A Physical Layer Implementation of Reconfigurable Radio

Nikhil S. Bhatia

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

Dr. Peter M. Athanas, Chair
Dr. Mark T. Jones
Dr. Jeffrey H. Reed

December 3rd, 2004
Bradley Department of Electrical and Computer Engineering
Blacksburg, Virginia

Keywords: Reconfigurable Radio, FPGA, Software Radio,
System Generator, Partial Reconfiguration

A Physical Layer Implementation of Reconfigurable Radio

Nikhil S. Bhatia

(ABSTRACT)

The next generation of wireless communications will demand the use of software radio technology as the basic architecture to support multi-standard, multi-mode and future-proof radio designs. Software-defined radios are configurable devices in which the physical layer can be reprogrammed to support various standards. Field programmable architectures provide a suitable platform to achieve such run-time reconfigurations of the physical layer of the radio. This thesis explores the use of FPGAs in the design of reconfigurable radios. The results presented here demonstrate how FPGAs can be used to provide the flexibility, performance, efficiency and better resource utilization while meeting the speed and area constraints set by a particular design. The partial reconfiguration feature available in the state-of-the art FPGAs has been exploited to implement the baseband physical layer of reconfigurable radio which can be altered to support various modulations schemes for different wireless standards. The design flow for partial reconfiguration along with the implementation results on two different FPGA platforms is presented. The experiments presented in this thesis make use of System Generator for DSP, a productivity tool from Xilinx, to design and to simulate system-level models in a MATLAB/Simulink environment, and to obtain timing and resource utilization results before implementing the design on actual hardware.

Acknowledgements

I would like to express my sincere gratitude to Dr. Peter Athanas for being my advisor and for providing me with the opportunity to conduct research in the field of my interest. His continual support and guidance throughout my research work in Configurable Computing Laboratory at Virginia Tech made this thesis a success. I thank Dr. Mark Jones and Dr. Jeffrey Reed for their support. I appreciate their willingness to serve on my committee.

I am really grateful to all the members of CCM Lab, especially the SecCom team, who helped me at every stage of my research. I am indebted to Dr. Scott Harper, Dr. Jae Park, and Jason Zimmerman who helped me get started on this project. I would also like to thank Ryan Fong, Aditya Chaubal, Jonathan Graf and Deepak Agarwal for their co-operation.

Finally, I dedicate my thesis to my grandfather and parents for their endless support and blessings without which I wouldn't be where I am in life.

Nikhil Bhatia
December 2004

Contents

List of Figures

List of Tables

Chapter 1

1. Introduction

1.1 Motivation

Software radio[1] is an evolving technology that can resolve many of the existing problems arising from various incompatible cellular communications standards throughout the world. The phrase "software radio" refers to the class of reconfigurable radios in which the physical layer behavior can be significantly altered without the change in the hardware. Using a programmable hardware a reconfigurable radio can adapt to many different standards and provide benefits such as global mobility, multi-functionality, compactness and ease of upgrades [1]. The same hardware can be reprogrammed to support different bands of frequencies, different modulation types and different bandwidths resulting in significant reduction in product development times and at the same time offering high power efficiency and speed of operations.

The architecture choice of physical-layer signal processing of a reconfigurable radio is a critical design step. It determines the flexibility, modularity, scalability and performance of the final design [1]. Signal processing algorithms can be implemented using variety of digital hardware such as General Purpose Processors (GPPs), Digital Signal Processors (DSPs), Application Specific Integrated Circuits (ASICs) and Field Programmable Gate

[1] Based on [1], "software radio" has been used as a singular noun throughout the thesis.

Arrays (FPGAs). GPPs are often not the best hardware choice for real-time radio applications because of their high power usage, low level of parallelism and little or no support for specific signal processing tasks as offered by an application specific hardware. While ASICs provide the most optimized hardware choice suited for a particular application, their large market lead time, high initial cost and lack of flexibility make them unsuitable in the design of reconfigurable radios. This leaves the radio designer with two options – a design utilizing an FPGA for custom datapath design, or a DSP using conventional interpreted software. As discussed in [2], FPGAs have an edge over DSP in modular design flows as they provide high degree of parallelism and fast interconnections between different modules of a design. Table 1 shows the comparison between an FPGA chip and a DSP chip [2]. Although FPGAs are in general slower than ASICs and less dense due to the presence of large number of configuration points, the hardware flexibility and reduce design times are the two main reasons for their popularity in the design of reconfigurable radios.

This thesis focuses on the use of FPGAs in the design and implementation of baseband signal processing tasks of a reprogrammable radio. The hardware description language (HDL) describes the gate level functionality of the configurable logic blocks (CLBs), which are the basic building blocks within FPGA. Modular design flow, used for partially reconfiguring the chip, allows a part of the hardware to be dynamically changed while the rest of the part is active. This feature of FPGA design has been used to design a software radio application where the modulation and demodulation schemes are changed dynamically in the transmitter and receiver links to support different standards.

With the availability of the latest design tools, it's now possible to model FPGA-based system-level designs in a MATLAB/*Simulink* graphical environment and to simulate the designs for timing accuracy. This research explores one such design tool from Xilinx – *System Generator* for DSP [3], through which cycle accurate fixed point DSP blocks can be simulated and tested in *Simulink* before implementing on hardware. One notable feature offered by this tool is that it provides fast estimate of FPGA resources required to implement a design on a target device. This helps in estimating the resource usage by the

design on the actual hardware. This thesis compares the simulation results with the hardware implementation results and provides some useful insights using a proof of concept model for a reconfigurable radio design.

	FPGA Chip	**DSP Chip**
Programming Language	Verilog, VHDL	C, Assembly Language
Ease of Software Programming	Fairly Easy. However a Programmer needs to understand hardware architecture before programming.	Easy
Performance	*Can be very fast* if an appropriate architecture is designed.	*Speed is limited* by clock speed of DSP chip.
Reconfigurability	SRAM-type FPGAs can be reconfigured infinite times.	Can be reconfigurable by changing program memory content.
Reconfiguration method	Reconfiguration is done by downloading configuration data to a chip electronically.	Reconfiguration is done by reading a program at a different memory address.
Areas where FPGAs can outperform DSPs, and vice versa	FIR filters, IIR filters, correlators, convolver, FFT etc.	A signal processing program of sequential nature.
Power Consumption	*Can be minimized* if circuit is designed to save power, or power is dynamically controlled.	Even if program A is smaller than program B *power consumption does not change* as long as the number of the memory chips is the same.
Implementation method of MAC	Parallel multipliers/adders or distributed arithmetic.	Repeated operation of MAC function.
Speed of MAC	*Can be fast if parallel algorithm is used.* If the filter is implemented using distributed arithmetic the speed does not depend on number of taps.	*Limited by the speed of MAC* operation of a DSP chip. If a filter is implemented the speed becomes slower as the number of taps increases.
Parallelism	*Can be parallelized* to achieve high performance.	*DSP chip programming is usually sequential* and cannot be parallelized.

Table 1.1: A comparison of FPGA and DSP chips

1.2 Contributions

This thesis presents a partially reconfigurable physical layer design and implementation of a baseband radio. Although not all the blocks of baseband radio found in practical applications are implemented, this thesis provides an adequate proof of concept and insight in the design of reconfigurable radios. The primary contribution of this research can be listed as follows:

- The design of the baseband transmitter and receiver sub-systems in the MATLAB/*Simulink* environment using Xilinx's fixed-point *System Generator* modeling tool. The designs were tested for timing accuracy and resource utilization before actual hardware implementation.

- Use of a soft radio card as a digital transceiver to support bi-directional data transfers between radio frequency (RF) and intermediate frequency (IF). Data rates in the order of 1Mbps were achieved on a wireless channel at 2.04 GHz RF using soft radio card in the loop-back mode.

- The implementation of block encoder/decoder, convolutional interleaver/deinterleaver and various modulator/demodulator techniques on Xilinx Virtex and Virtex-II FPGA platforms. The blocks were implemented for an operational speed of 33 MHz utilizing minimal resources on an FPGA.

- The implementation of modular design flow for partial reconfiguration on Virtex and Virtex-II FPGA platforms to reconfigure a part of FPGA in order to support different modulation and demodulation schemes.

1.3 Organization of Thesis

Following this introductory chapter, Chapter 2 provides the necessary background information on software radios and FPGAs, and presents the idea behind configurable

computing. The theory behind different modulation/demodulation schemes, interleaving and error corrections techniques, which have been tested and implemented as a part of this research effort, are also discussed in this chapter. Chapter 3 surveys the architecture and essential features of two different FPGA platforms used in this research. The architecture of a soft-radio card used to implement the radio IF front end is also described in this chapter. Chapter 4 gives an overview on transmitter and receiver link designs on FPGA in conjunction with the soft-radio transceiver card, and provides a detailed description of the radio design using Xilinx *System Generator* hardware modeling tool. Chapter 4 also provides information on the regular FPGA design flow. Finally it describes in detail the modular design flow for partial reconfiguration of FPGAs. Results for simulations and hardware implementations along with their interpretations are presented in Chapter 5. Chapter 6 concludes the thesis with the summary of the research effort and suggestions for future work in the same direction.

Chapter 2

2. Background

2.1 Overview

This chapter presents an overview of software defined radio (SDR) architecture, discusses the characteristics and benefits of a software defined radio, and describes a model of a practical SDR. This chapter then introduces field programmable gate array (FPGA) devices and describes two Xilinx FPGA architectures used in the thesis. Finally, the chapter concludes with the theory behind baseband processing. This section focuses on the channel coding, interleaving and modulation schemes implemented in the reconfigurable radio.

2.2 Software Defined Radio

Software defined radio technology has rapidly evolved over the past ten years and is gaining more and more interest in the industry because of the enormous advantages it offers over conventional radio architectures [4, 5]. Though there exist no general consensus about the exact definition of SDR, the FCC has proposed to define SDR as a radio that includes a transmitter in which the operating parameters of the transmitter, including the frequency range, modulation type, and maximum radiated or conducted output power can be altered by making a change in software without any hardware

change [6]. According to the SDR forum [7], an industry organization for companies interested in the subject, a software defined radio is a radio in which a significant number of blocks in the transmit and/or receive pathways have their functionality defined by, or implemented by, software.

An SDR differs from an ideal Software Radio (SR), where the digitization of the received signal is at (or very near to) the antenna and all the processing required for the radio is performed by software residing in high-speed digital signal processing elements [8]. The digitization of the signal in SDR is performed at some stage downstream from the antenna, typically after bandpass filtering and low noise amplification, with the reverse process occurring for the transmit digitization. Digital signal processing in flexible and reconfigurable functional blocks defines the characteristics of the radio.

In the context of this thesis, an SDR is a reconfigurable radio that is implemented in hardware (FPGA) and the hardware configuration or functionality is altered by means of a configuration bitstream. A conceptual block diagram of an SDR receiver is shown in Figure 2.1. As indicated in the figure, the analog-to-digital (A/D) conversion is done after intermediate frequency (IF) processing and the radio baseband processing is under software control. This architecture is considered to be an SDR since most of the signal processing is accomplished in software using a configurable processing engine. It is possible for the digital baseband processing to be implemented in the ASICs, in which case the radio would loose the reconfiguration capabilities and would be considered to be a digital radio but not an SDR. If the A/D converter were to be moved closer to the antenna, immediately after the low noise amplifier and wideband filter, the architecture moves closer to the ideal software radio. Reed [1] describes a model of a software radio, where smart antennas and flexible RF hardware are used to provide the additional user programmability to the radio. A smart antenna can provide a gain versus directional characteristic to minimize interference, noise and multipath. Digitizing the signal close to the antenna can improve the flexibility of RF section.

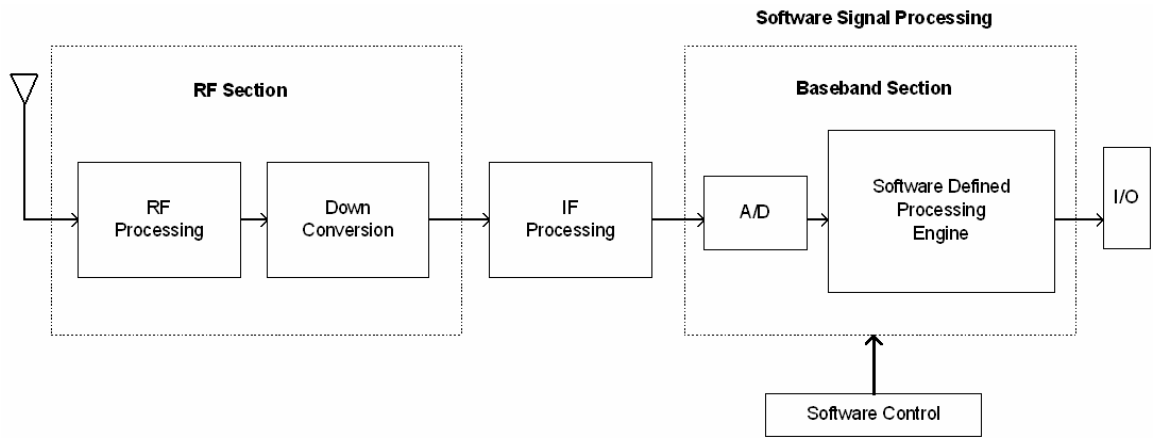

Figure 2.1: Block diagram of a software defined radio

A software defined radio offers a flexible radio architecture that allows upgrading and adding features to the radio that are easily implemented in software. For instance, security features, support for different data-transmission standards or media data types can be easily added just by altering the software. Besides, advanced signal processing techniques involving multiple antennas, beamforming and adaptive modulation techniques can be easily implemented without necessitating major hardware changes in the transceiver.

According to Reed [1], the major factors that are expected to push the wider acceptance of reconfigurable radios are: *Multi-functionality* - because of the radio's reconfiguration capabilities; *Global mobility* - because of the compatibility with most of the communication standards popular worldwide; *Compactness and power efficiency* - because same hardware can be reused to implement various systems; *Ease of manufacture* - because of the closer digitization of the signal to the antenna; and *Ease of upgrade* - because of the on-the-fly air interface capabilities.

2.3 Field Programmable Gate Arrays

The physical layer design of the reconfigurable radio is partitioned and implemented using programmable logic devices known as field programmable gate arrays (FPGAs).

These devices are capable of being reconfigured to implement any desired logic. This is made possible through a matrix of logic elements that can be interconnected in any desired configuration to implement a given algorithm. There are various configurations in which FPGA logic elements can be arranged. This section describes two popular architectures: Virtex and Virtex-II used in XCV1000 and XC2V4000 Xilinx FPGAs.

The Virtex [9] architecture overview is shown in Figure 2.2. The user programmable gate array comprises of two major configurable elements: configurable logic blocks (CLBs), which provide the functional elements for constructing logic, and input/output blocks (IOBs) that provide the interface between the CLBs and the outside world. Several CLBs interconnect through routing matrix that comprises of an array of routing switches. The VersaRing I/O interface provides the routing around the periphery of the device. Other circuits connected to the routing matrix include delay lock loops (DLLs) used for clock distribution throughout the device, dedicated block RAMs (BRAMs) and tri-sate buffers, associated with each CLB, that drive dedicated routing resources. The XCV1000 FPGA chip, based on Virtex architecture, supports one million gates and provides 512 I/O pads.

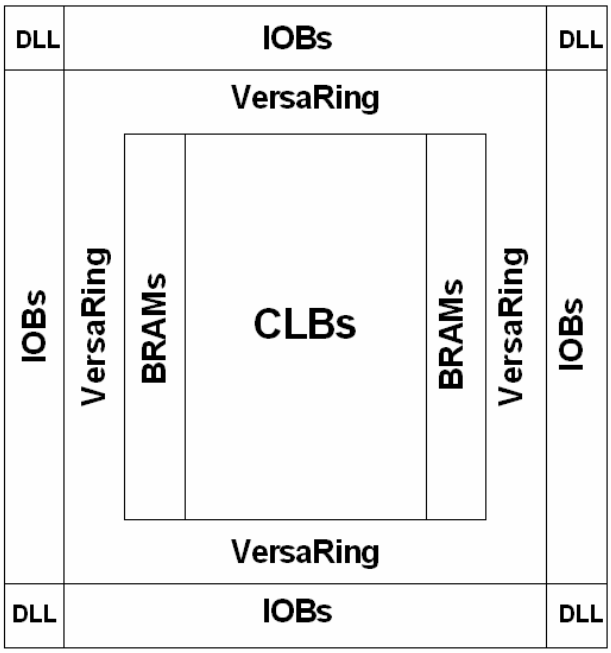

Figure 2.2: Virtex architecture overview

The basic building block of the Virtex CLB is a logic cell. It includes a function generator implemented as a 4-input look-up table (LUT), a carry/control logic and a D flip-flop for data storage. For the XCV1000 Xilinx FPGAs, each CLB contains four logic cells organized in two similar slices as shown in Figure 2.3. The output from the function generator drives both the CLB output and D input of the flip-flop.

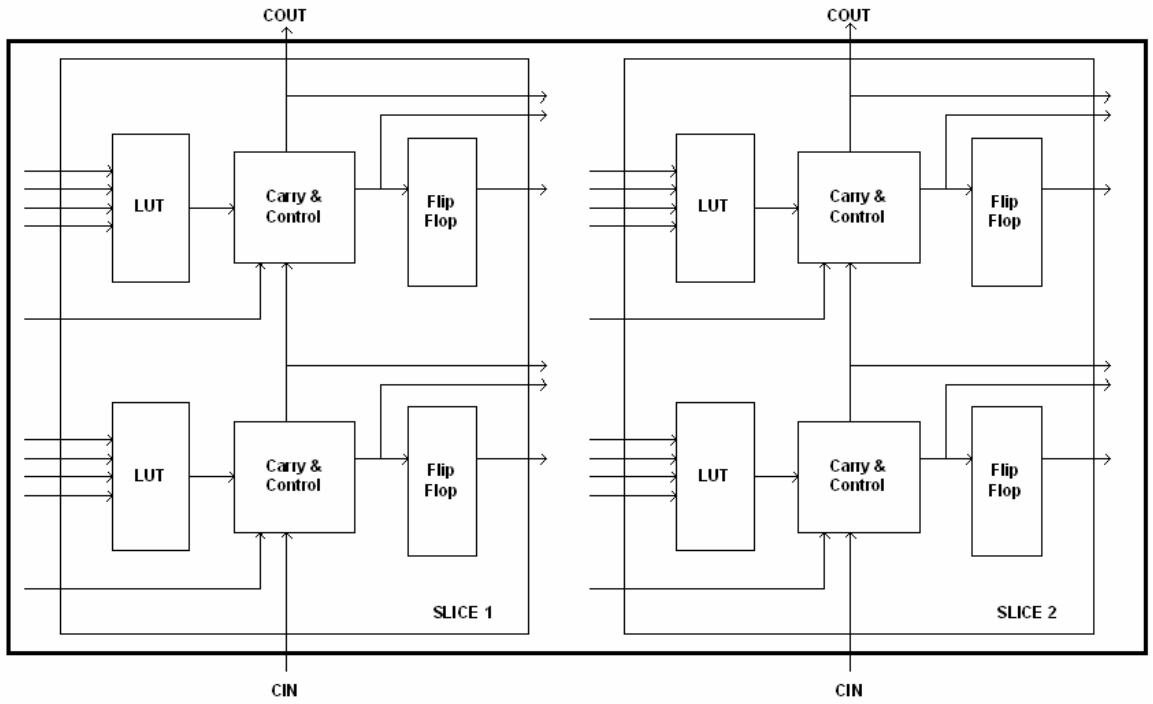

Figure 2.3: 2-Slice Virtex CLB

While the Virtex architecture can process data at rates up to 200 MHz, the Virtex-II architecture can support up to eight million gates at maximum rate of 420 MHz [10]. It is optimized for high speed and lower power consumption than the Virtex architecture. Figure 2.4 shows the Virtex-II architecture. Like Virtex, the basic building blocks of the Virtex-II architecture are CLBs and IOBs. Besides the regular internal configurable logic found in Virtex, this architecture also supports dedicated 18 bit-by-18 bit multipliers, larger block RAMs and Digital clock managers (DCMs), which provide self-calibrating digital solution for clock distribution delay compensation throughout the chip. The IOBs are programmable and can be configured as input, output or bi-directional pads.

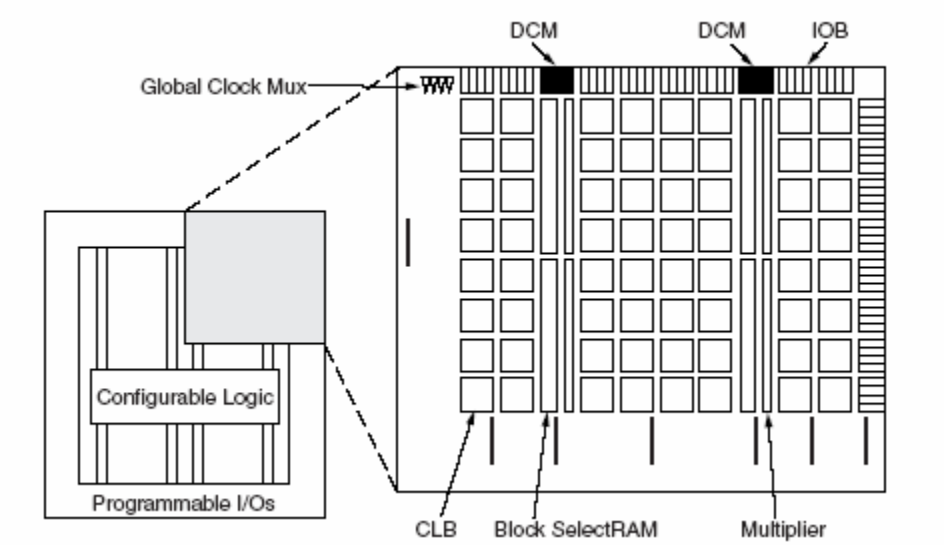

Figure 2.4: Virtex-II architecture overview

XC2V4000 FPGA, based on Virtex-II architecture, supports four million gates and provides 912 I/O pads. Each CLB includes four slices and two tri-state buffers. Each slice is equivalent and contains two 4-input function generators, two storage elements, multiplexer, arithmetic logic gates and carry logic. Figure 2.5 shows the configuration of one such slice. As seen from the figure, each 4-input function generator is programmable 4-input LUT, 16 bit distributed memory or 16 bit variable-tap shift register element.

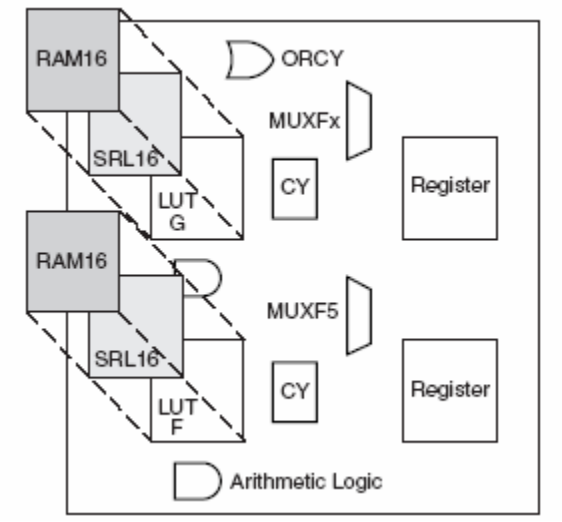

Figure 2.5: Virtex-II slice configuration

FPGAs are commonly used in logic emulation and prototyping in most systems. They are widely used in testing the design of a custom circuit. They are ideal for prototyping since the chips are reusable after the bugs have been fixed of system has been upgraded. Though software simulations can verify the accuracy of a circuit, they do so at a much slower rate than logic emulation circuits using FPGAs. In case of reconfigurable radios, FPGAs have the advantage of adding adaptability and flexibility in the final product. They at times can also help conserve silicon area since one chip can be configured to perform more than one function and the configuration can be changed on-the-fly [1].

2.4 Baseband Processing

Many SDR architectures for reconfigurable platforms have been suggested in the literature and one such paper [11] suggests a layered architecture that makes it possible to incorporate all the features of a software radio while minimizing complexity issues. The design can be broken down into different layers, such as application, network and physical layer, interfaced with each other and performing specific operations to achieve a desired task. The fundamental operations of a physical layer radio are studied and theory behind them is presented in this section.

The aim of the physical layer of a radio is to insure that the data bits obtained from the upper layers are passed to the other end of the channel with minimum loss and distortion so that the useful information can be retrieved from the data bits at the receiver end. This requires the use of basic techniques, namely channel coding, interleaving, and modulation. Channel coding enables the transmitted signal to better with-stand the effects of various channel impairments such as noise, interference and fading. Interleaving is used to nullify the effects of burst errors introduced by the channel by introducing time diversity in the communication system. Modulation is the process of encoding information from a message source in order to achieve required bandwidth efficiency.

Various schemes for achieving above mentions tasks are described in detail in the following sub-sections.

2.4.1 Channel Coding

Channel coding protects the data from errors by selectively introducing redundancies in the transmitted signal. The codes that are used to detect errors are called error detection codes whiles codes that can detect and correct errors are called error correction codes.

Shannon's law [12] for the capacity of a communications channel states that by proper encoding of the message signal, errors introduced by a noisy channel can be reduced to a desired level without sacrificing the rate of the message transfer. This law, as applicable to the gaussian channel, is given by

$$C = B\log_2\left(1+\frac{P}{N_o B}\right) \tag{2.1}$$

where C is the channel capacity (bits per second), B is the transmission bandwidth (Hz), P is the received signal power (Watts), and N_0 is the single-sided noise power density (Watts/Hz). Equation 2.1 gives a theoretical limit for the transmission rate for reliable, error-free data from a transmitter with a given power, over a channel with a given bandwidth, operating in a given noise environment. The aim of the design engineer is to achieve the desired throughput, with the acceptable Bit Error Rate (BER), within the given power constraints and bandwidth constrains for a particular application. Channel coding may help in fulfilling this requirement if the uncoded system is unable to achieve the required BER within these constraints.

Basic types of error correction and detection codes are described, with more emphasis given to linear block codes, which were implemented and tested on FPGA platforms.

Linear Block Codes

Linear block codes are the class of parity check codes that can be characterized by *(n,k)* notation [13], where the *n*-bit codeword is made up of *k* information bits and *(n-k)* redundant parity check bits. The rate or efficiency of a particular block code is *k/n*.,

which represents the ratio of message bits to the number of bits in the codeword. The redundancy is deliberately added to achieve error detection and correction at the receiver end. Figure 2.6 shows the structure of a systematic block code. The block code is termed as a *systematic* block code when the *k* information bits are transmitted contiguously as a block and the parity bits makeup the codeword as another contiguous block.

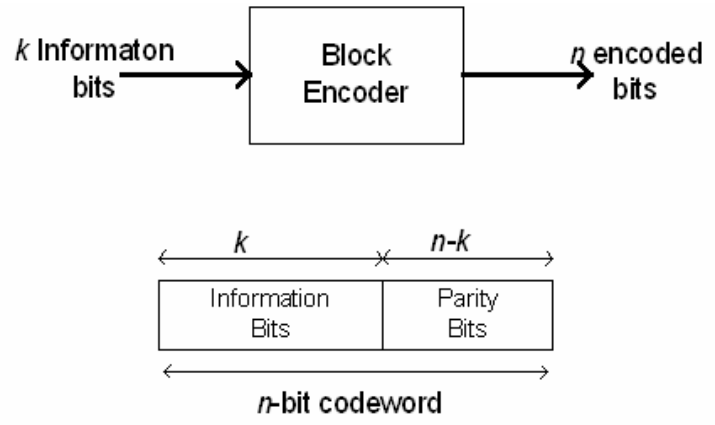

Figure 2.6: (n,k) systematic block code

The mapping of *k* message bits to *n* coded bits is one-to-one to ensure that the encoded original message data can be recovered at the receiver. The corruption of a codeword by channel noise is modeled usually as the additive process [14], shown in Figure 2.7.

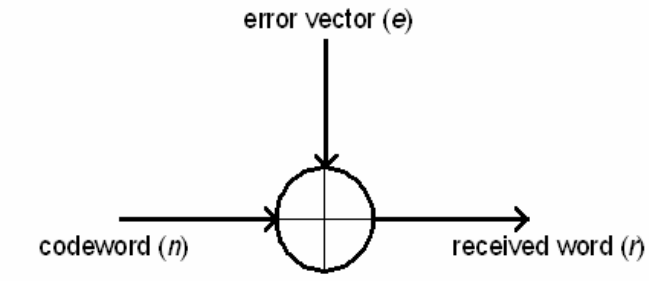

Figure 2.7: Model for additive noisy channel

The channel decoder examines the received word and determines whether or not it is a valid codeword. If the received word is found to be invalid, then it assumes that the

channel has caused one or more errors in the transmitted codeword. The determination of whether the errors are present in the received word is known as error detection. The decoder can either request for the retransmission of the codeword through automatic-repeat-request (ARQ) or can perform forward error correction (FEC) to correct for the errors introduced by the noisy channel. The arithmetic structure of the codeword is used to determine which of the valid code words is most likely to have been sent, given the erroneous received word [14]. The similarity between the received word and a valid codeword is determined by finding the *Hamming distance* between the two words. The Hamming distance between any two words n_1 and n_2 is given by

$$d_H(n_1, n_2) = wt(n_1 - n_2) \tag{2.2}$$

where *wt(n)* denotes the *Hamming weight* of *n* or the number of nonzero positions in *n*. The received word is mapped to one of the valid codewords with which the hamming distance is found to be minimal.

Syndrome decoding is one of the FEC techniques most widely used for decoding linear block codes. It is particularly useful for error correction of Hamming codes, which are the class of binary linear block codes. The parameters for binary Hamming codes are represented usually as a function of integer $m \geq 2$ as follows [13]:

Code length:	$n = 2^m - 1$	(2.3a)
Message length:	$k = 2^m - m - 1$	(2.3b)
Minimum Hamming distance:	$d_{\min} = 3$	(2.3c)

In syndrome decoding, the received word is multiplied with a parity-check matrix *H* to generate *(n-k)* pattern *s* known as *syndrome*. A memory table maps all the possible syndromes with the error vectors for a particular block code. Thus the *syndrome s* is used to generate the error vector *e* for the received word, which is subtracted from the received word *r* to detect the possible transmitted codeword *n*. The syndrome decoding algorithm is shown in Figure 2.8.

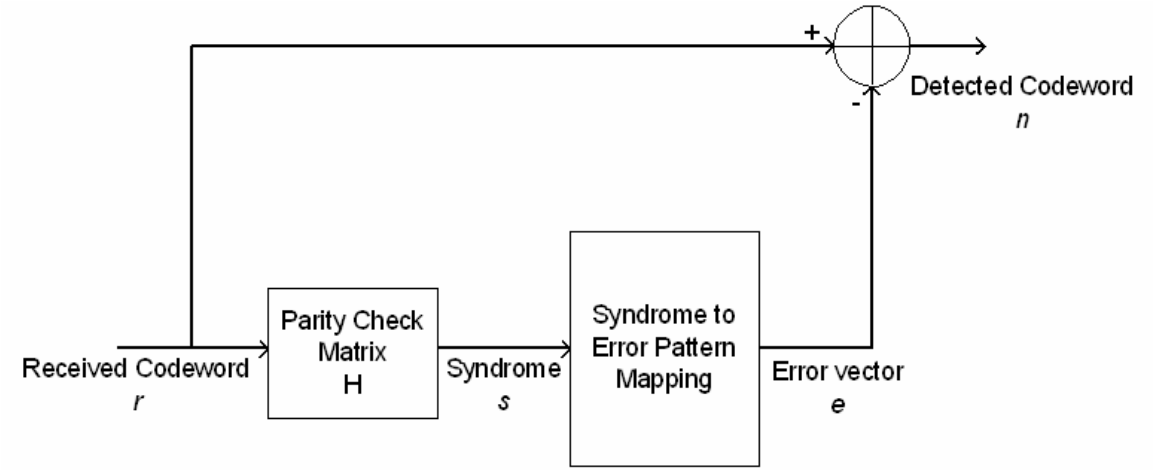

Figure 2.8: Syndrome decoding

Convolutional Codes

Convolutional codes were introduced by Elias [16] in 1955 as an alternative to block codes. They are described by three integers, *n*, *k* and *K*, where the ratio *k/n* is called the code rate and *K* is a parameter known as the constraint length; it represents the number of *k* stages in the encoding shift register. Convolutional codes differ from the block codes in that the encoder contains memory and the output word at any given time depends not only on the m input bits but also on *K* previous inputs. The constraint length *K* determines the capability and complexity of the code.

Figure 2.9 illustrates a simple (2, 1) binary convolutional encoder of constraint length *K*=3. There are *n=2* modulo-2 adder; thus the code rate is *(k/n)* is ½. The choice of the connections between the adders and shift register gives rise to the characteristics of the code. Any chance in the choice of the connection results in a different code. At each input bit time, a bit is shifted into the leftmost stage and the bits in the register are shifted one position to the right. The output switch samples the output of each modulo-2 adder, thus forming the code symbol pair associated with the present input bit. The sampling is repeated for each inputted bit.

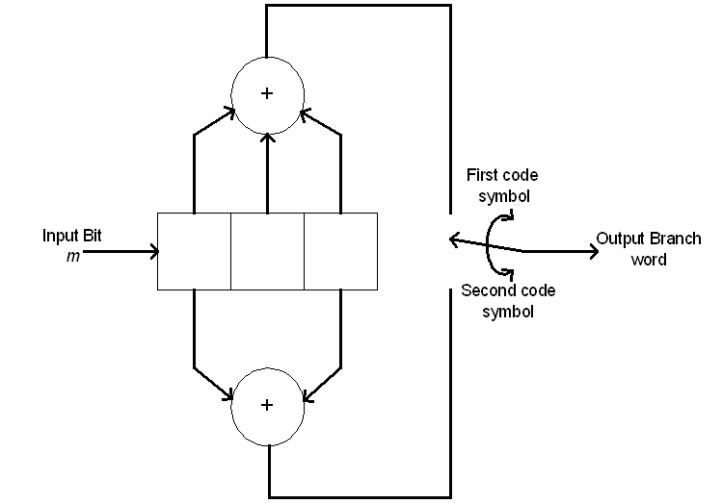

Figure 2.9: Convolutional encoder (rate ½, K=3)

There are three main types of decoding techniques for convolutional codes. They are based on sequential, threshold or Viterbi decoding techniques [15]. The Viterbi technique is by far the most popular. It was invented and analyzed by Viterbi [17] in 1967. The Viterbi decoding algorithm implements a nearest neighbour decoding strategy. It picks the path through the decoding trellis, which assumes the minimum number of errors between the received codeword and decoded codeword. Although the Viterbi algorithm essentially performs maximum likelihood decoding, it reduces the computational load by taking advantage of the special structure in the code trellis.

For many applications Convolutional codes can achieve the required performance for the desired information rate with low implementation complexity of the encoder-decoder compared to block codes. Convolutional codes are widely used in satellite, space and wireless communications.

Trellis Coded Modulation

The basic idea behind Trellis Coded Modulation (TCM) is that in TCM, channel coding and modulation are performed together, where one begins and other ends cannot be easily established. TCM schemes employ redundant non-binary modulation in combination with a finite state encoder which governs the selection of modulation signals to generate coded

signal sequences. Here, the codes are designed to maximize the free *Euclidean distance* rather than the Hamming distance. Figure 2.10 shows a basic trellis encoded M PSK transmitter.

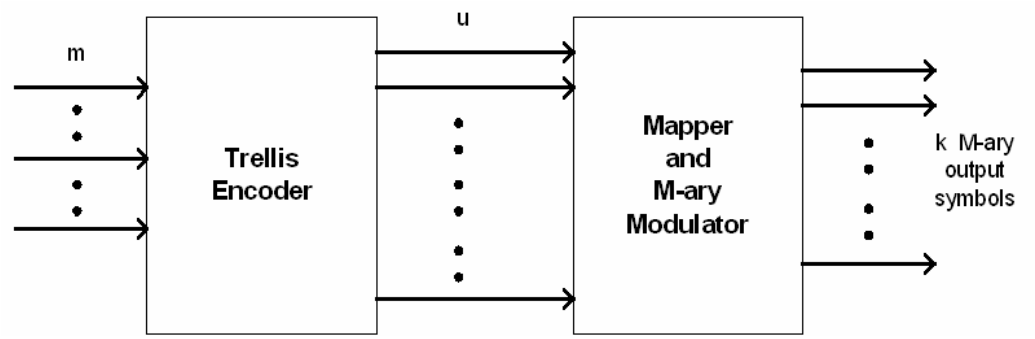

Figure 2.10: M-PSK Trellis encoder

The TCM transmitter shown in Figure 2.10 assigns waveforms to the bits (modulation) but the assignment is performed according to the criterion of better or worse distance properties (channel coding) [13]. Thus, modulation and coding take place together. This ensures maximum Euclidean distance between the adjacent transmitted sequences. Soft decision Viterbi decoding is used to decode the encoded data in the receiver. Ungerboeck showed in his original paper [18] that a simple four-state trellis-coded modulation scheme can provide improvement in the transmission against additive noise by 3dB without reducing data rate or requiring more bandwidth than conventional un-coded modulation schemes.

Trellis Coded Modulation (TCM) is most widely used in data modems which employ this scheme to transmit data reliably over voice band channels. TCM also finds applications in satellite, terrestrial microwave and mobile communications. This is due to the fact that we can increase the throughput rate or permit satisfactory operation at lower SNRs.

Turbo Codes

Turbo codes are nested or parallel convolutional codes that combine the capabilities of convolutional codes with channel estimation theory. The original concepts of turbo codes

were first presented in [19]. These codes allow coding gains which are far superior to all other coding schemes and allow a communication link to come amazingly close to realizing the Shannon capacity bound. Although this level of performance requires a complex receiver design, these codes have become highly popular because of their low BERs and are being incorporated in 3G wireless standards.

2.4.2 Interleaving

Interleaving is a process of permuting the data bits according to a pre-defined mapping. It is usually performed at the output of the channel encoder and before the modulation step. The corresponding deinterleaver uses the inverse mapping to restore the original sequence of the data bits obtained from demodulator and passes the output to the channel decoder. Interleaving the coded message before transmission and deinterleaving after reception causes the burst errors, introduced by the channel, to spread out in time and hence to be handled by the decoder as if they were random errors.

Interleaving techniques have been proven useful for all the block and convolutional codes and have been found to reduce the overall BER of the communication system. Two types of interleavers are commonly used block interleavers and convolutional interleavers [20].

Block Interleaving

Block interleavers work on a block of encoded data bits. The bits received from the encoder are stored in the interleaver's memory by filling the columns of an array. After the array is completely filled the bits are read row-by-row and fed to the modulator, which then transmits them over the channel. A reverse operation is performed at the receiver. The demodulated bits enter the deinterleaver by rows and read out by columns before feeding the decoder. Figure 2.11 illustrates the process of block interleaving. To illustrate this, a burst of three errors are shown in Figure 2.11 in the consecutive transmitted data bits- 1, 5, 9, is converted into isolated single errors in the de-interleaved data.

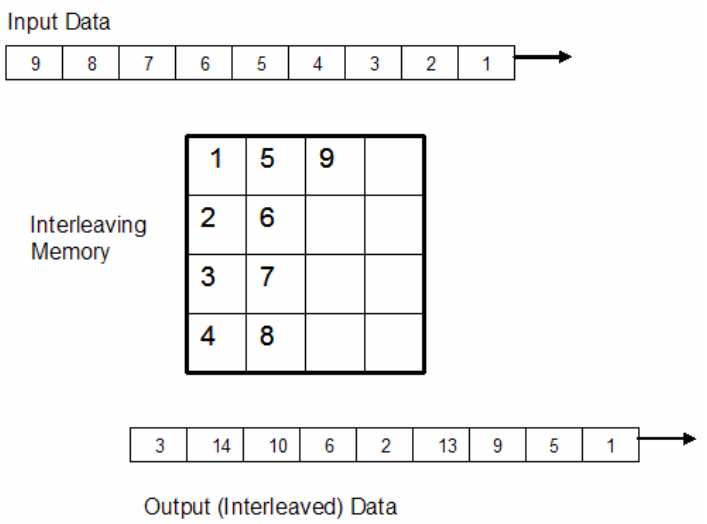

Figure 2.11: Block interleaving

Convolutional Interleaving

Convolutional interleaving was first proposed by Forney [21]. A convolutional interleaver consists of a set of progressive delay lines, each with a fixed delay. Figure 2.12 shows the structure of the interleaver, where Z^{1} denotes one clock cycle delay. With each new bit, the commutator switches to a new delay line and the new bit is shifted in while the oldest bit in that delay line is shifted out to the modulator.

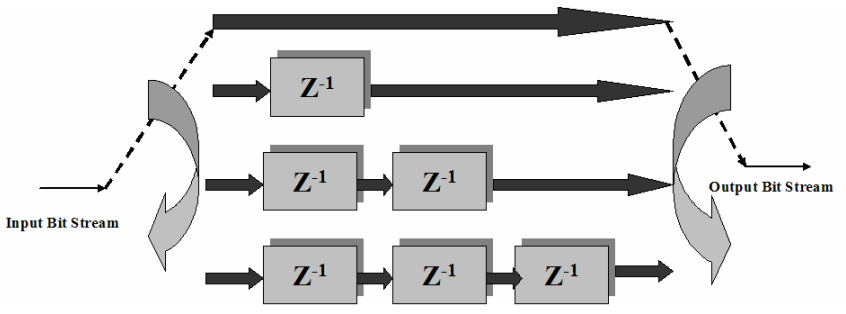

Figure 2.12: Convolutional interleaver

The deinterleaver, shown in Figure 2.13, performs the inverse operation to generate the original data sequence back. The input and output commutators for both interleaving and deinterleaving must be synchronized.

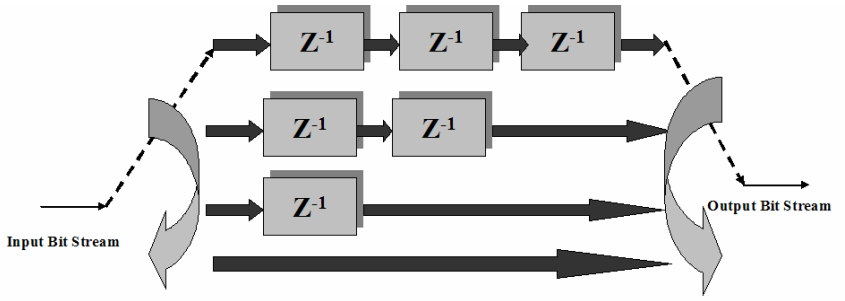

Figure 2.13: Convolutional deinterleaver

The error reduction capabilities offered by the interleavers and deinterleavers is achieved at the cost of additional time delay in the communication system. The time delays in the block interleaving is equal to the size of the array used for interleaving, while in convolutional interleaving the time delay is given by

(*Number of delay lines*)(*Maximum delay among all the delay lines*) (2.4)

The block interleavers are used in compact disks along with powerful block codes to reduce the errors due to scratches on the disk surface. The convolutional interleaver technique is used in telecommunication applications such as GSM and UMTS communication systems and radio systems to protect transmission channels from noise.

2.4.3 Modulation

Modulation is the process by which the message signals are transformed into waveforms that are compatible with the characteristics of the radio channel. Performance of any radio system is dependent on the efficiency of the modulation scheme used. The two most important factors that decide the overall efficiency of the modulation scheme are *power efficiency* and *bandwidth efficiency*. Power efficiency is the ability of a modulation technique to preserve the quality (e.g. BER) of the signal with minimal signal power. It is defined as the ratio of signal energy per bit to noise spectral density (E_b/N_o) required to achieve a particular BER. Bandwidth efficiency is the ability of a modulation

technique to transfer more data at the given bandwidth, which decides the system capacity. It is defined as the ratio of data rate in bits per second to allocated bandwidth in Hertz (*R/B*). There exists a fundamental tradeoff in any communication system between the power efficiency and bandwidth efficiency as one can be achieved only at the expense of the other. Based on these two important factors, digital modulation schemes are broadly classified as *linear* and *nonlinear*.

2.4.3.1 Linear Modulation

In linear modulation scheme the amplitude of the transmitted signal varies linearly with the variations in the information signal. The transmitted signal in this technique can be expressed as

$$\begin{aligned} s(t) &= \mathrm{Re}[A_c m(t) \exp(j2\pi f_c t)] \\ &= A_c[m_R(t)\cos(2\pi f_c t) - m_I(t)\sin(2\pi f_c t)] \end{aligned} \tag{2.5}$$

where, *s(t)* is the transmitted signal and *m(t)* is the message signal, A_c is the amplitude, f_c is the carrier frequency, $m_R(t)$ and $m_I(t)$ are respectively the real and imaginary parts of the complex envelope [22].

Linear modulation schemes are bandwidth efficient; hence, are most widely used in cellular communications, but as the amplitude of the carrier signal varies linearly with the modulating signal, they must be transmitted using linear RF amplifiers which have poor power efficiency [20]. Use of power efficient non-linear amplifiers can result in high sidelobes and, hence, high adjacent channel interference (ACI) in linear modulation techniques results. The most popular linear modulation techniques include QPSK and $\pi/4$ DQPSK.

Quadrature Phase Shift Keying (QPSK)

Phase shift keying (PSK) is the modulation scheme where the phase of the carrier wave is changed depending on the message signal. In case of binary phase shift keying (BPSK), a single message bit is transmitted every carrier cycle, while in QPSK two bits are transmitted in a single modulation symbol. Hence it has twice the bandwidth efficiency of BPSK. Figure 2.14 shows the block diagram of QPSK transmitter. First, it converts the bitstream into a Non-Return-to-Zero (NRZ) signal which is multiplied by an in-phase (*I*) and a quadrature (*Q*) signals. The two components are then summed to achieve the desired QPSK signal.

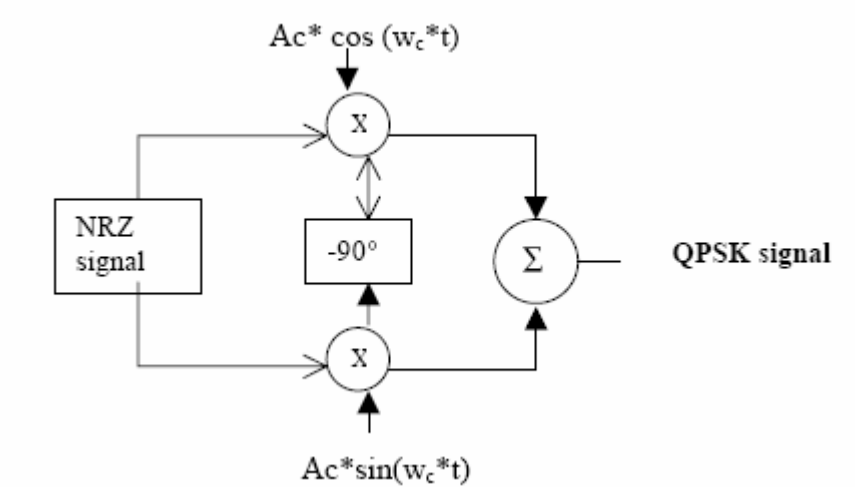

Figure 2.14: QPSK transmitter

The phase of the carrier takes one of four equally spaced values, such as $\pi/4$, $3\pi/4$, $5\pi/4$ and $7\pi/4$, where each value of phase corresponds to a unique pair of message bits. The constellation diagram for QPSK signal is shown in Figure 2.15a. Depending on the message signal a maximum of 180° phase variations are possible in the envelope of a QPSK signal. This may result in the spectral widening and generation of higher sidelobes in the signal spectrum. A modified form of QPSK, known as offset QPSK (OQPSK) is less susceptible to these effects. In QPSK modulation, the bit transition of *I* and *Q* bitstreams occur the same time instants, but in OQPSK, the *I* and *Q* bitstreams are offset in their relative alignment by half a symbol period (one bit period). This confines the maximum envelope variations to 90° reducing the spectral spreading. Figure 2.15b shows the constellation diagram for OQPSK scheme.

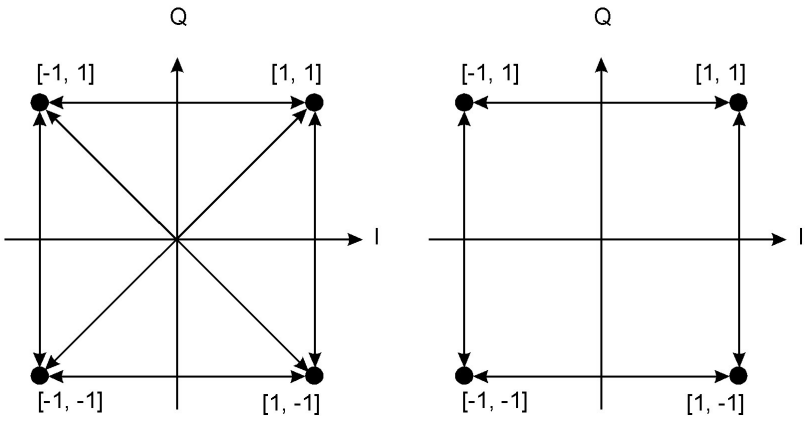

Figure 2.15: (a) QPSK constellation, (b) OQPSK constellation

QPSK signals can be demodulated using a coherent receiver, which generates exact replica of the transmitted carrier signal. The received signal is multiplied with the inphase and quadrature components of the carrier and passed through a low pass filter and decision circuit to recover the message bits. QPSK is currently used in applications such as cable modems and IS-95 (CDMA) systems.

$\pi/4$ Differential Quadrature Phase Shift Keying (DQPSK)

In $\pi/4$ QPSK, one QPSK constellation is used to modulate odd symbol numbers, and another $\pi/4$ shifted constellation is used for even symbol numbers, shown in Figure 2.16. This results in maximum phase transients of 135°, as compared to 180° for QPSK and 90° for OQPSK. Hence it preserves the constant envelope property better than QPSK, but is more susceptible to variations than OQPSK. However $\pi/4$ QPSK has an advantage over OQPSK as it can be differentially detected since a stream of identical 1's and 0's will always produce a phase change. These signals are often differentially encoded to facilitate easier implementation of differential detection. When differentially encoded, $\pi/4$ QPSK is called $\pi/4$ DQPSK. The constellation diagram of $\pi/4$ DQPSK is shown in Figure 2.17.

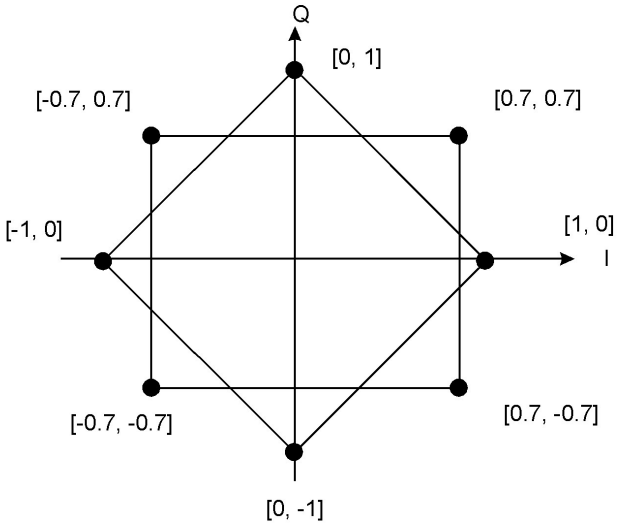

Figure 2.16: Two constellations used for $\pi/4$ QPSK signals

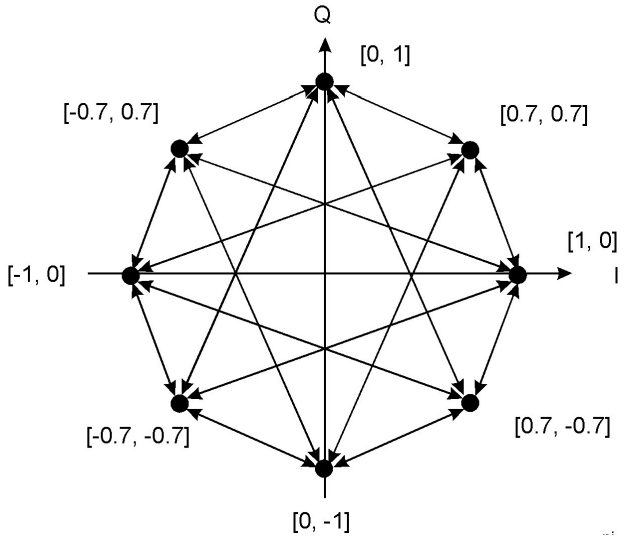

Figure 2.17: $\pi/4$ DQPSK constellation diagram

In $\pi/4$ DQPSK the data is encoded in the change in phase of the transmitted carrier. Let S_{k-1} represent the carrier transmitted for the previous symbol and θ is equal to the absolute phase, then the previous symbol can be expressed as

$$S_{k-1} = A_c \cos(w_c t - \theta) \tag{2.6}$$

The carrier for the present symbol can be expressed as

$$S_k = A_c \cos(w_c t - (\theta + \Delta\theta) \tag{2.7}$$

where, $\Delta\theta$ is the phase change. The above equation can be written as

$$S_k = I_k \cos(w_c t) + Q_k \sin(w_c t) \tag{2.8}$$

where,

$$\begin{aligned} I_k &= A_c \cos(\theta + \Delta\theta) \\ &= A_c \cos\theta \cos\Delta\theta - A_c \sin\theta \sin\Delta\theta \\ &= I_{k-1} \cos\Delta\theta - Q_{k-1} \sin\Delta\theta \end{aligned} \tag{2.9}$$

and

$$\begin{aligned} Q_k &= A_c \sin(\theta + \Delta\theta) \\ &= A_c \sin\theta \cos\Delta\theta + A_c \cos\theta \sin\Delta\theta \\ &= Q_{k-1} \cos\Delta\theta + I_{k-1} \sin\Delta\theta \end{aligned} \tag{2.10}$$

Thus, as seen from the above equations (2.9 and 2.10), the present *I* and *Q* values depend on the previous *I* and *Q* values and the phase change of the carrier. Table 2.1 shows the relationship between input data and $sin\,\Delta\theta$, $cos\,\Delta\theta$.

$A_k B_k$	$A_{kG} B_{kG}$	$\Delta\theta$	$cos\,\Delta\theta$	$sin\,\Delta\theta$
0 0	-1 -1	$5\pi/4$	-0.7	-0.7
0 1	-1 +1	$3\pi/4$	-0.7	+0.7
1 0	+1 -1	$\pi/4$	+0.7	-0.7
1 1	+1 +1	$7\pi/4$	+0.7	+0.7

Table 2.1: Relationship between input data and $\sin\Delta\theta$, $\cos\Delta\theta$

Using the above table, equations (2.9) and (2.10) can be rewritten as:

$$I_k = (I_{k-1}A_{kG} - Q_{k-1}B_{kG})0.707 \quad (2.11a)$$

$$Q_k = (Q_{k-1}A_{kG} + I_{k-1}B_{kG})0.707 \quad (2.11b)$$

The above equations were used to implement a $\pi/4$ DQPSK transmitter on the FPGA. The aim of the receiver is to recover data symbols A_k and B_k, from the four received phase signals I_k, Q_k, I_{k-1}, Q_{k-1}. Data recovery can be done by calculating the sine and cosine phases of the received signal using equations (2.9) and (2.10). However, this can result in a time consuming operation and may also require large number of hardware resources. The data recovery is simplified using the following table which shows the relationship between the received phases and corresponding data bits.

$cos\theta$	$sin\theta$	$\Delta\theta$	$A_{kG}\,B_{kG}$	$A_k\,B_k$
+0.707	+0.707	$\pi/4$	+1 +1	1 1
+0.707	-0.707	$7\pi/4$	+1 -1	1 0
-0.707	+0.707	$3\pi/4$	-1 +1	0 1
-0.707	-0.707	$5\pi/4$	-1 -1	0 0

Table 2.2: Relationship between the received phases and data bits

From the above table it is evident that

1) If ($cos\theta_k$>0) then A_k =1, otherwise A_k=0
2) If ($sin\theta_k$>0) then B_k =1, otherwise B_k=0

Therefore only the sign of $cos\theta_k$ and the sign of $sin\theta_k$ need to be calculated in order to determine the message bits. This can be expressed using following equations:

$$Sign(\cos\theta_k) = sign(Q_kQ_{k-1} + I_kI_{k-1}) \quad (2.12a)$$

$$Sign(\sin\theta_k) = sign(I_{k-1}Q_k - I_kQ_{k-1}) \quad (2.12b)$$

The $\pi/4$ DQPSK modulation scheme has been adopted for the United States and Japanese digital cellular time division multiple access (TDMA) systems and Personal

Communication Systems (PCS). The rational for such a choice is the high bandwidth efficiency and its applicability to non-coherent detection.

2.4.3.2 Nonlinear Modulation

Nonlinear modulation schemes, also known as constant envelope modulation, are used in practical communication systems to improve the power efficiency of the system. Here the amplitude of the carrier is constant, regardless of the variations in the modulation signal. Hence, non-linear power efficient Class C amplifiers can be used without introducing degradation in the spectrum occupancy of the transmitted signal. GMSK is the most popular constant envelope modulation scheme, which is used in Europe's GSM cellular standard.

Gaussian Minimum Shift Keying (GMSK)

In GMSK a Gaussian filter is used prior to modulation to maintain the constant envelope property of the modulation scheme [20]. The block diagram of a GMSK transmitter is shown in Figure 2.18. The NRZ signal is first passed through an integrator and a Gaussian filter before it is multiplied by the in-phase and quadrature components of the carrier.

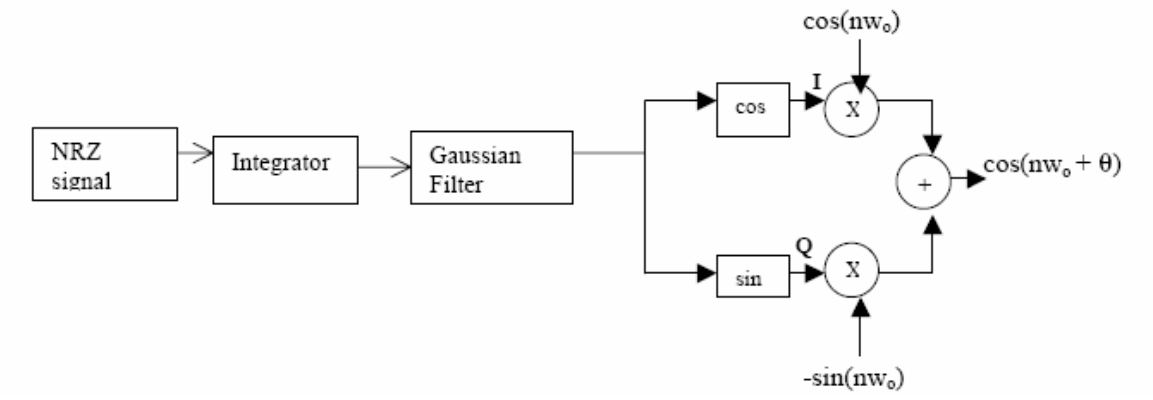

Figure 2.18: GMSK transmitter

The GMSK pre-modulation filter has the impulse response given by [20]

$$h_G(t) = \frac{\sqrt{\pi}}{\alpha}\exp(-\frac{\pi^2}{\alpha^2}t^2) \tag{2.13}$$

The parameter α is related to the 3 dB baseband bandwidth B by

$$\alpha = \frac{0.5887}{B} \tag{2.14}$$

The pre-modulation filter is completely defined by the bandwidth *B* and baseband symbol duration *T*; hence the GMSK signal is defined by its *BT* product. The *BT* factor controls the power spectrum of the final modulated signal. Figure (2.19) plots the simulated RF power spectrum of the GMSK signal for various values of *BT*. A *BT=0.3* is used in GSM cellular standard.

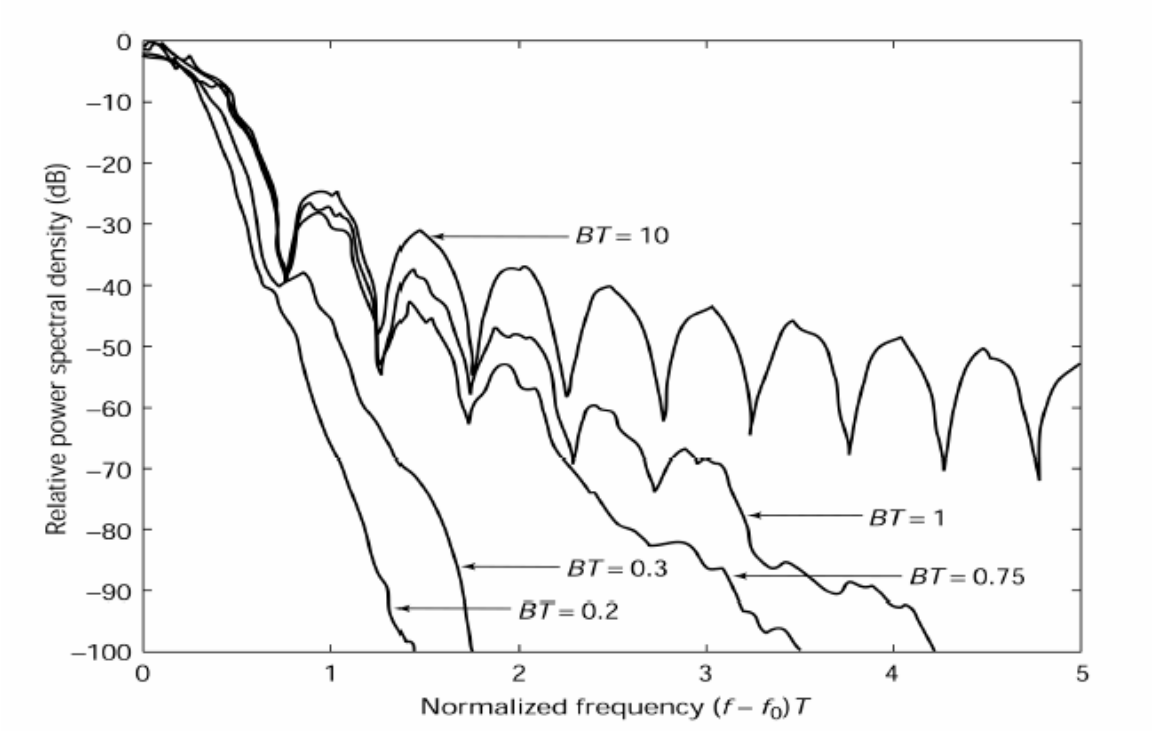

Figure 2.19: RF power spectrum of GMSK signals

The gaussian filter reduces the sidelobes in the transmitted spectrum and thus results in compact spectrum, but the filtering introduces inter symbol interference (ISI) in the transmitted signal. Thus GMSK sacrifices the irreducible error rate caused by gaussian filtering in exchange for good spectral efficiency and constant envelope properties.

Chapter 3

3. System Architecture

3.1 Overview

The high degree of parallelism and modular design flow capabilities provided by the Virtex and Virtex-II architectures make an FPGA-based configurable computing machine an ideal platform for reconfigurable radio applications. Layered radio architecture, involving all the layers from physical to application, can be built by integrating more than one such field programmable devices on a single platform. This chapter surveys two such platforms used in this research to implement the physical layer radio design. It highlights their salient features and capabilities along with their system architectures and advantages. Lastly, this chapter describes the architecture of a soft-radio card which was used in conjunction with the FPGA platform to perform signal processing tasks at the intermediate frequency.

3.2 SLAAC1V FPGA Platform

The baseband signal processing sub-system blocks for the reconfigurable radio have been implemented on the Virtex FPGAs of SLAAC1V platform that offers a fast, powerful and flexible way of implementing configurable designs.

SLAAC1V is the result of System Level Application of Adaptive Computing (SLAAC) project conducted at University of Southern California Information Sciences Institute (USC-ISI) [23]. It provides a high-speed network cluster of heterogeneous FPGA-accelerate nodes constructed on a 64-bit PCI board. Figure 3.1 shows the top-level architecture of SLAAC1V from user's perspective.

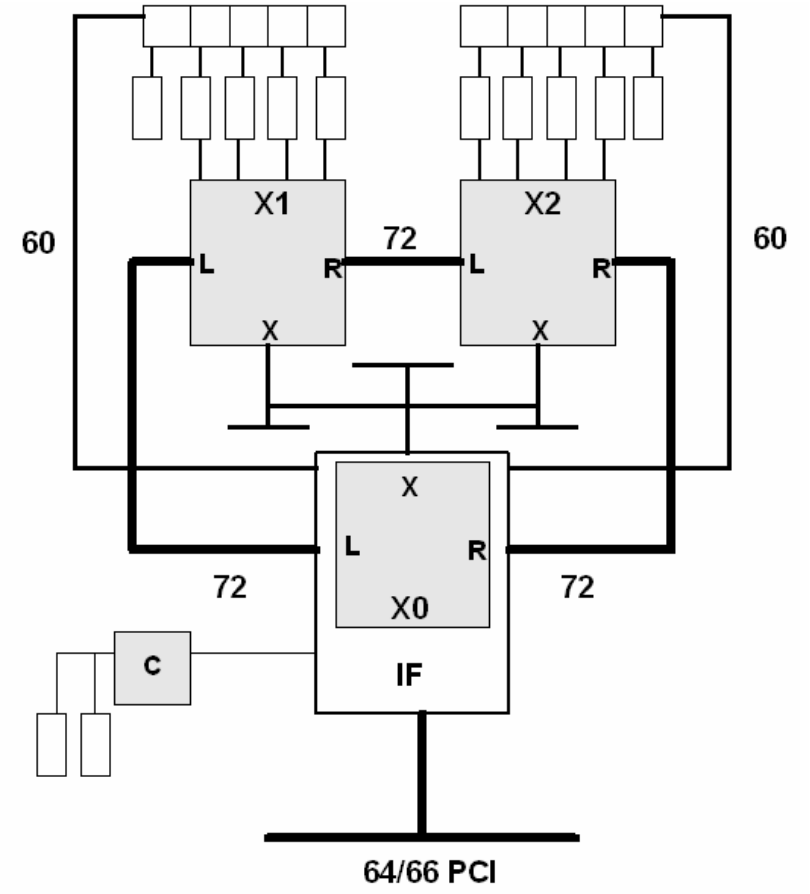

Figure 3.1: Top level architecture of SLAAC1V platform

This platform is comprised of three Xilinx Virtex XCV1000fg560-6 FPGAs. The three user-programmable FPGAs X0, X1 and X2 are organized in a ring structure through a systolic bus and a crossbars bus each 72-bit wide supporting bi-directional data transfers. Also, 2- bit bi-directional handshake lines connect each FPGA to each other. X0 FPGA acts as a control element that manages the user data flow and X1 and X2 FPGAs act as computing elements. As evident from the Figure 3.1, the control FPGA contains PCI interface that communicates with the host. The configuration controller FPGA XCV100

attached to the control FPGA programs the three processing elements (PE) using the configuration bitstreams stored in the memory attached to it.

The board features twelve 256x36 100MHz ZBT synchronous SRAMs; four of these memory banks are connected to X1, four to X2, two to X0 and two to the configuration controller. Each memory bank provides one megabyte of storage capability and a 36-bit wide memory access data path. These memories are connected through bus switches controlled by X0 which can swap one of its SRAM with that of X1 and other SRAM with that of X2. The memory architecture as described in SLAAC1V User VHDL guide [24] is shown in the Figure 3.2.

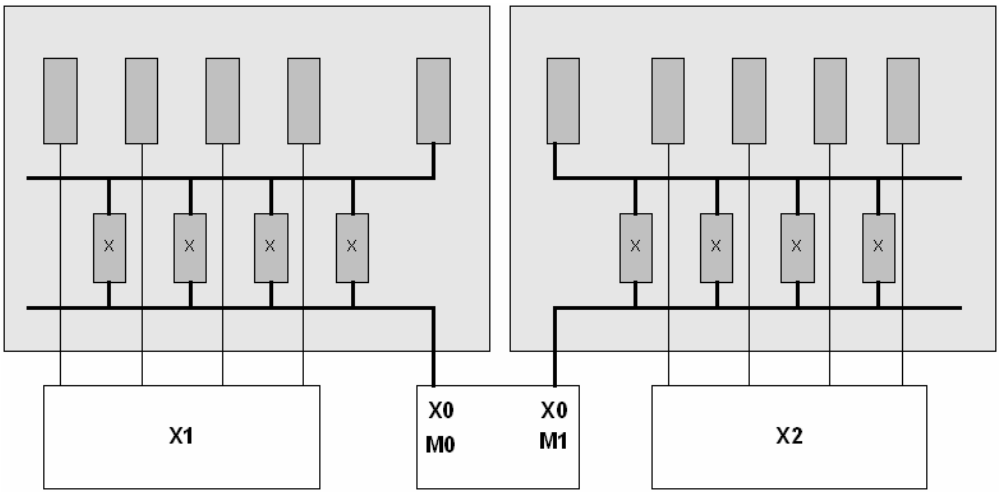

Figure 3.2: SLAAC1V Memory architecture

The SLAAC1V SDK manual [25] describes in detail the API for controlling the board. The baseband processing modules for the radio design have been implemented on X2 FPGA that communicates to the soft-radio card through X0 control FPGA. The data transfers between the two FPGAs takes place using memory bank switching as described above. The design and implementation results are presented in Chapter 5 of the thesis.

3.3 DINI FPGA Platform

The modular design flow for partial reconfiguration of the radio has been implemented on Virtex-II FPGA on DN3000k100 platform developed by the DINI group [26]. Figure 3.3 shows the block diagram of this board.

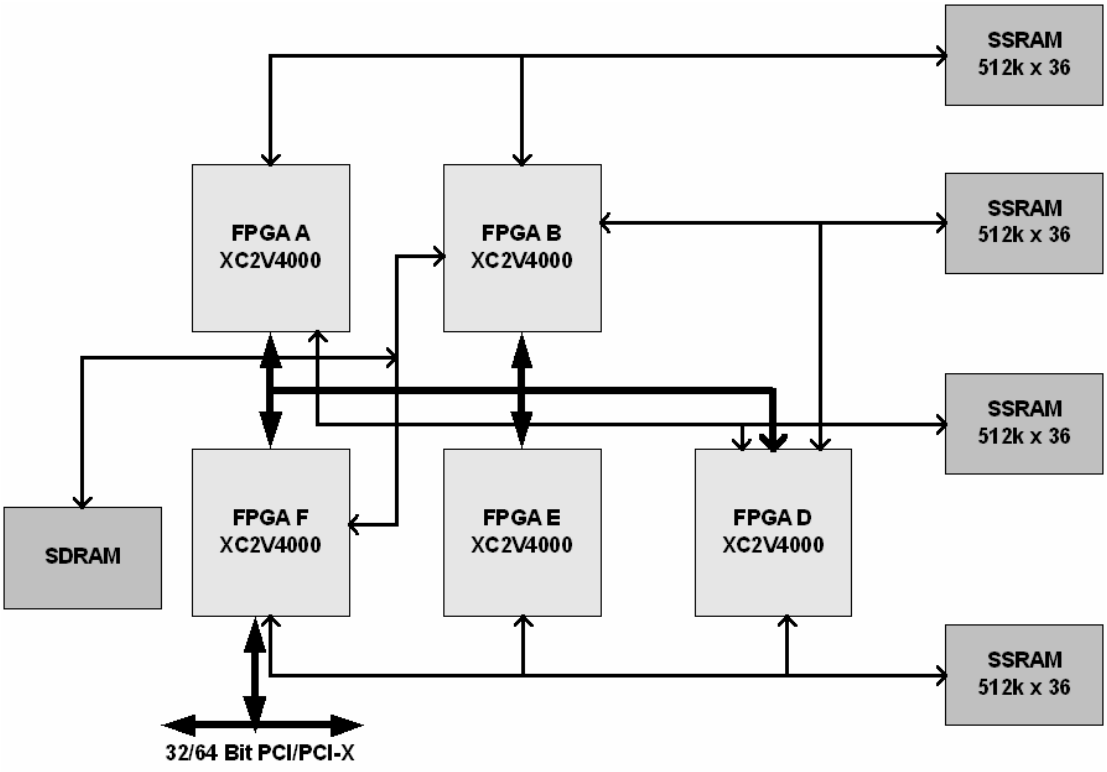

Figure 3.3: DN300k100 platform block diagram

This is another multi-FPGA platform, more powerful than SLAAC1V, which contains five Xilinx Virtex-II XC2V8000/6000/4000 devices. The board used in the research is the one that comprises of five XC2V4000 FPGAs, four 512x36 sync burst SRAMs and one gigabyte SDRAM. The five FPGAs are tightly interconnected through a 227-bit bus and control signals for handshake and data transfers between them. FPGA-F acts as a control FPGA and communicates with the host machine through 32/64-Bit 3.3V PCI interface [27]. The FPGA configuration can be done using a FLASH card and an onboard microcontroller. However, the board configuration has been modified to allow FPGA programming over the PCI bus[2]. The availability of an RS232 port makes configuration and operational status and control checks possible. FPGA-A has been used to implement

[2] The program to configure the DINI board over the PCI bus was written by Dr. Peter Athanas

modular design flow using the advanced capabilities of the Virtex-II architecture. The communication with the host computer is done via FPGA-F. The partial reconfiguration design process is discussed in detail in the next chapter.

3.4 Soft-Radio Card Architecture

Soft Radio card used for converting the data from baseband to IF and vice versa is a Red-River [28] programmable poly-channel wideband digital transceiver capable of supporting eight parallel channels and 40MHz of analog I/O bandwidth. Figure 3.4 describes the detailed block diagram of the transceiver.

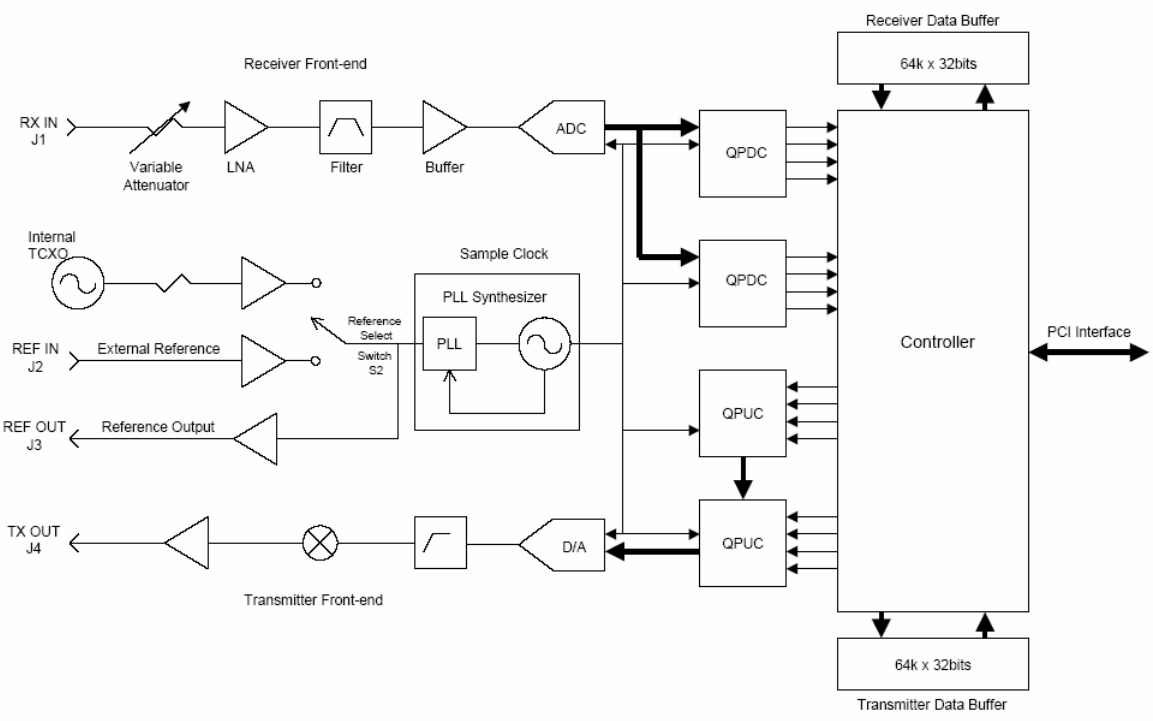

Figure 3.4: Soft radio card block diagram[3]

In the transmitter data path, the data samples to be transmitted either sourced from the PCI interface or from the host memory using DMA are stored in the transmitter data buffer that acts as a multi-queue FIFO, wherein each queue is assigned to each active channel. These samples are buffered out of FIFO to match the clock frequency at the

[3] Source: Red River [29]. Used by permission.

input of digital up-converter. Two quad programmable up-converter (QPUC) chips provide eight independent data channels. The up-converter filters and interpolates the data before providing it to the D/A converter. The analog signal obtained from the D/A converter is passed through two converting stages to provide the IF signal at 70 MHz. To obtain the output at baseband the analog mixer after D/A converter is bypassed and the signal is fed to a lowpass baseband filter prior to transmission. A list of transmitter configuration specifications is shown in Table 3.1.

Parameter	**Value**
Sample Clock Rate	93 MHz
Phase Noise (Internal Sample Clock)	-80dBc/Hz @ 10kHz offset
Internal Reference Clock Stability	+/- 2.5 ppm (@10 MHz, -20 to 70 C)
Spur Free Dynamic Range	75 dB
Bandpass Configuration Specifications	
Center Frequency	70 MHz
Output Bandwidth (3dB)	40 MHz
Full-scale Output Power (50 ohms)	-12 dBm
Output Third Order Intercept (OIP3)	+12 dBm
Baseband Configuration Specifications	
Frequency Range (3 dB)	0.1 to 40 MHz
Full-scale Output Power (50 ohms)	-12 dBm
Output Third Order Intercept (OIP3)	TBD

Table 3.1: Transmitter Configuration Specifications [29]

The receiver section of the card accepts the analog signal at 70 MHz. This signal is routed through a digitally controlled attenuator and a low noise amplifier (LNA). For baseband operations a lowpass filter is employed and for bandpass operations a 40MHz surface acoustic wave (SAW) filter is employed prior to the amplification. The buffer amplifier boasts the analog signal before proving it to the A/D converter. The resulting signal is provided to two quad programmable digital down-converter chips (QPDC) that extracts the information from 40MHz input band and provide eight independent channels

of digitized data. The 32-bit complex data samples obtained from down-converter are stored in a receiver buffer, which like transmitter buffer acts as a multi-queue FIFO for different active channels. The digital data can be read from the FIFO either performing PCI bus read or can be stored in the host memory by DMA control. Table 3-2 shows a list of key receiver configuration specifications.

Parameter	**Value**
Common (Baseband & Bandpass) Configuration Specifications	
Analog Attenuation Range	20 dB (precision) 70 dB total
Maximum Input	+15dBm (minimum attenuation)
Sample Clock Rate	93 MHz
Internal Reference Clock Stability	+/- 1.5 ppm (@ 10 MHz, -20 to 70 C)
Linear Dynamic Range (1 MHz Bandwidth)	90 dB
IMD Rejection	75 dB (typical)
Bandpass Configuration Specifications	
Center Frequency	70 MHz
Output Bandwidth (3dB)	40 MHz
Full-scale Input Power	-12 dBm (minimum attenuation)
Noise Figure	10.4 dB
Input Third Order Intercept (IIP3)	0 dBm (minimum attenuation)
IMD Rejection	75 dB (typical)
Baseband Configuration Specifications	
Frequency Range (3 dB)	0.1 to 40 MHz
Full-scale Input Power	-14 dBm (minimum attenuation)
Noise Figure	7.9 dB
Input Third Order Intercept (IIP3)	+6.2 dBm (minimum attenuation)
IMD Rejection	80 dB (typical)

Table 3.2: Receiver Configuration Specifications [29]

The controller acts as an interface to the host machine and is used to configure QPUC and QPDC chips, FIFO memory space and program receiver attenuator. It features other

functions such as auto DMA capabilities and interrupts alerts to the host machine which are used for data transfers between transceiver buffers and host memory space. The transmitter and receiver links work at 10MHz clock provided by internal temperature compensated crystal oscillator (TCXO). Alternatively, the transceiver can accept an external clock signal through its front panel.

The card offers a memory-mapped host interface with direct access to all the control registers, transceiver data flow control, channel memory allocation, down-converter and up-converter configuration spaces. The interrupts provided by the interface are used to alert the host of an error condition or the data service request. The transceiver programming is done using the software configuration tool provided by Red River [28] that automates the process of computing and setting the register values based on the desired receiver and transmitter characteristics, such as receiver attenuation control, number of channels, channel bandwidth, filter coefficients, baseband or bandpass mode selection, signal output power etc. The tool generates C header files containing the complete memory map of the device. A snippet of one such header file used for configuring the board for the loop-back test is shown in the Appendix. Figure 3.5 shows a snapshot of the configuration tool used for generating the configuration files for a particular application.

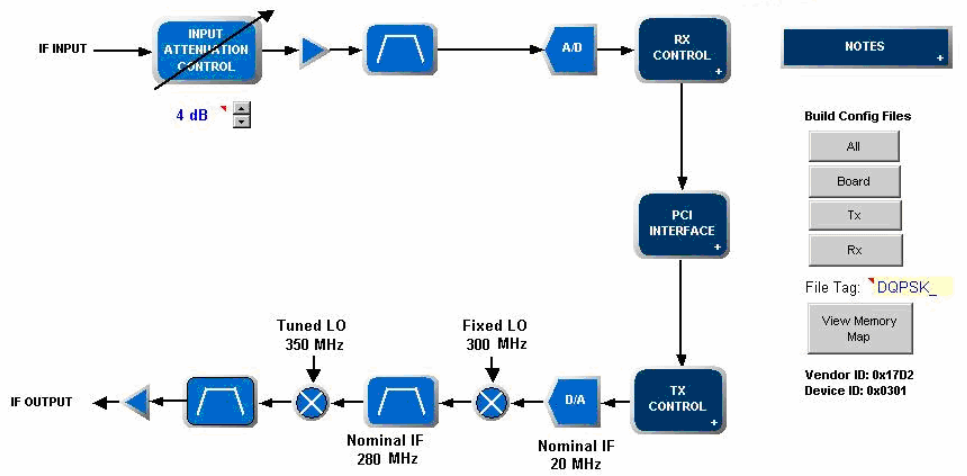

Figure 3.5: Configuration tool for soft-radio card

In this project the transceiver card has been used as a stand alone node outside the host PC and it directly communicates with the SLAAC1V platform for data transfers between its internal buffers and the control FPGA on SLAAC1V card. The transceiver works in the bandpass mode and acts as a bridge between FPGA platform and the RF front end. The configuration header files for the different modulation schemes, as obtained using the software configuration tool, are stored in the control FPGA. These files are used during the system initialization to configure the radio card for the desired application.

Chapter 4

4. System Design

4.1 Overview

The previous chapter described the architecture of two different FPGA platforms used for implementing the physical layer of a reconfigurable radio. This chapter first presents the transmitter and receiver link designs of the physical layer and describes the integration of the various baseband processing modules for data transfers between the network and RF components. The chapter then presents *System Generator*, a hardware modeling tool from Xilinx that was used to test the designs in the MATLAB *Simulink* environment, as a tool for radio design. Finally, the chapter describes in detail the FPGA design process. It highlights the essential steps involved in regular FPGA design and modular design for partial reconfiguration.

4.2 Transceiver Link Design

The physical layer handles the transfer of data between the network layer and the RF section of the radio. Physical-layer tasks include clock synchronization between the network layer and the physical layer, data format conversion, and baseband processing of the data in the transmitter and the receiver links. The FIFO interface between the network and the physical layer provides the required buffering needed by the two layers

running at different clock speeds. Figure 4.1 presents the block diagram of the entire system. The transmitter and receiver links, for data transfers between the network and the antennas, can be seen from the figure. The *user-processing* FPGA and the *control* FPGA on the SLAAC1V card work in conjunction with the soft-radio card and the RF front end to covert the data from baseband to RF and vice-versa. The link designs are described in detail in the following sub-sections.

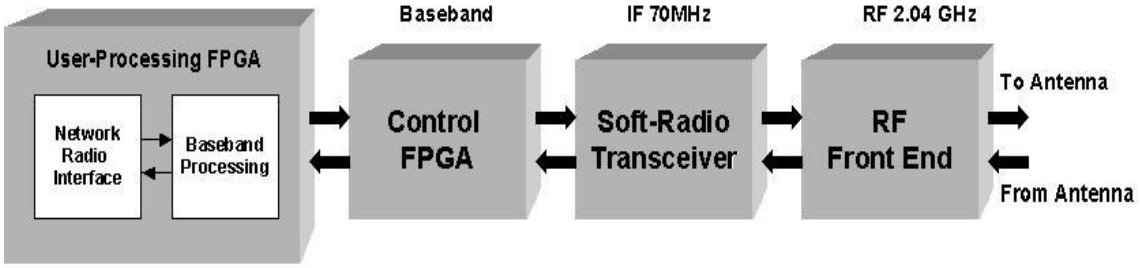

Figure 4.1: Physical Layer Radio Block Diagram

4.2.1 Transmitter

The transmitter section receives the data from the network FIFO in an 8-bit word format. The 8-bit word is broken down into two 4-bit words that act as the input to the channel encoder, which is the first step in baseband processing. The encoder performs block processing and adds the parity bits to the incoming data to enable forward error correction (FEC) in the received data at the other end of the radio. The information signal along with the redundant data bits is passed on to the interleaver block. The interleaver block implements convolutional interleaving that permutes the data bits using the progressive delay line structure, as described in Chapter 2. It also compensates for the delay introduced in the interleaving operation. The delayed and permuted data stream is then passed through a baseband modulator. This block first converts the serial input stream into a parallel data stream for in-phase and quadrature channel processing. Depending on the type of modulation scheme selected it then performs QPSK, $\pi/4$ DQPSK or GMSK modulation on the interleaved data. In the case of GMSK the

interleaved data is first passed through a pre-modulation gaussian filter of the required *BT* product. The transmitter data flow fro baseband processing is shown in Figure 4.2.

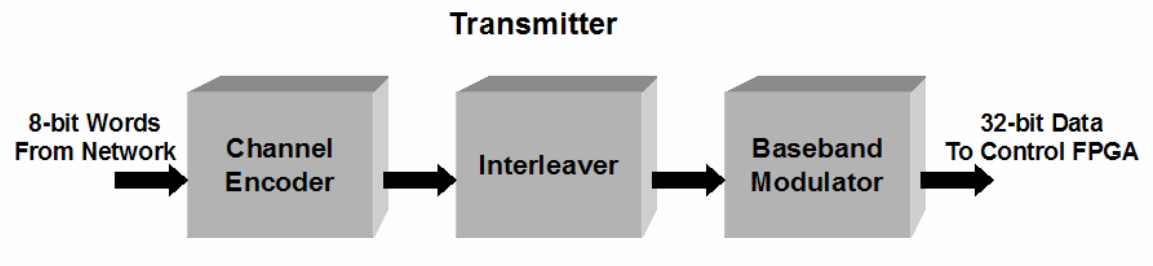

Figure 4.2: Transmitter data flow

The above described baseband processing blocks, implemented in a *user-processing* FPGA, feed the data to the *control* FPGA on the SLAAC1V board in a 32-bit format consisting of 16-bit *I* and 16-bit *Q* outputs. The *control* FPGA communicates with the soft-radio card that up-converts the baseband data to 70 MHz IF for user-defined channel bandwidth, bandpass filter coefficients and output power. The analog data at IF is finally passed through the RF section and transmitted through a specialized four-point antenna [30] as a 2.04 GHz waveform.

4.2.2 Receiver

The receiver performs the exactly opposite operations as transmitter and provides the data back to the network layer in an 8-bit word format. The 2.04 GHz waveforms received though the antennas is down-converted to 70 MHz IF and passed to the soft-radio card. The radio card converts the incoming analog signal to a digital stream and performs digital down conversion to provide the baseband data using the programmable AGC, filter and channel settings. The 32-bit data words obtained from the soft-radio card are passed to the *control* FPGA of the SLAAC1V board that communicates with the *user-processing* FPGA. The receiver data flow on the SLAAC1V board is shown in Figure 4.3.

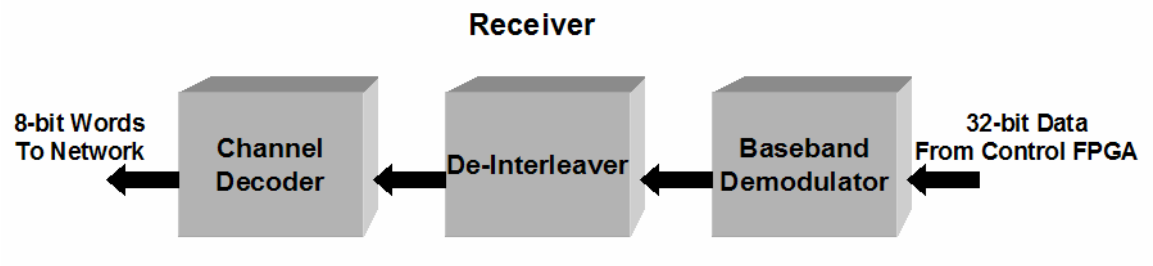

Figure 4.3: Receiver data flow

The 32-bit data word is separated into 16-bit *I* and *Q* words to perform demodulator operation. The demodulator output is passed through a parallel-to-serial converter that provides a serial bit stream to the deinterleaver. The deinterleaver performs the inverse operation as interleaver and generates the original data sequence back. This data sequence is provided as a 7-bit word to the channel decoder that removes the parity bits from the data and extracts the 4-bit message bits. Two such 4-bits words are combined together and provided to the network layer in the required 8-bit word format. The packets are stored in the receiver FIFO and used by the network layer at the right clock instant.

4.3 Design using System Generator

Simulation and testing of the transceiver design flow were done using *System Generator*, a system level modeling tool from Xilinx [31]. This tool can be used for designing and testing DSP systems for FPGAs in a visual data flow environments such as MATLAB *Simulink*. It offers flexibility to test the signal processing algorithms and verify their functionalities at the bit-level without translating the design into hardware. Designs implemented and verified using *System Generator* can be directly mapped onto an FPGA using an automated flow from *Simulink* models to configuration bitstreams.

Figure 4.4 describes the design flow using *System Generator* [3]. The system model is created in the MATLAB environment using Xilinx library. The library includes various hardware models for communications, DSP, controls, data analysis and visualization.

The basic elements include standard building blocks for digital logic such as registers, counters, delays, accumulators, adders, probes, FIFO, logical and arithmetic expressions etc. Besides, the library offers advanced blocks such as single-port and dual-port RAMs, FIR/IIR filters, state machines, up/down samplers, data converters, FFT, DDS etc.

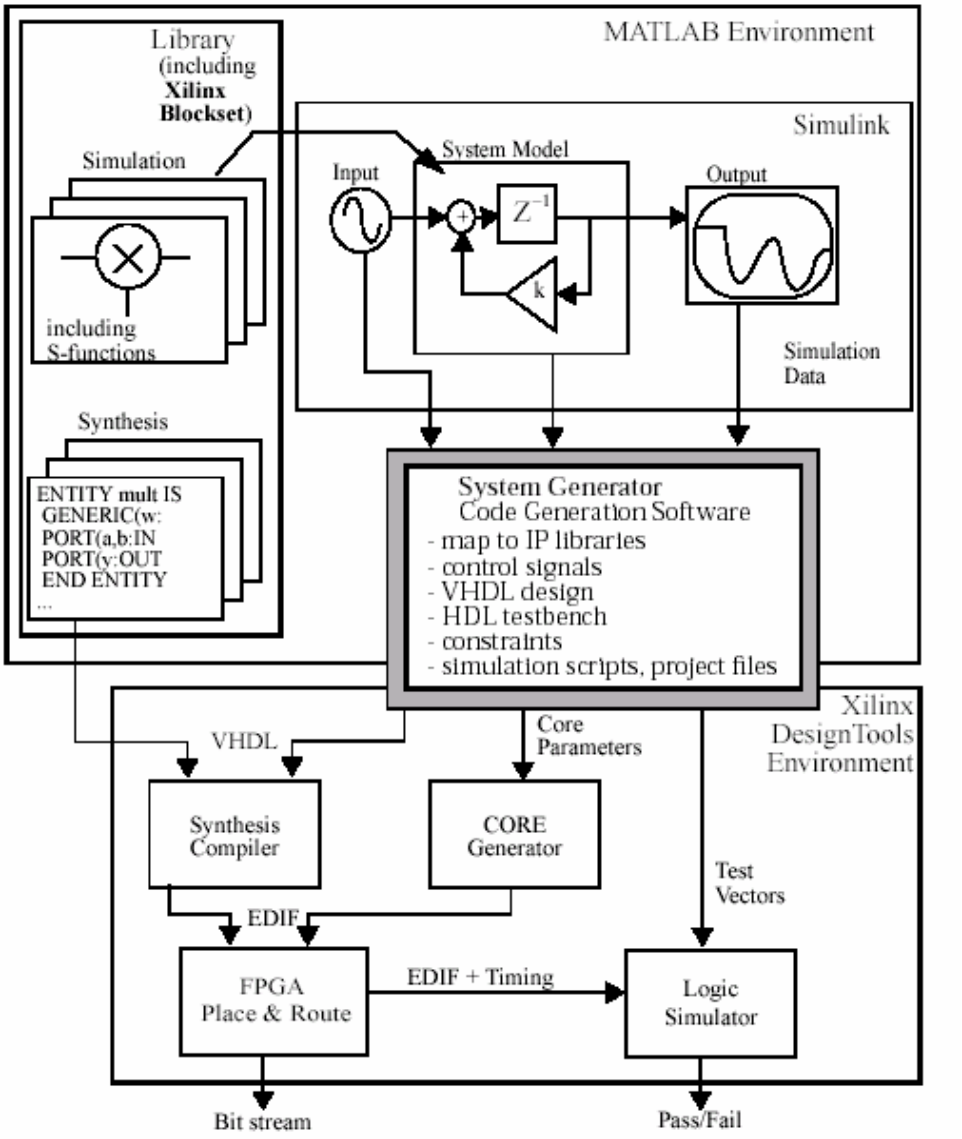

Figure 4.4: System Generator design flow

In addition to the present IP cores, user-defined IP cores can also be integrated using black boxes. These blocks can be used to create fixed- point systems in *Simulink*. The user can specify the target device and the synthesis tools to be used for implementing the

design. *System Generator* automatically generates the HDL files, timing constraint files and test-benches for the target device, and project files for the target synthesis tools. The generated files can be used in the specified design tools, such as *Xilinx ISE* or *Synplify*, to create the bitstreams and download into the FPGAs. The test-benches generated can be used to simulate the design using hardware simulation tools such as *ModelSim* for functional verifications. Data flow models, traditional hardware design languages (VHDL, Verilog, and EDIF), and functions derived from MATLAB programming languages can be simulated and synthesized together into a working hardware.

A $\pi/4$ DQPSK modulator designed in *System Generator* is illustrated in Figure 4.5. The design is created using Xilinx block sets. The system accepts the floating point serial data stream from a MATLAB file and converts into a fixed point data stream using *Gateway In* data converter. The fixed-point serial stream is separated into *I* and *Q* channels using a serial-to-parallel converter. The *M code* block is used to implement the state machine for the DQPSK modulator. This block allows the user-defined functions written in MATLAB to be converted into hardware. The feedback, required for the algorithm, is provided using registers with a single clock cycle delay. Final output is stored in a data file in the floating point format using *Gateway Out* converter. The *System Generator* block allows the selection of a target device, speed, package type, and the synthesis tool. The *Resource Estimator* block estimates the FPGA hardware resources needed to implement the design. Estimates include number of slices, LUTs, block memories, flip-flops, embedded multipliers, I/O blocks and tri-state buffers. These estimates make it easy to determine how design choices influence hardware requirements.

The designs for the entire transmitter and receiver links of the radio, using *System Generator* modeling tool, are described in the Implementation and Results chapter of the thesis.

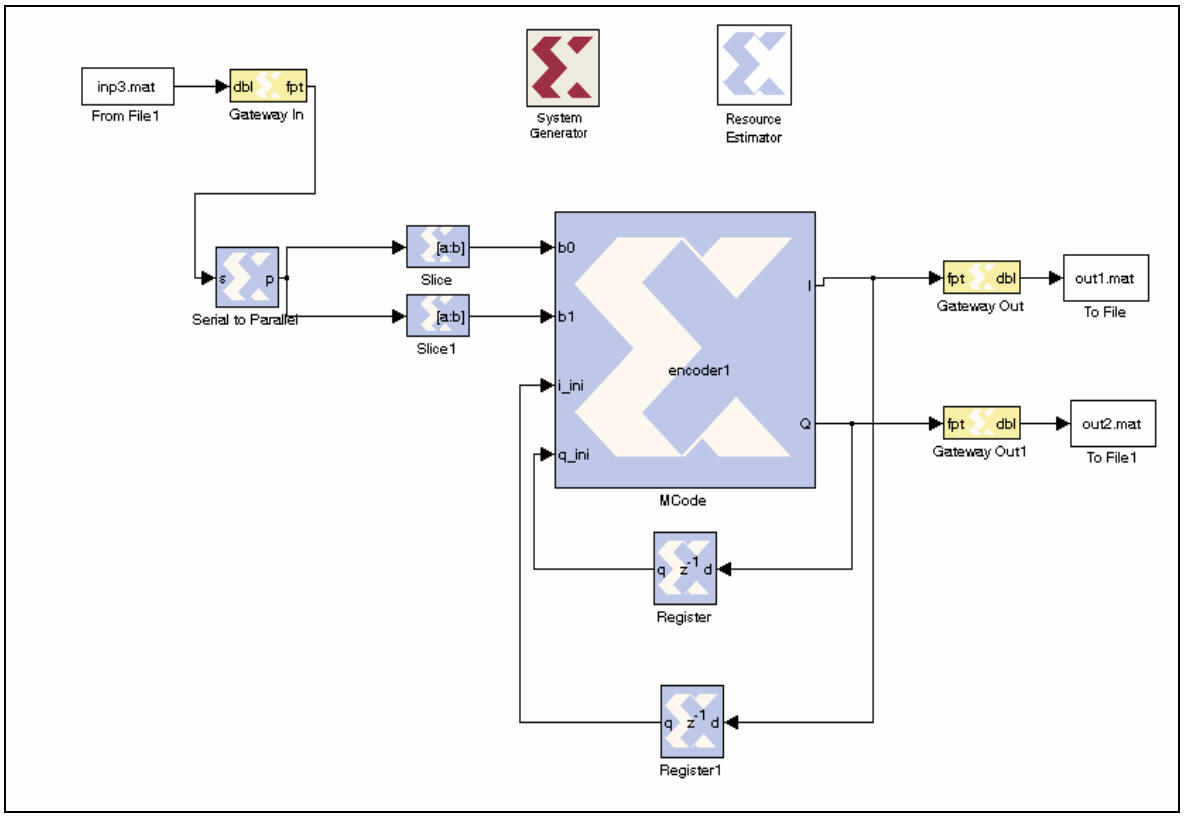

Figure 4.5: DQPSK Modulator design using System Generator

4.4 FPGA Design Process

The FPGA design process starts with the functional specifications of a digital design and culminates with the device programming by loading the configuration bitstreams corresponding to the given specifications. The regular design process involves one time configuration of the entire chip for a particular logic, while in modular design flow the parts of the FPGA can be individually reconfigured after the device has been configured once on power-up. This allows changing the functionality of the device run-time by using partial bitstreams. The design flows for both these processes are described in the following sub-sections.

4.4.1 Regular Design Flow

The process of implementing a design on an FPGA can be broken down into several stages as described in Figure 4.6. Design specification is the first and important step of the design flow. Choosing the right FPGA family and technology for design implementation are governed by the functional specifications [32]. The digital circuits can be modeled in HDL languages such as VHDL or Verilog. The design entry can also be done in a schematic form using a schematic editor. In the next step (Synthesis), the RTL design is optimally translated to the gate level design that can be mapped to the logic blocks in the FPGA meeting the timing and area constraints as desired by the user. Design Implementation is the step where the layout of the chip is computed. This involves using vendor specific tools to place various functions in the available blocks in the FPGA device family and to route the blocks. Design implementations for the Xilinx FPGAs are done in three steps, namely Translate, Map, Place and Route (PAR). The first step requires reading a netlist file and creating a generic database describing the logical design reduced to Xilinx primates using *ngdbuild* tool. In the next step the layout of the design is mapped onto the FPGA. Timing analysis of the design is also done in this step. The last step of design implementation is place and route of individual components on the FPGA fabric. This can be done manually or automatically using PAR tools such as *FPGA Editor*. The final step in the design flow is loading the configurable bitstream onto the device. The software tools such as *BitGen* is used to create the bitstream and Xilinx *iMPACT* offers an easy way of downloading the bitstreams onto the FPGAs using serial or parallel cables.

Design verification is an important step of testing the functionality and performance of the design at every step of the flow. Functional simulation is performed after the design entry to check for the bugs in the design entry. It involves simulating the HDL or schematic design files and checking for logical and timing errors. It can be performed by writing test benches or creating and simulating test benches using *HDL Bencher* in Xilinx ISE environment. Post synthesis simulations can be done using *ModelSim* hardware simulation tools. Timing analysis is a process that examines a synchronous design and determines its highest operating frequency by calculating the worst case delays through

the critical paths in the circuits. *Timing Analyzer* performs the described operation after the implementation step. In-circuit verification allows on-chip debugging of the designs. This can be performed using logic analyzer or Xilinx verification tool called *ChipScope Pro* that integrates key hardware components with the target design without the need of external probes.

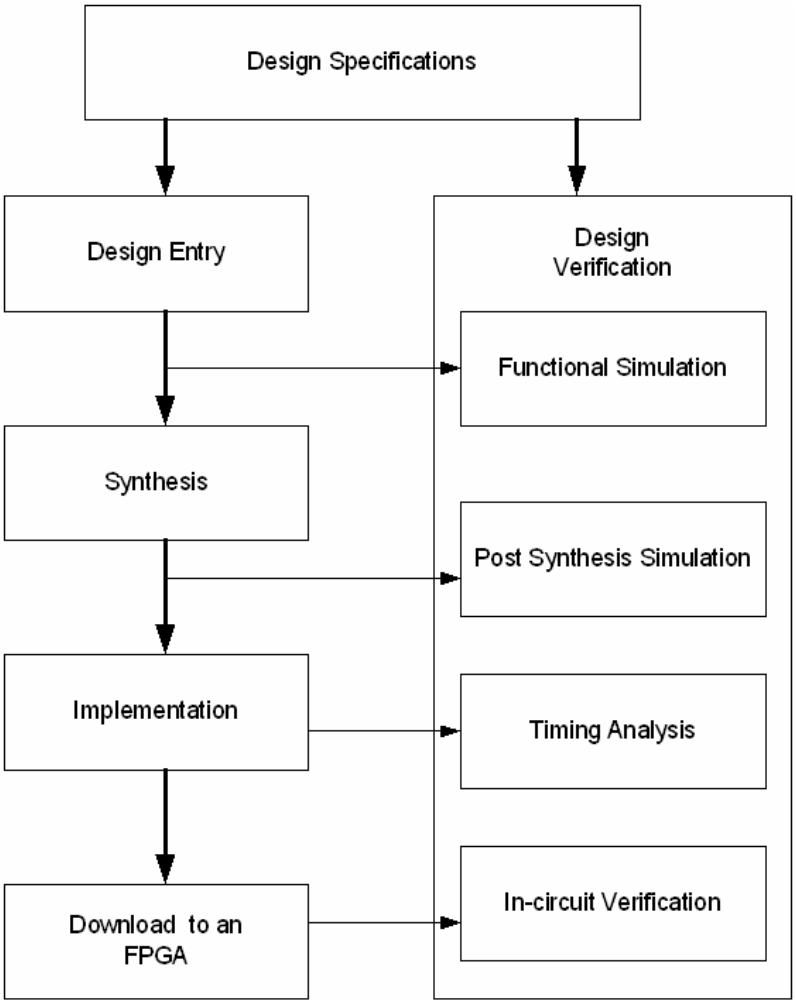

Figure 4.6: FPGA design flow

4.4.2 Modular Design Flow

Partial reconfiguration (PR) allows selective access to the configuration memory by configuring only a part of the device while rest of the circuit is fully operational. It thus

offers efficient use of the available board space and runtime reconfiguration capabilities. There are two styles of PR for the Virtex devices: Module based and Difference based [33]. Difference based style is useful when a small modification needs to be done to the existing design. It can be accomplished by making manual modifications to the design in the *FPGA Editor* and generating the bitstreams based on only the differences between the new and existing design. Modular design flow is used when one or more modules need to be changed or replaced in the device. The modular design flow is described in detail in this section.

Like in a regular FPGA design, the modular design starts with the specifications for the required reconfigurable system. The basic steps involved in the design are shown in the Figure 4.8. The design entry is done in conformance with the PR guidelines that require the overall structure to be a top-level design including global I/Os, modules instantiated as black boxes, and the signals that connect the modules with each other and to external I/O ports. The communication between reconfigurable and other modules is done through fixed data paths defined by bus macros. Bus macros are the tri-state buffers that allow information to pass from one direction to other. Each bus macro provides 4-bit of inter-module communication as shown in Figure 4.7.

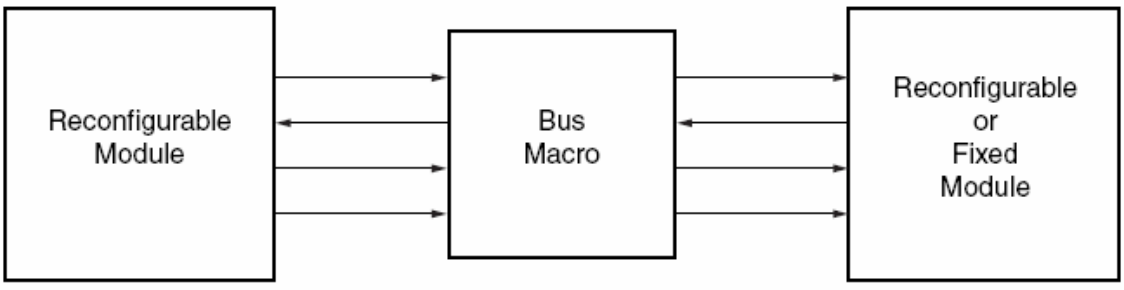

Figure 4.7: Bus Macro used for Inter-module signals

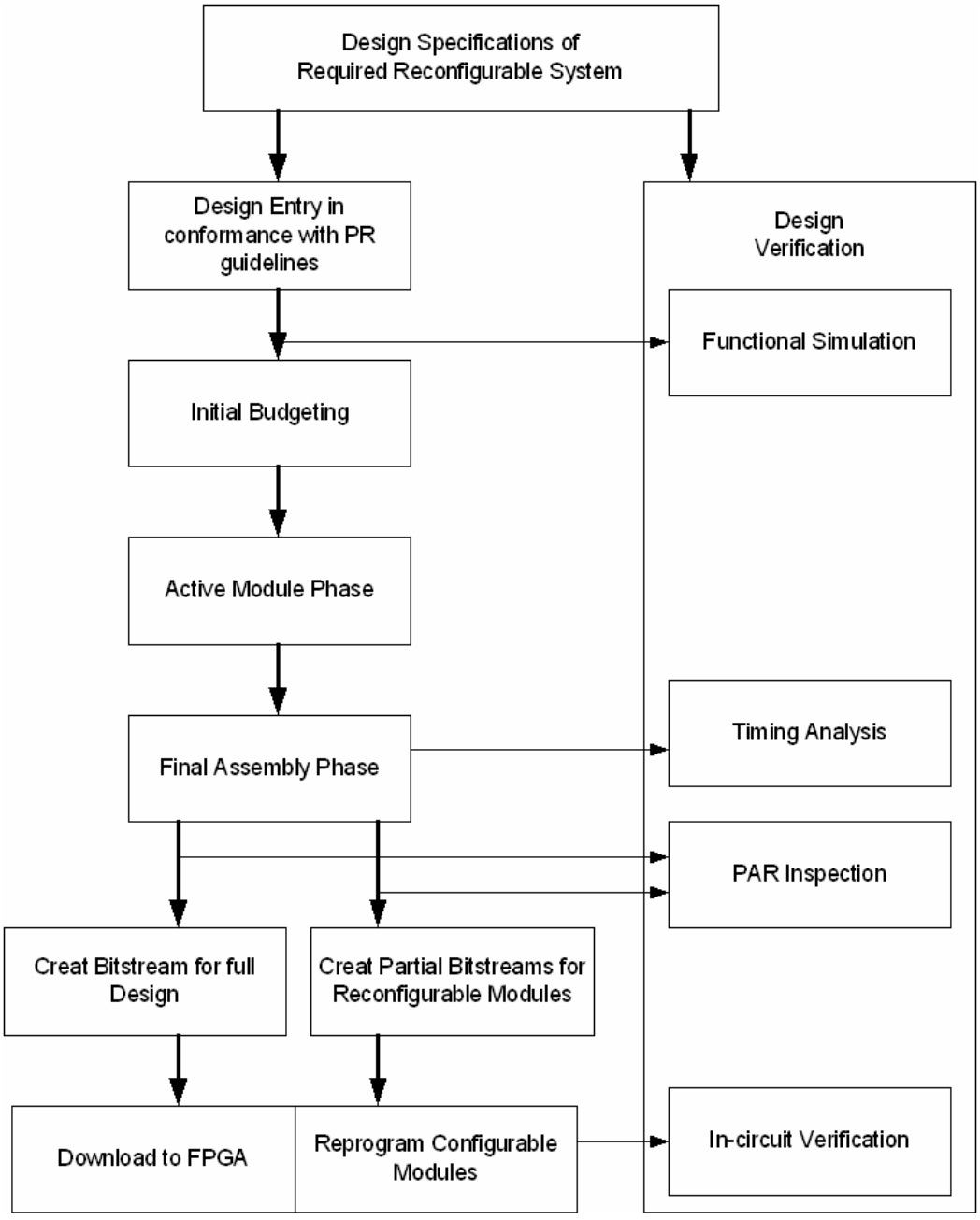

Figure 4.8: Modular design flow

Modular design implementation includes three phases, namely Initial Budgeting phase, Active Module phase and Final assembly phase [33]. In the first phase, the top-level location and timing constraints are defined for the design. *Floor planner* can be used to assign location constraints, while *Constraint Editor* is used for assigning timing constraints. All the modules are individually implemented in the Active Module phase.

The modules are placed and routed and checked for the desired timing constraints. In the final phase of implementation all the modules are assembled into one complete design by running *ngdbuild* on the top-level design. This is followed by map, place and route of the entire design. A bitstream is created for full design for initial power-up configurations and partial bitstreams are created for each individual module. The device is first programmed using power-up configuration. The reconfigurable modules are programmed using their partial bitstreams.

Design verification includes functional simulations for logic and timing for each module, timing analysis to calculate for the delays through the critical paths, PAR inspection to ensure proper intermodule communications and in-circuit verifications for on-chip debugging like in the case of regular design flow.

Chapter 5

5. Implementations and Results

5.1 Overview

The previous chapter outlined the radio transceiver design without delving into the details of the actual implementation. This chapter describes the implementation details of the radio and presents the results for the designs. First, the chapter describes the entire system modeling in the *System Generator* tool. This was used as the first step to check for the functional verification of the design. Following this section, the next section highlights the FPGA implementations issues, presents hardware simulations and resource usage results. The results for the modular design flow for partial reconfiguration are described in the next section. The physical assembly of the entire system as a stand-alone reconfigurable node, and the results for the real-time data transfer are presented in the final section of this chapter.

5.2 System Modeling

As stated in the previous chapter, the *System Generator* modeling tool is useful in designing DSP systems for FPGAs using the visual data flow environment. This section presents the modeling of the entire transmitter and receiver baseband blocks of the radio using the above mentioned tool. The baseband processing algorithms, their integrations

and bit-level testing of the designs were done using this tool before implementing the same on the FPGAs.

5.2.1 Transmitter Link Modeling

Figure 5.1 shows the modeling of the transmitter baseband sub-sections. The link was modeled for Virtex XCV1000 Xilinx FPGA. The floating point data, representing the data received from the network layer, is obtained from an input file. This data is first converted into an 8-bit fixed point format for the actual hardware modeling of the design.

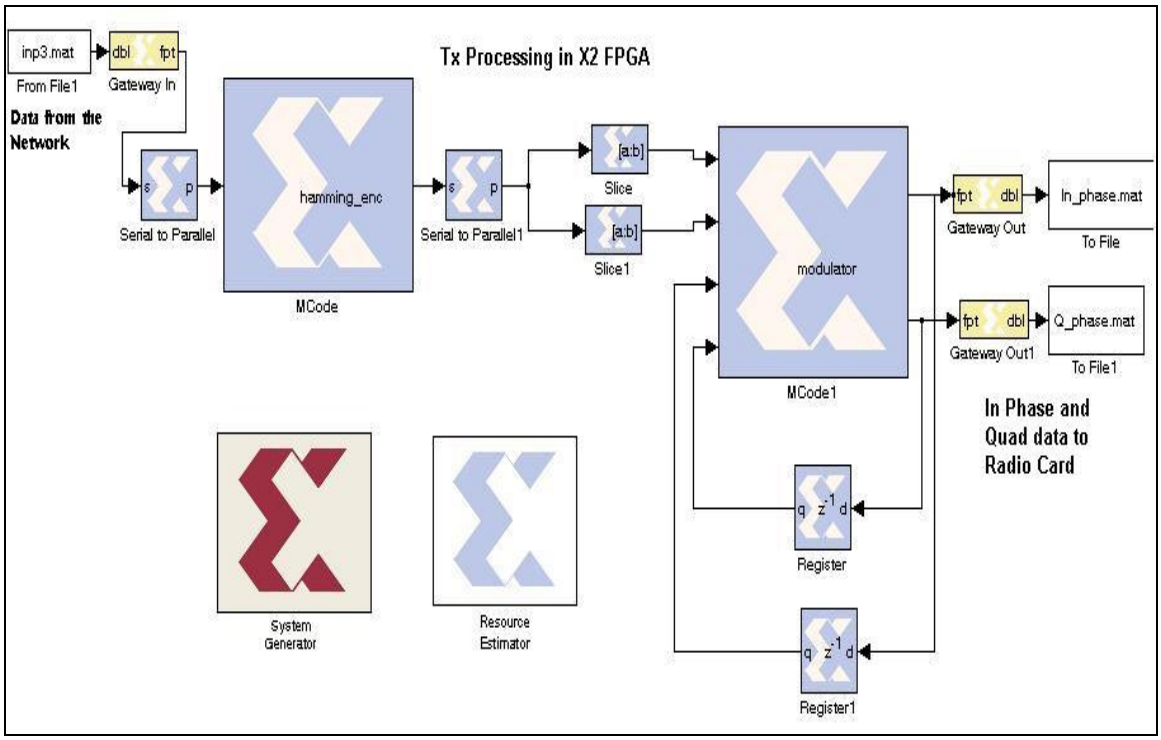

Figure 5.1: Transmitter Link Modeling

The serial data stream is passed through a serial-to-parallel converter to generate 4-bit words as the input for the block encoder. The block encoder is implemented as a MATLAB function that describes the look-up-table (LUT) for channel encoding. The *M code* block, shown in the figure, incorporates this MATLAB function. The output of the encoder passes through a serial-to-parallel converter that generates two separate *I* and *Q*

channels. The *slice* blocks separate the MSB and the LSB from the 2-bit output of the converter. The two channels are then passed on to another *M code* block that incorporates the interleaver and a state machine for the modulator. The interleaving operation is performed on both the channels and the output is passed to the $\pi/4$ DQPSK modulator state machine. The modulator works on the present *I* and *Q* inputs and the previous outputs to produce the differentially modulated output bits. The 16-bit fixed point *I* and *Q* data are converted back to the floating point format and stored in a file.

5.2.2 Receiver Link Modeling

The receiver link, shown in Figure 5.2, accepts the *I* and *Q* data streams from two input-files and converts the same into a 16-bit fixed point format. The demodulator, the first block in the receiver link, performs differential detection by using a present and a previous set of in-phase and quadrature bits. The demodulation is performed according to the equations (2.12a) and (2.12b) described in Chapter 2. The outputs from *AddSub* blocks are fed to the *M code* block that decides the sign for the *I* and *Q* bits and performs the de-interleaving operation on the resulting bits. The parallel data streams are made into a single serial stream using the *Concat* block. The serial data stream in formatted into a 7-bit word for the channel decoding operation. The decoder extracts the 4-bit information and stores in the output file in the floating point format.

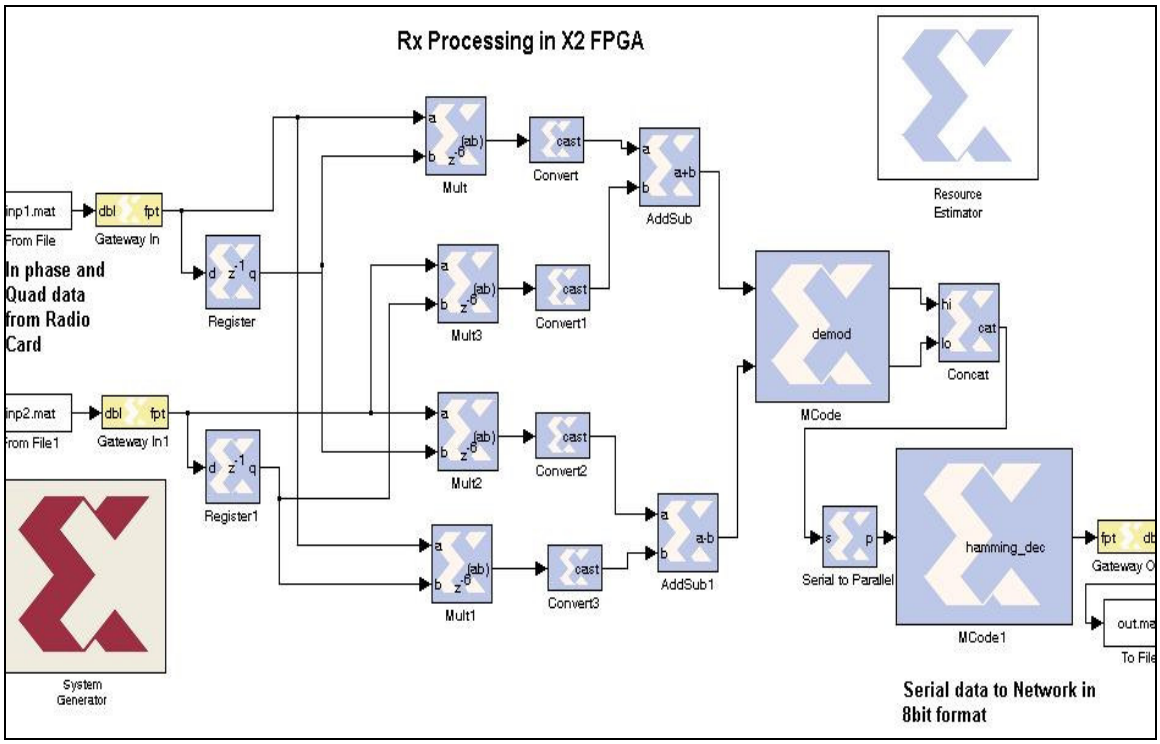

Figure 5.2: Receiver Link Modeling

The outputs obtained from the receiver link were compared with the input data file for the transmitter link to check for the accuracy of the design. The *Resource Estimator* block was used to obtain the approximate hardware resource usage by the designs. The combined transmitter and receiver links were found to use about 3 to 4 % resources of the target Virtex FPGA that did not include the resources needed by the functions incorporated in the *M code* block.

5.3 FPGA Implementations

Following the system modeling and functional verifications, the physical layer of the radio was implemented on the FPGA platform. The transmitter and receiver links were implemented in the loop-back scenario for different modulations schemes, and logic and timing simulations were performed to test the FIFO interface between the radio and the network. The sub-sections in both the links were integrated in a form of a state machine to perform various baseband operations.

5.3.1 State Machines

The transmitter state machine, involving five states is shown in Figure 5.3. State 0 checks the status of the FIFO using the *ready_out* signal and accepts the data if the *fifo_empty* signal is low. The received data is passed on to State 1 that implements (7, 4) block encoding. The 14-bit output, corresponding to two 4-bit input words, is passed to the interleaving state. Interleaving provides a 24-bit output after permuting the encoded data bits obtained from State 1. State 3 performs baseband modulation after separating the received data bits into two channels and feeds the output in a combined 32-bit format to the next state. The last state sends the *ready_out* signal whenever the valid data is ready to be sent out.

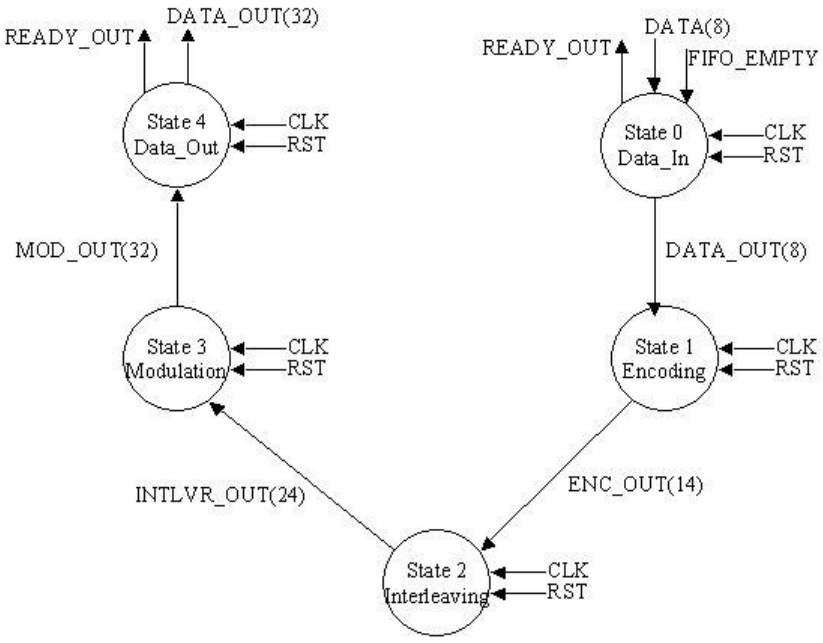

Figure 5.3: Transmitter State Machine

The *ready_out* signal from the last state of the transmitter state machine acts as the *ready_in* signal for the first state of the receiver state machine, shown in Figure 5.4. The available data flows to the demodulation state, which provides a 24-bit output to the deinterleaving state. This state recovers the original sequence back from permuted input bits and passes the data in a 14-bit format to the decoder state. The decoder state extracts

the message signal after removing the parity bits and provides the output in an 8-bit format. The data is written into the FIFO whenever the *ready_out* is high and *fifo_full* signal is low. The simulation results for these state machines are presented in the following section.

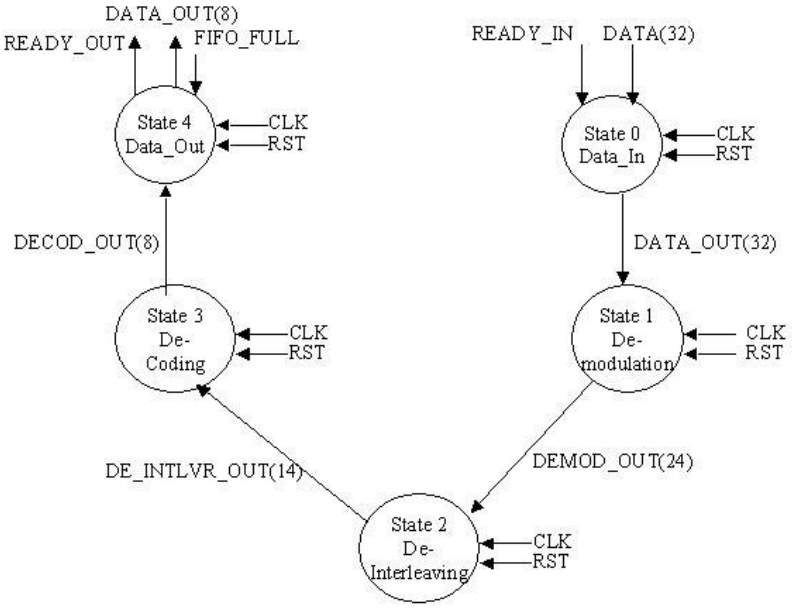

Figure 5.4: Receiver State Machine

5.3.2 Simulation Results

Pre-synthesis simulations of the above described state machines were performed for testing the designs on a Virtex-II device. The simulations were performed for two different modulations and demodulations schemes. In the first case, $\pi/4$ DQPSK scheme was used along with channel encoding and convolutional interleaving. As explained in the previous section, the transmitter accepts the data from FIFO interfaced with the network and pass it to the receiver after encoding, interleaving and modulating the network data. The receiver performs reverse operations and extracts the message back and provides it to the receiver FIFO.

Figure 5.5 shows the simulations results for the above discussed case with a 33 MHz clock. In the figure *din* represents the input data from the network FIFO and *dout* represents the output data to the network FIFO. As network and radio work at different clock speed FIFO interface provides the necessary synchronization between the two. As evident from the figure, the transmitter does not try to read the data from the FIFO when the *fifo_empty* signal goes high and receiver checks for the status of *fifo_full* signal before writing the data into the FIFO. The received data frame (*dout*) was found to exactly match the transmitted frame (*din*) with the latency of 14 clock cycles between them. The latency was mainly due to the delays introduced by the various signal processing algorithms along the two links.

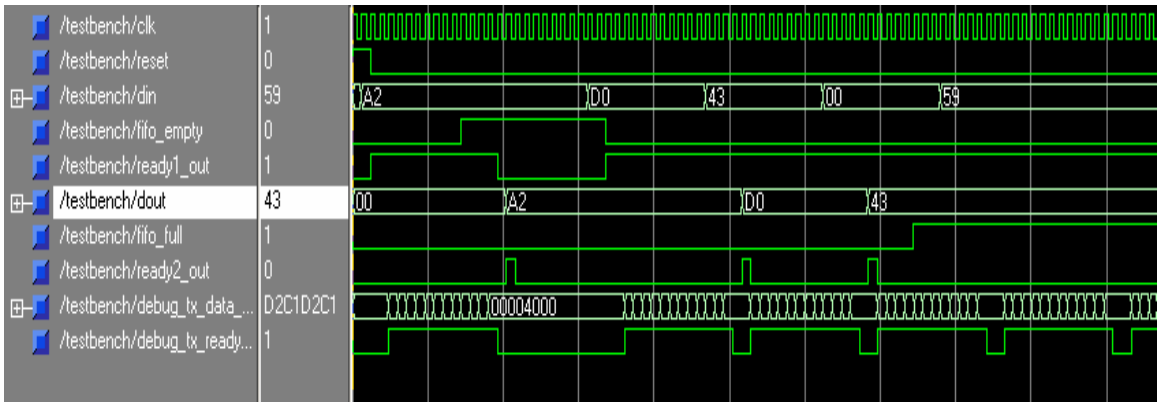

Figure 5.5: DQPSK Simulation Results

The above simulations were repeated using QPSK modulation type, which involved simple mapping of the input data stream to in-phase and quadrature data according to gray scaling. The encoding and interleaving schemes were kept the same as in the previous case. In this case the transmitted data takes only two values of either +1(0x7FFF) or –1 (0x8001). This simulation was tested for FIFO empty and FIFO full cases to check the timing synchronizations between network and radio card. The result is shown in Figure 5.6.

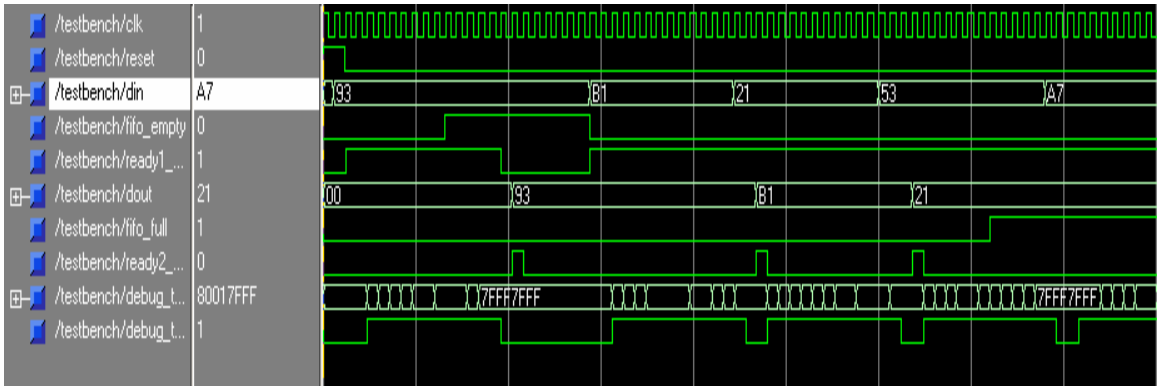

Figure 5.6: QPSK Simulation Results

As in the previous case, the transmitted and recovered data frames were found to be the same and clock latency between the two was found to be equal to 12 clock cycles. The lower latency compared to the previous case was due to fact that the QPSK modulation and demodulations do not require the time consuming multiplication operations as seen in the DQPSK scheme.

The hardware resource utilizations for both the cases are listed in Table 5-1. The results shown are for the Virtex-II FPGA. Because of the simple modulation and demodulation schemes the QPSK case requires less hardware resources compared to the computationally intensive DQPSK case.

	DQPSK	**QPSK**
Selected Device	2v4000ff1152-4	
Number of Slices	138 out of 23040	99 out of 23040
Number of Slice Flip Flops	106 out of 46080	92 out of 46080
Number of 4 input LUTs	259 out of 46080	172 out of 46080
Number of bonded IOBs	54 out of 824	54 out of 824
Number of GCLKs	1 out of 16	1 out of 16

Table 5.1: FPGA Resource Utilization for Virtex-II

5.4 Partial Reconfiguration of Radio

To make the physical layer radio architecture reconfigurable for different applications, the design was implemented on the Virtex-II device using modular design flow for partial

reconfiguration as described in Chapter 4. The design would allow switching between different baseband modulation and demodulation schemes for different applications (e.g. QPSK or $\pi/4$ DQPSK). According to the PR guidelines, the design was partitioned into two modules, namely *static* and *reconfigurable* and the top-level design incorporated these modules as black boxes. The modulator and demodulator sections of the radio in transmitter and receiver link respectively were defined as the reconfigurable modules, while the static module comprised of the rest of the sections. The routing resources across the boundary between the two modules were kept static by using the fixed bus macros acting as the routing bridge. Figure 5.7 shows the physical implementation of a bus macro that uses eight tri-state buffers to provide four bits of information exchange between the static and the reconfigurable modules.

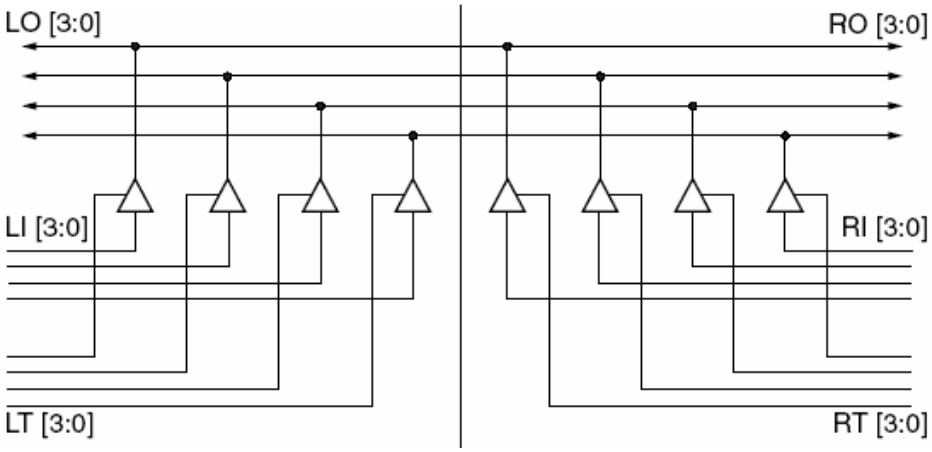

Figure 5.7: Physical Implementation of Bus Macro

The three main phases of the implementation process are described in the following sub-sections.

5.4.1 Initial Budgeting Phase

The initial budgeting operations involved floor planning for each of the modules, I/O buffers (IOBs) and all the global logic in the top-level file. The modules were placed in such as a way that each module occupied the full height of the device and covered a

minimum of four slice width on the FPGA fabric. All the IOBs were contained within the space of their associated modules and the global logic was constrained in the specific sites on the device. As a part of the first phase of implementation, the location constraints were manually inserted for each of the modules and the bus macros in the user constraint file (UCF) and checks were performed to insure that the bus macros exactly straddled the boundaries between the static and reconfigurable modules.

The output of the initial budgeting phase is the area-based floor plan for the entire design. Figure 5.8 shows the floor planning of the top-level module on the targeted XC2V4000 FPGA on the DINI platform.

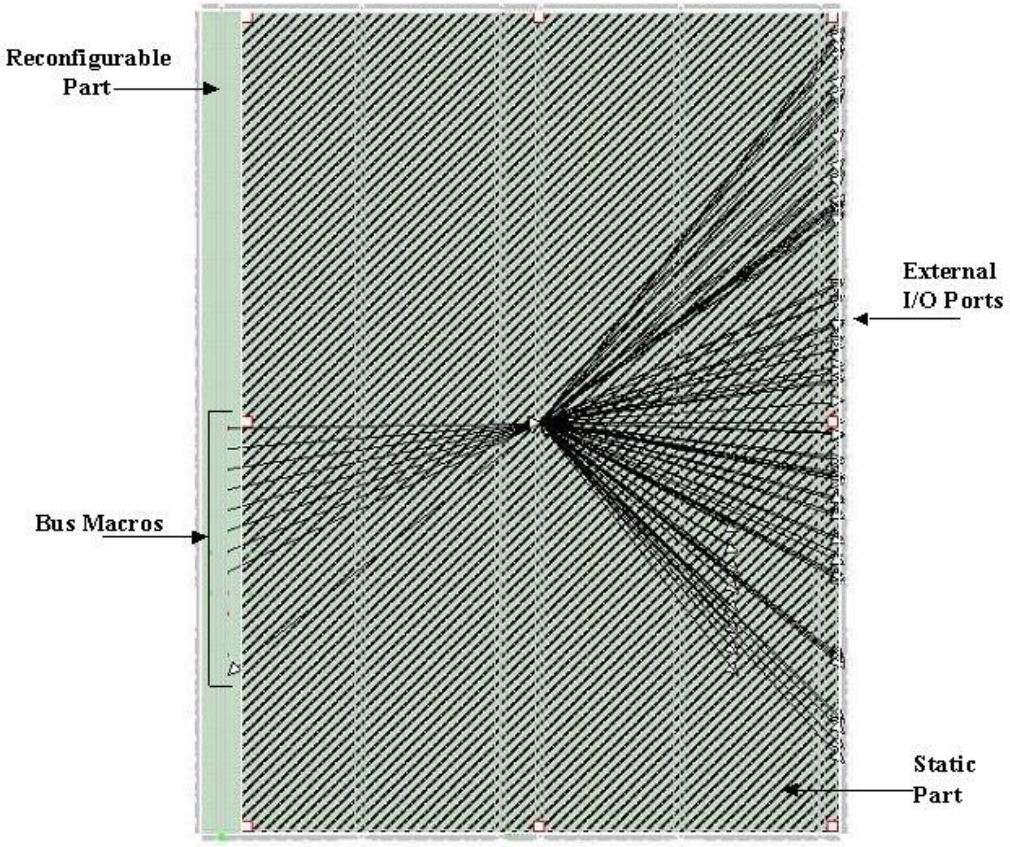

Figure 5.8: Modular Design Floor-planning

The reconfigurable part, that implements the state machine for the DQPSK modulator, occupies the full height and left slices on the device. This module gets replaced with the

state machine for the QPSK modulator on reconfiguration. The rest of the fabric is the part of the static module. The location constraints for the bus macros can be seen placed at the boundary of the reconfigurable module.

5.4.2 Active Module Phase

After the design was synthesized, constrained and floor-planned in the initial budgeting phase, each module was separately implemented though the place and route (PAR) process and separate bitstreams were generated for both the reconfigurable modules and for the static module in the active module phase of the design flow.

5.4.3 Final Assembly Phase

In the final phase of design flow each of the individual modules were assembled back to create a complete FPGA design. The PARs implemented during active module phase were preserved, thus maintaining the performance of each of the modules. This phase created a full bitstream for the device required for initial device load.

The place and route result for transmitter part of the radio after final assembly is shown in Figure 5.9. The final routed design has three distinct modules – Reconfigurable Part, Static Part and DCM that provides a common clock signal for the design. The static part communicates with the outside world through I/O ports of the device and with the reconfigurable part through the bus macros.

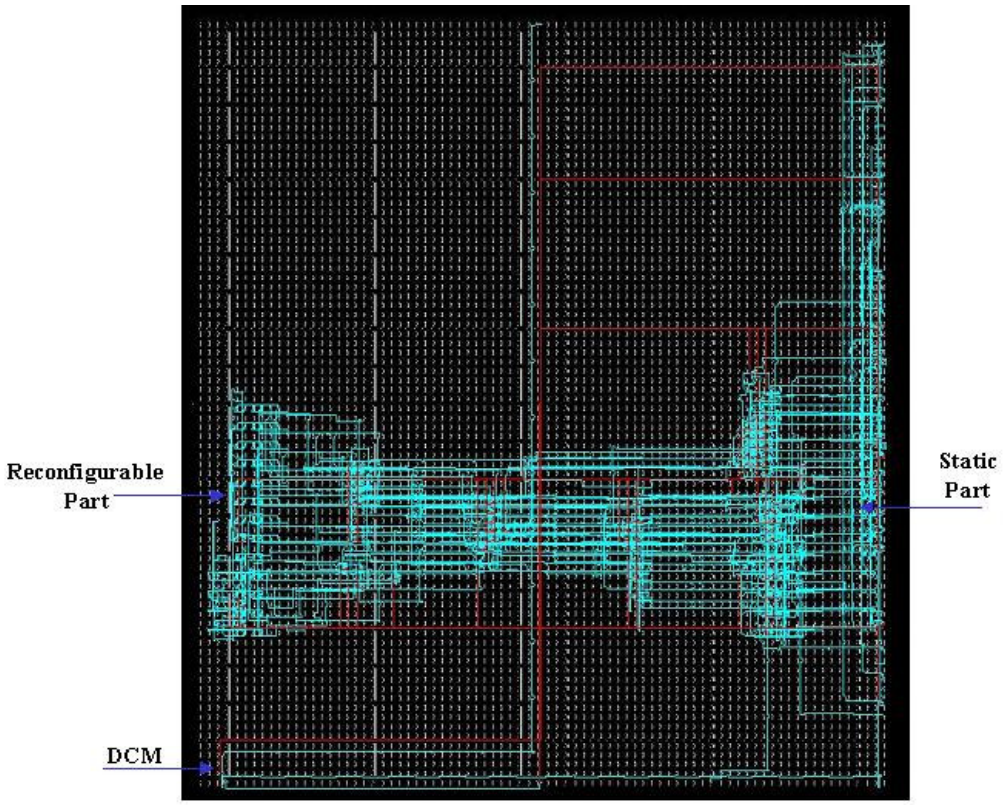

Figure 5.9: Routed Design after Final Assembly

The full bitstream generated in the final assembly phase was used to program the device on power-up and the partial bitstreams, generated in the active module phase, were used to reconfigure the associated reconfigurable module on the device.

5.5 Stand-alone Node Development

The designs described so far were implemented in a host-machine (PC) for testing and verification purposes. As the final step of the project, the FPGA platform and the soft-radio card were assembled as a stand-alone node outside the host-machine for real-time implementations and data transfers. This was achieved by using a custom made PCI back-plane[4] to host the SLAAC1V board, the radio card and the network card. The radio transceiver, which is a PCI Mezzanine card (PMC), was connected to the PCI back-plane

[4] The PCI back-plane for the stand-alone node was designed and fabricated by Aditya Chaubal

using a PCI-to-PMC carrier card. The physical assembly of the stand alone node is shown in the Figure 5.11. The FPGA platform and the radio card can be seen attached to the PCI back-plane. The network card is installed on the other side of the SLAAC1V board (not seen in the Figure). The *user-processing* and *control* FPGAs on the SLAAC1V platform were programmed using JTAG parallel cables, as seen in the Figure, however, the design was modified to program the FPGAs over the network using the configuration files obtained from the bit-stream server.

As mentioned before, the core for baseband processing is implemented in the *user-processing* FPGA (X2) of SLAAC1V, while the communication between the soft-radio transceiver and FPGA platform is through the *control* FPGA (X1). The data transfer between *user-processing* and *control* FPGAs takes place through their SRAM bank switching. Since only one of the memory banks of X1 can be switched at a time with that of X2, there needs bus arbitration logic to share the clock cycles to support both read and write operations on a single port memory bank for bi-directional data transfers. Figure 5.10 shows the block diagram describing the handshake between the two FPGAs. Memory bank swapping takes place when both the memories are completely read and written with the new data. This allows the transfer of data between radio and network through X1 FPGA. Finally the output from the soft-radio card is fed to the RF front end for up-conversion. The front end comprises of the mixer and a local oscillator that generates the carrier at 2.04 GHz.

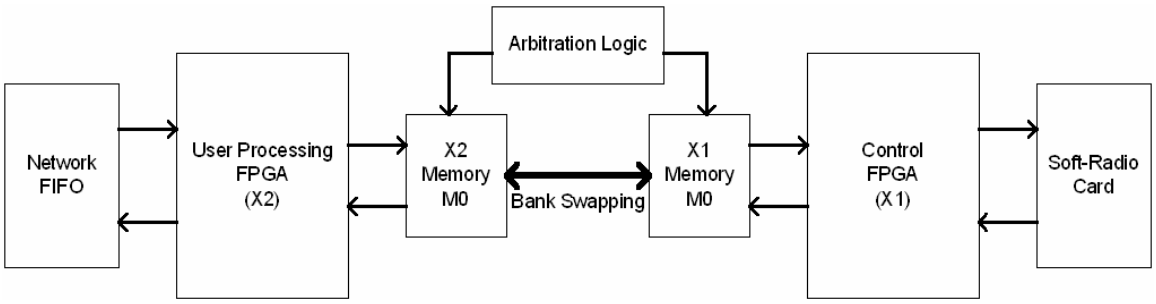

Figure 5.10: Block diagram, showing handshaking between two FPGAs

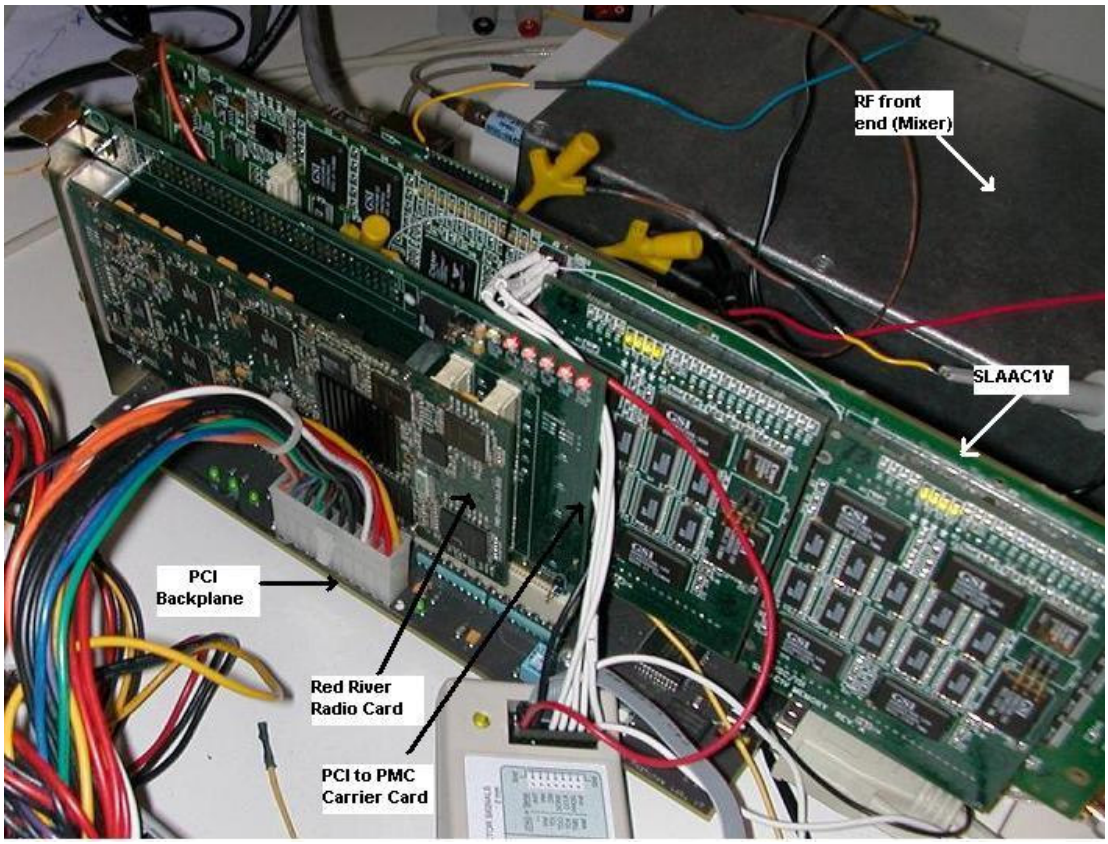

Figure 5.11: Physical Assembly of Stand-Alone Node

The data transfer between the transmitter and receiver nodes was established using a pair of Four-point antennas designed and built by Virginia Tech Antenna Group [30]. The antenna, shown in Figure 5.12 can support bi-directional signal in frequency range from 1.5 GHz to 3 GHz with 30dB isolation between *E*-plane and *H*-plane. The antenna gain obtained at 2.04 GHz is around 8dBi.

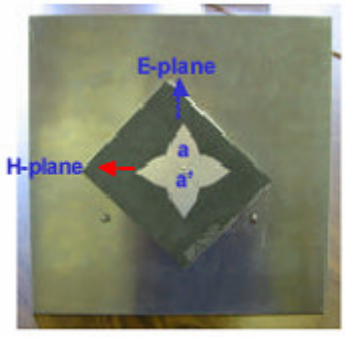

Figure 5.12: Transmitter/Receiver Antenna

5.5.1 Implementation Results

The primary aim of this project was to make the reconfigurable radio communicate at high data rates with minimum inter-symbol interference (ISI) and lowest BER. To demonstrate the effectiveness of the implemented designs, the tests were performed using a loop-back scenario, wherein the signals transmitted by the antenna were received by the receiving antenna attached to the same node. The recovered data was compared with the transmitted message signal to calculate the BER and supportable data rates. The radio was configured to imitate a single GSM channel supporting data rates up to 270 kbps. GSM uses GMSK modulation scheme to support data rate up to 270 kbps. As explained in Chapter 2, GMSK can be thought of as a gaussian filtered frequency modulated scheme with minimum frequency deviation. In this scenario the transmitter section in the soft-radio card was configured in pre-filtering FM modulation mode and the FIR filters were designed to provide the required gaussian filtering before the modulation. In the receiver section the SAW filters and the FIR filters were designed to support the rate of 1.08 Msps, thus sampling at four times higher rate than the transmitter in order to avoid the aliasing effects. This was done by designing the filters in MATLAB and using the coefficients in the soft-radio configuration tool. The C header files generated by the tool were loaded onto the FPGA to configure the system on initialization.

Feedback tests were performed on the radio to check the performance in terms of ISI and BER. For test purposes the transmitter DMA buffer was set for 8K samples and receiver buffer for 64K samples. To achieve proper clock synchronization between the transmitter and receiver sections of the radio, the transmitter was first made to transmit 0.5K samples of clock followed by 7.5K samples of data, which consisted of repeated frame of [-1 –1 1 1 –1]. Figure 5.13 (a) shows the transmitted clock signal and the transmitted data frame is shown in Figure 5.13 (b).

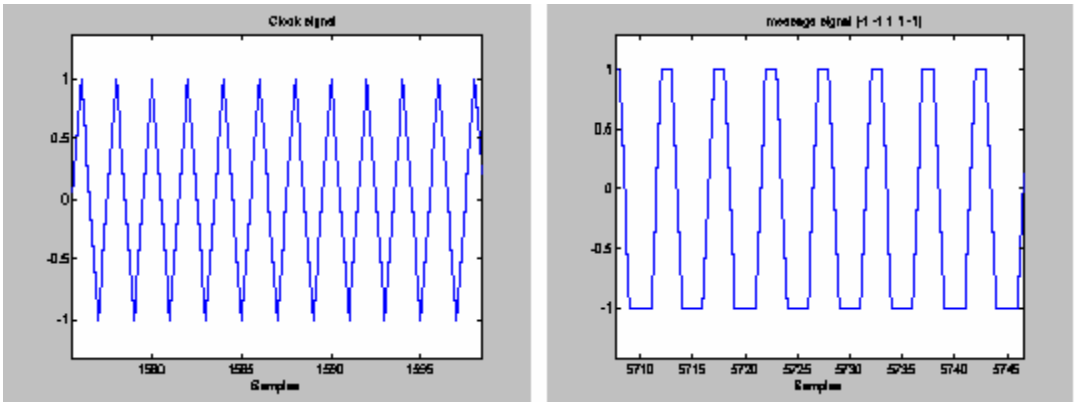

Figure 5.13: (a) GMSK Transmitted Clock Signal, (b) Transmitted Data Frame

The modulated data at intermediate frequency of 70 MHz (± 5MHz) as seen at the output of the soft-radio card on a spectrum analyzer is show in the Figures 5.14 and 5.15. Figure 5.14 shows the spectrum of the transmitted signal when a regular pattern of clock signal is transmitted. The spectrum of clock signal looks symmetric is shape with the highest peak at the center frequency of 71.8MHz. Figure 5.15 shows the spectrum of the carrier signal modulated by a GMSK waveform. This signal was up-converted to 2.04GHz after passing through RF front end.

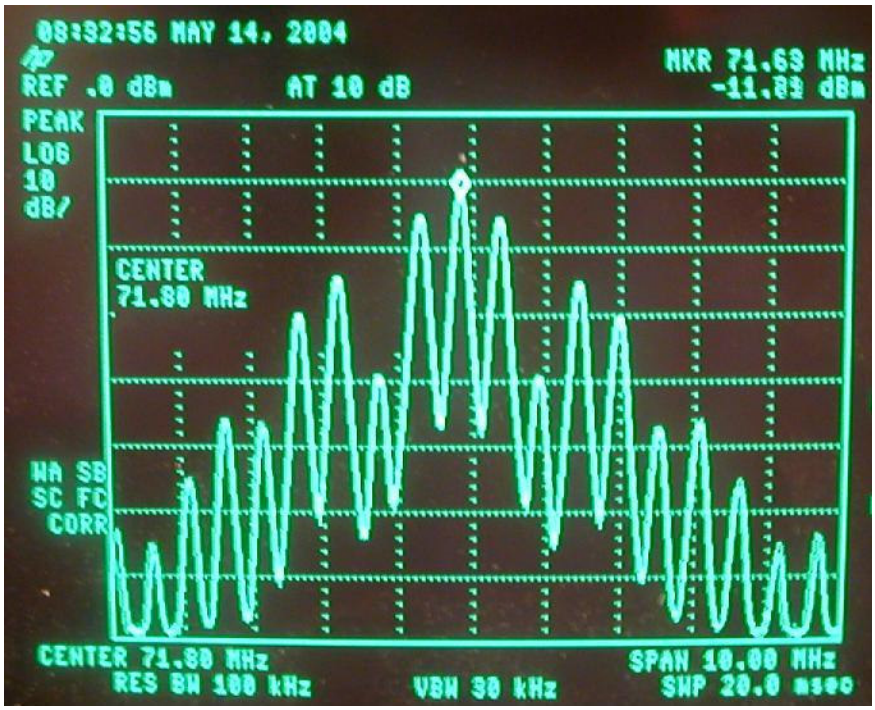

Figure 5.14: GMSK Modulated Clock Signal

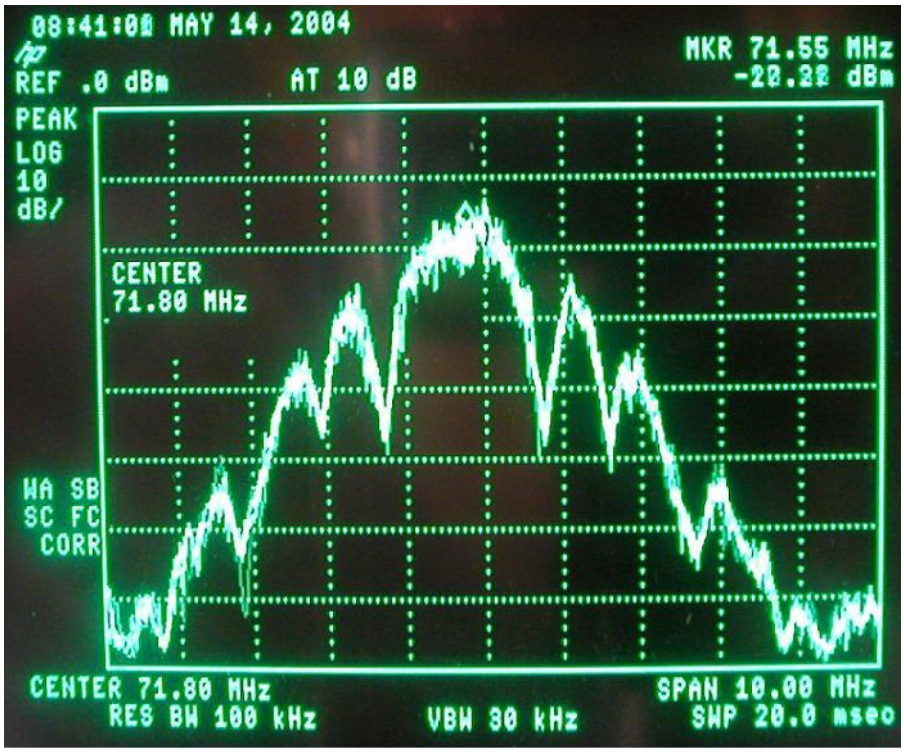

Figure 5.15: GMSK Modulated Random Data

The 64k samples as received by the receiver buffer after down conversion to baseband are shown in Figure 5.16, wherein the data samples can be seen preceded by the clock samples. The received samples were then down sampled and converted into the message signal consisting of NRZ data after performing baseband processing operations. Figure 5.17 (a) and (b) show the recovered clock and data frames respectively. It was found that data was retrieved with almost full fidelity. BER was on an average about 4/65536.

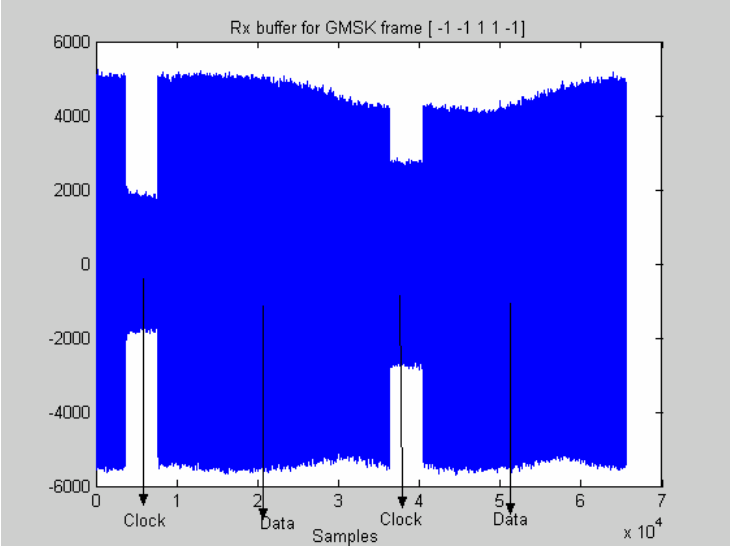

Figure 5.16: GMSK Received 64k Samples

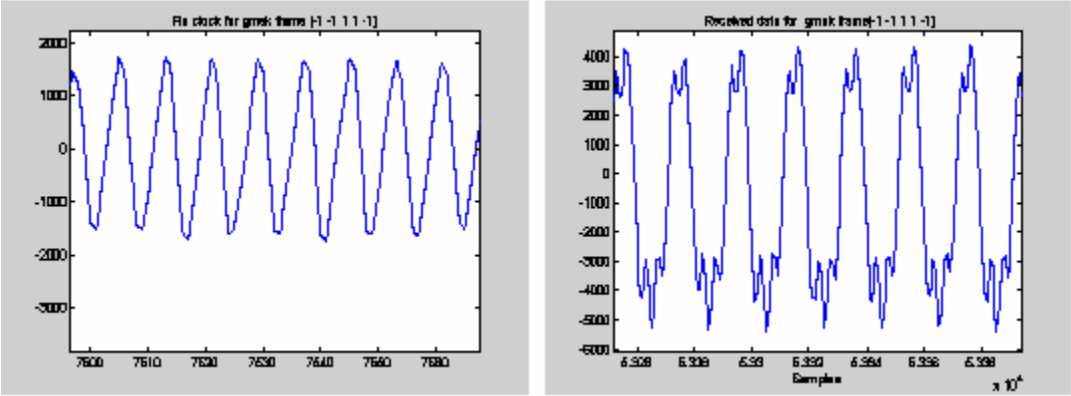

Figure 5.17: (a) GMSK Recovered Clock Signal, (b) GMSK Recovered Data Frame

Similar tests were performed for FSK modulation scheme where the transmitter sampling rate was kept 125 kbps and receiver sampling rate at 1Msps, thus sampling at eight times higher rate than the transmitter. Keeping the buffer sizes same as in the previous case, one set of data buffer was received with clock followed by the actual data. The transmitted clock and data frame for FSK modulation are shown in Figures 5.18 (a) and (b), while the received samples and recovered data frame are shown in the Figures 5.19 (a) and (b) respectively. The NRZ transmitted message was extracted from the recovered frame by detecting the peaks in the signal. The bit error in this case was found to be about 2 samples in the entire 64K data buffer.

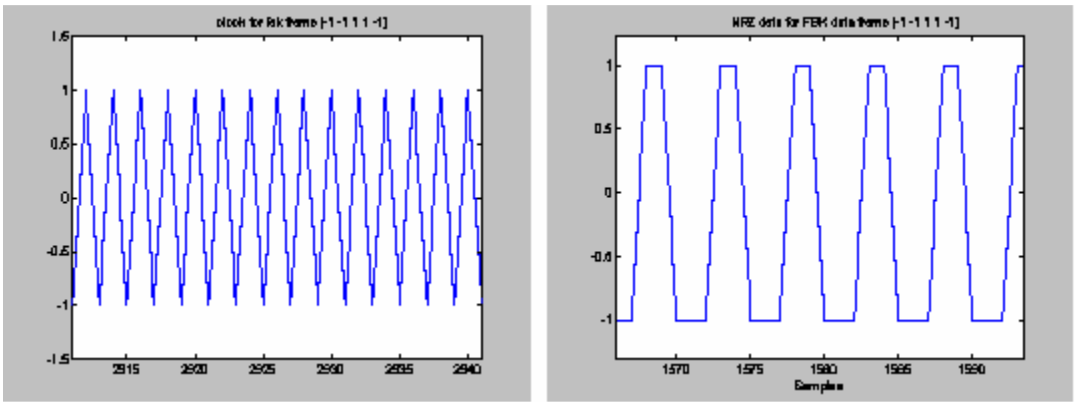

Figure 5.18: (a) FSK Transmitted Clock, (b) FSK Transmitted Data Frame

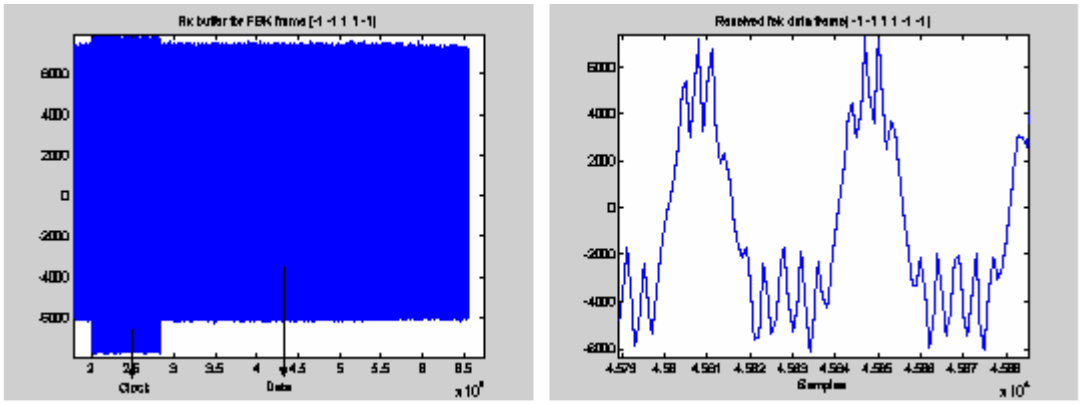

Figure 5.19: (a) FSK Received 64k Samples, (b) FSK Recovered Data Frame

Chapter 6

6. Conclusions

6.1 Summary

This thesis detailed the design and implementation of the physical layer of a reconfigurable radio. The research investigated the use of advanced FPGA platforms in the development of partially reconfigurable radio architecture. Chapter 1 compared various hardware options available for implementing baseband processing functionalities of a software radio. The motivation for using FPGAs-based configurable architecture over the contemporary DSPs and ASICs was elaborated in this chapter.

Chapter 2 provided the background in SDR and highlighted their advantages in the design of multifunctional and multi-mode radios. The chapter also described in detail the architecture of the Virtex and Virtex-II devices used in the Xilinx FPGAs. The theory behind various baseband processing algorithms was discussed in this chapter. It provided the in-depth look into several popular channel coding, interleaving and modulation schemes.

Chapter 3 provided the insight into SLAAC1V and DINI FPGA platforms that host Virtex and Virtex-II devices respectively. The architecture, capabilities, and advantages of these two platforms were presented and compared. The characteristics and

specifications of the soft-radio transceiver card, that was used to implement the IF section of the radio, were also described in this chapter. The programmable transceiver allows digital up-conversion and down-conversion of the data between baseband and IF.

Chapter 4 first presented the system design and detailed the transmitter and receiver link designs of the baseband sections of the radio. The chapter then introduced *System Generator*, the system modeling tool used for simulating the hardware models in the MATLAB/*Simulink* visual dataflow environment. Next, two different design flows for FPGA implementations were described. The regular design flow allows configuration of the entire device, while the modular design flow offers the flexibility of reconfiguring only the user-defined parts of the device. This advanced feature, available in the Virtex devices, provides the potentials for configurable computing.

The implementation results and analysis were presented in Chapter 5. The chapter discussed the results for the functional simulations performed using *System Generator* tool. The transmitter and receiver state machines used for FPGA implementations, along with their pre-synthesis simulations and hardware resource usages were presented in this chapter. The results for partial reconfiguration of the radio on the DINI platform were demonstrated and the important implementation steps involved in the modular design flow were highlighted. Lastly, the chapter discussed the physical assembly of the stand-alone reconfigurable node and presented the results for over-the-air data transfers for a loop-back scenario.

The main contributions of this effort were the design and modeling of the baseband radio using *System Generator* toolset, implementation of partially reconfigurable radio architecture on an FPGA-based platform and development of a stand-alone reconfigurable node for over-the-air data transfers.

6.2 Future Work

This research effort creates several opportunities for future investigations. The single channel radio design that was implemented as a part of this research could be extended to a multi-channel design to boast the data rates of the system. The signal time stamp facilities available in the latest soft-radio transceiver card could be exploited in order to make a multiple access system capable of handling more than one user simultaneously. The BER performance of the radio could be improved significantly by implementing complex but more effective channel encoding and decoding schemes. Higher bandwidth efficiency could be achieved by adopting M-ary and phase-amplitude modulation schemes such as M-PSK and QAM etc. The partial reconfiguration capabilities of the FPGAs could be further exploited by moving more parts of the radio from static to reconfigurable sections, thus making the architecture more flexible and user controllable. The FPGA-based radio structure presented in this thesis could be used to implement various digital signal processing systems for industry level applications.

6.3 Conclusions

This thesis has demonstrated a practical approach for implementing the baseband radio architecture for reconfigurable platforms. The FPGA-based radio is designed and implemented to perform the essential physical layer functions involved in the wireless data transfer applications. The implemented single user radio design is based on the modular design flow that offers efficient use of the hardware resources and runtime reconfiguration capabilities. Furthermore, it provides scalability to increase the capacity and data rates of the system, and flexibility to modify the functionalities of the radio.

The results for the experiments conducted using the implemented design have been presented for various linear and non-linear modulation and demodulation schemes. While QPSK offers a simple and least complex solution, it is susceptible to the variations in the signal spectrum due to large phase transients. On the other hand, $\pi/4$ DQPSK scheme preserves constant envelope property better than QPSK and can be differentially detected, but only at the cost of the computational load and hardware resources due to its

complexity. A non-linear technique such as GMSK results in the compact signal spectrum but it sacrifices the error rate caused by the filtering in exchange of good power and spectral efficiencies.

Glossary of Acronyms

ACI	Adjacent Channel Interference
API	Application Program Interface
ARQ	Automatic Repeat Request
ASIC	Application Specific Integrated Circuit
BER	Bit Error Rate
BRAM	Block Random Access Memory
CDMA	Code Division Multiple Access
CLB	Configurable Logic Block
DCM	Digital Clock Manager
DDS	Direct Digital Synthesis
DLL	Delay Lock Loop
DMA	Direct Memory Access
DQPSK	Differential Quadrature Phase Shift Keying
DSP	Digital Signal Processing/Processor
EDIF	Electronic Design Interface Format
FCC	Federal Communication Commission
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FSK	Frequency Shift Keying
GMSK	Gaussian Minimum Shift Keying

GPP	General Purpose Processor
GSM	Global System for Mobile Communications
HDL	Hardware Description Language
IF	Intermediate Frequency
IIR	Infinite Impulse Response
IOB	Input Output Block
ISI	Inter Symbol Interference
LNA	Low Noise Amplifier
LUT	Look Up Table
NRZ	Non Return to Zero
OQPSK	Offset Quadrature Phase Shift Keying
PAR	Place and Route
PCI	Peripheral Component Interconnect
PCS	Personal Communication System
PMC	PCI Mezzanine Card
PR	Partial Reconfiguration
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
QPDC	Quadrature Programmable Down Converter
QPSK	Quadrature Phase Shift Keying
QPUC	Quadrature Programmable Up Converter
RAM	Random Access Memory
RF	Radio Frequency
SAW	Surface Acoustic Wave
SDK	Software Development Kit
SDR	Software Defined Radio
SLAAC	System Level Application of Adaptive Computing
SNR	Signal to Noise Ratio
TCM	Trellis Coded Modulation
TCXO	Temperature Controlled Crystal Oscillator
TDMA	Time Division Multiple Access

UCF	User Constraint File
UMTS	Universal Mobile Telecommunication System
VHDL	Very High Speed Integrated Circuit (VHSIC) Hardware Description Language
ZBT	Zero Bus Turnaround

Appendix A: Transceiver Header Files

The parts of the configuration files, as obtained from the Waveform configuration tool, used for programming the soft-radio transceiver card are included in this appendix. The files display some of the various user-controlled parameters that can be modified to change the functionalities of the card.

Transmitter Parameters

```
SbAddressValuePair_t XmitConfig_s [6304] = {
  0x000101FC,0x00000002,    /* DDC1 Hardware Reset(For Channel Sync) */
  0x00010030,0x00001000,    /* Main Control uP HOLD (to load coeffs) */
  0x00010050,0x00000000,    /* Gain Profile Coef 1 (RAM Data)        */
  0x00010050,0x0000001C,    /* Channel 0 Real Shaping Coef 1(RAMData)*/
  0x00010008,0x00000000,    /* Fixed Integer Divider<31:16>  MSW     */
  0x0001001C,0x00000000,    /* Carrier freq static phase offset      */
  0x00010028,0x00007FFF,    /* Gain <14:0>                           */
  0x00010034,0x000060D1,    /* FIR control <15:0>                    */
  0x00010040,0x00000008,    /* Polarity Control<3:0>                 */
  0x00010048,0x00000000,    /* I Serial Time Slot <9:0>              */
  0x0001004C,0x00000000,    /* Q Serial Time Slot <9:0>              */
};
SbConfig_s TxConfig_s={"Radio Configuration",0,0,6304,XmitConfig_s};
```

Receiver Parameters

```
SbAddressValuePair_t RecvConfig_s [2851] = {
  0x0007E000,0x00000000,    /* Test Control Register <31:0>          */
  0x0007E004,0x0000700F,    /* Bus Routing Control Register <23:0>   */
```

```
  0x0007E008,0x7F7F7F7F,    /* Reset/Sync/Interrupt Source Register  */
  0x0007E00C,0x00000009,    /* Serial Clock Control Register <5:0>   */
  0x0007E014,0x001800FF,    /* Input Level Detector Configuration Reg*/
  0x0007E068,0x00000012,    /* uP FIFO Read Order Control Register 0 */
  0x00040400,0x00000001,    /* Filter Instruction 0 - Bits 31:0      */
  0x00041100,0x005C9000,    /* Channel 0 FEC RAM 1 Coefficient 0     */
  0x00040000,0x00000800,    /* Channel Input Select/Format Registers */
  0x00040004,0x00000000,    /* PN Gain Register <15:0>               */
  0x00040008,0x0000000D,    /* CIC Decimation Factor Register        */
  0x00040010,0x00064018,    /* Carrier NCO/CIC Control Register      */
  0x0004001C,0x00000000,    /* Timing NCO Frequency Control Reg      */
  0x00040044,0x0C730C73,    /* AGC Gain Limits Register              */
};
SbConfig_s RxConfig_s ={"Radio Configuration",0,0,2851,RecvConfig_s};
```

General Purpose Board Parameters

```
SbAddressValuePair_t BdConfig_s [178] = {
  0x00000040,0x00000000,    /* Interrupt Control (enable interrupt)  */
  0x00000044,0x00000000,    /* Interrupt Mask Register               */
  0x00000048,0x00000000,    /* Rx Interrupt Mask Register            */
  0x0000004C,0x00000000,    /* Tx Interrupt Mask Register            */
  0x00000050,0x00000000,    /* Discrete Control Out                  */
  0x00000094,0x02000000,    /* DMA Ctrl[Block Size|No TxMA|no RcvMA|]*/
  0x00000400,0x00000010,    /* Rx DMA Memory Area 1 (0 to 1023)      */
  0x00000480,0x00000010,    /* Tx DMA Memory Area 1 (0 to 1023)      */
  0x00001104,0x00000000,    /* Manual DMA Receive Memory Select      */
  0x00001108,0x00750008,    /* Receive Time Control Register         */
  0x0000110C,0x001116FB,    /* Receive Epoch Offset                  */
  0x00001110,0x004CE77E,    /* Receive Frame Repetition Interval     */
  0x00001114,0xC63FFE7F,    /* Receive Time Mask 1 [IQ3|IQ2|IQ1|I]   */
  0x00001200,0x00222DFD,    /* Rx Timing Control Bank Address        */
  0x00002200,0x000124F5,    /* Tx Timing Control Bank Address        */
  0x00009000,0x00011800,    /* Rx Clock PLL Controller               */
  0x00009010,0x0000600A,    /* Tx Clock PLL Controller               */
};
SbConfig_s RadioConfig_s ={"Radio Configuration",0,0,178,BdConfig_s };
```

Bibliography

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

[2] M. Cummings, S. Haruyama, "FPGA in Software Radio," *IEEE Communications Magazine*, vol. 37, pp. 108-112, February 1999.

[3] Xilinx Inc., San Jose, CA, http://www.xilinx.com/system_generator

[4] W. H. W. Tuttlebe, "Software –defined radio: facets of a developing technology," *IEEE Personal Communications*, vol 6, pp. 38-44, April 1999.

[5] C. Bergstrom, S. Chuprun, D. Torrieri, "Adaptive spectrum exploitation using emerging software defined radios," *IEEE RAWCON 99*, pp. 113-116, August 1999.

[6] B.Z. Kobb, *Wireless Spectrum Finder*, McGraw Hill, NY, 2001.

[7] Software Defined Radio forum, http://www.sdrforum.org

[8] W. Tuttlebee (Ed.), *Software Defined Radio Enabling Technologies*, Wiley, England, 2002.

[9] Xilinx Inc., San Jose, CA, *Virtex FPGA Data Book*, 1999.

[10] Xilinx Inc., San Jose, CA, *Virtex-II FPGA Data Book*, 2002.

[11] S. Srikanteswara, J.H. Reed, P.M. Athanas, "Implementation of a reconfigurable soft radio using a layered radio architecture," *IEEE Signals, Systems and Computers*, vol. 1, pp. 360-364, November 2000.

[12] C. E. Shannon, "A Mathematical Theory of Communications," *Bell Systems Technical Journal*, vol. 27, pp. 379-423, 1948.

[13] B. Sklar, *Digital Communications Fundamentals and Applications*, Pearson Education, Upper Saddle River, NJ, 2000.

[14] I. S. Reed, X. Chen, *Error-Control Coding for Data Networks*, Kluwer Academic Publishers, Norwell, MA, 1999.

[15] I. A. Glover, P. M. Grant, *Digital Communications*, Prentice Hall, Europe, 1998.

[16] P. Elias, "Coding for Noisy Channels," *IRE Conv. Rec.*, Part 4, pp. 37-47, 1955.

[17] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans., Inf. Theory*, vol. 13, pp. 260-269, April 1967.

[18] G. Ungerboeck, "Trellis-Coded Modulation with Redundant Signal Sets: Part I and Part II," *IEEE Communications Magazine*, vol. 25, pp. 5-21, February 1987.

[19] C. Berrou, A. Glavieux, P. Thitimajshima, "Near Shannon Limit Error-Correction Coding and Decoding: Turbo Codes," *IEEE Int. Comm. Conf. (ICC)*, Geneva, pp. 1064-1070, May 1993.

[20] T. S. Rappaport, *Wireless Communications Principles and Practice (2nd Ed.)*, Pearson Education, Upper Saddle River, NJ, 2002.

[21] G. D. Forney, "Burst-Correction Codes for the Classic Bursty Channel," *IEEE Trans. Comm. Tech.*, vol. 19, pp. 772-781, October 1971.

[22] R. E. Ziemer, R. L. Peterson, *Introduction to Digital Communications 2nd Ed.*, Prentice Hall, Europe, 2001.

[23] Information Science Institute, USC, Arlington VA, http://slaac.east.isi.edu

[24] Information Science Institute, USC, Arlington, VA, *SLAAC1V VHDL Users Guide*, Release 0.1.3, 2001.

[25] Information Science Institute, USC, Arlington, VA, *SLAAC1V SDK Users Guide*, Release 0.1.3, 2001.

[26] The DINI Group, La Jolla, CA, http://www.dinigroup.com

[27] The DINI Group, La, Jolla, CA, *DN3000k10 User's Manual*, version 1.4, 2003.

[28] Red River, Richardson, TX, http://www.red-river.com

[29] Red River, Richardson, TX, *WaveRunner* Plus *Hardware Reference Manual*, 2003.

[30] Virginia Tech Antenna Group, http://antenna.ece.vt.edu

[31] J. Hwang, B. Milne, N. Shirazi, J. Strommer, "System Level Tools for DSP in FPGAs," Xilinx Inc., 2001.

[32] Xilinx Inc., San Jose, CA, *Development System Reference Guide*, User's Manual, 2001.

[33] Xilinx Inc., San Jose, CA, *Two flows for partial reconfiguration: Module based or Difference based*, Application Note, November 2003.

Vita

Nikhil Bhatia was born in New Delhi, India in 1980. He did his Bachelor of Engineering (B.E) from Nirma Institute of Technology affiliated with Gujarat University in 2002, with Electronics and Communication as his major. He joined the Bradley Department of Electrical Engineering of Virginia Tech in August 2002 as a Master's student. In November 2002 he started working at the Configurable Computing Laboratory, under the guidance of Dr. Athanas, where he researched the implementation of digital communication systems on reconfigurable architectures. His research interests include Signal Processing, Digital Communications, Digital Logic Design, and Configurable Computing. He undertook an internship with the Center for Remote Sensing, Inc., Fairfax, VA in 2004 and will join the company as a full time employee after graduation.