PicoRF: A PC-based SDR Platform using a High Performance PCIe Plug-in Card Extension.

Karim A. Said

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Jeffrey H. Reed, Chair
Peter M. Athanas
Carl B. Dietrich

June 22, 2012
Blacksburg, Virginia

Keywords: SDR, RFIC5, GNU Radio, Partial Reconfiguration, PCI Express

PicoRF: A PC-based SDR Platform using a High Performance PCIe Plug-in Card Extension

Karim A. Said

(ABSTRACT)

Although SDRs have been around in the research community for over a decade, they have not reached the point of transitioning to the mass consumer market, size being one of the major obstacles. Numerous SDR hardware platforms have been developed demonstrating successful functionality, yet to this day most of them remain trapped in desktop/benchtop form factors which are not suited for mobility. A main factor contributing to the size of SDR units is the RF front end. Using current technology, wide-band operation of SDR RF front-ends is achieved by aggregating multiple dedicated components, each covering a portion of the frequency range. Recent technology advances have enabled the integration of wide frequency functionality inside a single integrated package. One example is a prototype RFIC transceiver chip from Motorola Research Labs which contains a complete direct conversion RF transceiver in a single chip, with a frequency coverage range of 100MHz-2.4GHz. RFIC5, the latest version of the chip, has additionally integrated high speed ADC and DAC units, leading to a significant reduction in the component count and the overall size of the SDR hardware.

This thesis describes the implementation of a highly compact, SDR PC plug-in card, known as PicoRF. PicoRF is based on Motorola's RFIC chip for the RF front-end functionality, while the combined computational power of a Xilinx Virtex 5 FPGA and a PC host is used for waveform signal processing. An overlay gird consisting of an interconnection of PR slots is reserved on the FPGA to host the components of a signal processing pipeline which can be modified during runtime. Through a PCIe connection, partial bitstreams can be downloaded from the host PC to the FPGA at a very high speed making it possible for the radio to modify its function in very short time intervals and greatly reducing the service interruption time. Control software running on the PC host manages the overall system operation including the RFIC which is controlled through a custom developed API. The combination of the laptop host and the plug-in card form a small form factor, mobile SDR node that is one step towards satisfying both the performance and ergonomics demand of the consumer market.

This work has been funded in part by the Egyptian Government through the VT-MENA program and the US Department of Defense.

Acknowledgments

I would like to thank Dr. Reed for being my advisor, and for the great experience opportunity that he has made available to me by involving me in the MPRG Lab.

I would also like to thank Dr. Athanas who has tolerated me all this time as a heavy guest in his lab, occupying both resources and space even though I have not been working on his projects. Thank you for helping me with all the mundane issues that I have faced in the lab, such as helping me log in when I thought I had no account, and it turned out that I was misspelling my own username.

I would also like to thank Dr. Carl Dietrich who has offered me guidance and help by making resources available whenever I needed them.

Special thanks goes to my colleagues Eyosias Imana and Randall Nealy.

Thank you Eyosias for helping me with the details of the RFIC and RF in general, as well as your efforts in helping me build the RFIC API. You have been a great partner and a great friend.

Randall, without your hardware skills and experience in detecting hardware bugs, I would have remained stuck at point zero. Thank you for dedicating time for our meetings and discussions.

Finally, I would like to dedicate this work to my parents and my family. Thank you for being supportive all this time and tolerating me at times of frustration. I apologize for missing out on you and for not calling u enough.

Contents

List of Figures

List of Tables

List of Abbreviations

ACK Acknowledgement

ADC Analog to Digital Converter

ADPLLs All Digital PLL

API Application Programming Interface

ASIC Application Specific Integrated Circuit

CLB Configurable Logic Blocks

CPU Central Processing Unit

DAC Digital to Analog Converter

DC Direct Current

DCOC DC Offset Correction

DDC Digital Down Conversion

DDS Direct Digital Synthesizer

DMA Direct Memory Access

DSA Dynamic Spectrum Access

DSP Digital Signal Processor

DUC Digital Up Conversion

FFT Fast Fourier Transform

GPP General Purpose Processor

GPU Graphics Processing Unit

GRC GNU Radio Companion

GSM Global System for Mobile Communications

GUI Graphical User Interface

ICAP Internal Configuration Access Port

IEEE Institute of Electrical and Electronics Engineers

IPC Inter Process Communication

JTAG Joint Test Action Group

KUAR Kansas University Agile Radio

LNA Low Noise Amplifier

LO Local Oscillator

LUT Look Up Table

MAC Medium Access Control

MGT Multi Gigabit Transceiver

MOSFET Metal Oxide Semiconductor Field Effect Transistor

MSI Message Signalled Interrupt

OSSIE Open Source SCA Implementation::Embedded

PAN Personal Area Network

PAR Place and Route

PCB Printed Circuit Board

PCIe Peripheral Component Interconnect Express

PH Programmable Hardware

PIO Programmed Input Output

PLL Phase Locked Loop

PPC Power PC

PR Partial Reconfiguration

Qos Quality of Service

QPSK Quadrature Phase Shift Keying

RCB Radio Control Board

RF Radio Frequency

RFIC Radio Frequency Integrated Circuit

SAW Surface Acoustic Wave

SDR Software Defined Radio

SIMD Single Instruction Multiple Data

SNR Signal-to-Noise Ratio

SPI Serial Peripheral Interface

TLP Transaction Layer Packet

USB Universal Serial Bus

USRP Universal Software Radio Peripheral

VCO Voltage Controlled Oscillator

VGA Video Graphics Array

VGA Voltage Controlled Gain Amplifier

WARP Wireless Open-Access Research Platform

WLAN Wireless Local Area Network

Chapter 1

Introduction

1.1 Motivation

In the last few decades, wireless technology has witnessed an explosion in interest and demand due to the convenience it offers users. Radio technologies have been developed to enable a wide range of useful applications that have become an integral part of our daily lives, including cellular communication, PAN (Personal Area Network) such as Bluetooth and Zigbee, WLAN(Wireless LAN) etc.

In order to address the requirements of different applications, such as data-rate, multiple-access, range etc, several communication standards had to exist. For each standard, a different set of values for the radio signal parameters is used, such as carrier frequency, channel bandwidth, modulation scheme etc. Even within the same communication standard, variations exist in the radio parameters. Table 1.1 lists the parameter values across a number of common cellular communication standards. From the hardware implementation perspective, this translated into a specific set of hardware optimized to handling a radio signal defined by a single standard. Thus, such classical radios due to their fixed hardware can operate over a single standard, and as a consequence are limited to providing a single service.

Table 1.1: Parameters for Different Cellular Communication Standards

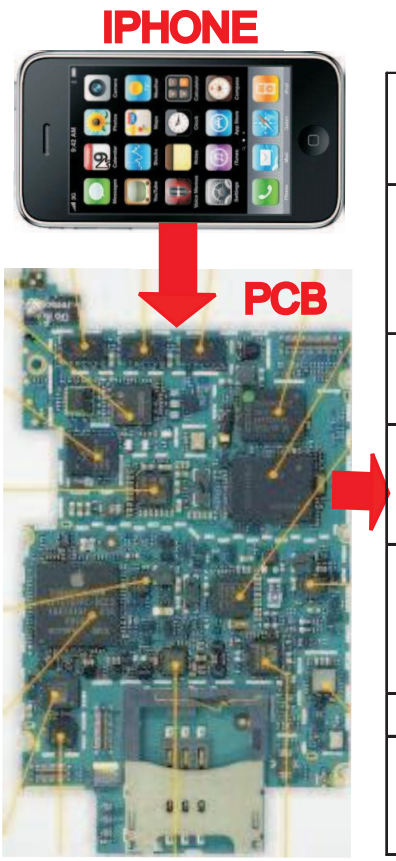
	GSM	cdmaOne	cdma2000	WCDMA
Frequency Range (MHz)	GSM850: DL(869-894) UL(824-849)	DL(869-894) UL(824-849)	450; 700	DL(2110-2170) UL(1920-1990)
	GSM900: DL(935-960) UL(890-915)	DL(1930-1990) UL(1850-1910)	800; 900	DL(1930-1990) UL(1850-1910)
	GSM1800: DL(1805-1880) UL(1710-1785)		1700; 1800	DL(1805-1880) UL(1710-1785)
	GSM1900: DL(1930-1990) UL(1850-1910)		1900: 2100	
Modulation	GMSK, 8-PSK (EDGE only)	QPSK/OQPSK	QPSK, OQPSK, HPSK	UL:Dual BPSK DL:QPSK, 16QAM (HSDPA only)
Multiple Access	TDMA/FDMA	CDMA/FDMA	CDMA	CDMA/FDMA
Duplex (UL/DL)	FDD	FDD	FDD	FDD
Channel bandwidth	200 KHz	1.25 MHz	1.25 MHz	5 MHz
Peak data rate	14.4 kbit/s, 53.6 kbit/s (GPRS)	14.4 kbit/s (IS-95-A)	307.7 kbit/s (CDMA2000 1x)	2 Mbit/s
	384 kbit/s (EDGE)	115.2 kbit/s (IS-95-B)	2.4 Mbit/s (CDMA2000 3x)	10 Mbit/s (HSDPA)

In the past, most of the research in the area of wireless communications has been directed to producing cost-effective RF transceivers dedicated to a single standard. Classical radios were designed to operate over a fixed frequency band and to handle a single type of modulation. Now the situation has changed due to the demand by customers to have multiple services that are offered across various standards. In addition, the varying spectrum allocations as well the varying standards from one geographical region to another, and the desire to achieve global service availability has rendered radios with single frequency operation ineffective. For example, even though the GSM(Global System for Mobile Communications) standard exists in most parts of the world, unless the radio device can operate in all four GSM bands, shown in Table1.1, global service availability cannot be guaranteed.

Given the constant increase in wireless applications and the number of participant users, all sharing the same spectrum, a new challenge is starting to surface in the horizon, the problem of **Spectrum Overcrowding**. Studies have shown that classical static spectrum allocations are highly inefficient in terms of the temporal utilization, with some frequency band utilizations going down to 10% [11]. Given such measurements, an opportunistic approach to spectrum access can help alleviate the problem of spectrum overcrowding. Rather than being restricted to a static range of frequencies on which to operate, radios will be allowed to use whatever unused spectrum or white space they find free. This dynamic behavior of radios can be extended to the different operating parameters including transmit power, modulation scheme, bit-rate etc. A radio can sense its environment, and based on current measurements and previous experience it can change its mode of operation to optimize a certain metric such as power consumption, SNR etc., such a radio application is commonly known as **Cognitive Radio.**

All the above has motivated the industry to look for radios that are flexible enough to change their mode of operation, and dynamic enough to complete those changes in run-time while guaranteeing service continuity.

A straightforward approach to addressing this problem is by concatenating hardware units per standard in a single device. The iPHONE is one state of the art product that follows this ap-



LIST OF TRANSCEIVER CHIPS

Manufacturer	Multi Source Probability	Component Description	Cost
Broadcom	Low	Bluetooth/FM/WLAN Single Chip, WLAN IEEE802.11b/g, Bluetooth V2.1+EDR, with FM and RDS/RBDS Receiver	$5.95
Numonyx	High	Memory MCP, 128 Mb, NOR Flash and 512 Mb Mobile DDR	$3.65
Infineon	Low	RF Transceiver, Quad-Band GSM/EDGE,Tri-Band WCDMA/HSDPA, 130nm RF CMOS	$2.80
Infineon	Low	GPS Receiver, Single Chip, 0.13 um, with Integrated Front-End RF, PLL, PM, Correlator Engine and Host Control Interface	$2.25
Infineon	Low	Power IC, RF Function	$1.25
Murata	Low	FEM, Quad-Band GSM, Tri-Band UMTS Antenna Switch and Quad-Band GSM RX RF SAW Filters	$1.35

Figure 1.1: iPHONE: Internal PCB and transciever chips used [31].

proach in providing multiple wireless services. It can support GSM, WCDMA, and 3G standards, IEEE802.11b/g for WLAN access, allows navigation via GPS, and supports data exchange via Bluetooth. Figure 1.1 shows the internal pcb of an iPHONE along with a list of the different transceiver chips and their manufacturers that are built in the device. It is clear that for each wireless standard (GSM, Bluetooth, WLAN, GPS) a separate transceiver chip is used.

This approach, however, suffers from the problem of scalability. In order to operate over the hundreds of standards available a proportional amount of hardware is required. As a result, this will lead to an increase in cost, power consumption and form factor, making this approach only practical for a dual-band or a tri-band solution.

SDRs(Software Defined Radio) represents an ideal candidate in this situation since they have the

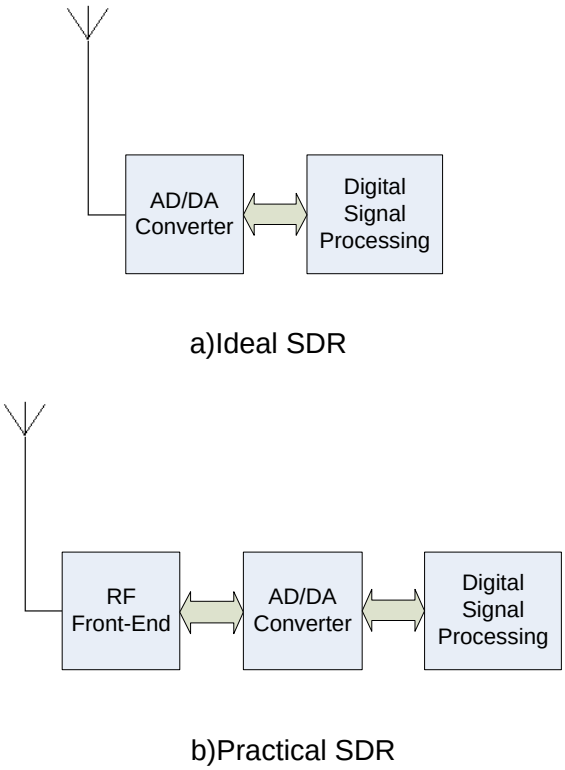


Figure 1.2: a)Ideal SDR. b)Practical SDR

potential to handle channels across a wide frequency spectrum, using an arbitrary modulation scheme and bandwidth. The strength of an SDR lies in the fact that it relies heavily on digital reconfigurable components, which benefit from hardware reuse in enabling multi-mode operation using a fixed set of hardware resources. A SDR at its core is based on transforming all classical radio processing operations from the analog domain to the digital domain in order to benefit from the programmability of digital components. The classic view of an SDR is based on what Mitola pictured in 1995 [6], where a radio is purely digital except for the antenna to which a set of analog-to-digital (ADCs) and digital-to-analog converters (DACs) are connected. This heavily digitized radio concept provides the highest degree of reconfigurability. Different modes of operation can be obtained by reprogramming the digital components to implement the signal processing functions required by a particular standard.

The Ideal SDR architecture assumes an ADC/DAC connected directly to an antenna. In this architecture, the data converter samples the RF signal and the downconversion process is completed in the digital domain, thus relieving the designer from having to use unruly analog components. Such an architecture places a set of extreme technological demands on the ADC/DAC components. [8] provides a list of such demands:

1. A high sampling rate to support wide signal bandwidths.
2. A large number of effective quantization bits to support a high dynamic range.
3. An operating bandwidth of several GHz to allow the conversion of a signal over a greatly varying (and theoretically arbitrary) range of frequencies.
4. A large spurious free dynamic range to allow for the recovery of small scale signals in the presence of strong interferers while producing very little distortion, minimal power consumption and cost.

In the same reference, an exhaustive study of ADCs and technology improvement trends has been conducted and has found that the bandwidth of a direct sampling receiver is limited by the ADC bandwidth, dynamic range, and aperture jitter. The study has concluded that "A receiver digitizing all signals between DC(Direct Current) and, say, 2.5 GHz is impossible, not just unrealizable given current technology limitations". Thus, the only way a wideband SDR can be realized is through using a flexible RF front. The RF(Radio Frequency) frontend will serve as a downconversion stage that lowers the input frequency to a suitable range that can be handled by the ADC.

Practial SDRs are typically divided into two main subsystems, RF frontend and baseband processing. The baseband subsystem consist of digital circuits that perform DSP(Digital Signal Processing) operation on the digital baseband signals to extract information, while the RF front end consists mainly of analog circuits that lie between the antenna and ADC/DAC blocks.

For the baseband processing, an SDR is required to be highly flexible in its processing capabilities to support the numerous already existing and constantly evolving communication standards,

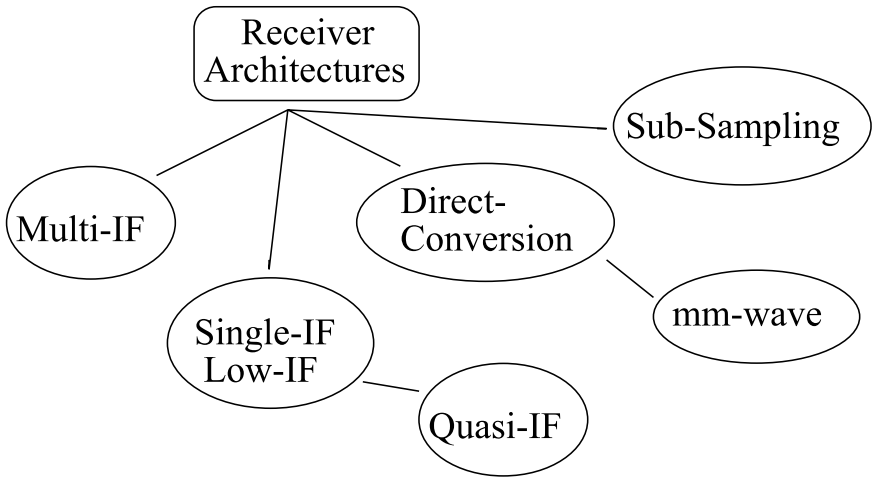


Figure 1.3: Common Radio Receiver Architectures

that ideally can be hosted in a handset form factor. Thus, a processing resource with highly conflicting requirements is needed, namely flexibility, high performance and low power consumption. The three main technologies available for hosting the digital signal processing operations are GPP(General Purpose Processors)s, DSPs and FPGA(Field Programmable Gate Array)s. FPGAs with their parallel architecture are more suited for high speed data streaming operations which exist in SDR physical layer functions such as DDS(Direct Digital Synthesizer), mixers, filters etc. GPPs and DSPs on the other hand, due to their sequential architecture, are more suited for control oriented functions that exist in the higher level layers of an SDR.

The FPGA reconfiguration model has evolved greatly since its early beginnings. FPGAs originally could only be programmed monolithically. Latest technology improvements have allowed partial resource regions of the FPGA to be reconfigured independently without interrupting the operation of the remaining FPGA resources [19]. This enables software-like programmability for FPGAs where individual component bitstreams can be loaded into the FPGA during runtime on-demand, which is known as **Dynamic Reconfiguration**. Using this technology, FPGAs offer high performance, as well as the programmability to modify the waveform signal processing component structure in runtime.

The second subsystem in the SDR is the RF frontend which, as discussed previously, is required

due to the limitations of data converter technology. The function of the RF frontend is to bridge the gap between ADC/DAC input/output signals and the RF frequency input signals. This includes components that perform functions such as down/up conversion(mixing), amplification, filtering etc. All such components share the attribute that they are designed to perform their respective functions at the high RF frequencies, and hence are collectively referred to as the RF frontend.

RF frontends can be implemented in several configurations, Figure 1.3 shows a classification of the most commonly used architectures. One particular architecture that offers attractive benefits for SDR is the direct conversion architecture [3]. It uses a minimum number of analog RF circuit blocks which helps to increase flexibility, minimize cost and power consumption. It also enables wide-band operation as required in SDRs by not having to rely on the use of tunable analog components which are either expensive or do not provide uniform tunability over a wide range [21].

The direct conversion architecture, however, has a number of drawbacks due to its particular configuration. One of them mainly being the DC offset, which is caused by leaked LO(Local Oscillator) output to the antenna due to poor isolation with the LNA(Low Noise Amplifier), the leaked signal in turn reflects back from the antenna due to impedance mismatch to cause self-mixing [21]. Another drawback that is specific to this architecture is the increased noise level due to low frequency 1/f noise contributed by MOSEFT(Metal Oxide Semiconductor Field Effect Transistor) and BiPolar active devices [24]. Also, the direct conversion architecture demands the use of I/Q mixers to retain both I and Q components. Using I/Q mixers in the analog domain makes the architecture prone to performance degradation due to I/Q mismatch. Finally, a direct conversion SDR is based on a single LO which is required to operate a very wide tuning range. Quadrature Local Oscillator (LO) generation over a wide frequency range is typically addressed with multiple high frequency VCO(Voltage Controlled Oscillator)s and complex divider schemes. However, this approach often leads to spotty and discontinuous coverage [6].

Fortunately, solutions exist to counter the shortcomings of direct conversion architectures. In [9] a direct conversion transceiver is described that uses techniques to address most problems that have been stated previously. For the DC offset problem, a DC offset correction circuitry (DCOC) is

implemented that consists of a 1-bit ADC (comparator), control logic, and a 5-bit current mode DAC that injects current into the feedback resistors of a VGA(Voltage Controlled Gain Amplifier) to cancel out the offset voltage. The control logic implements a successive approximation algorithm that converges on the correct 5-bit word that compensate for the baseband filter's DC offset. In addition, a novel chopper-based matching technique described in [7] is applied that mitigates second order inter-modulation distortion as well as 1/f flicker noise. Finally for the LO, a DDS-based frequency synthesizer is used, which helps provide the desired wide continuous tuning range, as well as cycle-to-cycle frequency and phase switching precision that is not possible using traditional phase locked loops. Although traditional direct digital synthesizers suffer from higher spurious frequency content and power consumption compared to PLL-based synthesizers, by using a ROM-less architecture in the DDS the power consumption has been reduced to below 120 mW. Using a non-zero-mean dither in the generated signal, the spurious frequency components can be limited to below -35 dBc [9].

1.2 Objective

The goal of this work is to build a powerful, flexible, yet compact SDR platform that can serve as a truly mobile SDR node, known as PicoRF. By relying on state of the art technology, such as the RFIC5 chip from Motorola which covers a frequency range of 100-2400MHz, universal frequeny operation can be achieved using a single piece of hardware. In contrast, other platforms such as the USRP(Universal Software Radio Peripheral) require multiple RF daughterboards to operate over a wide range of frequencies [15].

The PicoRF platform consists mainly of three components. The first component is the RFIC5, which is the successor of the RFIC4 [9] with the extra addition of a ADC/DAC pair integrated on the chip. This addition is of great value in reducing the form factor as it helps reduce PCB(Printed Circuit Board) area that would be occupied by external ADC/DAC components and associated circuitry, as well as simplifying PCB routing. Also, by integrating the ADC/DAC components

greater performance is achieved due to better immunity to noise. The RFIC5 provides DDS-based frequency synthesizers that control the sampling rate of the ADC/DAC. This allows the control of the sample rate during runtime which can be managed by cognitive control algorithms to optimize performance and power consumption.

The second component of the system is an FPGA hosted on PCIe card from PICO computing [1]. The FPGA performs mainly two functions. First, it hosts a PCIe controller that serves as the interface to the PC. Second, it hosts blocks to perform high speed signal processing operations on the baseband signals prior/post the RFIC stage. The FPGA also hosts a partial reconfiguration infrastructure that enables the loading/unloading of processing blocks in run-time. This feature enable the implementation of cognitive radio applications that require a reconfiguration in the radio structure based on run-time conditions.

The third component in the system is the laptop/PC host. An API(Application Programming Interface) function set exists which provides a means of controlling the PicoRF platform. It can be used either by custom developed C/C++ programs or interfaced to cognitive radio engines. A GNU radio block set is also provided which includes a PicoRF source and a sink block pair that can source/sink data into/out of the GNU Radio signal processing environment. In addition, a block is provided that can be used control the run-time reconfiguration aspects of the FPGA. Using the GRC(GNU Radio Companion), graphical control/widgets can be used to build a flexible graphical control panel that mimics the behaviour of a knobs-based radio.

Summary of Contributions:

- A high-performance PCIe driver tailored to streaming radio applications.
- Implementation of a FPGA-based high performance PCIe Transaction Layer for streaming applications.
- A run-time radio reconfiguration infrastructure with sub-ms component switch time.

- A software API that provides an organized and consistent programming model of the internal subsystems inside the RFIC5 chip.
- PicoRF blocks for instantiation inside the GNU radio environment.
- Accurate documentation of the RFIC5 initialization and control sequence for the different internal blocks.

1.3 Organization

The organization of the thesis is the following. Chapter 2 provides a survey of the different classes of existing SDR platforms and examples of each, followed by a discussion of some of the main shortcomings and advantages of each platform. Chapter 3 contains a detailed description of the the two main subsystems that comprise the PicoRF platform, the hardware and software subsystems and the constituent components of each. Chapter 4 introduces the RFIC5 chip and the various internal blocks and a list of the associated API control functions. The 2 GNU Radio blocks PicoRF_rx_src and PicoRF_tx_sink will be presented with the different parameters of each block defined. Chapter 5 presents performance measurements including max read and write throughput and reconfiguration time, in addition to full system runs that verify the correct overall operation of the platform starting from the RF front end all the way to the GNU Radio control GUI on the PC. A multiple waveform radio application will be run that involves switching waveforms in run-time by reconfiguring the FPGA hardware both manually and automatically based on a trigger event from the radio environment. Finally, concluding remarks, unfinished and future work are outlined in Chapter 6.

Chapter 2

Background

Wireless radio communication began in the late 1800's and early 1900's with many different methods of implementation. These methods have changed drastically over the last century starting with devices like the crystal radio all the way to today's modern digital radios. With the introduction of SDRs since the beginning of the last decade, radio implementation methodologies have witnessed a paradigm shift. Radio architectures, however, remain the same consisting of the same set of basic radio building blocks such as oscillators, mixers, filters, amplifiers etc. In SDRs, components are being replaced with their digital or software-like counterparts. For example, a crystal based oscillator is replaced with a DDS to perform the function of the LO, PLL(Phase Locked Loop)s are being replaced with ADPLLs(All-Digital PLL) and mixers are replaced with arithmetic multipliers. Such components exist in code or software-like form that can be implemented on any digital processing technology such as ASIC(Application Specific Integrated Circuit)s, FPGAs, GPPs, GPU(Graphics Processing Unit)s etc.

This chapter will attempt to cover some of the popular implementation methodologies and enabling technologies used in SDR platforms.

2.1 Classes of SDR Platforms

A typical SDR architecture is based on a small set of components, yet with high performance, to provide the required flexibility. The components are: a high speed ADC/DACs that can operate in the range of 100s of megahertz , a powerful digital processor such as an FPGA, DSP or GPP, and a flexible RF front end. Advances in technology have lead to the availability of high performance SDR components for a reasonable cost and a low power consumption. This has sparked the interest of industry and academia over the course of the last decade to explore the various possible configurations of components and technologies in implementing SDR platforms.

A common classification criterion for SDR platforms is the digital signal processing technology employed [12]. According to this criterion we can find three main classes:

1)GPP-centric SDRs:

In GPP-centric SDRs, the signal processing blocks manifest in the form of software functions made up of sequential operations that can run on a GPP. This architecture assumes the existence of a standard operating environment that will serve as an abstraction layer to higher level software composed of signal processing blocks, such as the LINUX OS. This way the standard PC development environment can be used in developing the radio components, which is both familiar and simple. Using OS facilities such as piping and IPC(Inter Process Communication), standard PC applications can be linked to the radio software to create new and innovative applications. For the user interface, a standard PC environment is typically accompanied with a powerful graphical subsystem that can be used to build virtual panels including knobs, sliders, buttons etc., to control the various parameters of the radio.

The main disadvantage of GPP SDRs lies in their performance, mainly due to the sequential processing nature of a GPP. Although the GPP can be scheduled to execute multiple operations in a time-shared fashion, the existence of an OS layer that handles the scheduling and communication between signal processing blocks adds overhead that reduces the effective performance capacity of the GPP.

2)PH(Programmable Hardware) SDRs:

In Programmable Hardware SDRs, all or most of the waveform signal processing operations are carried out on technologies such as FPGAs and CPLDs. These technologies are made up of an array of programmable primitive hardware resources such as gates or LUT(Look Up Table)s, and a grid of programmable interconnects. By programming the primitives and the interconnects, different complex functions can be realized.

PH SDR architectures offer high performance due to their abundance in hardware resources that can be programmed independently enabling parallel processing. A set of resources can be dedicated to implementing each function block in a signal processing pipeline greatly increasing throughput. In addition, parallel architectures such as in FPGAs, allow processing operations to be executed at lower clock rates leading to reduced power consumption as compared to GPPs.

Although programmable hardware offers a certain level of flexibility as compared to conventional hardware technologies such as ASICs, the desired level of programmability needed for SDR applications has not yet been achieved, which is that of software-like programmability. Another major problem with PH SDRs is that of development time. For example, standard FPGA design flows lack the incremental capability of software build tools which leads to wasted time in compiling unmodified blocks. However, recent research efforts have developed new techniques to improve FPGA development productivity [13] [30]. Also, a PH SDR lacks the advantage of a mature user and application interface environment that a PC-based GPP SDR can provide.

3)Hybrid SDRs

Hybrid platforms combine the best of both worlds by including both a GPP and PH components. In terms of processing power, the heavy processing load can be delegated to the PH, while the cognitive decision control functions can be handled by the GPP. The PH will be physically placed closer to the signal source/sink(ADC/DAC) where the sample stream rate is high. There it can handle the initial high performance, high speed tasks of channelizing, decimation/interpolation and downconversion/upconversion. On the other hand, the GPP will be placed downstream where the sampling rate is low to carry out light-weight signal processing tasks as well as user interfacing

functions such as, control knobs etc. In addition, cognitive engine software runs in the application layer and use algorithms that typically follow a non-linear control flow for which GPPs are more suited.

The following section reviews some of the SDR platforms that are either developed or commonly used in the research community. The architecture of each platform will be discussed, its advantages and disadvantages.

2.2 Examples of SDR Platforms

2.2.1 USRP(Universal Software Radio Peripheral)

Arguably, the first and most popular platform to attract the wide attention of the research community. It has enabled the actual implementation of over-the-air SDR algorithms in real time and not being limited to simulation. It's open source nature and ease of use facilitated by the GNU radio environment has simplified building radios to the point that the mass audience of programmers and hobbyists, with minimal background in radio engineering, can build their own radios. The USRP is a peripheral designed to interface with general purpose computers to function as high bandwidth software radios. The basic design philosophy behind the USRP is to do all the waveform-specific processing on the host CPU, while taking care of a set of fixed high-speed general-purpose operations, including digital up and down conversion, decimation and interpolation, on an external FPGA.

The platform consists of a motherboard that hosts high speed ADC/DACs, a FPGA and slots for RF frontend board modules known as daughterboards [1] [15]. The board connects to the PC using a USB connection that can handle data rates up to 32 Mbytes/sec. Later versions of USRP use a GigE connection with the PC that allows data rates up to 1Gb/s. The FPGA interfaces with the ADC/DAC and performs signal processing operations on the high speed signal stream which

[1] USRP1 has four daughterboard slots, 2 for RX and TX each, while the USRP2/N-xxx have 2 slots, one for TX and RX each.

include DDC(Digital Down Conversion) and DUC(Digital Up Conversion). Figure 2.1 shows the external appearance of the actual board as well as a view of the internal blocks. The PC side is responsible for two main functions. It provides a user interface to control the parameters of the radio blocks such as gain, baseband bandwidth, decimation/interpolation factors etc., as well as a graphical set of blocks to help the radio programmer view the signal characteristics such as a scope, FFT(Fast Fourier Transform), constellation plot etc. The second function is to process the baseband samples through signal processing blocks running within the GNU radio environment.

The initial version, USRP1, contained an ADC that operated at a sample rate of 64MS/sec. The maximum sample rate that can be processed at baseband is limited by the USB transfer rate(32MB/s) to 16MS/s and a bandwidth of 8MHz. In the latest versions, USRP-N series, the ADC runs at 100MS/sec for 8-bit sample sizes and 50MS/sec for 16-bit samples. However, the limit on the maximum allowable sample rate, and thus signal bandwidth, is determined by the computational load involved in the processing operations applied on the signal samples. Equation 2.1 shows the factors that impact the maximum sampling rate. In the case of a GPP, all operations are executed sequentially leading to a sample period proportional to the number of processing operations required. While an FPGA can handle multiple operations in parallel allowing for a smaller sampling period, scaling the GPP clock to N times the FPGA clock will not provide a performance similar to that of an FPGA that can run N operations simultaneously. This is mainly due to the overhead involved in the time scheduling process which scales with the number of operations, in addition to the latency introduced by the queueing.

$$Number\,of\,Operations/Sample \times Sec/Operation = Sampling\,Period \tag{2.1}$$

Even though the USRP contains a PH component i.e. the FPGA, it is only statically programmed to perform a predefined set of functions, while the user defined runtime processing blocks are limited to the PC side. Wider bandwidth signals or waveforms with greater processing demands could be have been attained if DSP operations are offloaded to the FPGA. However, the USRP's FPGA capacity is almost fully occupied by the DDC and DUC functions leaving no room for

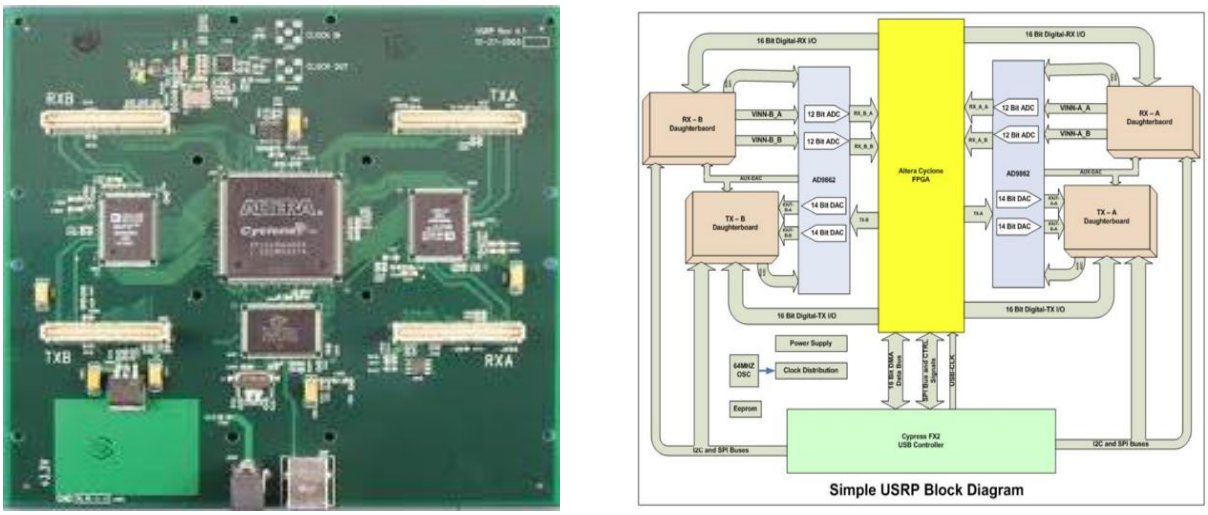


Figure 2.1: (a)USRP Motherboard, ©Matt Ettus. Used with Permission, see Appendix A. (b) USRP Internals Block Diagram.

user functions. In addition, downloading user-defined functions to FPGA is not straightforwardly supported by the GNU radio environment nor the USRP. Doing so involves modifications to both the FPGA firmware as well as the PC drivers, which is typically too much effort or considered a relatively advanced skill for the average radio designer.

2.2.2 WARP(Wireless Open-Access Research Platform)

WARP is a mutli-node SDR platform developed at RICE university to prototype wireless communication algorithms for educational and research oriented applications [12] [23] [4]. It is very similar to the USRP design in that it also based on a motherboard hosting an FPGA, and slots for RF frontend daughterboards but leaves the ADC/DACs on the daughterboards instead of being hosted on the motherboard. In contrast to the USRP, all the waveform signal processing blocks are hosted on the FPGA enabling much higher bandwidth and performance. The board connects to a host PC through an Ethernet link. One of the unique features of this platform is its scalabilty. By using the multi-gigabit transceivers built into the Xilinx FPGAs, with each MGT(Multi Gigabit Transceiver) providing a full duplex connection with a speed up to 3+ Gb/sec, a high speed low-

latency connection can be made to connect FPGAs on multiple WARP nodes. This can be used to either provide a higher processing capability through parallelism, or an extension to resources to accommodate larger designs.

The PHY layer processing is handled mainly through dedicated signal processing blocks using FPGA logic. The FPGA used is a Xilinx Virtex-II Pro [2] which contains a PPC embedded processor core. The PPC(Power PC) core serves the function of interpreting and routing commands from the PC to the different FPGA blocks as well as running high-level control algorithms [12].The PPC also serves the function of implementing the MAC and Network layer functions that interface with the PHY functions implemented using the other FPGA resrouces.

The embedded GPP i.e. the PPC core, can offer several advantages compared to being in a physically separate chip. Better coupling can be achieved between components running on the GPP and others implemented using FPGA logic leading to lower latency. The on chip domain allows for larger data interfaces as well as a common clock which can help create low-overhead interfaces and enhance performance. On the other hand, the embedded GPP setup is usually associated with a limited memory resource as compared to a PC-based GPP, making it unsuitable for standard PC applications since they are based on the virtual memory model where a large memory resource is assumed. Custom code development tools will be required, losing all the benefits of the standard PC development flows and tools. Also, the lack of memory requires the sacrifice of supporting software components such as the OS, resulting in an increased effort in code development and loss in flexibility.

In order to the aid the radio designers in accessing the wide array of features and capabilities of the WARP platform, a set of tools and packages known as the Platform Support Packages are provided. WARP provides four layers of functionality with well defined interfaces between the layers. By dividing the design across the four layers the designer can focus on a single layer at a time and is not overwhelmed by the complexity of the full system. The Platform Support Packages provide support for each of the four layers, which are:

[2]The latest version of the WARP uses the more advanced Virtex 4 FPGA.

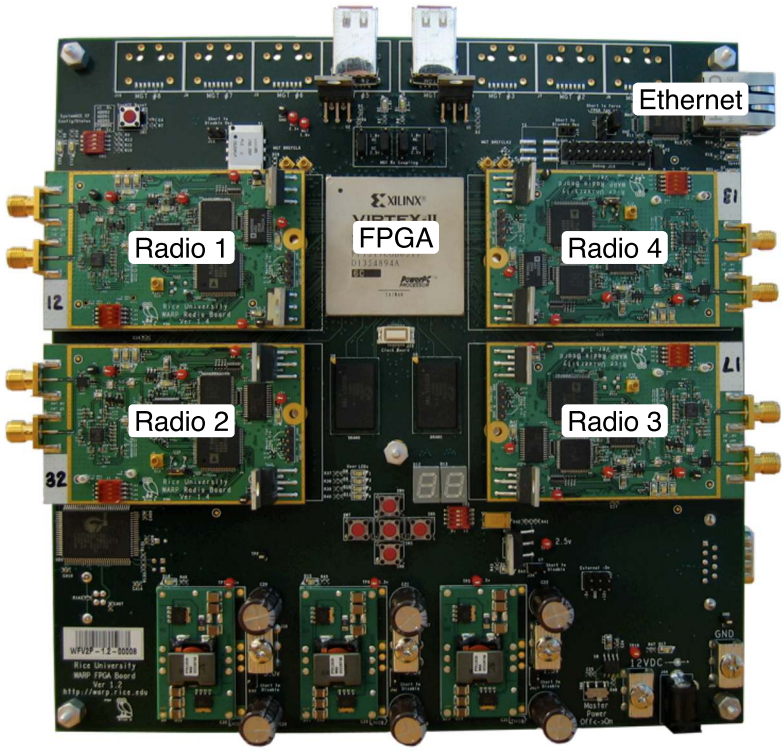


Figure 2.2: WARP Motherboard with 4 RF Daughterboards, ©Christopher Hunter. Used with Permission, see Appendix A.

- Low-Level Support: This is the lowest level and is concerned with PHY layer components implemented solely in FPGA logic.
- Logic-PowerPC Support: The second level, consisting of code running on the PPC to implement MAC and Network functions. Support is provided in the form of an interface mechanism to connect between PPC code and the PHY layer components implemented in the FPGA resources.
- Peripherals Support: The third level, mainly responsible for the interfacing between the FPGA and the various off-chip peripherals such as RF frontend boards.
- Board-to-Board Support: The final layer of support packages, which enables board-to-board communications for applications which require multiple FPGAs working in concert.

2.2.3 KUAR(Kansas University Agile Radio)

KUAR, an SDR platform developed at the University of Kansas [22]. It is based on the hybrid architecture which includes both a GPP and a PH that share the implementation of the radio functionality. It was developed with the intent to facilitate research in the areas of wireless radio networks, dynamic spectrum access and cognitive radios.

Most SDR platforms rely on an external PC host for control, user interface and development, which effectively increases the overall size of the SDR platform. KUAR integrates multiple board subsystems within a single package, including a GPP subsytem that can run a Linux OS and is equipped with standard connectors such as USB(Universal Serial Bus), VGA(Video Graphics Array) and ethernet for connecting peripherals. This particular feature makes KUAR a self-contained SDR unit that can function on its own without requiring an external PC, and consequently leading to a reduction in the overall size.

The KUAR is designed with modularity in mind, it consists of three printed circuit boards as shown in Figure 2.3, a power supply board, a digital board that hosts a GPP, a FPGA and the ADC/DAC chips, and finally an transceiver board that hosts the RF frontend. The modular design facilitates the replacement of the RF frontend board to operate over a different frequency band, as well as the use of third party prototypes for the purposes of experimentation and testing. The boards are packaged within a shielded box that has a relatively small form factor, 7 inches tall, 3 inches wide, and 6 inches deep.

In KUAR there are three main processing components: FPGA, PPC cores inside the FPGA and a GPP. The designer has the flexibility of locating signal processing blocks across all three components as he sees fit. This allows extremely parallel and time-sensitive operations to be moved into reconfigurable logic on the FPGA while more complex operations involving convoluted control flow can be implemented in software.2.4 shows the three possible scenarios for allocating waveform signal processing components across the different processing resources. The first scenario is a full hardware implementation where all radio functionality is implemented exclusively using

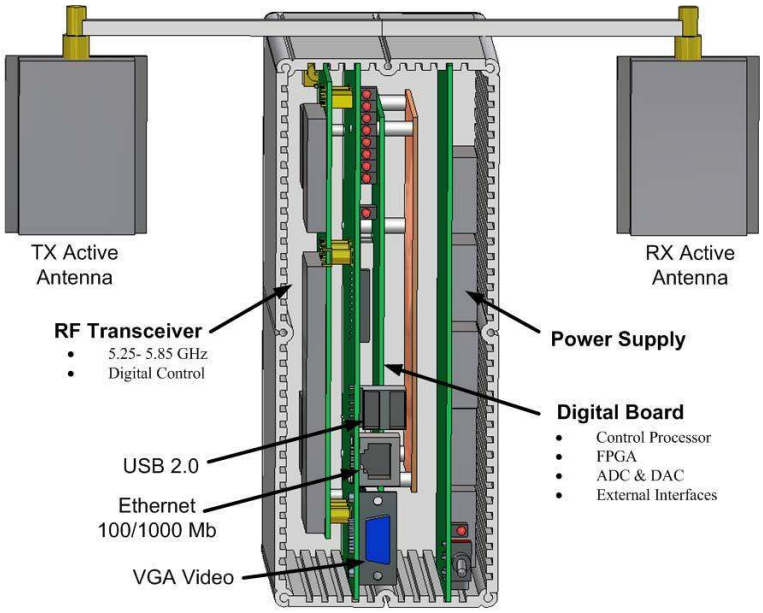


Figure 2.3: KUAR SDR Unit [22]

FPGA resources. The GPP plays no role in the signal processing and only sends the raw data into the FPGA through a specialized interface known as the KUAR memory interface which maps a set of control registers on the FPGA to addresses in the GPP memory space. The second possible scenario is a hybrid hardware/software implementation. This involves the two PowerPC cores on the FPGA which are used as general purpose RX and TX processors running general purpose code. Components implemented in the FPGA logic connect to the PPC cores through standard PowerPC peripheral buses to serve as hardware accelerated functions that are called by the PPC code. The third and last possible scenario is the full software scenario where all processing components are implemented as software running on the GPP.

2.2.4 SORA(High Performance Software Radio Using General Purpose Multi-core Processors)

SORA [29], a project by Microsoft that aims at enabling a high performance SDR platform based solely on the capabilities of commodity PCs. The target is to push as much communication func-

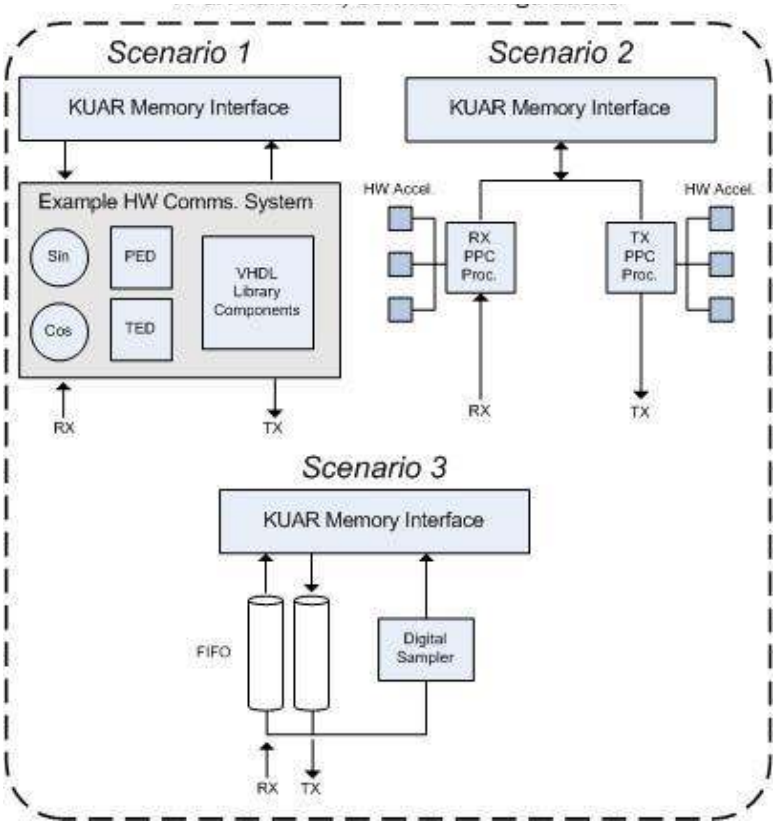


Figure 2.4: Different scenarios of component allocation across hardware and software [22].

tionality as possible into software, while keeping hardware additions as simple and generic as possible. The use of pure GPP SDR platform as compared to specialized devices such as DSPs or FPGAs can provide a number of useful benefits [26]:

1. **Portability**: Due to the more mature and standardized run-time environments used in the PC domain, code written for one general purpose CPU(Central Processing Unit) can easily be ported to other architectures and operating systems. This facilitates software reuse by sharing pieces of code between developers in the form of libraries of common functions, saving a great deal of development effort, especially if the code is well abstracted and modularized. In addition, due to the independence of code of the underlying processing hardware, hardware upgrades can be easily introduced to provide performance gains while guaranteeing that the software will function without any modifications.

2. **Availability of Development Tools**: Development tools for general-purpose computers are widely available at low or almost no cost including a large number of tools under open source licenses.

3. **Wider user base**: Due to the ubiquitous application of PCs, GPP SDRs can easily target an enormous user base. For the developer base, which is also huge compared to other specialized devices, GPP SDR development will be a familiar task requiring minimal extra knowledge.

4. **Application Integration**: SDR code can be integrated with other standard software running on the same computer enabling new models on how applications configure and use communication primitives available to them resulting in an array of innovative applications

Nevertheless, GPP-based SDR platforms are commonly known to lack the capability to handle the requirements of modern wideband standards, such as IEEE 802.11 [26] [29]. [29] identifies some of the shortcomings that have hindered classical GPP-based SDR platform such as the USRP from handling modern wireless standards and how they have been addressed in SORA.

1. **High system throughput**: The interfaces between the radio frontend and PHY as well as between some PHY processing blocks must possess sufficiently high throughput to transfer high-fidelity digital waveforms. In the case of a GPP SDR, the weakest link is the interface between the radio frontend peripheral and the PC. For example, to support a 20MHz channel for 802.11 the interfaces must sustain at least 1.28Gbps(4 samples per cycle, 16 bits I and Q). Conventional interfaces like USB 2.0 (480Mbps) or Gigabit Ethernet(1Gbps) cannot meet this requirement.

2. **Intensive computation**: High-speed wireless protocols require substantial computational power for their PHY processing. Conventional GPP techniques are at a disadvantage compared to DPSs with their architectural enahncements, and FPGAs with their abundant resources that enable multiple processing units to run simulateneously.

3. **Real-time enforcement**: Wireless protocols have multiple real-time deadlines that need to be met for correct operation. Some MAC(Medium Access Control) protocols also require precise timing control at the granularity of microseconds, one example is the channel sensing to detect if the channel is free for transmission. Another case is the ACK(Acknowledgement) response latency in the 802.11 MAC protocol which is required to be on the order of tens of microseconds. Since the SDR software is assumed to be running on top of a general non-realtime operating system, the non-determinism of the OS due to its time shared nature between multiple tasks will introduce some form of latency or jitter in moving samples from the application to the physical layer. At the physical interface level, certain bus standards like Gigabit Ethernet introduce a considerable amount of latency. All this makes responding to external events in a timely manner and with low latency impossible using the conventional existing software architectures.

By addressing each one of these shortcomings, the SORA platform has been able to attain the high performance needed for handling wideband standards. For the waveform sample transfer rate and latency, SORA uses a minimal RCB(Radio Control Board) that bridges the RF frontend signal source to PC memory over a PCIe bus. With this bus standard, the RCB can support 16.7Gbps

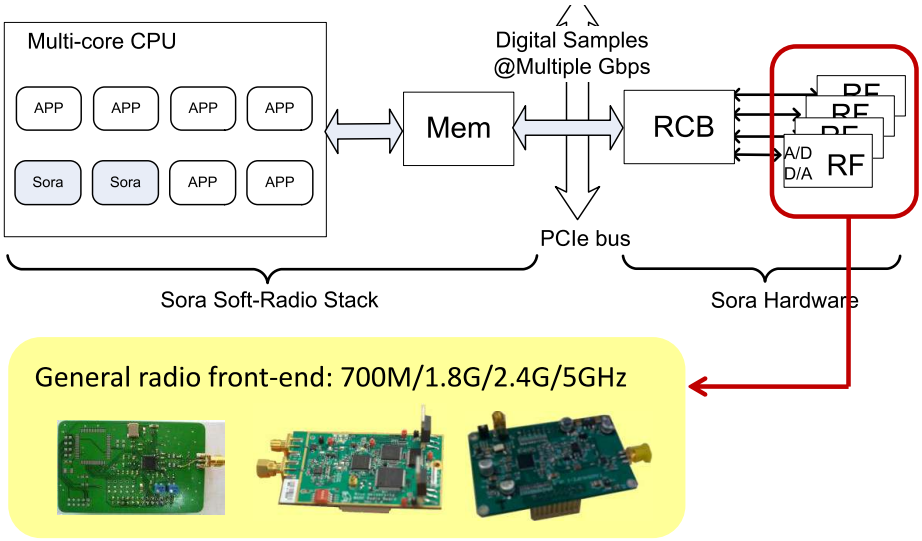


Figure 2.5: SORA Architecture.

(x8 mode) throughput with sub-microsecond latency, which satisfies the throughput and timing requirements of modern wireless protocols while performing all digital signal processing on host CPU and memory.

In order to enhance the computational capabilities required for waveform processing, SORA makes full use of various features of widely adopted multi-core architectures in existing GPPs. By relying on the abundant low-latency cache memory, signal processing components can use LUT-based implementations which greatly reduces the computational load. In addition, SIMD (Single Instruction Multiple Data) extensions in existing processors can help further accelerate PHY processing. For real-time tasks, SORA provides a new kernel service known as core dedication, which frees processor cores exclusively for a set of tasks. This helps prevent any interruption or delay due to scheduling activity especially when a large number of applications are running.

Although a library of components is provided in SORA, developers will likely have to develop their own components for their custom radios. The specialized techniques used in implementing components might require knowledge of low-level OS details on the part of the radio programmer. Some of the techniques used in [29] include explicit manipulation of cache memory, other techniques involve using SIMD instructions which are all considered advanced knowledge for the

average programmer. This particular aspect compromises one of the main benefits of GPP SDRs, which is the simplification of radio development and prototyping.

2.3 Run-time Reconfigurability/Programmability of SDRs

One of the main goals in developing SDRs is to produce radio terminals with maximum flexibility. [10] describes that the flexibility of a terminal requires the system to be adaptive and reconfigurable. The system is adaptive if it can respond to application changes by properly altering the numerical value of a set of parameters. It is reconfigurable if it can be rearranged, at a procedural, structural or architectural level.

Given the dynamic nature of SDR based applications such as DSA(Dynamic Spectrum Access) and Cognitive Radio, change of components in the signal processing chain is expected to happen during run-time as part of the operation of the radio. For example, a cognitive radio might decide to change its modulation scheme based on the changes in the environmental conditions or to improve the QoS(Quality of Service). This in turn will require a change of certain blocks within the signal processing chain. For example, switching from QPSK to BPSK involves a change in the Bit-to-Symbol mapping block. To provide such behaviour, two requirements are implied about the capabilities of the processing technology used. First, programmability must be fine-grained i.e. down to the component level. Second, programming must be fast enough to complete within some real-time constraint in order not to lose any data during the component switching interval. This feature is known as **service continuity**.

The programming model of GPPs satisfies the first requirement very easily since processing components are in the form of software modules or functions that can be loaded into memory whenever needed, and there is no limit on the how small the component can be. The second requirement is also satisfiable depending on the size of the loaded module versus the switching time constraint. The case is different with FPGAs mainly due to their primitive programming model. The following section investigates the classical FPGA programming model and how it has evolved through

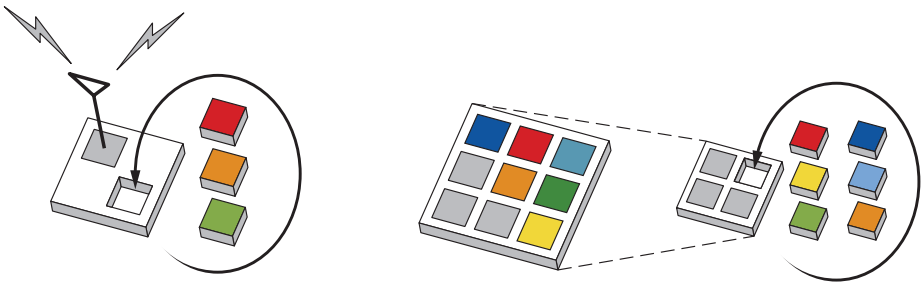


Figure 2.6: Partial Reconfiguration Concept.

efforts from both academia and the industry to allow higher programming flexibility.

2.3.1 Dynamic Switching/Swapping of FPGA components in SDRs

Different types of SDR platforms were discussed in a previous section, some of which relied on FPGAs as the sole processing element, others used a combination of both FPGAs and GPPs. In all such platforms, the FPGA could only be statically configured to perform a fixed set of tasks that cannot change during run-time. In order to modify the radio components on the FPGA, the system has to be brought to a halt, programmed and then restarted, unlike the case with GPP components which can be easily changed during run-time.

FPGA technology originally had a very limited programming model where an FPGA could only be programmed monolithically, and not on a partial basis. This made the idea of dynamic module loading on FPGAs inconceivable. A trivial solution would be to use a large FPGA that will contain all the possible processing components and to switch between them as needed. However, this is a highly inefficient approach especially when only a small subset of the FPGA components will be active at a certain time, leading to an increased cost due to the usage of a large FPGA which is highly underutilized. This is in addition to the problem of wasted power consumption caused by the idle components in the system. The other trivial solution is to reconfigure the entire FPGA with a new design containing the modified components. However, in most practical cases only a small subset of the design is required to change. Following this approach will lead to an unnecessarily long configuration time especially for modern high density FPGAs. Another

problem with this approach is that the operation of the whole FPGA will have to be interrupted. This is not desirable given the parallel nature of FPGA designs and the possibility of having several independent subsystems running on the same FPGA.

With the advent of partial reconfiguration technology [20], new usage models for FPGAs have become possible, bringing them closer to the software-like programability of GPPs. Using PR configuration the granularity of programming can go down to the component or module level. This has brought about the possibility of switching components of a design during run-time without interrupting the overall operation of the system.

In [10], an analysis of the main characteristics of widespread communication standards for physical layer functions has shown that most standards share a number of common functions. Thus, in order to switch between standards only a small subset of the components is required to change. The physical layer is most critical in terms of computational power and required throughput. FPGAs along with the capability of being partially reconfigured present an ideal candidate for enabling re-configurability at this layer, which is the main target of SDRs. In the same reference, the Sundance FPGA/DSP platform has been used to demonstrate the feasibility of FPGA partial reconfiguration in switching between different standards. For the sake of evaluation, a switch between QPSK(Quadrature Phase Shift Keying) to 8PSK mapping was used. This involved changing the bits-to-symbol mapping component and the clocking logic to generate the appropriate bit clock as shown in Figure 2.7.

Using partial reconfiguration, the authors' findings was a 50% saving in the bitstream storage. Instead of having to store 2 versions of the full bitstream that correspond to the 2 processing chains, only one version of the full bitstream plus a negligible partial bitstream are required to be stored. This is important in embedded environments where memory storage resources are limited. With respect to reconfiguration time, a 90% saving is achieved because of the negligibly small size of the partial bitstream as compared to the full bitstream. Thus, a quick switch between the two modulation schemes can be completed reducing the chances of data loss in the process and possibly introducing a small latency.

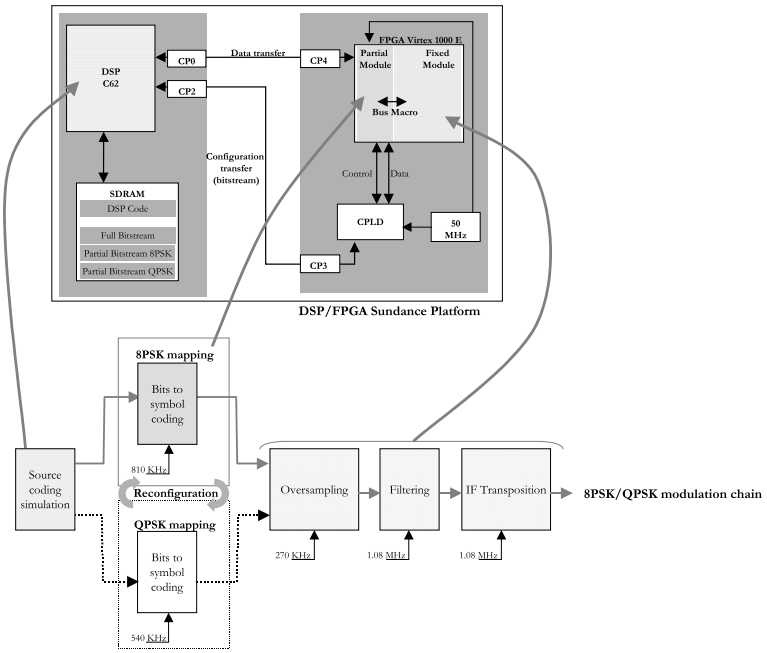


Figure 2.7: Component modification to switch between 8PSK and QPSK modulation schemes [10].

2.3.2 Standard FPGA Partial Reconfiguration Model

The concept of partial reconfiguration existed prior to its integration into mainstream tools. JBITS [14] was a tool that provided a means for low-level bitstream modifications. It also provided support for partial reconfiguration by automatically detecting the difference between the original and modified bitstreams and downloading only the difference frames. It wasn't until partial reconfiguration was integrated into the mainstream vendor tools that they started to gain wide acceptance. This was mainly due to the proprietary nature of tools such as Jbits and the lack of proper documentation.

Partial reconfiguration was first introduced into mainstream tools by Xilinx [20]. The initial design flow had restrictions like requiring PR regions to occupy the full device height. Since then the design flow has improved greatly allowing the flexibility of allocating rectangular PR regions of arbitrary size given the width is no less than 4 CLB(Configurable Logic Blocks)s. The Xilinx methodology for partial reconfiguration is based on separating the design into two regions, a static

region and a set of dynamic regions called **slots**. The static region will contain all the components that will be fixed throughout the run-time of the design. Slots are resource regions that are unused by the static design and are reserved for the modules that will be dynamically loaded during run-time. A slot is made up of a rectangular region of resources of fixed dimensions, and a fixed interface i.e. port width and number, and fixed position for a selected device.

The main problem with the slot-based model is that only one component can be inserted into a slot at a time. Given the possibility that components targeting a slot may vary widely in their resource requirements, a slot will have to be sized to accommodate the largest component, and this will lead to slot resources being wasted for most of the time. For example, a collection of filters designed for placement in a given slot where one filter is substantially larger than the others will make the targeted slot underutilized the majority of the time. This situation becomes worse when a design allocates more than one slot where the sum of wasted resources in all slots, if utilized, could have been used to accommodate extra components.

2.3.3 Improved Partial Reconfiguration Models

In [5] a new model for partial reconfiguration was developed that addresses the shortcomings of the classical model. Instead of allocating isolated slots to host dynamic components, a large ”sand box” region is allocated into which multiple modules can be inserted. This way, the number of components is not limited by the number of slots, and the resources are more efficiently divided between the dynamic modules.

Unlike the slot-based model, in this model the assumption is that the number and possibly the arrangement of dynamic components inside the sandbox can vary during run-time to provide maximum flexibility. This raises the issue of inter-component routes which have an infinitely large number of possibilities and thus will have to be generated dynamically during run-time. This is in contrast to the slot-based model where the interconnections between slots are fixed and are predetermined during compile time. Addressing this issue using the standard vendor routing tools

is impossible due to their long execution times which are not suited for quick run-time operation. In addition, the vendor tools follow a monolithic approach that involves the full design (static+dynamic) in the PAR (Place and Route) process, and this will degenerate the model to a redesign of the whole system which defeats the original premise of dynamic reconfiguration.

The authors of [5] have developed a custom set of light-weight tools to handle the inter-component routing and placement inside the sand box. Modules are relocated and placed based on an algorithm aimed at efficient use of the reconfigurable region and reducing routing delays between modules. The router used for module connections is both lightweight in terms of memory usage and fast in execution [28].

2.3.4 Extended GNU Radio

In the classical GNU Radio flow, the FPGA is a static component that does not participate in the same dynamic flexibility as the the Python flow graph components which run on the PC GPP. The FPGA is statically configured with an assortment of parametrized components. As a result, the principle agility of GNU Radio comes from software alone.

In [17], an augment based on the AgileHW project [17] is added to the GNU Radio framework which allows part of the signal flow graph component to target an FPGA co-processor. The aim of this is to achieve higher signal processing performance, especially for radio designs that demand processing requirements beyond what GPPs have to offer, by relying on the power of the FPGA. The partial reconfiguration capabilities of the AgileHW allows the rapid synthesis of FPGA bitstreams, helping maintain the productivity and flexibility of the GNU Radio environment.

2.4 SDR form factor and mobility

Advances in technology have helped realize the dream of SDR radios by making available the needed building blocks such as high-speed ADCs, programmable hardware, RF frontends etc. An

Table 2.1: SDR platform features that impact mobility

SDR Platform	WARP	KUAR	SORA	GNU Radio + USRP
Setup/Size	2 units, External board + Control Laptop	1 unit : 7 x 3 x 6	Desktop computer with PCI express card.	Laptop + USRP board
RF frontend.	Uses multiple boards.	Uses multiple boards.	Uses multiple boards.	Uses multiple boards.
Power Supply	Requires external power supply	Battery-powered	Powered from the PC	Requires external power supply

SDR being a wireless device, can only realize its true potential by being mobile. Mobility implies mainly two things: small size and low power consumption. These two requirements are in conflict with the flexibility and generality required in an SDR. In contrast, classical radios due to their specific function can optimize their hardware to reduce both the size and power consumption of a radio terminal.

One of the main contributing factors to the size of an SDR is the generic RF frontend. SDR applications demand that the hardware be capable of operating over a wide range of frequencies. The RF front end is the only block in the SDR that contains analog components, which are known to be bulky and not easily integrated on chip. This includes components such as SAW(Surface Acoustic Wave) filters, which are commonly used in super-heterodyne receivers. A larger number of components that are even bulkier become necessary when a wide range of frequencies is to be covered, such as tunable RF filters. Direct conversion architectures [3] use a minimum number of external analog off-chip components and this has made them the preferred RF frontend architecture for SDRs. However, compact general-purpose SDR terminals are yet to become a reality.

In section 2.2, a number of common SDR platforms have been examined and it can be argued that none of the platforms can serve as a truly compact mobile SDR terminal. Table 2.1 shows a comparison between the different SDR platforms and some of the main features that impact their size and mobility.

As pointed out by Table 2.1, in order to achieve universal frequency coverage all SDR platforms mentioned rely on multiple RF frontend boards. For example, the USRP has the following set of daughterboards for operating over the different frequency ranges [15], basic TX/RX boards, LFTX/RX[0-30MHz], TVRX[50-860MHz], DBSRX [0.8-2.4GHz], RFX. Having to use multiple frontend boards not only increases the size of the setup, but also causes the inconvenience of having to interrupt the operation of the radio when switching between frequency ranges.

Applications where simultaneous operation over multiple frequency bands is required are not possible using the multiple swappable board scenario. One example is developing SDRs for solving the public safety interoperability problem [16]. The problem occurs when different public safety radios are unable to communicate with one another. One aspect of the problem is that some public safety radios operate in the VHF range, around 150 MHz, others in the 700/800 MHz public safety band, others in the 400 MHz band. Typical public safety radios can operate in one of these ranges, but not the other two. They can communicate with other public safety radios in only one frequency range. Building a software radio solution using the standard USRP would require using multiple USRP boards since a USRP can hold only 2 daughterboards(USRP 2 can hold only one), each holding a set of different daughterboards.

2.4.1 MOTOROLLA RFIC

In [9] a flexible low power RFIC transceiver is described that can serve as a basis for a compact universal RF frontend. It can be programmed to operate in any frequency within the range of 100 MHz - 2.5 GHz, with channel bandwidths from 8 KHz to 20 MHz. The RFIC offers great flexibility by providing a large array of programmable parameters such as baseband bandwidth, transmit/receive gain, LO frequency etc.

The RFIC uses a number of techniques, including novel ones, to overcome problems that have encumbered wideband transceivers in the past. Since the RFIC is based on the direct conversion architecture, it has to deal with the common problems of DC-offset and flicker noise. Eliminating

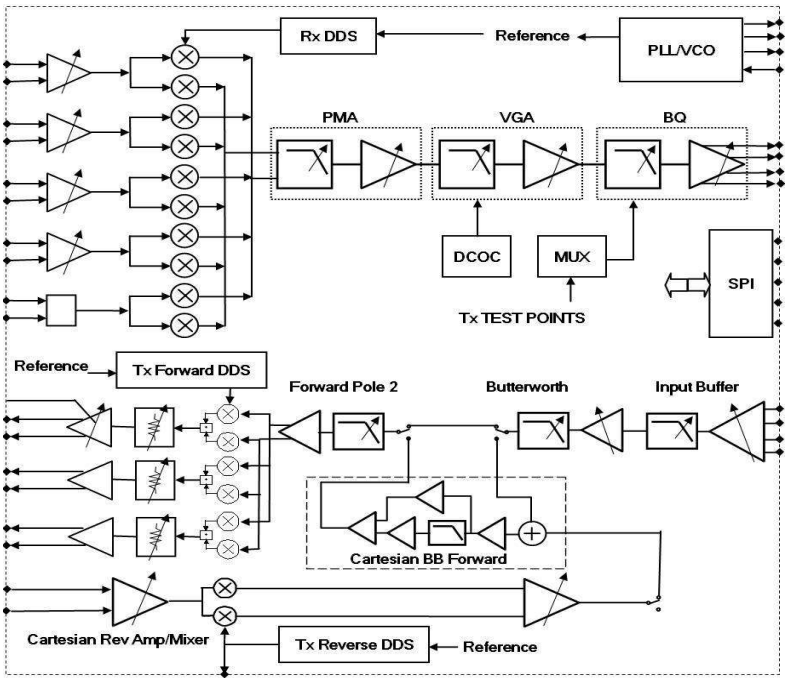


Figure 2.8: RFIC4 Internal Block Diagram [9]

the DC offset at an early stage in the receiver chain is critical to preventing the saturation of the following blocks. A DC offset correction circuit (DCOC) is used for this purpose. It is based on a control loop that uses a successive approximation algorithm to automatically eliminate the DC offset. Also, a novel chopper stabilization technique is used to help mitigate the undesirable effects of flicker noise and second order modulation distortion.

The RFIC4 chip has been used in [16] to build a wideband RF frontend daughterboard for the USRP. Currently the newer version of the RFIC chip exists, the RFIC5. It has all the same capabilities as its predecessor in addition to the integration of the ADC and DAC components on the same chip. This addition helps to further reduce the size of the RF frontend. The RFIC5 will serve as a central component in the SDR platform described in this thesis.

2.4.2 Desirable Characteristics in an SDR platform

In the light of the experience gained from the study of previous SDR platforms, it is my attempt in this thesis to improve upon previous SDR designs by building an SDR platform with the following feature set.

1. **PC-based platform.**The device technology market trend is driven by the force of convergence and the desire to have multiple services and applications running within a single device. The PC model seems to be the convergence point where all applications will ultimately meet. A witness to this is how modern emerging smart phones are starting to look more and more like computers to users through their familiar GUI(Graphical user Interface)s and navigation controls, and to developers by providing APIs for accessing system software and hardware resources. SDRs in a PC-based platform can provide the following benefits:

   (a) A flexible familiar user interface using standard i/o devices(keyboard, mouse/touchpad etc), unlike the case in embedded platforms(switches, LEDs, pots/knobs).

   (b) A powerful API provided by the operating system, including the ability to link applications together which is known as inter-process communication. This way any standard application can be linked to SDR software to extend its functionality. For example, an SDR-based wireless ad-hoc network can be established to enable a distributed application that can run multiple nodes, in order to increase the processing power.

   (c) The SDR software can be installed as a system software component, similar to the X graphical subsystem in Linux or the TCP/IP stack, acting as server which handles requests from multiple applications requiring communication services.

2. **High performance, programmable co-processing component.** An FPGA is the best candidate for the co-processing element. The FPGA will host the high-performance streaming signal processing blocks, while the GPP is free to handle the higher-level communication functions as well as the application layer. This methodology is used in graphics pro-

cessing applications and is highly successful, the co-processing element is known as the GPU(Graphics Processing Unit).

3. **A high speed interconnect linking the host and the co-processor.** The processing pipeline will span both the co-processor and the host processor, therefore, a high speed link is required that can handle high waveform sample rates which are typically multiples of the bit rate. Low-latency is also required as certain communication standards demand response to events within tens of microseconds, ex: carrier sensing in 802.11. The PCIe bus is an ideal candidate since it can support speeds up to 2Gbps and guarantees latency in the range of microseconds [18].

4. **A universal RF frontend unit.** In order to maintain a compact form factor, wide frequency coverage should be provided without having to resort to a multiple board solution. This will also avoid the need to interrupt the operation of the SDR for replacing RF boards when switching between frequency bands.

5. **A compact overall setup suitable for mobility.** The host PC, co-processing FPGA and the RF frontend must altogether form a portable setup. For the host PC, a laptop form factor will be used since its the only mobile option. The co-processor and RF frontend will be based on a small size plug-in card that can connect to the host through an external card port, helping avoiding the use of lengthy cables. The plug-in card power requirements will be supplied by the host avoiding any need for an external supply.

In the chapters to come, the proposed SDR platform will be described in detail as well as the software API for controlling its various subsystems.

Chapter 3

PICO-RF SDR Platform

This chapter introduces the PICO-RF SDR platform. The PICO-RF is a compact, high-performance, universal, SDR platform. It is based on the hybrid architecture described in chapter 2. The system consists of a laptop host and a PCIe plug-in module. The plug-in module is made up of 2 boards. The first board is a PICO [1] Virtex-5 FPGA board which connects to the second board hosting the RFIC5 and related components.

3.1 System Overview

The system consists of three main components: host PC, FPGA and the RFIC5 as shown in Figure 3.1. There is provision for an optional fourth component which is a flexible RF filter that serves to improve selectivity.

PC Host

The PC host will be typically running high-level software such as GNU radio, OSSIE(Open Source SCA Implementation::Embedded) or any custom C/C++ application that interfaces with an API to

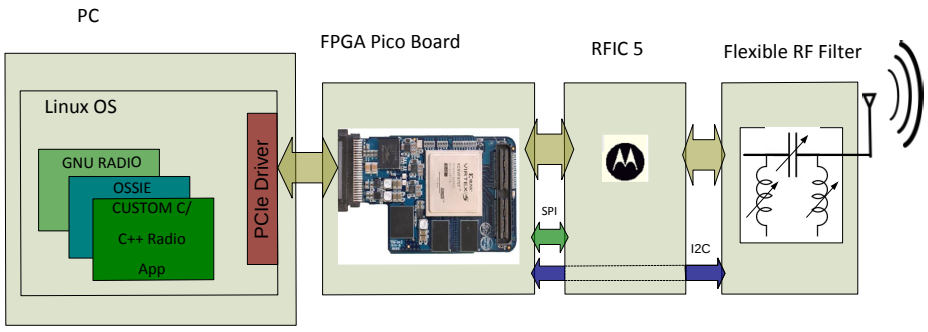


Figure 3.1: Components of the PICO-RF Platform

control the radio. The software can be used to present a user interface to control (knobs) and monitor (sensors) the radio operating parameters. Alternatively, cognitive engine software can be run to automatically control the radio parameters to optimize system performance based on user QoS settings. The software side can also be used to implement some of light-weight blocks of waveform/PHY pipeline, while the heavy-duty operations run on the FPGA. By taking advantage of the extensive dsp block libraries in GNU radio and OSSIE, the radio designer can quickly and easily build working prototypes on the software side and then later move the blocks into hardware if more performance is needed. By using operating system facilities such as linux pipes, standard applications can be linked to the SDR PHY to establish a wireless communication link with applications running on other hosts.

FPGA

The primary function of the FPGA is to serve as a bridge between the PC and the RFIC5. The FPGA contains a PCIe controller to interface with the PC from one side and logic to control the RFIC hardware from the other side. Through the PCIe connection, the PC can send command signals to control the RFIC parameters, as well as transfer high speed data samples using bus-mastered DMA .

The second function of the FPGA is to host most of the signal processing pipeline, especially high

performance blocks that cannot run on the PC. Using partial reconfiguration, DSP blocks can be loaded/unloaded to the FPGA during run-time based on user command or cognitive engine decision. By relying on the high speed PCIe connection, signal processing blocks can be downloaded to the FPGA in very short intervals. This will be helpful in a cognitive radio or dynamic spectrum access scenario where the waveform standard might change on a packet by packet basis. A fast switch will enable a smooth transition preventing loss of samples or the introduction of excessive latency.

RFIC

The RFIC5 hosts all the RF frontend functionality. This includes LNAs for signal amplification, I/Q mixers for frequency translation, and a set of DDS-based frequency synthesizers that can generate frequencies from 100MHz up to 2.4GHz. The RFIC also contains a set of programmable gain amplifiers and a baseband filter with programmable bandwidths from 5KHz to 20MHz.

In addition, the RFIC5 contains a digital subsystem. This includes ADC and DAC components that digitize the signals on the receive path and vice versa on the transmit path. The RFIC uses high-speed serial differential signal lines for communicating the digital sample bit-stream with the FPGA. An SPI(Serial Peripheral Interface) port on the RFIC is used for writing to the RFIC register set through which the different programmable parameters are controlled, ex: bandwidth, center frequency, sampling clock, decimation etc.

Flexible RF Filter

A common problem in SDRs with wide band frontends is the saturation of components in the stage following down-conversion, including Analog Baseband and ADC, due to the high resultant signal amplitude of the superposition of the large number of signals that get through the bandwidth of the front end. This becomes particularly problematic in the presence of strong out-of-band interferers. Reducing the gain to fix this situation is likely to cause the signal of interest to be buried in noise

or otherwise have a very low SNR(Signal-to-Noise Ratio). Using RF filters can help alleviate this situation by attenuating out-of-band interferers and reducing the input noise, but has the potential of limiting the frequency range of operation of the SDR since RF filters tend to operate over a fixed band. By using a programmable RF filter, the interference problem can be addressed while maintaining wide frequency operation.

3.2 System Internals

The core functionality of the system is managed through an interaction between two main subsystems. The first is the software subsystem running on the PC, and the second is the hardware subsystem running on the FPGA.

3.2.1 Software Subsystem

The software subsystem follows a stack structure made up of 3 layers. At the top layer, SDR oriented applications such as GNU radio and OSSIE or custom designed C/C++ applications will be running. Through the API layer, the PicoRF system can be controlled. In graphical environments such as GNU radio and OSSIE, blocks that encapsulate the API functions can be used to represent the PicoRF functionality as part of a bigger radio graph.

The following section will describe in detail the 2 bottom layers of the stack as the application layer is not part of this work.

PicoRF API

The PicoRF API is a set of C functions that present an abstract interface for the programmer to access the various features of the system. The API is divided into three subsets as depicted in Figure 3.2. Each subset consists of a group of related functions that are organized in a certain

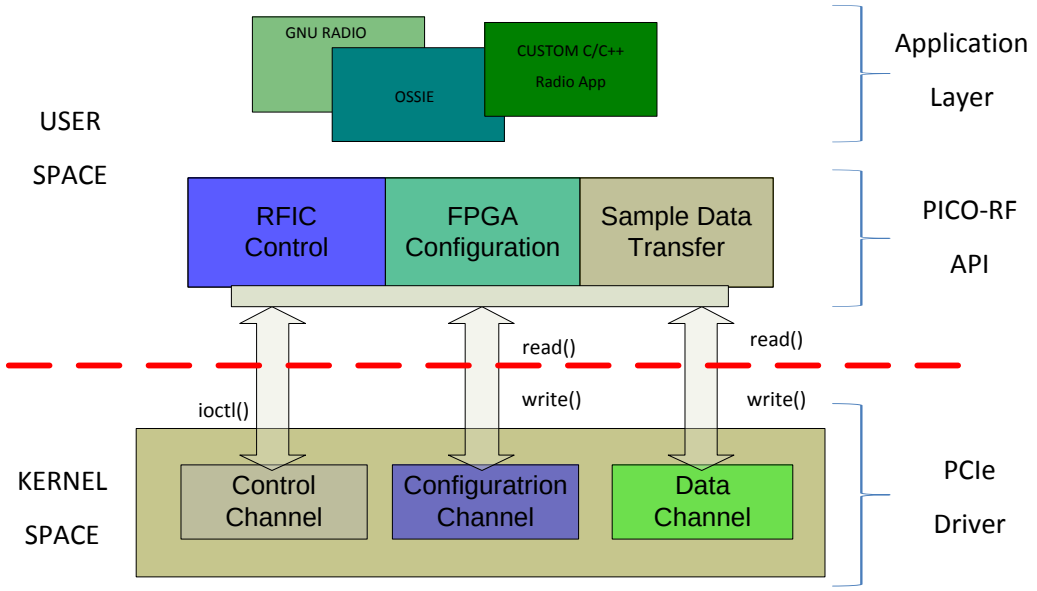


Figure 3.2: Software Subsystem Architecture

structure. The following is a description of the function and structure of each API subset.

RFIC Control

At the hardware level, the RFIC functional blocks are controlled through an internal register set which can be accessed through an SPI port. Groups of contiguous registers are dedicated to controlling each functional unit as shown in Table 3.1. The RFIC API subset presents a high level interface that isolates the programmer from the numerous low level details, providing instead a well defined set of functions for setting standard radio parameters such as bandwidth, LO frequency etc. The high level commands are translated into the required sequence of SPI transactions to the RFIC register set.

The register set can be divided into two groups. One group is for non-runtime settings, including registers used in the initialization procedure of certain blocks. The second group is for controlling the run-time parameters such as LO frequency, amplifier gains, sampling rate, decimation factor,

Table 3.1: RFIC Internal Blocks and Corresponding Register Ranges

Register Range	Functional Unit	Register Range	Functional Unit
0-24	RX Analog block settings	672-688	QUAD GEN RX
64-84	TX Analog block settings	704-720	QUAD GEN TX FW
85-87	PLL settings	736-752	QUAD GEN TX FB
88	VAG GEN	768-784	QUAD GEN ADC
89	West VAG GEN	800-816	QUAD GEN digIF
90-99	Test Mux	1280-1453	busIf (Bus Interface)
128-205	RX QUIET	1536-1610	digIf (Digital Interface)
256-333	TX FW QUIET	1664-1690	ADC Decimation Filters
384-438	TX FB QUIET	1696-1709	Rx ADC (PWM & JADC)
512-589	ADC QUIET	1792-1836	TX DAC
640-645	Dynamic matching MRCG		
648-653	RCLK MRCG		

baseband bandwidth, ADC bit-width etc.

Registers provide a very detailed and primitive interface to the RFIC blocks. The API serves to abstract away the irrelevant details and present a standard radio interface to the programmer. For example, setting the LO frequency for the RX path involves programming the RFIC QUIET frequency synthesizer with a 32 bit divider value which is stored in five registers. This will require the programmer to know low level details such as the reference frequency used and the bit resolution of the divider (7 bits integer, 26 bits fraction). However, a typical radio interface should only require the desired frequency value as an input from the programmer.

The RFIC API subset is structured in 2 layers as shown in Figure 3.3. The lower layer consists of the SPI bus access functions, which serves as the basis for the higher level functions. This layer communicates to the PCIe driver by passing the address and value of the target register through the standard ioctl() driver interface function. In response, the driver initiates a PIO(Programmed Input Output) transaction though the PCIe bus to the SPI control hardware on the FPGA, which in turn generates the SPI bus signals to the RFIC.

The upper layer is divided into two groups of functions. A group of initialization functions used to initialize the state of the RFIC after booting. The other group is the set of functions that control the operational parameters of the RFIC, and are used frequently during run-time. In chapter 5, the upper layer functions will be described in detail.

Low-level Functions

Three base functions constitute this layer:

1. SPI_read : char SPI_read(int reg_addr)

   This function takes a register address as an argument and reads back the value contained in that register.

2. SPI_write : int SPI_write(int reg_addr, char reg_value)

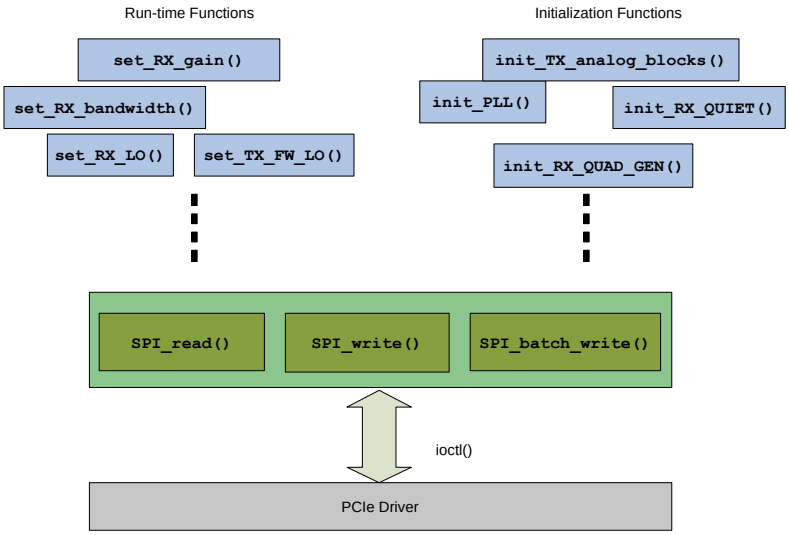


Figure 3.3: RFIC API Organization

The function takes a register address and the value that is to be written into that register. A read-after- write is performed to check if the value been written correctly to the addressed register. The function returns zero if the write was successful and a negative value otherwise. The situations in which the function fails are when attempting to write to read only registers, and when writing illegal values or values corresponding to hardware settings that do not exist, for example a 2 bit mux that has only 3 valid input selections cannot take the 2 bit value 11.

3. SPI_batch_write() : int SPI_batch_write(int base_addr, int cnt, char* reg_value)

   This function takes 3 arguments: base address, register count and a pointer of type char. It is used to write values to a contiguous set of registers in one SPI transaction. The base_addr is the address of the first register, cnt is the number of registers and the reg_value pointer points to an array containing register values. This takes advantage of the RFIC SPI controller's ability to handle burst transactions. In a burst transaction, only one base address is used followed by a sequence of 8-bit register values. Using an internal counter that increments automatically, the RFIC tracks the destination register for each value.

Certain blocks in the RFIC are triggered to act only after all registers in a certain set are updated, even though some of the register values might have not changed. Such is the case with the QUIET frequency and phase control register sets. All five registers in the set must be updated in order for the new output to appear. Using the batch_write() function helps reduce the latency between the command and the action, as opposed to using the single payload SPI_write function multiple times as shown in Figure 3.4. This latency affects the maximum rate at which certain block outputs can change, which is important in applications such as frequency hopping and frequency or phase shift keying in the case of the QUIET block.

FPGA Configuration

This API subset contains functions related to all aspects of the FPGA configuration.

1. PROM configuration : int prom_config(char* full_bit_file)

   This function is used to download an FPGA bit-stream into the boot PROM. The bit-stream file name is provided as an input argument and a value indicating success or failure is returned. Using this function saves the programmer the need to switch to a different software environment to set the initial FPGA configuration, or to use external hardware such as a JTAG(Joint Test Action Group) cable. Permanent information that needs to be persistent across power cycles can be stored into the PROM using this function. This might include system configuration information such as RFIC calibration settings, etc.

2. Loading DSP blocks : int pr_config(int component_id, int slot_num)

   On the FPGA, a resource region that takes the form of a grid of interconnected slots is reserved for dynamic partial reconfiguration, which is called the PR(Partial Reconfiguration) Grid. Each slot will serve as a host for dsp blocks that will be loaded during run-time. Through the pr_config function, partial bitstreams can be downloaded from the PC host into a target slot in the PR-Grid. The function takes two input arguments, component identifier and slot identifier, and returns a value indicating success or failure. At initialization,

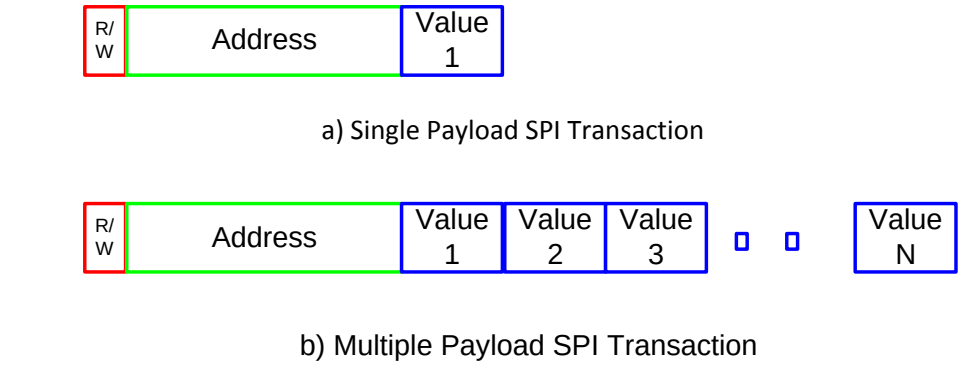


Figure 3.4: Single Payload v.s. Burst SPI Transactions

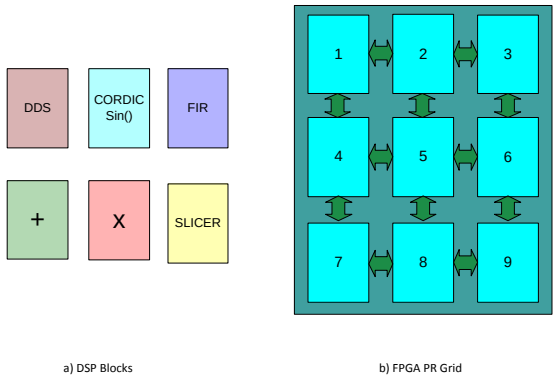


Figure 3.5: PR Grid and DSP Components

PR files of components that are expected to be needed in the design are pre-loaded into a DMA(Direct Memory Access) buffer in the PC memory. The instant a component is needed in the design, the pr_config function is called and a burst DMA transfer is initiated to transfer the component's partial bitstream. The pre-loading helps speed up component download and reduce switch time between waveforms.

Sample Data Transfer

For transferring radio samples between the PC host and the FPGA, two modes of operation can be used:

1. Block Mode: In this mode, the programmer can send or receive fixed size blocks of samples to/from the FPGA. Using the interface functions the programmer specifies the number of samples in a block, sample bit-width and a pointer to the receive/send buffer on the PC side. Block functions can be called repetitively in a loop to transfer data between the PC host and RFIC in a synchronized fashion. However, the overhead involved will reduce the maximum possible data transfer rate.

   (a) PC-to-FPGA : void block_send(char* buffer, int count, int bit_width).

(b) FPGA-to-PC : void block_receive(char* buffer, int count, int bit_width).

2. Streaming Mode: This mode offers the ability to receive or transmit a continuous stream of samples between the RFIC and the PC host with the lowest overhead. Once the function is called, data is streamed into/out of the PC DMA buffers as fast as possible depending on the sampling rate of the RFIC ADCs/DACs. No flow control mechanism exists in this mode. If the receive path on the PC side involves lengthy processing operations, the receive buffers are very likely to be overflowed. Similarly for the TX path, if the PC can't supply data fast enough the buffers will be underflowed. However, overflow and underflow detectors exist that report the events to the control software to take the appropriate action.

   (a) PC-to-FPGA : void tx_stream_start(), void tx_stream_stop().

   (b) FPGA-to-PC : void rx_stream_start(), void rx_stream_stop().

PCIe Driver

In order for PC applications to communicate with external hardware, a special purpose piece of software is required, which is known as a device driver. It serves as a gateway between user space and kernel space. In kernel space, the programmer has access to the low-level system facilities such as DMA buffers and memory-mapped registers. Memory-mapped registers have alias locations in the PC memory space such that when the user operates on this memory range, operations are actually carried out on the hardware registers.

Although generic PCIe drivers exist such as the Jungo WinDriver [2], we have opted to build our own custom driver. Generic drivers do not make any assumptions about the underlying hardware, and thus are not optimized to a specific device. Given the critical role the driver plays in the data transfer, optimizing the driver to the hardware will potentially increase the data transfer rates. For example, the PicoRF registers are 32 bits in size and the driver is designed to receive a 32 bit input from user space. This make it possible to write or read a register using a single instruction. On the other hand, a generic driver might provide a byte interface which would require 4 write/read op-

erations if used with the PicoRF. Also, possessing the driver source code provides the opportunity for future modifications, to meet hardware changes or for bug fixes.

There are two main modes used by device drivers for transferring data between PC and hardware peripherals, which are PIO and DMA. For this platform, both modes are implemented as each mode is suitable for serving a different set of high level tasks.

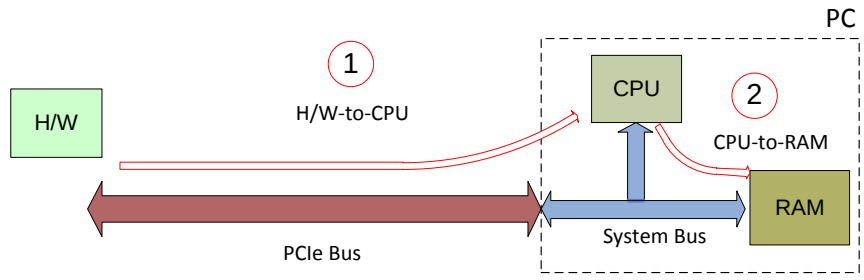
1. PIO data transfer: In this mode, data is transferred between the hardware and PC memory under the explicit control of the CPU. In the context of PCIe, PIO corresponds to single payload TLPs(Transaction Layer Packets) initiated by the CPU for either reading or writing data to the hardware. PIO is typically used by the CPU to write control commands or read status signals from the hardware, especially when synchronization is necessary.

2. Bus-mastered DMA: Used for transferring large amounts of data between the PC and FPGA. Best data rate performance is attained in this mode of operation mainly due to two reasons, bus mastering and low overhead. Bus mastering refers to the fact that the transaction is only initiated by the CPU, while the remaining part of it is managed by the hardware side. This helps free the CPU for other tasks as well decrease transfer time as shown in Figure 3.6. Low-overhead refers to the fact that in DMA transfers TLP(Transaction Layer Packet)s have a large payload size as compared to the header which increases the effective data rate. In such TLPs, only the start address is included since all data go into consecutive locations in memory.

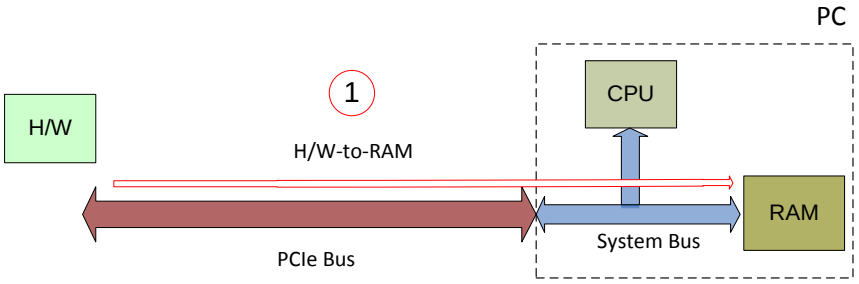
In our platform, a DMA operation involves the following steps:

1. DMA initiation by PC : In this step, the PC uses PIO transactions to set the DMA parameter registers in the hardware and then requests the DMA operation to start. In the PicoRF there are 3 DMA control parameters:

   (a) TLP Size: Sets the size of the TLP packet. The maximum possible value is determined by both the hardware and host PCIe controller. In our current setting the max value is



a) Programmed I/O : 2 operations to transfer data.



b) DMA : 1 operation to transfer data

Figure 3.6: Comparison between number of steps involved in PIO and DMA read done by the PC

128 bytes.

(b) TLP Count: Sets the number of TLP packets that constitute one transaction. This is equal to the total amount of data requested divided by the TLP size.

(c) Target Address: The hardware being the bus master will directly access host main memory to either read or write data. This address provides the target location in main memory at which data is to be stored or retrieved. A special type of address known as bus-address or physical address [25], is provided to the hardware controller. Bus addresses are addresses that are used on the PCIe bus and are translated to a corresponding range of addresses in the host main memory through address translation hardware. This is done to enable DMA to target different ranges of addresses in main memory.

2. TLP stream management by hardware peripheral: In this step, the hardware controller sends/requests a sequence of TLPs to/from the PC. The DMA controller keeps track of the count of TLPs received/sent so far and the main memory address of each TLP request.

3. Transfer completion interrupt: After all TLPs are transferred, hardware sends an interrupt to signal the completion of the data transfer. Status flags are set to indicate to the PC whether it is a write or read completion. The PC then clears the flags in the interrupt service routine. PCIe provides 2 methods for sending interrupt signals to the host CPU [25], legacy and MSI(Message Signalled Interrupts). Legacy interrupts use the classical method where a dedicated signal wire is connected to the CPU. MSI on the other hand uses in-band signalling, where a special purpose TLP is sent over the PCIe link to indicate the occurrence of an interrupt.

The PCIe driver implementation is structured into two independent functionalities, a PIO channel and a DMA channel .

1. DMA Channel: This channel is used to transfer of large amounts of data at the highest rates, and hence operates in DMA mode. The channel is further divided into 2 logical channels,

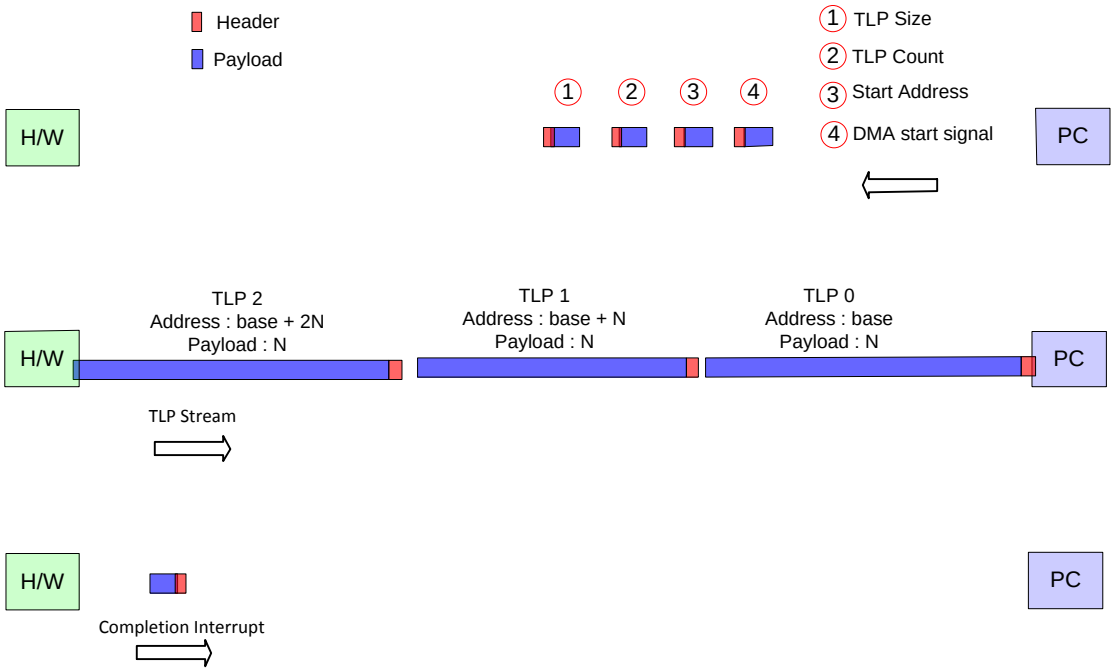


Figure 3.7: Steps involved in a DMA Operation

a Data channel and a Configuration channel. The two channels are multiplexed across the PCIe link, with a higher priority assigned to the configuration channel. To represent each channel, the PCIe driver allocates two instances of the x5pcie_func data structure shown in Figure 3.8, one per channel. Additional instances of this data structure can be used to represent more channels in the design if needed.

Two dev node files with a different minor number are used to access the PCIe driver as indicated in Table 3.2. Based on the minor number, the driver can multiplex user space request between the two channels.

2. PIO Channel: This channel is used to transfer small amounts of data at relatively low rates. Low rate transfers are used mainly for control purposes such as writes to system registers to set control signals, and reads to get status. Through the ioctl() function, user space applications can provide input codes that trigger certain control operations in the hardware. Using a switch case structure, the input parameter is decoded into the corresponding operation as

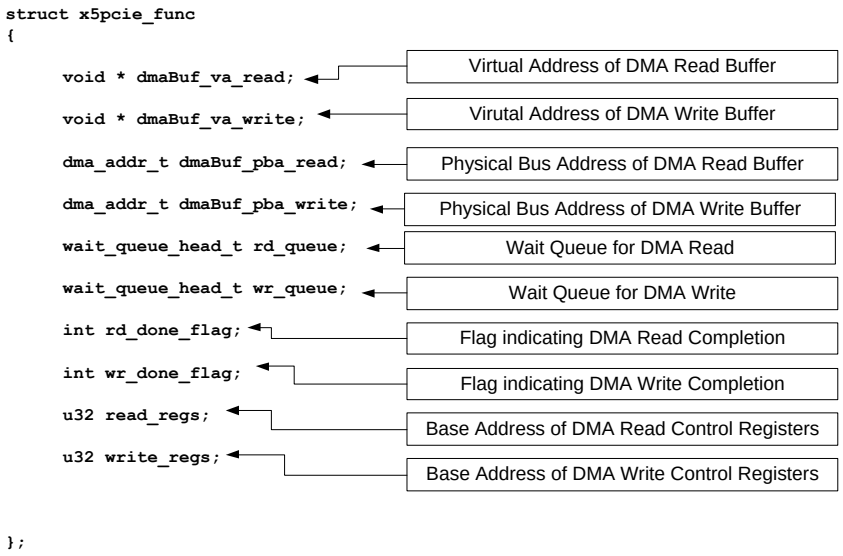


Figure 3.8: x5pcie_func Data Structure

Table 3.2: Devnode Files used to represent the different Logical Channels

Channel	Filename	Minor no	Function
Data	/dev/dmaX5Data	0	Full-duplex channel, used to transfer radio samples between the host PC and RFIC.
Configuration	/dev/dmaX5Config	1	Half-duplex channel, used to download configuration bit-streams to the FPGA.

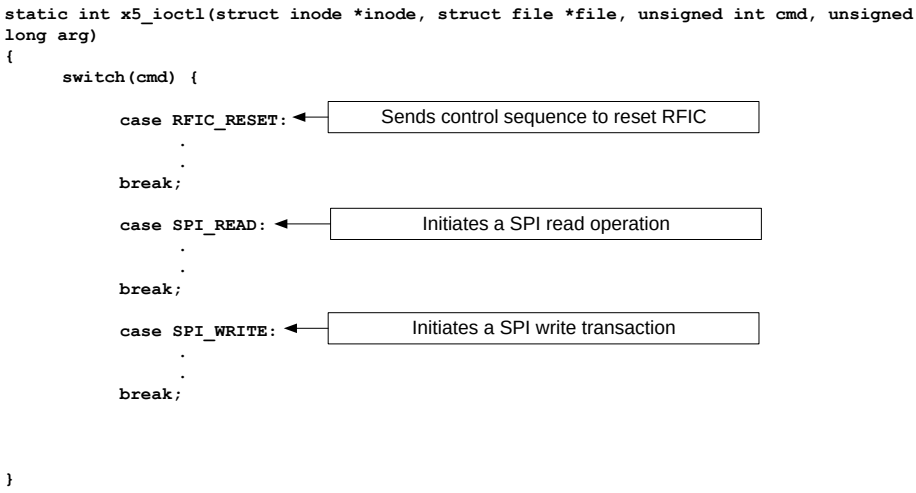


Figure 3.9: ioctl() function switch case structure to decode input codes into different operations

shown in Figure 3.9. All the RFIC sub-API functions access this channel.

3.2.2 Hardware Subsystem

This is the hardware complement that interacts with the software subsystem to implement the different system functions. The hardware units are hosted on the FPGA and are divided into 3 main groups as shown in Figure 3.10.

Xilinx PCIe Core

At the physical level, PCIe uses a high speed differential signalling scheme to transfer data bits at a rate of 2.5Gbps, in both directions. Higher layers of functionality impose a packet structure on the physical bit stream and implement mechanisms that make up a sophisticated communication protocol. The protocol structure is described by the PCIe standard [27] [32]. In the standard, three layers of functionality are defined which are, physical, data link and transaction layers.

A one lane configuration of the Xilinx Endpoint Block Plus v1.14 CORE is used in the design

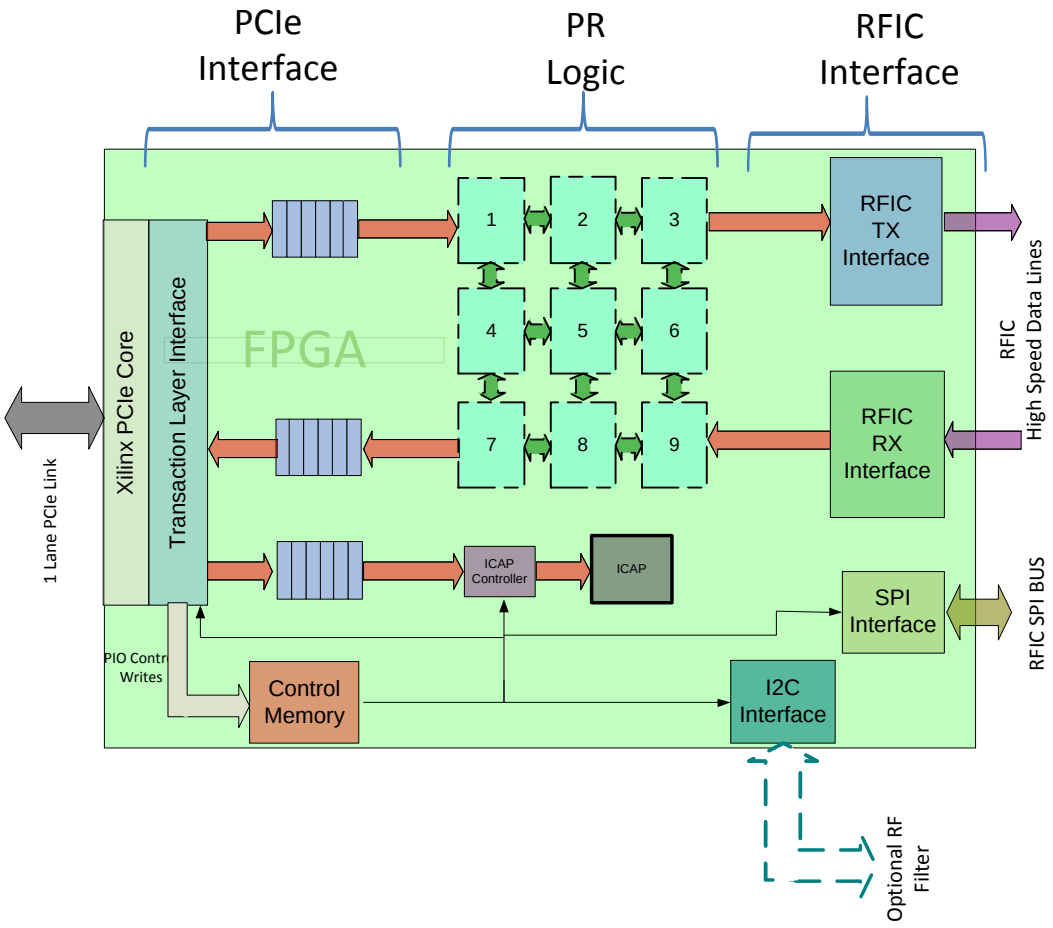


Figure 3.10: Hardware Subsystem Components Implemented on the FPGA

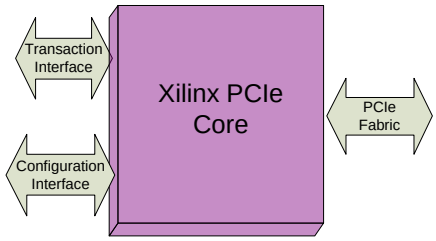


Figure 3.11: Xilinx Endpoint Block Plus v1.14 Core Interface Ports

to implement the PCIe protocol functionality. The block diagram of the core is shown in Figure 3.11. The CORE exposes only transaction layer packets that are relevant to the user, and responds automatically to other messages. For example, the core automatically transmits completions to a remote device in response to configuration space requests, and sends error-message responses to inbound requests malformed or unrecognized by the core. On the other hand, the user application is responsible for constructing completions in response to requests such as memory read requests.

User logic interacts with the PCIe core through two interfaces:

1. Transaction Interface: This interface provides a mechanism for the user to generate and consume TLPs, which form the basic data transfer units.

2. Configuration Interface: This interface provides an access port to configuration memory space [27] [32]. The configuration memory contains status signals and device configuration parameters set by the PC host such as PCIe bus addressing information. It also provides signals that trigger special messages such as interrupts.

Transaction Manager

On the FPGA, there are 4 data sources and sinks that exchange data with the PC through the PCIe link as shown in Figure 3.10. The blocks are, sample write buffer, sample read buffer, configuration write buffer and control memory. The Transaction Manager decodes the incoming TLP messages and takes the necessary action by either reading or writing to the different sink and source blocks.

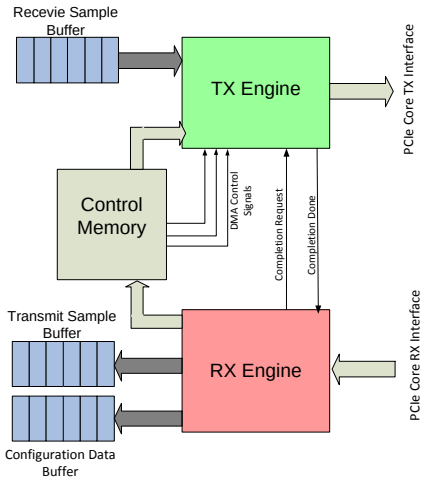


Figure 3.12: Transaction Manager Architecture

It also manages the multiplexing of data across the different logical channels, such as the data and configuration channels. For DMA transactions, the Transaction Manager handles the responsibility of keeping count of data transmitted or received and the corresponding target memory address until the DMA transaction is complete.

Given that the PCIe link is a full duplex connection with dedicated bandwidth for each direction, the Transaction Manager dedicates independent blocks for transmit and receive as shown in Figure 3.12. This architecture is adapted from the reference implementation described in [33]. The TX engine packetizes data coming from either the receive sample data buffer or control memory into outgoing TLPs. While the RX engine handles incoming TLPs and directs the payload data to the configuration buffer, transmit sample data buffer or control memory.

The most basic form of data exchange between the two ends of a PCIe link involves a sequence of two packets, a request TLP sent by one side followed by an optional completion TLP response from the other side. In this fashion, data transfer between the two ends can take place in four ways as shown in Figure 3.13.

The Data transfers handled by the TX and RX engines can be classified into two types:

1. Control Memory Reads/Writes: This type of transfer is initiated by the PC side to either

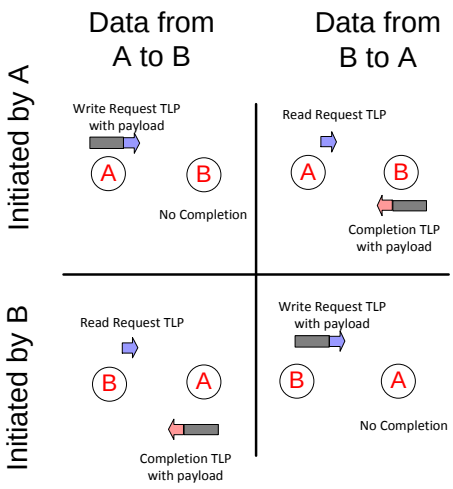


Figure 3.13: Possible Command Sequences for Data Exchange between two endpoints on a PCIe bus

read or write a 32 bit word into the memory registers. Register writes are done through write request TLPs which contain the data in the packet payload. The RX engine handles the write request and directs the data to the destination register in the control memory. Similarly, read request TLPs are used for register reads. When the RX engine receives a read request TLP it signals the TX engine to respond with a completion TLP whose payload is filled with the contents of the addressed register. The sequence of operations involved in memory reads and writes is shown in Figure 3.14.

2. Buffer Reads/Writes : Unlike the control memory, the buffers connect to high speed streaming data source and sink blocks. Two buffers connect to the signal processing pipeline prior to the RFIC interface. The transmit sample buffer serves as the source point for the processing pipeline in the radio transmit path, while the receive sample buffer serves as the sink point in the receive path. The third buffer connects to the ICAP(Internal Configuration Access Port) controller and supplies the FPGA partial bit stream which is directed to the ICAP port.

   Bus-mastered DMA provides the highest data transfer rates, and thus is used for both the radio sample and configuration data streams. In contrast to control memory transfers, read

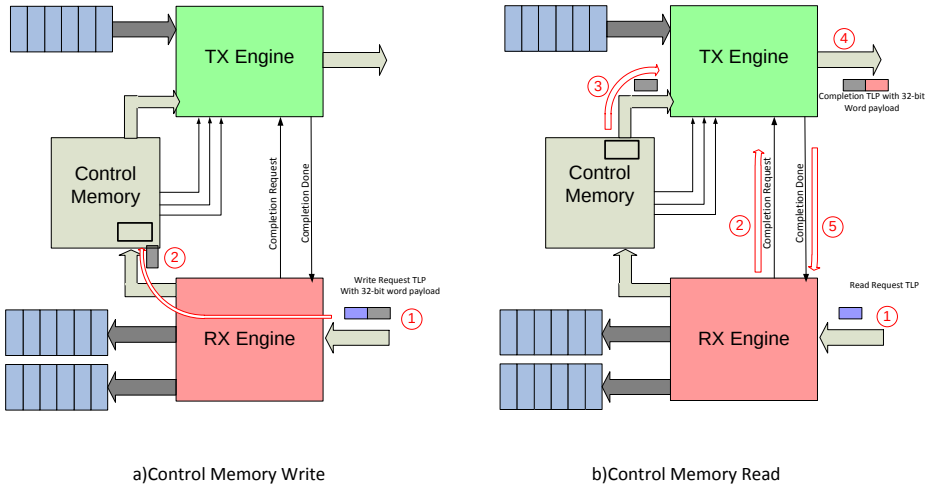


Figure 3.14: Control Memory Access

and write request TLPs are initiated by the FPGA side. Nevertheless, the PC host must first set the DMA control parameters in the control memory before the DMA transfer can start. Flow graphs in 3.15 describe the sequence of actions by the TX and RX engines that make up a DMA operation. TX engine actions are represented by green blocks and RX engine actions by red blocks.

TX Engine

The TX engine starts by looping in an initial reset state waiting for an input signal. Based on one of four external trigger signals the TX engine will take action. The four paths branching from the reset state correspond to the four possible actions as shown in Figure 3.16. The external trigger signals are represented by the button blocks in the state machine. Table 3.3 lists the trigger signals and their corresponding actions, ordered according to priority.

A traversal of the state machine from start to end corresponds to sending a single TLP, whether it is a completion TLP, a read request TLP or a write request. For DMA operations, the input trigger signals (cf_start, rd_start, wr_start) remain asserted, causing the state machine to be tra-

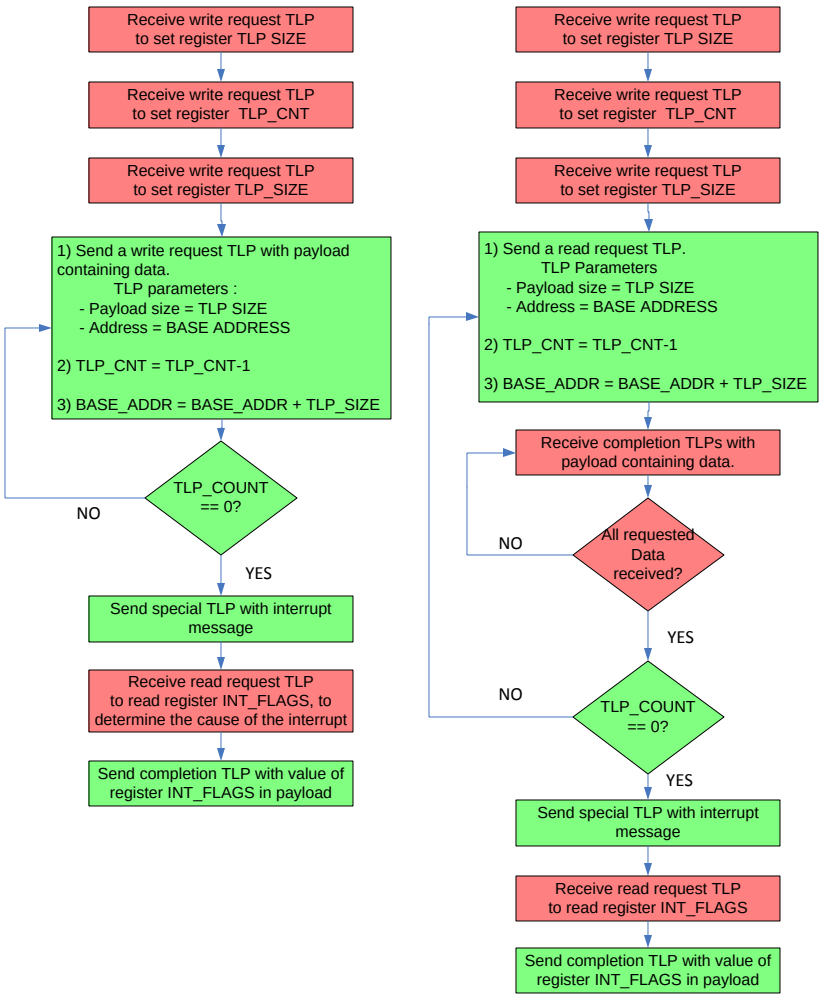


Figure 3.15: RX and TX Engine actions involved in a DMA operation: a)FPGA to PC. b)PC to FPGA

Table 3.3: TX Engine Trigger Signals and Corresponding Actions

Trigger Signal	Priority	Signal Source	Action
compl_req	0(Highest)	RX Engine	TX engine sends a completion TLP.
cf_start	1	Control Memory	TX sends a read request TLP for reading configuration data. from PC side.
rd_start	2	Control Memory	TX sends a read request TLP for reading sample data from PC side.
wr_start	3	Control Memory	TX sends a write request TLP for writing data to the PC side.

versed multiple times, until all TLPs have been sent. The internal variables (cf_tlp_cnt, rd_tlp_cnt, wr_tlp_cnt) keep count of the TLPs sent. When the counter reaches its final value (CF_TLP_CNT, RD_TLP_CNT, WR_TLP_CNT) a signal is set to indicate completion (cf_done, rd_done, wr_done).

RX Engine

The RX engine loops in an initial reset state until a TLP arrives from the PCIe core. The TLP header is parsed to determine the type of the packet. The received TLPs can be one of three types :

1. Memory read request from the PC.
2. Memory write request from the PC.
3. Completion TLP for DMA read operations initiated by the PC.

PR Logic

The classical Xilinx slot-based method has been chosen for allocating dynamic reconfiguration resource regions. Known as a PR slot, it is a rectangular region of resources, fixed in size and

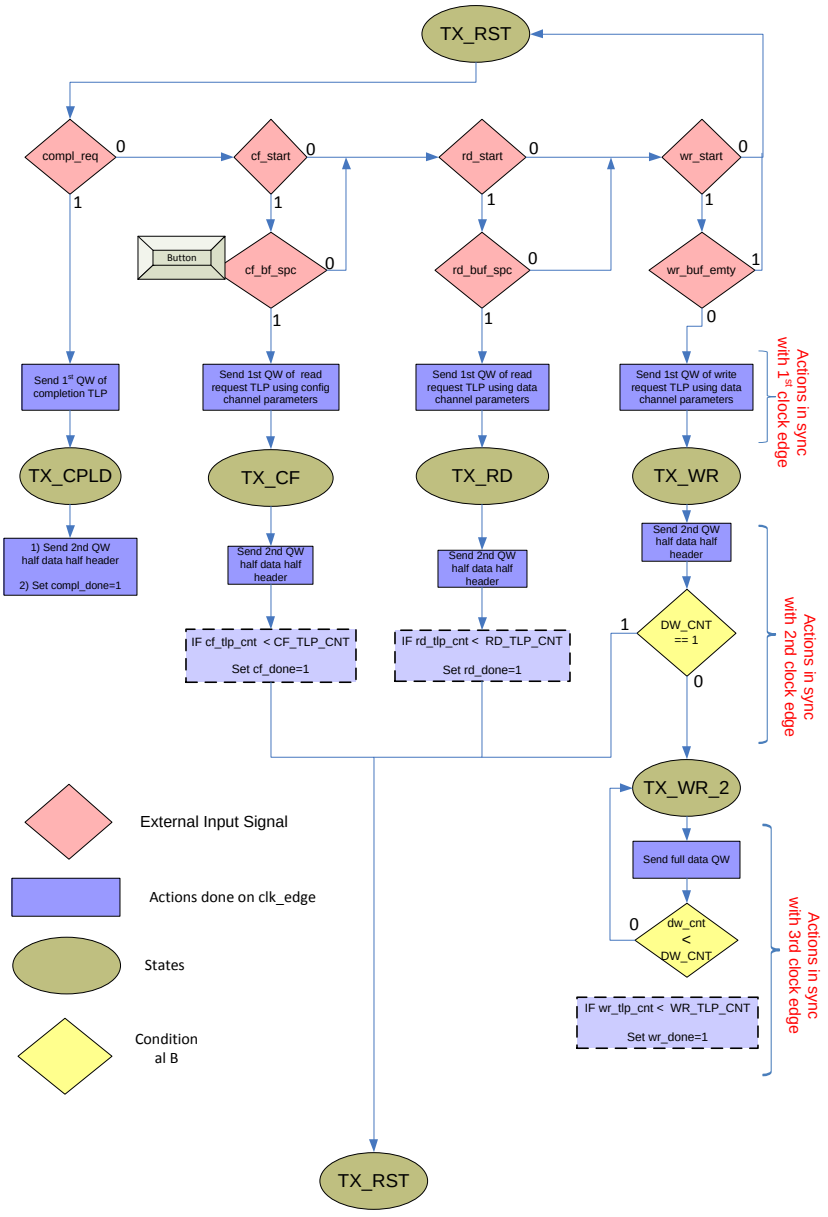


Figure 3.16: TX Engine State Machine.

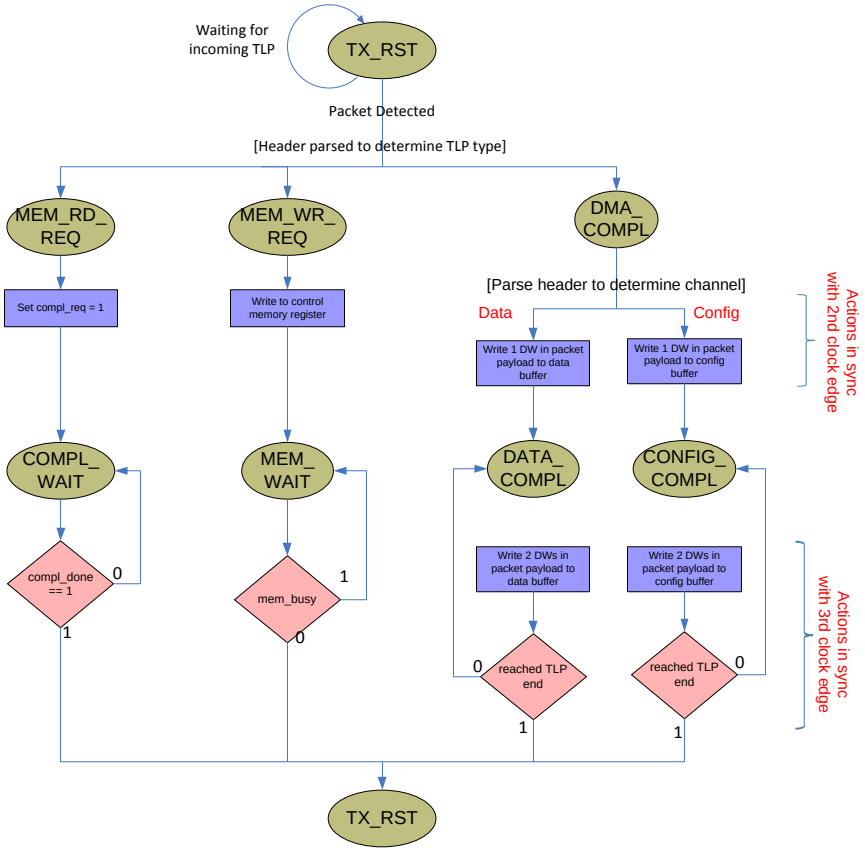


Figure 3.17: RX Engine State Machine.

position, whose internal configuration can change during run-time. On the other hand, the slot's external ports and connections to the remaining parts of the system remain fixed. DSP components are precompiled against the target slots and are stored as PR bitstreams ready for download in the runtime stage.

The slot-based model imposes two main constraints on components that target PR slots. First, the component resource requirements must be fully confined within a single slot, i.e. the PR bit file resources cannot be distributed across more than one slot. Second, a slot cannot host more than one component. As a result, two main problems arise. For small components, only a small percentage of the slot resources are utilized. This causes the remaining resources in the slot to be wasted since they cannot be used to host additional components. On the other hand, large components might require resources spanning more than one slot, making the use such components infeasible.

Various custom partial reconfiguration flows have attempted to address the shortcomings of the slot-based model. One example is AgileRadio [5] [28], the approach used in it is to assign the whole of the dynamic reconfiguration area to a single large slot known as a Sandbox. Custom routing and placement tools are then used to place and connect components inside the slot, since this capability is not supported by the vendor tools. This way, the resource area is efficiently used to both accommodate large components as well as multiple small-sized components.

For the radio run-time reconfiguration problem, the data flow through the path where a component is being reconfigured will be stopped and incoming samples will have to be buffered until the reconfiguration completes. Due to the sample buffering, latency will be introduced in the sample flow, and the longer the reconfiguration period the greater the delay and the larger the chances of buffer overflow and sample loss. Although AgileRadio is more efficient in terms of resource utilization as compared to the slot-based model, its required reconfiguration time will be significantly greater due to the extra steps of routing and placement that have to be executed for every reconfiguration action. Thus, the slot-based model is more suited for achieving shorter reconfiguration times.

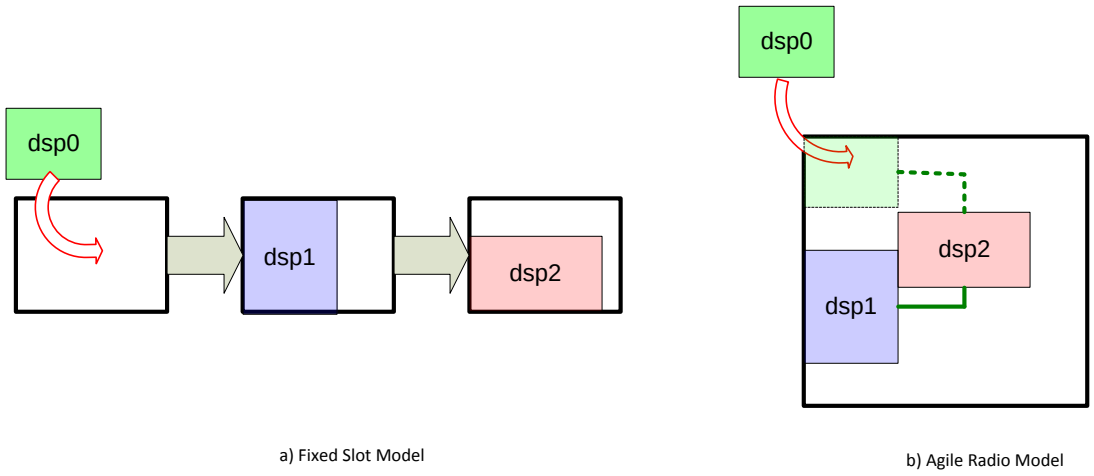


Figure 3.18: a) Fixed Slot Model: Inserting a component involves PR bit-file download. b) Sand Box Model: Inserting a component involves PR bit-file download + place and route.

PR Grid

A rectangular grid of interconnected slots is used to host the DSP components as shown in Figure 3.19. The interconnections are bidirectional to allow for maximum flexibility and efficient utilization. All the slots are identical in size and resource arrangement, which makes it possible to have a single PR bit-file per component. Using a trivial manipulation of the PR bit file, the resource position information can be changed so the component can target different slots.

There are 2 grid variations that can be used. Grid A consists of 8 slots arranged in 4 rows and 2 columns. Grid B consists of 4 slots arranged in 2 rows and 2 cols, with a slot size twice that of Grid A. Grid A can be used in designs with fine component granularity, while grid B is more suited for designs with coarse component granularity. The grid type used in the design is determined based on the FPGA initialization bit-stream that will be loaded at system power-up.

The grid slots are shared by two signal processing pipelines, the receive pipeline which is part of the radio receive path, and the transmit pipeline, which is part of the radio transmit path. The rectangular structure of the grid provides the flexibility for dividing the slots between the transmit and receive pipelines based on the number of component required in each, Figure 3.20 shows the

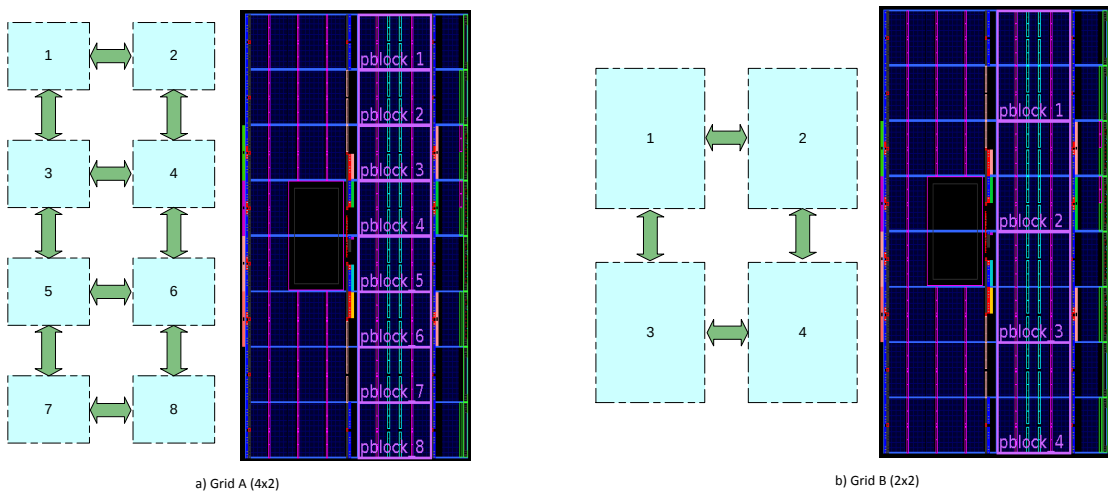


Figure 3.19: a) Grid A (4x2). b)Grid B (2x2)

possible scenarios.

PR Slot Wrapper

As can be seen from Figure 3.20, each PR slot can be connected to a neighboring slot in any of the four directions. The slot consists of a wrapper of static logic surrounding the dynamic PR region where the DSP block is inserted. The wrapper logic is controlled by the PR controller. The multiplexers select the source port and the destination port out of the four directions. The AND gates are used to stop the flow of data into and out of the slot when a new component is being inserted. The PR controller generates the required sequence of control signals to the wrapper logic that facilitates a smooth and data lossless insertion of the new dsp component.

PR Components

PR components have a common internal structure and external interface as shown in Figure 3.22. The different components are expected to vary in the number of clock cycles required to output a sample, due to the variation in the complexity of the dsp operations, especially if small area is

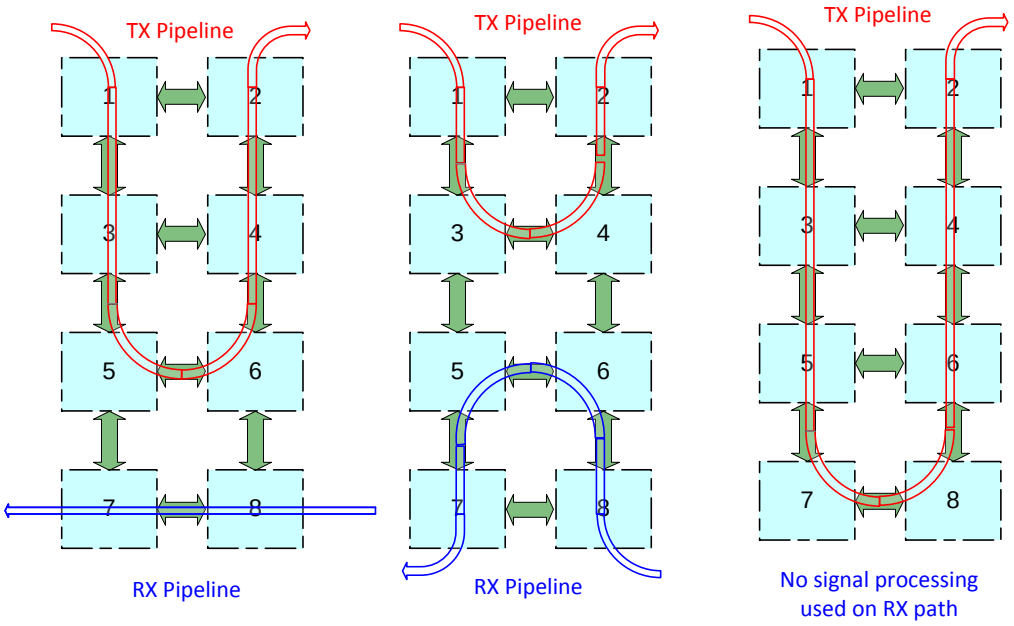


Figure 3.20: Scenarios for dividing PR slots between TX and RX pipelines

required. This causes a difference between the output rate of one block(X), which is the rate of input into the following block, and the rate at which the consecutive block(X+1) can accept input. The input buffer allows the transfer of data to occur between two blocks that are not operating synchronously or are operating at different rates. It also serves as a flow control mechanism, by signalling the source component to stop sending data when the buffer is full. This effectively throttles the input rate down to match the output or processing rate of the component.

A component(X) interfaces with the one preceding(X-1) it in the pipeline through three interface ports:

1. wr_en: signal from preceding component X-1 to write data into X's input buffer.
2. d_in: 32 bit input data port (16 bit I, 16 bit Q).
3. rdy: signal from component X to X-1 indicating readiness to accept data.

Interface to following component(X+1):

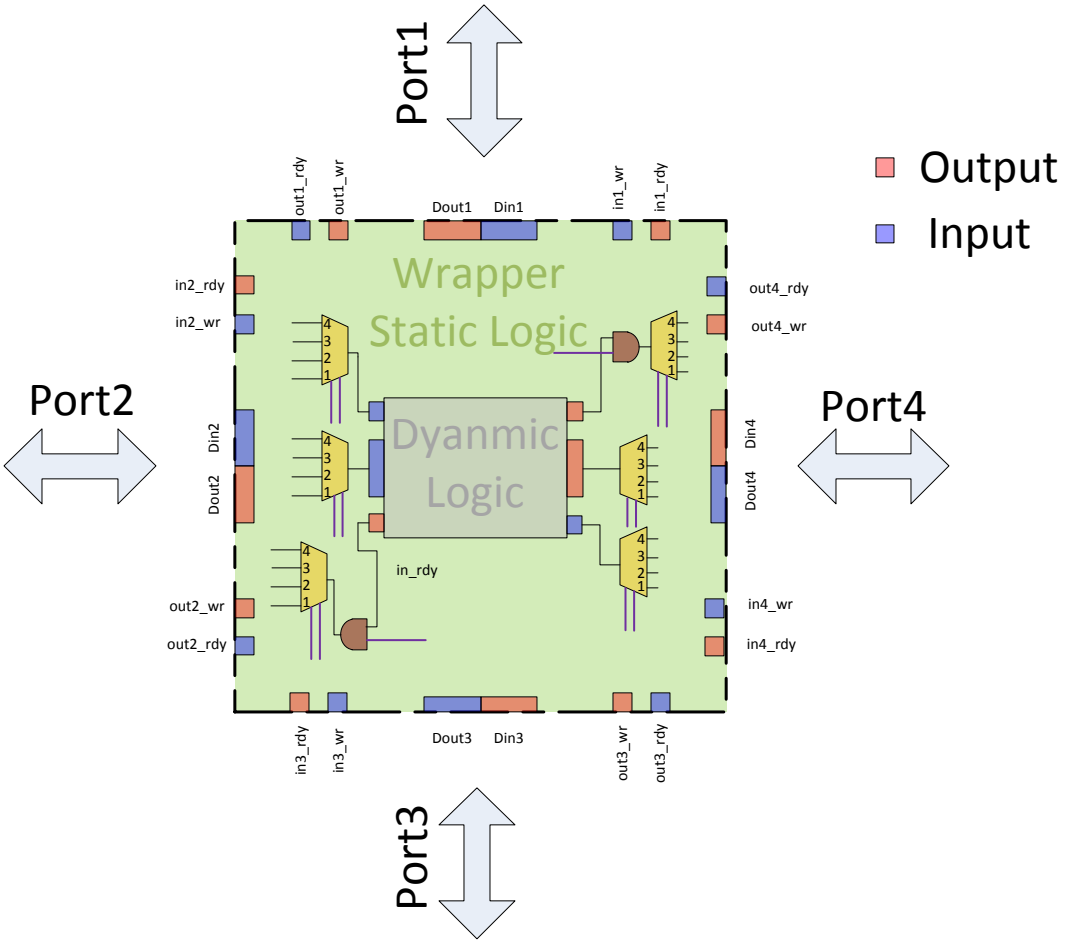


Figure 3.21: Slot wrapper containing static logic to manage dynamic component insertion

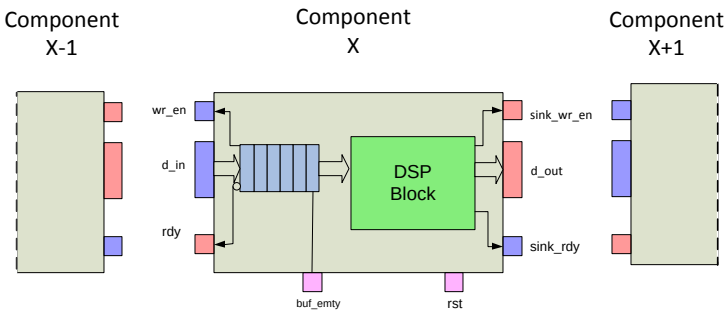


Figure 3.22: PR Component Architecture

1. sink_wr_en: signal to write data into following component X+1.
2. d_out: 32 bit output data port (16 bit I, 16 bit Q).
3. sink_rdy: signal driven by component(X+1) signalling component(X) to keep sending in data.

A third set of ports is controlled by PR controller during a reconfiguration operation:

1. rst : initializes the state of a newly reconfigured component.
2. buf_empty: monitors the status of the component input buffer.

Partial Reconfiguration Sequence

Radio reconfiguration involves modifying components in the signal processing pipeline, amidst runtime, while input data is flowing through. If not handled correctly, there is a chance that a portion of the data samples will be lost or corrupted during the period of reconfiguration. The ICAP controller block shown in Figure 3.10 is responsible for managing the reconfiguration operation. It channels the reconfiguration bit-stream from the pc host into the ICAP reconfiguration port, in addition to applying control signals to the slot being reconfigured in a sequence that guarantees the preservation and integrity of data samples.

The following is a description of the sequence of steps applied by the PR controller to complete the reconfiguration process:

1. PR controller blocks input data flow: The PR controller applies a zero signal at the AND gate pin to stop data flow into the input data port.
2. PR controller waits until input buffer is empty : The PR controller checks the buf_empty signal to make sure that no samples are left in the buffer. Any samples left in the buffer will be lost when the component is reconfigured.

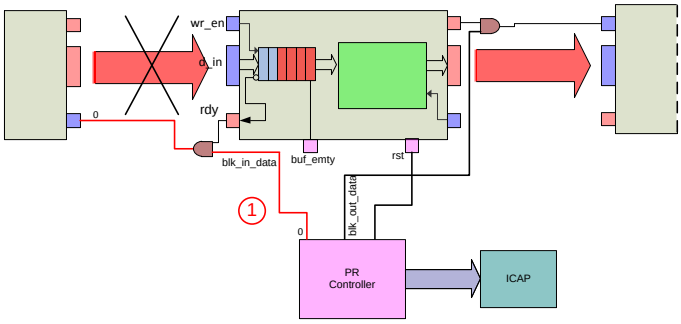


Figure 3.23: PR Sequence Step 1: Block Input Flow

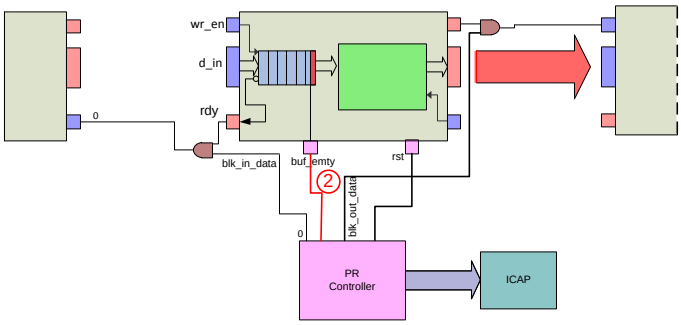


Figure 3.24: PR Sequence Step 2: Wait until Component Internal Buffer is Emptied

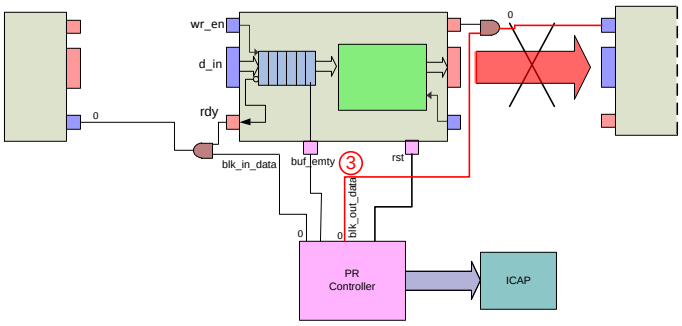


Figure 3.25: PR Sequence Step 3: Block Output Flow

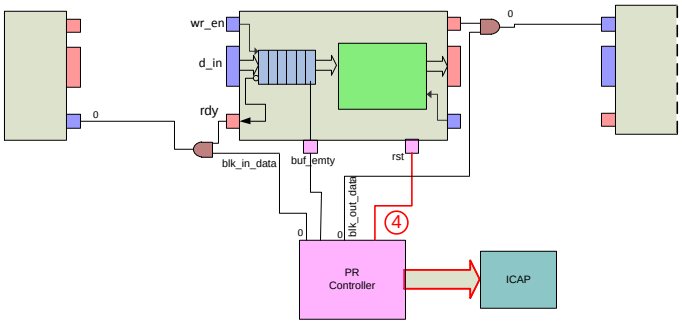


Figure 3.26: PR Sequence Step 4: Stream PR Bit File Data into ICAP Port to Start Actual Reconfiguration

3. PR controller blocks output data flow: The PR controller applies a zero signal at the AND gate pin to prevent random writes into the destination component(X+1) that might occur while the component(X) is being reconfigured.

4. PR controller starts reconfiguration : The PR bit-stream is streamed from the PC into the ICAP port and the rst pin of the component is asserted throughout the reconfiguration period.

5. PR controller checks status of reconfiguration : After all the whole bit-stream has been downloaded, the status of the ICAP port is checked to see if the reconfiguration was successful. If successful the reset signal is de-asserted, if not, an error signal is reported to the host.

Chapter 4

RFIC control API and GNU Radio Interface

The RFIC is a central component in the PICO-Rf platform since it contains all the RF radio functionality. It is based on a highly programmable architecture where almost every component in the transmit and receive chains has controllable parameters. For example in the receive chain, the following component parameters can be controlled: LNA gain, down-mixer LO frequency, base-band filter bandwidth, base-band gain, ADC sample rate and ADC sample word size. Such a highly flexible architecture presents an ideal candidate for a cognitive radio where a wide range of parameters are allowed to vary in order to optimize a given performance metric. However, this high programmability comes at the cost of complexity where the RFIC has over 1800 registers to control the different block parameters. In order to manage this complexity a more abstract and organized interface must encapsulate the native register interface. This is the function of the RFIC API which consists of a set of C-functions organized in groups, each group associated with a particular RFIC block.

Due to the vastness of the RFIC API, this entire chapter has been dedicated to describing the main blocks in the RFIC and the associated API functions. Finally, the GNU radio block pair PICO_RF_RX and PICO_RF_TX will be presented. The blocks encapsulate the PICO-RF functionality in a GNU radio wrapper block that can communicate with the GNU Radio environment.

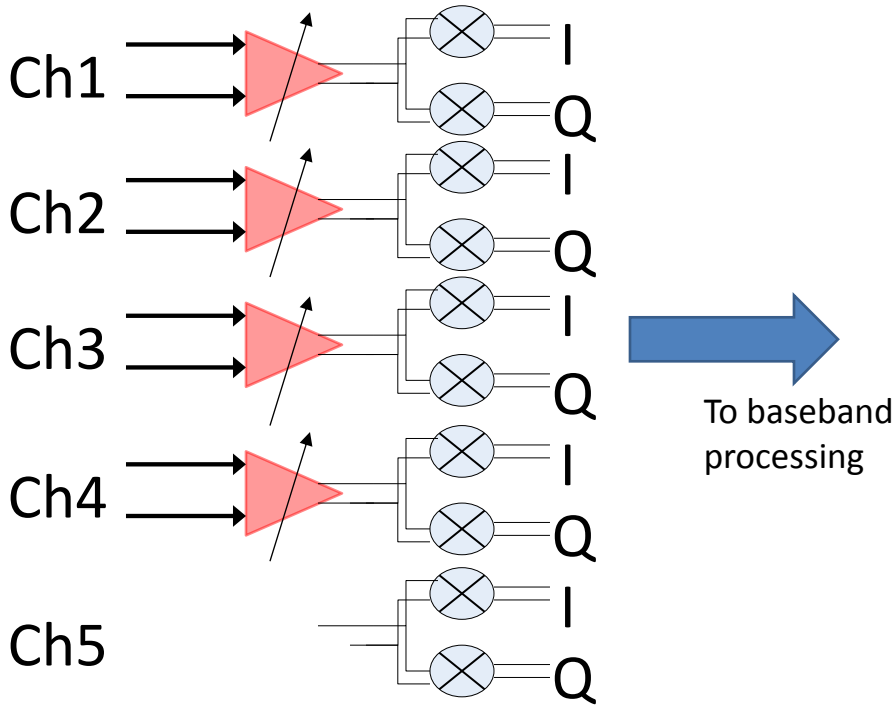


Figure 4.1: RF Processing Block(LNA+Down-conversion Mixers)

4.1 RFIC Blocks and API Functions

The RFIC blocks can be divided into four groups: RX Path, TX Path, Frequency Synthesis and Digital Baseband Link.

4.1.1 RX Path blocks

This consists of all the blocks that apply processing operations on the received signal, starting all the way from the antenna down to the digital baseband samples.

RF Processing (LNA + Down-conversion Mixers)

The RFIC has five receive input ports which are software selectable through the SPI register interface. Inputs 1 through 4 each include an LNA with 50-Ohm differential input impedance, followed

Table 4.1: RF Processing API Functions

Function Signature	Usage	Allowable argument values	Return value
int set_LNA_bias(char)	sets bias current of LNA	LNA_BIAS_2MA, LNA_BIAS_4MA, LNA_BIAS_6MA, LNA_BIAS_8MA	0 (success), -ve (failure)
int select_rx_ch(char)	select rx input port	RX_CH1, RX_CH2, RX_CH3, RX_CH4, RX_CH5,	0 (success), -ve (failure)

by a pair of I/Q down-conversion mixers. Input 5 goes directly to the down-conversion mixers allowing for the use of a very low noise external LNA to improve the noise figure if desired. The bias current of the LNAs is programmable, which can be used to control the third order inter-modulation performance (IP3) of the receiver front end. Multiple physically-spaced antennas can be connected to the input ports to receive independent versions of the signal achieving receiver diversity.

Analog Base-band Processing (Gain and Bandwidth Control)

After down-conversion to DC, the baseband signal goes through a baseband processing block for filtering and amplification. The filter architecture has four poles of filtering with two real poles and one complex pole pair in a Sallen-Key Biquad.

The user has independent control of the pole locations of the PMA, VGA, and Biquad as well as control of the Biquad filter Q. This allows the flexibility to trade off filter shape and attenuation for pass-band amplitude and phase distortion. Filter bandwidth is programmable from 5 KHz to 10 MHz in 6.25% steps or less.

The API provides two methods for the user to control the filter frequency response. The first method is to manually set the individual component values to get the desired filter shape. The

second method is to use a function that automatically sets the component values to produce a butterworth filter shape with a user selectable bandwidth out of a number of predefined selections. The second method is more convenient for varying bandwidth in runtime.

Analog to Digital Converters

The RFIC contains two analog to digital converter (ADC) sets. Each set consists of a pair of I and Q ADC cores. The first set is called PWMADC and is based on the PWM(Pulse Width Modulation) technique which is an advanced form of a sigma-delta modulator. The second data converter set is called the JADC (Jack ADC) which is a more conventional sigma-delta modulator. As shown in Figure 4.2, one of the ADCs is selected for operation using a multiplexer, making only one ADC useable at a time.

Three possible sources can be chosen for the ADC clock as shown in Figure 4.2:

1. An external clock source connected through the Ext Pin line.
2. The main PLL of the chip, which is set for this platform to run at 2GHz.
3. The third source is the AD/DA QIUET frequency synthesizer. This provides maximum flexibility as it allows the precise control of the sampling frequency (Fs) for matching the sample rate to an integer multiple of the symbol rate of a particular standard.

Programmable Decimator

The programmable decimator structure consists of a cascade of three [1-8] decimators followed by a FIR filter as shown in Figure 4.3 . The FIR filter is used mainly to compensate for the droop introduced in the pass-band by the comb filters. It is configurable so that it can operate as a half-band or non-half-band FIR filter. The number of coefficients can be programmed from two up to

Table 4.2: Analog Base-band API Functions(continued)

Function Signature	Usage	Allowable argument values	Return value
int set_OBUF_gain(char gain)	sets gain of output buffer amplifier	OBUF_GAIN_18, OBUF_GAIN_12, OBUF_GAIN_6, OBUF_GAIN_0,	0 (success), -ve (failure)
int set_VGA_gain(char gain)	sets gain of VGA	VGA_GAIN_14, VGA_GAIN_12, VGA_GAIN_10, VGA_GAIN_8, VGA_GAIN_6	0 (success), -ve (failure)
int set_biquad_Q(char Q)	sets the value of Q in the biquad section	BQUAD_Q_0.8, BQUAD_Q_1, BQUAD_Q_1.33, BQUAD_Q_2, BQUAD_Q_1.5, BQUAD_Q_3	0 (success), -ve (failure)
int set_biquad_R(char val)	sets the value of biquad resistor Rbq in Kohms	BQUAD_R_1.4, BQUAD_R_11, BQUAD_R_22, BQUAD_R_44, BQUAD_R_88, BQUAD_R_176, BQUAD_R_352	0 (success), -ve (failure)
int set_biquad_C(int val)	sets the value of the biquad capacitor Cbq in pF(natural frequency of the biquad is Rbq*Cbq)	0<val <(2 ^10)-1	0 (success), -ve (failure)
int set_vga_R(char val)	sets the value of the VGA resistor Rvga in Kohms	VGA_R_5, VGA_R_20, VGA_R_40, VGA_R_80, VGA_R_160, VGA_R_320	0 (success), -ve (failure)
int set_vga_C(int val)	Sets the value of the VGA capacitor Cvga in pF(pole frequency Sets the value of the VGA capacitor Cvga in pF(pole frequency equals Cvga * Rvga)	0 <val <(2 ^11)-1	0 (success), -ve (failure)

Function Signature	Usage	Allowable argument values	Return value
int set_pma_Rin(char val)	sets the input resistor value that sets the gain of the PMA (Rf/Rin)	PMA_RIN_1, PMA_RIN_2, PMA_RIN_4, PMA_RIN_8, PMA_RIN_16	0 (success), -ve (failure)
int set_pma_Rf(char val)	sets the value of the feedback resistor of the PMA which sets the gain and the pole frequency.	PMA_RF_5, PMA_RF_10, PMA_RF_20, PMA_RF_40	0 (success), -ve (failure)
int set_pma_C(int val)	sets the value of the PMA capacitor Cpma in pF(pole frequency equals Cpma * Rf)	0 <val <(2 ^12)-1	0 (success), -ve (failure)

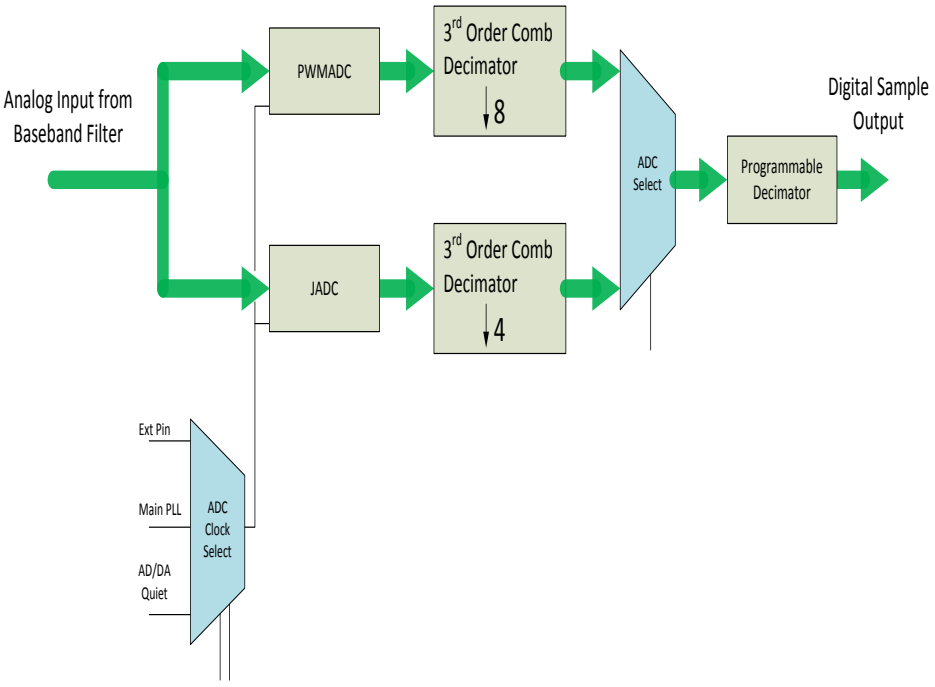


Figure 4.2: ADC Subsystem

Table 4.3: ADC Subsystem API Functions

Function Signature	Usage	Allowable argument values	Return value
int set_ADC_clk_src(char clk_src)	selects the clk source of the ADC.	ADC_EXT_CLK, ADC_PLL_2GHZ_OUT, ADC_AD_DA_QUIET	0 (success), -ve (failure)
int set_rxDSPClk_src(char dsp_clk_src)	selects the clk source of the decimation block.	DSP_ADC_CLK_DIV_8, DSP_ADC_CLK_DIV_4	0 (success), -ve (failure)
int init_PWM_ADC()	executes the initialization sequence for the PWMADC	NONE	0 (success), -ve (failure)
int init_JADC()	executes the initialization sequence for the JADC	NONE	0 (success), -ve (failure)

eight unique coefficients. This corresponds to up to a 15 tap non-half-band filter, or up to a 27 tap half-band filter.

The output decimation factor for both the PWMADC and JADC can be found using the following formulae:

$$Decimation_{PWM} = 8 \times [1-8] \times [1-8] \times [1-8] \tag{4.1}$$

$$Decimation_{JADC} = 4 \times [1-8] \times [1-8] \times [1-8] \tag{4.2}$$

4.1.2 TX Path blocks

The following section describes the TX path of the signal and all the processing blocks involved. Figure 4.4 is a simplified block diagram of the TX path. The digital baseband signal originates from

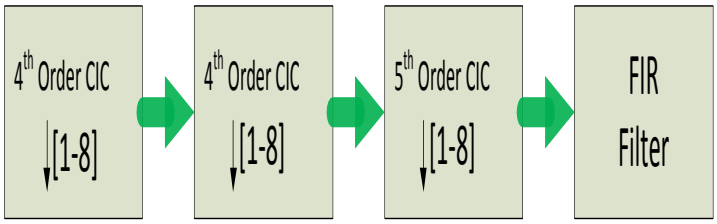


Figure 4.3: ADC Programmable Decimator

Table 4.4: Programmable Decimator API Functions

Function Signature	Usage	Allowable argument values	Return value
int set_ADC_decimation(char dec1, char dec2, char dec3)	sets the three decimation factors described in eqn 4.1 & 4.2	Integer values from 1 to 8	0 (success), -ve (failure)
int set_fir_coef(char coe_num, int coe*, char type)	sets the FIR filter coefficients	[coe_num] is the number of coefficients which can b any integer value from 2 to 8. [coe*] is a pointer to an array containing the coefficient values. [type] selects the type of the filter whether half-band or normal(FIR_NORMAL, FIR_HALF)	0 (success), -ve (failure)

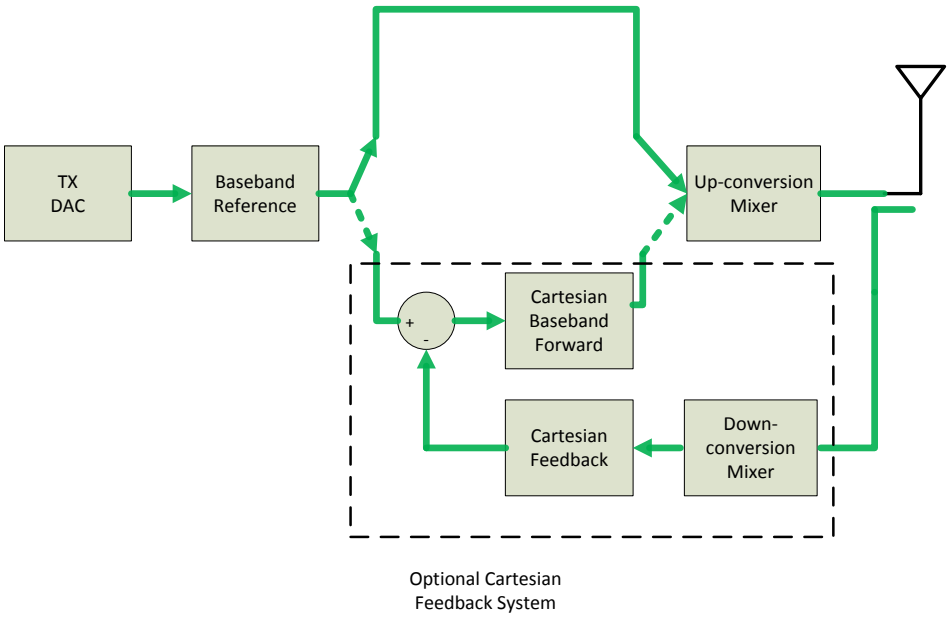


Figure 4.4: TX Path Block Diagram

the baseband processor and gets converted into analog form through the DAC block. The following block provides baseband filtering and signal level control through programmable attenuation. The signal can go through an optional summing node which adds a feedback signal sampled from the furthest point at the output(the antenna). The loop acts to correct for the imperfections introduced by the components, such as gain, phase and DC offset, so that the final output at the antenna matches the input reference signal.

TX DAC

The TX DAC used is based on a multi-bit sigma-delta architecture. It consists of a sigma-delta modulator followed by a thermometer-coded arrangement of current switching pairs to provide the different voltage levels.

A block in the RFIC known as the VAG voltage reference generates a stable bias current relative to temperature, which supplies the DAC core tail currents sources. The load resistors of the DAC are automatically adjusted to maintain a constant maximum output signal swing. For larger current

Table 4.5: DAC API Functions

Function Signature	Usage	Allowable argument values	Return value
int init_DAC()	initializes the state of the DAC	NONE	0 (success), -ve (failure)
int set_bit_num(char bit_num)	sets the unmodulated bit resolution by specifying the number of operational current diff pairs.	BITS_5, BITS_4, BITS_3	0 (success), -ve (failure)
int set_tail_cur(char curr)	sets the tail current of the diff pairs.	TAIL_CUR_15, TAIL_CUR_20, TAIL_CUR_25	0 (success), -ve (failure)

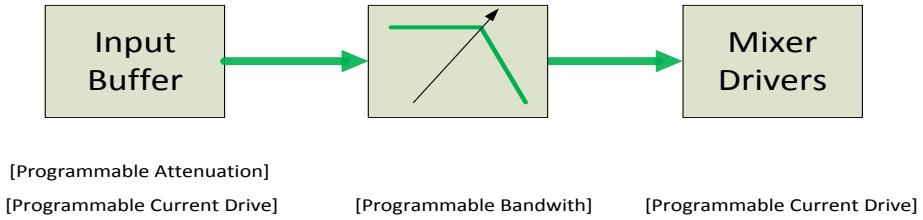


Figure 4.5: Baseband Reference Block

values, the value of the load resistors will be smaller resulting in lower thermal noise in exchange for greater power consumption. The number of operating current differential pairs can be varied to trade foff power consumption for quantization noise.

Baseband Reference(Signal Bandwidth + Level Control)

This block consists of three components as shown in Figure 4.5.The first component is an input buffer which interfaces the signal from the DAC and provides programmable attenuation to control the signal level. The second component is a butterworth filter with programmable bandwidth. It serves as an anti-imaging filter for the reconstructed signal coming from the DAC. The third component is a set of drivers with programmable current that drive the following up-conversion mixer blocks.

Table 4.6: Baseband Reference API Functions

Function Signature	Usage	Allowable argument values	Return value
int in_buf_attn(char attn)	sets the attenuation level of input buffer.	BBREF_BUF_ATTN_0dB, BBREF_BUF_ATTN_5dB, BBREF_BUF_ATTN_10dB, BBREF_BUF_ATTN_15dB	0 (success), -ve (failure)
int in_buf_drv(char cur_drv)	sets the current drive of the input buffer.	BBREF_BUF_DRV_2.8mA, BBREF_BUF_DRV_2.1mA, BBREF_BUF_DRV_1.4mA, BBREF_BUF_DRV_1.1mA	0 (success), -ve (failure)
int set_BW(char bw)	sets bandwidth of the butterworth filter	BBREF_BW_0.4MHz, BBREF_BW_0.8MHz, BBREF_BW_1.8MHz, BBREF_BW_3.5MHz, BBREF_BW_7.0MHz, BBREF_BW_10.5MHz	0 (success), -ve (failure)
int set_mixer_drv(char drv)	sets the current of the forward mixer drivers	BBREF_MIX_DRV_3.6, BBREF_MIX_DRV_1.8, BBREF_MIX_DRV_1.2, BBREF_MIX_DRV_0.9	0 (success), -ve (failure)

Table 4.7: RF Forward API Functions

Function Signature	Usage	Allowable argument values	Return value
int select_TX_ch(char ch_sel)	selects the TX output channel	TX_OUT_CH1, TX_OUT_CH2, TX_OUT_CH3	0 (success), -ve (failure)
int set_RF_passive_attn(char attn)	sets the attenuation of the RF passive attenuators	RF_PASS_ATTN_0dB, RF_PASS_ATTN_5dB, RF_PASS_ATTN_10dB, RF_PASS_ATTN_15dB	0 (success), -ve (failure)
int set_RF_out_driver_attn(char attn)	sets the attenuation of the RF output driver	RF_DRIVE_ATTN_0dB, RF_DRIVE_ATTN_10dB, RF_DRIVE_ATTN_20dB, RF_DRIVE_ATTN_30dB	0 (success), -ve (failure)

RF Forward Block

This is the final stage in the TX path before the signal leaves at the output ports of the chip. The signal can take one of three paths corresponding to three different channels. Each path contains the same set of components, an RF mixer, a passive step attenuator and an output driver, but with different capabilities. The first channel is optimized for power control covering a range of 80dB, the second channel is optimized for high linearity and the third channel is covers the widest frequency range, 2-6GHz.

Cartesian Forward Block

As explained previously, the Cartesian subsystem can be optionally inserted in the TX path, as shown in Figure 4.4, to form a feedback loop that counters the imperfections of the components in the TX path. The Cartesian Feedback block forms the feedback branch of the loop while the Cartesian Forward block forms the feedforward branch.

Table 4.8: RF Forward API Functions

Function Signature	Usage	Allowable argument values	Return value
int set_CART_FWD_[n]_gain(char gain) n= 1,2,3,4	sets the gain of the FWD blocks	CART_FWD_GAIN_20dB, CART_FWD_GAIN_15dB, CART_FWD_GAIN_10dB	0 (success), -ve (failure)
int set_CART_FWD_pole(char pole)	sets the Cartesian forward pole bandwidth (depending on the external caps used)	CART_FWD_POLE_12E6Hz CART_FWD_POLE_8E5Hz CART_FWD_POLE_4E5Hz CART_FWD_POLE_2E5Hz CART_FWD_POLE_1E5Hz In units of BW(Hz)/1nF Caps	0 (success), -ve (failure)
int set_CART_FWD_zero_gain(char zero)	sets the zero gain	CART_FWD_Z_GAIN40dB, CART_FWD_Z_GAIN35dB, CART_FWD_Z_GAIN30dB, CART_FWD_Z_GAIN25dB, CART_FWD_Z_GAIN20dB	0 (success), -ve (failure)

Cartesian Feedback Block

This block represents the feedback path in the Cartesian Loop system. It is effectively a complete receiver chain that samples the RF output at the transmit power amplifier and down-converts it to baseband before being fed into the summing node. The coupled RF signal first passes through a programmable RF gain block which either attenuates or amplifies the signal before going into the down-conversion mixer. The signal is then down-converted to baseband and into a programmable gain block followed by a programmable bandwidth filter. The composite gain and filter settings on both the forward and feedback Cartesian paths will determine the overall loop response as well the final output signal level.

Table 4.9: RF Feedback API Functions

Function Signature	Usage	Allowable argument values	Return value
int set_CART_FB_RF_gain(char gain)	sets the gain/attenuation of the RF amplifier	CART_FB_RF_GAIN_20dB CART_FB_RF_GAIN_10dB CART_FB_RF_GAIN_0dB CART_FB_RF_GAIN_-5dB CART_FB_RF_GAIN_-10dB CART_FB_RF_GAIN_-15dB	0 (success), -ve (failure)
int set_CART_FB_BB_gain(char gain)	sets the gain of the baseband amplifier	CART_FB_BB_GAIN_40dB CART_FB_BB_GAIN_35dB CART_FB_BB_GAIN_30dB CART_FB_BB_GAIN_20dB CART_FB_BB_GAIN_15dB CART_FB_BB_GAIN_10dB	0 (success), -ve (failure)
int set_CART_FB_filter_BW(char bwidth)	sets the bandwidth of the filter in the Cartesian feedback path	CART_FB_FILTER_BW_125 CART_FB_FILTER_BW_95 CART_FB_FILTER_BW_70 CART_FB_FILTER_BW_55 CART_FB_FILTER_BW_35	0 (success), -ve (failure)

Table 4.10: RF Feedback API Functions

Function Signature	Usage	Allowable argument values	Return value
int init_QUIET_[]() [RX] [TX_FW] [TX_FB] [ADC]	initializes the DDS blocks	NONE	0 (success), -ve (failure)
int init_QG_[](char mul_factor) [RX] [TW_FW] [TX_FB] [DIG_IF] [ADC]	initializes the QG blocks	DIV_BY_2, MUL_BY_2, MUL_BY_4	0 (success), -ve (failure)
int set_freq(float f_MHz)	sets the frequency of the output clk	100 <f_MHz <1000 Any real value in the above range is allowed.	0 (success), -ve (failure)
int set_phase(float p_rad)	sets the phase of the output clk	0 <p_rad <2 $\times pi$ Any real value in the above range is allowed.	0 (success), -ve (failure)

4.1.3 Frequency Synthesis

This subsystem consists of quadrature signal source blocks that supply a clock with programmable frequency and phase to the different RFIC blocks. The blocks that require the programmable clocks are: RX down-mixer, TX up-mixer, TX FB down-mixer, ADC and DAC subsystems and the Digital Baseband Link Interface.

DDS

The frequency synthesizer block is of a digital nature similar to the classical DDS(Direct Digital Synthesizer), but is based on a novel architecture which uses a technique known as digital to timer conversion(DTC) [9]. It has the same advantages of classical DDS such as precise control of both frequency and phase, fast switching and the ability to do DDM (Direct Digital Modulation). This architecture also suffers from spurious frequency content but uses special techniques that insert time domain corrections using controlled dither to reduce the spur level.

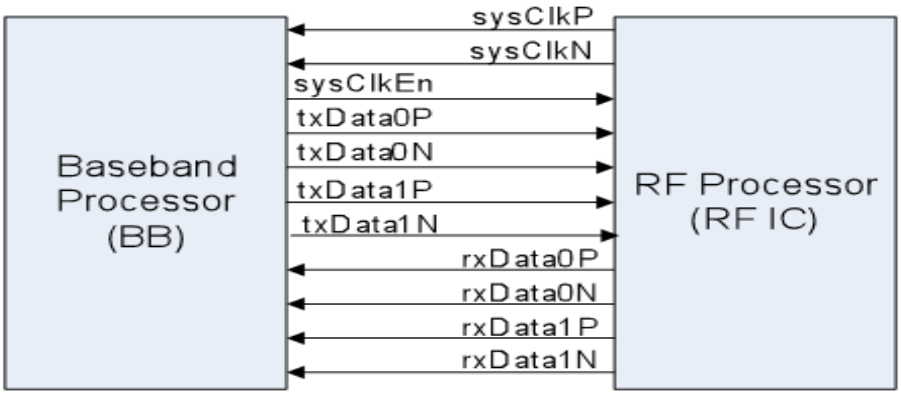


Figure 4.6: Digital Baseband Link Signals

4.1.4 High-speed Digital Baseband Interface

This interface is used to transfer digital samples between the baseband processor (BB) and the RFIC in both transmit and receive directions. It is a high-speed serial interface that uses Low Voltage Differential Signaling (LVDS) for the serial data bit stream as well as for the line clock. The line clock is sent by the RFIC to the Baseband IC and serves as a reference that determines the bit transfer rate across the link. There are four lines in each of the TX and RX direction(2 diff pairs for both I and Q), in addition to a diff pair for the clk signal and a single-ended line for a clk enable. Figure 4.6 shows the different signal lines between the BB processor and the RFIC.

The bits across the serial link are framed in a packet structure shown in Figure 4.7. The payload field contains the sample data in both the transmit and receive packets. Multiple possible sample word sizes can be used to tradeoff quantization noise for sample rate. Optional scrambling can be applied to the data bits in the payload which helps reduce level of spurs on the board signal lines that might cause interference with a received RF frequency of interest.

The packet structure consists of four fields:

1. Sync Word: It contains a sync pattern that is used by the receiver to identify the beginning of a frame.

2. Header:

Figure 4.7: Digital Baseband Link Packet Format

(a) Contains information about the combined size of the Time Stamp and Payload.

(b) Contains a CTS(Clear To Send) signal for the the transmit side.

3. Time Stamp: This is an optional field. In Rx packets it represents the value of the system time the first sample in the payload was acquired by the ADC. In Tx packets it represents the value of the system time the first sample in the payload should be sent to the DAC.

4. Payload: This field contains a sequence of signal samples. The size of this field in bytes is determined by the contents of the payload size register.

4.1.5 GNU Radio Interface Blocks

This section describes a pair of GNU Radio blocks that can be used to interface the PicoRF platform to the GNU Radio environment. The PicoRF-RX block encapsulates API functions that implement receiver functionality and acts as a signal source to other GNU radio blocks, while the PicoRF-TX block encapsulates API functions that implement transmitter functionality and acts as a signal sink.

PicoRF-RX Source Block

As shown in Figure 4.8, there are various input and output signals that communicate with the GNU radio block. The following list describes the different signals and their function:

1. Data out Port : This port produces the received signal samples in complex from where the real part is the I component and the imaginary part is the Q component.

Table 4.11: Digital Baseband Interface API Functions

Function Signature	Usage	Allowable argument values	Return value
int init_dig_IF()	initializes the digIF block	NONE	0 (success), -ve (failure)
int set_digIF_clk_src(char clk_src)	selects the digIF clk source	HS_TX_FB_QUIET, HS_AD_DA_QUIET	0 (success), -ve (failure)
int set_digIF_link_clk()	sets the frequency of the bit clk on the digIF link.	DIG_IF_CLK_DIV_2, DIG_IF_CLK_DIV_8	0 (success), -ve (failure)
int set_payload_sz(char p_sz)	sets the size of the packet payload in bytes	0 <p_sz <64. Any integer value within this range is allowed.	0 (success), -ve (failure)
int set_SYNC(short sync_word)	sets the 2 byte sync word	any 16 bit value is allowed	0 (success), -ve (failure)
int config_rx_link(char flags)	selects the various configurations for the Rx link packet using flags that can be or-ed together	Enable/disable time stamp RX_EN_TSTAMP Enable/disable scrambling RX_EN_RAND Sample size in bits RX_AD_SAMP_SZ_8b RX_AD_SAMP_SZ_12b RX_AD_SAMP_SZ_16b RX_AD_SAMP_SZ_20b Enable both I & Q lanes RX_EN_2_LANE	0 (success), -ve (failure)
int config_tx_link(char flags)	selects the various configurations for the Tx link packet using flags that can be or-ed together.	Enable/disable time stamp TX_EN_TSTAMP Enable/disable scrambling TX_EN_RAND Sample size in bits TX_AD_SAMP_SZ_8b TX_AD_SAMP_SZ_12b TX_AD_SAMP_SZ_16b TX_AD_SAMP_SZ_20b Enable both I & Q lanes TX_EN_2_LANE	0 (success), -ve (failure)

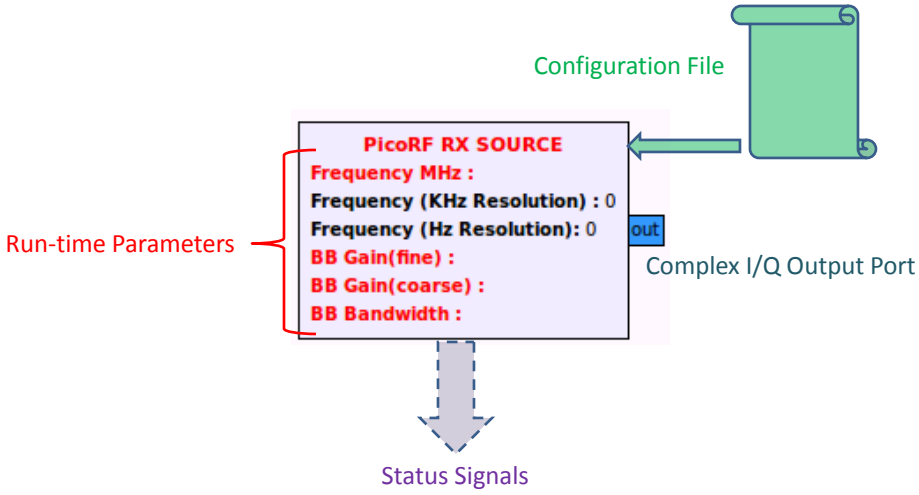


Figure 4.8: PicoRF RX Source Block: Data Port, Parameters and Status Signals

2. Run-time Parameters : This includes receiver parameters that can be changed during operation such as :

   (a) Frequency : This parameter determines the frequency of the down-conversion mixer. The frequency is specified using three inputs for each of the fractional parts of the frequency value(MHz, KHz, Hz).

   (b) BB Gain-Fine : This parameter specifies the gain applied to the signal in the baseband stage.

   (c) BB Gain-Coarse : This parameter also specifies gain in the baseband stage but can be changed in coarse steps of 6dB.

   (d) BB Bandwidth : This parameter specifies the bandwidth of the baseband filter.

3. Configuration File: This file holds the values of configuration parameters for the receive path components in the RFIC that cannot be changed during run-time. At initialization, parameter values are parsed from the file that are used by the system and remain constant throughout the period of operation. Such parameters include sample word size, decimation, packet size, LNA bias, RX channel number etc..

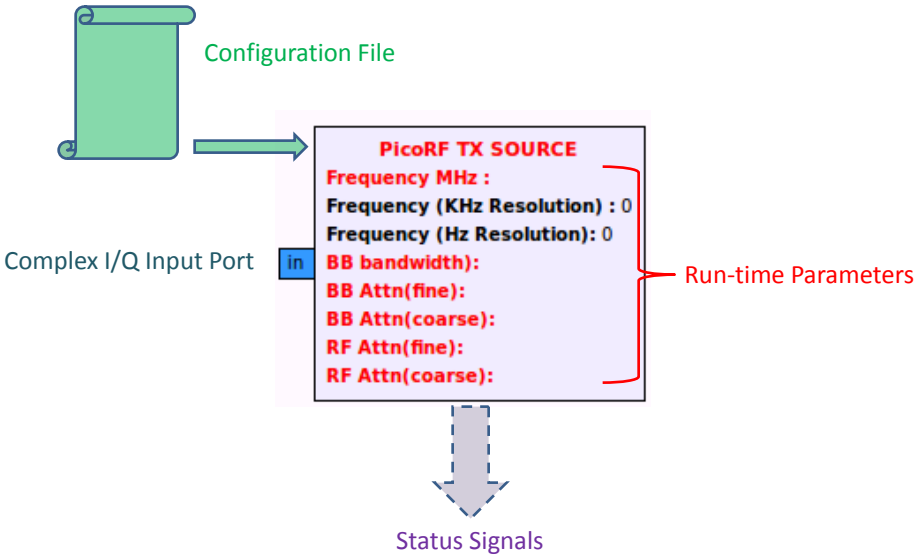


Figure 4.9: PicoRF TX Sink Block: Data Port, Parameters and Status Signals

4. Status Signals: These are a set of signals that can be interrogated (using get() functions) to monitor system information and the state of the different receive blocks. Such signals can be useful for debugging or can be used by a cognitive engine for automatic control. Such signals include:

   (a) Buffer Overflow : Indicates that the output sample rate from the PicoRF RX blocks is higher than the rate at which signals can be processed in GNU Radio.

   (b) Signal Present : Indicates the presence of a signal at the RF input having a power level above a certain threshold.

   (c) Sample Rate : Estimates the instantaneous rate of the incoming samples.

PicoRF-TX Sink Block

Having an almost similar interface to its RX counterpart, Figure 4.9 shows the PicoRF TX block and the various interface signals, which are described in the following list:

1. Data in Port : This port takes in signal samples in complex I/Q form.

2. Run-time Parameters : The set of parameters that can be changed during run-time:

   (a) Frequency : Same as the RX block but this for up-conversion.

   (b) BB Bandwidth : This parameter controls the bandwidth of the signal in the baseband stage.

   (c) BB Attn: This parameter controls the baseband attenuation.

   (d) RF Attn-Fine : This parameter controls the attenuation in the RF stage in fine steps.

   (e) RF Attn-Coarse : Control attenuation in the RF stage in coarse steps.

3. Configuration File : Has a similar function as for the RX block, but contains configuration parameters for components in the transmit path. Such as, TX channel No., DAC number of bits, DAC tail current etc..

4. Status Signals: Has a similar function as in the RX block. Signals include:

   (a) Buffer Underflow: Indicates that the output sample rate from GNU radio is low compared to the rate at which samples are being consumed by the PicoRF-TX.

   (b) Sample Rate : Estimates the instantaneous rate of the outgoing samples.

Chapter 5

Measurements and Results

This chapter includes tests and measurements that verify the targeted specifications of the PicoRF platform. The chapter is divided into two sections. The first section presents a number of basic tests and their results that verify the correct operation of various sub-blocks of the system in isolation. Next, detailed tests verifying the operation of the full system including the RFIC, FPGA and GNU radio control software are demonstrated.

5.1 Basic Sub-system tests

5.1.1 Data Receive/Read Channel (FPGA to PC)

Figure 5.1 shows the internal structure of the receive channel buffer that represents the gateway to the PCIe Interface and plays a significant role in determining the maximum channel throughput. It consists of two 32 bit wide buffer that receive inputs in an alternating fashion from the data input port. The output can be read in 32 bit size words by alternating between buffers or in 64 bit size by reading from both buffers simultaneously. This particular arrangement provides an efficient connection with the Xilinx PCIe Core Transaction 64-bit Interface by allowing data input to be delivered to the core every clock cycle. Due to the

fact that the TLP is organized in 32 bit units, or DWORDS, while the Transaction Interface accepts data in 64 bits a complication arises in two situations. The first situation is when sending the second 64 bit word to the core transaction interface with half header information and half 32 bit data, and the second is when sending the last DWORD in an even-sized TLP packet. The dual read size read capability helps take care of these situations.

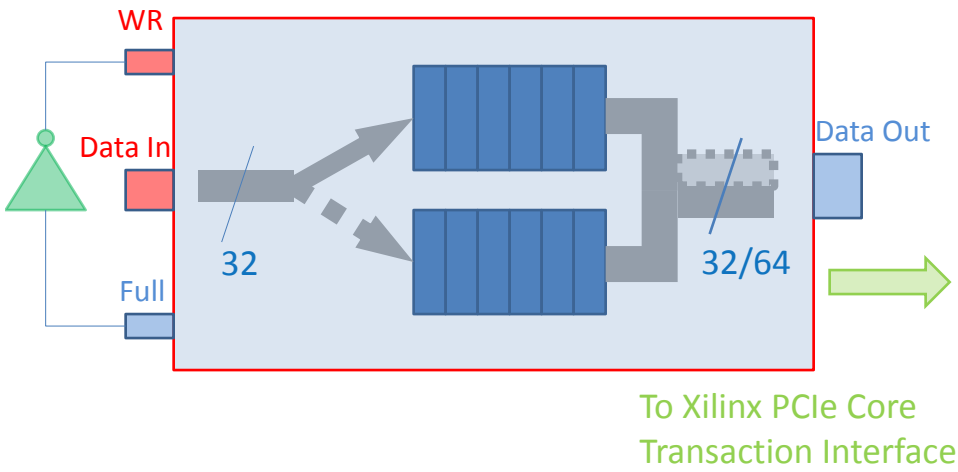


Figure 5.1: Dual Read Size Buffer for Efficiently Interfacing with the Xilinx PCIe Core

In order to measure that maximum possible data transfer throughput from the FPGA to the PC, the buffer wr line is connected as shown in Figure 5.1 forcing maximum rate input into the buffer.

Results

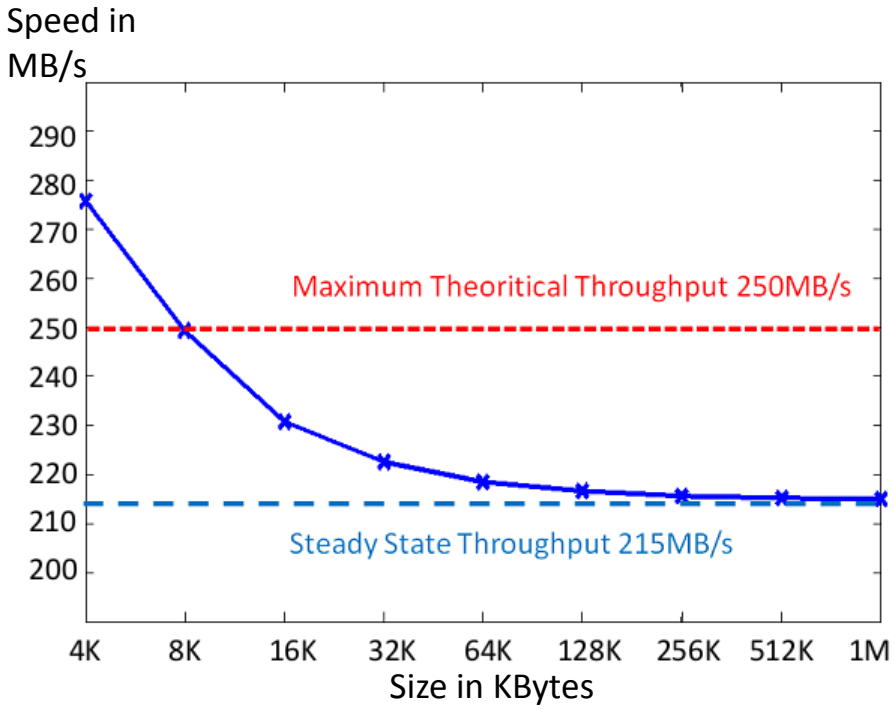


Figure 5.2: Data Read Channel Maximum Throughput Averaged over Different Data Transfer Sizes

Table 5.1: Data Read Channel Max Throughput Measurements for Different Data Transfer Sizes

Data Block Size	Max Time(us)	Min Time(us)	Avg Time(us)	Speed(MB/s)
4K	14.84	14.84	14.84	275.86
8K	32.896	32.784	32.8464	249.4
16K	71.02	70.96	70.98	230.82
32K	147.32	147.2	147.24	222.5337
64K	299.92	299.728	299.8048	218.5956
128K	604.928	604.736	604.8432	216.7041
256K	1215.1	1214.9	1215	215.754
512K	2435.5	2435.1	2435.3	215.287
1M	4876.4	4875.6	4875.9	215.0548

Analysis

The following observations can be made about Figure 5.2:

- The throughput exceeds the theoritical limit when calculated over a data transfer size of 4K. The PCIe core interface operates at 62.5MHz and can accept 64 bits every clock cycle(except for start, stop and header clk cycles), thus theoritically supporting up to 500MB/s. However, since the the actual PCIe link can only support up to 250MB/s(2Gbps), the core will limit the incoming input rate to match the link rate using flow control signals. The measured rate is that of the consumption of data by the PCIe core i.e. input rate, while the desired rate is that of the link which is the output

rate of the core. Given the buffering present inside the PCIe core, the observed input rate can be higher than the output rate thus explaining the exceeding of the theoritical limit.

- The throughput approaches a steady value of 215MB/s. This is a more accurate estimate of the max throughput since it is calculated over large data sizes where buffering effects become negligible and the input rate to the PCIe core approaches its output rate, which is the link rate.

- At steady state, the dynamic behaviour of the buffer in Figure 5.3 shows that 50% of the time 64 bit words are sent to the input of the PCIe Core every clock cycle, leading to a net througput of 250MB/s which is the theoritical limit. A single TLP transfer takes 18 cycles, 16 cycles which are purely data and the other two are for header information. This leads to an efficiency of 88% resulting in a throughput of 220MB/s which is very close to the measured steady state rate. This shows that the hardware interface used in this design can be considered optimal.

- The maximum sample rate supported by the RFIC5 is 250MS/s with a maximum sample size of 20 bits, which implies that the Data link capacity must be 5Gbps or 625MB/s. However, the digital interface of the RFIC can only support bit rates up to 1Gb/s or 125MB/s. Since we have shown that the Receive Channel can sustain a rate of 215MB/s, all sample rates allowed by the RFIC5 will be accommodated.

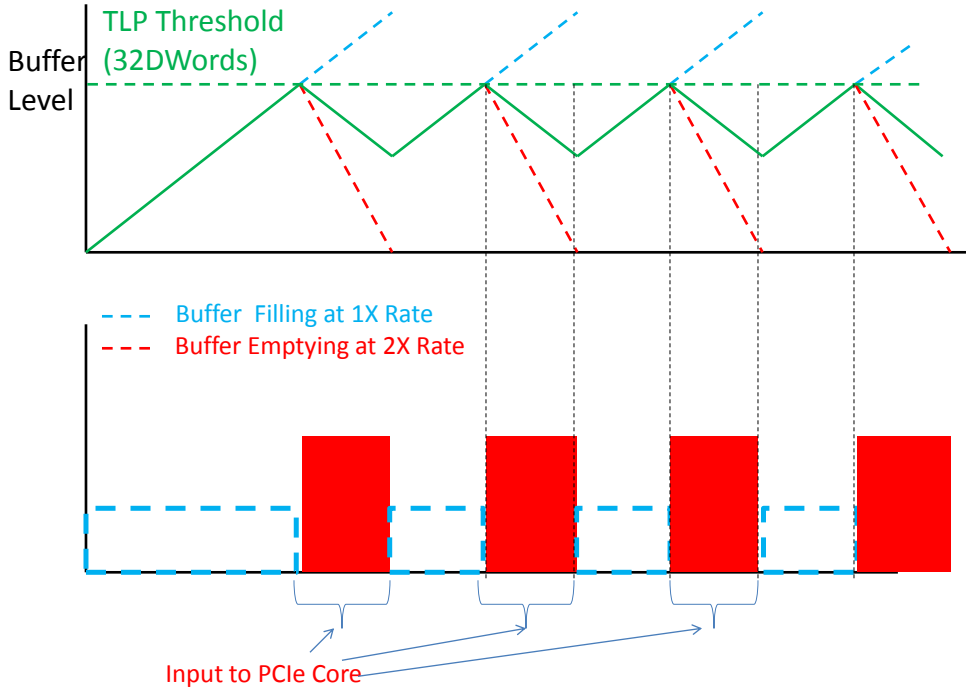


Figure 5.3: Receive Channel Buffer Dynamics

5.1.2 Data Transmit/Write Channel (PC to FPGA)

A dual write size buffer is also used in the transmit channel which does the opposite function of the buffer used in the receive channel. Its internal structure is shown in Figure 5.4. The buffer rd line is connected to the complement of the buffer empty signal to force the maximum data output rate from the buffer.

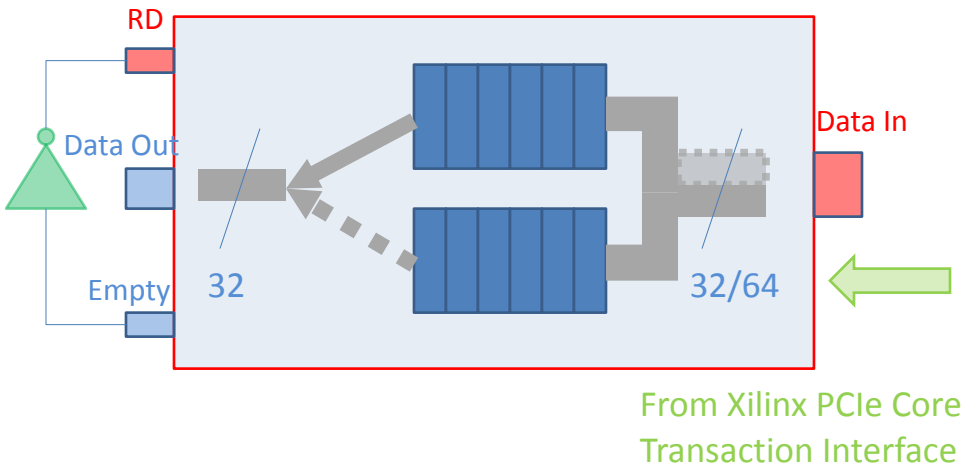


Figure 5.4: Dual Write Size Buffer for Efficiently Interfacing with the Xilinx PCIe Core

Results

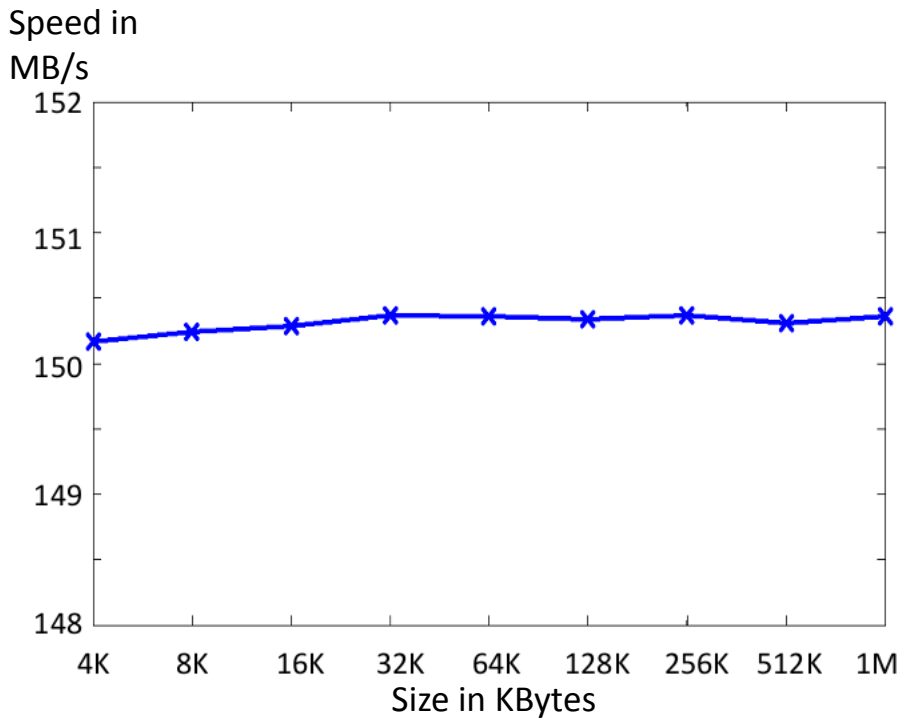


Figure 5.5: Data Write Channel Maximum Throughput Averaged over Different Data Transfer Sizes

Table 5.2: Data Write Channel Max Throughput Measurements for Different Data Transfer Sizes

Data Block Size	Max Time(us)	Min Time(us)	Avg Time(us)	Speed(MB/s)
4K	27.376	27.168	27.2768	150.1642
8K	54.576	54.4320	54.5264	150.2392
16K	109.168	108.816	109.0208	150.2832
32K	218.128	217.68	217.9184	150.3682
64K	436.0160	435.664	435.8672	150.3577
128K	872.24	871.536	871.8311	150.341
256K	1743.6	1743	1743.3	150.3708
512K	3494.4	3486.9	3488	150.3103
1M	6974.3	6973.4	6973.8	150.3588

Analysis

The following observations can be made about Figure 5.5:

- The max throughput is steady at 150.3MB/s and is much lower than the receive channel throughput(almost less by 30%) and the max theoritical limit. This can be explained by the fact that the process of writing packets from PC to FPGA(reading by FPGA from PC) involves more overhead than the reading process. In this design and all modern designs in general, data transfers are done using bus-mastered DMA, which means that all data transfers have to be initiated by the hardware side i.e. the FPGA. This makes the process of data transfer along the two directions asymmetric since data

writes by the hardware(reads by the PC) involve sending TLPs containing data to the PC side, while reads(writes by the PC) on the other hand involve first sending a zero payload request packet to which the PC responds by a completion packet after a certain amount of latency. Thus, there are two main overheads involved in the write operation:

- Request TLPs sent by the hardware.
- Latency between request TLP and response by PC side.

This is illustrated in Figure 5.6, which is a chipscope capture of a PC write(hardware read) operation.

- Even though the achievable transfer rate is not the absolute maximum, 150MB/s is enough to cover all sample rates allowed by the RFIC5.

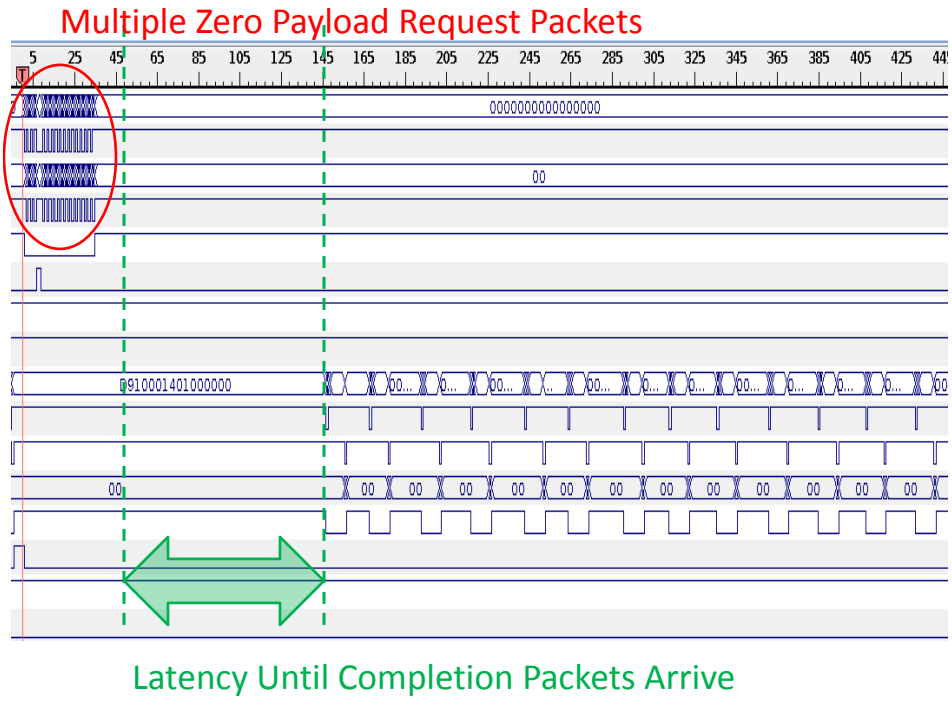


Figure 5.6: Chipscope Capture Demonstrating Packet Sequence in Write(Hardware Read) Operation

5.1.3 FPGA Configuration Channel

The FPGA Configuration channel uses a similar hardware interface to that of the Data Write channel and thus is expected have a similar performance. A slot is defined by the region of resources marked by red shown in Figure 5.7. The slot contains a sufficient set of resources to implement relatively complex communication building blocks. Implementations have shown that half the slot size(0.5X) was sufficient to implement the following components:

- FIR Filter with 31 taps.
- AM Receiver made up of CORDIC square root and DC removal block.
- FM Receiver made up of CORDIC inverse tan function and conjugate multiplier(3 multipliers).
- ADPLL(All Digital Phase Locked Loop).

Results

Table 5.3: Configuration Channel Download Times for PR Modules of Different Size

Relative Size	Max Time(ms)	Min Time(ms)	Avg Time(ms)	Size(KB)	Speed(MB/s)
0.5X	0.492	0.502	0.496	70.148	141.4
1X	0.969	0.97	0.97	140.296	144.6
2X	1.927	1.932	1.93	280.593	145.384
4X	3.847	3.872	3.858	561.187	145.46

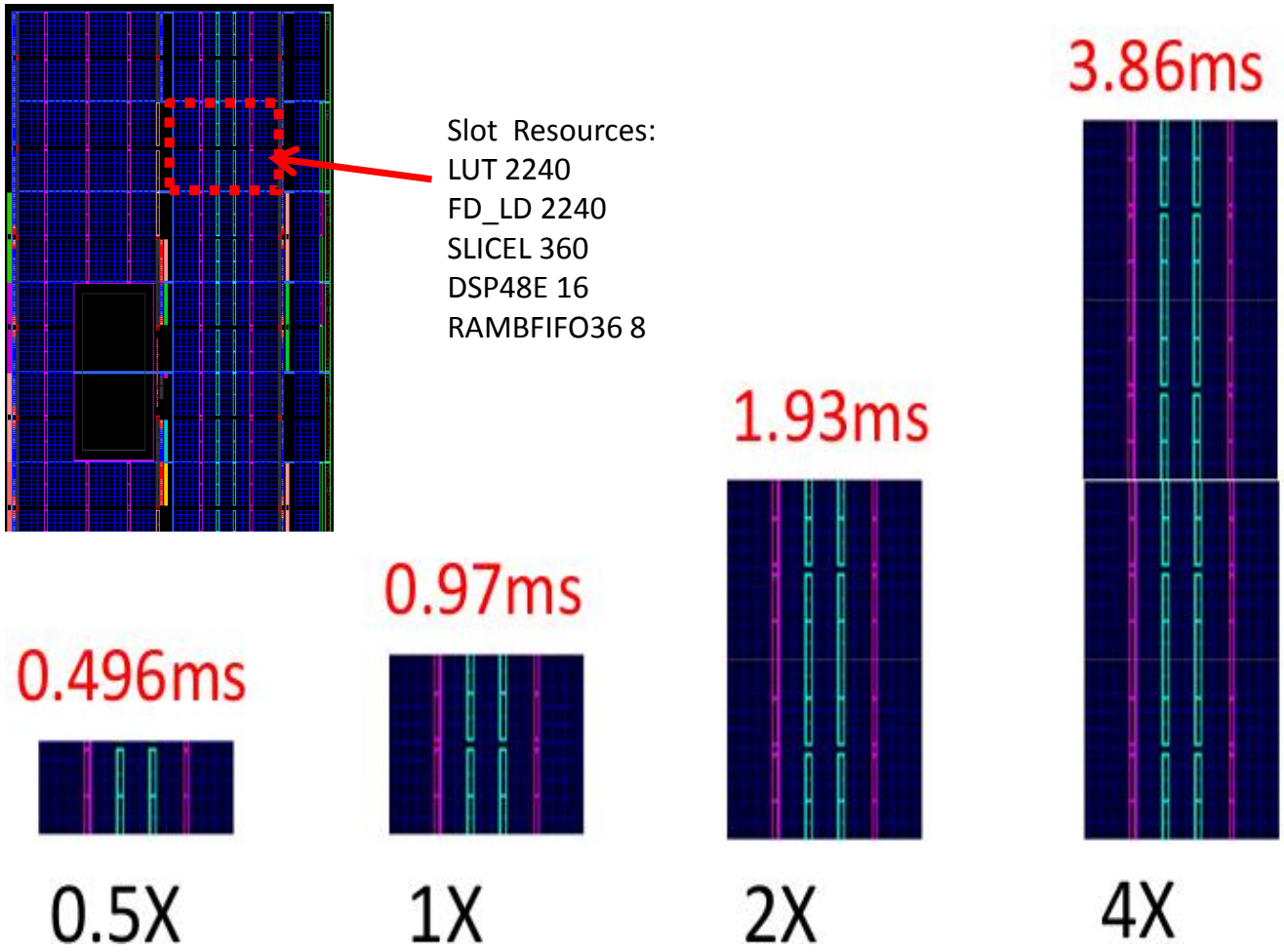


Figure 5.7: Reconfiguration Time versus Area relative to the Basic Slot Size

Analysis

The following observations can be made about Figure 5.7:

- The max configuration speed is close to the write channel throughput but slightly lower at 145MB/s. This is because, unlike the test used in the case of the write channel where the buffer was outputting at max rate, the rate of emptying the buffer is determined by the configuration controller which is reading from the buffer. This was necessary since the intent is to measure the time required to complete a reconfiguration operation not

the max throughput.

- At 4X, reconfiguration time is less than 4ms which implies that the entire PR grid(8X) can be reconfigured in less than 10ms.
- Even though the PC-to-FPGA lane of the PCIe link is shared by both the Configuration and Data Write channels, Configuration Channel transfers are assigned a higher priority over Data Write transfers guaranteeing a fixed throughput and a constant component loading time to satisfy radio run-time constraints.

The above tests show that the individual subsystems are all functioning reliably and with sufficient performance to satisfy the requirements of the target high performance SDR platform with run-time reconfiguration capabilities.

5.2 Full System Tests

In this section, full system runs will be carried out to verify the correct operation of all the individual subsystems while interacting together to perform the overall system function.

5.2.1 RX Path Functionality

This test is very simple yet its success implies the correct operation of all the components in the system along the RX path, which are:

- RFIC hardware and link to FPGA.
- FPGA link to PC through PCIe.
- RFIC API Functions.

Test Setup

As shown in Figure 5.8, an RF sinusoidal signal is used as input to one of the RX ports of the RFIC. The baseband signal is examined on the PC side using GNU Radio tools, FFT and Scope. The control panel shown in Figure 5.9 is used to control and monitor the various parameters of the PicoRF receive path. By varying the controls and observing the signal characteristics, the correctness of the API control functions can be verified.

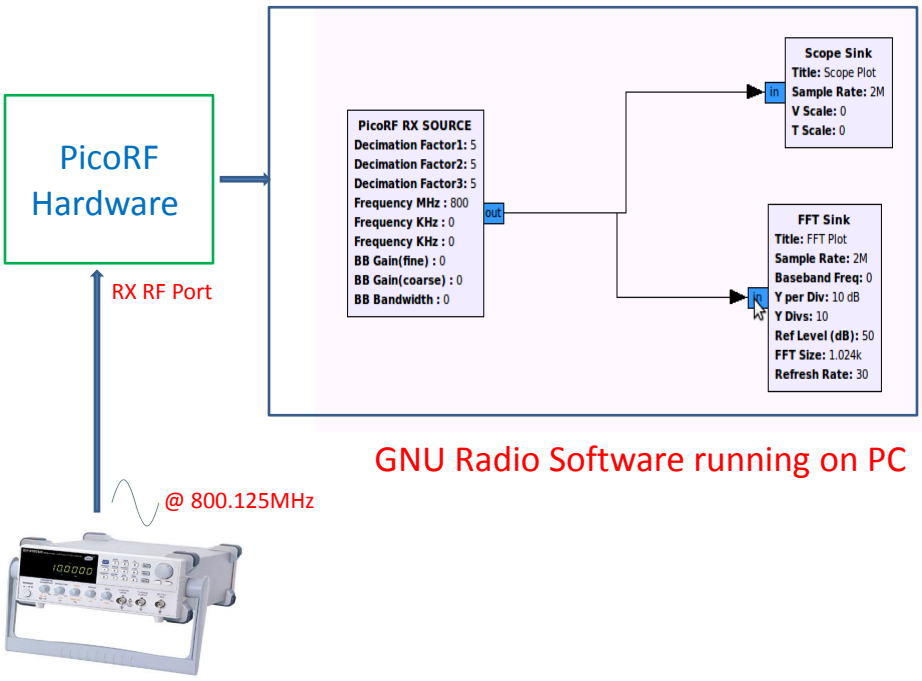


Figure 5.8: RX Path Test Setup

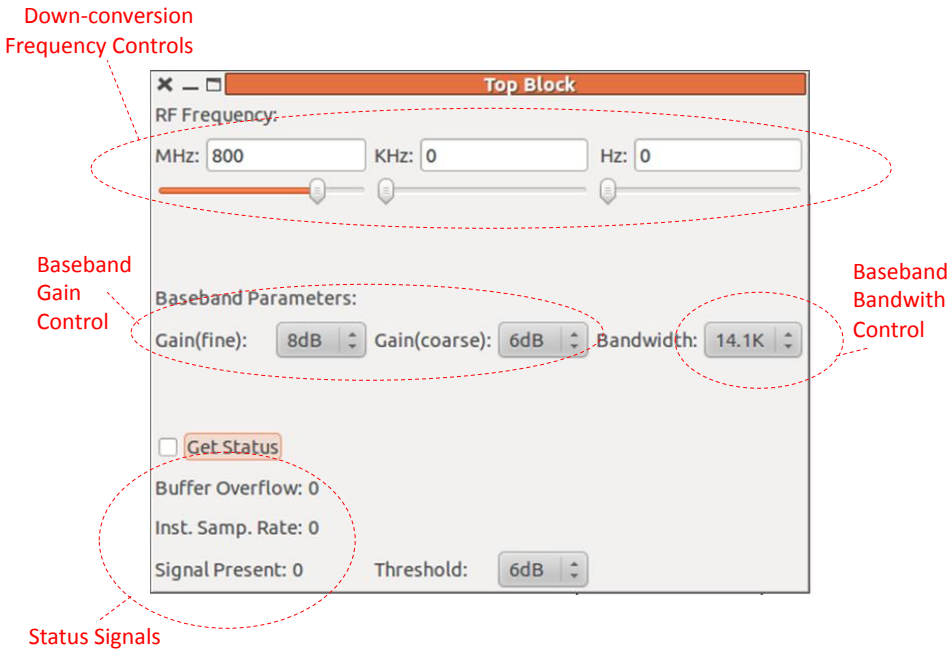


Figure 5.9: Graphical Control Panel for controlling the parameters of the PicoRF_rx_src block

Basic Signal Plots

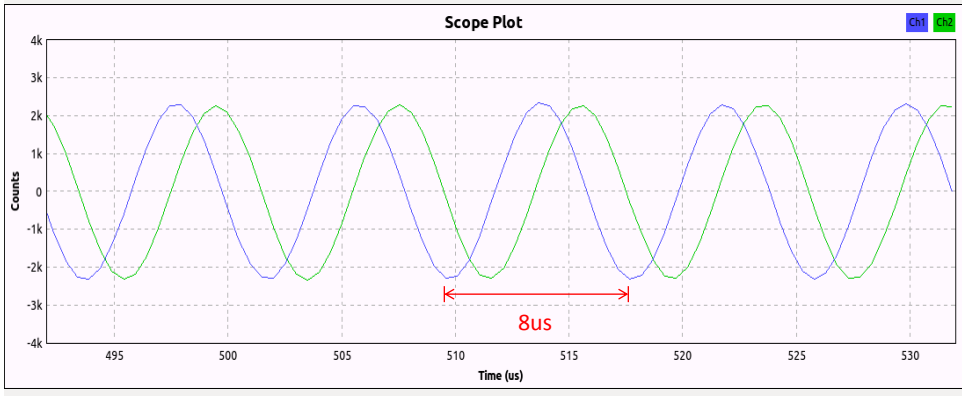


Figure 5.10: Scope Plot showing I/Q components for a 125KHz sinusoid

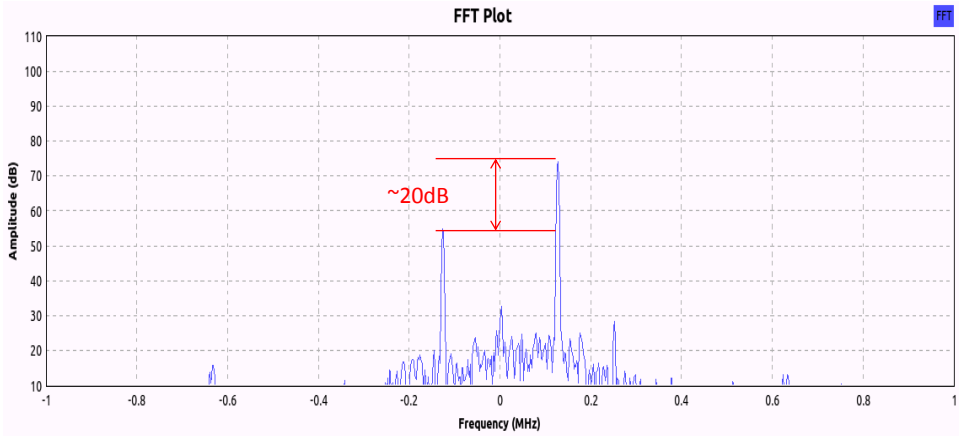


Figure 5.11: FFT Plot showing peaks at +/-125KHz with 20dB side-band rejection

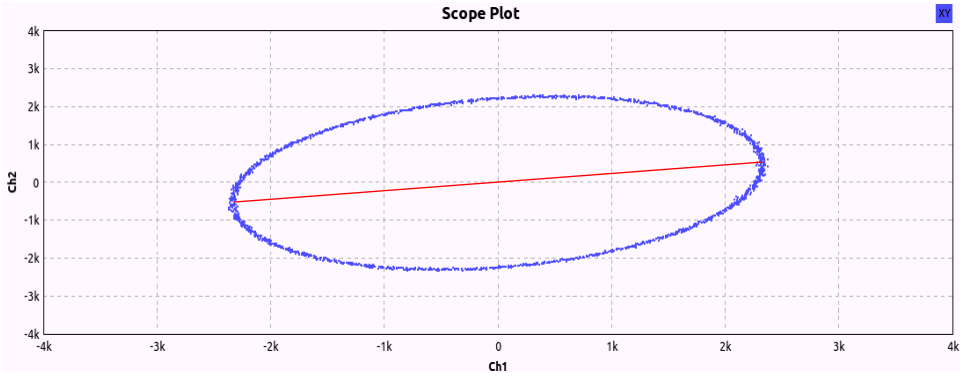


Figure 5.12: XY scope Plot showing a slight I/Q phase mismatch(axis of the ellipse)

By closely examining Figures 5.10, 5.11 and 5.12 , we can verify the correctness of the following aspects of the platform:

- Time Base: Given that the input signal frequency is known to be 125KHz, the time base/sample rate is verified to be programmed at the correct value since the scope plot, which takes a user specified sample rate parameter, shows the correct time period which is 8us. This also confirmed by the FFT plot which shows a spike at the correct frequency.

- DCOC(DC Offset Correction): By observing the scope plot, it can be seen that there is no DC offset in the signal which implies that the DCOC unit in the RFIC is functioning

correctly. When the DCOC block was disabled, it was found that the I/Q components were offset from the x-axis with unequal amounts. Also, the FFT plot shows the absence of a DC component at zero frequency.

- Amplitude Imbalance: The scope plot shows that both the I and Q components are at equal amplitude as desired with no amplitude imbalance. This is not always the case for all frequencies. Experimenting has shown that Amplitude and Phase imbalance is not constant across the frequency operating range of the RFIC[100MHz-2.4GHz].

- I/Q Phase Imbalance: A quick glance at the scope plot gives the impression that the I/Q components are out of phase by exactly 90 degrees as desired. However, by examining the constellation plot it can be seen that this is not the case as indicated by the ellipsoidal pattern that has a slightly tilted axis. This manifests in the frequency domain as a frequency component showing up at the lower side band frequency with a smaller amplitude compared to the upper side band. This difference in amplitude is known as the SRR(Side-band Rejection Ratio). The FFT plot shows that the SSR is approximately 20dB. It should be pointed out that this I/Q phase imbalance can be reduced by programming the correct values into the RFIC LO synthesizer control registers. However, the required values are not provided in the manufacturer datasheet, an alternative solution is to implement an I/Q imbalance compensation block in the digital domain.

The RFIC covers an extremely wide range of frequencies [100MHz-2.4GHz] which can be swept in Hz resolution. This, in addition to all the possible parameters that can vary, makes the full characterization of the RFIC a truly formidable task, requiring a great deal of effort and expertise which is beyond the scope of this work. By using the above test setup and the RFIC API functions, test procedure scripts can be designed to automate the characterization of the RFIC, which can greatly reduce the required time and effort.

API Control Functions

On of the very useful features of the RFIC is its support for programming a wide-range of parameters during run-time. Using the control panel in Figure 5.9, the baseband filter bandwidth and gain can be varied interactively while monitoring the signal on the scope and FFT plots. This will verify if the response of the RFIC is both consistent with the programmed values and repeatable.

- Baseband BW Control: A wideband noise signal is applied to show the effect of the variation of the baseband filter bandwidth.

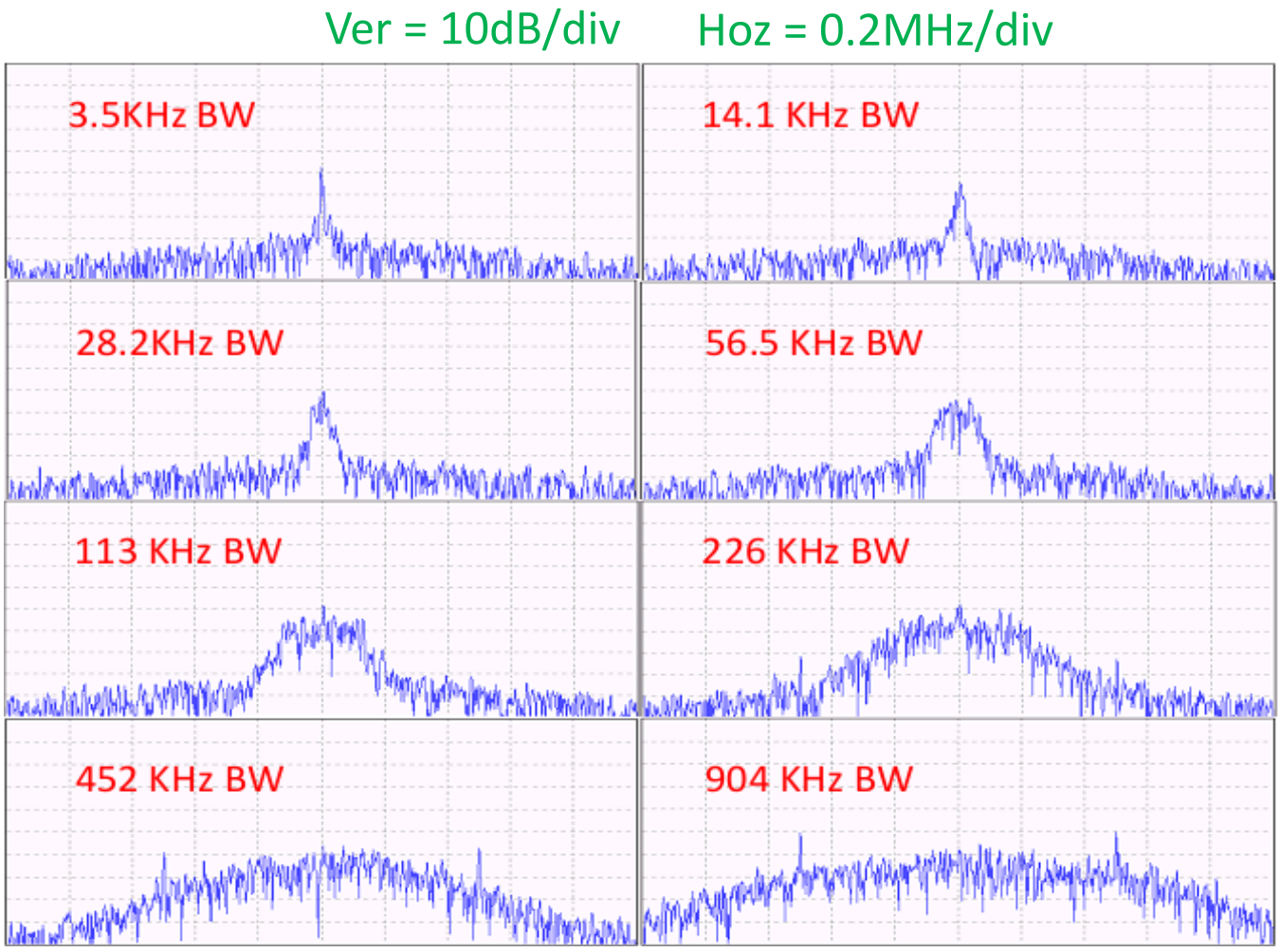


Figure 5.13: Resulting baseband bandwidth for different filter selections

- Baseband Fine Gain Control:

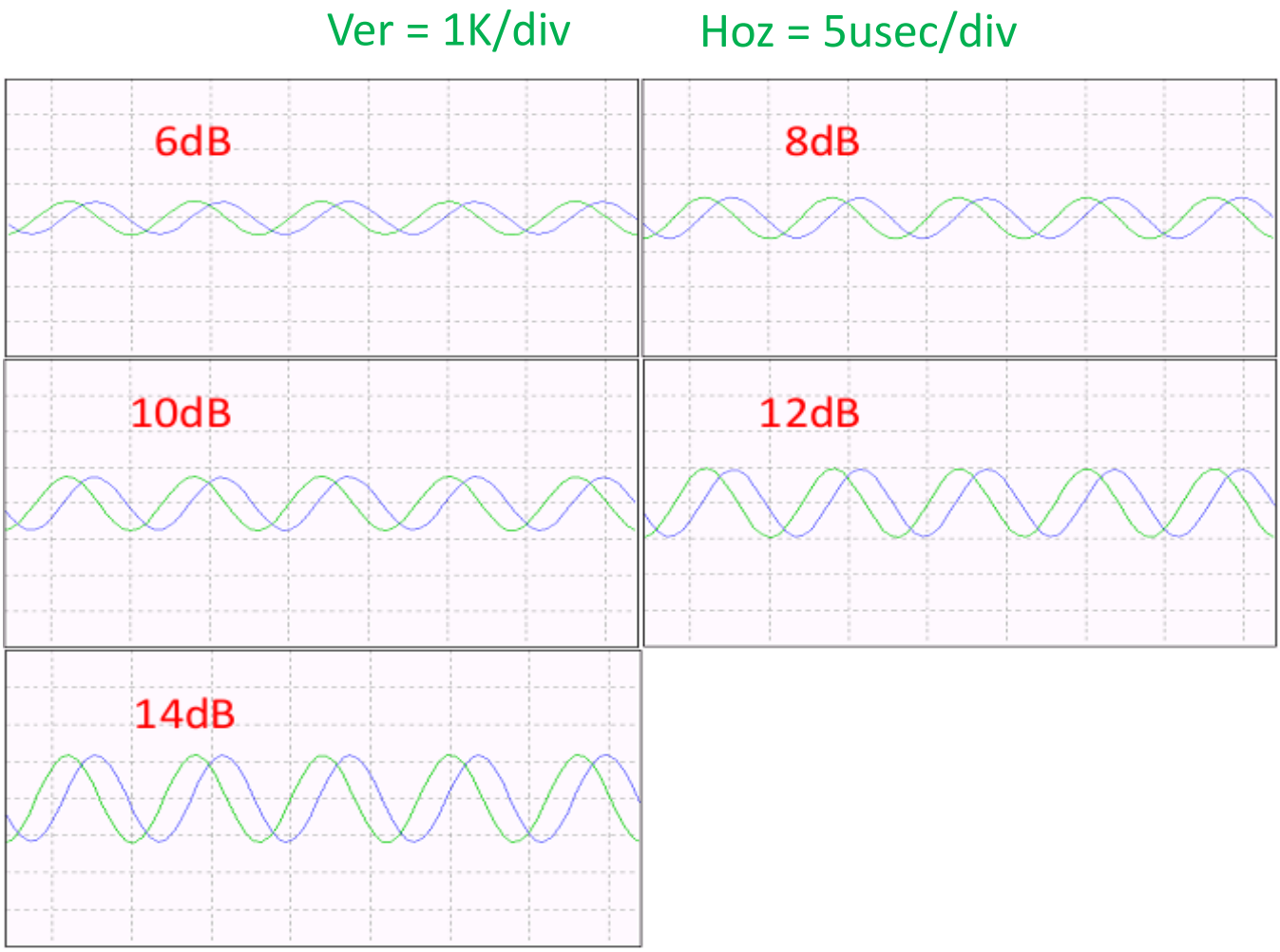


Figure 5.14: Fine change in amplitude of input sinusoidal signal for different gain selections

- Baseband Coarse Gain Control:

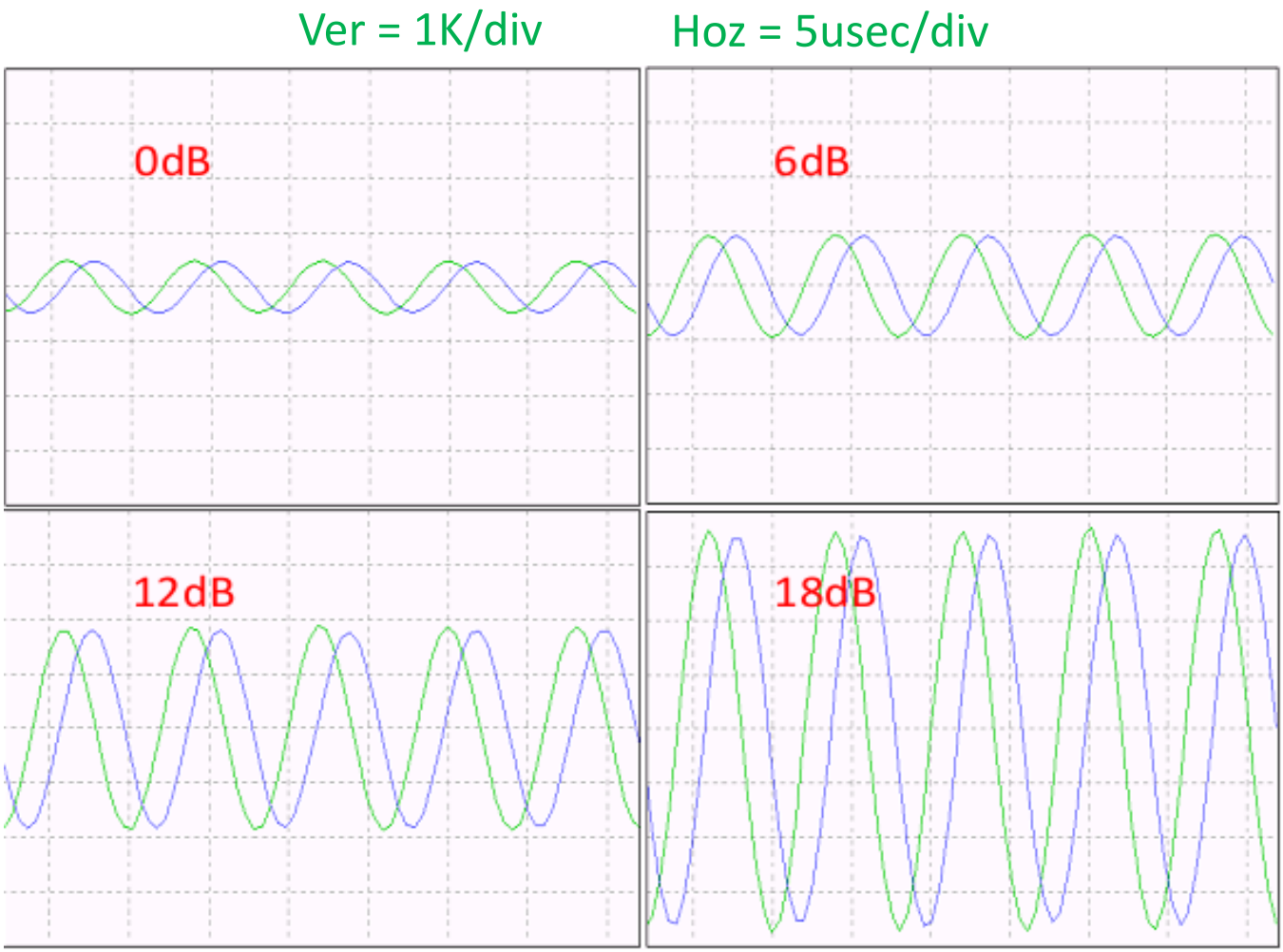


Figure 5.15: Coarse change in amplitude of input sinusoidal signal for different gain selections

The following observations can be made about the above plots:

- For the scope plots, comparing the dB difference of the programmed gain values(shown in red) with the ratio of measured amplitudes(using the Amplitude/div scale) for two different gain settings shows that they are both in agreement.
- Similarly, the FFT plot shows the doubling of bandwidth of the passed noise when doubling the programmed value of the filter bandwidth. For this test, the input port was left open to simulate the effect of a wideband noise signal input, however a more

accurate test would be to use an actual noise signal generator which will have a flatter gain that can more clearly show the filter frequency response function.

Sample Rate Selection

The RFIC can be programmed through 3 decimation factors, as described in the previous chapter, to determine the sample rate of the received signal. The decimation factors are passed to the PicoRF_rx_src block as initial parameters. Unlike the case in the USRP platform where part of the FPGA logic is occupied by decimation and down-conversion logic, the decimation and down-conversion is handled completely inside the RFIC which saves the FPGA resources for other functions.

In Figure 5.16, the sampled signal is captured at different sample rates to verify the correctness of the sample rate API control functions.

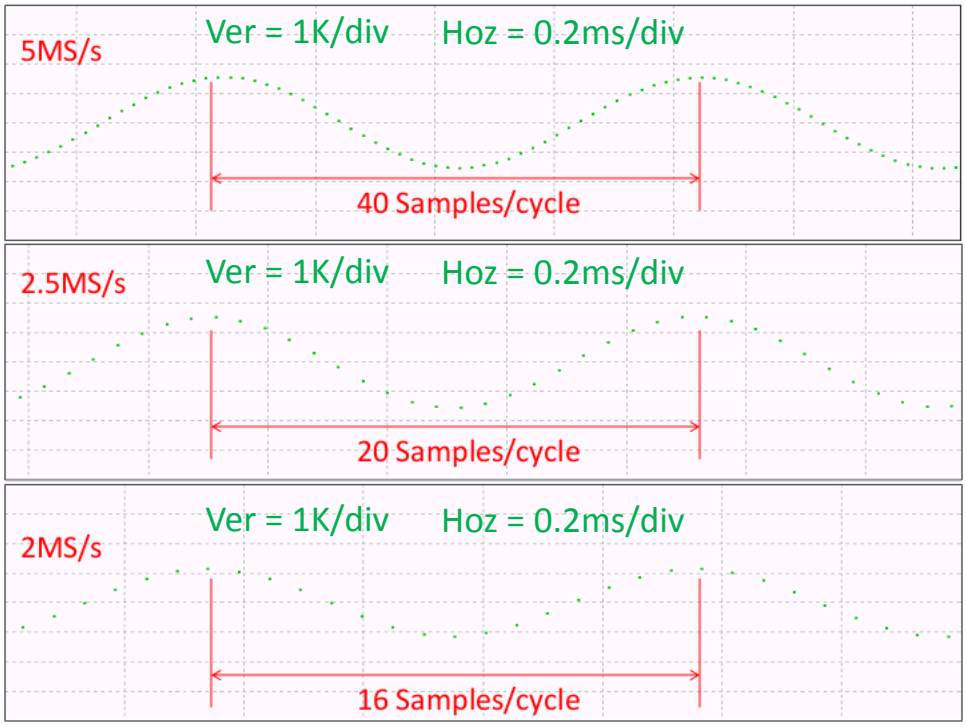


Figure 5.16: Signal samples for different sampling rates

Based on the above results, it can be said that API functions are functioning correctly and can be relied on when designing radio applications that target the RFIC.

5.2.2 Manual Run-time Reconfiguration of the RX Path Waveform

The purpose of this section is to demonstrate the correct operation of the run-time reconfiguration functionality described in chapter 4, in the context of a radio receiver application that can demodulate multiple waveforms using the same set of hardware resources.

Test Setup

Figure 5.17 shows the system setup. The RX path utilizes 2 PR slots out of the PR grid for hosting the waveform processing blocks. Using the graphical control panel shown in Figure 5.18, a reconfiguration action is triggered manually to change the waveform processing blocks in the RX path. The radio buttons in the control panel are used to select either bypass or AM and FM waveforms.

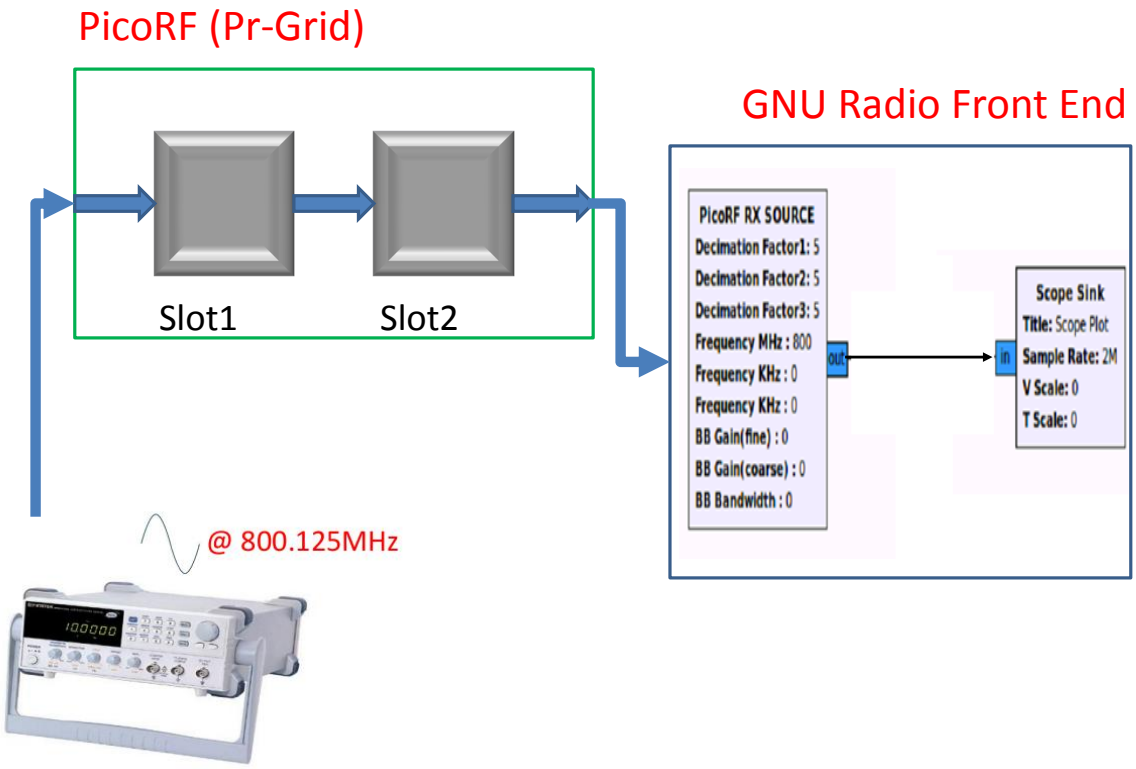


Figure 5.17: 2 PR Slots are included in the RX Path to host waveform processing blocks

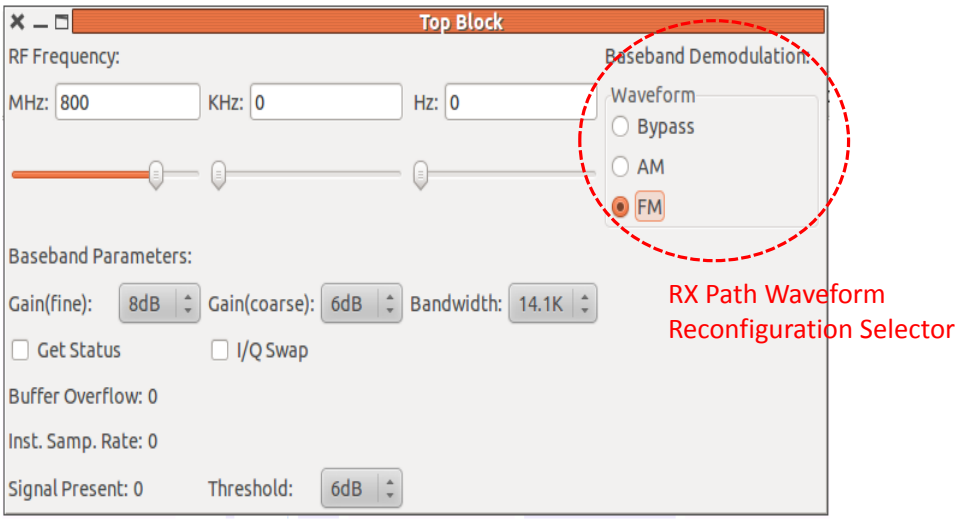


Figure 5.18: Control Panel with Radio Buttons to trigger a Partial Reconfiguration operation to modify the Demodulation Waveform

Figure 5.19 shows the DSP components used in the slots for the AM and FM waveforms. For the AM waveform, the first slot contains a bypass or a pass-thru block and the second slot contains a CORDIC square root function. For the FM waveform, slot1 contains a multiplier block that multiplies each input sample by the conjugate of the previous one. In the second slot, a CORDIC inverse tan function is used. The final output is the phase difference between 2 consecutive samples, which is a good estimate of the instantaneous frequency given the sample rate is high enough.

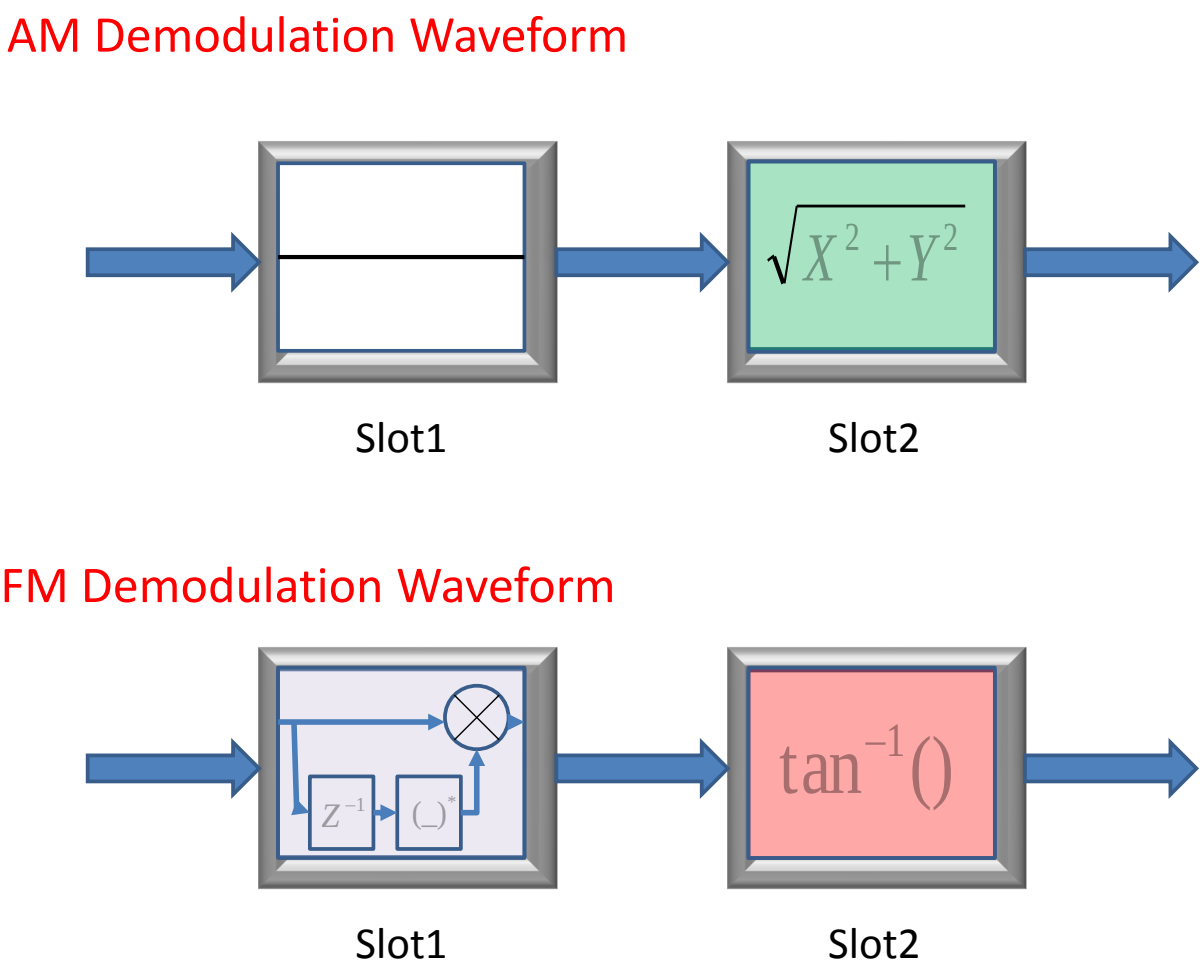


Figure 5.19: DSP Blocks used to implement each of the AM and FM Demodulation Waveforms

AM Waveform

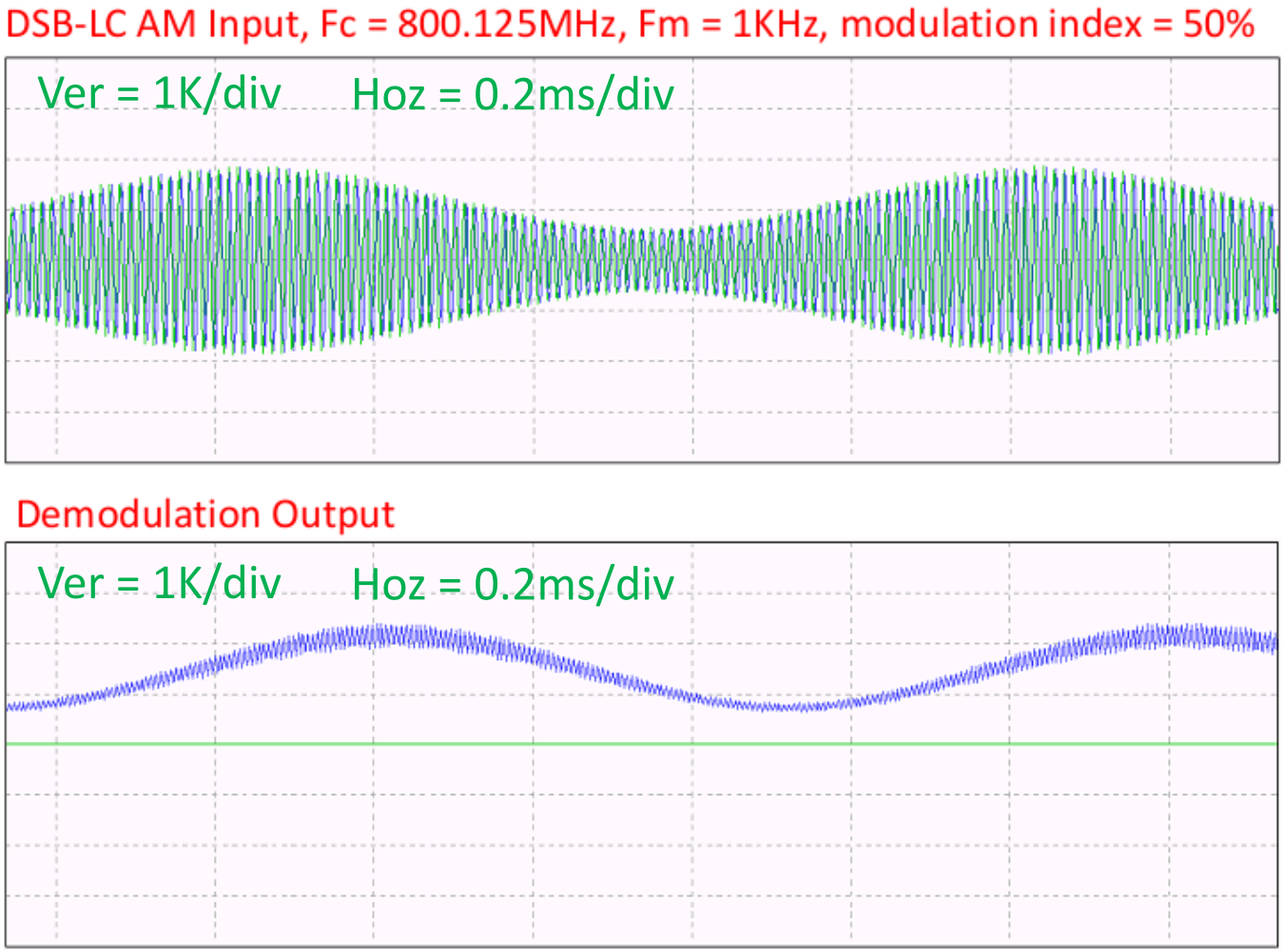


Figure 5.20: Input AM Signal and Demodulated Output

By looking at Figure 5.20, it can be seen that this is the expected output since it matches the envelope of the AM input signal. In addition, a foreign high frequency ripple component is superposed on the signal which is a deviation from the expected ideal output. After further investigation this was found to be a reault of the I/Q phase imbalance cause by the RFIC RX path components. The following section presents a mathematical analysis of this effect and a Matlab simulation in Figure 5.21 that confirms this explanation.

Analysis of I/Q Phase Imbalance on AM Demodulation Performance

A AM modulated signal can be represented in complex form by the following equation:

$$f(t) = Ae^{j\omega_c t} + m(t)e^{j\omega_c t} \tag{5.1}$$

where $m(t)$ is the modulation signal, A is the carrier amplitude and ω_c is the carrier frequency.

For a sinusoidal modulation signal:

$$f(t) = Ae^{j\omega_c t} + \sin(\omega_m t)e^{j\omega_c t} \tag{5.2}$$

Expanding in complex I/Q form:

$$f(t) = (A + \sin(\omega_m t))\cos(\omega_c t) + j(A + \sin(\omega_m t))\sin(\omega_c t) \tag{5.3}$$

Where:

$$I(t) = (A + \sin(\omega_m t))\cos(\omega_c t) \tag{5.4}$$

$$Q(t) = (A + \sin(\omega_m t))\sin(\omega_c t) \tag{5.5}$$

To simulate the effect of I/Q phase imbalance, a phase error term ϵ is added to Q component as follows:

$$I(t) = (A + \sin(\omega_m t))\cos(\omega_c t) \tag{5.6}$$

$$Q(t) = (A + \sin(\omega_m t))\sin(\omega_c t + \epsilon) \tag{5.7}$$

The modulating signal can be extracted by calculating the envelope of the I/Q components

using the following formula:

$$O(t) = \sqrt{I^2 + Q^2} \tag{5.8}$$

Where

$$I(t)^2 = (A + \sin(\omega_m t))^2 \cos^2(\omega_c t) \tag{5.9}$$

$$\begin{aligned} Q(t)^2 &= (A + \sin(\omega_m t))^2 (\sin(\omega_c t)\cos(\epsilon) + \cos(\omega_c t)\sin(\epsilon))^2 \\ &= (A + \sin(\omega_m t))^2 (\sin^2(\omega_c t)\cos^2(\epsilon) + \cos^2(\omega_c t)\sin^2(\epsilon) + \frac{1}{2}\sin(2\omega_c t)\sin(2\epsilon)) \end{aligned} \tag{5.10}$$

For $\epsilon <<$, the following approximations can be used in (5.10),

$$\sin(\epsilon) = \epsilon, \sin^2(\epsilon) = 0, \cos^2(\epsilon) = 1 \tag{5.11}$$

Then we have,

$$Q(t) = (A + \sin(\omega_m t))^2 (\sin^2(\omega_c t) + \epsilon \sin(2\omega_c t)) \tag{5.12}$$

Substituting (5.9) and (5.12) into (5.8) we get,

$$\begin{aligned} O(t) &= \sqrt{(A + \sin(\omega_m t))^2 (\cos^2(\omega_c t) + \sin^2(\omega_c t) + \epsilon \sin(2\omega_c t))} \\ &= (A + \sin(\omega_m t))\sqrt{1 + \epsilon \sin(2\omega_c t)} \end{aligned} \tag{5.13}$$

Assuming ϵ is small enough, (5.13) can be further reduced to:

$$\begin{aligned} O(t) &= (A + \sin(\omega_m t))(1 + \frac{\epsilon}{2}\sin(2\omega_c t)) \\ &= (A + \sin(\omega_m t)) + (A + \sin(\omega_m t))\frac{\epsilon}{2}\sin(2\omega_c t) \end{aligned} \tag{5.14}$$

Eqn (5.14) shows that the high frequency ripple occurs at twice the carrier frequency and thus can be easily removed by a simple low pass filter. The amplitude of the ripple is proportional to the phase error ϵ, as well as the amplitude of the modulating signal. This can be clearly seen in Figure 5.21 where the amplitude of the ripple is greater for the positive half cycle than in the negative one.

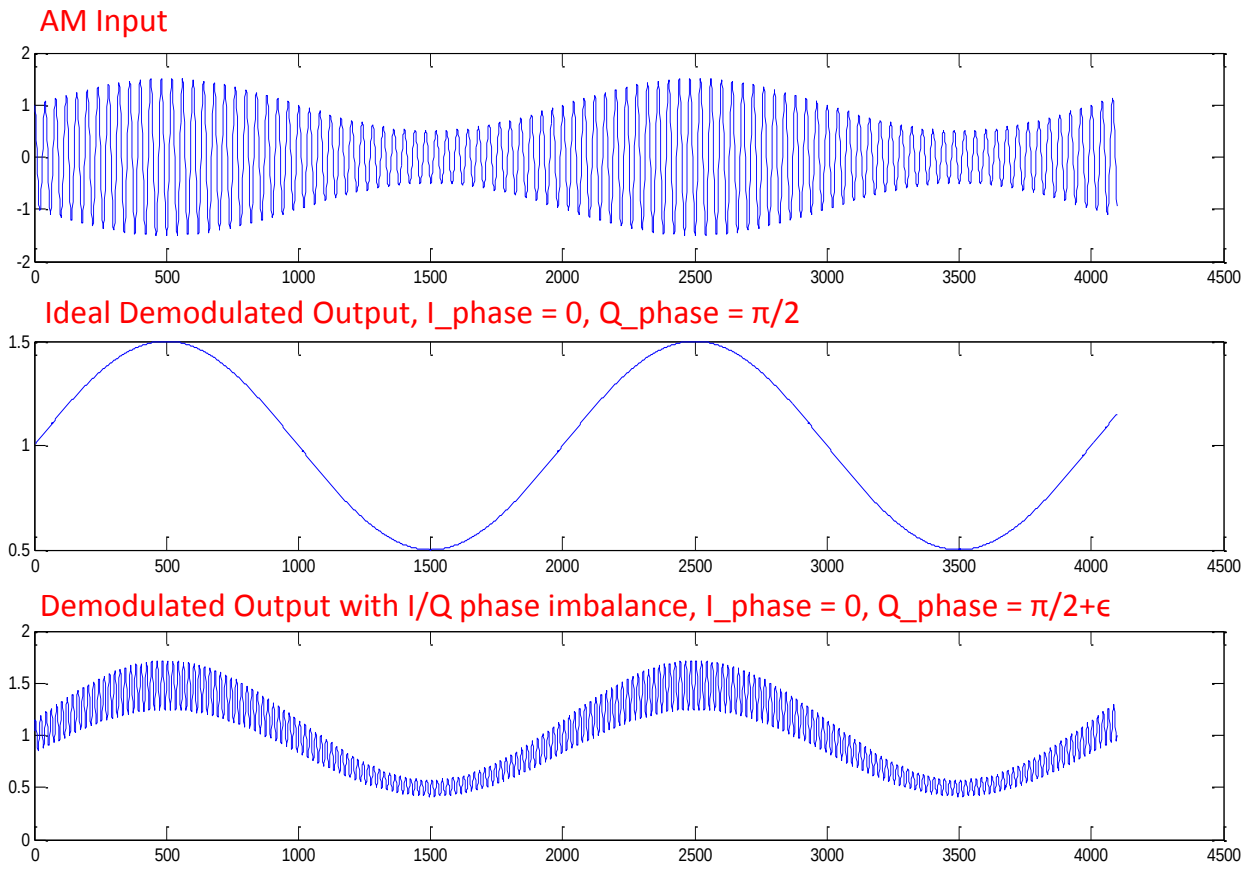


Figure 5.21: Matlab Simulation showing the effect of I/Q phase imbalance on AM Demodulation Output

FM Waveform

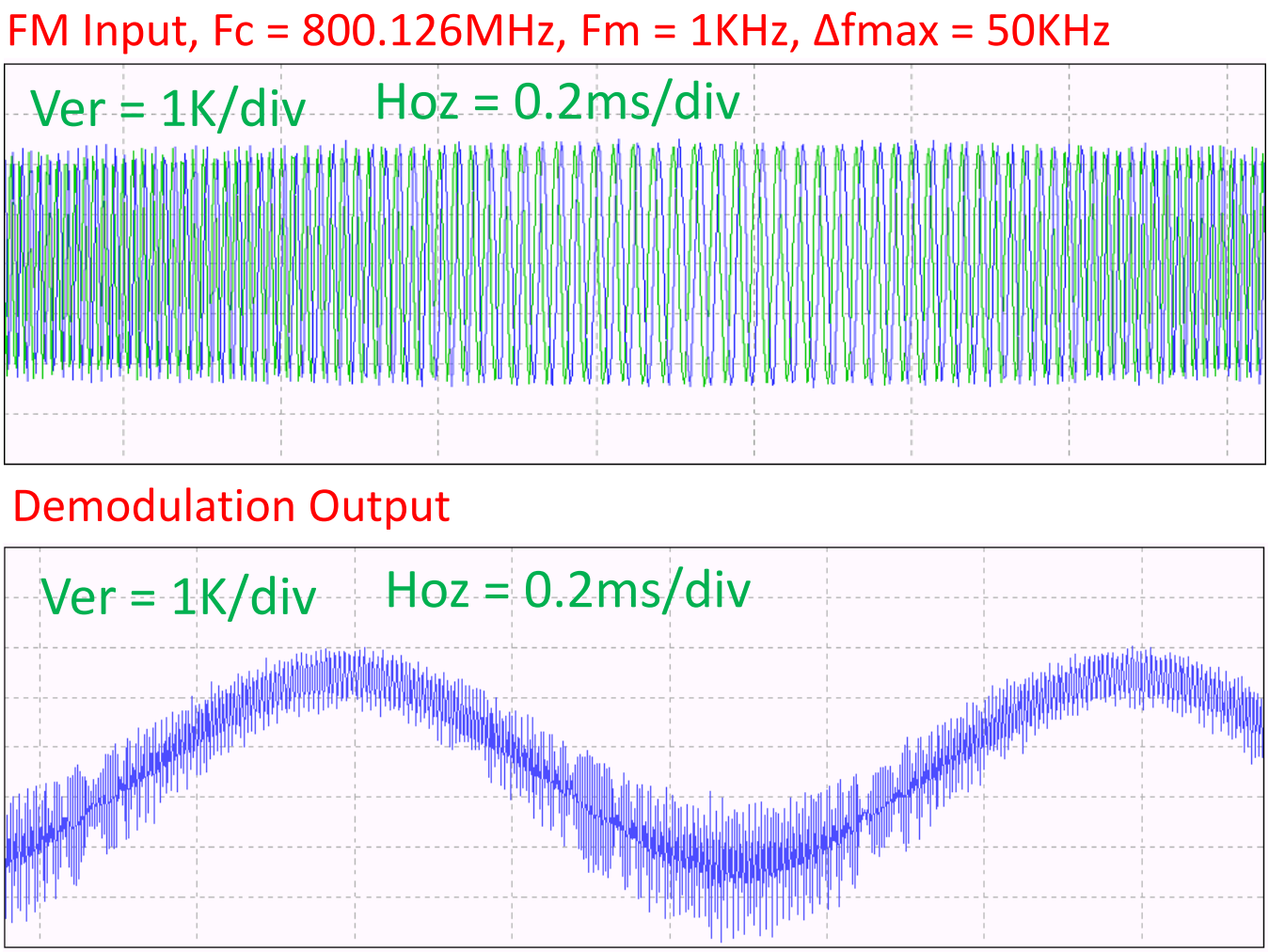


Figure 5.22: Input FM Signal and Demodulated Output

The first impression when looking at Figure 5.22 is that the output demodulated signal has a relatively low SNR for a direct input signal. However, the reality is that the noisy appearance of the demodulated output is an artifact of the GNU radio scope plotting function. This happens when the sampling rate of the input signal is high and a large value is chosen for the time/div scale to view a low frequency signal component. In the test setup, the sampling rate is 2MS/s and a large time/div was chosen to view 1 cycle of the modulating signal which is at 1KHz, i.e. a scale factor of 2000. Figure 5.23 shows proof that this is the case by zooming

in on portions of the signal at the large time/div setting, where the artifact shows up, and a view of the same signal portions at a smaller value for the time/div.

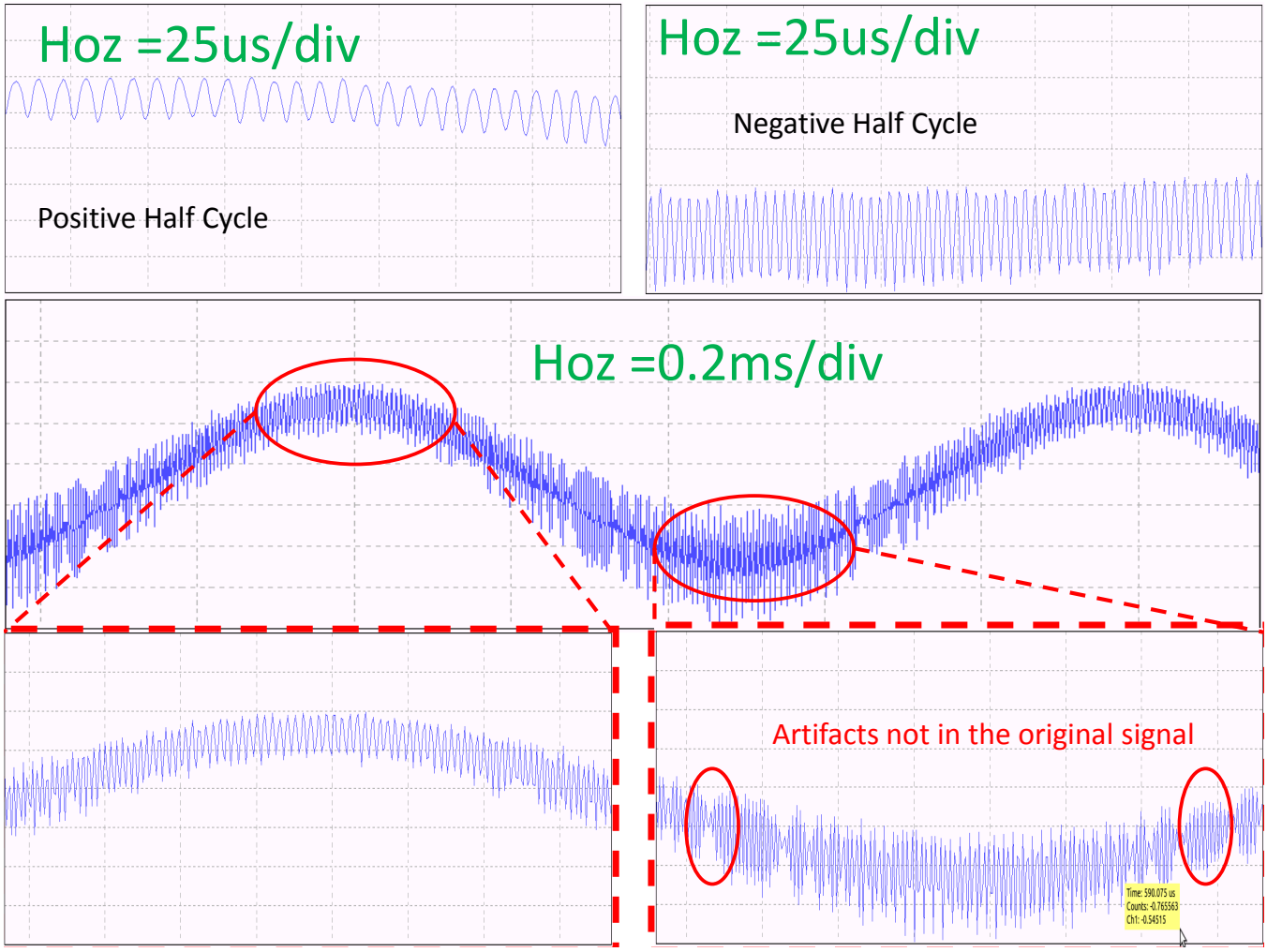


Figure 5.23: GNU Radio Plotting Artifacts for a large time/div Setting.

The upper plots show the actual detail of the signal, while the lower plot shows a zoom of the compressed scope plot obtained by the large time/div setting. The upper plots are much smoother while the lower plots have a more spiky appearance, which implies that the GNU radio scope plot is throwing away samples for large time/div settings. Second, am modulation-like artifacts can be seen in the lower plots which are also not present in the original signal.

Analysis of I/Q Phase Imbalance on FM Demodulation Performance

A FM modulated signal is represented by the following equation

$$f(t) = Ae^{j\omega_c t + \theta(t)} \tag{5.15}$$

where $\theta(t)$ is the instantaneous phase,

$$\theta(t) = \int_{-\infty}^{t} m(t) \tag{5.16}$$

and $m(t)$ is the frequency modulating signal.

Expanding in complex I/Q form:

$$f(t) = \cos(\omega_c t + \theta(t)) + j\sin(\omega_c t + \theta(t)) \tag{5.17}$$

Where:

$$I(t) = \cos(\omega_c t + \theta(t)) \tag{5.18}$$

$$Q(t) = \sin(\omega_c t + \theta(t)) \tag{5.19}$$

Introducing the phase error term ϵ we get,

$$I(t) = \cos(\omega_c t + \theta(t)) \tag{5.20}$$

$$Q(t) = \sin(\omega_c t + \theta(t) + \epsilon) \tag{5.21}$$

To demodulate an FM signal, the phase signal is first extracted and then differentiated as follows,

$$\theta(t) = \arctan(\frac{Q(t)}{I(t)}) \tag{5.22}$$

then applying differentiation,

$$O(t) = \frac{d}{dt}\arctan(\frac{Q(t)}{I(t)}) \tag{5.23}$$

By expanding the derivative operation we get,

$$\begin{aligned} O(t) &= \frac{I^2}{I^2+Q^2}\frac{I\acute{Q} - Q\acute{I}}{I^2} \\ &= \frac{I\acute{Q} - Q\acute{I}}{I^2+Q^2} \end{aligned} \tag{5.24}$$

Now we compute the following terms,

$$I^2 = \cos^2(\omega_c t + \theta(t)) \tag{5.25}$$

$$Q^2 = \sin^2(\omega_c t + \theta(t))\cos^2(\epsilon) + \cos^2(\omega_c t + \theta(t))\sin^2(\epsilon) + \frac{1}{2}\sin(2\epsilon)cos(2\omega_c t + 2\theta) \tag{5.26}$$

$$\acute{I} = -\sin(\omega_c t + \theta(t))(\omega_c + \theta'(t)) \tag{5.27}$$

$$\acute{Q} = \cos(\omega_c t + \theta(t))(\omega_c + \theta'(t))\cos(\epsilon) - \sin(\omega_c t + \theta(t))(\omega_c + \theta'(t))\sin(\epsilon) \tag{5.28}$$

$$I\acute{Q} = \cos^2(\omega_c t + \theta(t))(\omega_c + \theta'(t))\cos(\epsilon) - \frac{1}{2}\sin(2\omega_c t + 2\theta(t))(\omega_c + \theta'(t))\sin(\epsilon) \quad (5.29)$$

$$Q\acute{I} = -\sin^2(\omega_c t + \theta(t))(\omega_c + \theta'(t))\cos(\epsilon) - \frac{1}{2}\sin(2\omega_c t + 2\theta(t))(\omega_c + \theta'(t))\sin(\epsilon) \quad (5.30)$$

Substituting (5.29) and (5.30) into the numerator of (5.24) we get,

$$\begin{aligned} I\acute{Q} - Q\acute{I} &= \cos^2(\omega_c t + \theta(t))\cos(\epsilon)(\omega_c + \theta'(t)) + \sin^2(\omega_c t + \theta(t))\cos(\epsilon)(\omega_c + \theta'(t)) \\ &= \cos(\epsilon)(\omega_c + \theta'(t)) \end{aligned} \quad (5.31)$$

and (5.25) and (5.26) into the denominator of (5.24) we get

$$I^2+Q^2 = \cos^2(\omega_c t+\theta(t))+\sin^2(\omega_c t+\theta(t))\cos^2(\epsilon)+\cos^2(\omega_c t+\theta(t))\sin^2(\epsilon)+\frac{1}{2}\sin(2\epsilon)\cos(2\omega_c t+2\theta(t)) \quad (5.32)$$

Using Taylor series approximation for $cos(\epsilon)$ and $\sin(\epsilon)$ in 5.33 and keeping first order terms we get,

$$\begin{aligned} I^2 + Q^2 &= \cos^2(\omega_c t + \theta(t)) + \sin^2(\omega_c t + \theta(t)) + \epsilon\cos(2\omega_c t + 2\theta(t)) \\ &= 1 + \epsilon\cos(2\omega_c t + 2\theta(t)) \end{aligned} \quad (5.33)$$

Now we substitute both (5.33) and (5.31) into (5.24) we get,

$$O(t) = \frac{\cos(\epsilon)(\omega_c + \theta'(t))}{1 + \epsilon\cos(2\omega_c t + 2\theta(t))} \tag{5.34}$$

Assuming ϵ is small enough, we can apply the approximation $\frac{1}{1+x} \approx 1 - x$ to (5.34) and replace $\cos(\epsilon)$ with 1, then we get,

$$\begin{aligned} O(t) &= (\omega_c + \theta'(t))(1 - \epsilon\cos(2\omega_c t + 2\theta(t)) \\ &= (\omega_c + \theta'(t)) - (\omega_c + \theta'(t))\epsilon\cos(2\omega_c t + 2\theta(t)) \end{aligned} \tag{5.35}$$

A result similar to the case of AM demodulation is obtained which predicts the ripple component in the observed measurements. Interestingly, it can be seen (5.35) and (5.14) are almost identical. In fact, by comparing the ripple components in both cases, it can be found that the ripple term in 5.35 is exactly the derivative of the ripple term in 5.14.

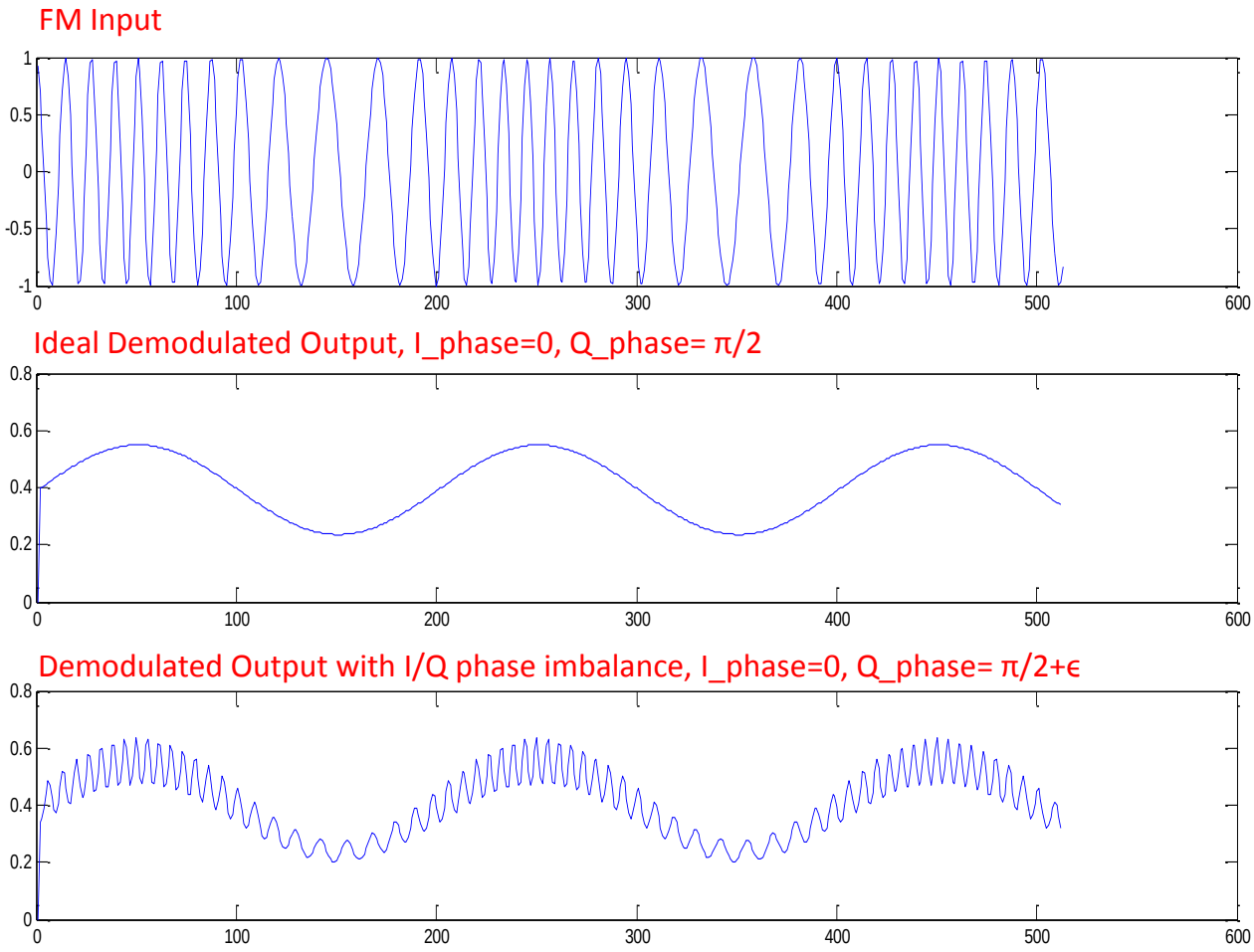


Figure 5.24: Matlab Simulation showing the effect of I/Q phase imbalance on FM Demodulation Output

Chapter 6

Conclusion and Future Work

6.1 Conclusion

One of the unique features of an SDR is its ability to modify its waveform processing functionality during operation either manually or autonomously based on decisions from a cognitive engine. The ideal case is when all components across the different stages of the radio chain are allowed to be programmed or reconfigured. The three typical stages that exist in an SDR are:

- RF Processing Block: This consists of the antenna and all related components such as switches, duplexers, and filters that operate directly on the rf signal.
- Post-RF Analog Processing Block: This consists of the set of components that provide the transition from the rf frequency to low or baseband frequency for additional processing and finally digitization. The components in this subsystem include baseband filters, gain blocks, signal level detectors, ADCs and DACs.
- Digital Processing Block: This stage consists mainly of signal processing block that perform complex operations exploiting the flexibility and simplicity offered by the digital domain.

Given that SDRs are expected to change their function as part of their run-time operation, a consequence of this is that the reconfiguration procedure must complete within a certain time frame to satisfy run-time constraints. Existing SDR platforms that have been examined in this thesis are limited in their run-time programmability to software components that run on a GPP. On the other hand, FPGA-based platforms can only host pipelines of static signal processing chains that perform a fixed function throughout the entire runtime. At best, FPGA components can be designed with input ports to accept parameters that can modify their behaviour to a certain extent, such as loading new FIR filter coefficients. Such parametrized components usually occupy more resources and can only perform a variation of a certain predefined function.

The PicoRF presented in this work provides a very high degree of programmability which is very close to the ideal target. Except for the RF stage, the following two stages are highly programmable in run-time. This has been enabled by two novel components/features that cannot be found in other existing SDR platforms:

- RFIC5: This represents the Post-RF processing stage. With its large number of programmable parameters and state-of-the-art technology, the RFIC5 enables a high degree of flexibility and superior performance compared to RF daughter-boards used for the equivalent function in other platforms. It can operate over a wide frequency range [100MHz-2.4GHz], in addition to integrating an ADC and DAC pair in the same compact package.

- High-speed FPGA Configuration Channel: Supported by a 2Gbps PCIe link to the PC host memory storage, a wide range of waveforms can be hosted on a moderate sized FPGA. Given the short reconfiguration times, switching from one waveform to another becomes possible during run-time. This can enable innovative radio modes of operation that require switching waveforms within the same communication session, such as using a different waveform per packet.

As for the RF stage, the PicoRF provides an I2C bus port to communicate with a highly flexible RF filter board that will enhance the RF performance. However, using this additional board will hinder the portability of the PicoRF setup due to its additional size.

6.2 Future Work

6.2.1 Incomplete Tasks

Due to the experimental nature of the RFIC5, the provided documentation is an initial draft version that does not contain sufficient detail for certain features, and in some instances provided inconsistent information. This has several times required lengthy trial and error experimentation to determine the exact register values needed for a desired behaviour. Due to insufficient time, a fully functioning interface to a number of the blocks in the RFIC5 have not yet been completed, which are listed below. The issues that need to be addressed for each block have been narrowed down to specific points that will simplify the task for whoever continues this work in the future.

(a) TX DAC: Although functioning, accurate control of the DAC sample rate is not yet achieved due to insufficient programming information in the datasheet. It is either possible to get additional information from the manufacturer or go through the following possible trial and error procedures.

- The TX output port can be connected in feedback to the RX input port of the RFIC while ensuring that sufficient attenuation is used to set the appropriate power level. This way the output of the DAC in the time domain can be viewed using the GNU radio scope, and by comparing the observed sample rate to the programmed values in the DAC control registers the problem can be pinpointed.
- The other option is to direct the TX DAC output to one of the external test pins of the RFIC5, this way the DAC output can be monitored using an external scope.

It should be mentioned that a large number of tests for the RFIC5 functions have been greatly simplified on the PicoRF platform due to the existence of a flexible high-level control API. It is very likely that some of our test results concerning the dynamic programming of the RFIC registers in the middle of operation have not been discovered by the manufacturers due to the simplified testing platform that they have been using, such as the RFIC5 development platform that they have supplied with this project. An example of this is the programming of the RX VGA gain control. In certain instances, the value written into the register would produce the desired gain but when the actual contents of the register are readback they do not match the expected value.

(b) QUAD Gens: In the results section in ch5, it was evident that the SBR(Side-band Rejection) performance of the RFIC5 was not very impressive, particularly due to the I/Q phase imbalance. During experimentation, it was observed that the values programmed into the QUAD Gen registers had an important impact on this and was slightly hinted in the documentation of the previous RFIC chip, the RFIC4. In addition, the set of optimal programming values varied according to the operating frequency range. Such values have not been provided in the manufacturer's data sheet. Unfortunately, the exact meaning of the values programmed into the registers are not known and appear to be random. This makes the trial and error procedure blind and undirected, potentially requiring to test all possible values, which is an extremely tedious process.

Apart from the above, the API provides well tested and operating functions to control the different blocks of the RFIC.

6.2.2 Innovative Radio Applications

Any-wave Autonomous Radio

By incorporating a cognitive engine into the PicoRF that controls the radio reconfiguration process, a fully autonomous radio can be realized. A modulation classifier will be initially loaded into the pr slots and will attempt to determine the characteristics of a sensed signal. By relying on run-time reconfiguration, the classifier capabilities can be expanded greatly by loading in components that implement different classification algorithms. After the signal characteristics have been determined, the waveform classifier is unloaded and the demodulator components are loaded in its place.

Bibliography

[1] Pico e-17 [online], June 2012. http://www.picocomputing.com.

[2] Windriver [online], June 2012. http://www.jungo.com.

[3] A.A. Abidi. Direct-conversion radio transceivers for digital communications. *Solid-State Circuits, IEEE Journal of*, 30(12):1399–1410, 1995.

[4] K. Amiri, Y. 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*, pages 53–54. IEEE, 2007.

[5] P. Athanas, J. Bowen, T. Dunham, C. Patterson, J. Rice, M. Shelburne, J. Suris, M. Bucciero, and J. Graf. Wires on demand: Run-time communication synthesis for reconfigurable computing. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 513–516. IEEE, 2007.

[6] R. Bagheri, A. Mirzaei, M.E. Heidari, S. Chehrazi, M. Lee, M. Mikhemar, W.K. Tang, and A.A. Abidi. Software-defined radio receiver: dream to reality. *Communications Magazine, IEEE*, 44(8):111–118, 2006.

[7] E.E. Bautista, B. Bastani, and J. Heck. A high iip2 downconversion mixer using dynamic matching. *Solid-State Circuits, IEEE Journal of*, 35(12):1934–1941, 2000.

[8] C.W. Bostian. Trade study of implementation of software defined radio (sdr): Fundamental limitations and future prospects. Technical report, DTIC Document, 2008.

[9] G. Cafaro, T. Gradishar, J. Heck, S. Machan, G. Nagaraj, S. Olson, R. Salvi, B. Stengel, and B. Ziemer. A 100 mhz 2.5 ghz direct conversion cmos transceiver for sdr applications. In *Radio frequency integrated circuits (RFIC) symposium, 2007 IEEE*, pages 189–192. IEEE, 2007.

[10] J.P. Delahaye, C. Moy, P. Leray, and J. Palicot. Managing dynamic partial reconfiguration on heterogeneous sdr platforms. In *SDR Forum Technical Conference*, volume 5, 2005.

[11] Linda E. Doyle. *Essentials of Cognitive Radio*. Cambridge University Press, 2009.

[12] M. Duarte, P. Murphy, C. Hunter, S. Gupta, and A. Sabharwal. Warplab: Multi-node prototyping with real wireless data.

[13] T. Frangieh, A. Chandrasekharan, S. Rajagopalan, Y. Iskander, S. Craven, and C. Patterson. Patis: using partial configuration to improve static fpga design productivity. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

[14] S.A. Guccione, D. Levi, and P. Sundararajan. Jbits: A java-based interface for reconfigurable computing. In *2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, pages 1–9, 1999.

[15] F.A. Hamza. The usrp under 1.5 x magnifying lens! *GNU Radio community*, 2008.

[16] S.M.S. Hasan, R. Nealy, T.J. Brisebois, T.R. Newman, T. Bose, and J.H. Reed. Wideband rf front end design considerations for a flexible white space software defined radio. In *Radio and Wireless Symposium (RWS), 2010 IEEE*, pages 484–487. IEEE, 2010.

[17] C.R. Irick. *Enhancing GNU Radio for Hardware Accelerated Radio Design*. PhD thesis, Virginia Polytechnic Institute and State University, 2010.

[18] M.J. Koop, W. Huang, K. Gopalakrishnan, and D.K. Panda. Performance analysis and evaluation of pcie 2.0 and quad-data rate infiniband. In *High Performance Interconnects, 2008. HOTI'08. 16th IEEE Symposium on*, pages 85–92. IEEE, 2008.

[19] D. Lim and M. Peattie. Two flows for partial reconfiguration: Module based or small bit manipulations. *Application Note XAPP*, 290, 2002.

[20] D. Lim and M. Peattie. Two flows for partial reconfiguration: Module based or small bit manipulations. *Application Note XAPP*, 290, 2002.

[21] J.H. Mikkelsen. Front-end architectures for cmos radio receivers, 1996.

[22] G.J. Minden, J.B. Evans, L. Searl, D. DePardo, V.R. Petty, R. Rajbanshi, T. Newman, Q. Chen, F. Weidling, J. Guffey, et al. Kuar: A flexible software-defined radio development platform. In *New Frontiers in Dynamic Spectrum Access Networks, 2007. DySPAN 2007. 2nd IEEE International Symposium on*, pages 428–439. IEEE, 2007.

[23] P. Murphy, A. Sabharwal, and B. Aazhang. Design of warp: A wireless open-access research platform. In *European Signal Processing Conference*, pages 1804–1824, 2006.

[24] J.H. Reed. *Software radio: a modern approach to radio engineering*. Prentice Hall Professional, 2002.

[25] A. Rubini and J. Corbet. *Linux device drivers*. O'Reilly Media, 2001.

[26] T. Schmid, O. Sekkat, and M.B. Srivastava. An experimental study of network performance impact of increased latency in software defined radios. *WiNETCH07*, 2007.

[27] PCI SIG. Pci express base specifications revision 1.0 a. *PCI SIG*, 2003.

[28] J. Suris, C. Patterson, and P. Athanas. An efficient run-time router for connecting modules in fpgas. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 125–130. IEEE, 2008.

[29] 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. *Communications of the ACM*, 54(1):99–107, 2011.

[30] A. Tavaragiri. *A Management Paradigm for FPGA Design Flow Acceleration*. PhD thesis, Virginia Polytechnic Institute and State University, 2011.

[31] M. Vidojkovic. Configurable circuits and their impact on multi-standard rf front-end architecture, 2011.

[32] A.H. Wilen, J.P. Schade, and Ron. Thornburg. *Introduction to PCI Express*. Intel, 2003.

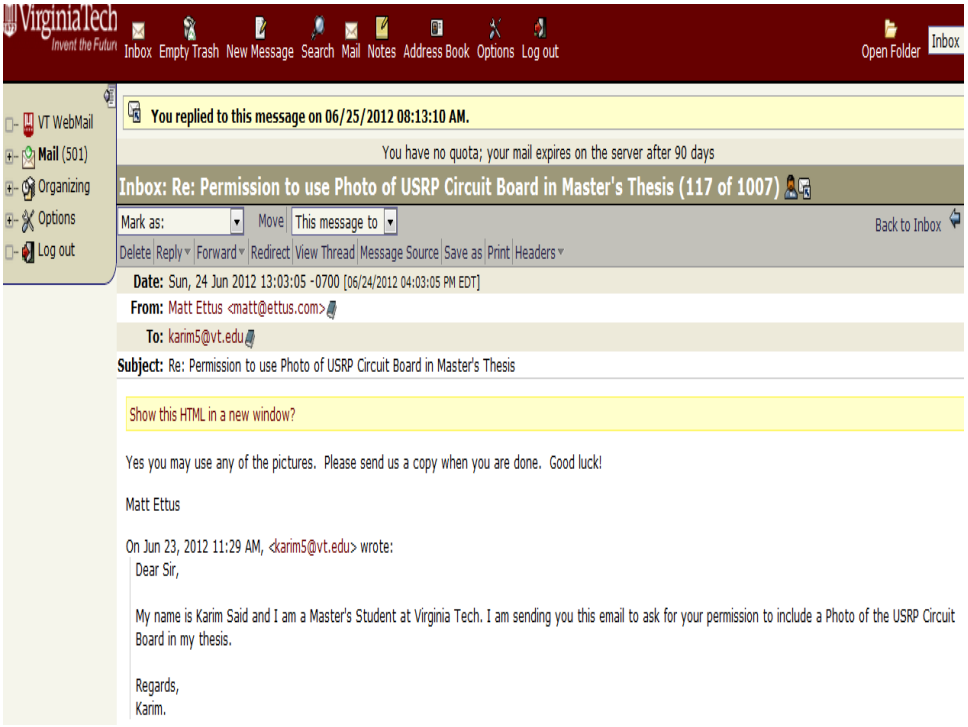
[33] J. Wiltgen and J. Ayer. Bus master performance demonstration reference design for the xilinx endpoint pci express solutions. 2008.

Appendix A

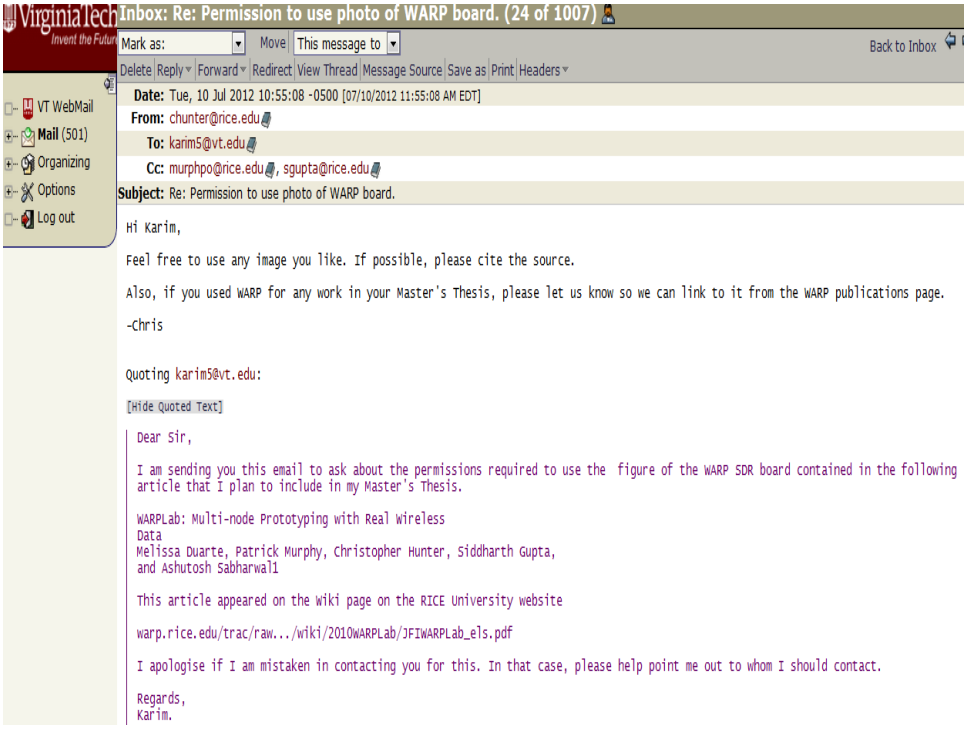
Reprinting Copyrighted Material

Permissions

Permission from Matt Ettus to use images of the USRP board



Permission from Christopher Hunter to use images of the WARP board



Virginia Tech
Invent the Future

VT WebMail
Mail (501)
Organizing
Options
Log out

Inbox: Re: Permission to use photo of WARP board. (24 of 1007)

Mark as: Move This message to Back to Inbox

Delete Reply Forward Redirect View Thread Message Source Save as Print Headers

Date: Tue, 10 Jul 2012 10:55:08 -0500 [07/10/2012 11:55:08 AM EDT]
From: chunter@rice.edu
To: karim5@vt.edu
Cc: murphpo@rice.edu, sgupta@rice.edu
Subject: Re: Permission to use photo of WARP board.

Hi Karim,

Feel free to use any image you like. If possible, please cite the source.

Also, if you used WARP for any work in your Master's Thesis, please let us know so we can link to it from the WARP publications page.

-Chris

Quoting karim5@vt.edu:

[Hide Quoted Text]

> Dear Sir,
>
> I am sending you this email to ask about the permissions required to use the figure of the WARP SDR board contained in the following article that I plan to include in my Master's Thesis.
>
> WARPLab: Multi-node Prototyping with Real Wireless
> Data
> Melissa Duarte, Patrick Murphy, Christopher Hunter, Siddharth Gupta,
> and Ashutosh Sabharwal1
>
> This article appeared on the Wiki page on the RICE University website
>
> warp.rice.edu/trac/raw.../wiki/2010WARPLab/JFIWARPLab_els.pdf
>
> I apologise if I am mistaken in contacting you for this. In that case, please help point me out to whom I should contact.
>
> Regards,
> Karim.

Permission for reusing any IEEE material in Thesis/Dissertation documents

Copyright Clearance Center RightsLink®

Home | Account Info | Help

IEEE Requesting permission to reuse content from an IEEE publication

Title:	KUAR: A Flexible Software-Defined Radio Development Platform
Conference Proceedings:	New Frontiers in Dynamic Spectrum Access Networks, 2007. DySPAN 2007. 2nd IEEE International Symposium on
Author:	Minden, G.J.; Evans, J.B.; Searl, L.; DePardo, D.; Petty, V.R.; Rajbanshi, R.; Newman, T.; Chen, Q.; Weidling, F.; Guffey, J.; Datla, D.; Barker, B.; Peck, M.; Cordill, B.; Wyglinski, A.M.; Agah, A.
Publisher:	IEEE
Date:	17-20 April 2007

Copyright © 2007, IEEE

Logged in as:
Karim Said
Virginia Tech
LOGOUT

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.