A Raw Processor Interface to an 802.11b/g RF Front End

by

Benjamin Philip Eugene Zaks Walker

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2004

©Benjamin Philip Eugene Zaks Walker, MMIV. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 18 2005
LIBRARIES

Author .....
Department of Electrical Engineering and Computer Science
August 5, 2004

Certified by .....
Anant Agarwal
Professor, CSAIL
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

BARKER

A Raw Processor Interface to an 802.11b/g RF Front End

by

Benjamin Philip Eugene Zaks Walker

Submitted to the Department of Electrical Engineering and Computer Science
on August 5, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Raw microprocessor is a tiled architecture processor designed by the Computer Architecture Group at MIT. Raw was designed in part to be extremely good at performing streaming-type algorithms such as those found in wireless communciations processing. This thesis describes the design and implementation of an interface between the Raw microprocessor and a modified 802.11a/b/g wireless access point. Combined with the Raw processor, this interface replaces a custom digital chip in the access point. The interface can be used with the Raw microprocessor to perform communications research, and as a system demonstrating Raw's capabilities. The interface has been built, tested, and functionally qualified.

Thesis Supervisor: Anant Agarwal
Title: Professor, CSAIL

Acknowledgments

I would first like to thank Professor Agarwal for his time and support in supervising this thesis. I cannot thank Nathan Shnidman enough for his help, time, and excellent guidance along the way. Thanks also to Jason Miller, Dave Wentzlaff and the rest of the Raw team. Without the infrastructure they built, this thesis would not have been possible. Thanks to Justin Morin for his help in lab. Thanks also to Dave Shoemaker, Ron Cook and Eddie Rubin of Engim, Inc. for their generosity and time.

Thank you to my family for supporting me in everything I've done, and for giving me every opportunity they could. Thank you Jeremy, Sean, Jen, Paul and Ravi. The past year at 3 Leonard was that much better because you were a part of it. Thank you to Miranda. Without your support I would not be here.

This research was supported by DARPA, the NSF, and the Oxygen alliance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the use of wireless systems has grown rapidly. In particular, wireless LANs for mobile computer use have become increasingly popular, with a new high-datarate standard, 802.11g [3], released this year. As the popularity of wirelessly connected computers has increased, so has the demand for bandwidth, to open up new application domains, and to increase the competitiveness of wireless vs. wired systems. Current standards allow for datarates of up to 54Mb/s, but researchers are working on wireless LAN technology with datarates of up to 1Gb/s [1].

In addition to a push for higher datarates is an interest in flexible wireless systems. An all-software GSM base station was recently demonstrated by Vanu, Inc. [6]. This system's flexibility lies in its software implementation, which allows it to be easily upgraded and modified.

The push for higher datarates and more flexibility means that wireless systems are an active research area with many opportunities and challenges. The software algorithms required for wireless systems are also *streaming applications*, an increasingly important group of applications.

A streaming application is one where the dataset is assumed to be semi-infinite in size (the stream of data begins at some time t, but continues on potentially forever after t) and the data being operated on at any given time is only a small consecutive set of the stream. In these types of applications, processors continuously take data in, process it, and then direct the results. Other examples of streaming applications

(besides wireless systems) are realtime video processing and speech processing.

Current general-purpose processor architectures are not well suited to streaming applications. They often suffer from very limited input/output (I/O) bandwidth, and an inability to efficiently take advantage of certain types of parallelism. As an example, it can be difficult to input large quantities of data for processing, because of the processor's complex memory hierarchy. In streaming applications such as 802.11 wireless systems, a customized processor or application-specific integrated circuit (ASIC) is often used in order to overcome the above limitations [5]. These ASICs often consist of custom programmable hardware blocks, augmented with general-purpose processor cores. This situation can result in a lack of flexibility because of the use of fixed custom hardware blocks.

This lack of flexibility means that many wireless systems cannot easily adapt to using new standards for communication or take advantage of optimizations requiring more flexibility than that defined in a particular standard. For instance, certain modulation schemes might be more power-efficient than others depending on environmental conditions. A system that could switch between modulation schemes like this could gain an advantage over a less flexible system.

The Raw processor [7] was designed to be able to process streams of data more efficiently than current processors, in addition to being able to perform more general-purpose tasks, providing an important advantage over today's general-purpose processors.

Raw contains a tiled array of processors, each of which is capable of executing an independent stream of instructions. This structure allows Raw to very efficiently perform parallel computation. In addition, Raw contains several high-speed, low-latency, on-chip networks that connect individual tiles to one another. These networks extend off the edge of the chip to allow Raw to interface easily with the outside world and provide huge amounts of I/O bandwidth. These features allow Raw to very efficiently perform streaming-type operations.

Raw's ability to process streams of data in a flexible fashion makes it well-suited to meet the current challenges in wireless systems. Thus, a project was started to

create a wireless system using Raw as the digital processing element. This system requires an interface connecting the Raw system to the wireless front end. This thesis describes the design and implementation of such an interface.

1.1 Thesis Statement

This thesis describes the design and implementation of a Raw processor interface to an 802.11 radio frequency (RF) front end. The interface described by this thesis will be used in a system that demonstrates Raw performing the baseband digital signal processing for the 802.11g wireless standard. This demonstration system will show that Raw is capable of performing the DSP-like computation found in wireless baseband signal processing. In addition the flexibility of the RF front end permits this system to be used in communications research, where Raw serve as a substrate on which to implement novel ideas in digital communications.

The remainder of this thesis is organised as follows: Chapter 2 provides background information on the 802.11 wireless local area network (LAN) standard and briefly discusses how an 802.11 system could be built. Chapter 3 provides an overview of the demonstration system in order to place the interface in context. The next two chapters go into the details of the blocks in the interface. Chapter 4 describes the circuitry added to the RF front end to interface it to the Raw system. Chapter 5 describes the wireless board controller. Chapter 6 describes the implementation of the interface, and goes over the steps taken to test it. Chapter 7 describes the significant accomplishments to date, and future work for the interface. Chapter 8 concludes.

Chapter 2

Background

This chapter provides some background information on the 802.11 wireless LAN standard, and briefly discusses how 802.11 signals are processed.

2.1 802.11

802.11 is a wireless local area network (LAN) standard [2] from the Institute of Electrical and Electronics Engineers (IEEE) that defines methods to allow computers to connect to each other wirelessly. The 802.11 standard can be broken into two major layers: a Medium Access Control (MAC) layer and a physical (PHY) layer. The MAC layer defines a protocol that organises communication activity to ensure that each user of the network is given an opportunity to transmit data. The PHY layer determines how the data is actually transmitted (i.e. how it is formatted, encoded, and modulated onto an RF carrier). The PHY layer also specifies on what frequencies users can transmit data.

There are three commonly used wireless PHY layers for 802.11: 802.11a, 802.11b and 802.11g. 802.11a operates in the 5GHz band, and specifies an Orthogonal Frequency Division Multiplexing (OFDM) technique for transmitting data. This PHY allows datarates of up to 54Mb/s. 802.11b operates in the 2.4GHz band, and specifies a Direct-Sequence Spread Spectrum (DSSS) technique for transmitting data at rates up to 11Mb/s. Finally, 802.11g also operates in the 2.4GHz band, but uses a

transmission technique similar to that of 802.11a, which allows datarates of up to 54Mb/s.

The demonstration system is designed to use the 802.11g standard.

2.2 802.11 Processing Overview

The processing required in 802.11 wireless LAN systems can be broken into four different blocks (See Figure 2-1):

1. RF transceiver for performing analog RF upconversion/downconversion.
2. ADC/DAC for performing conversion between the analog and digital domains.
3. A baseband block implementing the digital processing required as part of the 802.11 PHY layer.
4. A MAC block implementing the digital processing for the 802.11 MAC layer.

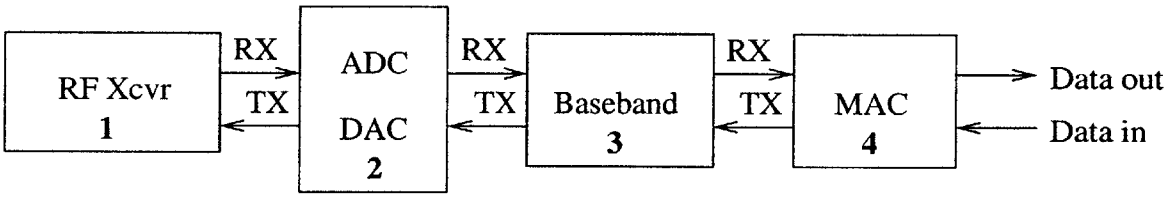


Figure 2-1: Block diagram showing a breakdown of the processing elements in an 802.11 system.

These four blocks are usually built in multiple chips, although there are some systems that provide all four blocks on a single chip. Often, blocks 2-4 or 3-4 are implemented on the same chip. In particular, blocks three and four are usually implemented as custom hardware with some built-in programmability [4] [5].

The overall system described in this thesis replaces blocks 3 and 4 with the Raw processor. The key difference in using the Raw processor as the digital processing element is its flexibility. Since Raw's basic computational elements are general-purpose processing blocks, these blocks can be easily reconfigured to optimize processing at different levels of abstraction.

Chapter 3

System Overview

3.1 Overall System

This section describes the overall system in which the interface will be used, in order to provide a context for understanding the design of the interface.

The goal of the overall system is to demonstrate Raw performing the baseband processing for the 802.11g wireless standard.

3.1.1 System Description

The overall system consists of three major logical blocks: an RF front-end, a software processing section, and the interface, which is designed to allow the first two blocks to communicate. Figure 3-1 shows the relationship between these three blocks. Comparing Figures 3-1 and 2-1, note that the interface block and the software processing block replace the baseband and MAC blocks of the 802.11 processing chain.

The three blocks listed above are physically implemented on two different Printed Circuit Boards (PCBs). The first PCB, the Raw wireless board, is a modified 802.11a/b/g access point board provided by Engim, Inc. This board contains the circuitry for the RF front-end, as well as some additional components added as part of the interface. The second PCB, the Raw Handheld board, contains the Raw processor, as well as some additional components such as DRAMs and FPGAs which

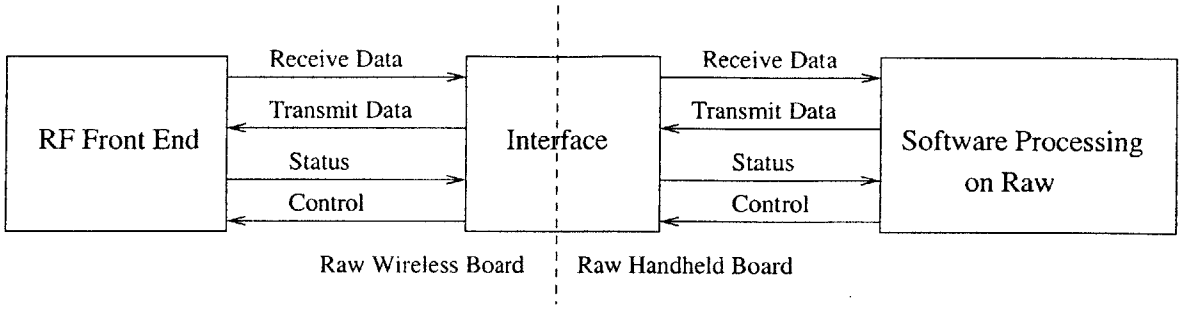


Figure 3-1: Block diagram of the 802.11g demonstration system. The vertical dashed line indicates on which PCB the different blocks are implemented.

support the operation of the Raw processor. Portions of the interface are also implemented in an FPGA on the Raw Handheld board.

3.2 RF Front End

The RF front end is implemented on a modified 802.11a/b/g access point from Engim, Inc. Figure 3-2 shows a chip-level diagram of the front end.

3.2.1 RF Front End Operation

The front end consists of two separate transmit/receive paths, one for the 2.4GHz 802.11b/g standards, and one for the 5GHz 802.11a standard. Each path consists of an RF transceiver and an ADC/DAC chip. The RF transceiver deals with down-conversion/upconversion and any filtering that needs to be done on the baseband or RF signals. The ADC/DAC chip converts analog received baseband signals to digital, and provides dual DACs for converting the in-phase and quadrature digital baseband signals to analog. In this thesis, only the 2.4GHz (802.11b/g) section was used, although the interfaces to the two sections are identical. The 2.4GHz section also contains a delay line which feeds back a portion of the transmit signal to the transceiver. This can be used for transmit cancellation in the receiver.

Figure 3-3 shows the processing performed by the 802.11b/g RF front-end section.

Specifically, the RF transceiver chip modulates and demodulates signals in the 2.4GHz range, with a 76MHz baseband analog bandwidth. The ADC section of the

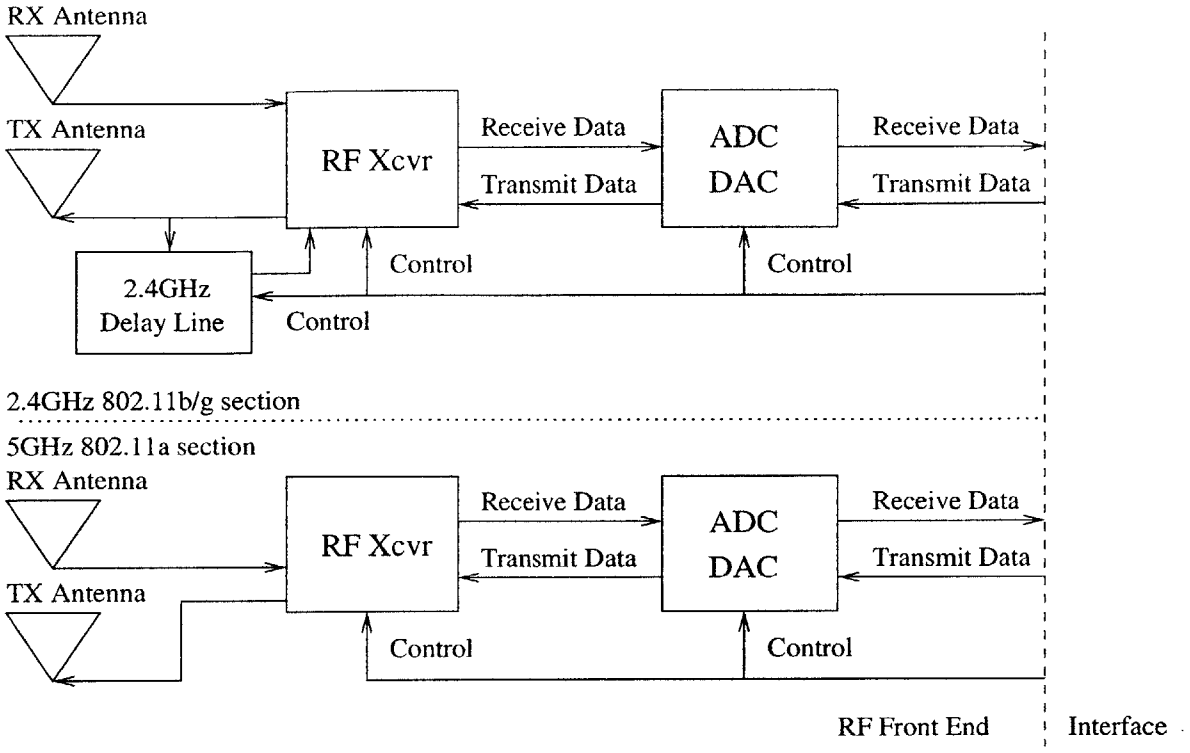


Figure 3-2: Chip-level diagram of the RF front end. Note that the front end contains two identical paths, one for 802.11a (5GHz band), and one for 802.11b/g (2.4GHz band).

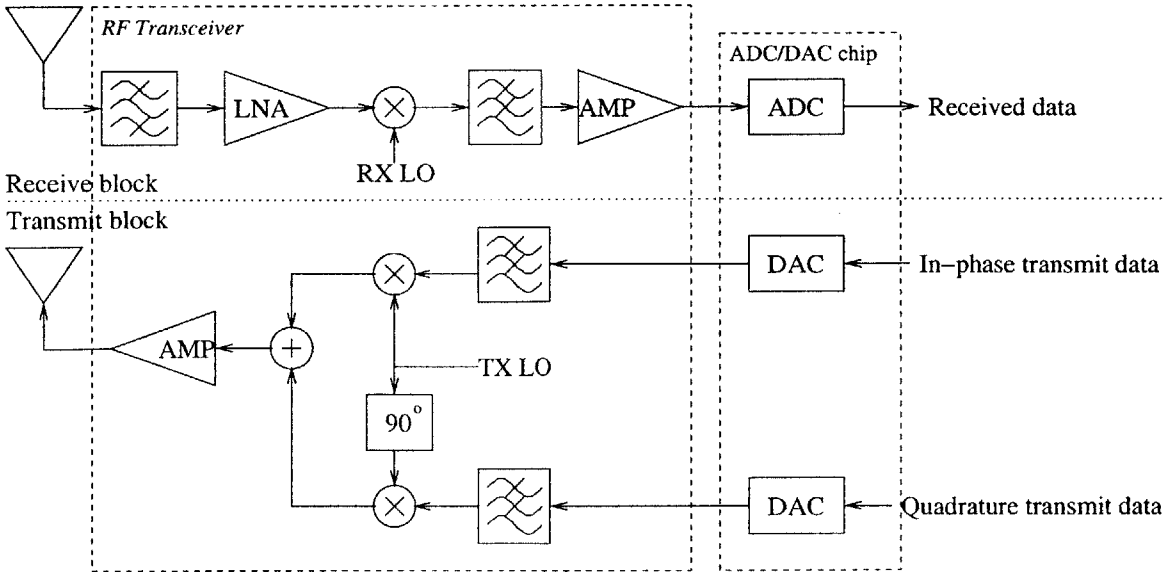


Figure 3-3: Diagram showing processing performed by 802.11b/g RF front-end.

ADC/DAC chip has a sampling rate of 180MSa/s with a 12-bit resolution. The DAC section contains two 10-bit DACs with a sampling rate of 180MSa/s. Each of the DACs is capable of handling signals containing frequencies up to 40MHz. A useful feature of this front-end is the wide baseband bandwidth that is used. A typical 802.11g channel occupies 20MHz of bandwidth. The received baseband bandwidth allows the digital processing system simultaneous access to up to 3 channels worth of data. This additional information can be an advantage in terms of overall throughput and when designing digital processing algorithms.

3.2.2 Control/Data Signals

ADC/DAC Data Signals

The ADC has a 12-bit digital output that uses 1.5V LVCMOS signalling technology. The 12-bit digital output of the ADC is accompanied by a 180MHz clock signal. The two DACs have 10-bit digital interfaces that use 1.5V LVCMOS signalling technology. A 180MHz sampling clock is also provided by the ADC/DAC chip for the DAC interface.

ADC/DAC Control Signals

The ADC/DAC chip is controlled by a 3-wire serial interface. The three lines consist of a clock line, a data line, and a serial enable line. This serial interface is used to write values to the 32-bit control register in the chip. To begin a transaction, the serial enable line is first pulled low, and then 32 rising edges of the clock line are used to sample the data line. After this the serial enable line is pulled high. A simplified timing diagram is provided in Figure 3-4.

There is also a reset signal that is connected to the ADC/DAC chip that can be used to reset the device. The BOARD_RESET signal is connected to a switch on the wireless board, and to a control line from the wireless controller.

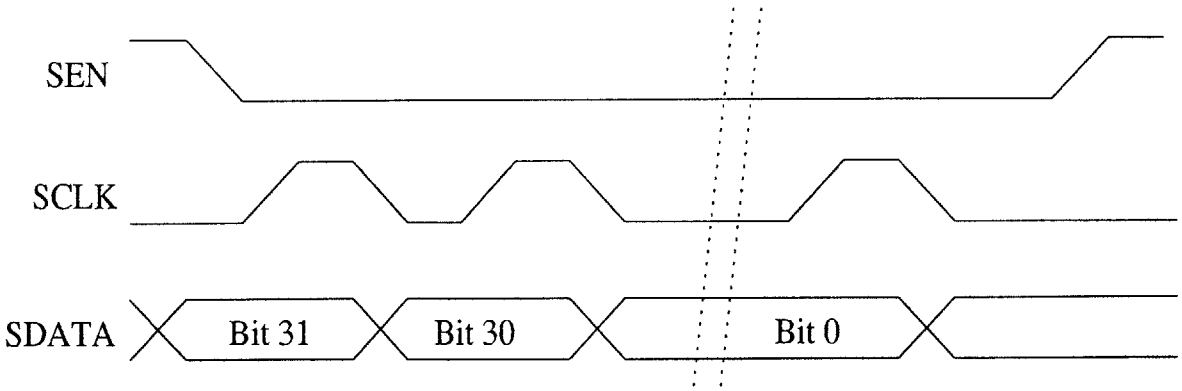


Figure 3-4: Timing diagram for 3-wire serial control interface on RF and ADC/DAC chips.

3.2.3 RF Control Signals

The RF chip has two different types of control interface. One is a 3-wire serial interface identical to that on the ADC/DAC chip. The other consists of parallel buses of one or more lines.

A key difference between the RF serial interface and the ADC/DAC chip serial interface is in the number of control registers each chip contains. The ADC/DAC chip contains a single 32-bit control register, while the RF chip contains 8 29-bit control registers. In an ADC/DAC serial transaction, all 32 bits sent on the serial bus are written to the control register. In an RF serial transaction, the last 3 bits sent are used to determine which register is to be written, and the other 29 bits are written to that register.

The parallel buses are used for automatic gain control (AGC) and diversity selection and can be pulled high or low at any time without reference to any type of clock signal.

The RF transceiver has a single-bit status output, RF_LOCK, that indicates whether either the receive or transmit PLL is locked.

The 2.4GHz delay line chip also has a parallel control bus, RF_DLY, which can be set to change the length of the delay line.

3.3 Software Processing Section

The software processing block is implemented on the Raw processor on the Raw Handheld board. In the demonstration system, this block is responsible for performing the digital baseband processing as specified by the 802.11g standard. However, for general wireless research, there are no set specifications for this block. It is only responsible for producing digital data to be sent to the DAC, and processing the received data from the ADC.

3.3.1 Raw Processor

The Raw processor [7] is a tiled architecture consisting of a 4x4 grid of tiles (see Figure 3-5). Each tile contains a tile processor, a static switch, and a dynamic network router. Each tile processor can independently execute its own set of instructions, and contains its own data and instruction cache. Tile processors are connected to each other by four on-chip networks. Two of these networks are 'static' networks, which means that the static switches at each tile must be programmed to perform the correct routing. The other two networks (the Memory Dynamic Network (MDN) and the General Dynamic Network (GDN)), are dynamic. These networks take in blocks of data (messages) and dynamically determine how to route the message from its source to its destination. Each of these networks uses the SIB protocol to ensure that no data is lost. See [7] for more information on the SIB protocol.

On the edges of the chip, the networks extend off the side of the Raw chip. This allows Raw to directly interface with devices that can speak the SIB protocol. The Raw chip is mounted on a motherboard known as the Raw Handheld board.

3.3.2 Raw Handheld board

The Raw Handheld board contains the Raw processor surrounded by 6 field programmable gate arrays (FPGAs) connected to the 16 ports on the Raw chip (see Figure 3-6). These FPGAs are used to communicate with the Raw chip and to connect it to various external devices.

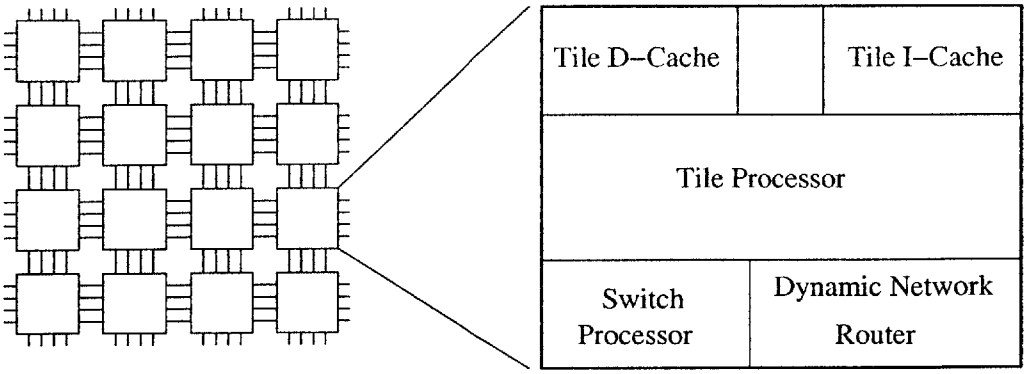


Figure 3-5: Figure showing the Raw processor.

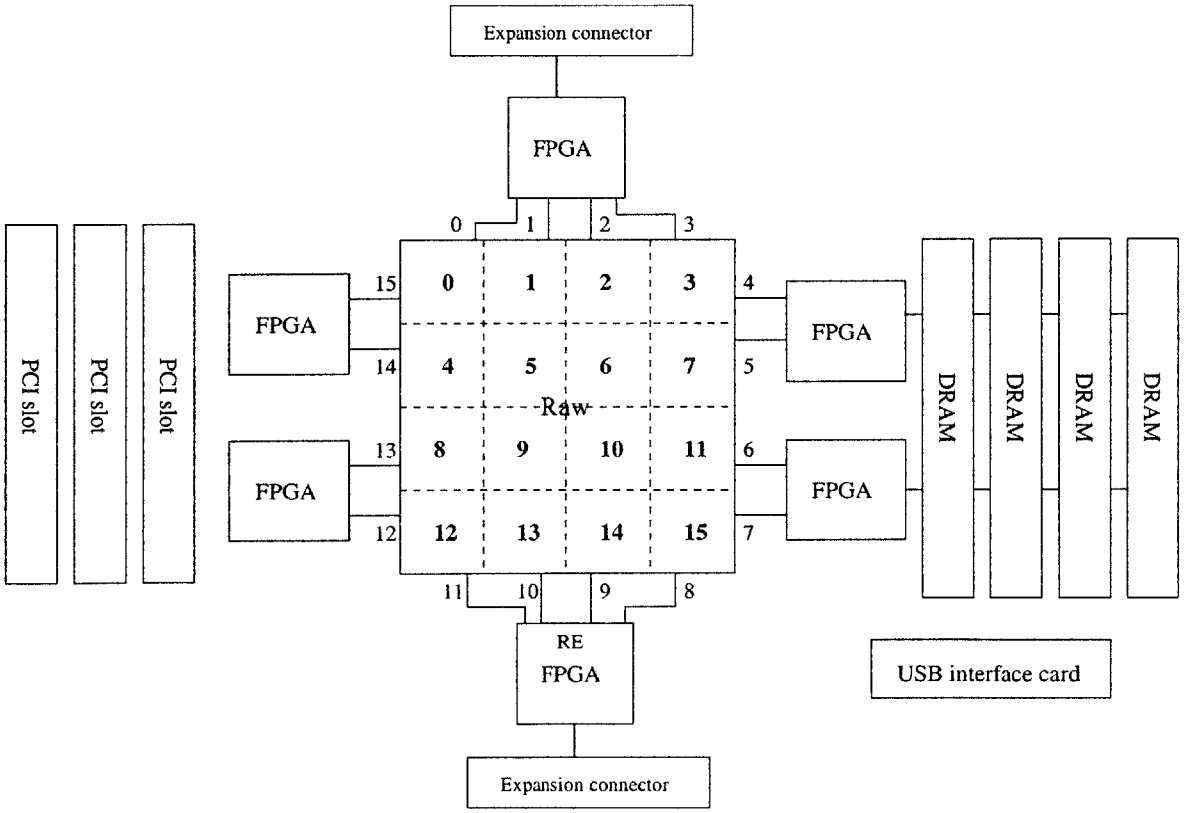


Figure 3-6: Block diagram of the Raw Handheld board. Note that the Raw processor has been divided into 16 tiles by dashed lines. Bold numbers in the center of the tiles indicate the tile's number; numbers on the outside edges of the Raw chip indicate port numbers.

Two of the FPGAs (connected to ports 4-7) act as memory controllers, and allow Raw to access four 512MB Dynamic Random Access Memory (DRAM) chips on the Handheld board. Another FPGA is used to connect Raw to a computer via a Universal Serial Bus (USB). This USB interface is used to load programs onto Raw and allow data to be transmitted to and received from Raw.

Another FPGA, designated as RE, is connected to Raw ports 8-11, and to a 190-pin high-speed matched-impedance connector. This connector is what is used to mechanically and electrically connect the Raw wireless board to the Raw Handheld board. The controller portion of the interface is implemented on the RE FPGA.

3.4 Interface

The interface, which is the subject of this thesis, can be broken down into two major blocks: a wireless board interface and a wireless board controller. Figure 3-7 shows these two blocks in the context of the overall system.

The wireless board interface consists of chips and components on the Raw wireless board and is used to interface the wireless board to the Raw Handheld board electrically and mechanically.

The wireless board controller is implemented in an FPGA on the Raw Handheld board and is used to drive the control lines of the wireless board. It also takes transmit data from Raw and sends it to the wireless board, and takes receive data from the wireless board and sends it to Raw. The wireless board controller interfaces with the Raw processor so that software running on Raw can control the operation of the wireless board.

The next two chapters describe these two blocks.

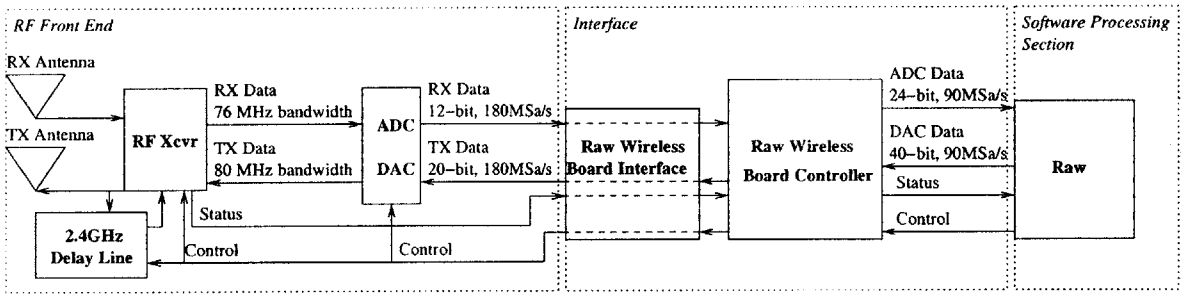


Figure 3-7: Block diagram showing the primary components of the overall system and datarates of the primary datapaths. The dotted lines indicate the three logical blocks of which the overall system is composed (RF Front End, Interface, Software Processing Section).

Chapter 4

Raw Wireless Board Interface

This chapter describes the Raw wireless board interface, which is implemented on a modified Engim, Inc. access point. See Figure 4-1 for a chip-level diagram.

4.1 Design Goals

The modifications to the Engim board design were made to allow the board to interface mechanically and electrically with the Raw Handheld board and to remove the Engim digital processing element. There were five primary design goals that needed to be met:

1. The FPGA on the Raw Handheld board is set up to use a 3.3V signalling technology, while the outputs of the ADC and the inputs to the DACs use a 1.5V LVCMOS signalling technology. Thus, some conversion mechanism was needed to interface the two chips.

2. The inter-board connection between the Raw wireless board and the Raw Handheld board, along with the additional wiring distance on both boards, could compromise the signal integrity of the ADC outputs and the DAC inputs since these two groups of signals are very high speed (180MSa/s). The outputs of the ADC are also slew-rate limited to reduce digital noise; however, slew-rate

limiting reduces the capacitive load that they can drive. Thus, signal integrity issues needed to be considered in the design.

3. There is a significant time delay from the output of the FPGA on Raw to the input of the DACs on the Raw wireless board. If the signals from the FPGA are not correctly synchronized to the DAC sampling clock, the signals that arrive at the DACs could violate the setup or hold time requirements of the DAC inputs.

4. A sampling clock for the ADC outputs is provided by the ADC/DAC chip. Since the sampling clock is also travelling with the data, it is important that the skew introduced between the clock and the data by any interface circuitry be minimized.

5. Changes to the analog sections of the Engim board should be as minimal as possible.

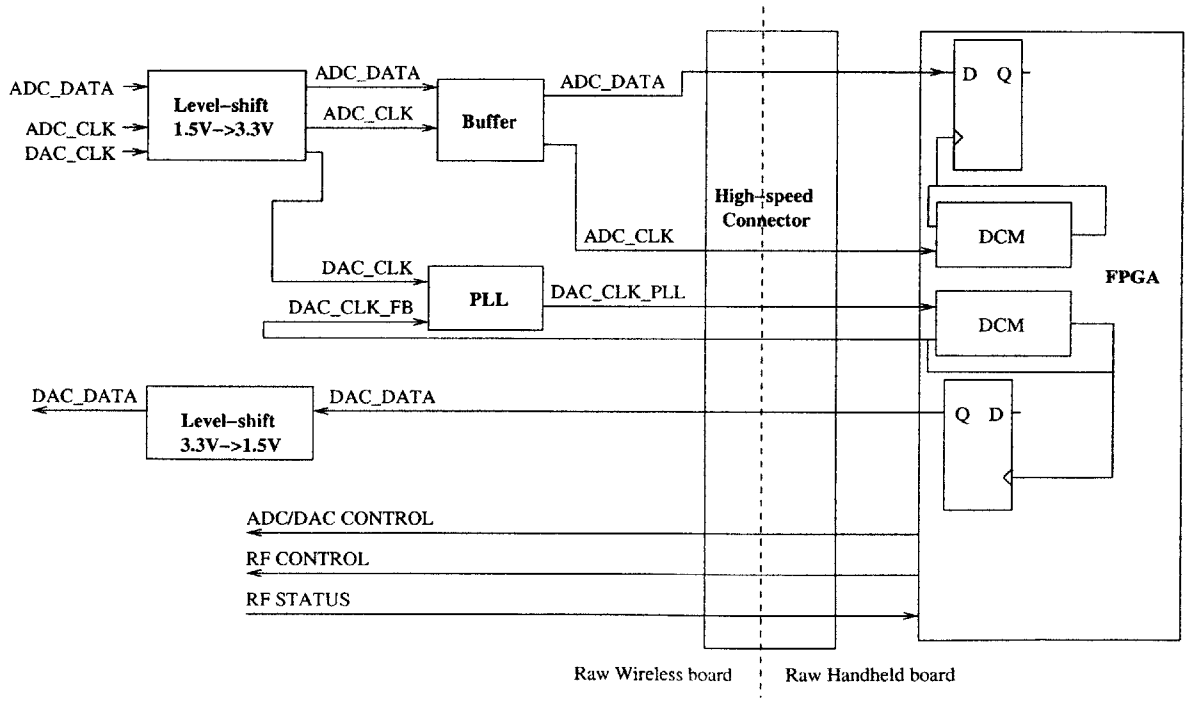


Figure 4-1: Block diagram showing the Raw wireless board interface. The FPGA block was included to aid understanding of the interface design. The blocks labelled DCM are Digital Clock Managers, and are used on the FPGA to de-skew clock signals and to provide any necessary phase adjustment.

4.2 Design

The design was chosen to minimize the complexity of the additional circuitry added, and thus reduce the possibility for errors in the hardware. In particular, the analog section was untouched, since that part of the circuitry is more sensitive to changes, and would have required significant effort to modify it.

The design consists of digital level-shifting chips, coupled with buffers in the case of the ADC outputs, and a phase-locked loop (PLL) which is used to synchronize the output of the FPGA to the input of the DAC. Also, a high-speed matched-impedance connector was used to connect the two PCBs together. This connector presents a 50Ω impedance and is useful for high-speed signals where impedance matching is important.

The remaining subsections describe how the design works for different groups of signals.

4.2.1 ADC Outputs

The outputs of the ADC are first level-shifted from 1.5V to 3.3V. The 3.3V signal is then fed into a buffer that drives the signal from the wireless board to the Handheld board. Since there is no termination at the Handheld board, the buffers chosen feature a dynamic output impedance which is designed to minimze reflections when driving unterminated transmission lines. One thing to note about this design is that even if the overall propagation delay from the ADC outputs to the inputs of the FPGA exceeds the clock period ($\approx 5.5ns$), there won't be any setup time violations. The reason is that since the clock is being forwarded along with the data, any delay in the data is ideally also seen in the clock signal. Thus, as long as the delays through the data lines and the clock line are matched, there should be no setup or hold violations at the FPGA inputs. This assumes that reflections on the signal lines are minimal.

Finally, the ADC_CLK signal is sent into a Digital Clock Manager (DCM) on the FPGA which provides a mechanism to shift the phase of its output clock with respect to its input clock. This provides some control over the phase of the clock used for the

ADC_DATA input flip-flops.

4.2.2 DAC Inputs

The DAC inputs come from flip-flops on the FPGA. The flip-flop outputs then pass through the expansion connector and are level-shifted from 3.3V to 1.5V by chips on the wireless board. The 1.5V outputs of the level-shifters are then sent into the DAC. Since there is a significant delay from the output of the FPGA to the input of the DAC, it is important that the clock used for the output flip-flops of the FPGA be synchronized correctly with the DAC_CLK provided by the ADC/DAC chip. This is achieved by adding a PLL chip on the wireless board. The PLL skews its output clock enough such that the DAC_DATA signals arrive at the correct time with respect to DAC_CLK.

In addition to ensuring the correct synchronization between the FPGA outputs and the DAC sampling clock, the signal integrity (SI) of the outputs from the FPGA must be taken into account since they are high speed, and they have to travel between two PCBs. In order to overcome any SI issues, the digitally controlled impedance (DCI) functionality of the FPGA was used. DCI is a feature on Virtex2 devices that provides controlled-impedance drivers for various signalling standards. In this case, the signalling standard used is 3.3V LVCMOS, and the output impedance of the drivers is 50Ω, which matches the connector impedance.

4.2.3 Wireless Board control signals

All of the control signals for the wireless board use 3.3V LVCMOS signalling technology. Since the FPGA is configured to use this signalling technology, and the control signals are all low-speed, no special circuitry was added or required.

Chapter 5

Wireless Board Controller

5.1 Design Goals and Overview

The wireless board controller for the interface is implemented in firmware on a Xilinx Virtex2 FPGA. The controller was written in Verilog and then synthesized to the FPGA hardware. The controller is responsible for three main tasks:

1. Taking the digitized receive data from the ADC, formatting it for use in Raw, and sending it to the Raw chip.

2. Taking the digital transmit data from the Raw chip, formatting it for use by the RF front end, and sending it to the DACs.

3. Controlling operation of the chips in the RF front end.

These overall goals are also supplemented by two additional design goals:

1. Make the controller as simple as possible. One of the goals of the demonstration system is to prove that the Raw chip is capable of performing the necessary processing for an 802.11g system. Providing minimal support in the FPGA, and thus using Raw for most of the control helps support this argument. Also, fixing bugs on the FPGA is more time-consuming than bugs in software on Raw. For example, recompiling the FPGA code for this project takes approximately 45 minutes, while recompiling the Raw controller code takes less than a minute.

2. Keep the interface to Raw simple, while still exposing most of the 'knobs' in the control interface to software running on Raw. The control interface provides access to many different variables such as receive local oscillator (LO) frequency, transmit LO frequency and low noise amplifier (LNA) gain. It is important that the software system on Raw be able to adjust as many of these variables as possible to make the system useful for wireless research and to implement the MAC layer on Raw.

These goals resulted in the following overall design for the controller (Figure 5-1). The controller is split into four sections: the Control section, the ADC section, the DAC section, and the SpeedGasket. The SpeedGasket, designed by David Wentzlaff, provides a high-speed asynchronous interface to the Raw processor. The Control section drives the control lines for the wireless board, and presents an interface to Raw for controlling the chips on the wireless board, and the ADC and DAC sections of the controller. The Control section receives commands from Raw, and then implements those commands. This command interface sends and receives data on the static network on Port 11 on Raw.

The ADC section takes the digitized receive data from the ADC chip and formats it for use in Raw. Data from the ADC section is streamed into the static network on Port 9 on Raw. The DAC section takes the digital transmit data from Raw, and formats it for use in the DAC chip. Data for the DAC section is streamed from the static network on Port 8 on Raw.

It is important to note that the controller is a passive piece of hardware; it only processes commands from Raw, and will not effect any changes other than through commands from Raw. The ADC and DAC sections are similar in that they will only send data to Raw or to the DACs upon receiving a command from the Control section (which ultimately is sent by Raw).

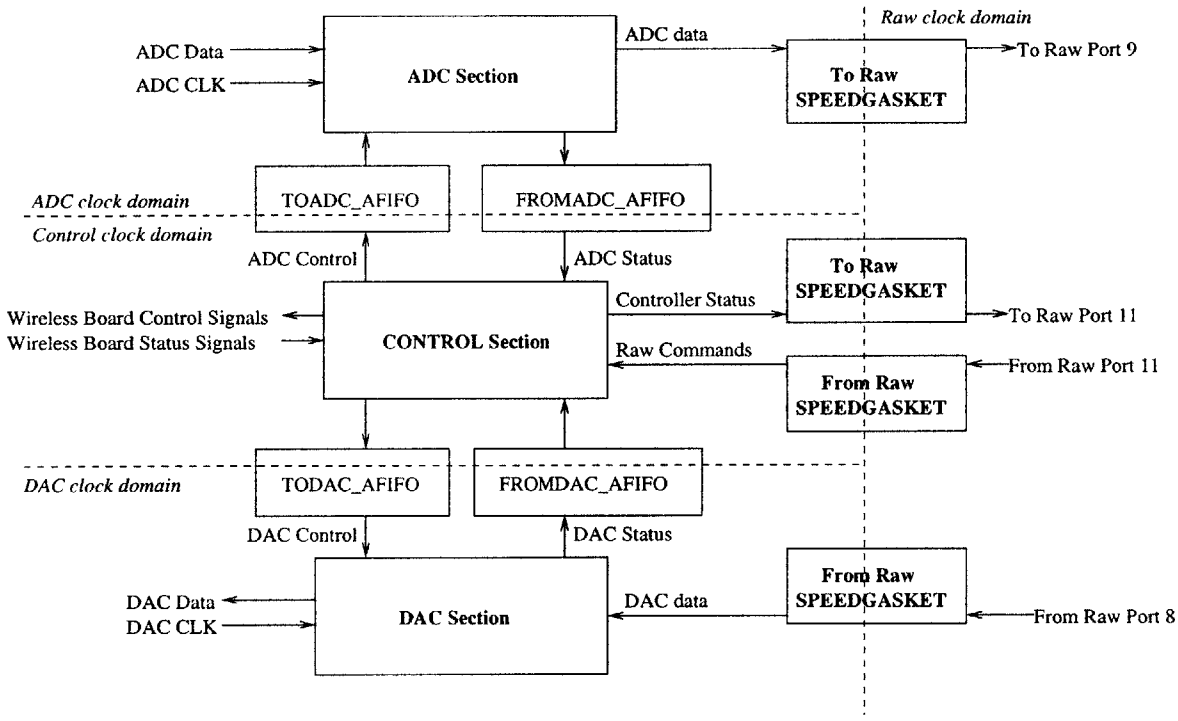


Figure 5-1: Block diagram showing the overall design of the controller. Dashed lines delineate boundaries between different clock domains. Note that the DAC_CLK signal is provided by the ADC/DAC chip and is an *input* to the DAC section.

5.1.1 Controller Interface to Raw

In order to make the software interface to the controller as simple as possible while still exposing the flexibility of the wireless board, the controller operates on commands sent from Raw. The Control section processes these commands, and sends back any data if required by the command. For example, The GETSTATUS command causes the Control section to send a 32-bit status word back to Raw (see Table B.1).

A command consists of two 32-bit words: an instruction word and a data word. Figure 5-2 shows an example command. The instruction word specifies what action the controller is to perform (see Tables B.1 - B.2 for a listing of the different commands and the actions they perform), and the data word supplies any data that might be required for the instruction. For example, the command ADCSER writes a 32-bit word into the control register on the ADC/DAC chip. The 32-bit word that is written is the one supplied in the data word. Even though a data word may not be required for some instructions (e.g. GETSTATUS), it still must be supplied; in these cases its value is simply ignored.

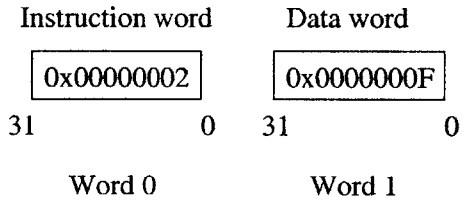


Figure 5-2: An example of a command. The first word sent (Word 0) specifies the instruction. In this case it is an RF_AGC instruction. The second word sent (Word 1) supplies any data required for the command. In the case of the RF_AGC instruction, the bottom 6 bits from the data word are loaded onto the RF AGC bus (described in Chapter 3)

5.1.2 Clock Domains

Figure 5-1 illustrates the four main clock domains in the controller. Each of the ADC and DAC sections operates in a separate clock domain. The clocks in each of these domains are synchronous to the 180MHz clocks provided by the ADC/DAC chip.

Note that there are two 180MHz clocks supplied, one for the ADC, and one for the DACs.

However, only a small portion of each of these sections operates using the supplied 180MHz clock, while the majority operates using the supplied clock divided by two (90MHz). Both the ADC and DAC sections convert the 180MSa/s datastreams to double-wide, 90MSa/s datastreams. The ADC section converts the 12-bit, 180MSa/s datastream to a 24-bit, 90MSa/s datastream. The DAC section operates on a 40-bit, 90MSa/s datastream, and converts it to a 20-bit, 180MSa/s datastream before sending it to the DACs.

This is done for two reasons:

1. Reducing the datarate makes writing the verilog code easier, since the timing constraints are easier to meet.

2. In the case of the ADC section, since the words written to Raw are always 32 bits wide, packing two samples into every word sent to Raw is a more efficient use of the available bandwidth.

The Control section operates in a different clock domain, using the Raw clock divided by 10. This is done because the Raw clock is the only active clock when the system is started. The ADC and DAC clocks are active only after the Controller explicitly turns on the ADC/DAC chip via its 3-wire serial control interface. Since the Control section is in a different clock domain from the ADC and DAC sections, asynchronous FIFOs (AFIFOs) are needed to communicate commands to and get status from these sections.

Lastly, the portion of the SpeedGasket which communicates with Raw operates in another clock domain using the Raw clock. The rest of the SpeedGasket operates using the clock of the section it is connected to. For example, the part of the SpeedGasket connected to the ADC section uses the ADC section clock.

5.2 SpeedGasket

The purpose of the SpeedGasket is to provide an asynchronous means of communicating data to and from the Raw chip. On the Raw side, the SpeedGasket uses the SIB protocol [7], and is clocked using the Raw clock. On the other side, a different clock can be used, although the SpeedGasket still provides the same SIB-style interface.

5.2.1 SpeedGasket Modifications

For this thesis, a 'From Raw' block of the SpeedGasket was modified to provide a 64-bit wide interface, instead of the 32-bit interface the SpeedGasket normally provides. This change was necessary in order to interface to the DAC section, which requires 40-bit samples. The primary difference between a modified SpeedGasket block and an unmodified one is that the original block uses a SIB interface, while the new one does not, although it uses the same input and output signals. The key differences are as follows:

1. The DataValid signal is no longer a single-cycle strobe. DataValid will remain high until the Data on the line is Yummied.

2. If there is valid data on the Data line (i.e. DataValid is high), no new data will be output until the current data is Yummied.

3. If the Yummy line is pulled high, and there is new data available to output, then new data will be put onto the Data line on the first clock cycle after the Yummy line is pulled high (i.e. 0 latency between Yummy being asserted and new data being output).

Figure 5-3 has a timing diagram illustrating the operation of this block, and a block diagram showing the directions of the different signals (CLK, DATA, VALID, YUMMY).

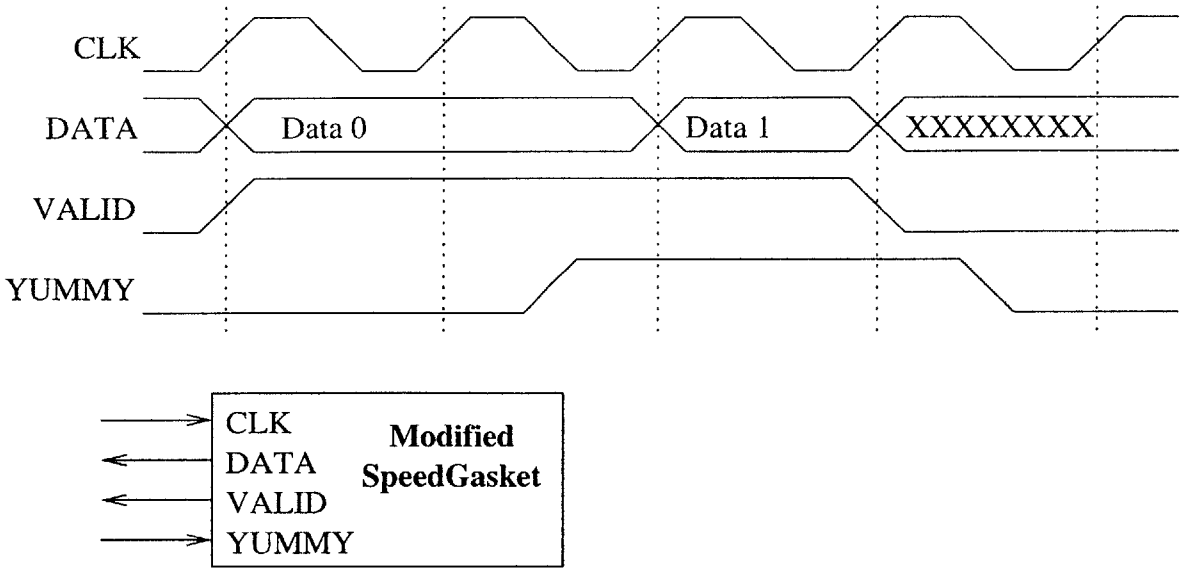


Figure 5-3: This figure shows a timing diagram illustrating the operation of the modified SpeedGasket block. Below the timing diagram is a block diagram showing the direction of the input and output signals. In the timing diagram it is assumed that there are only two pieces of data available: Data0 and Data1. Thus, after Data1 is Yummied, the output on the Data line is unknown. Vertical dotted lines delineate each clock cycle.

5.3 ADC Section

5.3.1 Overview

The ADC section takes the incoming data from the ADC (12-bit, 180MSa/s), converts it to a 24-bit, 90MSa/s stream, and then either stores the data in a RAM, or sends it directly to Raw. The ADC section can also read data out of the RAM and send that to Raw. It is important to note that the ADC only performs these actions based on commands from the Control finite state machine (FSM) (and ultimately, the Raw chip). The ADC section will wait for a command from the Control FSM (which in turn is caused by a command from Raw), and then process that command. Commands specify what action to take, and for how many 24-bit samples that command should be executed (e.g. send 52 24-bit samples to Raw, write 64 24-bit samples to RAM).

5.3.2 Design

The ADC consists of six major blocks: a 12-bit to 24-bit converter, a counter, a RAM, an FSM, a 32-bit 2-input mux, and a TO_RAW module. The 12-bit to 24-bit converter converts the 12-bit 180MSa/s datastream to a 24-bit, 90MSa/s datastream. The counter is used both to count the number of 24-bit words that have been processed, and to address the RAM. The RAM can be used to store data on the FPGA temporarily, in case the Raw chip is unable to keep up with the datastream from the ADC. The mux is used to select which data are sent to Raw: the output of the RAM, or the current data from the ADC. The TO_RAW module converts the output from the ADC section to the SIB protocol. Lastly, the FSM reads the data coming from the Control section and generates the correct control signals for all the other blocks. A schematic is shown in Figure 5-4.

The 24-bit output from the ADC section contains two 12-bit samples from the ADC. Figure 5-5 shows the breakdown of the 24-bit output. The earlier sample is stored in the most significant 12 bits (bits 12-23), while the later sample is stored in the least significant 12 bits (bits 0-11).

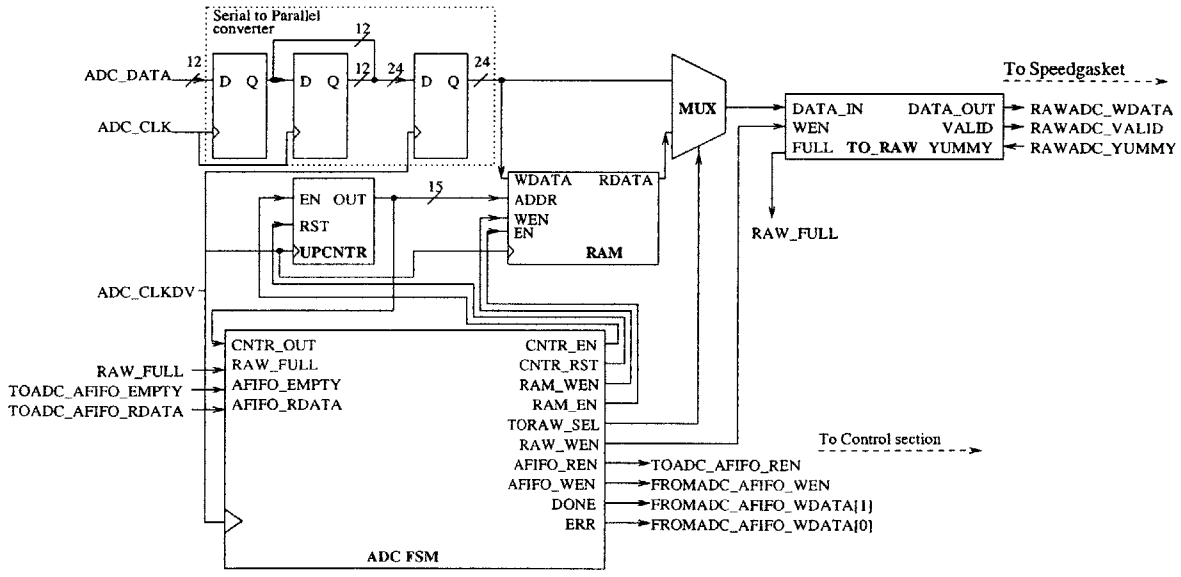


Figure 5-4: Diagram of the ADC section of the wireless board controller. ADC_CLK is the 180MHz sampling clock from the ADC/DAC chip. ADC_CLKDV is a 90MHz clock derived from ADC_CLK.

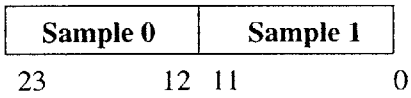


Figure 5-5: Diagram showing the breakdown of the 24-bit output from the ADC section. The 24-bit word contains two 12-bit ADC samples.

5.3.3 ADC Section FSM

The FSM in the ADC section supplies the control signals for the other blocks in the ADC section (RAM, Raw data output mux, address counter). The state transition diagram of the FSM is illustrated in Figure 5-6. Initially, the FSM waits until the TOADC_AFIFO has data in it. It then reads the data out and determines which command it has received (ADCDIRECT, ADCTORAM or ADCFROMRAM). Based on which command it has received, it then goes into a series of states suitable for that command. The series of states always terminates in the DONE state, which sends a done signal back to the control section, indicating that the ADC FSM has finished processing that command. The FSM then returns to the STALL state, where it waits until another command arrives.

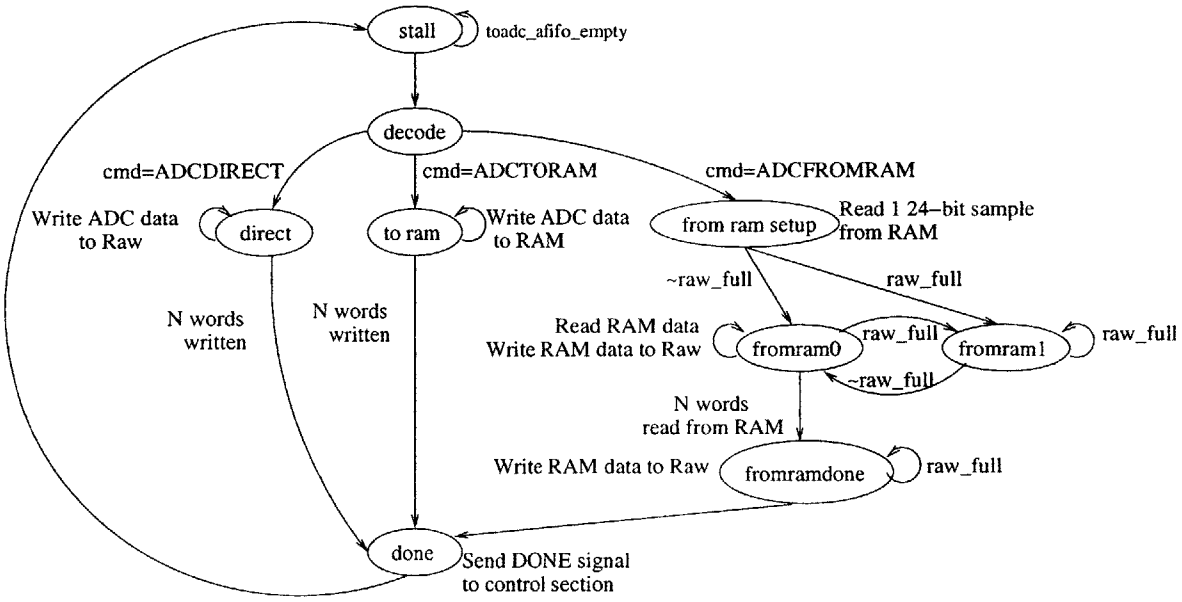


Figure 5-6: Diagram of the ADC section FSM.

5.4 DAC Section

5.4.1 Overview

The DAC section takes the incoming data from the SpeedGasket (40-bit, 90MSa/s), and either stores that data in a RAM, or sends it to a 'parallel to serial' converter which will convert the stream to a 20-bit 180MSa/s stream. The output of the converter is then sent to the DACs on the wireless board. Additionally, the DAC section can stream the contents of the RAM out to the parallel to serial converter. Similarly to the ADC section, the DAC section receives commands from the Control section, and processes these commands. When commands are not being processed, the output of the DAC section will set the DAC chip outputs to 0V.

5.4.2 Design

The DAC section consists of a 40-bit to 20-bit converter, a counter, a RAM, a flip-flop pipeline stage, a 40-bit 3-input mux, and an FSM. The 40-bit to 20-bit converter converts the 40-bit 90MSa/s datastream to a 20-bit 180MSa/s datastream to send to the DACs. Its input comes from the output of the mux. The mux selects between data from the RAM, data from Raw, or a constant value that sets the DAC outputs to 0V. The counter is used to count the number of 40-bit samples processed, and as an address input for the RAM. The RAM can be used to temporarily store data from Raw. The pipeline stage is in front of the mux, and was used to ease the timing constraints on this block. Finally, the FSM generates all the control signals for these blocks, and processes commands from the Control section. Figure 5-7 shows a schematic of the DAC section.

The 20-bit output of the DAC section consists of two 10-bit data streams (In-phase and Quadrature). The 40-bit datastream from the SpeedGasket contains two samples each of the In-phase and Quadrature data streams (see Figure 5-8).

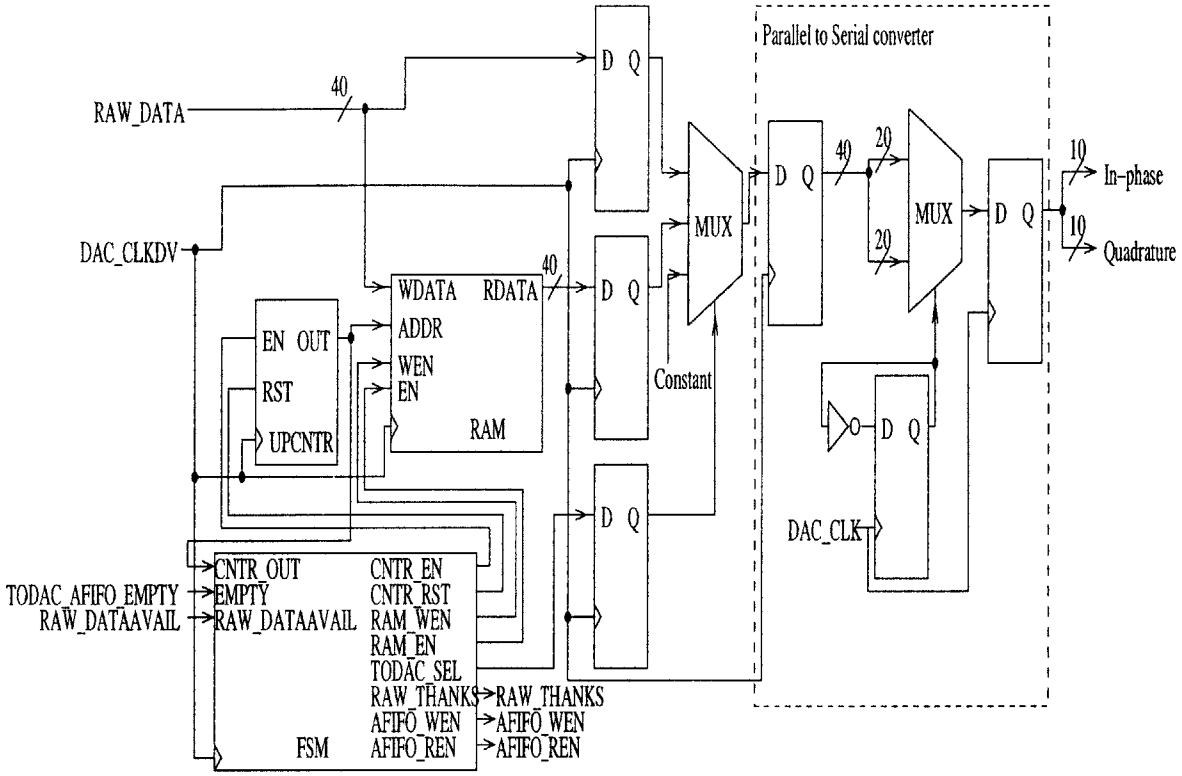


Figure 5-7: Diagram of the DAC block of the controller. DAC_CLK is the 180MHz sampling clock provided by the ADC/DAC chip. DAC_CLKDV is a 90MHz clock derived from DAC_CLK.

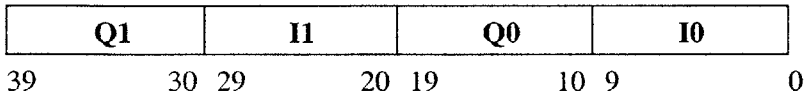


Figure 5-8: Diagram showing the breakdown of the 40-bit input to the DAC section. The 40-bit word contains two I and two Q samples.

5.4.3 DAC Section FSM

The FSM in the DAC section supplies the control signals for the other blocks (RAM, DAC data output mux, address counter). The state transition diagram of the FSM is shown in Figure 5-9. Initially, the FSM is idle, until the TODAC_AFIFO contains data. Once this occurs, the FSM reads the data from the AFIFO and determines whether the command is a DACDIRECT, DACTORAM or a DACFROMRAM command. It then enters a series of states appropriate for the particular command. All of the paths terminate in the DONE state, which causes a done signal to be sent back to the Control section via an AFIFO. The FSM then returns to the STALL state, where it waits until another command arrives.

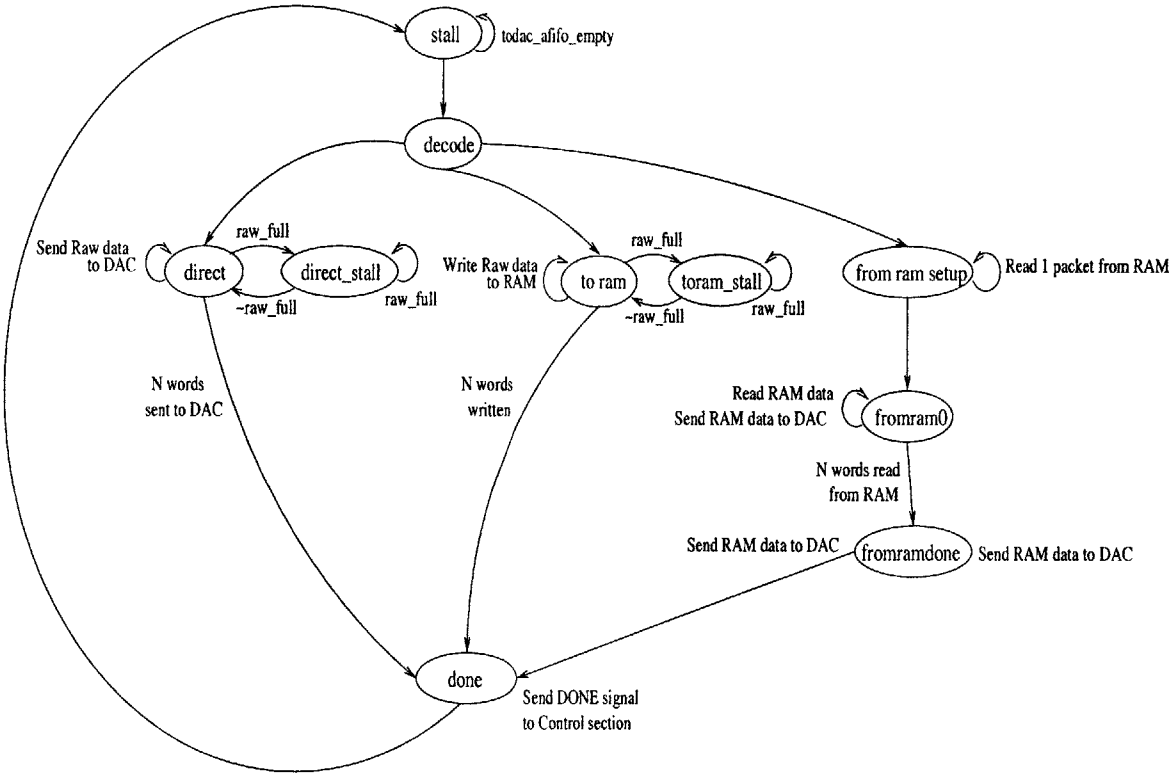


Figure 5-9: Diagram of the DAC section FSM.

5.5 Control Section

The Control section deals with the command and status interface to Raw (see Tables B.1 - B.2 for a command listing), the control of the ADC and DAC sections, and the control lines on the wireless board. Figure 5-10 shows a high-level block diagram of the Control section.

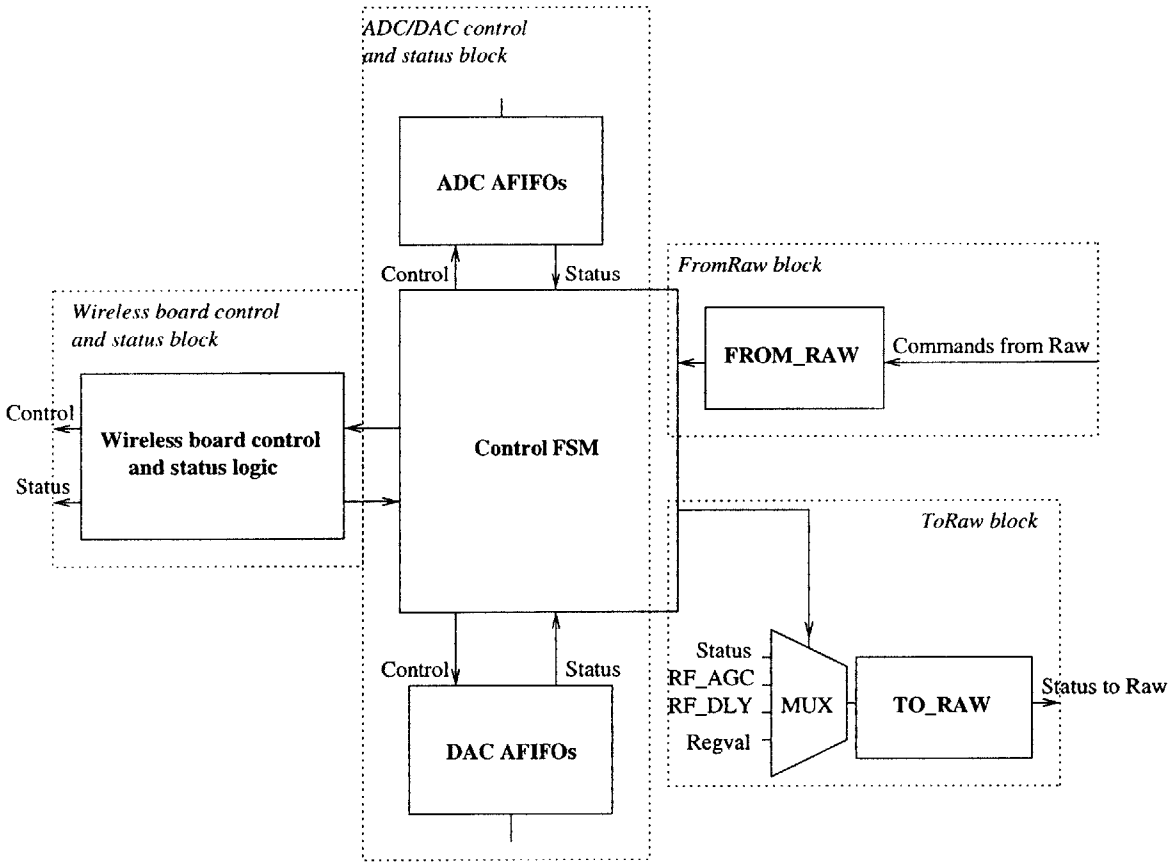


Figure 5-10: High-level block diagram of the Control section. Dotted lines show the breakdown of how the Control section is described in this thesis.

The different blocks of the Control section are described in the following subsections.

5.5.1 FromRaw Block

The FromRaw block (see Figure 5-11) shows the modules of the Control section involved in receiving commands from Raw. Commands are sent from Raw to the

FPGA, where they first go through the SpeedGasket (see Figure 5-1), and then to the FROM_RAW module in the Control section. The FROM_RAW module acts as a converter between the SIB protocol used by Raw and the SpeedGasket, and a FIFO-like interface, with ReadEnable, Empty, and Data signals. The Empty signal is sent to the Control FSM, where it is used to determine whether any command words are available.

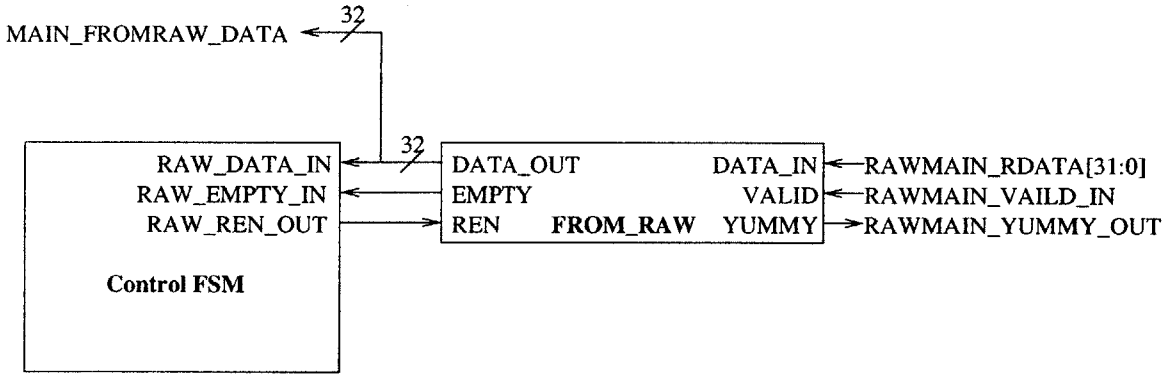


Figure 5-11: Figure showing the details of the FromRaw block in the Control section.

5.5.2 ADC/DAC Control and Status Block

The ADC/DAC control and status block encompasses those modules that deal with sending control signals to and receiving status signals from the ADC and DAC sections. This block consists of four AFIFOs, two each for the ADC and DAC sections; one for sending and one for receiving. The Control FSM takes care of the control and data signals for these four AFIFOs. The data output from the FROMADC and FROMDAC AFIFOs is sent to SR flip-flops, which hold the status of the ADC and DAC sections (adc_busy, adc_err, dac_err, dac_busy). The ERR_RST signal is used to reset the SR flip-flops holding the adc_err and dac_err bits. This signal is toggled whenever the GETSTATUS command is processed.

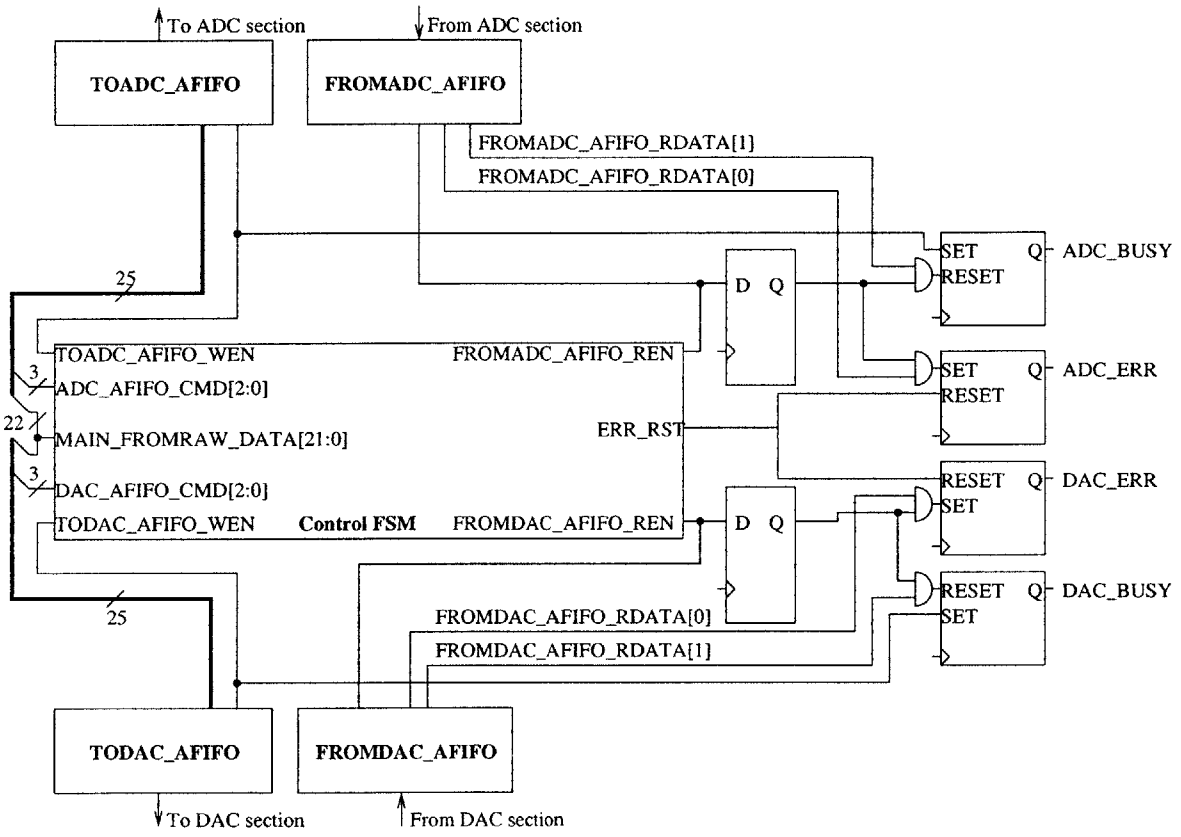


Figure 5-12: Figure showing the details of the ADC/DAC control and status block.

5.5.3 Wireless Board Control and Status Block

This subsection discusses the logic in the Control section dealing with the wireless board control and status signals. This block contains two serial modules, one each for the ADC/DAC and RF transceiver chips. These serial modules take in a 32-bit word and output that word according to the specification for the 3-wire serial control interface as defined in Section 3.2.2. This block also contains loadable D flip-flops to drive the RF_AGC and RF_DLY parallel buses, and input D flip-flops to synchronize the BOARD_RESET and RF_LOCK status signals. There is also a small RAM used to store the 32-bit values that are written to the different RF chip registers, and the ADC/DAC chip control register. The Control FSM handles the control of these modules.

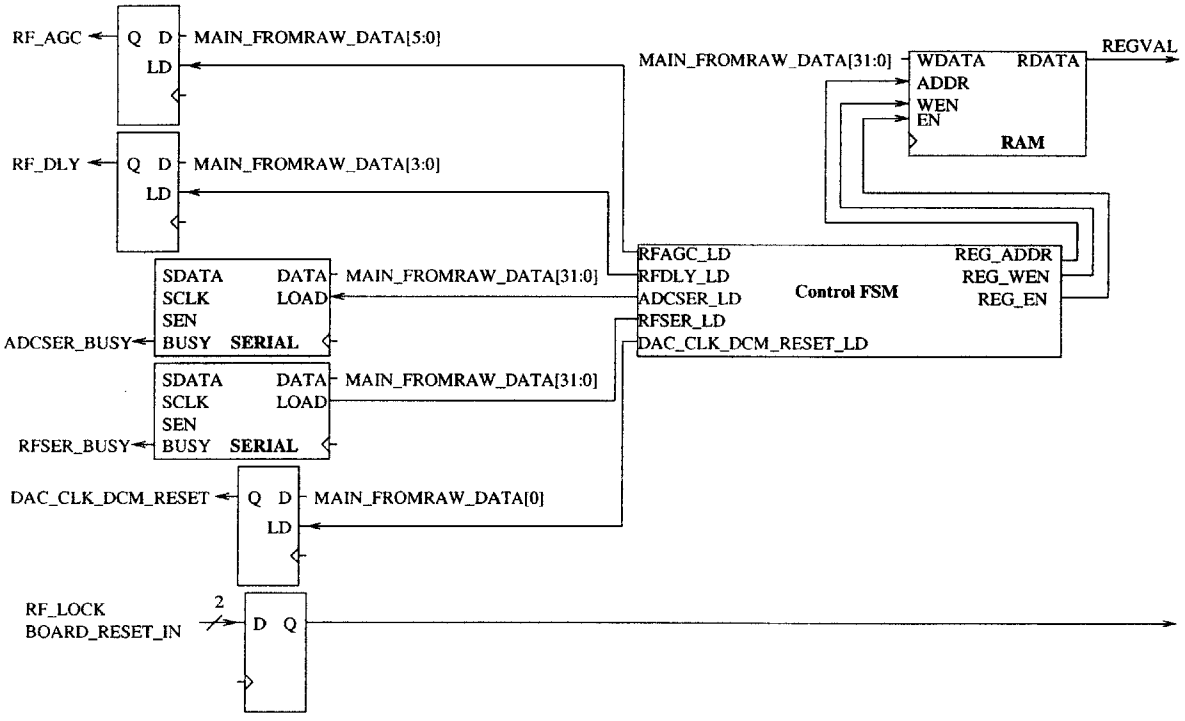


Figure 5-13: Figure showing the details of the wireless board control and status block.

5.5.4 ToRaw Block

The ToRaw block contains those modules involved in sending status to Raw. Specifically, it contains a TO_RAW module, which presents a FIFO-like interface at its

input, and outputs data according to the SIB protocol. It also has a mux which selects what data is to be sent to Raw. From Figure 5-14 it can be seen that not all of the inputs to this mux are 32 bits wide. The widths of the inputs are extended to 32 bits as necessary by adding 0s to the most significant bit. These two modules are controlled by the Control section FSM.

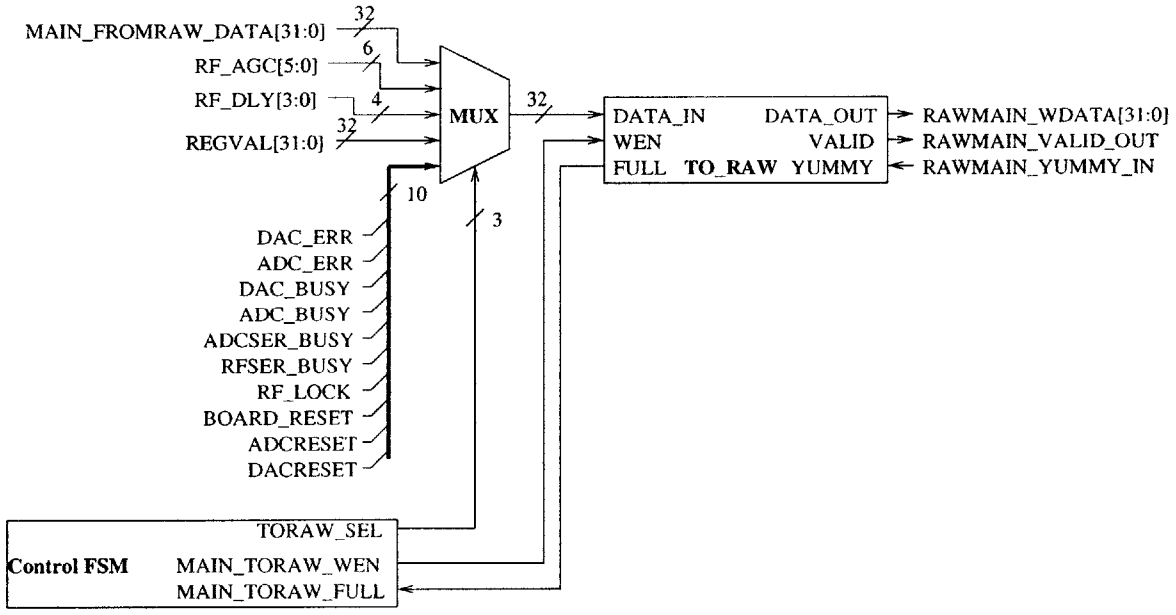


Figure 5-14: Figure showing the details of the ToRaw block.

5.5.5 Control Section FSM

The FSM in the control section is very simple, although it contains many different states. It waits in its 'stall' state until there is a command word waiting in the FIFO, indicated by the main_fromraw_empty signal going low. It then reads the word out, and moves into the decode state. The decode state determines what command was issued, and then either stays in the decode state (if the second word of the command is not yet available), or moves into the next appropriate state, which depends on what command word was sent. All of the states terminate by returning to the 'stall' state. Figure 5-15 shows the states involved with processing commands that deal with control of the wireless board or the ADC/DAC sections. Figure 5-16 shows those states involved with processing commands that are used to probe the status of the wireless board or the controller. Note that the 'stall' and 'decode' states are duplicated in these two figures for clarity.

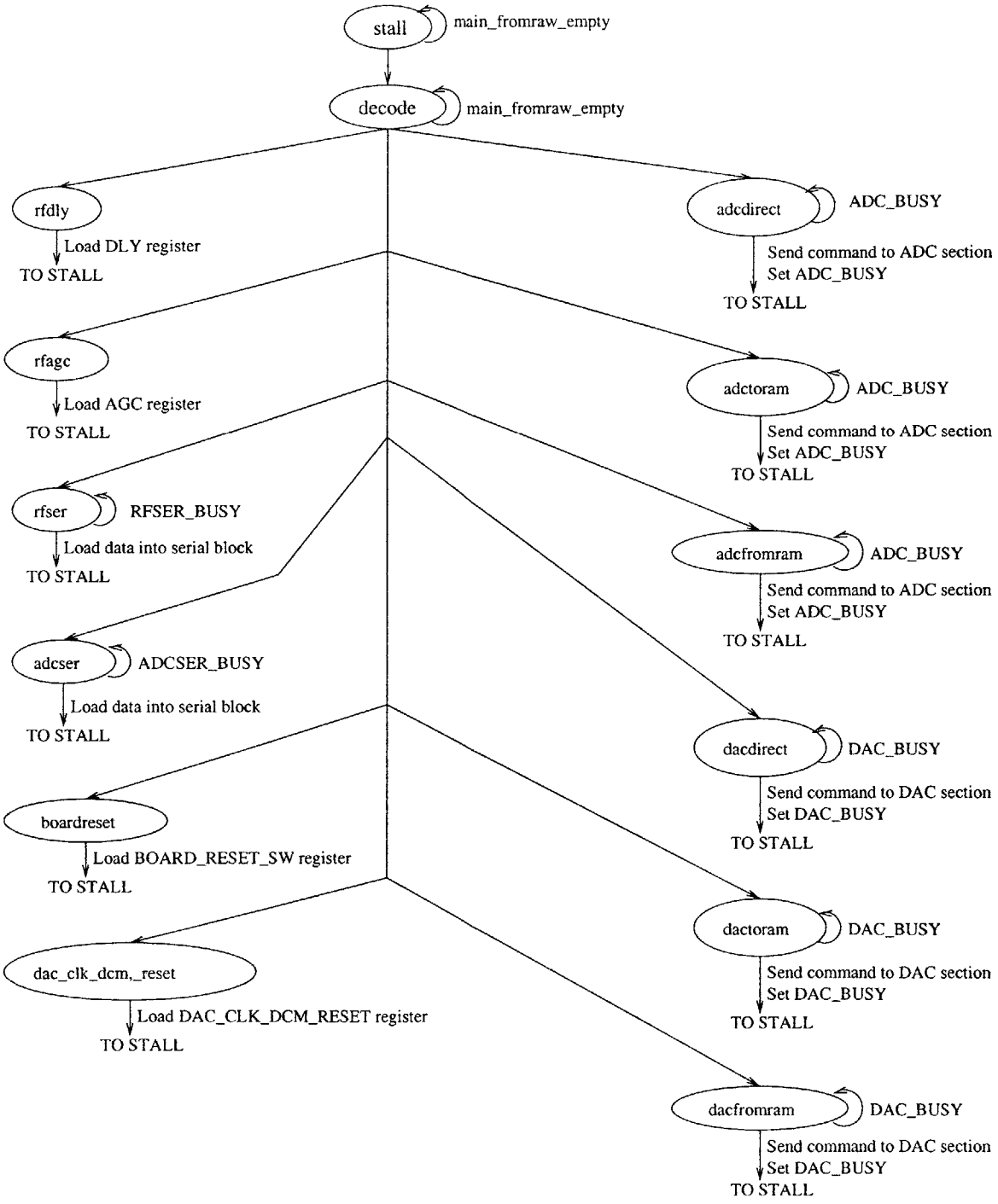


Figure 5-15: Diagram of the Control section FSM - those states involved with the wireless board and ADC/DAC section control.

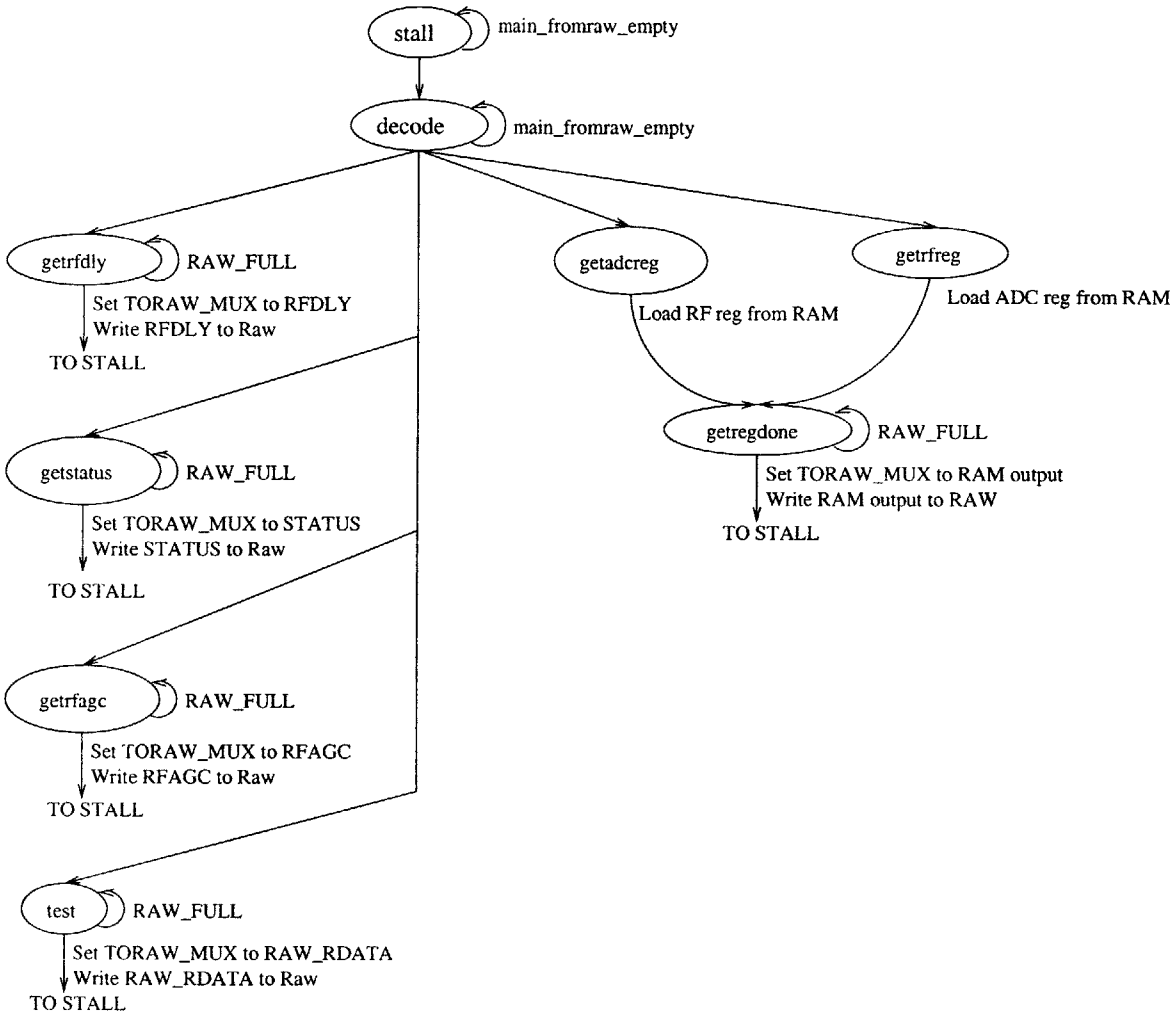


Figure 5-16: Diagram of the Control section FSM - those states involved with the status of the wireless board.

Chapter 6

Implementation and Testing

This chapter describes how the Raw wireless board and the wireless board controller were built, and what steps were taken to test them.

6.1 Implementation

6.1.1 Raw Wireless Board Interface

The Raw wireless board interface was built on a modified Engim, Inc. circuit board. Since Engim provided their schematic and layout files, the first step was to remove their digital processing chip from the schematic, and add the components and signals for the Raw wireless board interface. Once the schematic editing was complete, a design review was held and several changes were made to aid in testing and verification such as adding test points and grounding any unconnected digital inputs.

After the review, the changes were made in the schematic and a netlist was generated for board layout. The new circuitry was then placed and routed by hand and the board was sent to the manufacturer for a Design For Manufacturing (DFM) check. After correcting several DFM-related issues such as trace widths and clearances, 10 boards were fabricated, and two boards were assembled.

6.1.2 Wireless Board Controller

Since the manufacturing and assembly for the Raw wireless board took more than a month, the wireless board controller was designed and implemented during this time.

The wireless board controller was implemented in verilog and simluated using VCS. Initially, only the ADC and Control sections were built and integrated. The DAC section was implemented but not integrated until later in order to allow testing of the Raw wireless board to proceed. The wireless board was finished being manufactured before the DAC section had passed tests in simulation.

After the first version of the controller (including only the ADC and Control sections) was implemented and had been simulated, the SpeedGasket was added, and the design was synthesized. To meet timing, several iterations were needed in order to determine the optimal placement for the SpeedGasket blocks. Once the DAC section was complete, it was integrated and the full version of the controller synthesized. Again, a series of iterations was required to determine the optimal placement for the SpeedGasket blocks.

6.2 Testing

The following subsections describe the testing that was done on the Raw wireless board and the wireless board controller.

6.2.1 Raw Wireless Board

Once the Raw wireless board was received from the assembly house, a series of tests was performed to validate different sections of the board incrementally.

First, only the Raw wireless board interface section was powered to ensure that the added circuitry was safely connected. Next, a square-wave signal from a function generator was input to the 1.5V to 3.3V level shifters (to simulate an output from the ADC/DAC chip), and the output at the connector was probed to check its signal integrity.

The next test checked whether the ADC/DAC and RF chips could be powered up safely on their own. An FPGA configuration was created that output constant digital values onto the control pins of the wireless board. This ensured that the digital control inputs to the various chips on the board would not be floating, and that the chips would power up in the correct state. First, only the RF section was powered, and the different VCC pin voltages measured. Next, the entire board was powered up and the total board current draw measured.

After the board had been burned in, another FPGA configuration was created that included hardware to control the 3-wire serial interfaces for the RF and ADC/DAC chips. This configuration was created because at this point in the wireless board testing, the controller had not yet met timing in synthesis.

Initially, the ADC/DAC chip was programmed to keep both sections off, which is the state in which the ADC and DACs start up, in order to check that programming in the startup state caused no changes. It should be noted that this test only provides information if the test fails. If the test succeeds (as it did in our case), then no information is gained since a successful test results in no change.

Next, only the ADC was turned on, and the output clock and data lines probed. The clock and data signals were also probed at various points as they passed through the level-shifters and buffers. After verifying that the ADC section was activated, an LED on the Raw Handheld board was used to indicate whether the DCM in the FPGA was able to lock to the ADC sampling clock.

Once the DCM was able to lock, the connection between the RF chip and the input to the ADC was removed and a signal from a function generator was input to test the ADC. At this point the controller was complete, and was used to aid in further testing.

Raw was used to command the controller to grab a number of samples from the ADC chip. These samples were sent to a host computer for processing in Matlab. With Matlab, the time-domain signal could be analyzed, and frequency-domain data obtained. The ADC was tested with several different inputs; this is discussed in Chapter 7.

The next step was to test the DACs. The controller was used to turn the DACs on, and the output clock was probed, along with the PLL output. Similarly to the ADC clock, an LED on the Handheld board was used to determine whether the FPGA DCM locked to the DAC clock. Once the DCM locked, data was sent from Raw to the DAC section on the controller, and output to the DACs on the wireless board. The output from the DACs were probed and the oscilloscope data was recorded. This is also discussed in Chapter 7.

Last, an end-to-end test was performed. This involved turning on the ADC, the DACs, and the RF chip. Digital samples of a sine wave were generated and sent to the DRAMs on the Raw Handheld board. Software on Raw then streamed these samples to the DAC section of the FPGA. The DAC section then sent the samples to the DACs on the wireless board. The outputs from the DACs were fed into the transmit section of the RF chip, where they were used to modulate an RF carrier. The output from the RF chip was attenuated by 50dB and cabled to the receive section of the RF chip. The receive section then downconverted the RF carrier to baseband and output the baseband analog signal to the ADC input. The digital output from the ADC was sent to the ADC section of the controller, which streamed the digital samples into DRAMs on the Raw Handheld board. The samples in the DRAMs were then sent to a host computer for analysis in Matlab. The results from this test are discussed in Chapter 7.

6.2.2 Wireless Board Controller

Once the design was synthesized and met timing requirements, the configuration file for the FPGA was loaded onto a Raw Handheld board, and the controller was tested without the Raw wireless board connected.

Commands were sent from Raw to the controller, and the responses were checked to determine if the controller was working correctly. Specifically, the following tests were run:

1. The command TEST was sent, and the return value checked to make sure it

matched the data value sent with the TEST command.

2. The RF_AGC command was sent to set the RF_AGC bus, and then the GETRFAGC command was sent to check that the RF_AGC bus was set to the correct value.

3. The RF_DLY command was sent to set the RF_DLY bus, and then the GETRFDLY command was sent to check that the RF_DLY bus was set to the correct value.

4. The RFSER command was sent to program various registers on the RF chip. Then, the GETRFREG command was sent to check that the correct values had been loaded into the serial module. Also, the outputs from the RF serial module were also sent to a logic analyzer (LA) header on the Handheld board, where they were probed to ensure that the serial module correctly output the data.

5. The ADCSER command was sent to program the control register on the ADC/DAC chip. This module was tested in the same manner as the RF serial module above.

6. The ADC section and the Control/ADC section interface were tested by creating a version of the controller with an internal counter connected to the input to the ADC section. This counter provided a known input to the ADC section, which helped in testing. Then, ADCDIRECT, ADCTORAM, and ADCFROMRAM commands were sent from Raw to test whether the ADC section functioned correctly.

After these tests, the DAC section was completed and integrated into the controller. To test the DAC section, the two 10-bit outputs from the DAC section were connected to the input of the ADC section (with the most significant 4 bits from the DAC outputs being truncated, since the ADC input is only 12 bits). Data was sent from Raw to the DAC section, from the DAC section to the ADC section, and then back into Raw. The received data on Raw was then sent to the host and analyzed for correctness.

After testing the full controller on the FPGA, the Raw wireless board was connected to the Raw Handheld board, and the controller was used to facilitate further testing of the Raw wireless board.

6.3 Lessons Learned

This section details insights that were gained while implementing and testing the interface.

6.3.1 PCB Design

1. Constant-voltage inputs to chips (such as output enables, etc.) should not be directly connected to power or ground. Instead, they should be connected via a $10k\Omega$ resistor (or some other suitable value). This allows one to change the constant voltage at the pin if the need should arise, such as during testing. Also, in the schematic and layout, two resistors should be provided; one connected to power and the other to ground. Then, only one should be populated during assembly. This way, during testing, it will be easy to switch the constant voltage at that pin between power and ground.

2. Test points should be large enough to probe, and easy to reach/probe.

3. Many test points should be used, since it is far easier to put the time in during layout to do this, than it is to scrape away solder mask and probe thin traces. However, it is important to understand the effect test points may have on critical traces.

4. Power headers should be keyed. This will reduce the likelihood of applying reverse voltages.

5. Providing separate power headers for different sections can be useful for a couple of different reasons:

(a) Separate power headers can allow easy isolation of different sections of the board.

(b) The different headers allow easy measurement of the current/power draw of different sections.

6. Create places for ground posts or solder mask openings to allow for easy and nearby grounding for probes. Keep in mind the speed of the signals being measured, and make sure access to ground is close enough for proper measurement.

7. Make sure your design is checked over by a more senior engineer and your peers. There are also mailing lists (such as si-list) that engineers subscribe to, from which free advice can be obtained.

8. Always have a design review before proceeding to layout, and remember to have the reviewers provide tips on any layout issues that might come up as a result of the schematic-level design.

9. If possible, find out how large the panels are for your PCB manufacturing process. Take this into account when figuring out how large to make the board, if you're worried about cost vs. PCB size.

10. When having boards assembled, machine assembly is cheapest, but requires parts to be in cut-tape form, at a minimum of 8" in length.

11. Talk with the PCB manufacturing house and the board assembly house before beginning layout. At the least, you will need to find out design rules such as minimum trace width, trace clearances, and how much space must be left between parts to allow for machine assembly.

12. If doing machine assembly, either leave space on two edges of the board, or add breakaway rails so that a conveyor can move the board along without interfering with part placement.

13. Adding LEDs to power buses can provide a quick and easy way of knowing whether power is on or not.

6.3.2 FPGA Design

1. Check the timing specifications for the FPGA before attempting a large design in order to get a sense of how fast (roughly) designs can run, to get an idea of any pipelining you might require.

2. Figure out how many clocks will be needed, and try to use the minimum number of clocks necessary. If many clocks are needed, try to arrange the logic such that only a few major clock domains exist. Try to avoid having to communicate among multiple clock domains.

3. For testing purposes, if it is possible to run your design at a slower speed, then synthesize for the slower speed, since it will allow quicker turns on synthesis runs, allowing you to fix bugs faster. Only when it becomes necessary to run at full speed should you synthesize to that speed.

6.3.3 Board-level Testing

1. Using an active probe is helpful in ensuring that probing the signals does not affect the signals themselves significantly.

Chapter 7

Results and Future Work

This chapter discusses the current progress on the interface, and details three key results that have been obtained.

7.1 Overview

Thus far, the Raw wireless board has been fabricated, the ADC/DAC chip data and control paths have been functionally qualified, and an end to end test of the interface and RF front end has been performed. The wireless board controller has also been written and synthesized to the Virtex2 FPGA on the Raw Handheld board. The controller has been tested on its own and in conjunction with the Raw wireless board.

The most significant results to date have been:

1. The successful programming of the ADC/DAC chip to enable either the ADC, the DAC, or both.

2. The successful digitization and storage on Raw of an analog baseband signal.

3. The successful conversion (by the ADC/DAC chip) from digital to analog of a digital waveform stored on Raw.

4. A functionally successful end to end test involving the Controller, the ADC/DAC chip, and the RF chip.

The following four sections provide more in-depth information on the above results.

7.2 ADC/DAC Chip Control Path Qualification

In order to qualify the control path for the ADC/DAC chip, the following sub-goals were tested for and met:

1. The Raw wireless board controller on the FPGA is able to receive commands correctly from software running on Raw.

2. The Raw wireless board controller's serial modules function correctly.

To test the ADC/DAC chip control path, the path from Raw to the controller was tested to ensure that software on Raw could send commands to the controller. Next, the serial module in the controller was tested via commands from Raw to ensure that it met the specifications for the 3-wire serial control interface.

Finally, the full control path was tested using commands sent from Raw to the controller. The command to enable only the ADC section was sent, and the output clock of the ADC was probed using an oscilloscope. Since the ADC clock is active only when the ADC section of the ADC/DAC chip is enabled, the presence of the ADC clock indicated that the ADC/DAC chip was programmed correctly.

A similar test was performed for the DAC section, and the DAC clock was activated correctly as well.

7.3 ADC Data Path Functional Qualification

In order to test the ADC data path, several sub-goals were tested for and met:

1. The ADC data and clock signals are correctly converted to 3.3V, and sent to the FPGA pins on the Raw Handheld board.

2. The DCM on the FPGA is able to lock to the received ADC_CLK signal.

3. The skew between the data signals as they travel from the ADC to the FPGA is not too great to violate the setup/hold times on the FPGA input flip-flops.

4. The ADC section of the controller operates correctly.

5. The path from the ADC section to Raw operates correctly.

In order to perform this qualification, the path from the RF transceiver chip to the ADC/DAC chip was removed, and an analog signal from a function generator connected to the input of the ADC.

Initially, data received by the Raw chip had some spurious samples, but after some adjustment of the phase of the clock used on the FPGA, the ADC data was received correctly. See Figures 7-1 and 7-2 for Matlab plots of the digital samples received on Raw for different analog input signals. It is important to note that while these plots show that the output from the ADC chip looks functionally correct (i.e. a sinewave input produces a sinusoidal output at the correct frequency), no measurements were taken to ensure the ADC met the specifications given in the data sheet.

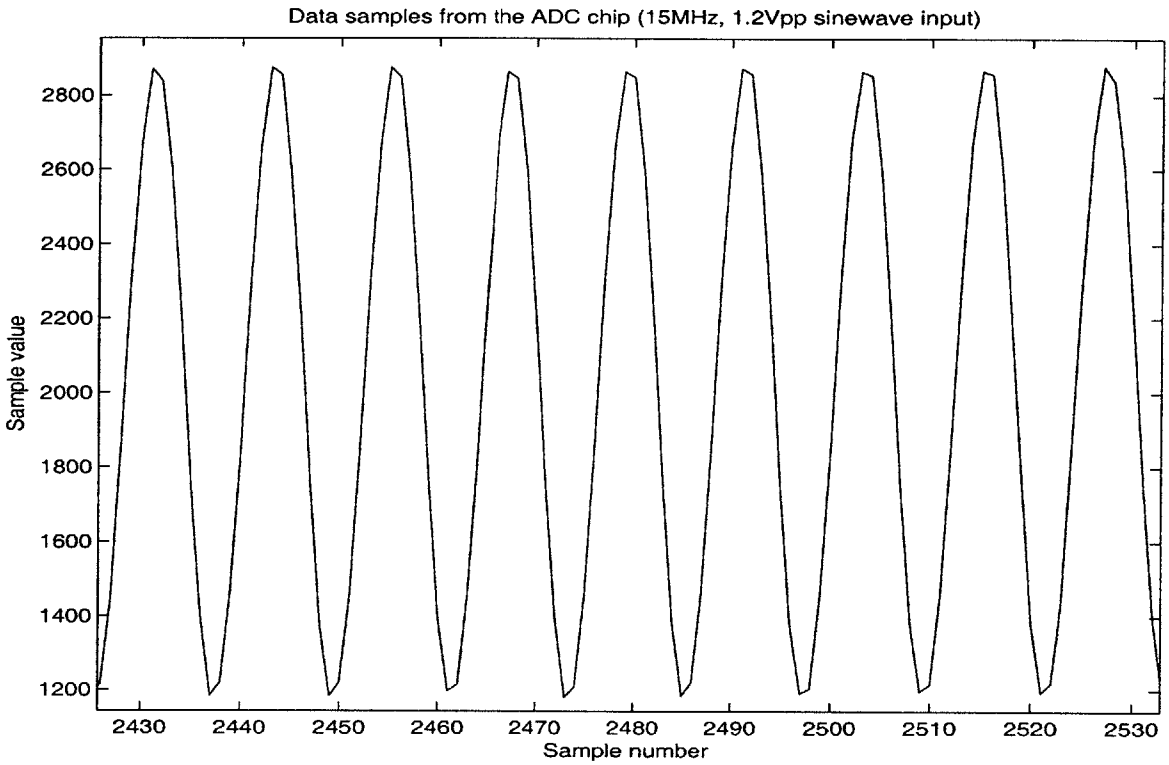


Figure 7-1: Matlab plot showing samples obtained from the ADC chip of a 1.2Vpp, 15MHz sine wave.

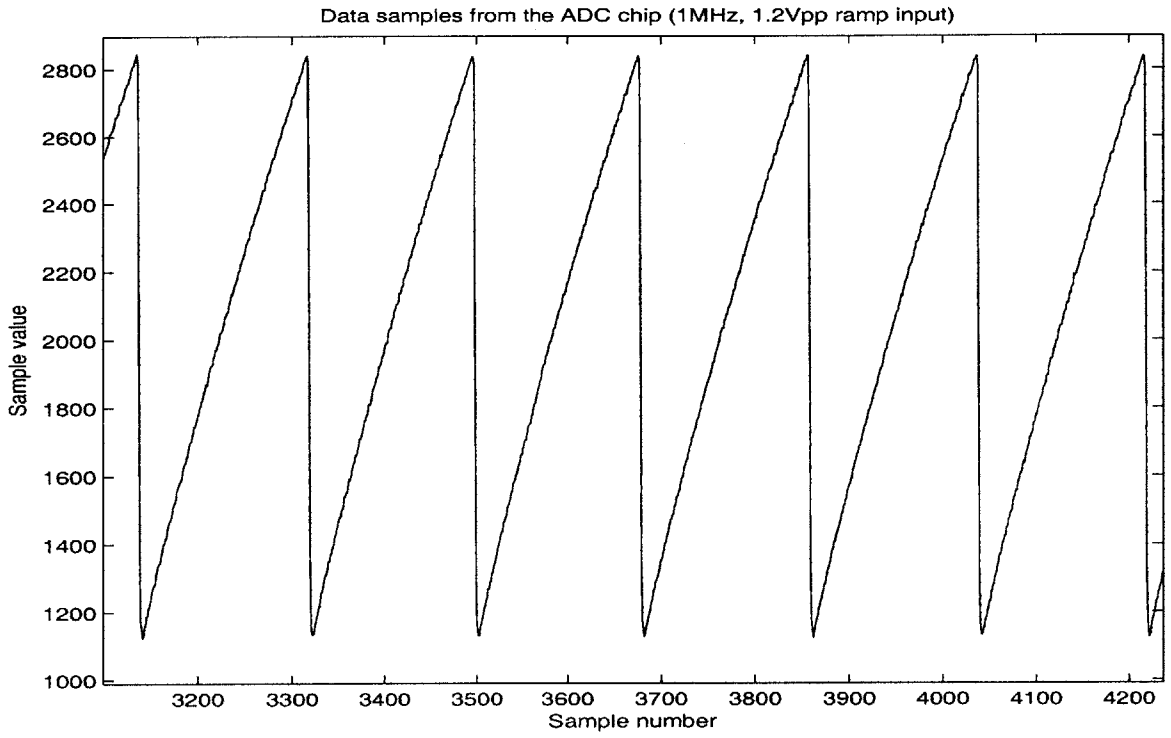


Figure 7-2: Matlab plot showing samples obtained from the ADC chip of a 1.2Vpp, 1MHz ramp wave.

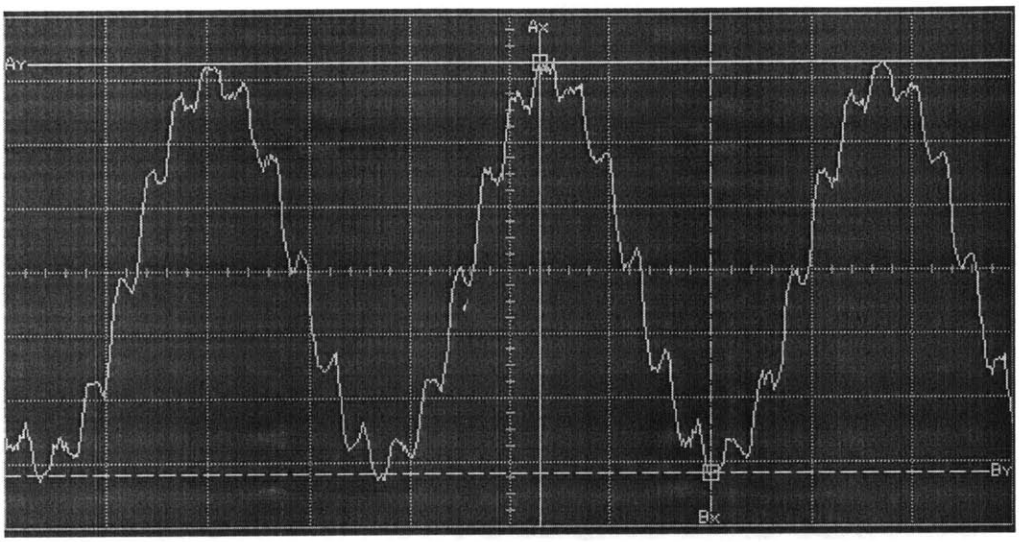
7.4 DAC Data Path Functional Qualification

In order to test the DAC data path, the following sub-goals were tested for and met:

1. The DAC clock signal is converted to 3.3V, sent to the PLL on the wireless board.
2. The DCM on the FPGA is able to lock to the received DAC_CLK_PLL signal from the PLL.
3. The PLL on the Raw wireless board locks correctly.
4. The path from Raw to the DAC section operates correctly.
5. The DAC section of the controller operates correctly.
6. The output signals from the DAC section of the controller arrive at the DAC at the correct time with respect to the DAC sampling clock.

To test the DAC data path, several different waveform samples were generated using Matlab and stored in the DRAMs on the Raw Handheld board. Software on Raw then read the samples from the DRAMs, and sent these samples to the DAC section of the controller. The DAC section of the controller then sent the samples to the DAC on the Raw wireless board. The outputs of the DAC were probed using an oscilloscope on the Raw wireless board. See Figures 7-3 - 7-5. It is important to note that while these figures show that the DAC output looks functionally correct (e.g. a sinewave input produces a roughly piecewise-constant sinusoidal output), the output was not measured to check that it met the specs given in the DAC data sheet.

Saved: 27 JUN 2004 16:47:51



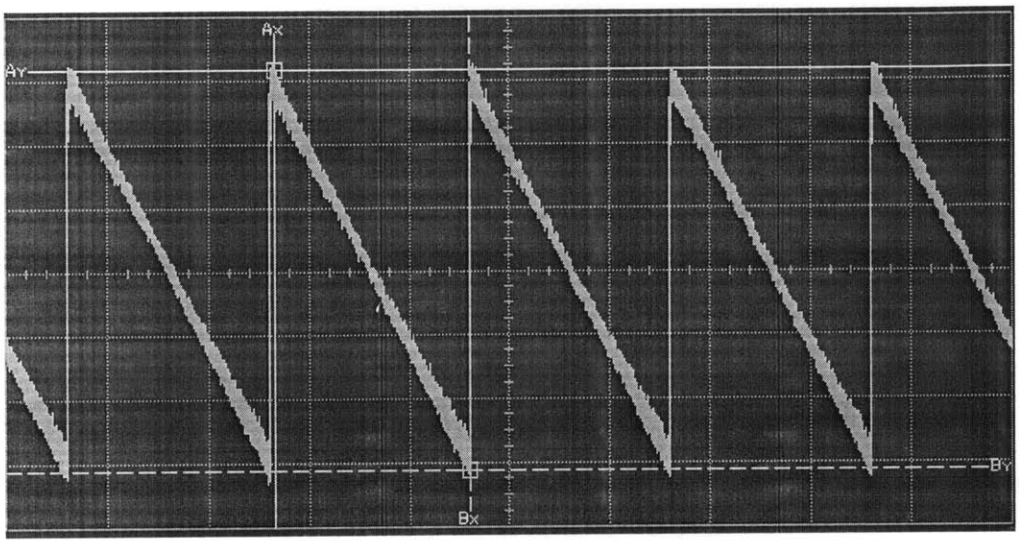
Acquisition	Sampling mode real time Normal Memory depth automatic 64000 pts Sampling rate automatic Sampling rate 4.00 GSa/s Averaging off Interpolation on
Time base	Scale 20.0 ns/ Position 6.1967833 µs Reference center
Trigger	Mode edge Sweep triggered Hysteresis normal Holdoff time 80 ns Coupling DC Source channel 1 Trigger level 1.282 V Slope rising
Memory 4	Vertical scale 200 mV/ Offset 0.0 V Horizontal scale 20.0 ns/ Position 205.16300000000 ns

Marker

	X	Y
A——(m4) =	211.163 ns	652.8 mV
B---(m4) =	244.981 ns	-633.7 mV
Δ =	33.818 ns	-1.2865 V
1/ΔX =	29.5699 MHz	

Figure 7-3: Oscilloscope screen capture showing the differential output waveform of one of the DACs of a full-scale, 15MHz sine wave.

Saved: 27 JUN 2004 16:49:15



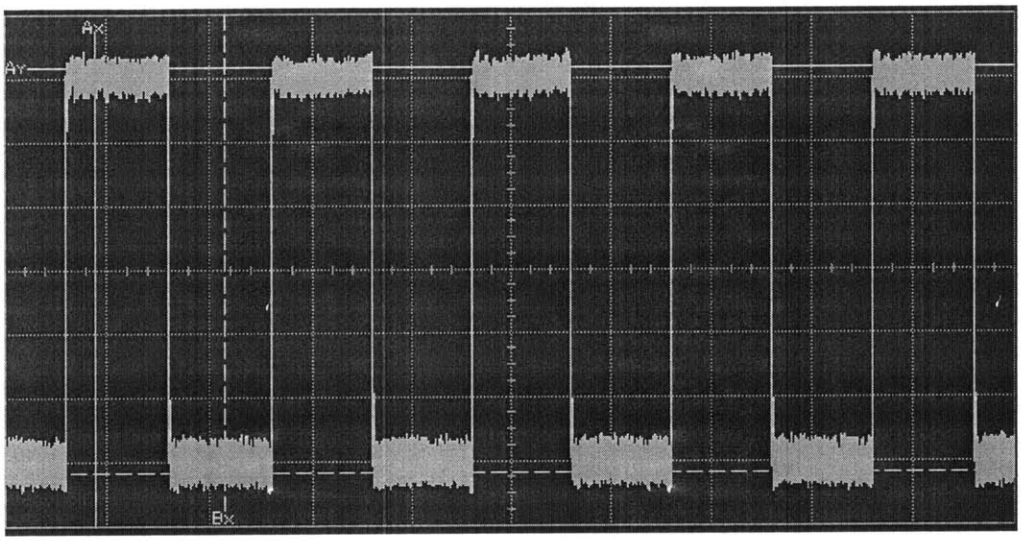
Acquisition	Sampling mode real time Normal Memory depth automatic 64000 pts Sampling rate automatic Sampling rate 4.00 GSa/s Averaging off Interpolation on
Time base	Scale 20.0 ns/ Position 6.1967833 µs Reference center
Trigger	Mode edge Sweep triggered Hysteresis normal Holdoff time 80 ns Coupling DC Source channel 1 Trigger level 1.282 V Slope rising
Memory 4	Vertical scale 200 mV/ Offset 0.0 V Horizontal scale 500 ns/ Position 6.1967800000000 µs

Marker

	X	Y
A——(m4) =	5.019516 µs	635.3 mV
B---(m4) =	5.992243 µs	-625.4 mV
Δ =	972.727 ns	-1.2606 V
1/ΔX =	1.028037 MHz	

Figure 7-4: Oscilloscope screen capture showing the differential output waveform of one of the DACs of a full-scale, 1MHz ramp wave.

Saved: 27 JUN 2004 16:46:19



Acquisition	Sampling mode real time Normal
	Memory depth automatic 64000 pts
	Sampling rate automatic Sampling rate 4.00 GSa/s
	Averaging off Interpolation on
Time base	Scale 500 ns/ Position 6.1967833 µs Reference center
Trigger	Mode edge Sweep triggered
	Hysteresis normal Holdoff time 80 ns Coupling DC
	Source channel 1 Trigger level 1.282 V Slope rising
Memory 4	Vertical scale 200 mV/ Offset 0.0 V
	Horizontal scale 500 ns/ Position 6.1967800000000 µs

Marker

	X	Y
A——(m4) =	4.146788 µs	634.0 mV
B---(m4) =	4.769517 µs	-631.2 mV
Δ =	622.729 ns	-1.2653 V
1/ΔX =	1.605835 MHz	

Figure 7-5: Oscilloscope screen capture showing the differential output waveform of one of the DACs of a full-scale, 1MHz square wave.

7.5 Functional End to End Test

As a final step in qualifying the interface, an end to end test was performed that involved all of the circuitry on the Raw wireless board and the wireless board controller.

Digital samples of a $5MHz$ sine wave were generated for the in-phase data, and a constant value corresponding to a 0V output from the DAC was generated for the quadrature data. These digital samples were sent to the DRAMs on the Raw Handheld board from the host computer. Software on Raw then streamed these digital samples to the DAC section in the wireless board controller. The DAC section then streamed the I and Q data to the DACs on the wireless board. The output from these DACs were fed into the transmit section of the RF chip, where they were used to modulate an RF carrier ($2448MHz$). The RF output from the board was then attenuated by 50dB and sent via cable to the receive section of the RF chip. The receive section downconverted the received signal using a LO frequency of $2396MHz$. The analog baseband output from the RF chip was then fed into the ADC and digitized. The digital samples from the ADC were then captured by the controller, and sent to the DRAMs on the Raw Handheld board. Software on Raw then streamed these samples to a host computer for processing in Matlab.

Figure 7-6 shows a plot of an FFT of the digital samples from the ADC chip. The magnitude (in dB) of the signal is plotted on the Y axis, and the frequency (in Hz) on the X axis. Note that the magnitudes have been normalized so that 0dB is the largest magnitude.

The two highest peaks in the figure are at $47MHz$ and $57MHz$. These correspond to the $5MHz$ sine wave that was used to modulate the transmitted RF signal. Initially the $5MHz$ wave modulated a $2448MHz$ carrier producing two peaks, one at $2448MHz - 5MHz = 2443MHz$, and one at $2448MHz + 5MHz = 2453MHz$. This spectrum was then downconverted using a $2396MHz$ LO, which moved the two peaks to $2443MHz - 2396MHz = 47MHz$, and $2453MHz - 2396MHz = 57MHz$. The third largest peak, at $52MHz$, is the downconverted transmit LO frequncy

($2448MHz - 2396MHz = 52MHz$). Ideally, the only peaks in the plot should be at $47MHz$ and $57MHz$. Despite the fact that there are other peaks in the plot (such as the one at $52MHz$), the fact that the two at $47MHz$ and $57MHz$ are the strongest demonstrates that the system works on a functional level. Further debugging and testing will be required to ensure that the system can be used to transmit 802.11b/g packets.

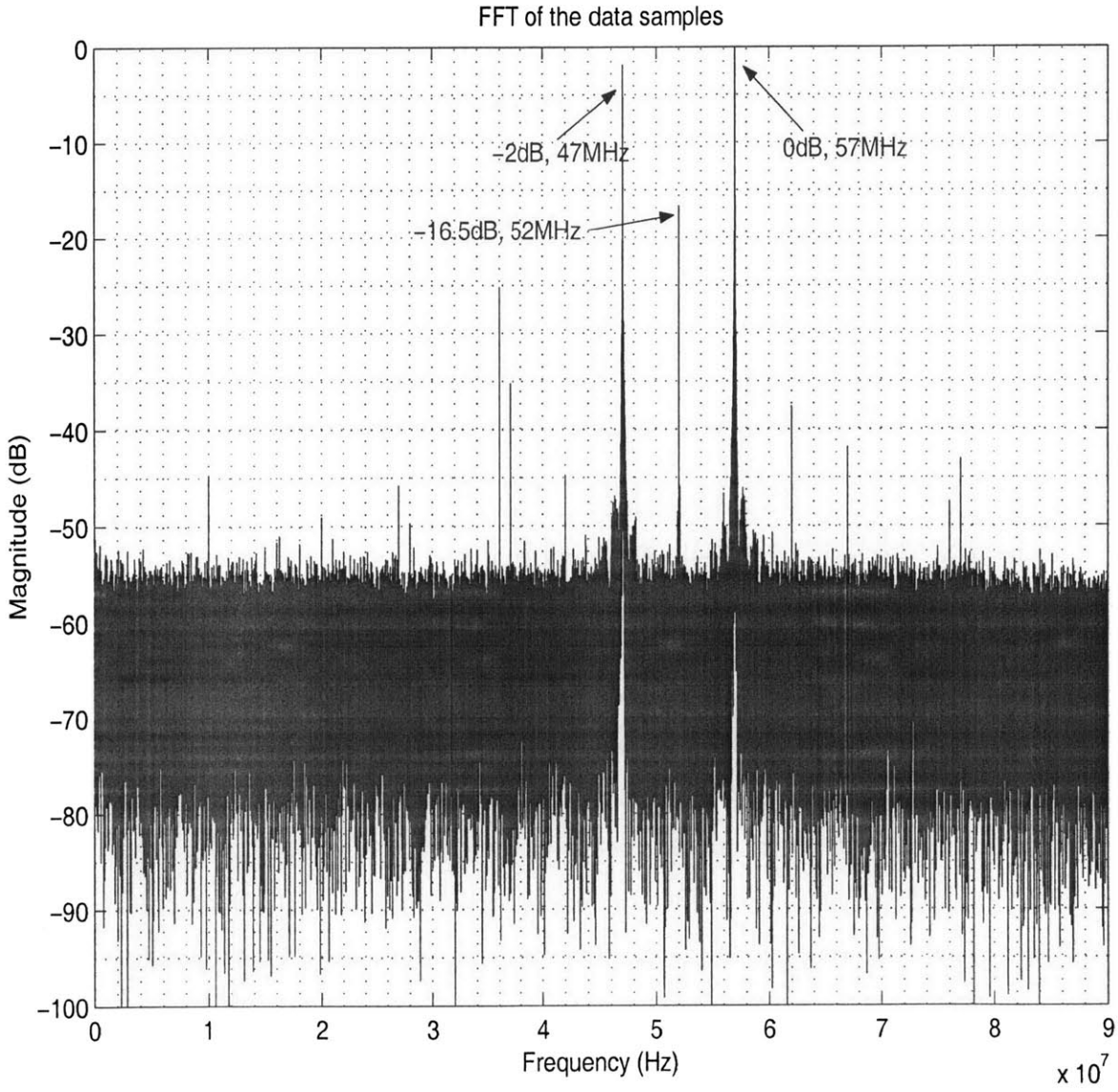


Figure 7-6: Matlab plot of an FFT of the digital samples from the ADC chip in the end to end test. A 5MHz sine wave was transmitted with a transmit LO of 2448MHz. The receive LO was set to 2396MHz.

7.6 Future Work

The future work for the interface is broken down into two sections: work that needs to be done to integrate the interface into the Raw wireless demonstration system, and improvements to the interface.

7.6.1 Integration with Raw Wireless demonstration system

As mentioned in Chapter 1, the interface will be used to create a system that demonstrates Raw performing the digital baseband processing for the 802.11g protocol. Currently, the interface works correctly, but there is more work to be done to use it in the demonstration system. Two key goals must be accomplished: The Raw wireless board needs to be qualified for sending 802.11g packets, and the interface needs to be integrated with the software processing block (see Chapter 3, Figure 3-1).

Currently, the Raw wireless board has been provisionally qualified, and an end to end test has been performed. While the end to end test showed that the RF front end and the interface work on a functional level, more testing needs to be done to ensure that the system is capable of transmitting and receiving 802.11g packets.

In terms of integration with the software processing block, some software for Raw needs to be written to send commands to the wireless board controller, and the data to and from the controller needs to be processed and formatted correctly for the software processing section. Specifically, the ADC data from the controller consists of three 802.11g channels, but the software processing section will handle only one channel at a time. So, filtering and decimation needs to be performed in order to both eliminate unwanted channels and to reduce the datarate. In addition, the software processing block assumes that the data is already split into in-phase and quadrature samples, and software will be needed to convert the data from the controller to this format.

7.6.2 Wireless Board Controller Improvements

The Raw wireless board controller currently works and is qualified, but there are some design changes that could be made and features added that would improve it.

1. Minimize the number of major clock domains. It would be advantageous to change the design such that there is one clock domain containing most of the logic, with other small clock domains added as necessary.

   Instead of having the three major clock domains as defined by the Control, ADC and DAC sections, create a single clock domain that has the logic for the Control, ADC and DAC sections in it. Since the ADC and DAC data is synchronous to different clocks, AFIFOs will be needed to bring the ADC data in, and send the DAC data out. See Figure 7-7 for a block diagram of the new design. From the figure, it can be seen that the number of overall clock domains remains the same. However, one of the clock domains contains most of the logic. This is advantageous in terms of clock distribution. Since FPGAs have a limited amount of clock distribution resources, it is advantageous to reduce the area over which a clock is routed. In terms of the controller, the large RAMs (implemented as BLOCKRAMs in the FPGA) located in the ADC and DAC sections means that the clocks associated with these sections need to travel everywhere within the FPGA. If these two sections shared a clock with the Control section, then routing the clock is easier, because only one clock would need to be routed throughout the FPGA.

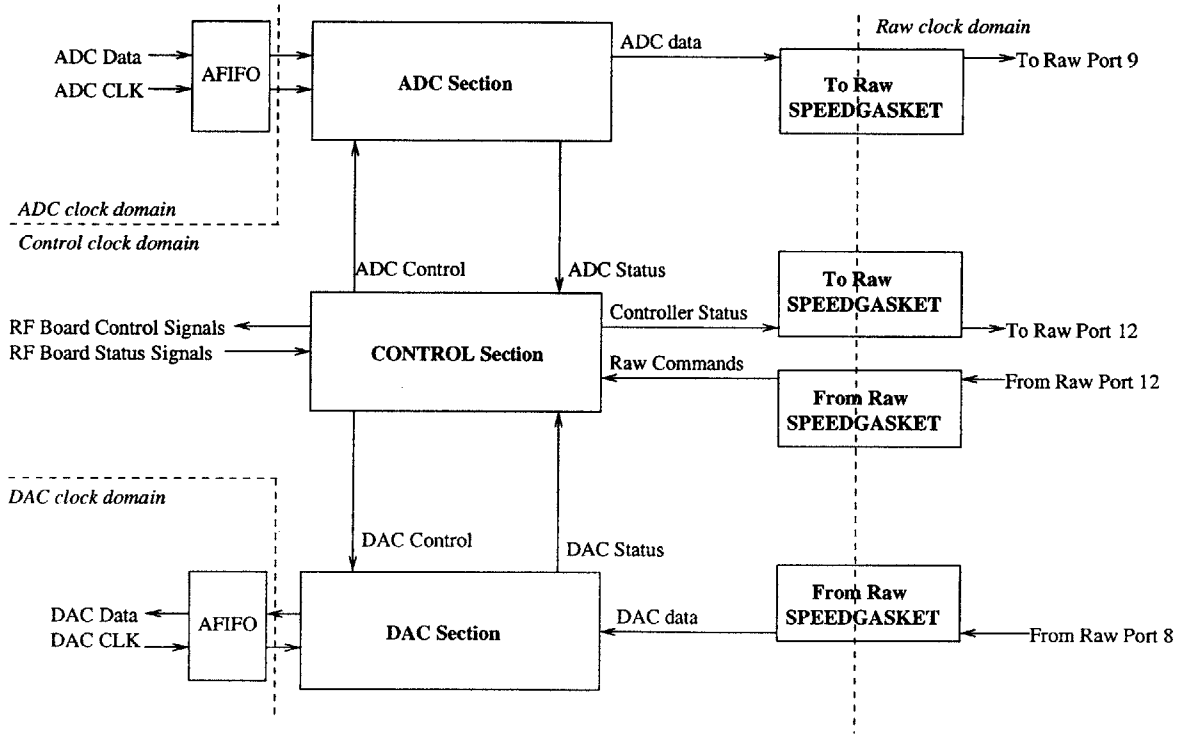


Figure 7-7: Figure showing the clock domains for a design that minimizes the size of the ADC and DAC clock domains. The dashed lines show the boundaries between different clock domains.

2. Change the network onto which the ADC section sends data, and from which the DAC section receives data.

   If the ADC section sent data over the dynamic network instead of the static network, then data could be sent directly to the DRAMs on the Raw Handheld board. Currently, data is sent to Raw via the static network, where a tile receives the data, and then transfers the data to the dynamic network, which sends the data to the DRAMs. This means that an entire tile on the Raw chip is used solely for transferring data from the static network to the dynamic network. If the ADC section sent the data on the dynamic network to begin with, this tile could be used instead for useful computation.

   The disadvantage of using the dynamic network compared to the static network is that it requires data to be sent in packets, with a header word sent first, identifying the length of the packet and where it should be routed. In addition, when using the dynamic network to send data to memory, another word specifying the address must also be sent, and packets are limited to 8 32-bit words.

   Since the ADC section is currently clocked at exactly the same rate as the incoming data, it would be impossible for the ADC section to also send out these additional header words. However, if the ADC section were clocked at a higher speed, say $112.5MHz$, the ADC section could use the extra time to send out the additional header words. Data still comes in at a rate of 90MSa/s, but the new output rate is 112.5MSa/s. This means that for every 8 samples of data, there are two extra slots that could be used to send out the header words.

   For the DAC section, no clock speed modification would be required, since only two words are needed to *request* data from the DRAMs. Thus, a DAC section running at 90MHz could actually request data more rapidly (1 request every 2 cycles = 8 data words every 2 cycles = 360MSa/s) than it could receive it (2 data words every cycle = 180MSa/s).

Chapter 8

Conclusion

8.1 Conclusion

This thesis described the design of an interface between the Raw microprocessor and an 802.11b/g RF front end. The interface was created to enable a system using the Raw chip as the digital processing element in a wireless system. This system can be used both as a demonstration of Raw's capabilities and as a tool for wireless systems research.

Specifically, the overall demonstration system was described to provide context in which to understand the design of the interface. Next, the designs for the two major blocks of the interface were described, along with a chapter describing their implementation and testing. Four of the key results that have been obtained so far were presented: the qualification of the ADC/DAC control path, the functional qualification of the ADC and DAC data paths, and a functionally successful end to end test of the RF front end and interface. Last, future work was discussed, describing the remaining steps to realizing the Raw wireless demonstration system, and describing some improvements that could be made to the interface.

Appendix A

Wireless Board Reference

This appendix is designed as a reference guide for those using the Raw wireless board. It presents a more detailed schematic of the board, along with a description of the signals on the board, and usage guidelines.

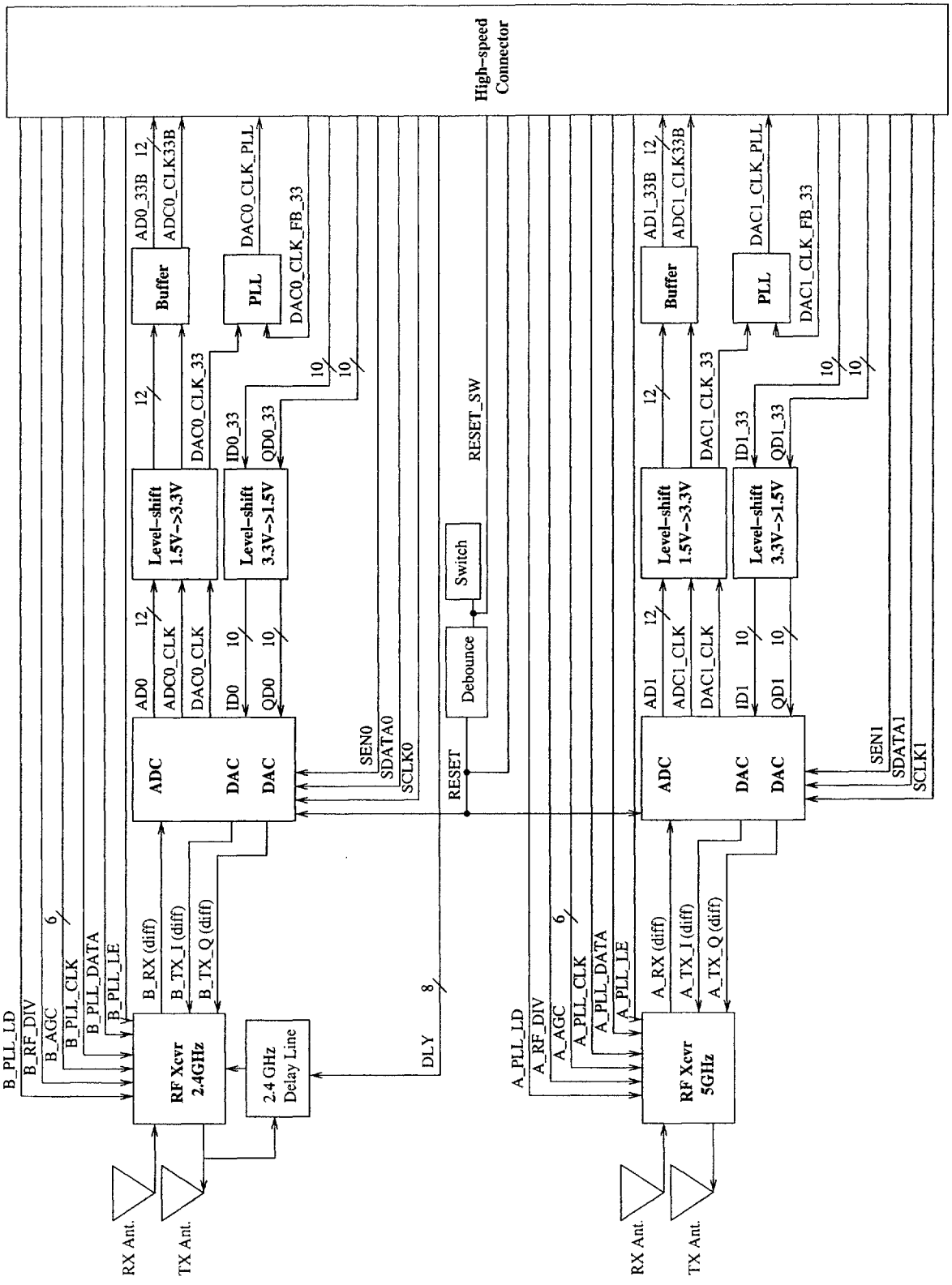


Figure A-1: A chip-level diagram of the Raw wireless board. This shows only the major chips on the board (e.g. those involved with data and control signals). The signal names match those used in the schematic.

A.1 Wireless Board Signals

The following table lists the important signals on the wireless board. It gives the name of the signal as used in the schematic/layout tools, and as it is used in the FPGA code (in a few cases the names are slightly different). It also provides a brief description of the signal.

Table A.1: Wireless board signals

Signal Name Schematic/Layout	Signal Name FPGA Code	Description
DAC0_CLK_PLL	DAC0_CLK_PLL	The output from the DAC_CLK PLL.
DAC0_CLK_FB_33	DAC0_CLK_FB_33	The feedback DAC_CLK signal from the FPGA.
ID0_33	ID0_33	10-bit In-Phase digital DAC data.
QD0_33	QD0_33	10-bit Quadrature digital DAC data.
ADC0_CLK33B	ADC0_CLK33B	The ADC_CLK signal output from the 3.3V buffer chip.
AD0_33B	AD0_33B	12-bit digital ADC data.
DLY_A1 DLY_A2 DLY_B1 DLY_B2 DLY_C1 DLY_C2 DLY_D1 DLY_D2	DLY_A1 DLY_A2 DLY_B1 DLY_B2 DLY_C1 DLY_C2 DLY_D1 DLY_D2	2.4GHz delay line settings (8-bit). Bits A2, B2, C2, D2 are inverses of A1, B1, C1, D1, respectively. Only 16 states are possible. A1 is the least significant bit and 0000 is the minimum delay.
B_RF_DIV	RF0_DIV	RF Diversity selection. This line controls whether the RF chip uses the diversity antenna port or not.
B_AGC_A5 B_AGC_A4 B_AGC_A3 B_AGC_A2 B_AGC_A1 B_AGC_A0	RF0_AGC[5] RF0_AGC[4] RF0_AGC[3] RF0_AGC[2] RF0_AGC[1] RF0_AGC[0]	6-bit RF AGC bus. These 6 bits specify the cascaded LNA and IF chain gain. See the RF chip documentation for more info, as the control of these is somewhat nonintuitive.

Table A.2: Wireless board signals

Signal Name Schematic/Layout	Signal Name FPGA Code	Description
B_PLL_DATA	RF0_SDATA	The data line of the 3-wire serial interface for the 2.4GHz RF transceiver chip.
B_PLL_CLK	RF0_SCLK	The clock line of the 3-wire serial interface for the 2.4GHz RF transceiver chip.
B_PLL_LE	RF0_SEN_N	Active low. The enable line of the 3-wire serial interface for the 2.4GHz RF transceiver chip.
RF_LOCK0	RF0_LOCK	Status signal from the 2.4GHz RF chip indicating whether either the RX or TX PLLs are locked. Bits 27, 28 of the SYNTH/MISC register in the RF chip control to which PLL this signal corresponds.
B_CP_PD	RF0_CP_PD_N	Active low. Enables power for the 2.4GHz RF transceiver RX and TX charge pumps. Not useful if Q101 and Q102 are not populated.
B_PA_PD	RF0_PA_PD_N	Active low. Enables power for the 2.4GHz RF power amplifier.
SDATA0	ADC0_SDATA	The data line of the 3-wire serial interface for the ADC/DAC chip.
SCLK0	ADC0_SCLK	The clock line of the 3-wire serial interface for the ADC/DAC chip.
SEN0	ADC0_SEN_N	Active low. The enable line of the 3-wire serial interface for the ADC/DAC chip.
RESET	BOARD_RESET_N	The Reset signal from the wireless board. The Reset signal is an input to the ADC/DAC chip.
RESET_SW	BOARD_RESET_SW_N	The software reset signal. Pull this signal low to cause Reset to go low as well. This is connected to a pullup resistor, so it should be tristated to allow Reset to go high. This is currently unusable because resistor R300 and R302 are unpopulated.

A.2 Usage Guidelines

This section presents several useful points about how to use the wireless board.

1. When working with the board, always use an anti-static wrist strap that is properly connected to ground. The circuitry on the board, in particular the RF circuitry, can be very sensitive to static discharge.

2. Never power up the board unless its digital control and data lines are being driven. This is important to prevent floating input voltages from causing large amounts of shoot-through current in CMOS circuits. Also, if the digital lines are not driven during powerup, the device could be programmed into a harmful state by the floating voltages at its control inputs.

3. When probing high-speed digital lines, remember that the probe will add capacitance, and thus the total board current (i.e. power dissipation) will increase.

4. Use the hold-downs on the 190-pin high-speed connector in order to ensure proper seating of the connector. Even though the connector may seem to be seated correctly, some pins may not have a good connection.

5. The ADC/DAC chip requires an external fan to cool it (along with a heatsink).

Appendix B

Wireless Board Controller Reference

This appendix is designed as a reference guide for those using the wireless board controller. It provides detailed information about the commands available, along with a listing of caveats that could help a new user.

B.1 Controller Commands

As mentioned in the thesis, commands from Raw consist of two 32-bit words, sent one after the other. The first word is the instruction word and specifies what task the controller is to perform. The second word is a data word, and supplies any data that the command might require (e.g. The ADCSER command requires a 32-bit word to send on the ADC/DAC serial lines). Tables B.1 and B.2 provide a listing of all the commands, along with their encodings and descriptions. In the table the data word is referred to as Data. For example, Data[2:0] specifies a 3-bit word consisting of Data bits 0(LSB), 1, 2(MSB).

Table B.1: Wireless board controller commands.

Instruction (encoding)	Uses Data	Description
RFSER (0x00000000)	Y	Sends Data[31:0] to the RF chip over the 3-wire serial interface. Writes Data[31..0] to the Control section RAM.
ADCSER (0x00000001)	Y	Sends Data[31:0] to the ADC/DAC chip over the 3-wire serial interface. Writes Data[31:0] to the Control section RAM.
RFAGC (0x00000002)	Y	Sets the RF AGC bus to Data[5..0].
RFDLY (0x00000003)	Y	Sets the RF DLY bus to Data[3..0].
ADCDIRECT (0x00000004)	Y	Sends Data[21:0] 2-sample packets from the ADC directly to Raw.
ADCTORAM (0x00000005)	Y	Sends Data[21:0] 2-sample packets from the ADC to the ADC section RAM on the FPGA.
ADCFROMRAM (0x00000006)	Y	Sends Data[21:0] 2-sample packets from the ADC section RAM on the FPGA to Raw.
DACDIRECT (0x00000007)	Y	Sends Data[21:0] 2-sample packets from Raw directly to the DAC chip.
DACTORAM (0x00000008)	Y	Stores Data[21:0] 2-sample packets from Raw in the DAC section RAM on the FPGA.
DACFROMRAM (0x00000009)	Y	Sends Data[21:0] 2-sample packets from the DAC section RAM on the FPGA to the DAC chip.
GETSTATUS (0x0000000A)	N	Returns a status word to Raw.

Table B.2: Wireless board controller commands.

Instruction (encoding)	Uses Data	Description
GETRFREG (0x0000000B)	Y	Returns the RF chip register as specified by Data[2:0] to Raw.
GETADCREG (0x0000000C)	N	Returns the latest 32-bit value written to the ADC/DAC chip control register.
GETRFAGC (0x0000000D)	N	Returns the current value of the RF AGC bus.
GETRFDLY (0x0000000E)	N	Returns the current value of the RF DLY bus.
BOARDRESET (0x0000000F)	Y	Sets the BOARD_RESET signal to Data[0].
TEST (0x00000010)	Y	Returns Data[31:0].
DAC_CLK_DCM_RESET (0x00000011)	Y	Sets the DAC_CLK_DCM_RESET signal to Data[0].

B.2 Status Word

This section describes the status word returned to Raw by the controller. Figure B-1 shows the breakdown of the bits in the status word.

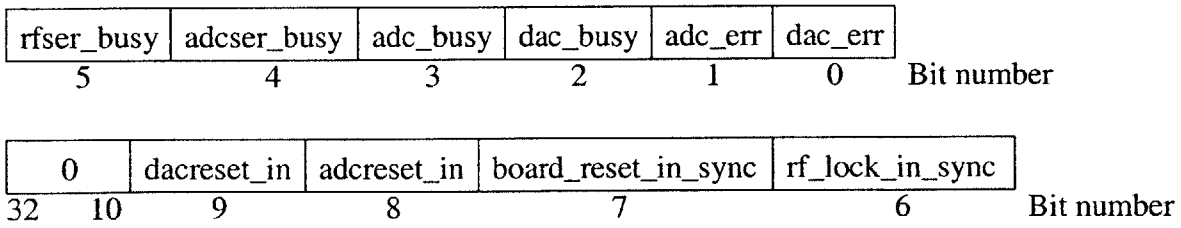


Figure B-1: Bit-level breakdown of the status word.

1. dac_err: This bit is high if an error has occurred in the DAC section. dac_err is set high if a DACDIRECT command is being processed and no data is currently available from Raw. DACDIRECT sends data to the DACs every cycle. If, at any given cycle, no data is available from Raw to send to the DACs, an error

signal is sent to the Control section, and dac_err is set high. dac_err is reset to 0 when the GETSTATUS command is processed.

2. adc_err: This bit is high if an error has occurred in the ADC section. adc_err is set high if an ADCDIRECT command is being processed and no data can be written to Raw because the buffers are full. ADCDIRECT sends a word from the ADC to Raw on every cycle. If that word cannot be written to Raw because the buffers are full, then the word will be lost, and an error signal is sent to the Control section. The Control section then sets adc_err high. adc_err is reset to 0 when the GETSTATUS command is processed.

3. dac_busy: This bit is high when the DAC section is busy processing a command. When the DAC section has finished processing the command, it sends a signal to the Control section, which resets the dac_busy bit.

4. adc_busy: This bit is high when the ADC section is busy processing a command. When the ADC section has finished processing the command, it sends a signal to the Control section, which resets the adc_busy bit.

5. adcser_busy: This bit is high when the ADC/DAC serial block is busy sending a 32-bit word to the ADC/DAC chip. This bit is reset low once the ADC/DAC serial block is finished.

6. rfser_busy: This bit is high when the RF serial block is busy sending a word to the RF chip. This bit is reset low once the RF serial block is finished.

7. rf_lock_in_sync: This is the synchronized RF_LOCK status signal from the RF chip. The RF_LOCK signal indicates whether the PLLs on the RF chip have locked. Specifically, it can indicate whether either the RX PLL has locked, or whether the TX PLL has locked. Which one the RF_LOCK signal corresponds to is set in the SYNTH_MISC register on the RF chip (register 6).

8. board_reset_in_sync: This is the synchronized BOARD_RESET signal from the wireless board. This signal is an input to the ADC/DAC chip and can be

controlled either via a pushbutton switch on the board, or via a control line from the FPGA. The command BOARDRESET sets this control line. The reset signal is active low.

9. adcreset_in: This is the reset signal for the ADC section. It is generated externally to the controller, in the top-level module for the FPGA. This signal will only release the ADC section from reset only if the ADC DCMs have locked, and no other global reset conditions exist. This signal is active high.

10. dacreset_in: This is the reset signal for the DAC section. It is generated in the same manner as the adcreset_in signal except that it will release the DAC section from reset only if the DAC DCMs have locked and no other global reset conditions exist. This signal is active high.

B.3 Caveats

This section covers useful miscellaneous information that a new user of the controller should be aware of. It is hoped that these notes will prevent some debugging issues.

1. There is an issue with using the DCMs to lock onto the ADC and DAC clocks. If the DCM is initially unable to achieve lock, then the DCM must be manually reset to cause it to reattempt lock. This is particularly important in the case of the DAC clock:

   The DAC clock sent to the FPGA is the output of a PLL on the wireless board. One input of the PLL is the DAC clock provided by the ADC/DAC chip. The other input is the DAC clock generated by the DCM on the FPGA. See Figure 4-1 for a schematic.

   When the board is first powered on, and both inputs to the PLL are 0V, the PLL will output a clock signal to the FPGA. This will cause the DCM to attempt to lock onto this signal. However, it will be unable to, and it will eventually fail to lock. This means that when the DAC is turned on, the DCM on the FPGA

will not attempt to lock since it has already failed to do so. This is the reason that the DAC_CLK_DCM_RESET command exists. With this command, the DCM on the FPGA can be kept in reset until the DAC has been turned on, and then can be allowed to lock.

So, when using the DAC section and the DAC, the following steps must be performed in this order.

(a) Turn the DAC on via the 3-wire serial interface.

(b) Set the DAC_CLK_DCM_RESET signal to 0 via the DAC_CLK_DCM_RESET command.

2. The RF_AGC control line is not intuitive. While the minimum gain setting is (Bit 5)000000(Bit 0), the maximum gain setting is (Bit 5)111101(Bit 0). See the RF chip documentation for more information.

Bibliography

[1] A. Chandrakasan, H.-S. Lee, and C.G. Sodini. Wireless Gigabit Local Area Network. Microsystems Technology Laboratories Annual Report, Massachusetts Institute of Technology, MIT, Cambridge, Massachusetts, December 2003.

[2] IEEE 802.11 Working Group. *Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications.* The Institute of Electrical and Electronics Engineers, Inc., 1999.

[3] IEEE 802.11 Working Group. *Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. Amendment 4: Further Higher Data Rate Extension in the 2.4 GHz Band.* The Institute of Electrical and Electronics Engineers, Inc., 2003.

[4] M. Krstic, M. Koushik, A. Troya, E. Grass, and U. Jagdhold. Implementation of an IEEE 802.11a Compliant Low-Power Baseband Processor. In *6th International Conference On Telecommunications In Modern Satellite, Cable and Broadcasting Service*, 2003.

[5] T.H. Meng, B. McFarland, D. Su, and J. Thomson. Design and Implementation of an All-CMOS 802.11a Wireless LAN Chipset. *IEEE Communications Magazine*, 41(8):160–168, August 2003.

[6] J. Steinheider, V. Lum, and J. Santos. Field Trials of an All-Software GSM Base Station. In *2003 Software Defined Radio Technical Conference*, November 2003.

[7] Michael Bedford Taylor. The Raw Processor Specification. Technical Memo, CSAIL/Laboratory for Computer Science, MIT, 2004.

6964-69