An FPGA Software-Defined Ultra Wideband Transceiver

Matthew Bruce Blanton

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
Computer Engineering

Dr. Peter M. Athanas, Chair
Dr. Mark T. Jones
Dr. Cameron D. Patterson

August 28, 2006
Bradley Department of Electrical and Computer Engineering
Blacksburg, Virginia

Keywords: ultra-wideband, FPGA, software defined radio, high-speed

An FPGA Software-Defined Ultra Wideband Transceiver

Matthew Bruce Blanton

(ABSTRACT)

Increasing interest in ultra-wideband (UWB) communications has engendered the need for a test bed for UWB systems. An FPGA-based software-defined radio provides both post-fabrication definition of the radio and ample parallel processing power. This thesis presents the FPGA design for a software-defined radio targeted to impulse ultra-wideband signals. The system is capable of an effective sampling frequency of up to 8 G-samples/s using time-interleaved sampling with eight 1-GHz ADCs. The system is also capable of transmitting UWB pulses using a transmitter board controlled by the FPGA. In this thesis, the FPGA design used to capture and export data from the eight ADCs is presented, along with two systems which make use of the transceiver: a pilot-based matched filter communications system, and a remote vital signs monitor.

Acknowledgements

I must first thank my advisor Dr. Peter Athanas for inviting me to join the Configurable Computing Lab and for guiding me throughout my time at the CCM Lab. Thanks also must go to my committee members, Dr. Mark Jones and Dr. Cameron Patterson, for knowledge gained from them both in the classroom and at lab.

I am indebted to my parents for all of the love and support they have given me throughout my time here at Virginia Tech. I would not be where I am today without them. Thank you also to Amy for always being there for me.

Thank you to all of my friends and colleagues at the CCM Lab for helping me learn and making this past year fun.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

As noted in [3, 4, 5], ultra-wideband communications are suitable for use in several applications. UWB is a blanket term used for systems that use large instantaneous spectral bandwidth, particularly those systems whose bandwidth is much greater than their information rate. The FCC has allocated a large spectral range for unlicensed UWB communications. This increasing interest in UWB technology has created the need for a powerful, configurable test bed for UWB applications. Specifically, a test bed for areas such as ultra-wideband communications, precision ranging, and position-location is needed. Researchers need the ability to test new modulation schemes, multiple access protocols, and applications in real time, without having to create custom systems. A system with a very high sampling frequency and significant computational power is required in order to allow for software-defined processing of UWB waveforms. In the past, work has been done to create custom RF front-ends which reduce the sampling rate for a UWB system enough to allow for conventional SDR processing [6], but there does not exist a system with a multi-gigahertz sampling rate designed for real-time software-defined processing of UWB waveforms. This thesis presents an FPGA-based software-defined radio that uses time-interleaved sampling to

digitize the received signal. An FPGA-based software-defined radio system offers the benefit of easy reconfiguration and the power of massively parallel computation. Time-interleaved sampling increases the effective sampling frequency without the need for an increase in the FPGA clock frequency and offers a considerable cost savings.

1.2 Contributions

This thesis describes a flexible transceiver that is geared towards ultra-wideband communications and offers a maximum effective sampling frequency of 8 G-samples/s. The focus of the thesis is on the FPGA design for this system. Two applications are discussed, a pilot-based matched filter receiver and a vital signs monitor. This thesis presents several contributions. Methods for ensuring the synchronization of the FPGA clocks and ADC sample data were developed, as was the base hardware needed to capture data with eight time-interleaved ADCs. An interface from a host PC to the FPGA hardware was implemented. The software required to implement the acquisition phase of the pilot-based matched filter hardware was developed. The hardware for the pilot-based matched filter was debugged and interfaced with the data capture hardware and acquisition software. A vital signs monitoring application which allowed for the remote measurement of a subject's breathing rate was targeted to the transceiver and tested.

1.3 Thesis Organization

Chapter 2 of this thesis provides background for this work, including past efforts in configurable signal processing, software-defined radio trends, and an overview of ultra-wideband technology. Chapter 3 discusses the time-interleaved sampling method used to digitize the RF input. It also reviews the major parts of the transceiver system and provides specifications for the transceiver and for the test system which was constructed previously. Chapter

4 discusses the FPGA subsystem design, including the hardware and software needed to implement the pilot-based matched filter receiver and the vital signs monitor. Chapter 5 details the results gained, including the success of the data capture hardware in receiving information from eight time-interleaved ADCs, the correct demodulation of data by the matched filter receiver, and the ability of the vital signs monitoring application to accurately measure a subject's breathing rate.

Chapter 2

Background

A high-speed yet configurable processor is needed for a high data rate software-defined radio (SDR) such as an ultra-wideband SDR. This chapter discusses previous efforts in research areas pertaining to this work, including high-speed reconfigurable signal processing, software-defined radio, and ultra-wideband communications. An overview of ultra-wideband technology and modulation schemes is also presented.

2.1 High-Speed Reconfigurable Signal Processing

The need for high-speed yet configurable digital signal processing has been addressed by researchers in the past. Some have chosen to use fine-grained, commercial off-the-shelf (COTS) FPGAs, while others have chosen to implement custom, coarse-grained architectures. One project that took the latter approach was the Dynamically Reconfigurable Architecture for Mobile Systems (DReaM). The DReaM architecture was an effort to target signal processing for mobile systems with a coarse-grained, domain-specific approach [7]. DReaM combined configurable routing with 8-bit integer operators, dual-ported RAMs, and Spreading Data Path (SDP) units to implement radio functionality. The SDP was used to perform

communication-specific tasks such as CDMA-based spreading or complex correlation used in QPSK modulation. One application for the DReaM architecture was a CDMA RAKE receiver that was able to accommodate a symbol rate of 32 M-symbols/s.

Another architecture that uses a custom coarse-grained architecture is detailed in [8]. This system combined a custom reconfigurable parallel processor with a DSP and an FPGA for configurable routing. The custom processor was the XPP-64A, which implemented an array of ALU Processing Array Elements (ALU-PAEs). The ALU-PAEs implemented a DSP-based instruction set that operated on 24-bit words. The ALE-PAEs also included dual-ported RAM and configurable routing. This architecture was targeted to a RAKE receiver and an OFDM decoder.

Other systems such as the Berkeley Emulation Engine (BEE) and BEE2 have exploited the ability of FPGAs to quickly process large amounts of data in parallel. The BEE and BEE2 platforms were designed to offer massively parallel processing of data with an emphasis on hardware emulation [9, 10]. The original BEE system was comprised of BEE Processing Units (BPUs), each of which contained 20 Xilinx Virtex-E FPGAs. Sixteen FPGAs in each BPU were used for computation and the remaining four were used as configurable crossbar switches. Tool flows using Simulink and Xilinx System Generator allowed for emulation of ASIC designs with a high level of design abstraction. One test of the BEE system was an emulation of a TDMA receiver with a 806 kHz symbol rate. This design used three processing FPGAs and one crossbar FPGA and was able to operate at a maximum frequency of 25.0 MHz [9]. A single-channel 2.4 GHz radio system tested on the BEE platform was able to operate in real-time with a 32 MHz system clock rate [11].

The BEE2 platform expanded on the BEE concept and took advantage of newer technologies. Each BEE2 compute module contained five Xilinx Virtex-II Pro FPGAs. Each FPGA was connected to four 400 MHz DDR DRAM DIMMs, giving up to 4 GB of memory per FPGA with a memory bandwidth of up to 12.8 Gbps. In each module, four FPGAs were used for computation, and one was used for control. The control FPGA had additional con-

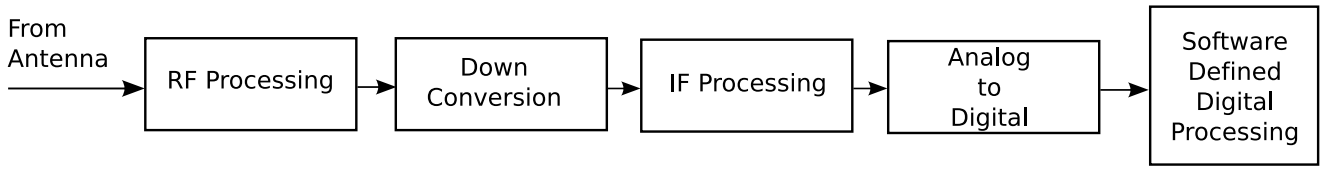
nectors for off-board communication. The FPGAs were connected together on-board with parallel LVCMOS connections and could communicate to off-board entities with high-speed serial transceivers. BEE2 was designed not only for emulation but also for implementation of high-speed DSP applications. One use of the BEE2 system was a cognitive radio test bed [12]. BEE2 performed signal processing functions for the test bed. In another application, the BEE2 processed 16 Gbps of digital data to implement a spectrometer with sub-hertz spectral resolution over 800 MHz.

While BEE2 has been shown to be useful in circuit emulation and in signal processing, there exists a drawback to using such a general computation engine. Because BEE2 was designed to use high-speed serial links for off-board communications, complex front-end boards were needed to interface with BEE2. In [12], the front end board contained a Xilinx Virtex-II Pro FPGA in addition to the analog filters, the ADC, and the DAC. The FPGA was needed for communication with the BEE2 board. A more application-specific processing system could use the FPGA connected to the ADC and DAC as the processing element, instead of offloading the processing to another system.

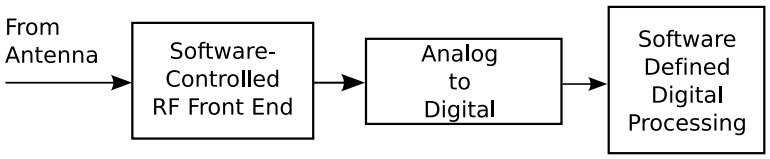
2.2 Software-Defined Radio

The goal of a software-defined radio is to move as much of the processing in the radio from fixed hardware to software, with the intention of making the function of the radio more configurable. Ideally, a software-defined radio consists of an analog-to-digital converter (ADC) connected to an antenna. The digital output is fed into a configurable computation device for processing. In practice, however, an analog front-end is necessary. RF processing and down conversion are performed in the analog domain before the ADCs [13]. Software-defined radio is evolving towards the ideal. Future SDRs might replace fixed analog hardware with an intelligent software-controlled RF front end [13].

The benefits of of a software-defined radio are evident. The operation of the radio can be



(a) Typical SDR Receiver



(b) Advanced SDR Receiver with Software-Controlled Front End

Figure 2.1: Block Diagrams of SDR Receivers

changed without significant hardware changes. As a test platform for wireless communications researchers, an SDR allows for the testing of different modulation and demodulation schemes, multiple access protocols, and any other aspect of the radio processing that can be done in the digital domain. As a final product, an SDR is attractive to several groups [14]:

- Wireless handset creators want to fix bugs more easily and to remotely re-flash devices.
- Military organizations are interested in interoperability, cost effectiveness, flexibility, and remote network management.
- Civil government and public safety groups want access to new services and new frequencies.
- Radio vendors want reduced costs and increased flexibility.
- The FCC is interested in the possibility of SDR-enabled spectrum sharing [15].

The original software-defined radio concepts came out of the defense sector [15]. One of

the first major SDR projects was SPEAKeasy. SPEAKeasy was a military undertaking with the goal of creating a "modular, reprogrammable modem with an open architecture [16]." At the end of Phase-2 for the SPEAKeasy project, the radio was able to handle AM and FM voice communications over a range of 4 MHz to 400 MHz. Both FPGAs and DSPs were used for digital processing. The units were so popular that production was initiated immediately instead of continued research and development. Other projects such as FM3TR, SoRDS, PMCS, and JCIT continued the research into SDR technology [15, 17, 18].

Recent developments in SDR have pushed for more standardization and interoperability [19]. The Joint Tactical Radio System (JTRS) is a government program which aims to replace legacy radio systems and to create a mobile ad-hoc networking program [20]. JTRS uses a open software architecture, the Software Communications Architecture (SCA), to define the communication between the various hardware and software processing elements in a software-defined radio. The SCA is designed to [21]:

- allow for portability of applications between SCA-compliant designs,
- use commercial standards,
- reduce software development time through module reuse,
- allow for evolving frameworks and architectures.

The SCA uses the Common Object Request Broker Architecture (CORBA) to allow an SCA-compliant radio system to perform its software processing across multiple processing elements. Although the SCA software is created to be run on general-purpose processors, the SCA acknowledges the use of FPGAs and DSPs as part of the digital processing in an SCA-compliant radio. Efforts are being made to standardize the interface between the general-purpose software of the SCA and specialized processors such as DSPs, FPGAs, and ASICs [22].

2.3 Ultra-Wideband Communications

A UWB signal, as defined by the FCC, is a signal with a minimum bandwidth of 500 MHz or a fractional bandwidth of at least 0.20 as measured from the -10dB emission points [23]. Fractional bandwidth is calculated as

$$2\left(\frac{f_H - f_L}{f_H + f_L}\right) \tag{2.1}$$

Where

f_H is the upper frequency of the -10dB emission point

f_L is the lower frequency of the -10dB emission point

There has been much interest shown in UWB technology. UWB has been used for low probability of detection (LPD) radar, precision location, and communications [24, 4]. The DRACO system used UWB as the underlying technology for its radio transceiver. DRACO was able to operate over a distance of up to 2km and supported encrypted or unencrypted voice and data. An FPGA was used to perform the transceiver functions. Other systems such as the ORION transceiver and the SPIDER radar altimeter also leveraged UWB [24].

Other groups are interested in UWB as a good fit for short-range communications. The WiMedia Alliance, with members including Intel, HP, Microsoft, Texas Instruments, Sony, and many others, has a mission of promoting worldwide UWB adoption and standardization [25]. UWB is the driving technology behind the Wireless USB specification [26, 27]. Wireless USB developers appreciate the ability of UWB to provide low cost and low power communications for consumer electronics. Wireless USB aims to provide 480 Mbps data rate at a range of 3 meters and 110 Mbps at 10 meters.

2.3.1 Impulse Ultra-Wideband Signals

There are two main types of UWB signals: multicarrier UWB (MC-UWB) and impulse UWB(I-UWB). Multicarrier UWB uses multiple carrier frequencies concurrently, whereas impulse UWB uses very short duration pulses. Because the transceiver described in this work is designed for use with I-UWB signals, MC-UWB is not discussed here. In the literature, the most popular I-UWB pulses are the Gaussian pulse and the derivatives thereof, so called because they are based on the Gaussian function [28]. The Gaussian pulse function is given as:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{x^2/(2\sigma^2)} \tag{2.2}$$

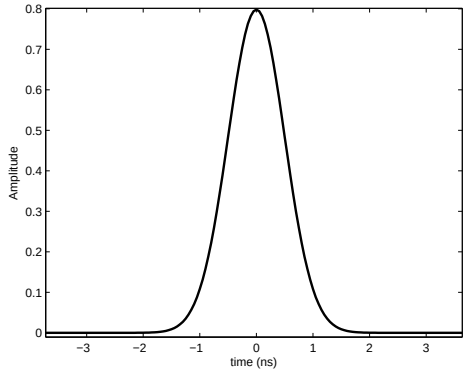
where

$\sigma = \frac{pulse\ width}{2\pi}$.

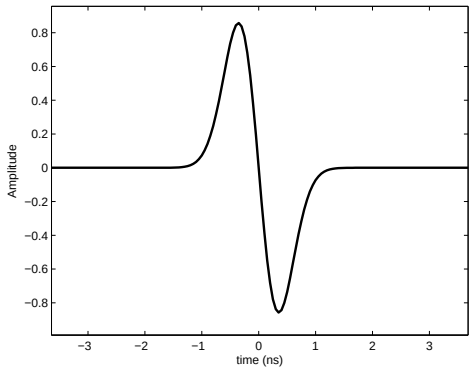
Graphs of a Gaussian pulse, a Gaussian monocycle, and a Gaussian doublet can be found in Figure 2.2. In practice, an I-UWB transmitter does not generate a true Gaussian pulse, monocycle, or doublet. Instead, an approximation to the pulse, monocycle, or doublet is generate by analog hardware. Future transmitters may use high-speed direct digital synthesis to generate pulse waveforms. For an overview of analog UWB pulse generator circuits, see [29].

2.3.2 Impulse Ultra-Wideband Modulation Schemes

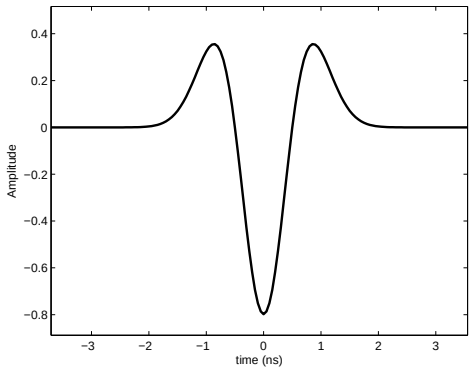
The modulation schemes available for an I-UWB system include Pulse Position Modulation (PPM), On-Off Keying (OOK), and Pulse Amplitude Modulation (PAM). With pulse position modulation, information is stored in the time at which each pulse is received. Given a known "base" arrival time for each pulse, a pulse which arrives at the base arrival time is demodulated as a "1" and a pulse which arrives before or after the base arrival time is demodulated as a "0". The base arrival time for each pulse is equal to the last pulse's base



(a) Gaussian Pulse



(b) Gaussian Monocycle



(c) Gaussian Doublet

Figure 2.2: Typical Impulse UWB Pulses

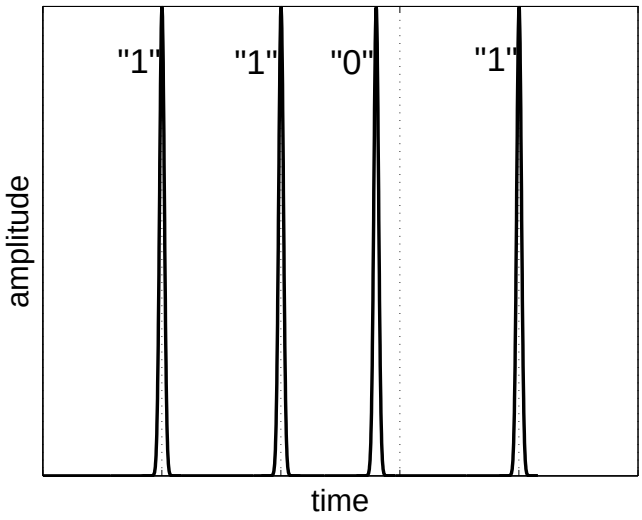


Figure 2.3: Pulse Position Modulation

arrival time plus the pulse rate. An example of PPM can be seen in Figure 2.3.

On-Off Keying is a very simple modulation scheme that stores information in the absence or presence of a pulse. For each expected pulse arrival time, if a pulse is detected, then the signal is demodulated as a "1." If no pulse is detected, then the signal is demodulated as a "0." An example of OOK can be seen in Figure 2.4.

Pulse Amplitude Modulation stores information in the amplitude of the transmitted pulse. For binary pulse amplitude modulation, either a positive or negative pulse is sent. A positive pulse is demodulated as a "1", whereas a negative pulse is demodulated as a "0." An example of PAM can be seen in Figure 2.5.

2.3.3 FCC Regulations

On February 14, 2002, the FCC adopted changes to its Part 15 rules to allow for the operation of UWB devices [23]. The Part 15 rules govern the emissions of unlicensed transmitters [30]. These rules were amended to allow for the use of UWB signals in a number of specific areas,

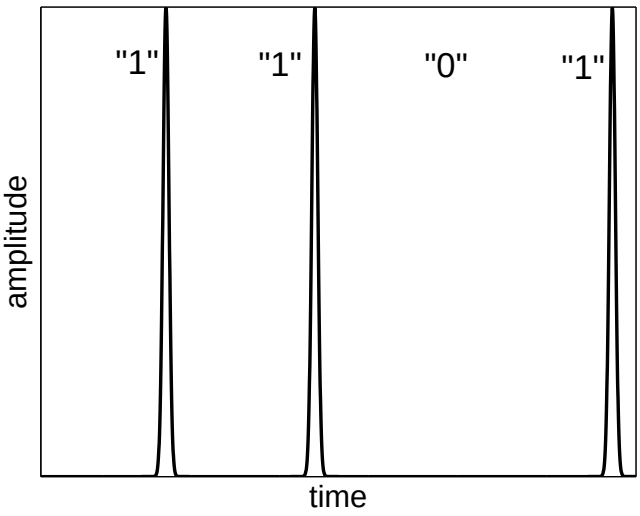


Figure 2.4: On Off Keying

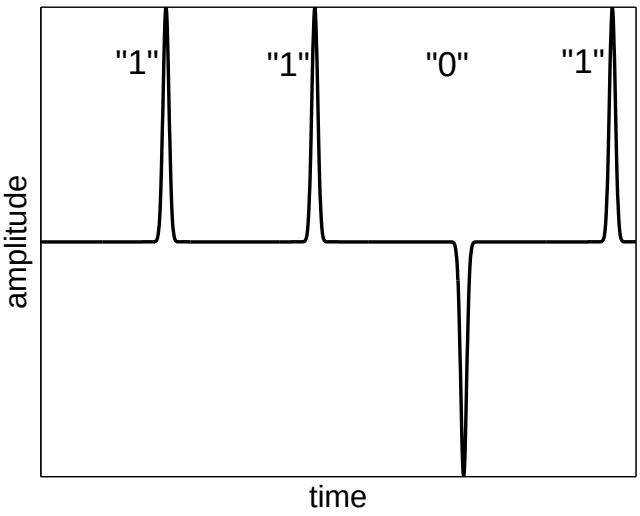


Figure 2.5: Pulse Amplitude Modulation

including:

- Imaging systems
  - Ground penetrating radar
  - Wall imaging
  - Through-wall imaging
  - Surveillance systems
  - Medical systems
- Vehicular Radar
- Communications and Measurement

The transmissions for systems in each of these areas are limited to specific frequency ranges [23]. For instance, ground penetrating radar systems and wall imaging systems must operate below 960 MHz or between 3.1 GHz and 10.6 GHz. Through-wall imaging systems can operate below 960 MHz or between 1.99 GHz and 10.6 GHz. Surveillance systems can operate in the 1.99-10.6 GHz band. Communications and measurement systems are limited to the 3.1-10.6 GHz band, as are medical systems.

2.4 Summary

Software-defined radio is a promising solution to the need for flexible radio systems. As a testbed for radio research, SDR offers the ability to perform experiments in real time without the need for application-specific hardware. SDR can be leveraged to create a flexible testbed for ultra-wideband communications, a technology which continues to gain interest from military and commercial sectors. FPGAs have been shown to be effective in signal processing tasks and are well suited to be used as the processing element in a software-defined radio.

Chapter 3

Ultra-Wideband Transceiver System

This chapter presents an overview of the UWB SDR system, including the concept of time-interleaved sampling, a discussion of the major parts of the system, and specifications for the system. The two-ADC testbed system built previously is discussed, along with the eight-ADC receiver system and the UWB transmitter system.

The main requirement that has driven the design and implementation of this Software-Defined Radio platform is the need for a flexible testbed for impulse ultra-wideband communications. To simplify the system design process, only COTS parts have been used. A Xilinx Virtex-II Pro FPGA has been chosen to handle the digital processing for the receiver. The design objectives for this system are as follows:

- The receiver should be able to handle multiple modulation schemes: Pulse Amplitude Modulation (PAM), Pulse Position Modulation (PPM), and On-Off Keying (OOK).
- The system should be able to operate at a maximum range of 10 meters.
- The receiver should be able to handle different receiver topologies: Leading Edge Detection and Matched Filter.
- Control over modulation and multiple access schemes, frame structure, and receiver

topology should be possible using software.

- The system should also be able to operate with waveforms other than UWB.

Portions of this chapter are adapted from [1] with permission by the author. For a more in-depth analysis of the design and implementation of the transceiver system, see [1].

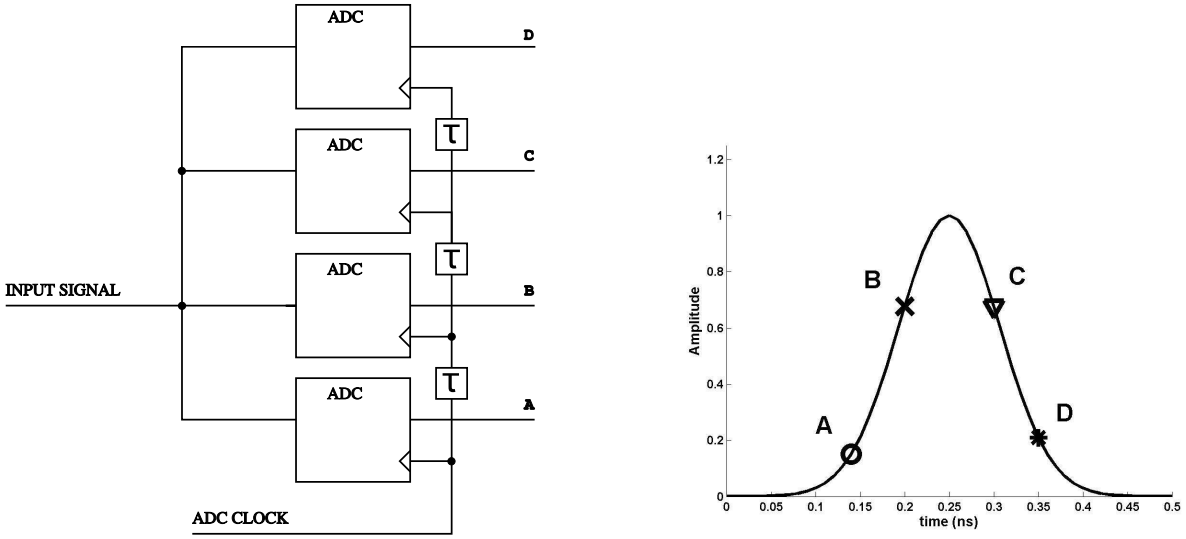
3.1 Time-Interleaved Sampling

This radio receiver platform uses Time-Interleaved (TI) sampling to digitize the analog input into the system. TI sampling uses an array of ADCs sampling the same signal consecutively at different instances in time to increase the sampling rate over a one-ADC configuration. If n ADCs are each sampling at a rate of f, then each ADC's clock signal is delayed by $\tau = \frac{1}{fn}$, as compared to the preceding ADC. Figure 3.1 illustrates the use of TI sampling with a four ADC system.

Using TI sampling increases the sampling rate without increasing the ADC clock rate. This allows for a greater data rate without an increase in the ADC-to-digital processor bus rate. Instead of one high-speed data bus, the ADCs present the processing element with multiple lower-speed data buses. TI sampling allows for easy parallel processing of the ADC data, because each ADC provides its own data stream. Both of these properties are attractive for an FPGA-based system. For multi-gigahertz sampling rates, single-ADC sampling is infeasible with an FPGA implementation without additional interface hardware.

3.2 Analog to Digital Converters

The ADC chosen for use in the radio receiver is the Maxim MAX104CHC. The MAX104CHC has a maximum sample rate of 1 G-samples/s and an input bandwidth of 2.2 GHz. This input bandwidth limits the UWB pulses that the receiver can receive to those with a bandwidth



(a) Block Diagram

(b) ADC Outputs [1]

Figure 3.1: Time-Interleaved Sampling with Four ADCs

of 2.2 GHz or less. The ADC provides its digital output in two eight-bit buses, each running at half of the sample clock rate. The clock and reset inputs are fed to each ADC through delay chips; this implements the time-interleaved sampling. Instead of connecting the delay chips in series, as in Figure 3.1(a), each ADC's delay is controlled directly by one delay chip, as shown in Figure 3.4. This allows for the correction of delay mismatches between ADCs.

The use of multiple ADCs introduces errors in the sampled signal, which would not be present in a single-ADC system. Gain or offset mismatches between ADCs distort the output. Variations in the aperture delay and the time between the ADC clock rising edge and the time at which the ADC samples the signal distort the signal. If the variation in the aperture delay between two ADCs is greater than or equal to the delay between ADC clock rising edges, then ADCs can sample at the same time or out of order. These phenomena all result in a degradation in the signal-to-noise ratio (SNR) of the receiver. These effects are illustrated in Figure 3.2. In this figure, ADC1 has ideal offset, gain, and timing characteristics while ADC2 has non-ideal characteristics. The input is a 100 Hz sine wave and each ADC samples

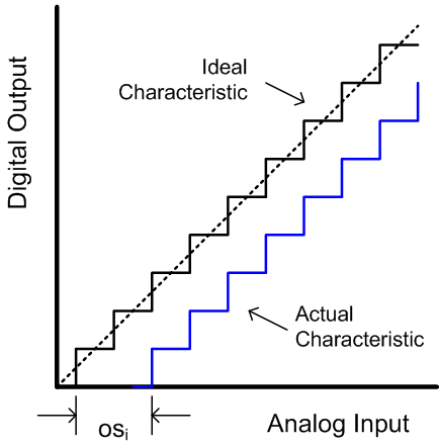
at a rate of 5 kHz. Tests show that these problems do not degrade the receiver SNR to an unacceptable level.

3.3 ADC Data Bus Timing Budget

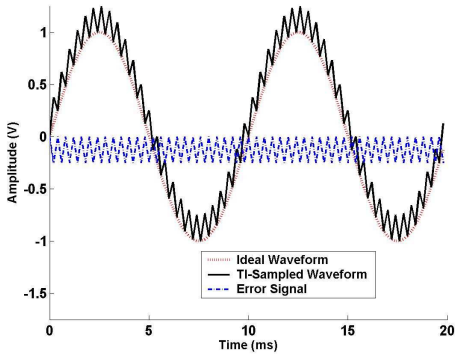
In addition to the timing requirements introduced by the use of time-interleaved sampling, the window of time in which the FPGA must clock in the the ADC data puts stringent limits on the skew and jitter associated with the components in the system. At a data rate of 500 MHz, the FPGA has 2 ns to clock in each sample. Simulations show an ADC data bus rise time of 300 ps. The FPGA register setup time for a Virtex-II Pro -7 speed grade is 840 ps. The FPGA register hold time is negative and does not affect the timing budget. The worst-case ADC-to-FPGA data bus skew is 160 ps. The DCM skew, DCM granularity, and DCM jitter add up to a total of 415 ps. All of these factors plus an additional 10% timing margin add up to a total of 1885 ps to clock in data, giving a margin of error of 115 ps.

3.4 Field-Programmable Gate Array

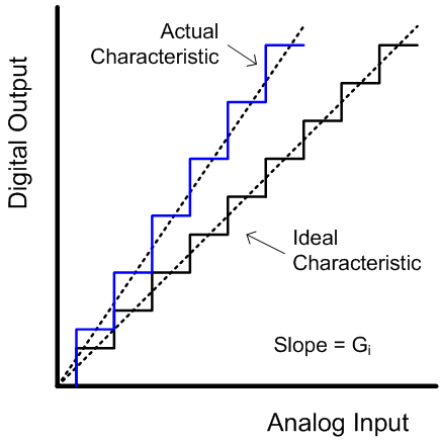
A Xilinx Virtex-II Pro FPGA provides the digital processing for the radio receiver. The Virtex-II Pro has programmable logic fabric, embedded PowerPC processors, fast hardware multipliers, and Block RAMs. This allows for both hardware and software processing of the ADC sample data. The Virtex-II Pro also has Digital Clock Managers (DCMs) that synchronize the FPGA clocks to the external ADC clocks and generate the system clocks needed by the hardware and PowerPCs.



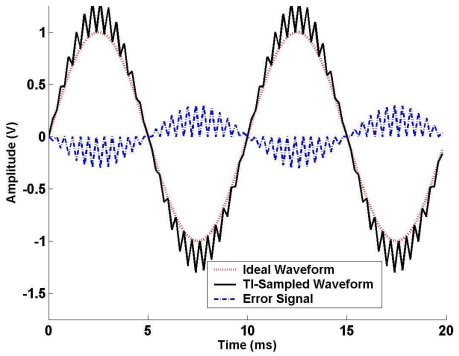
(a) Offset Mismatch Model



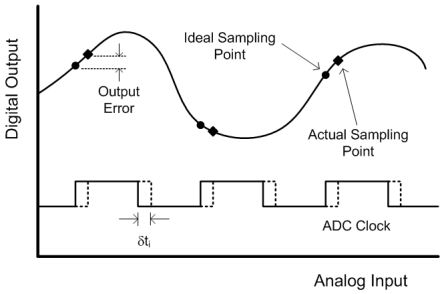
(b) Offset Mismatch Effects



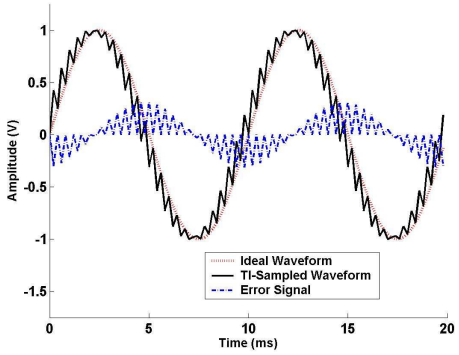
(c) Gain Mismatch Model



(d) Gain Mismatch Effects



(e) Timing Mismatch Model



(f) Timing Mismatch Effects

Figure 3.2: Effects of ADC Mismatches on a Two-ADC Time-Interleaved Sampling System [1]

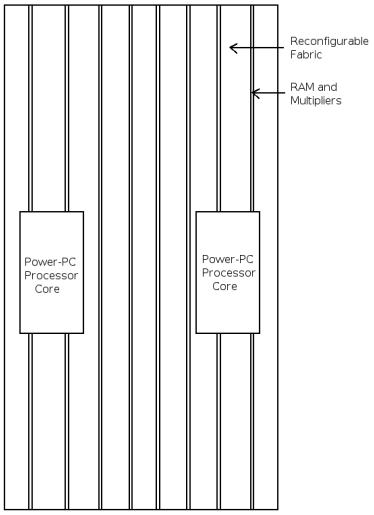


Figure 3.3: Xilinx Virtex-II Pro FPGA

3.5 RF Front End

The RF front end is designed with the objective of operating at a maximum range of 10 meters, while preserving the dynamic range of the system. The gain of the front end was chosen given the maximum tolerable input strength of the ADCs, the expected path loss at a range of 10 m, and the noise figure of the RF front end parts. A low pass filter prevents overdriving of the ADCs or amplifiers by out-of-band signals. A variable attenuator is used to fine tune the gain of the RF front end. The front end is implemented using MiniCircuits parts in order to save cost and development time.

3.6 Prototype Board

Before beginning the design of the eight-ADC radio receiver board, a two-ADC prototype board was created. This board was created to test the feasibility of an FPGA-based time-interleaved sampling radio receiver. The two-ADC prototype boards allowed for the testing of the TI sampling approach and served as a development platform for FPGA code. This

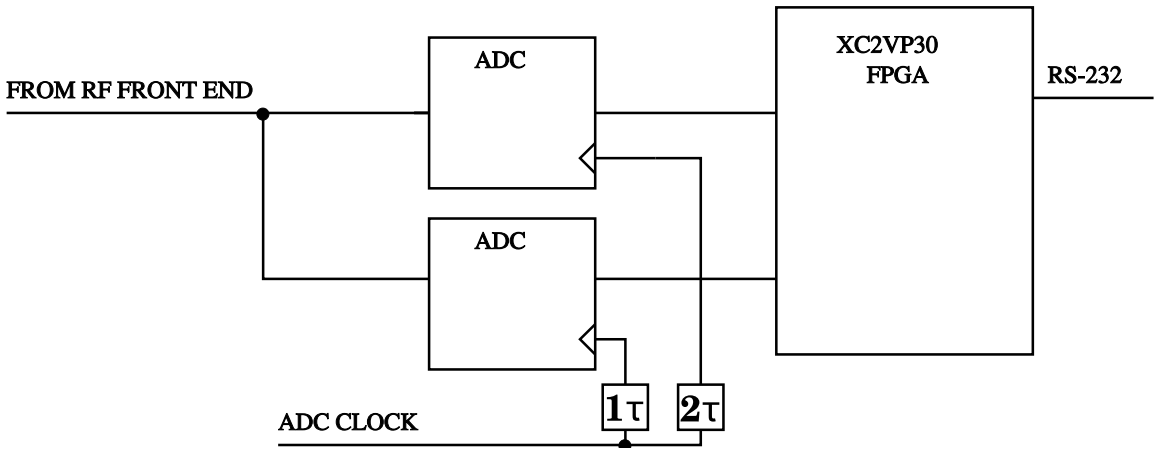


Figure 3.4: Test Board System Configuration

board used two 1 GHz Maxim MAX104CHC ADCs and a Xilinx Virtex-II Pro XC2VP30-6 FPGA. The programmable delay chips were controlled by DIP-switches.

3.7 Final Radio Receiver Board

The final radio receiver board is a scaled-up version of the test board. Instead of two ADCs, the radio receiver board uses eight Maxim MAX104CHC 1 GHz ADCs, for a maximum effective sampling rate of 8 G-samples/s. These ADCs are time-interleaved as they were on the test board. The programmable delay chips can be controlled by DIP switches or by the FPGA. A larger FPGA is used, the Virtex-II Pro XC2VP70-7. The XC2VP30 provides 13,696 logic slices, while the XC2VP70 provides 33,088 [31]. The XC2VP30 is insufficient for the eight-ADC system, since more logic cells are needed than the XC2VP30 provides. A basic design using eight ADCs that receives the ADC sample data and exports the data over RS-232 uses almost 12,000 slices. Whereas the XC2VP30 part would be almost full, the XC2VP70 part has many slices left unused that can be used to implement radio receiver functionality. The XC2VP70 also provides more Block RAMs, hardware multipliers, and I/O pins than the XC2VP30.

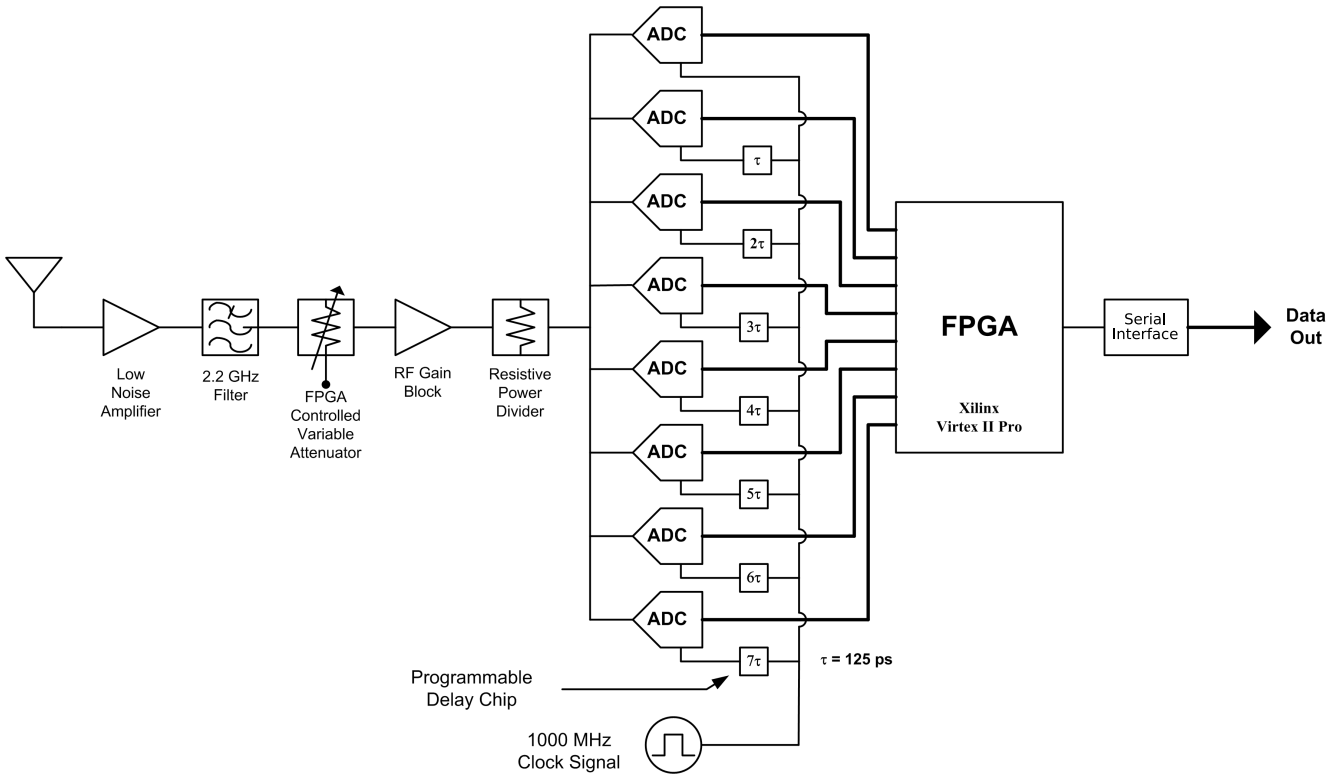


Figure 3.5: Final Board System Configuration [1]

3.8 Transmitter Board

The radio receiver board controls an impulse-UWB transmitter board. The transmitter produces a 2 ns positive or negative UWB pulse when given a trigger pulse. These pulses are generated by an analog circuit whose main component is a step recovery diode. An in-depth discussion of the transmitter electronics is beyond the scope of this work. For more information, refer to [1]. A header on the receiver board that is connected to FPGA I/O pins is also connected via a ribbon cable to the transmitter board trigger signal inputs. This gives the FPGA control of the transmitter board. A positive pulse is produced on the falling edge of the positive trigger signal and a negative pulse is produced on the rising edge of the negative trigger signal. Because the FPGA controls the transmitter board, changes to the modulation scheme can be made in software. The FPGA can choose whether to transmit a positive or negative pulse and can also choose when to transmit a pulse. This allows the FPGA to select modulation schemes such as On-Off Keying, Pulse Amplitude Modulation, and Pulse Position Modulation.

Chapter 4

FPGA Subsystem Design

The FPGA performs all of the digital processing in the radio receiver system. This chapter discusses the FPGA hardware and embedded PowerPC software for two systems: the pilot-based matched filter receiver and the vital signs monitoring application. Also discussed are issues involved in clock synchronization, which impact the time-interleaved sampling system.

4.1 ADC Data Capture

To process the ADC data samples, the FPGA must first move all of the ADC samples over into one master clock domain. In this system, the ADCs are each presented with a 1 GHz input clock. This input clock is used to clock in 1 G-samples/s of data; however, the ADCs do not present this data to the FPGA as a single 1 G-samples/s stream of data. Instead, each ADC presents the FPGA with two 500 M-sample/s data streams, the "`ABus`" and "`PBus`" data buses. The `ABus` and `PBus` data buses are in-phase with each other and the ADC presents a "`data ready`" signal to announce the presence of the next data samples on the buses. The FPGA uses the "`data ready`" signal from each ADC as a clock. This clock is further reduced to half of its frequency inside the FPGA and each 500 M-sample/s

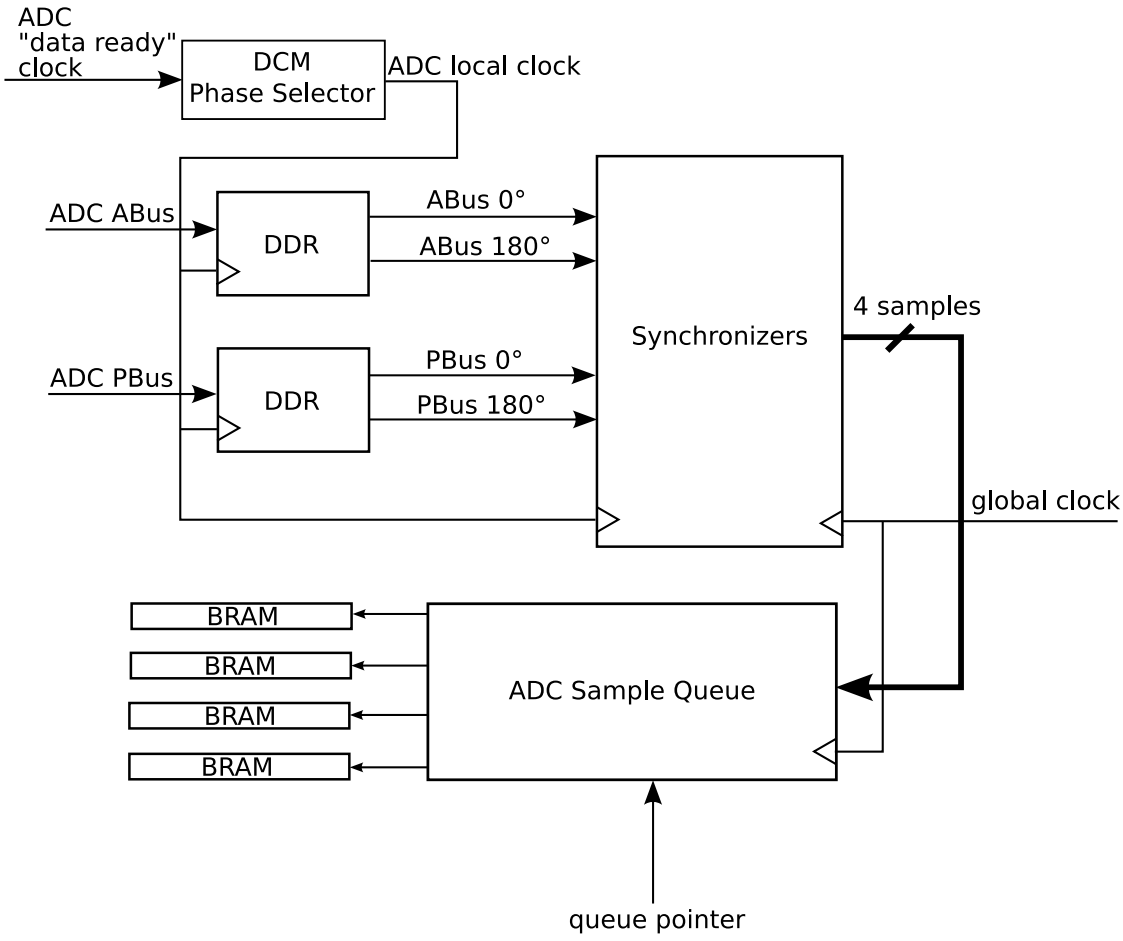


Figure 4.1: Data Capture Hardware Data Flow Diagram

bus is moved into two 250 M-sample/s buses, the "`0 degrees`" bus and "`180 degrees`" bus. A double-data rate (DDR) register inside an FPGA Input-Output Block (IOB) clocks data into the `ABus0` and `PBus0` buses on the rising edge of the 250 MHz clock generated by the FPGA and the `ABus180` and `PBus180` buses are clocked in on the falling edge of the same clock. These samples are then in eight different clock domains, one for each ADC. The samples from these clock domains are then moved to one master clock domain. The master clock domain is chosen to be the clock which is generated from the `data ready` signal from the earliest-sampling ADC. The samples are moved to the master clock domain by way of a synchronizer circuit, illustrated in Figure 4.2. This circuit breaks up the incoming 250 MHz data stream into two 125 MHz data streams, each clocked by the local clock. Each master clock cycle, the 125 MHz data stream which changed the earliest is clocked into the global clock domain. Moving the local data stream into two slower buses increases the amount of time from the change in the local bus to the global clock rising edge. This is illustrated in Figure 4.3.In this diagram, the top data stream is the local clock data stream. The middle two data streams are the two slower data streams created by the synchronizer circuit. The final data stream is the global clock domain data stream.

After all data streams have been moved into the global clock domain, a state machine moves the samples into Block RAMs (BRAMs) on the chip. The BRAMs are dual-ported memories that are connected to the hardware state machine on one port and a Processor Local Bus (PLB) on the other port so that the PowerPC can access the BRAMs. Section 4.2 describes this microprocessor system architecture. Figure 4.1 illustrates the flow of data from the ADCs to the BRAM, including synchronization steps discussed in the following sections.

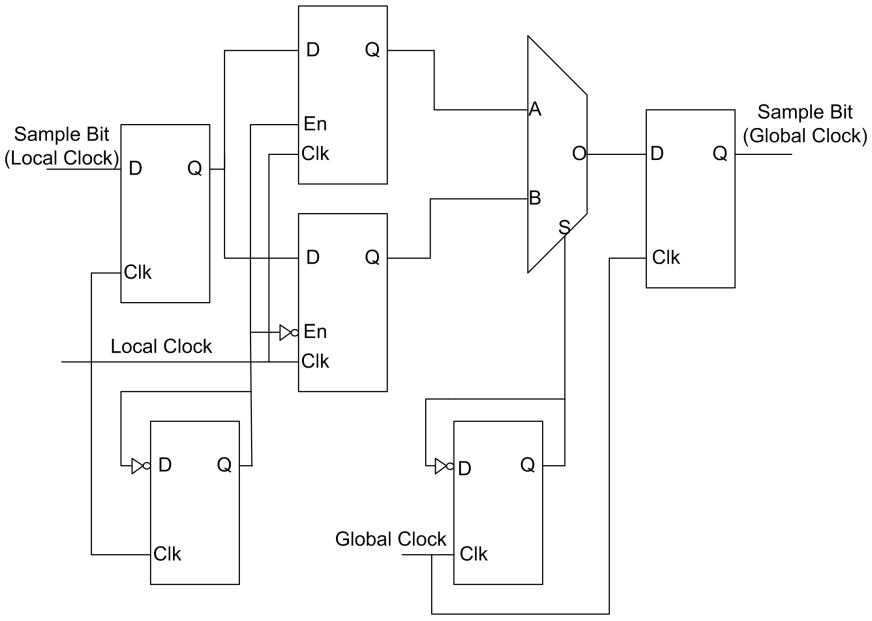


Figure 4.2: Data Synchronizer Circuit [2]

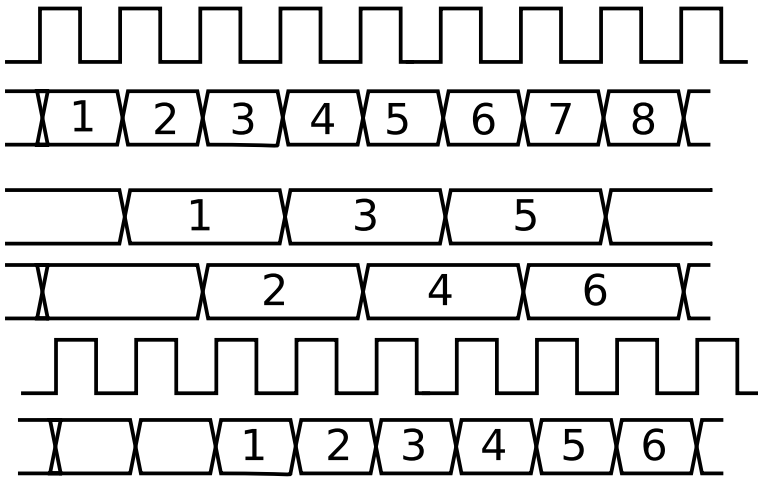


Figure 4.3: Data Synchronizer Timing Diagram

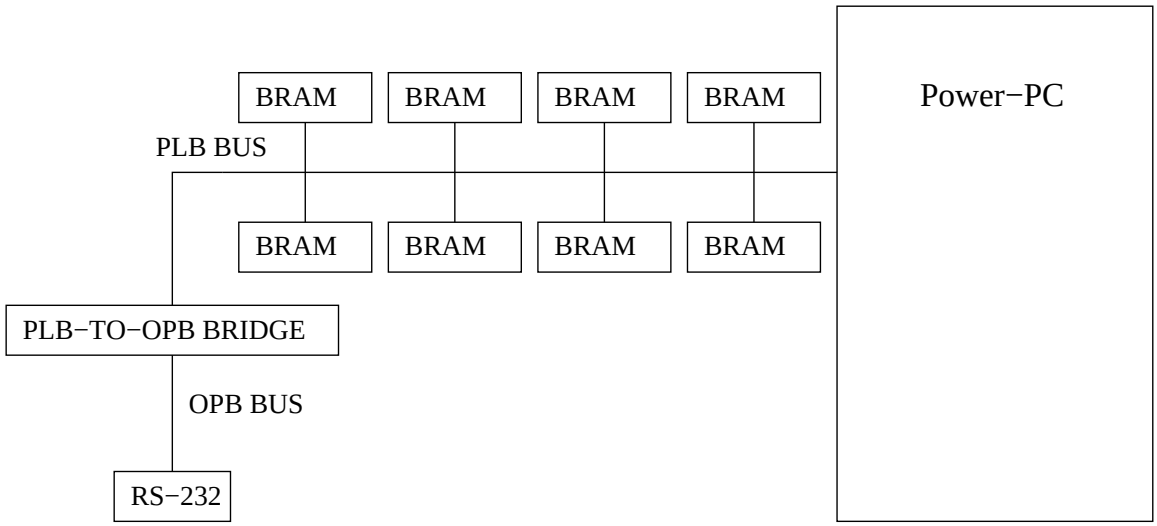


Figure 4.4: System Configuration for the Two-ADC Testboard

4.2 Processor System Architecture

The radio platform uses both configurable FPGA fabric and the internal PowerPC cores to process the ADC sample data. The internal dual-ported Block RAMs (BRAMs) are used to provide access to the sample data from the hardware to the processor. The PowerPC accesses the BRAMs and other peripherals through a Processor Local Bus (PLB). An RS-232 UART is accessed through a PLB-to-On-Chip Peripheral (OPB) bus bridge. The FPGA generates four data streams for each ADC and each data stream has its own Block RAM. Therefore, for the test board, which has two ADCs, eight BRAM controllers are needed. These eight BRAMs, along with the PLB-to-OPB bridge, are connected to the PLB bridge accessible by the PowerPC as illustrated in Figure 4.4.

This system architecture is not scalable to the eight-ADC radio board. For the eight-ADC system, 32 BRAM controllers are needed. The PLB bus core available for the Virtex-II Pro FPGA allows for a maximum of 16 slaves on a single PLB bus. A novel PLB-to-PLB bridge developed by Eric Lorden [32] allows the PowerPC to connect to the 32 BRAMs and the RS-232 UART. The PLB-to-PLB bridge is used to increase the maximum number of slaves

accessible by the PowerPC over the PLB bus. Three additional PLB buses are needed to connect all of the PLB slaves. Two PLB buses connect to 16 BRAM controllers each, and a third bus connects to the PLB-to-OPB bridge which connects to the RS-232 UART. Each of these PLB buses connects to the PowerPC PLB bus via a PLB-to-PLB bridge. The PLB-to-OPB bridge is placed on its own PLB bus rather than the PowerPC PLB bus. This is done because only PLB-to-PLB bridges may be connected to the PowerPC PLB bus if PLB-to-PLB bridges are to be used. Figure 4.5 illustrates this new system architecture.

4.3 Clock Uncertainty and Synchronization

A time-interleaved software-defined radio relies on the difference in phase between multiple clocks with the same frequency. When these clocks are reduced to half of their original frequency, uncertainty in the phase of the clocks is introduced. Steps must be taken to eliminate this uncertainty. There should be no uncertainty introduced into the system by the ADCs because the ADCs guarantee that at each "`data ready`" positive edge, the `ABus` is one sample older than the `PBus`. However, a problem with the ADC reset signals causes uncertainty to be introduced in the relative position of the ADC samples in their respective data streams. Individual ADCs reset at different clock cycles, causing the data streams for individual ADCs to be moved forward or backward in time. This problem is described in Section 4.3.2. A different problem dealing with clock uncertainty introduced by the FPGA is described in Section 4.3.1.

4.3.1 FPGA Clock Uncertainty

The generation of the 250 MHz clocks in the FPGA introduces uncertainty into the system. Figure 4.6 illustrates the two ways in which each 250 MHz clock can be generated by each 500 MHz ADC data ready signal. Each clock is generated by a different Digital Clock Manager (DCM) inside the FPGA. The DCM uses a "divide-by-2" attribute to automatically divide

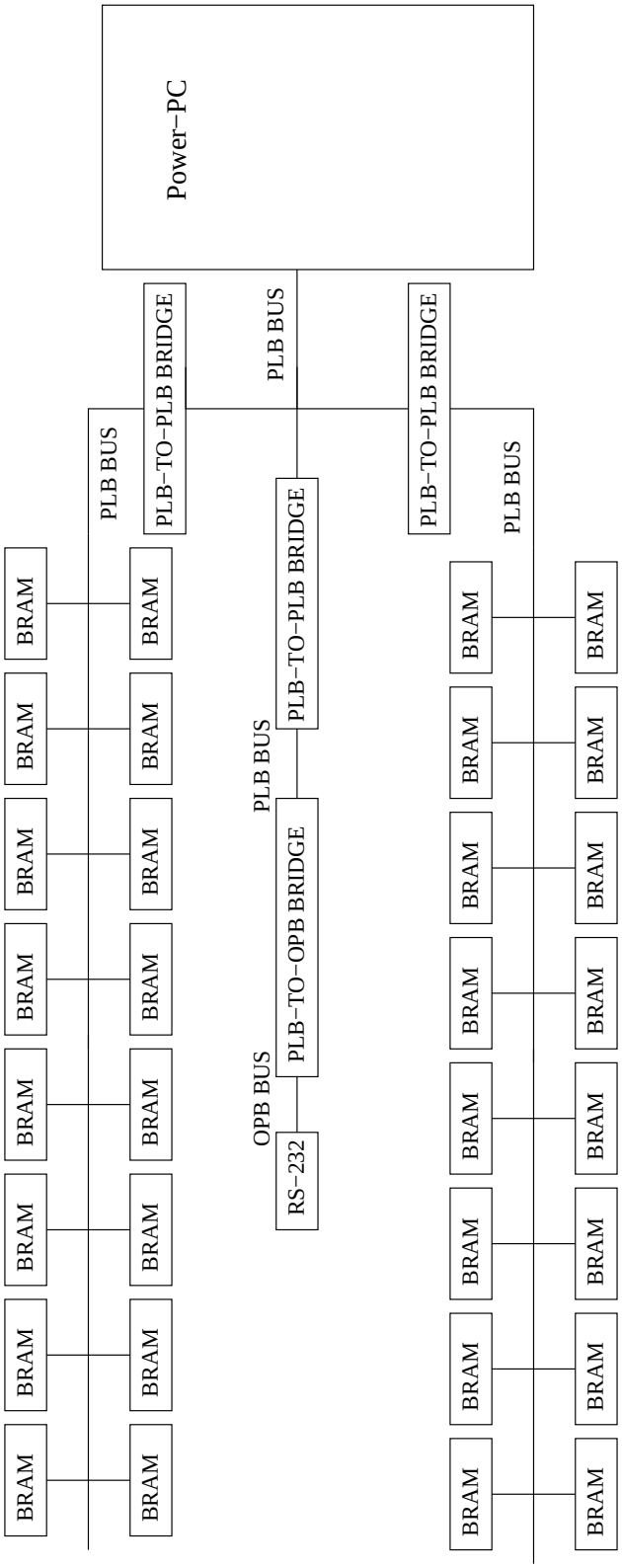
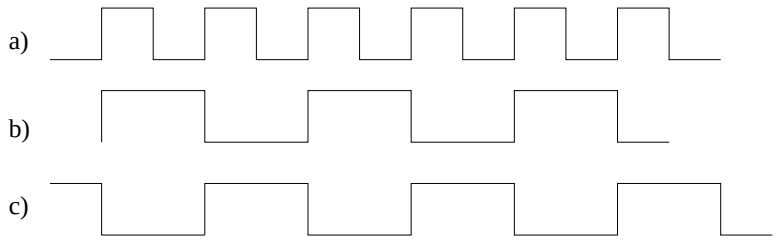


Figure 4.5: System Configuration for the Eight-ADC Radio Board



a) 500 MHz data ready output from ADC

b) 250 MHz clock generated by the FPGA, idealized with no skew

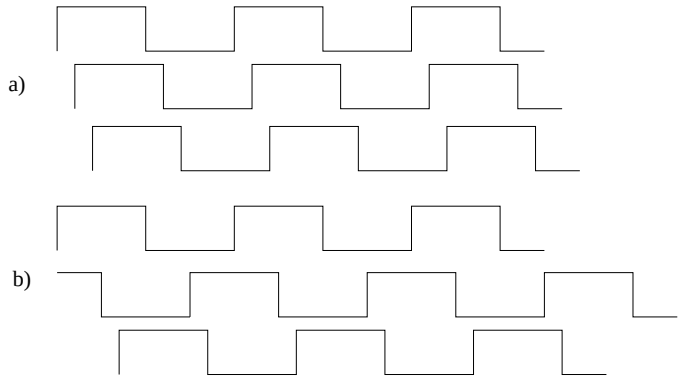
c) Alternate 250 MHz clock generated by the FPGA

Figure 4.6: The two ways in which the FPGA can generate the 250 MHz clock from the ADC `data ready` signal

the input clock to half of its frequency. A feedback path is used to lock the output clock to the phase of the input clock. The DCM "lock" output is asserted when the phase of the DCM output clock matches that of the input clock. Because the output clock has twice the period of the input clock, the DCM has two input clock rising edges to every one output clock rising edge with which to lock. Locking to one input clock rising edge as opposed to another gives a 180° phase difference on the output clock.

The FPGA generates eight clocks, one for each ADC. Because the system is using time-interleaved sampling, the ADC `data ready` clocks are out of phase with each other. Consequently, the FPGA-generated clocks are out of phase with each other. Ignoring skew, each ADC clock's rising edge occurs 125 ps later than previous clock's rising edge. If any of the FPGA-generated clocks are generated with the rising edge locked to the wrong edge of the input clock, then that clock's rising edge is 125 ps + 2 ns later than the previous clock's rising edge. This scenario is illustrated in Figure 4.7.

All eight FPGA-generated clocks must be relatively in phase, as in Figure 4.7-a. If they are not, then the out-of-phase clocks place the samples meant for the `ABus0` and `PBus0` buses into the `ABus180` and `PBus180` buses instead. This causes the `ABus` data and the `PBus` data



a) Three clocks relatively in phase

b) Two clocks relatively in phase, one clock 180° out of phase

Figure 4.7: Relative phases between different FPGA-generated clocks

to be one master clock cycle ahead relative to the other clocks. Figure 4.8 illustrates this problem.

Figures 4.8-a and 4.8-b show the ADC `data ready` clock signal and the `ABus` and `PBus` data buses. Figure 4.8-c shows the FPGA DCM-generated clock with the correct phase relative to the master FPGA, which is shown in Figure 4.8-k. The `ABus0`, `PBus0`, `ABus180`, and `PBus180` buses receive the data as shown in Figures 4.8-d and 4.8-e. These buses are clocked into the same local clock domain as in Figure 4.83-f. The same process occurs in Figures 4.8-h through 4.8-j, except the local clock is 180° out of phase from its correct phase. As seen in Figure 4.8-j, `ABus0` receives the next `ABus180` signal and `ABus180` receives the next `ABus0` signal. `PBus0` receives the next `PBus180` signal and `PBus180` receives the next `PBus0` signal. As shown in Figures 4.8-l and 4.8-m, when the samples are moved to the master clock domain, the correct FPGA clock causes `ABus` samples 1 and 2 and `PBus` samples 2 and 3 to be read. The incorrect FPGA clock causes `ABus` samples 2 and 3 and `PBus` samples 3 and 4 to be read.

The solution to this problem is to determine whether the phase of each clock is correct with respect to a master clock and to change the phase if it is incorrect. To determine if

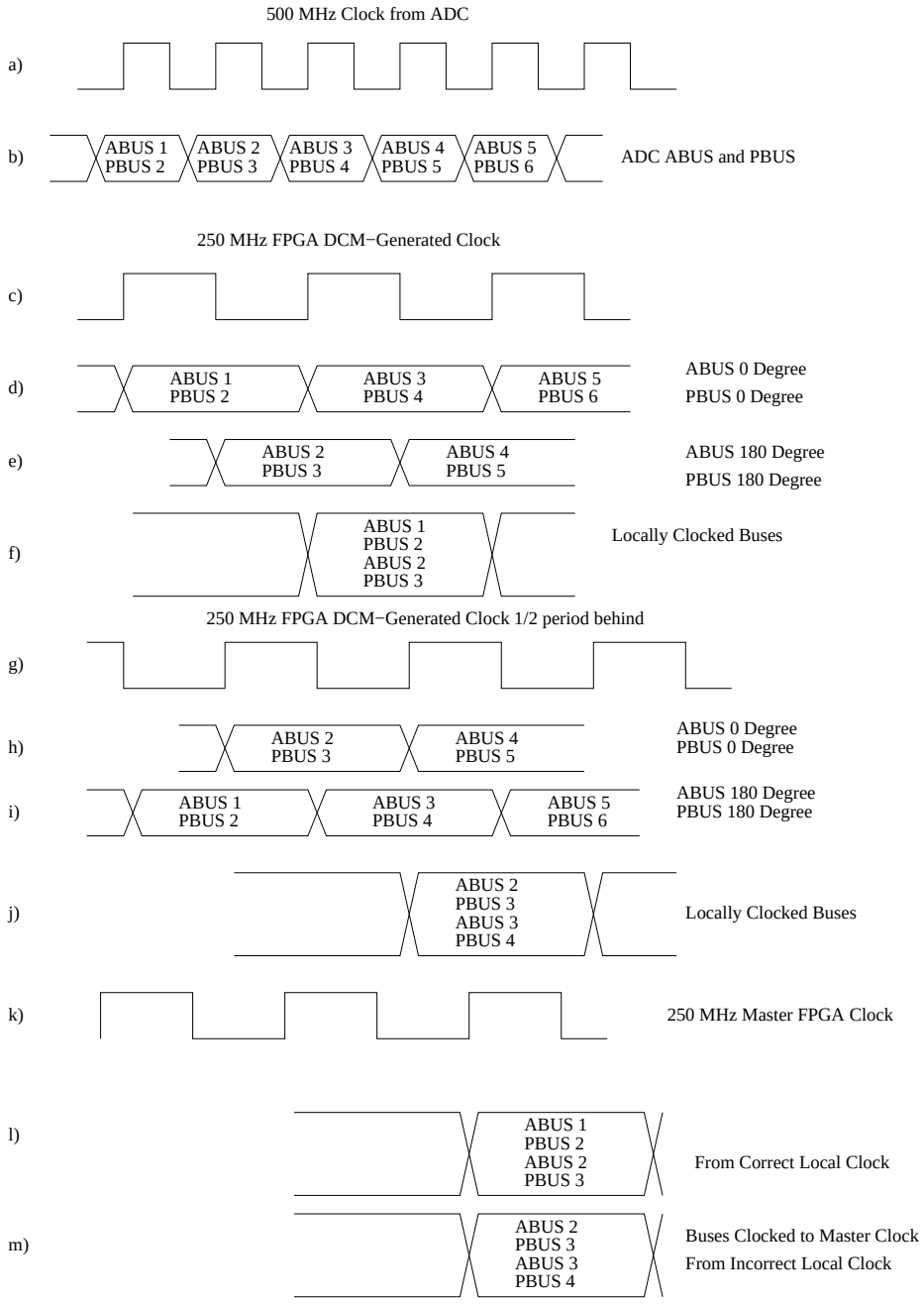


Figure 4.8: Input of Samples from Local Clock Domain with Correct and Incorrect Local Clocks

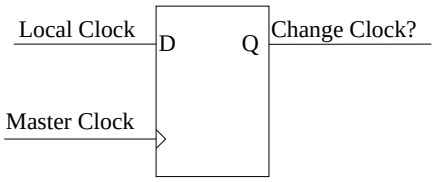
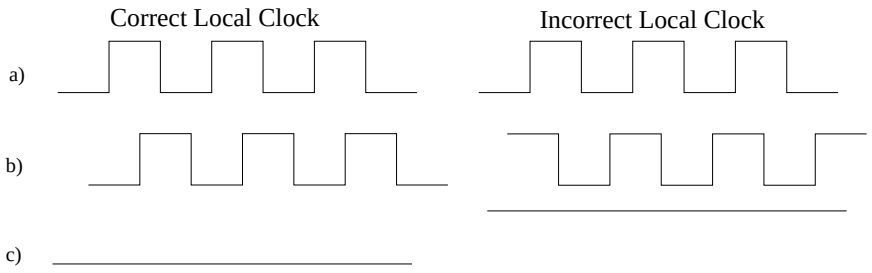


Figure 4.9: Method to Determine if Local Clock is Relatively In Phase with Master Clock



a) Master Clock

b) Local Clock

c) Change Phase Output

Figure 4.10: Waveforms from Phase Checking Circuit

a local clock is relatively in phase with the master clock, the local clock is clocked into a flip-flop which is clocked by the master clock. The master clock is the clock generated from the ADC1 `data ready` signal. This clock's rising edge will always occur before the other clocks' edges. If the local clock has the correct phase, then, when the master clock's rising edge occurs, the local clock is low. If the local clock has the incorrect phase, then, when the master clock's rising edge occurs, the local clock is high. These two scenarios are illustrated in Figure 4.10. On the left, the local clock has the correct phase, while on the right, the local clock has the incorrect phase.

The signal generated by this flip-flop can then be used to change the phase of the DCM. This can be accomplished in two different ways. The DCM `reset` signal can be used to change the phase of the DCM. When the DCM is reset, there is a 50% chance that the DCM will lock to the correct phase. If the DCM does not lock to the correct phase then the DCM is reset again. As shown in Figure 4.11, this can be accomplished by conditionally resetting the DCM if the DCM is locked and if the DCM's phase is incorrect. Correcting the DCM phase can also be accomplished by manually adjusting the phase. The DCM provides fine phase adjust signals which allow the user to adjust the phase by $1/256^{th}$ of the clock period at a time. When a clock is found to have the incorrect phase, a state machine controlling the DCM phase adjust signals can cause the DCM to adjust its phase 128 times to shift the phase by 180°. Either of these methods will solve the DCM phase uncertainty problem. The DCM reset solution is implemented in this system.

4.3.2 ADC Reset Uncertainty

As introduced earlier, the ADCs should not introduce any uncertainty into the system. Each ADC guarantees that its `ABus` contains the sample that is one sample earlier than its `PBus` during each `data ready` cycle. They are also reset using one master reset signal which is fed through the delay chips to the ADCs. This should ensure that each ADC comes out of reset at the correct time. However, there is an issue with the receiver board which causes

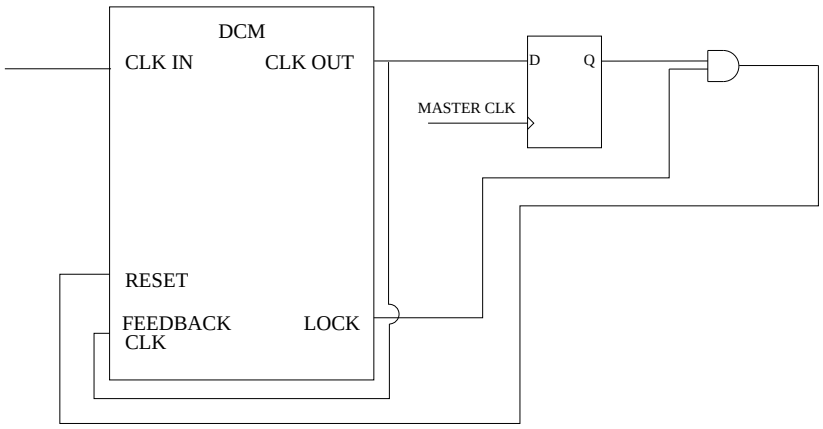


Figure 4.11: Method to correct DCM phase

the ADCs to reset at different ADC input clock cycles. Section 5.3 describes tests performed which show that this problem does not arise in the FPGA. If an ADC resets one clock cycle later than it should, then its `ABus` output contains the sample which should go in the current `PBus` output. The `PBus` contains the sample which should go in the next `ABus`. Testing has shown that there can be as much as a two-cycle difference between ADC sample streams. Because the relationship between the ADCs is different each time the ADCs are reset, a fix for this problem cannot be determined *a priori*. The solution chosen to counteract this problem is a queue in the FPGA for each ADC data stream. Each queue receives one ADC's `ABus0`, `ABus180`, `PBus0`, and `PBus180` signals every clock cycle. The queue stores three cycles' worth of samples and a pointer tells the queue what four signals to send out. The pointer is initialized to point to the middle cycle's samples, which allows the queue to move the ADC's data stream forward or backward up to four samples. Figure 4.12 shows an ADC sample queue pointing to the middle sample's `PBus0` sample, `PBus0 1`. Assuming all other ADC queues point to `ABus0 1` then this ADC's samples will be moved forward in time by one sample. The same can be done to move an ADC's sample backward in time by one sample by setting the pointer for that ADC to point to `PBus180 0`.

These queues each receive an initial pointer value when the FPGA is reset, but the pointer must be set correctly to counteract offsets in the ADC data streams created by incorrect

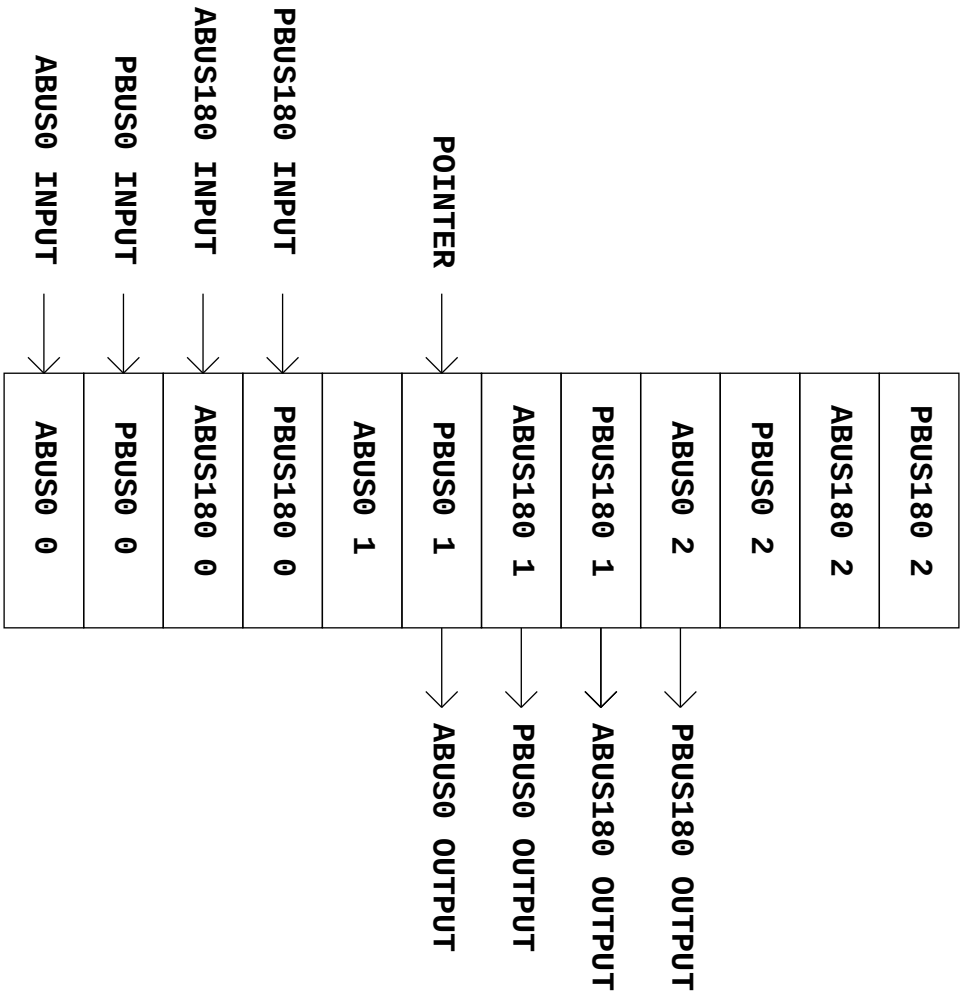


Figure 4.12: ADC Sample Queue for ADC Reset Correction

ADC resets. This is accomplished by giving the PowerPC inside the FPGA control over the queue pointers via the Device Control Register (DCR) bus. The DCR bus is a bus for which the PowerPC is a master and the FPGA hardware is a slave. Instructions exist which give the PowerPC software access to the DCR bus. The queue pointer values can be determined manually by an operator by analyzing the output of the data capture hardware given a known signal such as a sinusoid. The operator can then inform the PowerPC of the ADC queue pointer values via a serial port interface. It is also possible to create PowerPC software to automatically determine the ADC queue pointer values at startup given a known training sequence. Manual selection of the queue pointer values is implemented in this system.

After the DCM phases have been corrected, the samples are moved from the ADC ABuses and PBuses, through DDR registers, clock domain synchronizers, and the ADC sample queues, and into the BRAMs.

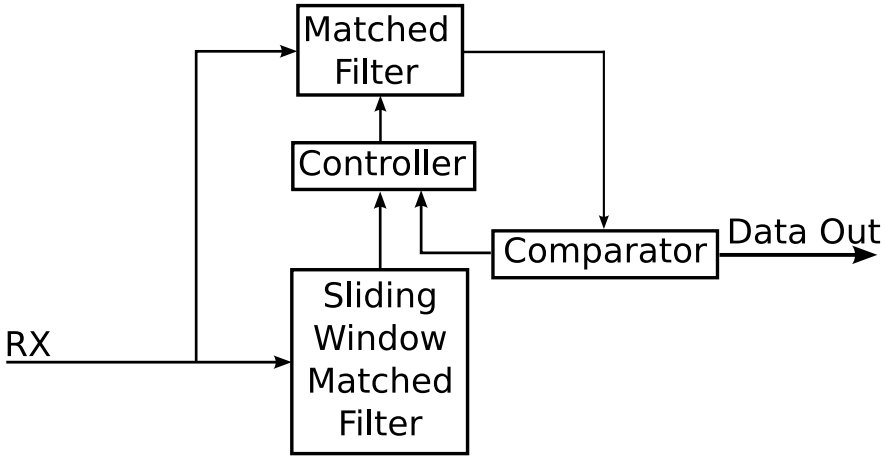
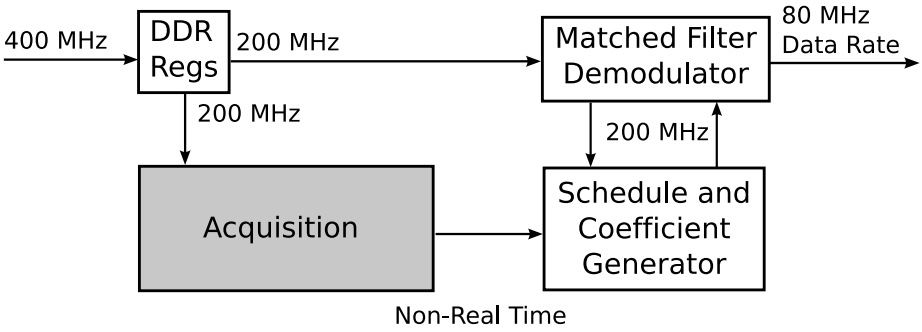


Figure 4.13: PBMF Receiver Block Diagram

4.4 Pilot-Based Matched Filtering Receiver

The receiver topology chosen for the UWB receiver is Pilot-Based Matched Filtering (PBMF). Pilot-Based matched filtering creates a stored copy of the incoming pulse to account for any



Shaded Area Performed in Software

Figure 4.14: PBMF Receiver Dataflow Diagram

multipath signals or distortions. This stored version of the pulse is used as the waveform against which the incoming samples are correlated using a matched filter operation. There are four steps involved in the PBMF receiver system: acquisition, pilot pulse reception, data demodulation, and real-time tracking. Figure 4.13 shows the communication block diagram for the system and Figure 4.14 shows the data flow diagram. These two figures are related to elements in the FPGA in Figure 4.15. An input data rate of 6.4 GHz is used for the PBMFR. This corresponds to an ADC input clock rate of 800 MHz. This data rate was chosen to make the creation of the FPGA bitstream more feasible in the allotted time. A decreased data rate equates to a decreased FPGA clock rate.

4.4.1 Acquisition

Before pulses can be demodulated correctly, the receiver must discover the arrival time of the incoming pulses. This is accomplished by finding the arrival time of a maximal-length sequence (m-sequence) sent by the transmitter. An m-sequence is used because m-sequences have attractive autocorrelation functions; the correlation between the m-sequence and itself is high whereas the correlation between the m-sequence and a shifted version of itself is very low [33]. This m-sequence is sent by the transmitter until the receiver acknowledges correct

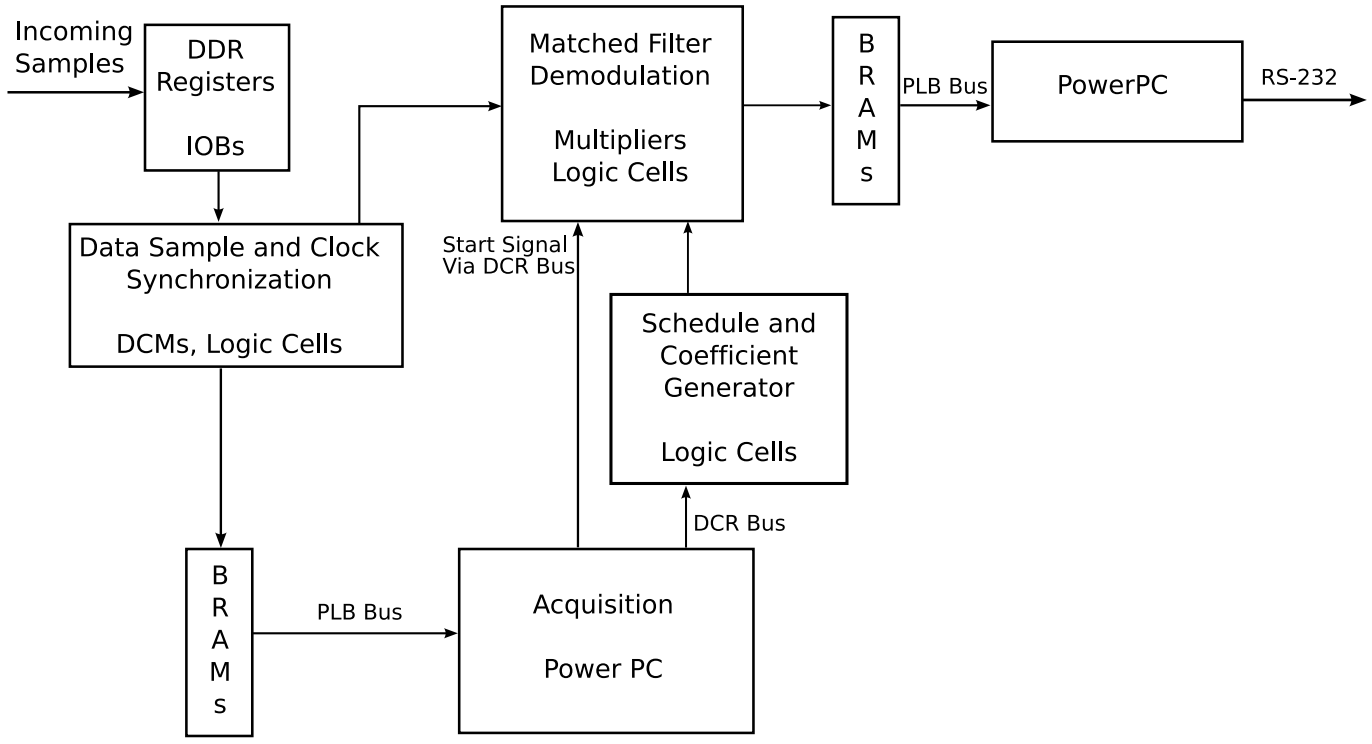


Figure 4.15: PBMF Receiver FPGA Diagram

reception of the sequence. To do so, the receiver captures a number of samples into the BRAMs so that the PowerPC can perform a sliding window matched filter operation on the samples. The receiver captures twice the number of samples per m-sequence so that any possible starting location can be tried against the matched filter.

The hardware and PowerPC software maintain synchronization with a hardware counter, `sampleCount` which wraps around at NP, where N is the number of pulses per m-sequence and P is the number of samples per pulse. For this particular implementation, a seven-pulse m-sequence is used, with 80 samples per pulse. Thirty-two samples are collected each master clock cycle, so the m-sequence repeats itself every 17.5 master clock cycles. To simplify the hardware, `sampleCount` is implemented as a modulo-35 counter in the master clock domain. This causes `sampleCount` to wrap around after two m-sequences have arrived. The PowerPC synchronizes with `sampleCount` using a DCR bus-controlled register, `triggerCount`. The PowerPC writes a value to `triggerCount` and, using another DCR bus register, signals the hardware to begin filling the BRAMs the next time `triggerCount` equals `sampleCount`.

Coarse Acquisition

The properties of the m-sequence were chosen given that the transmitter's oscillator and the receiver's oscillator are not perfectly synchronized. This imperfect synchronization causes them to drift apart during acquisition. Thus there is an upper bound on the time which acquisition can take. This uncertainty window is given by Equation 4.1 [2]:

$$U_{max} = \frac{2}{10^6} f_{osc} S_{osc} t_{elapsed} \tag{4.1}$$

where

U_{max} is the uncertainty window [Number of Samples],

f_{osc} is the frequency of the master oscillators [Hz],

S_{osc} is the stability of the master oscillators [Parts Per Million], and

$t_{elapsed}$ is the time required to complete acquisition [s].

The following is adapted from [2]. The maximum tolerable uncertainty window is found experimentally to be half the number of samples per pulse (40 samples for this system). The oscillators operate at 800 MHz with an estimated stability of 26.67 PPM stability. These parameters give an upper bound of 937.5 μs to the initial acquisition time. The PowerPC, operating at 300 MHz, is thus given 281,249 cycles to complete the initial acquisition. The sliding window correlation function is given by Equation 4.2 [2]:

$$C_K = \sum_{i=0}^{80N-1} x_{i+K} y_i \tag{4.2}$$

where

K is the starting index in the captured data,

C_K is the correlation magnitude,

x_i is the captured data at position i,

y_i is the template data at position i, and

N is the number of pulses in the acquisition m-sequence.

Because the UWB pulse is only 2ns long, which equates to 13 samples, the correlation only needs to be performed on 13 out of every 80 samples. For a N length m-sequence, the PowerPC performs $13N$ correlations. In [2] the authors estimate that each iteration requires 25 cycles. Because a starting location of several samples shifted from the correct starting location will still correlate strongly with the template waveform, the starting index can be incremented by more than one. The starting index K is chosen to be incremented by 8, giving an error of ± 4 samples from the correct starting sample in the m-sequence. This gives K_{MAX} possible starting indices [2]:

$$K_{MAX} = N(\frac{80}{8}) = 10N \tag{4.3}$$

This gives a total number of PowerPC cycles of [2]:

$$MaxCycles = (25)(13N)(10N) = 3250N^2 \tag{4.4}$$

The m-sequences can only have $2x-1$ pulses, where x is a positive integer. A seven-pulse m-sequence takes 159,250 PowerPC cycles to perform initial acquisition whereas a fifteen-pulse m-sequence requires 731,250 PowerPC cycles. Thus, a seven-pulse m-sequence is used, giving an uncertainty window of 22.65 samples.

To perform this "coarse acquisition" step, the PowerPC instructs the hardware to begin filling the BRAMs the next time `sampleCount` equals zero. The hardware signals the PowerPC via the DCR bus that at least $2NP$ samples have been put into the BRAMs. The PowerPC then uses the sliding window correlation function described in Equation 4.2 to find the approximate index of the start of the next m-sequence. This index is then used to perform the "fine acquisition" step.

Fine Acquisition

As described above, the coarse acquisition stage finds the start of the next m-sequence with an error of ± 11.32 samples. The fine acquisition stage reduces the error in the start index to ± 3 samples. In order to do so, the PowerPC performs another sliding window correlation. First, a new set of samples is acquired. Given the start index calculated by the coarse acquisition phase, the PowerPC sets `triggerCount` such that:

$$triggerCount = \frac{i_C - \frac{U}{2}}{32} \tag{4.5}$$

where

i_C is the start index computed by the coarse acquisition phase,

U is the uncertainty window,

N is the number of pulses per m-sequence, and

P is the number of samples per pulse.

This gives $triggerCount = \frac{i_C - 14}{32}$ for this system's parameters. After setting the `triggerCount` register and triggering the hardware to capture new data, the PowerPC performs the sliding

window correlation. However, instead of incrementing the start index by 8, the PowerPC increments the start index by 4 after each correlation. The PowerPC only searches the uncertainty window for the start of the m-sequence, so for an uncertainty window length of 22.64, the PowerPC will perform $\lceil \frac{28}{4} \rceil = 7$ correlations in the fine acquisition phase. These seven correlations will require

$$(13 * 6\ correlations/pulse * 7\ pulses * 25\ cycles/correlation) \tag{4.6}$$

cycles, approximately 13,600 cycles, to complete. After finding the new starting index for the next m-sequence, the PowerPC updates `triggerCount` so that the data demodulation hardware knows the starting location of the incoming pulses. The `triggerCount` register will only be updated if the fine acquisition phase finds that the actual start of the m-sequence is in a different cycle's set of 32 samples. The updated `triggerCount` is set such that:

$$triggerCount = (triggerCount + i_F/32) \bmod S_{MAX} \tag{4.7}$$

where

i_F is the start index computed by the fine acquisition and

S_{MAX} is the number before which `sampleCount` wraps around.

The next m-sequence will start when `sampleCount` equals `triggerCount`, at sample number i_F mod S_{MAX} in that set of 32 samples. The oscillator drift that occurs during the fine acquisition phase leaves an uncertainty window of two samples and the correlations only guarantee to find the true starting index within a ± 2 sample margin of error. These phase errors are corrected by a real-time tracking component which is described later.

4.4.2 Pilot Pulse Reception

After the receiver successfully finds the starting index of the next m-sequence, it signals the transmitter to begin sending test frames. Each frame consists of a known pilot pulse sequence

and data symbols. The pilot pulse sequence serves two purposes. The aforementioned real-time tracking component has a chance to correct any phase error. Also, the receiver is able to average several pilot pulses into a new template which accounts for any distortions and takes advantage of any multipath energy. The PowerPC is used to average the pilot pulses to create a template for the matched filter hardware. After processing the test frames, the receiver signals the transmitter to begin sending data frames which will be demodulated by the hardware.

4.4.3 Data Demodulation

The data demodulation hardware performs a matched filter operation on the incoming data samples. Because each incoming pulse is 80 samples wide and 32 samples are received each master clock cycle, specialized hardware is required to implement the multipliers and adder tree needed to perform the matched filter function. The matched filter function is [2]:

$$Z = \sum_{i=0}^{79} x_i y_i \tag{4.8}$$

Where

Z is the output of the matched filter function,

x_i is the i^{th} ADC sample value for the current pulse, and

y_i is the i^{th} template coefficient value for the current pulse.

Partial Correlation Unit

Each of the 32 data streams is given a dedicated multiply and accumulate module whose inputs are the current incoming sample, the current template sample associated with that data stream, and a schedule which controls the module's operation. The incoming sample and the template sample are both signed 8-bit values. This module, called the Partial Correlation Unit (PCU), is illustrated in Figure 4.16. When the schedule input to the PCU

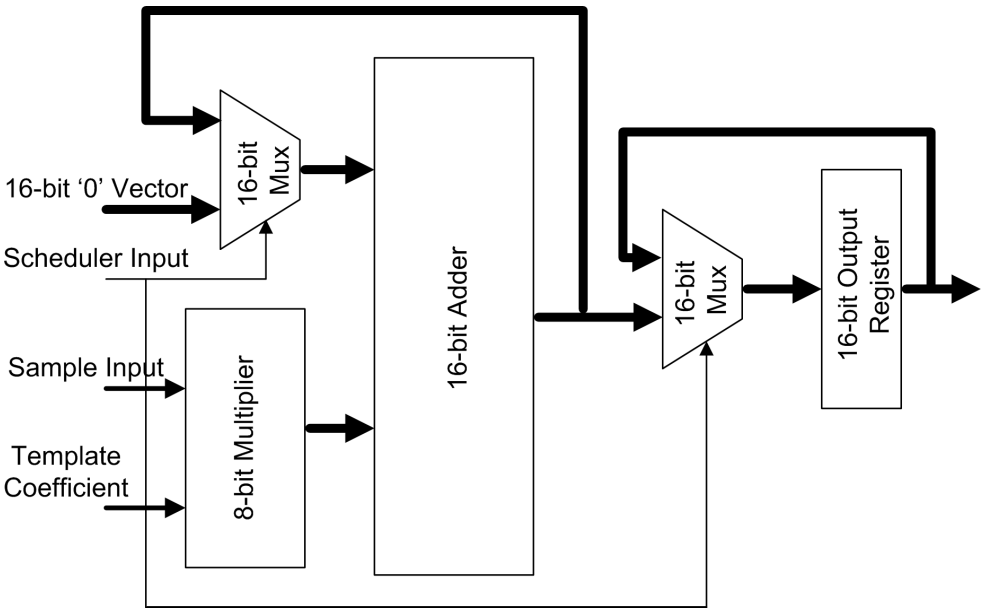


Figure 4.16: Partial Correlation Unit [2]

equals 1, the PCU adds the current multiplier output to the accumulator register. When the schedule input equals zero, the PCU's accumulator output contains the partial sum for that data stream's samples in the current pulse. Next cycle, the PCU will reset the accumulator register to the output of the multiplier. The partial sums generated by the 32 PCUs are fed to an adder tree which adds together the partial sums to generate a full sum for the current pulse.

Schedule and Coefficient Generation

The PCUs are each controlled by a schedule input. These schedule inputs are generated so that when a PCU encounters its first sample for a given pulse, its schedule input is 0, otherwise its schedule input is a 1. Because there are 80 samples per pulse and 32 samples per cycle, the 32-bit schedule register cycles between five different values. Figure 4.17 shows what the schedule register will be if even-numbered pulses begin at `ABus0` for ADC0, which is the zeroth data stream. Each set of 32 bits represents the schedule register value for a

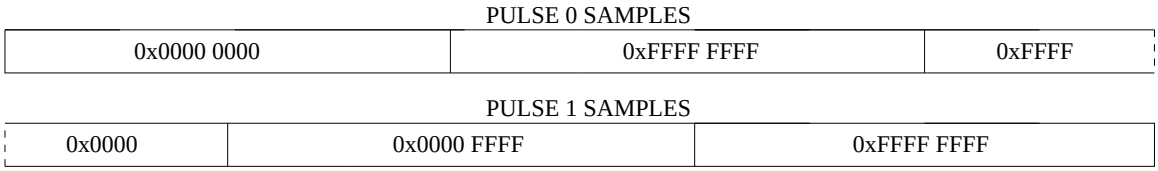


Figure 4.17: Schedule Register for Pulse Beginning at Sample 0

given cycle. If the even-numbered pulses begin n samples into the first set of 32 samples, then the schedule register will contain the same values, circular shifted to the right n times. The PowerPC determines the initial value of this schedule register during the acquisition phase. The schedule register will be shifted to the right $i_F \bmod 32$ times, where i_F is the start index computed by the fine acquisition.

The coefficients are generated in the same way. The PowerPC circular right shifts the standard pulse template by the same amount as the schedule register. The total schedule chain and coefficient chain are not stored as 160-sample chains. Only 80 samples are needed to store them; the last 80 schedule bits and coefficient values are the same as the first 80. After creating the schedule chain and the coefficient chain, the PowerPC sends them to the hardware one by one via the DCR bus. As the hardware receives the schedule and coefficient data it shifts them into the hardware `scheduleReg` and `coefficientReg`. These are the registers used to generate the schedules and coefficients for the 32 PCUs. As stated, these values cycle between five different states. Using the schedule register as an example, for PCU number i, the schedule input cycles between: $scheduleReg(i)$, $scheduleReg(i + 32)$, $scheduleReg((i+64) \bmod 80)$, $scheduleReg(i+16)$, and $scheduleReg((i+48) \bmod 80)$. The PowerPC sets `triggerCount` as shown in Equation 4.7 and triggers the schedule and coefficient generator to initialize its outputs when `triggerCount` equals `sampleCount`. This synchronizes the PowerPC to the schedule and coefficient generator. Using these values, the PCUs demodulate the incoming data samples.

Adder Tree and Comparator

The partial sums generated by the PCUs are added together into the full sum for the current pulse using a multi-stage adder tree. This full sum, a 16-bit signed value, is fed to a comparator which decides the value of the current bit given the full sum value. The threshold against which the full sum is compared is controllable by the PowerPC via the DCR bus. Given a full sum of Z, for binary pulse-amplitude modulation (2-PAM), $Z > 0$ indicates that a 1 was transmitted, while $Z < 0$ indicates that a 0 was transmitted. For On-Off Keying (OOK), $Z >$ a set threshold value indicates a 1 was transmitted while $Z <$ the threshold indicates a 0 was transmitted.

4.4.4 Real-Time Tracking

The receiver and transmitter oscillators will continuously drift apart due to imperfect synchronization. The real-time tracking module helps the receiver and transmitter maintain synchronization after the acquisition phase. To do so, the data demodulator provides three outputs instead of one: the on-time full sum, the full sum for the samples correlated to a template one sample late, and the full sum for the samples correlated to a template one sample early. The two additional coefficient and schedule registers are provided by the schedule and coefficient generator. The comparator module that determines the output bits from the on-time full sum also determines if the early template sum correlates better with the incoming samples. If so, the schedule and coefficient generator is signaled to shift its outputs so that the on-time template is set to the former early template. The same occurs if the late template is found to correlate better than the on-time template.

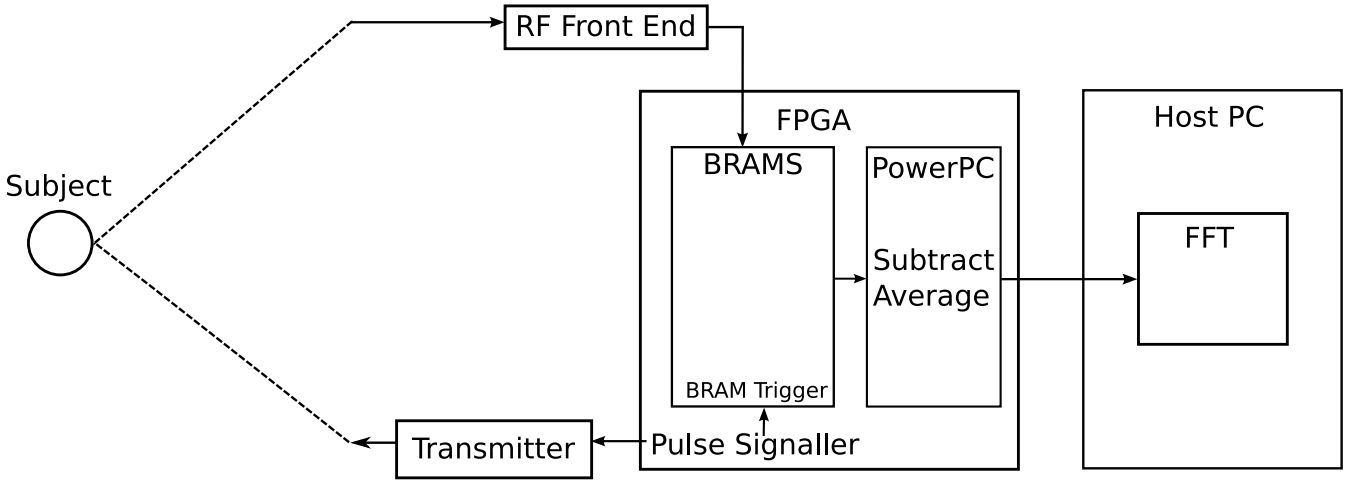


Figure 4.18: Vital Signs System Overview

4.5 Vital Signs Monitor

This system is also fit for use in problem domains other than communications, such as biomedical analysis. In [34], it is shown that Impulse-UWB can be used to remotely detect changes in a person's breathing rate or heart rate. A displacement of the chest causes a change in the multipath of an Impulse-UWB signal. Multiple pulses are sent and received and the average of all the pulses is subtracted out from each pulse to remove the static information. For a Fast Fourier Transform taken along the time where the multipath changes, the frequency that contains the peak of the FFT output is equal to the breathing rate of the subject. To perform this function, the data capture receiver hardware is used in conjunction with the transmitter board. FPGA hardware triggers the transmitter board to send pulses at a rate of 8 MHz. The pulse generator hardware also signals the BRAMs to begin capturing samples each time a pulse is generated. These pulses bounce off of the chest of the subject and the initial pulse and multipath signals are received by the FPGA and placed in the BRAMs. After multiple pulses have been acquired, the PowerPC averages the pulses together and subtracts the average from the captured data. The result is sent to a host PC connected via serial port. The host PC performs the FFT operation to find the breathing rate.

Chapter 5

Results

This chapter reports the results gained from the UWB SDR system. The eight-ADC system was fabricated and tests were constructed to demonstrate correct operation of the board and the FPGA. Time-interleaved data capture using eight ADCs with an effective sampling frequency of 6.4 G-samples/s was performed. The vital signs monitoring system was implemented and tested and the pilot-based matched filter receiver was partially implemented.

5.1 Initial System Testing

To test the functionality of the eight-ADC system after its fabrication, several basic FPGA bitstreams were created. First, an empty bitstream was created in Xilinx FPGA Editor to ensure that the FPGA could be programmed correctly. Another set of bitstreams tested the clocking resources on the board. One bitstream used several counters, each one clocked by a different ADC `data ready` clock. These counters were connected to the LEDs on the receiver board to demonstrate operation of the ADC clocks. A similar bitstream also connected a counter to the LEDs on the receiver board. However, this counter was clocked

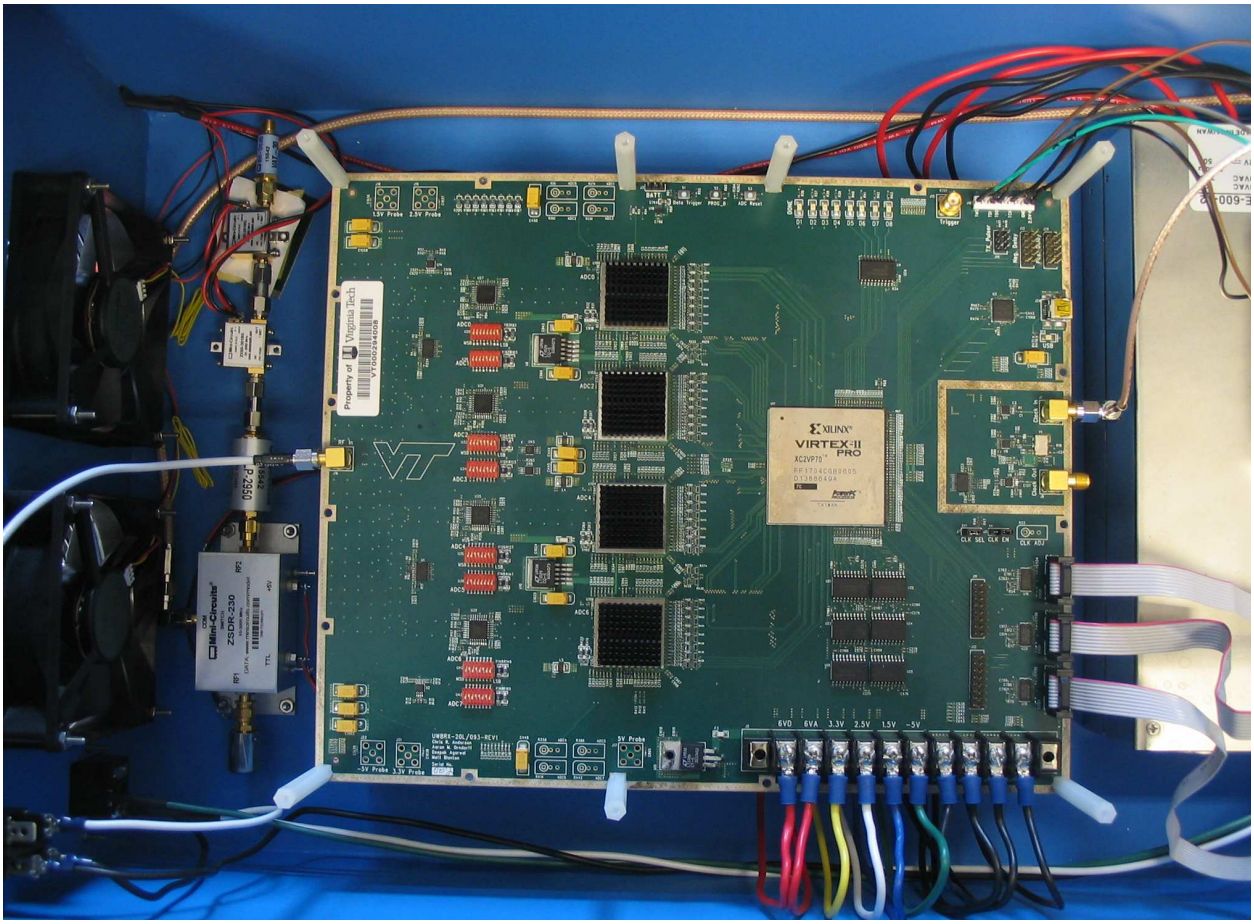


Figure 5.1: Radio Receiver Board

by the on-board 100 MHz oscillator to ensure that this oscillator was operational. Because of a shortage of DCMs on the FPGA, this oscillator was not used to generate the PowerPC clock as was intended, although the oscillator could be used for other systems which require a 100 MHz clock. Overall, these tests showed that the FPGA had at least rudimentary functionality.

5.2 Data Capture

The data capture hardware was built and tested for an ADC input clock rate of 800 MHz resulting in an effective sampling frequency of 6.4 G-samples/s. The hardware was built for 800 MHz, rather than 1 GHz to save development time, meaning that the data capture hardware inside the FPGA ran at 200 MHz instead of 250 MHz. Increasing the maximum frequency to 250 MHz would require much more processing time to create a valid bitstream. Many more runs of the Xilinx tool place-and-route would be needed to find a solution whose placement and routing allowed for a maximum frequency of 250 MHz. The data capture hardware required 35% of slices in the Xilinx XC2VP70 FPGA.

The dynamic range of the receiver system is illustrated in Figures 5.2 and 5.3. A 393 MHz sine wave was captured with a dynamic range of 37 dB. A 793 MHz sine wave was captured with a dynamic range of 28 dB. The theoretical maximum dynamic range, or signal-to-noise ratio, is:

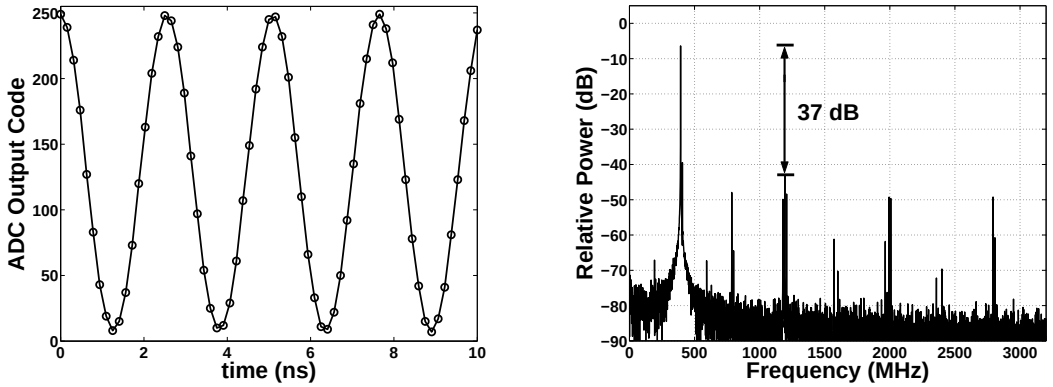
$$SNR = (B * 6.02) + 1.76 \tag{5.1}$$

where

SNR is the signal to noise ratio,

B is the number of bits of precision for the ADC.

This gives a maximum SNR of 49.92 dB. The reduction in SNR from the ideal best case is partially caused by ADC timing, gain, and offset mismatches introduced by the TI sampling



(a) Time Domain Output

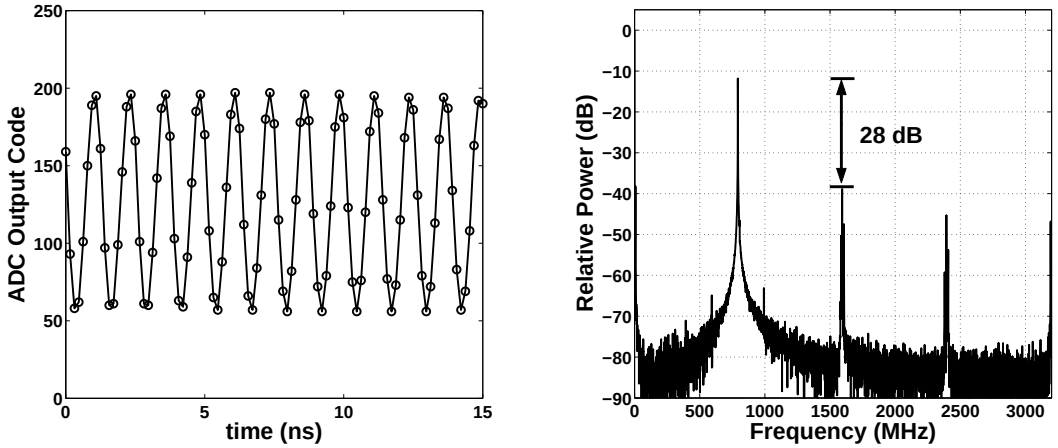
(b) Frequency Domain Output

Figure 5.2: 393 MHz Sine Wave Input Captured With 6.4 GHz Sampling Rate [1]

system [1, 35, 36, 37, 38].

An input data rate of 6.4 GHz was sufficient to accurately sample and reconstruct a UWB pulse. Figure 5.4 shows the difference in a UWB pulse captured by the 6.4 GHz input bandwidth data capture and the sample pulse captured by an oscilloscope. Figure 5.5 shows a positive UWB pulse with several multipath signals and Figure 5.6 shows a negative UWB pulse with several multipath signals.

Functioning data capture hardware running at 6.4 G-samples/s is an important achievement. All problems dealing with sample synchronization and clock and reset uncertainty have been addressed. This allows the FPGA to offer a simple-to-use hardware or software interface to the incoming ADC data samples. A simple method of communication with a host PC is also available. A researcher wishing to test a new application need only develop his hardware to accommodate the influx of 32 data samples per clock cycle.



(a) Time Domain Output (b) Frequency Domain Output

Figure 5.3: 793 MHz Sine Wave Input Captured With 6.4 GHz Sampling Rate [1]

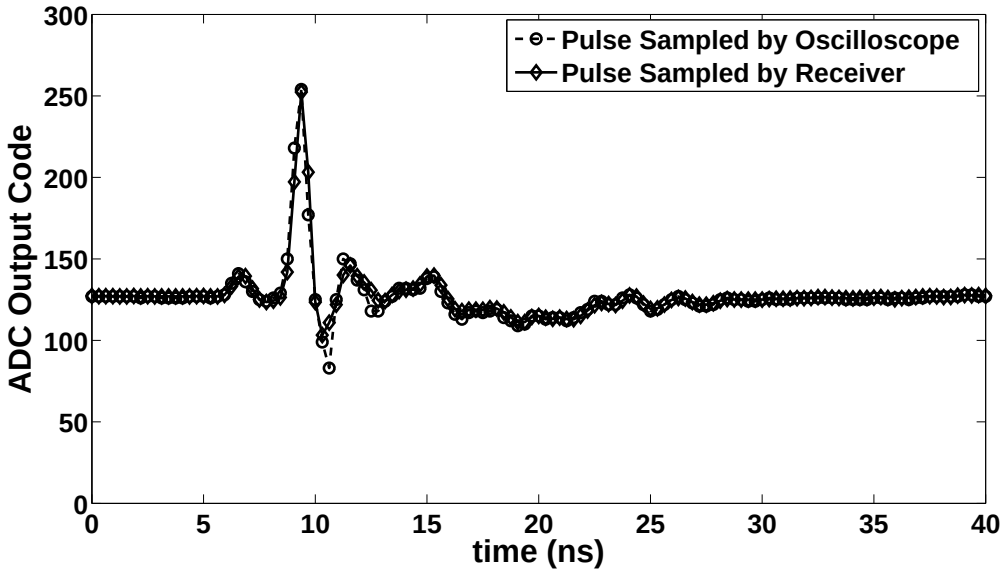


Figure 5.4: Comparison of UWB Pulse Captured by 6.4 GHz Data Capture and an Oscilloscope [1]

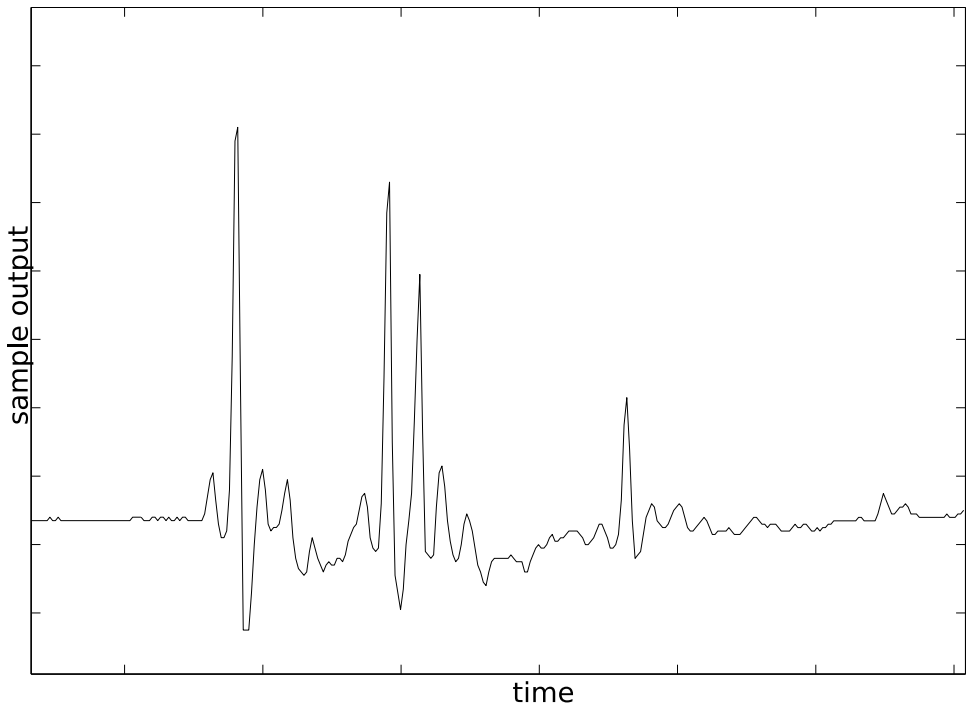


Figure 5.5: UWB Positive Pulse Including Multipath Captured with 6.4 G-samples/s Data Rate

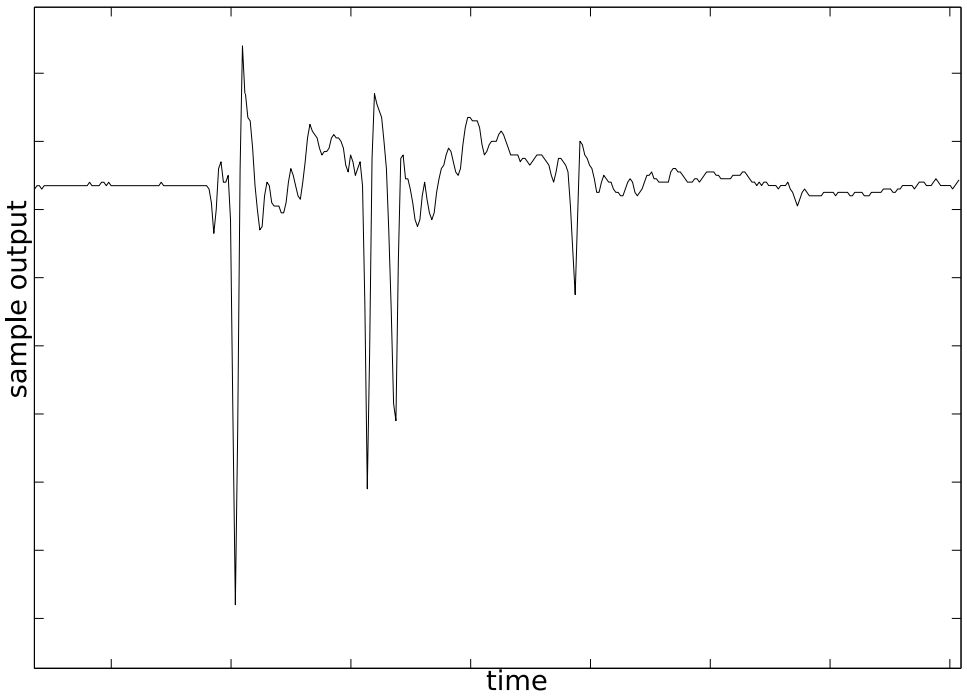


Figure 5.6: UWB Negative Pulse Including Multipath Captured with 6.4 G-samples/s Data Rate

5.3 ADC Reset Testing

Additional tests were required to confirm that the ADC reset problem described in Section 4.3.2 was in fact a problem with the board and not an FPGA problem. A test was created which brought the `ABus` and `PBus` signals from the ADCs into the FPGA and sent them to a Chipscope Logic Analyzer core. The Chipscope core allows a host PC to view the values of signals on the FPGA cycle by cycle. Using Chipscope, it was shown that the phenomenon seen in the ADC data was caused by something external to the FPGA. The FPGA design did not multiplex the `ABus` and `PBus` signals at all. For some ADCs, each `ABus` sample contained the data which should have been in the current `PBus` sample. The `PBus` contained the data which should have been in the next `ABus` sample. This caused that ADC's samples to be one cycle early. The opposite was also seen; each `ABus` sample contained the data which should have been in the last `PBus` sample. The `PBus` contained the data which should have been in the current `ABus`, causing that ADC's samples to be one cycle late. Figure 5.7 shows an example of this problem. A single sinusoidal signal was used as the input for this test. The ADC input clocks were set at 400 MHz, for an effective sampling frequency of 3.2 GHz. In this figure, the samples in ADC5 and ADC6 are one cycle early as compared to those in ADC1, ADC2, ADC3, and ADC4. The samples in ADC7 and ADC8 are one cycle late as compared to those in ADC1, ADC2, ADC3, and ADC4. For example, each cycle `ABus5` (ADC5's `ABus` stream) contains the data which should be in `PBus5`. The data which should be in `ABus5` is in last cycle's `PBus5` sample. The same holds for ADC6. ADC7 and ADC8 have the opposite problem. `ABus7` contains the data which should be in last cycle's `PBus7` sample. The data which should be in `ABus7` is in `PBus7`'s current sample. This problem is plotted in Figure 5.8. As discussed in Section 4.3.2, the solution to this problem was individual ADC FIFOs with adjustable pointers. The method used to choose the FIFO pointer values was manual selection using a sinusoidal reference signal.

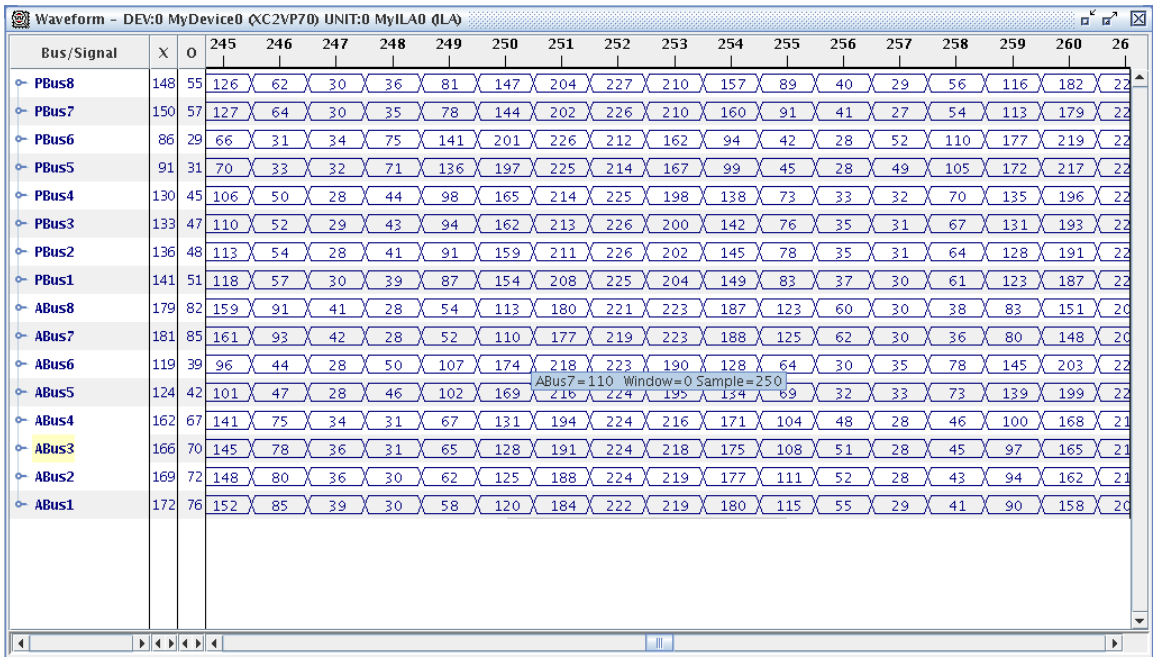


Figure 5.7: Chipscope Output Showing ADC Sample Misalignment

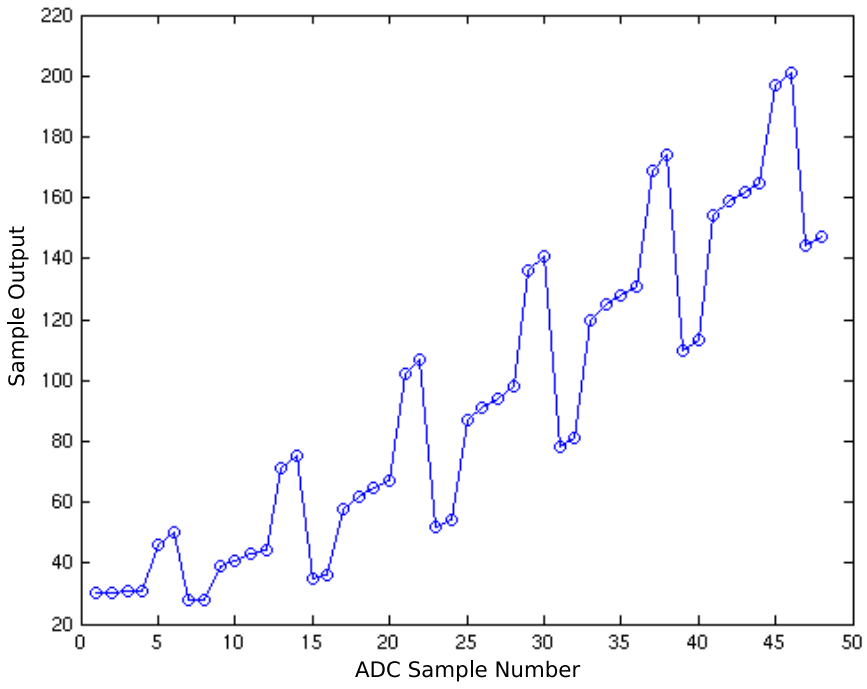
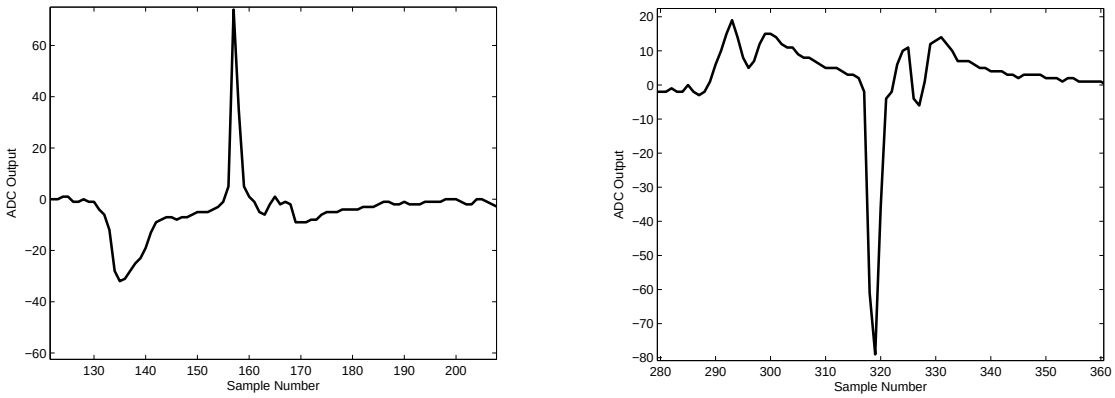


Figure 5.8: Graph of Chipscope Output Showing ADC Sample Misalignment



(a) Positive UWB Pulse

(b) Negative UWB Pulse

Figure 5.9: Transmitter Board UWB Pulse Output

5.4 Transmitter Board

The transmitter board was able to create a 2 ns positive and a 2 ns negative pulse. However, the transmitter does not create an ideal UWB pulse; both the positive and negative pulses are distorted as shown in Figure 5.9. The receiver board was able to control the pulse transmission delay in order to equalize the delay in sending a positive or negative pulse.

5.5 Pilot-Based Matched Filter Receiver

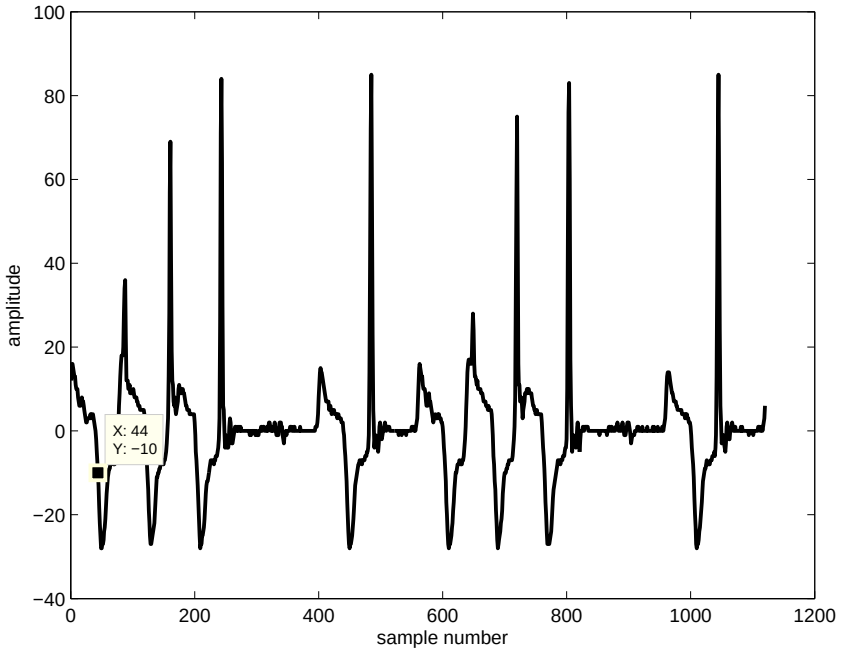
The full pilot-based matched filter receiver system is not yet operational. Signal integrity problems with the transmitter created difficulties in designing the receiver. Binary-PAM could not be used because when both positive and negative pulses were used, false triggers caused pulses to be sent erroneously. When OOK was used, a significant error signal was generated one symbol before a "1" was sent. Additionally, a large negative region was generated before each positive pulse, as illustrated in Figure 5.9(a). Some positive pulses were not generated correctly and had very small magnitude that was very close to the

magnitude of the error signal. This created difficulties in distinguishing a positive pulse from an error signal.

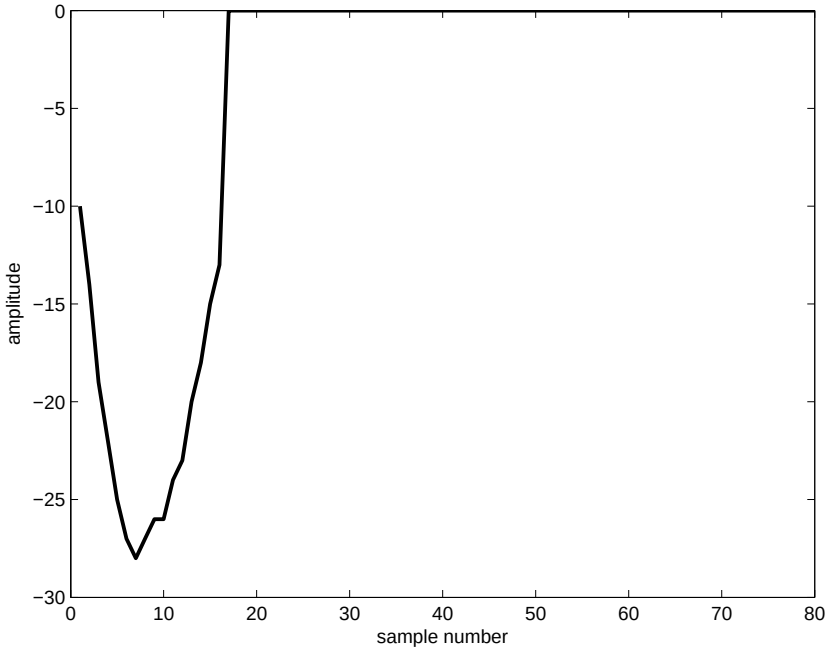
The current hardware available also added difficulties to the transceiver system. Another enclosure was needed for proper airflow before the ADC clocks on the transmitter system could be used. The on-board 1 GHz oscillator was the ADC clock, so use of this oscillator caused the ADCs to operate and generate heat. The enclosure was needed to assure the ADCs remained at an appropriate temperature during operation. The lack of the additional enclosure disallowed the use of the 1 GHz oscillator. The on-board 100 MHz oscillator could not be used because its stability of 100 PPM was too poor for the current design of the transceiver system. In order to perform testing on the transceiver system, a clock generated by the receiver was sent to the transmitter system. This clock was used to generate all of the clocks needed by the transmitter. This configuration introduced large amounts of jitter because the DCM-generated receiver clock was used as the input to a DCM in the transmitter system. This jitter reduced the ability of the receiver to perform acquisition and to correctly demodulate the signal.

OOK was used to test the receiver. However, instead of using the positive pulse as the signal to be demodulated, the negative region before each positive pulse was used. This region was generated before each true positive pulse but not before each error pulse, which allowed for the distinction between "0" and "1" symbols.

The acquisition phase of the receiver system worked correctly. The negative region of the pulse waveform was used as the signal. The receiver was consistently able to find the start of the negative region at the beginning of the m-sequence. Figure 5.10 demonstrates the operation of the acquisition software. Figure 5.10(a) shows the location that the acquisition software chose as the start of the m-sequence, and Figure 5.10(b) shows the template waveform used in the sliding window correlation. The PowerPC was able to generate the coefficients and schedule correctly and pass them to the receiver hardware via the DCR bus. The receiver was able to create a template for the hardware based on the average of incom-



(a) Start of m-sequence as Found by Acquisition Phase



(b) Signal Template Used During Acquisition

Figure 5.10: Results of Acquisition Phase

ing pilot pulses. The matched filter hardware requires additional debugging before it will be operational.

5.6 Vital Signs Monitor

The vital signs monitoring application functioned as anticipated. To test the system, a metal plate positioned several feet away from the transmit and receive antennae was used as a substitute for a person's chest. The vital signs monitoring FPGA hardware was set to capture 160 pulses at a rate of 8 Hz during the time in which the metal plate was moving. These 160 pulses were averaged in the PowerPC and the average pulse was subtracted from each pulse to remove the stationary signals. This resulting dynamic data was sent to a host PC. The pulse data was analyzed to find the point at which the dynamic data changed from pulse to pulse. One sample each was taken from the dynamic data from each pulse at this point. These samples were passed through an FFT to find the frequency of the oscillation of the metal plate. For the test whose output is shown in Figures 5.11 and 5.12, the metal plate was moved towards and away from the antennae at a rate of approximately 0.62 Hz. Figure 5.11 shows the time domain response of the pulse multipath to the oscillation of the metal plate. As expected, the output oscillates at approximately the rate of oscillation of the metal plate. The frequency response, displayed in Figure 5.12, shows graphically that the pulse multipath was able to capture the rate of oscillation of the metal plate, with the maximum power occurring at 0.6 Hz. Table 5.6 displays the success of the vital signs monitoring application.

Actual (BPM)	**Recorded (BPM)**
14.6	13.4
36.7	36.0
16.6	15

Table 5.1: Actual Breathing Rate vs. Recorded Breathing Rate in Breaths Per Minute

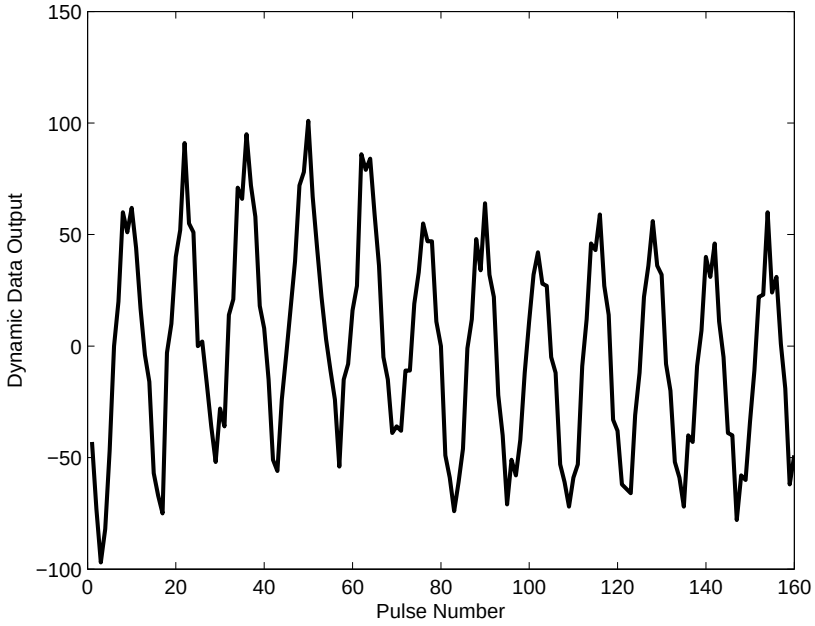


Figure 5.11: Time Domain Response of Pulse Multipath to Oscillation of a Metal Plate

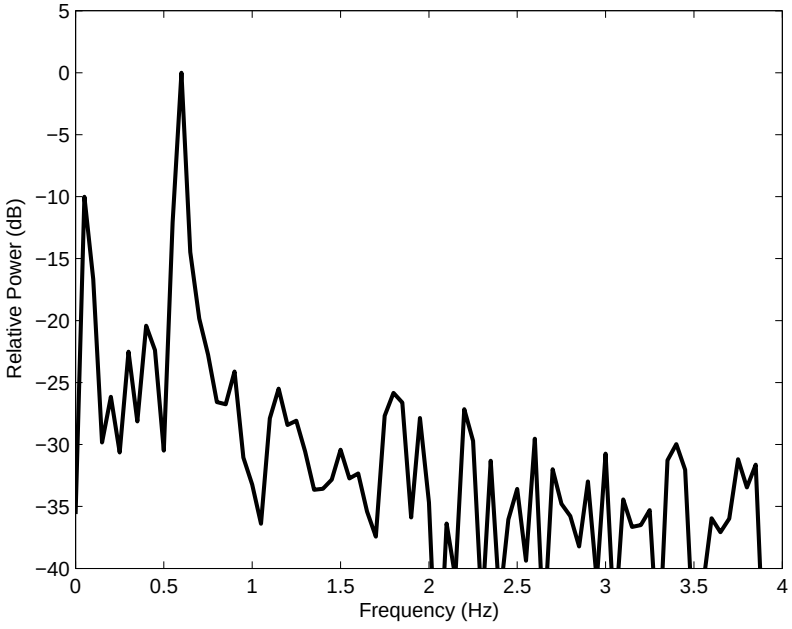


Figure 5.12: Frequency Domain Response of Pulse Multipath to Oscillation of a Metal Plate

Chapter 6

Conclusion

This thesis described a flexible, FPGA-based software-defined radio, with an emphasis on the FPGA design. The FPGA hardware and embedded software needed to implement the pilot-based matched filter receiver and the vital signs monitor were described. Results showed that the receiver was able to capture data from time-interleaved ADCs at a data rate of 6.4 G-samples/s with acceptable dynamic range. The feasibility of targeting impulse ultra-wideband signals with a time-interleaved sampling SDR was shown. The pilot-based matched filter transceiver system was created and was shown to be partially operational. The vital signs monitor was able to calculate a subject's breathing rate to within 1-9%.

This thesis contributed several key components of the UWB SDR:

- The hardware needed to receive and deinterleave data from eight time-interleaved ADCs was created.
- An interface from the ADC data to the embedded PowerPC was developed.
- The pilot-based matched filter hardware was debugged and interfaced with the FPGA hardware and the PowerPC software.
- Software to perform the acquisition phase for the receiver was developed.

- The hardware and filtering software for the remote vital signs monitor were developed.

There were several major challenges involved in this work. An understanding of high-speed FPGA design was necessary to create designs which met the timing demands of the system. Knowledge of FPGA clock generation and of limitations in the receiver system was needed to design hardware which properly deinterleaved the ADC sample data.

6.1 Future Work

The main goal of this project was to provide a test bed for ultra-wideband technology. This thesis provided FPGA hardware and software which created a simple interface to the ADC sample data. Researchers who wish to utilize this system need not worry about the FPGA timing and placement constraints for the data capture hardware or the synchronization issues inherent in a time-interleaved sampling system. An interface from the ADC data to the embedded Power PCs in the FPGA was also provided, which allows for software processing of ADC data on-chip. The door is now open for researchers interested in UWB technology to test new applications.

Bibliography

[1] C. Anderson, "A Software Radio Defined Ultra Wideband Transceiver Testbed for Communication, Ranging, and Imaging," PhD Dissertation, Virginia Polytechnic Institute and State University, Bradley Department of Electrical and Computer Engineering, Blacksburg, VA, Not yet Published.

[2] D. Agarwal, "An 8 GHz Ultra Wideband Transceiver Testbed," Masters Thesis, Virginia Polytechnic Institute and State University, Bradley Department of Electrical and Computer Engineering, Blacksburg, VA, October 2005.

[3] D. Porcino and W. Hirt, "Ultra-Wideband Radio Technology: Potential and Challenges Ahead," *IEEE Communications Magazine*, vol. 41, no. 7, pp. 66–74, 2003.

[4] R. J. Fontana, E. Richley, and J. Barney, "Commercialization of an ultra wideband precision asset location system," in *2003 IEEE Conference on Ultra Wideband Systems and Technologies*, 2003.

[5] L. Yang and G. B. Giannakis, "Ultra-Wideband Communications: An Idea Whose Time Has Come," *IEEE Signal Processing Magazine*, vol. 21, no. 6, pp. 26–54, 2004.

[6] C. Moy, S. Paquelet, A. Bisiaux, and A. Kountouris, "A SDR Ultra-Wideband Impulse Communication System for Low and High Data Rates."

[7] A. Alsolaim, J. Starzyk, J. Becker, and M. Glesner, "Architecture and application of a dynamically reconfigurable hardware array for future mobile communication systems,"

in *FCCM '00: Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 205–214.

[8] J. Helmschmidt, E. Schuler, P. Rao, S. Rossi, S. di Matteo, and R. Bonitz, "Reconfigurable signal processing in wireless terminals," in *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 202–44.

[9] K. Kuusilinna, C. Chang, M. J. Ammer, B. C. Richards, and R. W. Brodersen, "Designing BEE: A Hardware Emulation Engine for Signal Processing in Low-Power Wireless Applications," in *EURASIP Journal on Applied Signal Processing*, vol. 6, 2003.

[10] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: a High-End Reconfigurable Computing System," in *Design & Test of Computers*, vol. 22. IEEE Computer Society, 2005, pp. 114–125.

[11] C. Chang, K. Kuusilinna, B. Richards, A. Chen, N. Chan, R. W. Brodersen, and B. Nikolic, "Rapid Design and Analysis of Communication Systems Using the BEE Hardware Emulation Environment," in *Proceedings 14th IEEE International Workshop on Rapid Systems Prototyping*, 2003, pp. 502–513.

[12] S. M. Mishra, D. Cabric, C. Chang, D. Willkomm, B. van Schewick, A. Wolisz, and R. W. Brodersen, "A Real Time Cognitive Radio Testbed for Physical and Link Layer Experiments," in *In Proceedings of the IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN 2005)*, 2005.

[13] W. Tuttlebee, *Software Defined Radio: Enabling Technologies*. John Wiley and Sons, Inc., 2002.

[14] S. M. Blust, *Perspective on Software Defined Radio - Download and Reconfigurability for Radio Software*, SDR Forum, June 2002.

[15] W. Tuttlebee, *Software Defined Radio: Origins, Drivers, and International Perspectives*. John Wiley and Sons, Ltd., 2002.

[16] P. G. Cook and W. Bosner, "Architectural Overview of the SPEAKeasy System," vol. 17, no. 4. IEEE Computer Society, 1999.

[17] S. P. Reichhart, B. Youmans, and R. Dygert, "The Software Radio Development System," in *Personal Communications*, vol. 6, 1999.

[18] R. F. Higgins and C. C. Herndon, "JCIT, a Production-Ready Field Tested Non-Proprietary Software Definable Radio," in *MILCOM 2000. 21st Century Military Communications Conference Proceedings*, vol. 1, 2000.

[19] J. M. III, "SDR architecture refinement for JTRS," in *MILCOM 2000. 21st Century Military Communications Conference Proceedings*, vol. 1, 2000.

[20] "JTRS JPEO," http://enterprise.spawar.navy.mil/body.cfm?type=c&category=27&subcat=60.

[21] *Software Commuications Architecture Specification*, JTRS, May 2006.

[22] J. Kulp and M. Bicer, "Integrating Specialized Hardware to JTRS/SCA Software Defined Radios," in *Proceedings of the 2005 IEEE Military Communications Conference (MILCOM 2005)*. IEEE Computer Society, 2005.

[23] "Revision of Part 15 of the Commission's Rules Regarding Ultra-Wideband Transmission Systems, First Report and Order," pp. 3–16, Adopted February 14, 2002, Released April 22, 2002.

[24] R. J. Fontana, "Recent system applications of short-pulse ultra-wideband (UWB) technology, journal = IEEE Transactions on Microwave Theory and Techniques, year = 2004, volume = 52, number = 9, pages = 2087–2104,."

[25] "WiMedia Alliance," http://wimedia.org.

[26] G. Breed, "Wireless USB Uses Ultra Wideband (UWB) for High Data Rate," in *High Frequency Electronics*, September 2005.

[27] "USB.org - Certified Wireless USB," http://www.usb.org/developers/wusb/.

[28] M. Ghavami, L. Michael, and R. Kohno, *Ultra Wideband Signals and Systems in Communication Engineering.* John Wiley and Sons, Ltd., 2002.

[29] J. H. Reed, *An Introduction to Ultra Wideband Communication Systems.* Prentice Hall, 2005.

[30] "Part 15 - Radio Frequency Devices," Adopted February 16, 2006.

[31] *Virtex-II Pro and Virtex-II Pro X Platform FPGAs*, Xilinx, October 2005.

[32] E. Lorden, "Untitled," Masters Thesis, Virginia Polytechnic Institute and State University, Bradley Department of Electrical and Computer Engineering, Blacksburg, VA, Not yet Published.

[33] R. C. Nixon, *Spread Spectrum Systems With Commercial Applications*, 3rd ed. John Wiley and Sons, Inc., 1994.

[34] S. Venkatesh, C. R. Anderson, N. V. Rivera, and R. M. Buehrer, "Implementations and Analysis of Respiration-Based Estimation Using Impulse-Based UWB," in *Proceedings of the 2005 IEEE Military Communications Conference (MILCOM 2005).* IEEE Computer Society, 2005.

[35] C. Anderson, S. Venkatesh, R. M. Buehrer, and J. H. Reed, "Theoretical Analysis and Preliminary Results of and 8 GHz Time-Interleaved ADC Array for a Software Defined UWB Receiver."

[36] W. C. Black, "High Speed CMOS A/D Conversion Techniques," PhD Dissertation, University of California, Berkeley, December 1980.

[37] W. C. Black and D. A. Hodges, "Time Interleaved Converter Arrays." IEEE Computer Society, 1980, pp. 14–15.

[38] N. Kurosawa, H. Kobayashi, K. Maruyama, H. Sugawara, and K. Kobayashi, "Explicit Analysis of Channel Mismatch Effects in Time-Interleaved ADC Systems," vol. 48, no. 3. IEEE Computer Society, 2001, pp. 261–271.