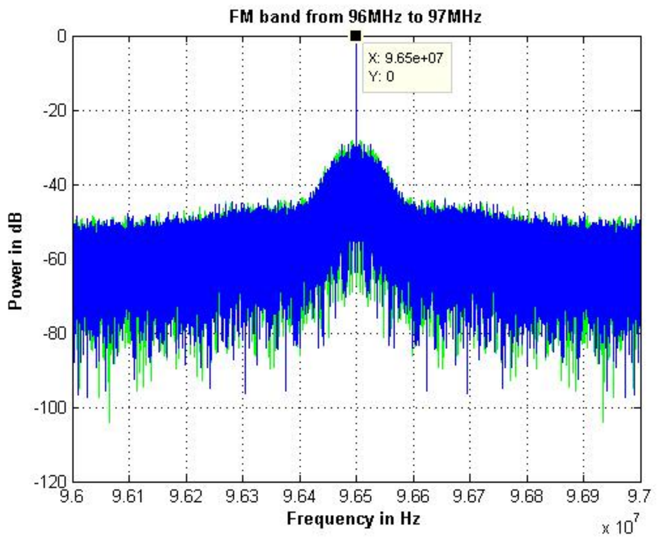


Figure 56. FM spectrum at 96.5 MHz

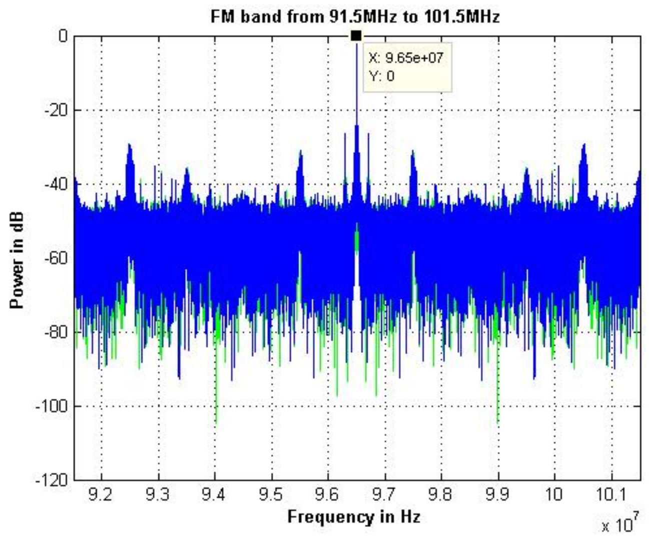


Figure 57. FM spectrum 91.5 MHz to 101.5 MHz

5.6 Summary

This chapter provided a brief introduction to the history of the USRP driver. Then the current generation UHD drivers were discussed. All the data and control packets of the USRP are transported using the VITA 49 standard. A description of the important VRT packets namely IF data packets and extension context packet was provided. An understanding of these packets is essential for understanding how control and configuration is achieved on the USRP. The movement of data within the FPGA was also discussed. Finally two C++ programs were created to enable transfer of the samples from the USRP to the PC using only the UHD driver and not GNURadio. This is a small but important step towards realizing a GPU accelerated software radio device.

CHAPTER VI

CONCLUSION AND FUTURE WORK

This work introduced as a first step, a novel reference architecture for a software defined radio platform, with a goal of providing a reprogrammable, commodity commercial component based architecture that can be tailored for the broadest range of wireless communication applications, including; single narrowband channels, multiple narrowband channel repeater or base-station, singular cellular telephone signaling or cellular telephone base station, multi-format, multi-band home wireless access point or as an emergency wireless access point to restore wireless communication infrastructure. The architecture employs one or more USRP devices to provide RF signal transmission and reception, high-speed interconnection sufficient to handle required data distribution and connection, GPU parallel processing accelerators, and multi-threaded, multi-core processor PC back-ends. A prototype version of this SDR was then built and tested so that the critical aspects and functionality could be demonstrated, assessed, and verified. In this work, the use of GPU acceleration for communication algorithm signal processing was demonstrated. The demonstration algorithm selected was for a wide-bandwidth polyphase filter bank channelizer. With concerns for high-speed real-time data interconnection and interfaces, a deeper analysis and performance demonstration of USRP to PC communications was completed. The Gigabit Ethernet interfaces and embedded protocols for SDR data communication were researched and defined, and a demonstration of high-rate data capture across the interface was completed. A summary of the outcomes of these research projects are discussed below:

6.1 Contributions

In the first project, a model generic software radio was defined with a USRP as an RF front end, FPGA board as an additional real-time DSP element and a GPU as a principle parallel software signal processing element. All the above elements were interfaced using high speed Ethernet, PCIe and USB standards. This reference architecture will allow modular upgrades to be performed during incremental technology advances, without having to do complete system redesign. To validate the reference design, a prototype platform was built using Quadro 600 as the GPU processing element, USRP N210 and B100 devices as the RF front ends and commodity PC with PCIe and Ethernet as the underlying interface. A GNURadio application was used as the software development environment for a PC in order to validate operation and prototype design for FM radio, video streaming and ADS – B applications. The performance results from these applications show that the design can be used as a flexible, programmable research platform for current generation high data rate, high bandwidth communication standards like Wi-Fi, 4G, and others.

The second project involved mapping a computationally intensive communication algorithm onto the parallel architectural elements of a GPU. For this dissertation a maximally decimated polyphase filter bank channelizer was implemented with high throughput and low latency. The inherent parallel PFB channelizer's structures were exploited by removing sequential load, shift operations, instead employing numerous threads available on the GPU to perform the required operations. This leveraged the single instruction multiple thread (SIMT) design of the GPU allowing us to reduce computation times drastically. A 3GPP WCDMA example was used to validate and measure performance. This example used a 10 ms frame with 60 MHz bandwidth resulting in a 10-15x speedups as compared to a GPP implementation. All GPU hosted programs were written in CUDA C.

In the final project a framework to transfer samples from the USRP RF front ends to the PC and the GPU was defined and implemented. This project utilizes the UHD driver from USRP negating the need for GNURadio and improving resource utilization. For this project, the USRP N210 with SBX daughtercard was chosen. The N210 allowed demonstration system configuration used a gigabit Ethernet link to transfer the digitized, complex RF signal to/from the air interface. A detailed discussion of the USRP software architecture, not available elsewhere was also provided. The validation included C++ routines that were written using classes and functions defined in the UHD driver. Real-time sample captures of the single channel

FM and multichannel FM broadcasts data provide validation for the software routines developed.

6.2 Future Work

The tasks completed have demonstrated and validated key elements of the generic SDR system architecture. With a working demonstration system, a wide range of additional performance demonstrations and validation still reaming. These include:

1. Parallel processing software development for additional wireless communication algorithms. The investigation and mapping of a range of wireless algorithms remains, including; narrow bandwidth frequency domain equalization, broad bandwidth equalization, timing and symbol synchronization, and other FPGA algorithms that may migrate to GPUs.
2. End-to-end demonstration system operation for single and multiple narrowband communication signals. Demonstrate operation as a FM or FSK transceiver base station, frequency repeater and possible multi-format translator focusing on emergency radios.
3. General purpose and/or emergency wireless access point development and demonstration. Using the prototype system embodiment, assess appropriate system component selections to support R&D proposals

for and development of a software programmable access point based on the end-to-end communication formats investigated.

During this investigation, technological advancements have been numerous, below are several opportunities for improving the current demonstration system within the confines of the generalized SDR architecture and demonstrating more advanced wireless communication signal processing tasks:

1. Hardware improvements: the gigabit Ethernet and the USB interfaces used in the N210 and the B100 devices are a limiting factor in terms of the maximum achievable bandwidth. In the process of this dissertation, Ettus research has released the X series devices that are PCIe based and capable of supporting up to 120MHz bandwidth. Apart from bandwidth capabilities, newer devices use the Xilinx 7 series FPGAs that have resources to implement an ARM core processor.
2. Arbitrary resampling polyphase: the polyphase channelizer implemented in this dissertation is of the maximally decimated kind and is unsuitable in numerous applications. Implementation of a polyphase channelizer with arbitrary resampling capabilities will provide support to a much wider scope of standards and application areas.

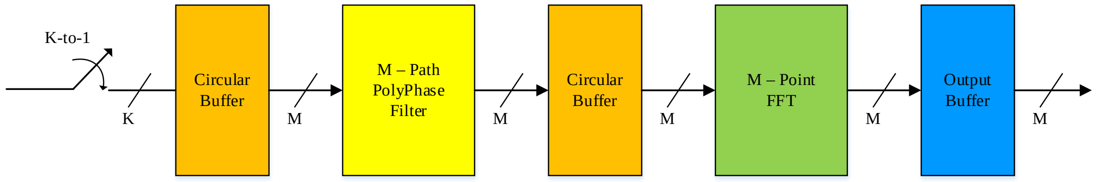


Figure 58. Arbitrary K/M polyphase channelizer

3. Peer to peer PCIe transfer: it is without a doubt that future RF front ends will start using the high speed PCIe backplane for data transfer to/from the PC. As the GPU is also PC based it would be efficient to use PCIe to perform a peer to peer transfer and eliminate the CPU from being involved in the data transfer process. In this case the CPU will essentially perform the task of configuration, control and application specific functions rather than physical layer processing.

GPU compute capability: The CUDA toolkit environment and the capabilities of the GPU cards are improving every year. Current CUDA devices with higher compute capabilities support additional features like invoking kernel calls from within kernels and dynamic reallocation of resources. These features will allow designers to perform 'bucket brigade' type processing for more complex algorithms like the arbitrary polyphase channelizer.

6.3 Summary and Conclusion

The field of software defined radios encompasses multiple disciplines like analog RF design, digital signal processing, high speed logic design, parallel computing and more. Research in this field has provided an opportunity to explore some of the latest technologies in these disciplines and find innovative ways to employ these technologies towards the goal of achieving universal software radios. Utilizing CUDA for wireless communication provided its own set of challenges; from mapping traditionally sequential algorithms on parallel structures to effectively utilizing the capabilities of the modern GPUs. Research on graphics acceleration for SDR is still in its infancy, but is one that is set to revolutionize future radio systems design. Reconfigurable logic, high speed interfaces and low power technologies are other areas that will have tremendous impact on future communication systems. This dissertation also provided an opportunity to gain insights into these technologies and understand implications of their shortcomings on overall system design.

Mobility and access to services anytime, anywhere are requirements that drive current and future wireless communication technologies. A look into modern smartphones will reveal the amount of processing power and technological innovations that are required to deliver these demands. In fact, current mobile devices have more processing power than laptops from the early millennium. It will not be surprising that SDRs and GPUs will be the enabling technologies that keep this trend continuing.

BIBLIOGRAPHY

[1] M. Salazar-Palma, A. Garcia-Lamperez, T. K. Sarkar and D. L. Sengupta, "The Father of Radio: A Brief Chronology of the Origin and Development of Wireless Communications," *Antennas and Propagation Magazine, IEEE,* vol. 53, pp. 83-114, 2011.

[2] M. Salazar-Palma, T. K. Sarkar and D. Sengupta, "A brief chronology of the origin and developments of wireless communication and supporting electronics," in *Applied Electromagnetics Conference (AEMC), 2009,* 2009, pp. 1-4.

[3] J. S. Belrose, "Fessenden and marconi: Their differing technologies and transatlantic experiments during the first decade of this century," in *100 Years of Radio., Proceedings of the 1995 International Conference On,* 1995, pp. 32-43.

[4] D. Raychaudhuri and N. B. Mandayam, "Frontiers of Wireless and Mobile Communications," *Proceedings of the IEEE,* vol. 100, pp. 824-840, 2012.

[5] I. Cisco, "Cisco visual networking index: Forecast and methodology, 2011–2016," *CISCO White Paper,* pp. 2011-2016, 2012.

[6] (10/24/2105). *VNI Mobile Forecast Highlights, 2014 – 2019*. Available: http://www.cisco.com/assets/sol/sp/vni/forecast_highlights_mobile/index.html.

[7] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO White Paper,* vol. 1, pp. 14, 2011.

[8] (11/01/2014). *Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015*. Available: http://www.gartner.com/newsroom/id/2905717.

[9] A. B. Intelligence, "More Than 30 Billion Devices Will Wirelessly Connect to the Internet of Everything in 2020," *ABI Research News,* vol. 9, 2013.

[10] S. Cherry, "Edholm's law of bandwidth," *Spectrum, IEEE,* vol. 41, pp. 58-60, 2004.

[11] R. W. Chang, "Synthesis of band-limited orthogonal signals for multichannel data transmission," *Bell System Technical Journal, The,* vol. 45, pp. 1775-1796, 1966.

[12] J. D. Poston and W. D. Horne, "Discontiguous OFDM considerations for dynamic spectrum access in idle TV channels," in *New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005. 2005 First IEEE International Symposium On,* 2005, pp. 607-610.

[13] H. Huang and R. A. Valenzuela, "Fundamental simulated performance of downlink fixed wireless cellular networks with multiple antennas," in *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium On,* 2005, pp. 161-165.

[14] A. Sendonaris, E. Erkip and B. Aazhang, "User cooperation diversity. Part I. System description," *Communications, IEEE Transactions On,* vol. 51, pp. 1927-1938, 2003.

[15] (01/01/2015). *3GPP*. Available: http://www.3gpp.org/.

[16] M. Buddhikot, G. Chandranmenon, S. Han, Y. W. Lee, S. Miller and L. Salgarelli, "Integration of 802.11 and third-generation wireless data networks," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies,* 2003, pp. 503-512 vol.1.

[17] C. K. Toh, *Ad Hoc Mobile Wireless Networks: Protocols and Systems.* Pearson Education, 2001.

[18] Seungjoon Lee, S. Banerjee and B. Bhattacharjee, "The case for a multi-hop wireless local area network," in *INFOCOM 2004. Twenty-Third AnnualJoint Conference of the IEEE Computer and Communications Societies,* 2004, pp. 894-905 vol.2.

[19] S. Paul, R. Yates, D. Raychaudhuri and J. Kurose, "The cache-and-forward network architecture for efficient mobile content delivery services in the future internet," in *Innovations in NGN: Future Network and Services, 2008. K-INGN 2008. First ITU-T Kaleidoscope Academic Conference,* 2008, pp. 367-374.

[20] R. H. Frenkiel, B. R. Badrinath, J. Borres and R. D. Yates, "The Infostations challenge: balancing cost and ubiquity in delivering wireless data," *Personal Communications, IEEE,* vol. 7, pp. 66-71, 2000.

[21] T. Seymour and A. Shaheen, "History of wireless communication," *Review of Business Information Systems (RBIS),* vol. 15, pp. 37-42, 2011.

[22] Proakis, John G.,Salehi, Masoud, *Communication Systems Engineering.* Upper Saddle River, NJ: Prentice Hall, 2002.

[23] M. Salazar-Palma, A. Garcia-Lamperez, T. K. Sarkar and D. L. Sengupta, "The Father of Radio: A Brief Chronology of the Origin and Development of Wireless Communications," *Antennas and Propagation Magazine, IEEE,* vol. 53, pp. 83-114, 2011.

[24] T. Sarkar*, History of Wireless.* Hoboken, N.J.: Wiley-Interscience, 2006.

[25] P. J. Nahin, "Maxwell's grand unification," *Spectrum, IEEE,* vol. 29, pp. 45, 1992.

[26] A. R. Constable, "The birth pains of radio," in *100 Years of Radio., Proceedings of the 1995 International Conference On,* 1995, pp. 14-19.

[27] J. Mervis and P. Bagla, "Bose Credited With Key Role in Marconi's Radio Breakthrough," *Science,* vol. 279, pp. 476-476, 1998.

[28] A. K. Sen, "Sir J.C. bose and radio science," in *Microwave Symposium Digest, 1997., IEEE MTT-S International,* 1997, pp. 557-560 vol.2.

[29] N. J. Sloane and C. E. Shannon, *Collected Papers.* IEEE press, 1993.

[30] Q. R. Skrabec, *The 100 most Significant Events in American Business: An Encyclopedia.* Santa Barbara, Calif.: Greenwood, 2012.

[31] M. T. G. Leiva and M. Starks, "Digital switchover across the globe: the emergence of complex regional patterns," *Media, Culture & Society,* vol. 31, pp. 787-806, 2009.

[32] B. Sklar, *Digital Communications : Fundamentals and Applications.* Upper Saddle River, NJ: Prentice Hall PTR, 2001.

[33] G. Moore, "The changing face of modems," *Electronics and Power,* vol. 32, pp. 651-654, 1986.

[34] F. J. Harris, C. Dick and M. Rice, "Digital receivers and transmitters using polyphase filter banks for wireless communications," *Microwave Theory and Techniques, IEEE Transactions On*, vol. 51, pp. 1395-1412, 2003.

[35] CTIA-The Wireless Association, *CTIA Semi-Annual Wireless Industry Survey*, 2010.

[36] V. Muthukumar, A. Daruna, V. Kamble, K. Harrison and A. Sahai, "Whitespaces after the USA's TV incentive auction: A spectrum reallocation case study," in *Communications (ICC), 2015 IEEE International Conference On*, 2015, pp. 7582-7588.

[37] M. Palkovic, P. Raghavan, Min Li, A. Dejonghe, L. Van der Perre and F. Catthoor, "Future Software-Defined Radio Platforms and Mapping Flows," *Signal Processing Magazine, IEEE*, vol. 27, pp. 22-33, 2010.

[38] U. Ramacher, "Software-Defined Radio Prospects for Multistandard Mobile Phones," *Computer*, vol. 40, pp. 62-69, 2007.

[39] J. Mitola III, "Software radios-survey, critical evaluation and future directions," in *Telesystems Conference, 1992. NTC-92., National*, 1992, pp. 13/15-13/23.

[40] W. H. Tuttlebee, *Software Defined Radio: Origins, Drivers and International Perspectives*. Wiley, 2002.

[41] J. Mitola, "The software radio architecture," *Communications Magazine, IEEE*, vol. 33, pp. 26-38, 1995.

[42] J. H. Reed, *Software Radio : A Modern Approach to Radio Engineering*. Upper Saddle River, NJ: Prentice Hall, 2002.

[43] H. Tsurumi and Y. Suzuki, "Broadband RF stage architecture for software-defined radio in handheld terminal applications," *Communications Magazine, IEEE*, vol. 37, pp. 90-95, 1999.

[44] E. H. Armstrong, "A New System of Short Wave Amplification," *Radio Engineers, Proceedings of the Institute Of*, vol. 9, pp. 3-11, 1921.

[45] (10/25/2015). *Communications | Analog Devices*. Available: http://www.analog.com/en/applications/markets/communications.html.

[46] M. S. Safadi and D. L. Ndzi, "Digital hardware choices for software radio (SDR) baseband implementation," in *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, 2006, pp. 2623-2628.

[47] L. J. Karam, I. AlKamal, A. Gatherer, G. A. Frantz, D. V. Anderson and B. L. Evans, "Trends in multicore DSP platforms," *Signal Processing Magazine, IEEE,* vol. 26, pp. 38-49, 2009.

[48] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang and G. M. Voelker, "Sora: High-performance Software Radio Using General-purpose Multi-core Processors," *Commun ACM,* vol. 54, pp. 99-107, jan, 2011.

[49] (10/29/2015). *Sora - Microsoft Research.* Available: http://research.microsoft.com/en-us/projects/sora/.

[50] Ji Fang, Zhenhui Tan and Kun Tan, "Soft MIMO: A software radio implementation of 802.11n based on sora platform," in *Wireless, Mobile & Multimedia Networks (ICWMMN 2011), 4th IET International Conference On,* 2011, pp. 165-168.

[51] K. Amiri, Yang Sun, P. Murphy, C. Hunter, J. R. Cavallaro and A. Sabharwal, "WARP, a unified wireless network testbed for education and research," in *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference On,* 2007, pp. 53-54.

[52] C. Chang, J. Wawrzynek and R. W. Brodersen, "BEE2: a high-end reconfigurable computing system," *Design & Test of Computers, IEEE,* vol. 22, pp. 114-125, 2005.

[53] S. Mellers, B. Richards, H. K. -. So, S. M. Mishra, K. Camera, P. A. Subrahmanyam and R. W. Brodersen, "Radio testbeds using BEE2," in *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference On,* 2007, pp. 1991-1995.

[54] A. Tkachenko, D. Cabric and R. W. Brodersen, "Cognitive radio experiments using reconfigurable BEE2," in *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference On,* 2006, pp. 2041-2045.

[55] (10/29/2015). *Ettus Research - Home.* Available: http://www.ettus.com/.

[56] A. Badá and M. Donati, "The software radio technique applied to the RF front-end for cellular mobile systems," in , E. Del Re, Ed. Springer London, 2001, pp. 375-386.

[57] F. J. Harris, C. Dick and M. Rice, "Digital receivers and transmitters using polyphase filter banks for wireless communications," *Microwave Theory and Techniques, IEEE Transactions On*, vol. 51, pp. 1395-1412, 2003.

[58] F. Harris, *Multirate Signal Processing for Communication Systems.* Upper Saddle River, N.J.: Prentice Hall PTR, 2004.

[59] S. C. Kim, W. L. Plishker and S. S. Bhattacharyya, "An efficient GPU implementation of an arbitrary resampling polyphase channelizer," in *Design and Architectures for Signal and Image Processing (DASIP), 2013 Conference On,* 2013, pp. 231-238.