INTEGRATION OF ERROR DETECTION AND CORRECTION IN THE INFOPAD: PROCEDURE AND SPECIFICATION

by

Heather Bowers

Memorandum No. UCB/ERL M97/32

12 May 1997

INTEGRATION OF ERROR DETECTION AND CORRECTION IN THE INFOPAD: PROCEDURE AND SPECIFICATION

by

Heather Bowers

Memorandum No. UCB/ERL M97/32

12 May 1997

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Integration of Error Detection and Correction in the InfoPad: Procedure and Specification

by

Heather Bowers

Master of Science
in
Electrical Engineering and Computer Science

University of California at Berkeley

Abstract

Error detection and correction can improve the reliability of communication over the indoor wireless link of the InfoPad system. This report discusses the implementation of a variable forward error correction (FEC) scheme that uses (15,11,1) and (15,5,3) BCH codes. This project involved integrating an FPGA implementation of these FEC schemes with the InfoPad system. Integration of the new FPGA was perfomed so as to maintain compatibility with an older FPGA design and the ARM processor kernel. Hooks for a variable quality of service (QOS) scheduler were also added in order to make bandwidth and reliability tradeoffs through changes to the FEC level. With the addition of the error correction schemes, the InfoPad can be used for measurements of error rates on the channel to characterize the effects of short FEC codes on the reliability of indoor wireless communication.

Table of Contents

List of Figures

List of Tables

Acknowledgments

As with any endeavor, this research project could not have been completed without the help and support of others. Foremost, I wish to thank my research advisor Professor Robert Brodersen for his guidance throughout the course of this project. I would also like to thank Professor Jan Rabaey for reviewing this thesis .

I am greatly indebted to the members of this project. Trevor Pering taught me a lot about system level design, as well as how to have the patience and perserverance to debug as complex a system as InfoPad. Thanks also to Tom Truman for our conversations on bas-estation design and error correction. Jeff Gilbert was another great resource on the project, and I am especially grateful for his TCL/TK application that enabled the demo at the last InfoPad retreat to run smoothly. Brian Richards also contributed to the debugging process and my system level understanding of InfoPad. Fred Burghardt was responsible for mesh-ing InfoNet with the hardware. I would also like to thank Sue Mellers, Kathy Lu, and Rich-ard Edell for their support throughout the project.

In addition, I would like to thank my colleagues and friends Dennis Yee and Ian O'Donnell . I would also like to thank Jonathan Maltz for his understanding and moral sup-port throughout the project. For their administrative support, Tom Boot, Elise Mills, and Sheila Humphreys receive my sincere gratitude. I would also like to thank the Advanced Research Projects Agency for their generous financial support.

1 The InfoPad System Overview

1.1. Overview of the InfoPad Project

Infopad is an indoor mobile computing solution that allows the user to communicate to a workstation via a pad terminal connected over a wireless link. A basestation serves as a buffer between the wireless link to the pad and a networked link to a host computer where the majority of the processing occurs. Users are able to work on a small, notepad sized terminal via a pen input, while maintaining wireless connectivity to their accounts. Hence, functionality is not sacrificed for portability.

The pad functions only as an I/O interface to remote computing resources which reside on a high-speed, high-performance backbone network. All compute intensive operations, application support, and network management are located in this host network. The main role of the Pad is to provide the network with input from the user, and to provide the user with output from the network.

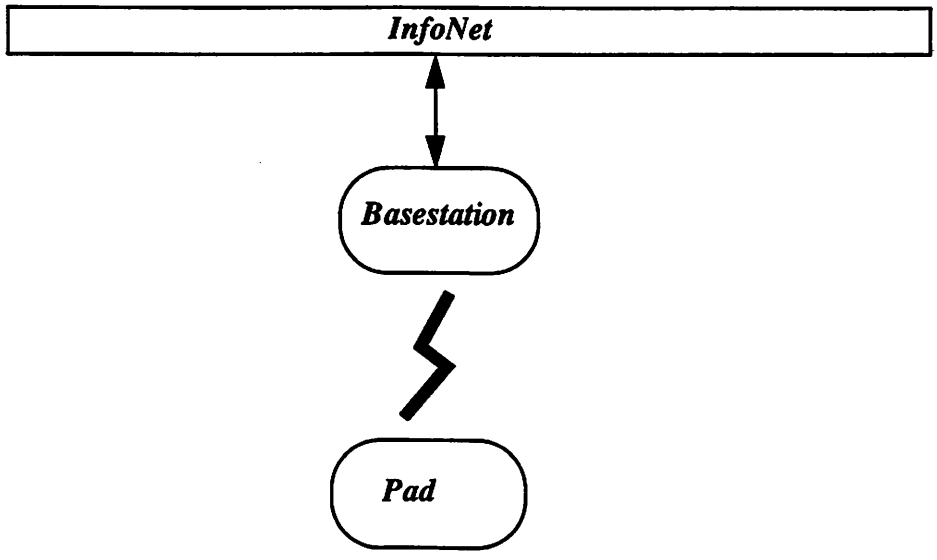
The pad is a simplified device that consists of transmit and receive radios, a display, a pen, and an audio system. Together, these components provide the user with a multimedia interface to the InfoPad system.

The hardware components of the system are a pad terminal and a basestation whose main purpose is to buffer the communications between the pad and network. The system nature of this project has provided valuable insight on systems research, as well as serving as a catalyst for supporting research in areas that enable the entire system operation. Developments in low-power technology, protocols, and radio design have progressed as a result of the InfoPad project. This master's project addresses the role of error correction in the context of the InfoPad system.

1.2. Overview of System DataPath

The InfoPad system consists of the pad, basestation, and InfoNet. InfoNet is a term used to refer to the general cloud of processing that occurs on the Sun Workstation, the software side of the system operation. Fred Burghardt is now responsible for this part of the system, and in this document, InfoNet can be conceived of as that part of the system which handles the exchange of data between the basestation and the application [Le95]. The basestation acts as a buffer between the wired GPIB connection to the Sun Workstation and the wireless radio connection on the pad. The final component of the system is the pad itself, the user interface that provides a display to the user and accepts pen and audio input.

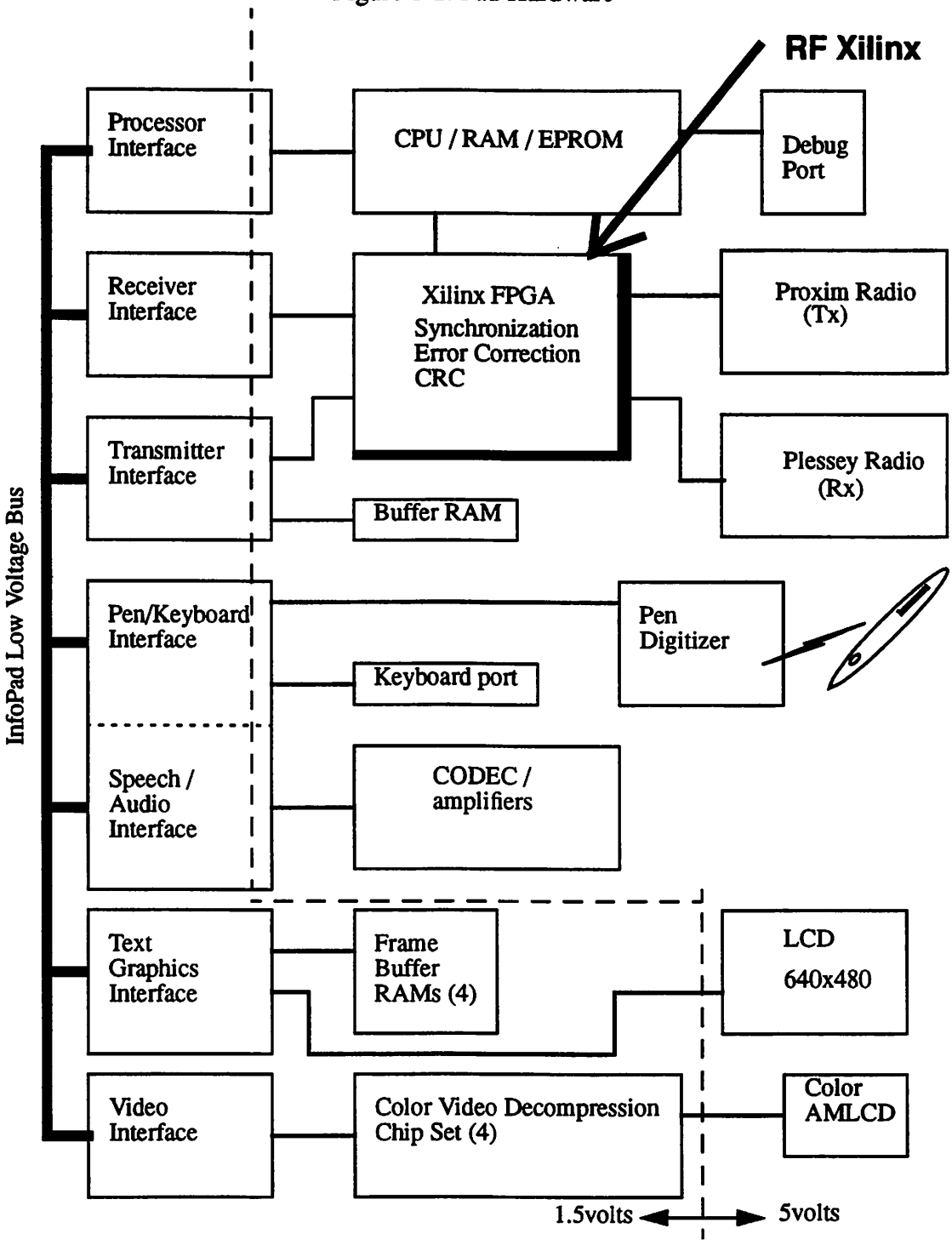
Figure 1-1. Overall System



This project addressed the hardware of the system, the pad and basestation. The pad and basestation hardware have identical architectures, so the following detail of the pad hardware in Figure 1-2 can also be used to see the operation of the basestation. The main differences between the pad and basestation are as follows: the radios used for transmitting and receiving in the pad have reversed operation in the basestation, the basestation has an interface to the GPIB port, and the pad has an LCD display and pen/audio inputs.

Specifically, this project addressed modifying the ARM kernel and the RF Xilinx design in order to add error detection and correction to the transmit and receive datapaths on the pad and basestation. These components are highlighted in Figure 1-2.

Figure 1-2. Pad Hardware

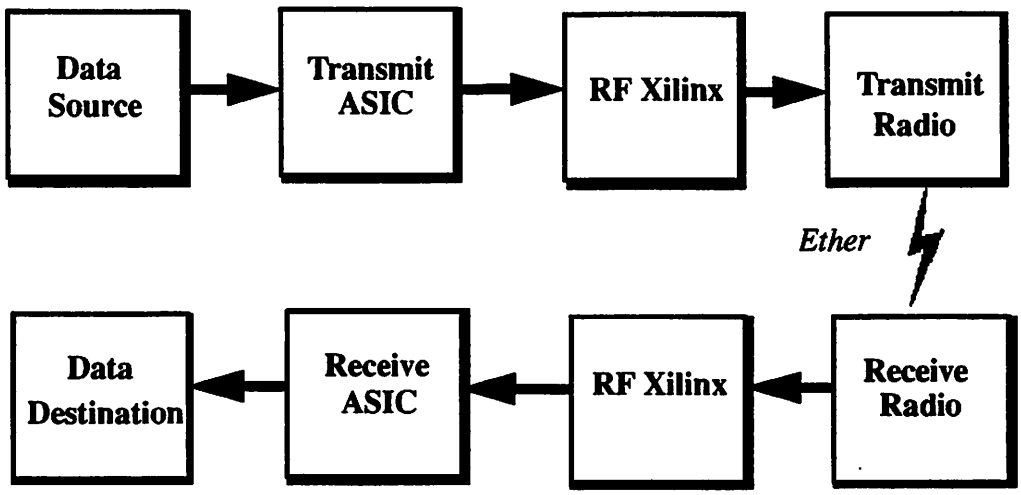


The ARM processor is responsible for the initial configuration of the RF Xilinx. After the RF Xilinx is configured, then the data path processing occurs as follows.

Data can originate from several sources within the pad, such as the pen interface, the keyboard interface, or the ARM processor. This data is sent over the IP bus to the TX chip, where the header data is added. The newly packaged data is then sent to the RF Xilinx. The RF Xilinx then optionally calculates and adds the header and body CRCs, as well as performs forward error correction on the packet. The header of each packet is always forward error corrected. The RF Xilinx serializes the packaged data and passes it to the transmit radio. In the case of the pad, the transmit radio is the Plessey, while on the basestation the transmit radio is the Proxim (See Section 1.3.1. and Section 1.3.2 for more information on the radios.)

On the receive end, the data enters the receive radio (the Proxim radio on the pad, the Plessey radio on the basestation) and then flows to the RF Xilinx. The RF Xilinx decodes the incoming data stream and performs a serial-to-parallel conversion on the data. This data is then sent to the RX chip. The RX chip sends the data over the IP bus to the appropriate destination determined by the header data.

Figure 1-3. InfoPad Data Path



1.3. InfoPad Radios

1.3.1. Plessey

The InfoPad uses the GEC Plessey Semiconductors' DE6003 digital radio transceiver (hereafter referred to as the Plessey radio) as the transmit radio on the basestation and the receive radio on the pad. The Plessey radio is a 2.45 GHz spread spectrum, frequency hop, wireless transceiver designed for data rates up to 625 Kb/s. The radio will operate over 101 channels spaced in 1 MHz intervals within the frequency band 2.400 to 2.500 GHz.

The radio has self contained RF functions but requires the host to provide control signals for data management, hopping channel assignment, and receive/transmit mode selection. The Plessey radio also requires the host to perform clock recovery. A table of all the signals in the Plessey interface can be seen in Table 1-1. Although the Plessey is a hopping radio, hopping was not enabled on the current version of the ARM kernel in InfoPad.

Table 1-1. Plessey Radio Interface

Signal	I/O	Description
Antenna Select	I	Diversity Control used to overcome signal nulling from destructive interference from multipath
Standby	I	Enables/Disables radio
Channel Select	I	Binary Frequency Channel Select. Channel Select values range from 0 (2.400 GHz) to 100 (2.500 GHz) in 1 MHz increments.
Channel Select Load Pulse	I	Strobe for Channel Select data
Transmit Data In	I	Serial Data to be transmitted
Power Level Control	I	Selects high or low transmit power input to the antenna
Power Amp Control	I	Allows transmit amplifier to be turned on after the mode has been set to transmit, eliminating unwanted emissions
Transmit/Receive Select	I	Switches between transmit and receive modes.
Receive Data Out	O	Serial Received Data
RSSI	O	Received Signal Strength Indicator
Synth Lock	O	Monitors lock condition of all PLLs to indicate if there is an error condition
Clk	O	Synthesizer Clock Output

[Ples94]

1.3.2. Proxim

The InfoPad uses the Proxim RDA-500/2 radio (hereafter referred to as the Proxim radio) as the transmit radio on the pad and the receive radio on the basestation. The Proxim radio is a direct sequence spread spectrum wireless transceiver designed for data rates up to 242 Kb/s. The radio will operate over the frequency band from 902-928 MHz and has three full channels. The Proxim RDA-500/2 was designed with a parallel data interface to provide compatibility with a microprocessor. However, the InfoPad was designed with a serial interface. Although the RDA-500/2 specification does not indicate the presence of a serial mode, the serial mode from the Proxim RXA series is still accessible in the RDA series. Hence, a return to the RXA specification is required to understand the serial interface, which appears as grounded pins in the RDA specification.

The radio has self contained RF functions but requires the host to provide control signals for data management, and receive/transmit mode selection. The Proxim radio provides its own clock recovery. A table of all the signals in the Proxim serial mode interface can be seen in Table 1-2.

Table 1-2. Proxim Radio Interface

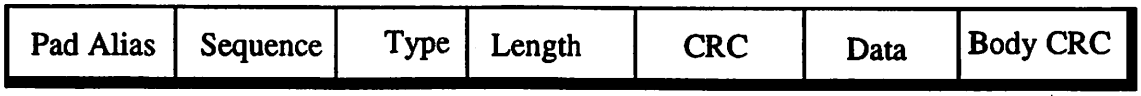
Signal	I/O	Description
TXDATA	I	Serial Data to be transmitted
SYNLD	I	When set high, SYNLD loads the data from the synthesizer's shift register into its control registers.
TXCLK	O	Transmit Clock used to shift the serial data into the radio.
SYNCLK	I	Synthesizer Clock. Used to clock SYNDATA into the synthesizer's shift register.
RXDATA	O	Serial Receive data.
SYNDATA	I	Serial data to program the synthesizer's frequency.
CD	O	Carrier Detect. Active when a valid carrier has been detected.
TX/RX	I	Transmit/Receive Mode Select Line.
STANDBY	I	When in Standby, the digital electronics are reset and the radio is put in a low power state.
RSSI	O	Received Signal Strength Indicator from demodulator.

[Prox92],[Prox90]

1.4. Wireless Transmission Protocol

The transmission protocol over the InfoPad's wireless link is packet based. The packet begins with a 32 bit frame sync which is followed by the header of the packet. The header of each packet can contain a pad alias, sequence number, data type, length, and Header CRC. Only the type has to be included in the header. All other features are optional, i.e they do not have to be included in the header. The packet of data is terminated by an end-of-packet byte which can include the Body CRC.

Figure 1-4. Protocol Format



Pad Alias

As the basestations are capable of supporting multiple pads, it is necessary to have an identification tag for each packet, hence the pad alias. A pad should only receive data which has a corresponding pad alias which matches its own pad identification number. When a pad transmits data, it also includes a pad alias in the headers of its transmit packets. This enables the basestation to identify the source of its received data. The pad alias is a one byte field in the header.

Sequence Number

The sequence number is included in the header in order to determine if packets have been dropped over the wireless link. The ARM programs the Transmit ASIC with the starting value of the sequence number, then the Transmit ASIC automatically increments this value and places it in the header of each packet to be transmitted. This sequence number is a one byte field in the header.

Type

The packet type is included in the header in order to determine the destination within the pad or basestation for the received packet. Examples of packet types are pen,

keyboard, video, text graphics, and control packets. The type space is limited to the lower 6 bits of the 1 byte word. The type field is 1 byte wide, and the most significant bit of the type field is the start of packet bit which is always set.

Length

Each packet type can be specified to have fixed or variable length. If the packet type in question has a fixed length, then the length field does not have to be included in the header. If the length of the payload is variable, then the length field must be present in the header. The length field refers to the number of bytes in the data payload and is a 2 byte wide field.

Header CRC

The header CRC is calculated on the header of the data packet by the RF Xilinx and appended to the header. The header CRC is a 1 byte wide field.

Body CRC

The body CRC is calculated on the entire header and payload by the RF Xilinx and is appended to the packet. The body CRC is a 1 byte wide field.

1.5. Error Detection and Correction (EDAC)

1.5.1. Background

The error correction codes used in this project are Bose-Chaudhuri-Hocquenghem (BCH) forward error correcting codes. BCH codes are linear cyclic block codes that can correct multiple random bit errors. They are the best known large class of linear cyclic block codes, and they allow for simple decoding for most practical cases. BCH codes exist for all integers m and t such that

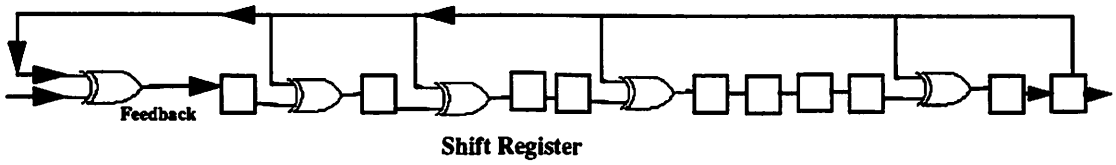
$$n = 2^m - 1$$
$$d = 2t + 1$$
$$n - k \leq m * t,$$

where n is the number of bits per word, k is the number for information bits, n-k is the number of bits of parity check bits (redundancy), m is the degree of the generator polynomial, and t is the number of errors per block that can be corrected.

The generator polynomial for the BCH codes can be chosen from pre-calculated code "cookbooks", or they can be derived with knowledge of finite field theory.

The BCH encoder is most commonly implemented as a shift register with feedback (Figure 1-5). The encoder adds redundancy to each message via this structure.

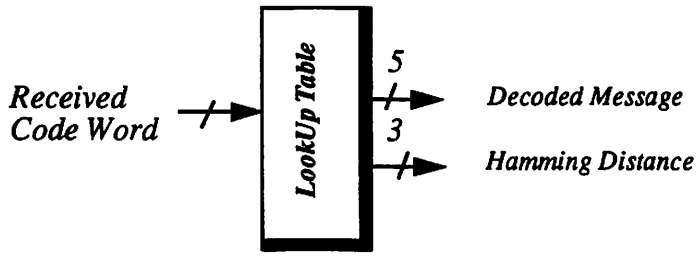
Figure 1-5. Typical BCH Encoding Structure



$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$$

Decoding is a more complicated procedure that usually involves a lookup table. The received word acts as the address to index the lookup table, whose output data is the decoded message and the Hamming distance (number of errors) of the decoded message from the received code word.

Figure 1-6. Typical BCH Decoder Structure



1.5.2. Application to InfoPad

The focus of this project is to provide the InfoPad system with error detection and correction (EDAC) in order to enable a more reliable data transfer across the wireless link. The theory supporting the implementation of this project can be referenced in Kathy Lu's master's thesis [Ylu93]. Three different levels of forward error correction are provided in order to provide a variable, dynamic error correction scheme. The motivation for having different levels of error correction derives from the varying degrees of error tolerance and bandwidth that are inherent in each data type. For example, pen data might require a higher level of error correction and will occupy less bandwidth than video data. Video data suffers very little from errors, as the display is constantly being refreshed. Using forward error correction on an error tolerant, high bandwidth data type such as video data could be a waste of resources. However, error correction could potentially benefit low bandwidth critical data packets, such as control packets.

The packet headers could also be encoded to add only a small bandwidth increase on the first 6 bytes of data, yet provide for more reliable packet routing. A CRC on the header of each packet can also be used to filter packets with corrupted headers. If the header is corrupted, then the destination for the packet and/or its length are uncertain, so rather than process this packet, it should be thrown out. Body CRCs can also be used on payloads in the same manner as header CRCs filter bad headers. However, often if the payload has a few errors, it is still acceptable for display, etc. and throwing out this packet based only on the payload CRC could cause a substantial loss of usable data.

The implementation in the InfoPad EDAC system is summarized below. A CRC is performed on the header and payload of the data packets. The header of each packet is maximally error corrected with a (15,5,3) BCH code. The Payload data can be transmitted using no encoding, a (15,11,1) BCH encoding scheme, or a (15,5,3) encoding scheme. The (15,11,1) code operates on block lengths of 11 bits and outputs block lengths of 15 bits that includes the added redundancy. These codes are capable of correcting 1 error per 15 bits and occupy 1.36 times the bandwidth of uncoded data. The (15,5,3) code operates on block lengths of 5 and outputs block lengths of 15 that includes the added redundancy. [LinCostello] These codes are capable of correcting 3 errors per 15 bits and occupy 3 times the

bandwidth of uncoded data. These error correcting codes are quite short in order to facilitate a smaller, less complex design due to the limitations in design size imposed by the Xilinx 4010 FPGA. Hence, the codes are block codes, implemented with a lookup table contained in an off-chip ROM. The benefit of a short block code is the simplicity of implementation and compactness of the design. However, short block codes can be ineffective in correcting burst errors. One proposed solution to this problem is interleaving the transmit and receive data in order to disperse block errors and allow the short block code to operate more effectively on the data.

Optimally, a scheduling algorithm would be developed to evaluate the bandwidth requirements, error tolerance of each packet, and state of the wireless link in order to select the most appropriate FEC level.

1.6. Report Format

This report serves as a specification for the EDAC (Error Detection and Correction) RF Xilinx design as well as a project report on the system integration procedure. The RF Xilinx design has evolved over time and is therefore the result of several people's efforts. This report is a summary of their past work, and a report on the latest developments concerning the design. While the author of this report performed modifications and system integration of the RF Xilinx design, the actual components were designed by other individuals. Craig Teuscher is responsible for clock recovery, while Dennis Yee and Craig Teuscher are responsible for the radio interfaces. Kathy Lu and Richard Edell implemented the error detection and correction algorithm. Tom Truman was instrumental in the ARM kernel, while Trevor Pering and Brian Richards maintained the RF Xilinx design since its inception. Lastly, Jeff Gilbert is the author of the InfoNet Control Panel.

Chapter 1 includes the introduction and overview, while Chapter 2 discusses the hardware implementation and software support necessary for the operation of the design. Chapter 3 discusses the system level issues that are need to be addressed in order to take advantage of the error correction capability in the RF Xilinx. Chapter 4 presents some measurements that were made concerning BER, area, and power consumption. Chapter 5 con-

cludes the report, and the Appendix includes the pin-out of the design in addition to the register descriptions and a listing of the location of reference files.

2 QOS Protocol

2.1. Overview

It is necessary to develop a protocol for the InfoPad that could allow a scheduler to provide varying qualities of service (QOS) to users. The actual algorithm for achieving this goal is a topic for further research, but in order to look toward the future, one must consider what hooks such an algorithm would need. The scheduler must have some knowledge of the state of the channel, a knowledge of the quality of service requirements of the user, and a way to relay feedback to the network infrastructure [Ylu96].

Using CRC information in combination with sampling the channel at regular intervals yields a practical system approach to a channel characterization process. A coarse knowledge of the channel can be obtained by accumulating the total of incorrect header CRCs and body CRCs over a period of time. The sequence number can also be used to determine how many packets have been dropped over the wireless link. Control packets can be used to transport a request for channel statistics and subsequent data obtained from this query. This is not a rigorous form of channel estimation, but it does provide a coarse measure to determine what level of error correction would be recommended for the channel.

2.2. Communication Through the System

One of the major issues in the system level design was how to communicate commands throughout the entire system, from an application running under InfoNet to the embedded control software running on the pad. Two methods were decided upon, the dynamic type system wherein the range of data types provided information about encoding

levels, and the control packet system which was used to communicate commands and responses throughout the system.

2.2.1. Static and Dynamic Data Types

The type space of the data protocol was arranged to support static data types whose error correction level is fixed (types 0x00 - 0x0f), and dynamic data types whose error correction is variable (types 0x10-0x3f).

The static data types are mapped to a fixed error correction level (that is usually the maximum encoding). These static types are low bandwidth and often carry critical control information, hence the bandwidth expansion of FEC has little significant effect and is justified by the theoretical gain in reliability.

Alternatively, the dynamic data types have optional forward error correction. A good channel and a high bandwidth application would warrant that error correction be disabled. For some applications, such as video, it is not apparent that forward error correction provides a qualitative gain in performance. FEC can actually detract from system performance by reducing the available bandwidth. Hence, types in the range of 0x10 to 0x1f are mapped to have no error correction. Each type in the range of 0x20 to 0x2f is mapped to medium error correction. Types in the range of 0x30 to 0x3f are mapped to the maximum error correction level. For example, video without error correction is designated as type 0x19. Video with medium level (15,11,1) error correction is designated as type 0x29, while video with the maximum level (15,5,3) of error correction is designated as type 0x39.

Specifying a dynamic range of data types based on error correction levels helps eliminate synchronization problems; there is no associated state on the receiving end. A request to change the error correction level results in a change of data type used on subsequent packet transmissions. The receive circuitry decides the error correction level based only on the data type and does not have to rely on synchronizing with the transmit error correction level.

2.2.2. Control Packet Protocol

As the type space defined in the transmission protocol is extremely limited, the control packet type was expanded by adding a subtype to the first four bytes of the data payload. Currently, the mobility control software shares this type space with the error correction control, however, only subtypes relative to error correction will be discussed here. The types are described in Table 2-1.

Table 2-1. Control Packet SubTypes Used in EDAC System

Control Subtype	Description
PACKET_TYPE_ERR_STAT_PAD_REQ	Request for error statistics from the pad
PACKET_TYPE_ERR_STAT_PAD_ACK	Acknowledge from pad including payload data that contains the error statistics
PACKET_TYPE_ERR_STAT_BASE_REQ	Request for error statistics from basestation
PACKET_TYPE_ERR_STAT_BASE_ACK	Acknowledge from basestation including payload data that contains the error statistics
PACKET_TYPE_CHANGE_PEN_ECC_REQ	Request to change the ECC level on the pen packets. 0 - No ECC 1- Medium Encoding 2 - Maximum Encoding
PACKET_TYPE_CHANGE_PEN_ECC_ACK	Acknowledge to a request for a change of the ECC level on the pen packets
PACKET_TYPE_PROTECT_PEN_REQ	Request to the basestation requesting that pen packets with corrupted payloads not be sent back up through the network 0 - Protect 1 - Unprotected
PACKET_TYPE_PROTECT_PEN_ACK	Acknowledge from the basestation indicating that pen packets will be protected according to the request

Hence, the control packets are responsible for requesting and supplying data on the channel characteristics. The control packets are also responsible for dynamically changing the error correction level on the pen packets, as well as whether or not the basestation will filter pen packets with corrupted payloads from the network. Note that because error correction levels are regulated at the source, uplink data traffic does not require some kind of synchronization protocol.

2.3. ARM Kernel Modifications

The mechanisms for utilizing the control packets were added to user level code in the ARM. This code performs the handling and decoding of the control packets that is necessary for providing feedback on the channel statistics as well as commands necessary for changing the error correction levels.

2.3.1. Control Packet Handling

A data stream dedicated to control packets was established in user code (*user.c*). Handlers for this data stream were registered in the ECC user code (*ECC.c*), so whenever a control packet arrives at the ARM on the control packet data stream, a handler is called that examines the first four bytes of data in order to establish the subtype of the control packet. Depending on the subtype, a specific subroutine is called to take action on the control packet's request and generate an acknowledge.

2.3.2. Changing the Pen ECC Level

The host generates a *Change Pen ECC Level* request which is sent over the GPIB link to the basestation. Upon encountering this particular control packet, the basestation forwards it to the pad, unchanged. The pad receives the control packet, identifies it, and examines the payload to determine which level of error correction is requested. Based on the level requested, the pen data type for the corresponding level of error correction is written to the Pen Chip. Hence, all subsequent data types from the pen interface correspond to this newly written data type. For example, the data type for a pen packet with no error correction is type 0x1b, while pen packets with medium error correction levels are of type 0x2b, and pen packets with maximum error correction correspond to type 0x3b. The pad then generates an acknowledge signal which goes to the basestation. The basestation, upon receiving the *Acknowledge a Change of Pen ECC Level* packet, forwards this packet to the GPIB interface and onward to the host. The acknowledge is used to ensure that the change of error correction level takes effect. The host also requires this acknowledge because the original request was implemented with an RPC protocol.

2.3.3. Changing the Protection on the Pen Packets

The host generates a *Change Pen Protection* request which is sent over the GPIB link to the basestation. Upon encountering this particular control packet, the basestation examines the payload to determine whether or not to forward pen packets with invalid payload CRCs to through the GPIB link to the host. Upon receipt of the control packet request, a call to the function *common_PassBadPen (common.c* in the kernel) is performed, and the protect/unprotected flag from the payload of the request control packet is passed to this function. If protection is selected, this function causes the basestation to discard pen packets originating from the pad if the payload CRC is invalid. If unprotection is selected, all pen packets are relayed back to the host, regardless of the validity of the payload CRC. Regardless of the protection level, the basestation generates an acknowledge signal that is sent to the GPIB interface and onward to the host. Again, the host requires this acknowledge because the original request was implemented with an RPC protocol.

2.3.4. Characterizing the Channel

The receive chip in the InfoPad has the capability to duplicate the headers on all received packets and send these headers to the arm on a pre-defined data stream. In this manner, the ARM can maintain counters that gather statistics on the uplink or downlink data (depending on whether the ARM is on the basestation or the pad). The number of packets with bad payload CRCs, the number of packets dropped over the wireless link, and the total number of packets that are received in a particular sample period are recorded. These values can be used to calculate the drop rate of packets across the uplink and downlink channels, as well as provide a sense of how many packets have errors in the payload.

This data would then ideally be fed to a variable QOS scheduler which could threshold the packet drop rates according to the quality of service required by the user. For example, if the user needed very high QOS, then a moderate packet drop rate would be identified with the need for maximum error correction. The scheduler would then monitor the packet drop rates, and change error correction levels based upon these predetermined thresholds.

Table 2-2. Counters maintained in ARM for Channel Characterization

Counter	Description
Header Count	Increments whenever a header is received
Bad Header Count	Increments whenever an invalid header CRC occurs
Transmit Count	Calculates how many packets were transmitted from sequence numbers received from packets with valid header CRCs *ECC_Tx_cnt += (seq - _ECC_seq_num + 256) % 256*
Pen Packet Count	Increments whenever a pen packet is received with a valid header CRC
Bad Pen Packet Count	Increments whenever a pen packet is received with a valid header CRC but an invalid payload CRC
Protected Text Packet Count	Increments whenever a protected text packet is received with a valid header CRC
Bad Protected Text Packet Count	Increments whenever a protected text packet is received with a valid header CRC but an invalid payload CRC
Text Packet Count	Increments whenever a text packet is received with a valid header CRC
Bad Text Packet Count	Increments whenever a text packet is received with a valid header CRC but an invalid payload CRC

When the host queries the basestation statistics, a *Basestation Error Statistics Request* control packet is sent across the GPIB link to the basestation ARM. The ARM packages the information from the counters into a *Basestation Error Statistics Acknowledge* control packet and sends this packet back through the GPIB link to the host. The ARM then clears the counters and begins collecting data for the next sample request from the host.

Similarly, when the host queries the pad statistics, a *Pad Error Statistics Request* control packet is sent across the GPIB link to the basestation ARM and forwarded to the pad. The pad generates a *Pad Error Statistics Acknowledge* control packet and sends this packet back to the basestation ARM, which forwards the packet back to the host. The pad ARM has meanwhile cleared its counters and begun collecting data for the next sample.

2.4. Host Interface to the Error Correction System

The control panel is an InfoNet utility that can be used to control FEC and ECC levels (Figure 3-1). The control panel communicates with the InfoNet monitor. As an automatic scheduler does not yet exist, the control panel acts as a manual interface to the error correction system. It allows the user to select the forward error correction level for the text graphics data packets, the protected text graphics data packets, and the pen packets. The user can also choose to utilize the payload CRCs to filter corrupted packets. The control panel also provides information on the channel characteristics via an automatic query or a manual query. The control panel performs other control related tasks, but as these are not directly related to the error correction system, further information can be found in Fred Burghardt's discussions on InfoNet.

2.4.1. Selecting the Error Correction Level

Whenever an error correction level for text graphics or protected text graphics data is chosen, the control panel sends a message to the monitor which re-maps the dynamic data type to the proper error correction level and sends this re-mapped type down to the basestation. This simple re-mapping in the monitor will not suffice for pen packets, however, as they originate from the pad itself. Hence, in order to change the error correction level on pen packets, the control panel sends a *Change Pen ECC Level* Control Packet down through the system to the hardware via an RPC protocol.

2.4.2. Selecting the Payload CRC Filter

Similar to the error correction level selection, in order to select a filter on text graphics data packet, a protected text graphics packet is sent down to the pad instead of the unprotected text graphics types. The re-mapping is performed in the InfoNet monitor. When the Text/Graphics chip receives the protected text graphics data type, it examines the payload CRC to determine if the data should be sent to the display. When filtering of pen packets based on the payload CRC is desired, the host sends a Protect Pen Control Packet down through the system to the hardware via an RPC protocol. When the ARM receives this control packet, it processes the request as described earlier in Section 2.3.3.

2.4.3. Channel Characterization

The control panel can be used to generate a single manual query of the channel statistics or a timed update that provides a new sample every second. Whether the manual or times query is used, the process for obtaining the samples is the same. A request for the channel statistics is sent to both the pad and basestation in the form of a control packet sent with the RPC protocol. The pad and basestation acknowledge the request with a packet that contains the data from the ARM's counters discussed in Section 2.3.4. The control panel application processes this data and displays the uplink and downlink characteristics. If a query is lost over the wireless link, then the acknowledge is never received by the host. In this situation the query is resent until a valid acknowledge is obtained. The question mark symbol ("?") is displayed in this instance, as not valid data is present. Divide by zero errors which are encountered in processing are displayed by a dotted line ("---").

2.4.3.1. Uplink

The uplink characteristics displayed are the transmit rate, the percent of packets received as compared to how many packets were transmitted, the receive rate of pen packets, and the percent of pen packets received that were error free.

The transmit rate is simply the value of the transmit counter that itself was derived from packet sequence numbers. Since the queries are approximately every second, this yields the approximate transmit rate in units of packets per second. This value is intended to indicate a coarse estimate of uplink traffic volume. The percent of packets received is calculated by dividing the header count by the transmit count. Additionally, pen packets receive rate is examined. This value is simply the value of the pen packet counter in the ARM at the time of sampling. The percent of error free pen packets received is calculated by dividing the number of good pen packets (pen packets received minus bad pen packets) by the number of pen packets received.

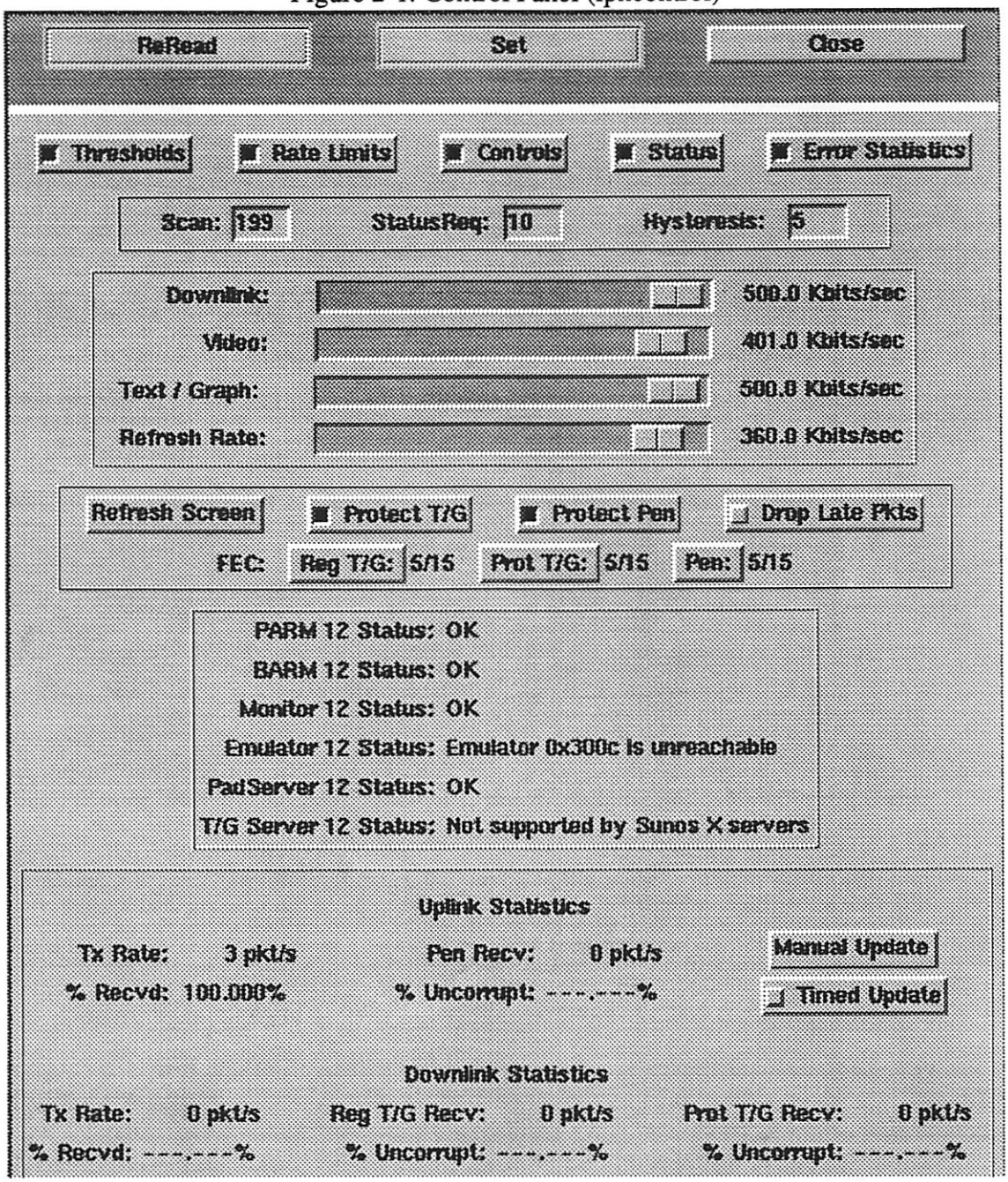
2.4.3.2. Downlink

The downlink characteristics displayed are the transmit rate, the percent of packets received as compared to how many packets were transmitted, the receive rate of protected

and unprotected text graphics packets, and the percent of protected and unprotected text graphics packets received that were error free.

Similar to the uplink case, the transmit rate is simply the value of the transmit counter that itself was derived from packet sequence numbers. Since the queries are approximately every second, this yields the transmit rate in units of packets per second. This value is intended to indicate the coarse estimate of downlink traffic. The percent of packets received is calculated by dividing the header count by the transmit count. Additionally, protected and unprotected text graphics receive rates are examined. These values are simply the value of the corresponding text graphics packet counter in the ARM at the time of sampling. The percent of error free text graphics packets received is calculated by dividing the number of good text graphics packets (text graphics packets received minus bad text graphics packets) by the number of text graphics packets received.

Figure 2-1. Control Panel (ipncontrol)



3 Implementation

3.1. Hardware

The RF Xilinx in the InfoPad originally did not support error detection and correction. It did, however, perform several crucial functions in addition to interfacing to the radios. A general purpose timer was implemented, as was an interface to the A/D converter in order to provide RSSI measurements. Serial-to-parallel conversion was performed on incoming data, and parallel-to-serial conversion was inversely performed on outgoing data. The RF Xilinx also provided the flexibility to perform work-arounds that were needed when problems with the custom chips were discovered. An example of this is when the frame sync was lengthened due to noise being mistakenly identified as the frame sync pattern for a data packet. Another example is when the Receive ASIC incorrectly received an infinite length packet. (The solution to this problem was realized by adding a kick circuit to the Xilinx that activates whenever an unreasonably long packet is detected).

This original RF Xilinx was in an evolutionary state due to the flexibility in debugging that it provided, hence the design was not frozen when the task of adding EDAC to the system was initiated. Further changes in the ARM kernel also impeded system integration. Another student, Kathy Lu, began the implementation of EDAC, but when her design was completed, her work was not compatible with the ARM processor's software kernel and the original Xilinx design.

This project addresses the integration of the EDAC RF Xilinx design with the requirements of the InfoPad system and includes the specifications for the resultant design. This new design has the feature that one combinational design maintained three separate designs for basestation, pad, and wired versions. While this combinational design mini-

mized design maintenance, it imposed a register file that differed from that driven by the ARM kernel. The timer was also missing from the design, as were the work-arounds of the receive kick circuit and the extended frame sync. A new method for reading RSSI values from the A/D converter was added in the EDAC RF Xilinx that proved to be incompatible with the ARM kernel. The different subsystems of the RF Xilinx will be discussed below.

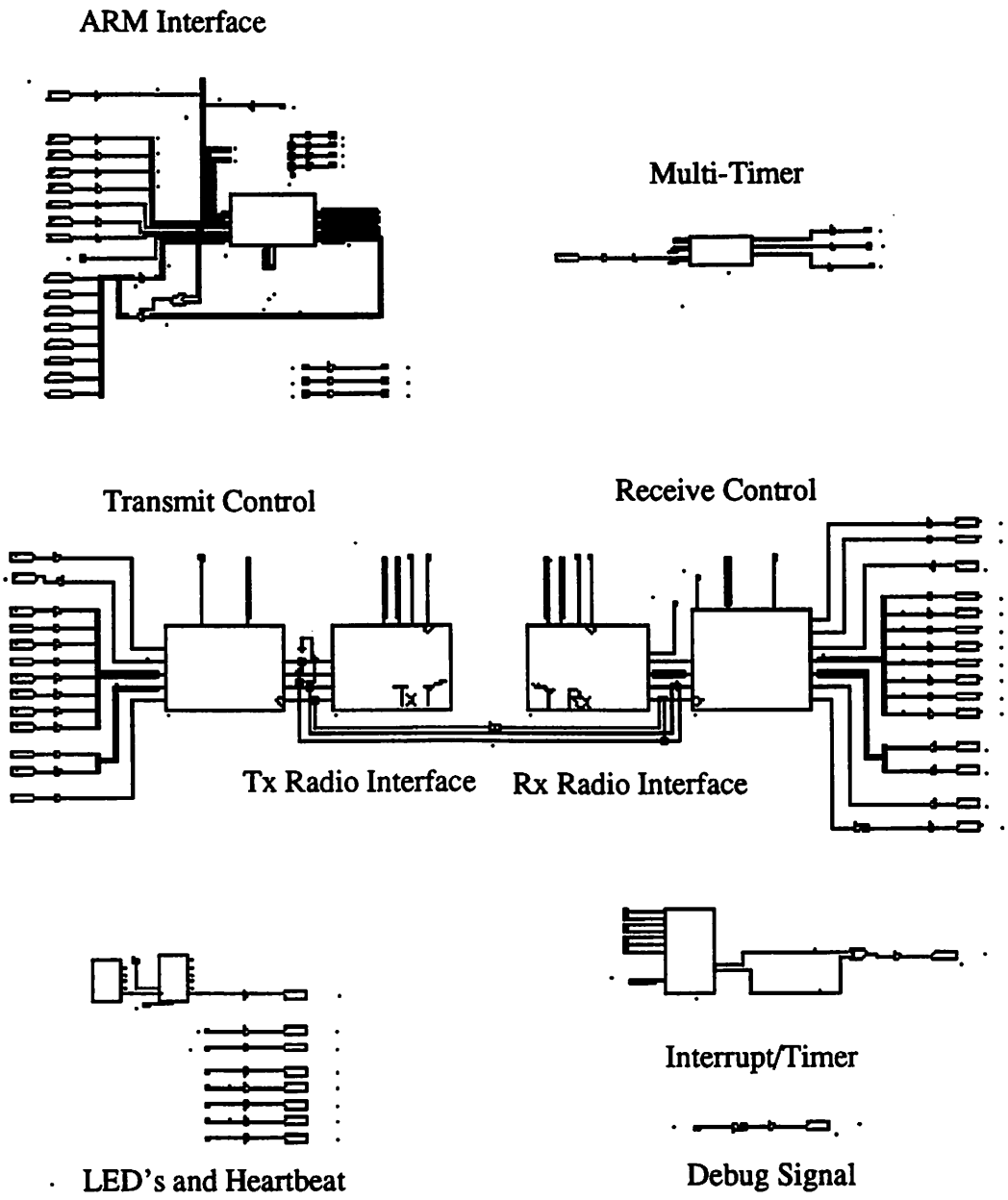
3.1.1. Top Level

There are three separate RF Xilinx designs maintained by a combination design in Viewlogic (*combo.1*). The first design is used in the pad when operating in wireless mode (*ecc_pad.1*). Likewise, there is a design for the basestation operating in wireless mode (*ecc_base.1*). Lastly, there is a design that can be used on either the pad or basestation that is used when operating in "wired" mode, i.e. the pad and basestation communicate via a cable, rather than over the wireless link (*ecc_wire.1*). The three different designs are necessary because the pad, basestation, and wired designs have differing interfaces to the radios.

The combination design contains several submodules that will be discussed in later sections. The radio interface submodules are black boxes that have the appropriate design placed in them during the *make* operation. In this manner, high level changes can be performed once, instead of on all three designs.

The Viewlogic diagram for the combination design can be seem in Figure 3-1.

Figure 3-1. Top Level Schematic for RF Xilinx Design



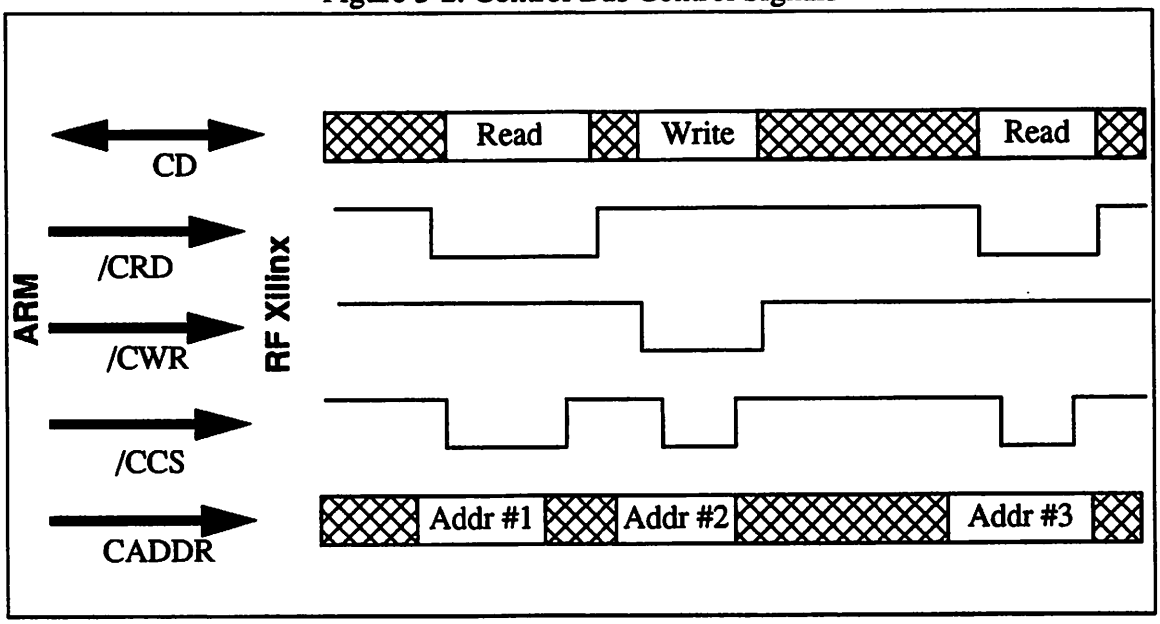
3.1.2. ARM Interface

The ARM interface subsystem of the RF Xilinx is responsible for providing an interface to the ARM and maintaining a register file that is used by the ARM, the RF Xilinx, and the radios. When the ARM desires to write to a register in the RF Xilinx, the chip

select for the RF Xilinx must be driven high and the proper address for the register must be placed on the 4-bit wide CADDR line. The ARMIF module's operation is described in VHDL (*armif.vhd*) and then depicted with a Viewlogic symbol in the top level of the RF Xilinx design (*combo.1*, *ecc_base.1*, *ecc_wire.1*, *ecc_pad.1*).

The ARM Interface block has basically two functions which are controlling reads and writes to the registers in the RF Xilinx.

Figure 3-2. Control Bus Control Signals



[IPBook]

3.1.2.1. Register Write Process

Internally, the RF Xilinx chip select, write strobe line, and the address from the ARM index a lookup table that outputs an internal write strobe line for each register (Figure 3-3). There are sixteen internal write strobe lines in the RF Xilinx, and sixteen 8-bit registers that are available for access. However, only thirteen registers are actually allocated for some purpose. These registers can be seen in Table 3-1.

Figure 3-3. Register Select for Write Operation

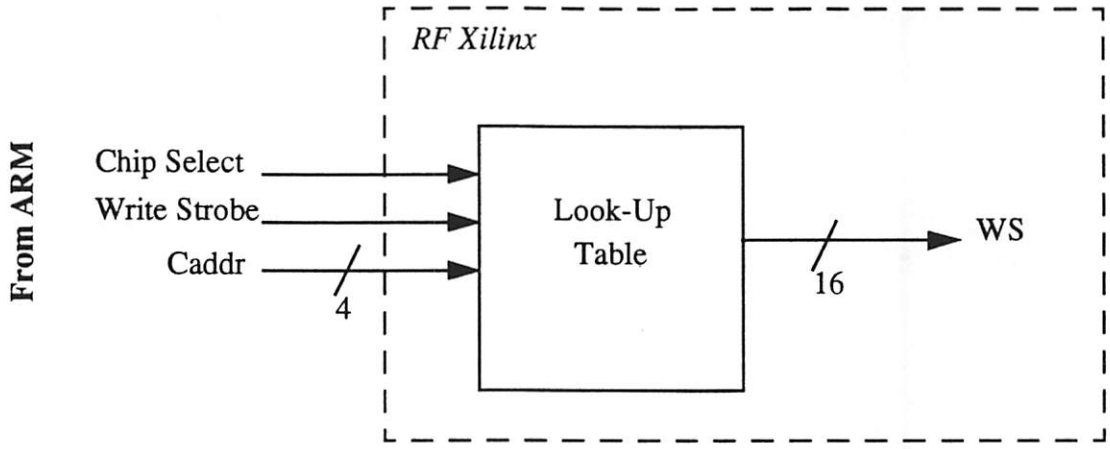


Table 3-1. Write only Registers in the RF Xilinx

Internal Write Strobe Line	Register Name	Description
0	Rx Sync Word (MSB)	Upper byte of the frame sync pattern for the receive radio
1	Rx Sync Word (LSB)	Lower byte of the frame sync pattern for the receive radio
2	Tx Sync Word (MSB)	Upper byte of the frame sync pattern for the transmit radio
3	Tx Sync Word (LSB)	Lower byte of the frame sync pattern for the transmit radio
4	Unused	
5	Rx Frequency Divider	Used for programming the receive radios
6	Unused	
7	Tx Frequency Divider	Used for programming the transmit radios.
8	Control	Text Graphics Reset. This test pin causes an active low reset for debugging purposes. Loopback Mode Enable. This bit causes the RF Xilinx to enter loopback mode which causes the transmit data to be routed back as received data, bypassing the radios. Multi-Time Reset.Resets the multi-time module in the RF Xilinx. System Reset. This is an internal system reset for the entire RF Xilinx.

Table 3-1. Write only Registers in the RF Xilinx

Internal Write Strobe Line	Register Name	Description
9	Rx Clock and Control	Receive Clock Divider Word. Used for programming the radios. Hop Reset (on the pad design)
10	Tx Clock and Control	Transmit Clock Divider Word. Used for programming the radios. Hop Reset (on the basestation design)
11	Timer (LSB)	General Purpose Timer Value (lower byte)
12	Timer (MSB)	General Purpose Timer Value (upper byte)
13	Interrupt Acknowledge	RSSI Interrupt Acknowledge Timer Interrupt Acknowledge
14	Unused	
15	LED Control	Control of the LEDs on the board. Used for feedback and debugging.

3.1.2.2. Register Read Process

There is one read port in the ARM interface register file that the ARM can access. When the ARM wishes to read a register, the ARM sets the lower two bits of the CADDR line to the appropriate register (Table 3-2). Only four registers are available for reading, and of these four only three are used.

Table 3-2. Read Only Registers in the RF Xilinx

CADDR[1:0]	Output	Description
00	Interrupt Status	Allows ARM to resolve which event (timer or rssi measurement) generated an interrupt.
01	RSSI(LSB)	Least significant byte of an RSSI reading.
10	RSSI(MSB)	Most significant byte of an RSSI reading
11	Unused	

3.1.3. Multi-timer Module

The multi-timer module is basically a set of clock dividers that take a 20 MHz clock input and convert it to the clock for the receive data path, the clock for the transmit data path, and the 1 MHz clock for the timer module. The multi-timer module can be reset by an internal reset called the Multi-Timer Reset. This reset is performed when the ARM

writes to the Multi-Timer Reset Register in the Control Register of the Xilinx.The multi-timer module is implemented entirely in a Viewlogic schematic (*multitime.1*). The counters and clock dividers used in the multi-timer module are implemented with Xilinx XBLOX macros to help ensure improved routing for the time critical clocks. Since the Multi-Timer module does address the problem of clock generation, it is important to mention the buffering entailed in the clock distribution. Since the Xilinx has a limited number of global buffers, they must be used prudently. The 20 MHz clock input enters the Xilinx on a pin that does not support a global buffer. Hence, this pin is re-routed to another pin that does have a dedicated global buffer at its I/O pad. This internal routing renders the input pin with the global buffer unusable, as its buffer is being occupied by the 20 MHz clock. The clocks that are output from the multi-timer module are buffered external to the multi-time module, often within the module that they are used to drive.

3.1.3.1. Generating the Transmit Clock

The transmit clock is generated by a 6 bit up counter that is implemented with an XBLOX counter module. The terminal count is determined by the value in the Transmit Clock Rate Register. Whenever the multi-timer reset occurs or the terminal count value is reached, the terminal count value is reloaded from the Transmit Clock Rate register. The counter is driven by the 20 MHz clock. In the present version of the InfoPad, the Transmit Clock Rate register value for the basestation (using the Plessey radio) is the decimal value of 48, corresponding to a 625 KHz transmit frequency. The Transmit Clock Rate register value for the pad (using the Proxim radio) is the decimal value of 14, corresponding to a 200 KHz transmit frequency.

3.1.3.2. Generating the Receive Clock

The receive clock is generated by a 4 bit up counter and includes the feature of an optional stall every sixteen cycles. The 4-bit up counter is implemented with an XBLOX counter module. The terminal count is determined by the value in the Receive Clock Rate Register. Whenever the multi-timer reset occurs or the terminal count value is reached, the terminal count value is reloaded from the Receive Clock Rate register (if the stall cycle option is disabled). The Xilinx can be programmed so that every 16 cycles the Receive

clock will stall for a cycle. This stall enable can be performed by writing to the Receive Control register of the Xilinx register file. The stall counter is implemented with an XBLOX module that counts to 16 before generating a terminal count signal. Both counters are driven by the 20 MHz clock. Whenever the stall enable occurs, the receive clock and the update of the Receive Clock Rate are disabled for one cycle. In the present version of the InfoPad, the Receive Clock Rate register value for the basestation (using the Proxim radio) is the decimal value of 6, corresponding to a 2 MHz receive frequency. The stall enable option is disabled so the Stall Enable register in the Receive Control Register must be zero. The Receive Clock Rate register value for the pad (using the Plessey radio) is the decimal value of 13, corresponding to a 6.25 MHz receive frequency. However, the stall option is enabled on the Plessey receive radio, so the Stall Enable Register must be set in the Receive Control Register.

3.1.3.3. Generating the Timer Clock

The Timer clock for the general purpose timer on the Xilinx is generated by using an XBLOX clock divider module. This module takes the 20 MHz clock input and divides it down to 1 MHz.

3.1.4. Transmit Data Path

The transmit control block is at the heart of the transmit data path. It is responsible for formatting the transmit data for the radios, performing forward error correction, and calculating hardware CRCs. This block is described in VHDL (*txpath_ecc.vhd*) and is instantiated in a Viewlogic schematic (*txctrl_ecc.1*). It begins processing of data whenever the transmit packet (TXP) line from the Transmit Chip is asserted. A parallel to serial conversion is performed on the data in order to format it for the transmit radio. While this parallel to serial conversion is being performed, the first eight bytes of every packet (the packet header) are forward error corrected (FEC) to the maximum FEC level. Subsequent bytes of transmit data are error corrected to the level specified by the TXECC signal. CRCs are calculated on the serial data stream, but are only injected when the TXCRC signal is asserted by the Transmit Chip. CRCs can be calculated on the header and on the entire packet. The final stage before sending the transmit stream to the radios is to exor the data with a code word that helps DC balance the signal in order to aid in transmission.

The I/O for the VHDL design is described below. The Viewlogic schematic merely replicates these input and output signals and does not include additional logic, so the schematic I/O description has been omitted.

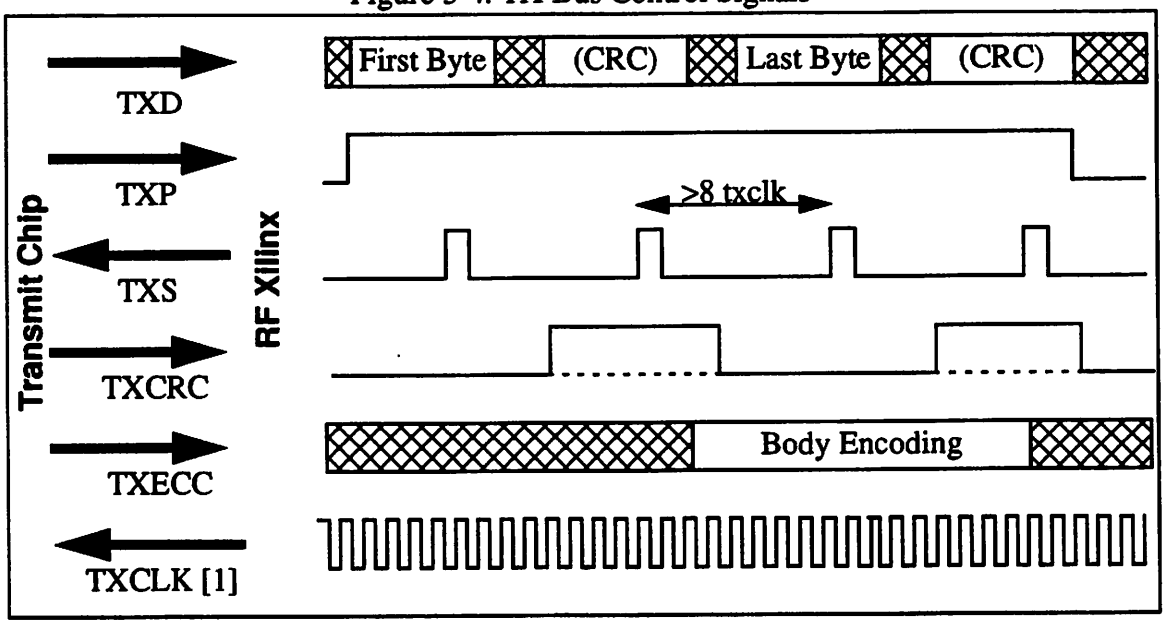
Table 3-3. I/O for the VHDL Design of the Transmit Data Path

VHDL Signal Name	I/O	Description
CLK	I	Transmit Clock
RST	I	Reset
TXP	I	Transmit Packet. Generated by TX Chip. High for the duration of a packet.
TXS	I	Transmit Strobe. Generated by TX Chip. Data Strobe for each byte of transmit data.
DIN[7:0]	I	Transmit Data from TX Chip. An acknowledge from the transmit radio indicating it is ready to transmit data.
TXECC[1:0]	I	Transmit Error Correction Level for Body Payload. 0 - No Error Correction 1 - Undefined 2 - Medium Level Error Correction (15,11,1) 3 - Maximum Level Error Correction (15,5,3)
TX_GRNT	I	Transmit Grant.

FRAME[15:0]	I	Frame Sync. Stored in the RF Xilinx register file in the TXSYNC register which was programmed by the ARM.
TXCRC	I	Transmit CRC. When this signal is asserted, the RF Xilinx injects the CRC into the data stream to the radios.
ZERO	I	Logical zero. Driven by ground.
ONE	I	Logical one. Driven by 5V.
DOUT	O	Output serial transmit data to transmit radio.
TX_REQ	O	Transmit request. A request from the RF Xilinx to the radio for a transmission.

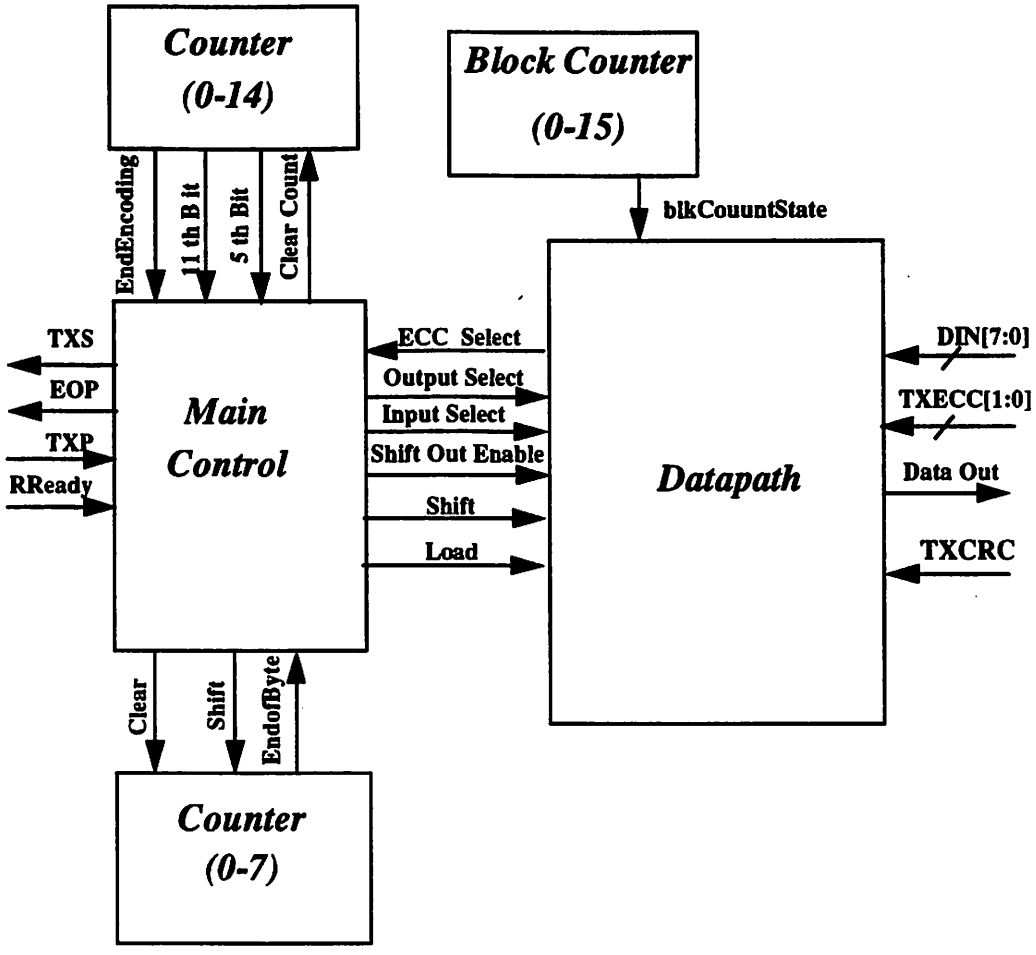
The interface timing signals for the TX Chip/RF Xilinx interface can be seen in the following Figure 3-4.

Figure 3-4. TX Bus Control Signals



[IPBook]

Figure 3-5. Encoder High Level Schematic

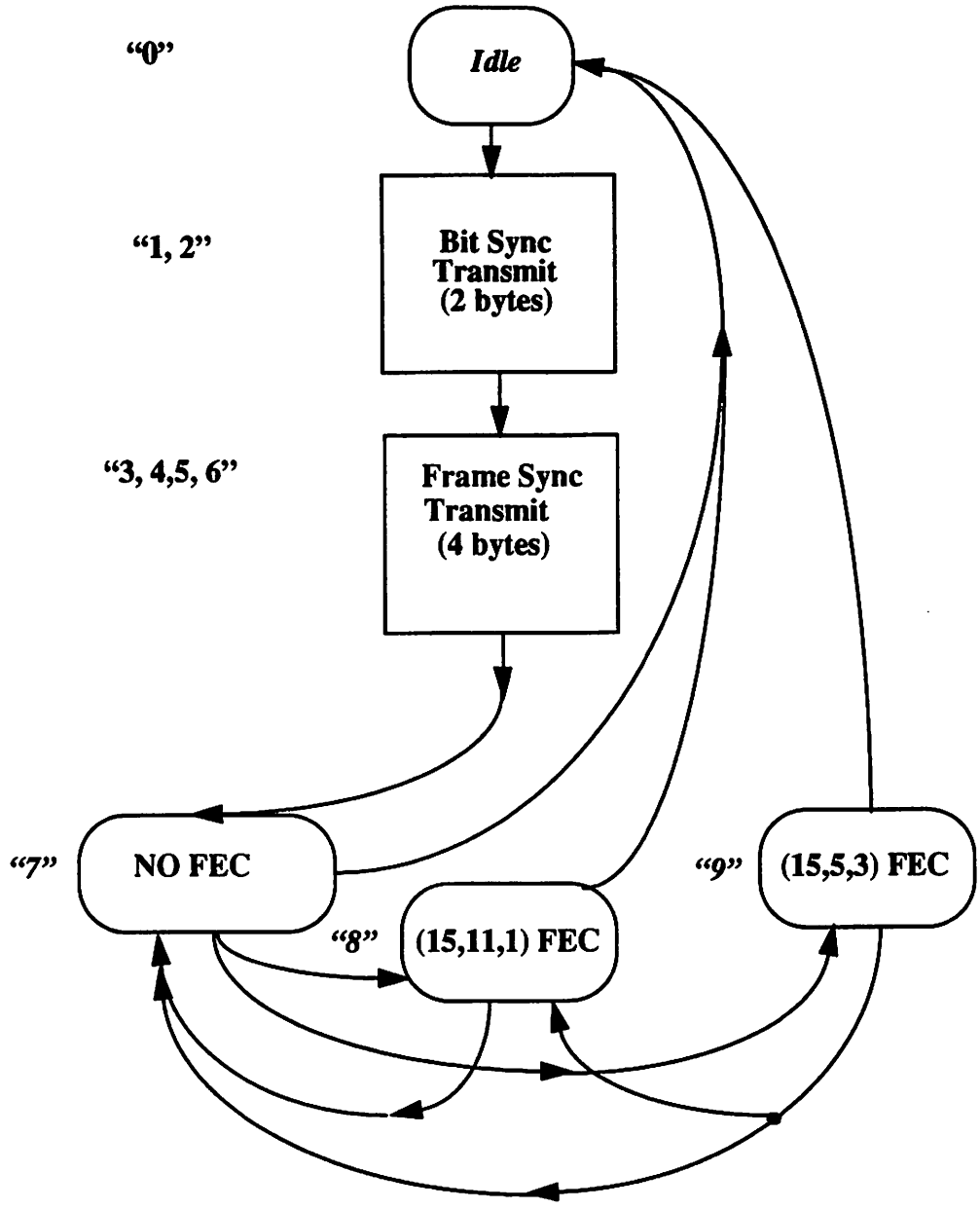


3.1.4.1. Control Logic

Main Control State Machine

The transmit control block is driven by a state machine with ten states called *Main Control*, given that it is the main control of the design. The simplified state diagram is presented below, and each state is discussed in more detail in the following section.

Figure 3-6. State Machine for the Main Control



Idle, State "0"

The select for the Input Mux is set to select the bit sync pattern, and the *load* signal is asserted. When *load* is asserted, the Bit Sync value is latched into the Input Data Shift Register. The Count8, Count15, and Block counters are initialized. The count15 and block counters are not actually used in this state, since they are mainly used for control of the

encoding.The state machine automatically advances to the *Bit Sync* state which begins the transmission of the bit sync.

Bit Sync, State "1, 2"

The bit sync is a two byte value that is sent to the transmit radio before any data is transmitted. The bit sync pattern is 0101010101010101. Once the state machine enters *State "1"*, the Count15 and Block counters are restarted. These counters are not used for the Bit Sync because the Bit Sync is not forward error corrected. The shift signal is asserted for each bit of the bit sync that is shifted out of the Input Data Shift Register. *Shift* begins the parallel to serial conversion of the data that was latched in *State "0"*, i.e. causing a shift of the Input Data Shift Register and hence an introduction of a new bit of data into the processing path. The select for the Output Mux is set to select non error corrected data.

The select for the Input Mux is set to select the second byte of the bit sync pattern (which is identical to the first byte) to be processed later in *State "2"*. When the *EndofByte* signal is asserted by the count8 counter indicating the processing of the first byte of the bit sync has been completed, the load signal is asserted to load this next byte of the bit sync into the Input Data Shift Register.

After the first byte of the bit sync has been transmitted, the state machine waits in *State "1'* until TXP is asserted by the TX Chip, indicating a packet has arrived and is ready to be processed. When TXP is asserted, the RF Xilinx generates a transmit request for the transmit radio. When the radio responds with a transmit grant, then the state machine advances to *State "2'* in order to shift out the second byte of the bit sync and to latch the first frame sync byte.

When the state machine enters *State "2"*, it performs similarly to *State "1"*. The count15 and block counters are cleared to 1, although they are not needed since FEC is not enabled. A transmit request is generated for each bit of data, and the *shift* signal is asserted to clock each bit of the bit sync out of the Input Data Shift Register. The *SetMoreStuff* signal is set high to indicate the processing of the byte was not completed, i.e. data is expected after the current bit being processed, which is also not needed unless FEC is per-

formed. The Output Mux select chooses non error corrected data output, and the Input Mux select chooses the first byte of the frame sync. When the entire byte of the bit sync has been clocked out of the shift register, i.e. *EndOfByte* is asserted, then the least significant byte of the frame sync is loaded into the input shift register. The state machine then advances to *State "3"*.

Frame Sync, States "3, 4, 5, 6"

The frame sync is 4 bytes wide. In an earlier implementation of the RF Xilinx, the frame sync was 2 bytes wide. These two bytes are programmable by the ARM and are stored in the RF Xilinx register file. However, problems arose when random noise in the transmit path was being falsely identified as the frame sync. Hence the original 2 byte pattern was repeated internally within the RF Xilinx to result in the current 4 byte pattern.

The state machine begins in state "3" by restarting the count15 and block counters. These counters are not used for the Frame Sync states because the frame sync is not forward error corrected. The state machine generates a transmit request to the radio for each bit of data to be transmitted. The *shift* signal is enabled on each transmit clock to advance the frame sync data out of the shift register. The select for the Output Mux is set to choose non error corrected data. The select for the Input Mux is set to choose the most significant byte of the frame sync from the RF Xilinx register file. The state machine stays in *State "3"* until the *EndOfByte* signal is asserted by the count8 counter. When the *EndOfByte* signal is asserted, the most significant byte of the frame sync is again loaded into the Input Data Shift Register. The state machine advances to *State "4"*.

The state machine begins in *State "4"* by restarting the count15 and block counters. Again, these counters are not used because FEC is not enabled. The state machine generates a transmit request to the radio for each bit to be transmitted. The *shift* is enabled on each transmit clock to advance the frame sync data out of the Input Data Shift Register. The state machine stays in *State "4"* until the *EndOfByte* signal is asserted by the count8 counter. The select for the Output Mux is set to choose non error corrected data. The select for the Input Mux is set to load the least significant byte of the frame sync pattern after the current byte is finished processing. As soon as the *EndOfByte* signal is asserted, the least

significant byte of the frame sync is loaded into the Input Data Shift register, and the state machine advances to State "5".

In *State "5"*, the state machine operates exactly as in *State "3"*. The least significant byte of the frame sync is again transmitted, and the most significant byte is latched after the *EndofByte* signal is asserted (for processing during the next state). The state machine then proceeds to *State "6"*.

In *State "6"*, the state machine operates similarly to the operation in *State "4"*. The most significant byte of the frame sync is again transmitted. This state completes the frame sync processing, and the first byte of input data is now latched into the Input Data Shift Register after the *EndofByte* signal is asserted (for processing during the next state). Since this state marks the beginning of the packet data, the CRC is initialized to a cleared state. The transmit strobe is also asserted to advise the TX Chip that the RF Xilinx is ready for the second byte of packet data, having already latched the present byte. The state machine then progesses to *State "7"*.

No FEC, State "7"

State "7" is a data processing state for data with disabled FEC. However, it is also a disembarkation state that is used for one clock after the processing of the frame sync is completed, regardless of the FEC level. When a transition from state "6" to state "7" occurs, the first byte of data is shifted serially out of the Input Data Shift Register. It is important to recognize that processing of the different levels of FEC occur simultaneously, and the only difference between choosing the FEC level is the select value present at the Output Mux and the differing control signals from this state machine. Hence, it is understandable how all levels of error correction share this state for the one cycle after the frame sync processing is completed.

This state begins like previous states, by performing the requisite shifts from the Input Data Shift Register, setting the Input Mux select to data for the next byte's processing, and setting the Output Mux select to non error corrected data. A transmit request is similarly generated for each bit of data to be transmitted. When the *EndofByte* signal is

asserted, the next byte of data from the TX chip is loaded into the input shift register, if there are remaining bytes to be transmitted. Otherwise the signals that indicate that more data is present are cleared.

If the end of an encoding block occurs, i.e. the count15 counter has counted down from 15 to 1, then the state machine will advance to the state corresponding to the requested error correction level as determined by the *ECCSelect* signal. If no error correction is desired, then the next state remains at *State "7"*. However, if maximum (15,5,3) error correction is desired, the state advances to *State "9"*, while the state advances to *State "8"* for medium level (15,11,1) error correction. If no data is remaining, i.e. the TX chip has not provided the next byte of data, then the state machine returns to the *Idle* state.

Medium Level FEC, State "8"

In State "8", the state machine shifts the data out of the Input Data Shift register and selects non error corrected output data for the Output Mux until the *End11bit* signal is asserted by the count15 down counter. At this point, the Output Mux selects medium error corrected data, and this data begins to be transmitted. The transmit request is still continually generated for the transmit radio for each bit to be transmitted. When the end of a byte of input data occurs, the RF Xilinx loads the next byte of data and a transmit strobe (TXS) is generated for the TX Chip to indicate that the RF Xilinx is ready for another byte of data, if additional data is present to transmit. If the TX Chip does not require further data to be transmitted, i.e TXP is unasserted, then the signals indicating that additional data is remaining are cleared.

When the *EndEncoding* signal is asserted by the count15 counter, then the error correction level is re-evaluated if there is additional transmit data to process. If no error correction is desired, then the next state transitions to *"State "7"*. However, if maximum (15,5,3) error correction is desired, the state advances to *"State "9"*, while the state remains in *"State "8"* for medium level (15,11,1) error correction. If no data is remaining, i.e. the TX Chip has not provided the next byte of data, then the state machine returns to the *Idle* state.

Maximum Level FEC, State "9"

In State "9", the state machine shifts the data out of the input shift register and selects non error corrected output data for the Output Mux while the *End5bit* signal is unasserted by the count15 down counter. When the *End5bit* signal is asserted, the Output Mux selects maximum error corrected data, and this data begins to be transmitted. The transmit request is still continually generated for the transmit radio for each bit to be transmitted. When the end of a byte of input data occurs, the RF Xilinx loads the next byte of data and a transmit strobe (TXS) is generated for the TX Chip to indicate that the RF Xilinx is ready for another byte of data, if additional data is present to transmit. If the TX Chip does not require further data to be transmitted, i.e TXP is unasserted, then the signals indicating additional data is remaining are cleared.

When the *EndEncoding* signal is asserted by the count15 counter, then the error correction level is re-evaluated if there is additional transmit data to process. If no error correction is desired, then the next state transitions to *"State "7"*. However, if maximum (15,5,3) error correction is desired, the state remains in *"State "9"*, while the state advances to *"State "8"* for medium level (15,11,1) error correction. If no data is remaining, i.e. the TXchip has not provided the next byte of data, then the state machine returns to the *Idle* state.

Control Counters

The Transmit Control has several counters that help to implement the control of the transmit data path. These are the count8, the count15, and the block counters. Each counter is implemented as a state machine in VHDL and is described separately below.

Count8

The count8 counter is a 2 bit up-counter that is initialized to 0 by the control logic in the Main Control state machine. The counter increments whenever the *shift* signal (also from the Main Control state machine) is asserted, and a flip flop latches the counter value at the rising edge of the transmit clock. When the counter reaches 7, an *EndOfByte* signal is generated, and the counter wraps around to 0.

Count15

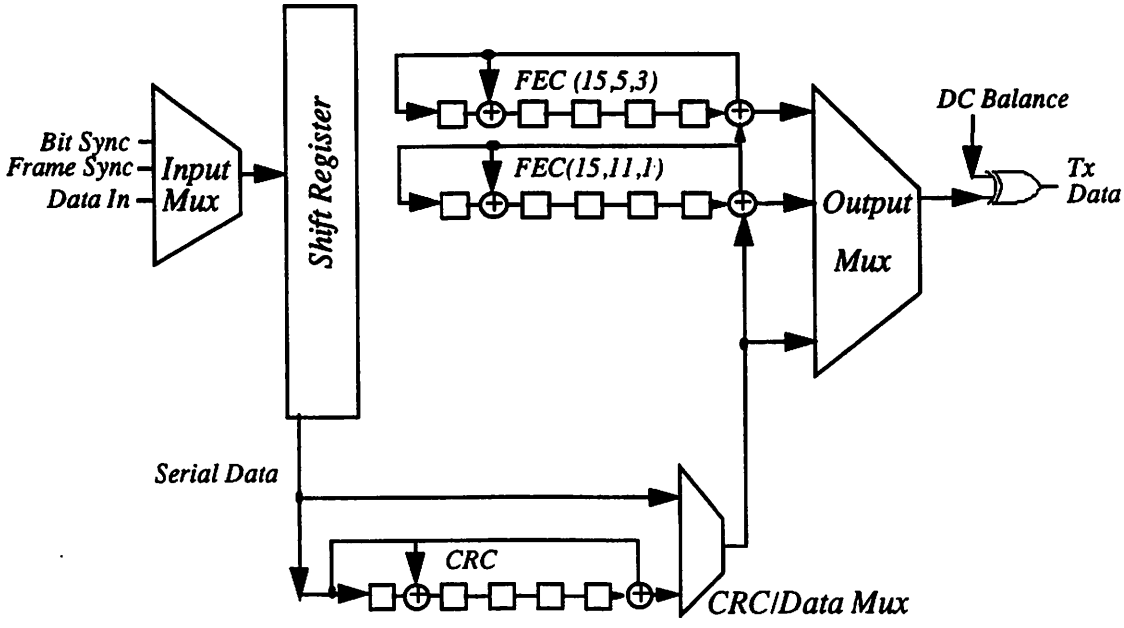
The count15 counter is a 3 bit down counter that is initialized to 14 by the control logic in the Main Control state machine. The count15 counter is necessary because the encoded data is output in 15 bit blocks. The counter is decremented every transmit clock, and a flip flop latches this value. Before each decrement, however, a line balance signal is set within this state machine. The line balance pattern is ultimately exor'ed with the output data in an attempt to help DC balance the transmitted data signal. Several other control signals are also determined by the count15 counter. After the count15 has counted down four states, the *End4bit* signal is generated. Similarly, after the count15 has counted down to the requisite state, the *End5bit*, *End10bit*,and *End11bit* signals are asserted. These signals help synchronize when to enable and disable encoding. When the counter reaches the value of 0, the counter generates the *EndEncoding* signal and wraps around to begin at 14 again.

Block Counter

The block counter is used to determine when to switch from the header encoding level to the encoding level prescribed by the TXECC signal. The block counter is an up-counter that counts from 0 to 15. The counter increments whenever the *EndEncoding* signal is generated. When the counter reaches 15, it does not wrap back around to 0. Instead it stays locked at 15, unless the counter is reinitialized by the Main Control state machine.

3.1.4.2. Transmit Data Path

Figure 3-7. High Level View Of Transmit Data Path



Input Mux

The input mux is used to select the 1 byte input to the Input Data Shift register. The choices include the bit sync, the MSB of the frame sync, the LSB of the frame sync, and the input transmit data.

CRC/Serial Data Mux and CRC Control

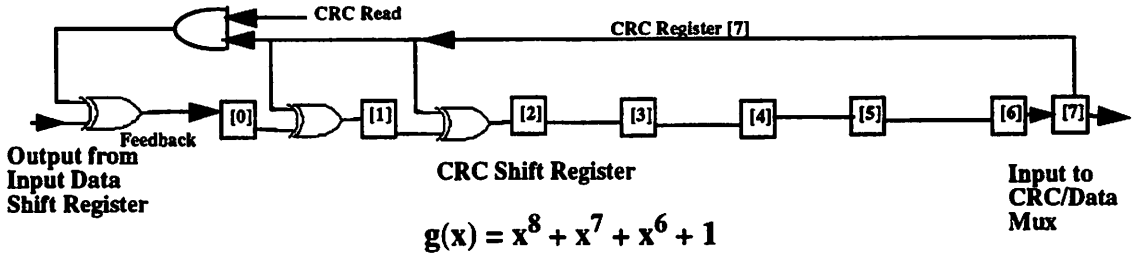
When the TXCRC line is pulsed high by the TX Chip, the RF Xilinx begins the transmission of the CRC instead of the serial input data. The RF Xilinx selects the CRC input of the CRC/Serial Data Mux for 8 cycles after the initial raising of the TXCRC signal. During the transmission, calculation of the CRC is suspended, i.e. feedback is set equal to zero. After 8 transmit clock cycles have passed, the CRC Shift Register is cleared, and the CRC/Serial Data Mux is set to pass the data from the Input Data Shift Register. The TX

Chip only raises the TXCRC signal for the header and body CRCs if the data type to be transmitted is configured to include CRCs.

CRC Algorithm

When the CRC Clear signal is asserted by the *Main Control* state machine, the CRC Shift Register is cleared to a value of 0. The CRC is implemented with an 8-bit shift register. The feedback signal for the CRC is generated by exor'ing the MSB of the CRC shift register with the input serial data, provided the CRC is not being transmitted. If the CRC value is being transmitted, the CRC feedback signal is disabled. The CRC is calculated by writing the feedback signal to the LSB of the shift register, writing the exor of the LSB and the feedback signal to the second bit in the shift register, and writing the exor of the feedback and the second bit of the shift register to the third bit of the shift register. Each time a new bit of data is shifted out of the input shift register, the CRC shift register updates its values. The MSB of the CRC shift register is the serial output of the CRC data stream.

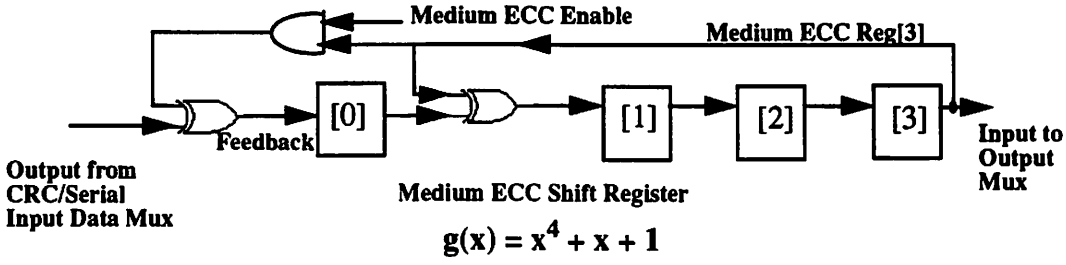
Figure 3-8. CRC



Medium ECC Encoder

The medium ECC encoder is implemented with a 4 bit shift register. The output of the CRC/Data Mux is exor'ed with the most significant bit of this shift register to generate the encoder feedback. The encoder feedback signal is only valid while the *End11bit* signal is zero, i.e. after count15 counts off the 11 bits of input data in an encoding block, the feedback is disabled. At the start and end of the encoding block, the Medium ECC Shift Register is cleared to zero. Otherwise, the encoder shifts in the feedback signal to displace the least significant bit (LSB) of the Medium ECC Shift Register. The feedback signal also is exor'ed with the LSB of the Medium ECC Shift Register and to constitute the second bit in the Medium ECC Shift Register.

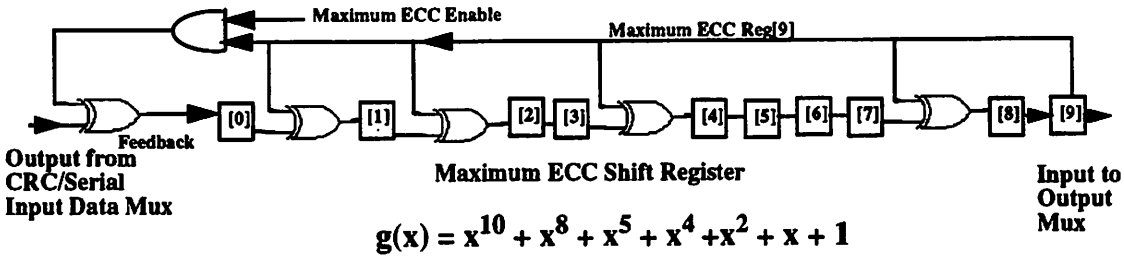
Figure 3-9. Medium ECC Encoder



Maximum ECC Encoder

The maximum ECC encoder is implemented with a 10 bit shift register. The output of the CRC/Data Mux is exor'ed with the most significant bit of this shift register to generate the encoder feedback. The encoder feedback signal is only valid while the *End5bit* signal is zero, i.e. after count15 counts off the 5 bits of input data in an encoding block, the feedback is disabled. At the start and end of the encoding block, the Maximum ECC Shift Register is cleared to zero. Otherwise, the encoder shifts in the feedback signal to displace the least significant bit (LSB) of the Maximum ECC Shift Register. The feedback signal also is exor'ed with the bits [0:1], [3], and [7] of the Maximum ECC Shift Register in order to generate the subsequent bits [1:2], [4], and [8] of the Maximum ECC Shift Register.

Figure 3-10. Maximum ECC Encoder



ECC Selection

The *EccReg* register contains the value for the error correction level. This value normally is latched from the TXECC input; however, the first eight bytes of the data payload (corresponding to the header) have a fixed forward error correction that is set to the maximum level of FEC.

Output Mux

The Output Mux chooses between the maximally error corrected data stream, the medium error corrected data stream, and the data stream without error correction based on the setting in the *EccReg* register. This selection becomes the serial output transmit data stream.

DC Balance

The output transmit data stream is exor'ed with the line balance pattern generated by the count15 counter. This pattern is used to aid in DC balancing the transmit data.

3.1.5. Transmit Interface for the Plessey Radio

Overview

The basestation uses the Plessey radio as the transmitter radio. In order to use the Plessey radio in transmit mode, it must first be programmed to a particular transmit frequency whose value is contained in the RF Xilinx register file. After the transmit frequency is programmed, the Plessey waits until a transmit request arrives from the RF Xilinx. When the transmit request arrives, the radio switches from a low power mode to a higher powered transmit mode and begins transmitting data until the transmit request line is driven low again by the RF Xilinx. The Plessey Transmit Interface block is responsible for this programming and implements the appropriate delays required.

The interface to the Plessey radio is designed in VHDL (*plessey_x.vhd*) with a schematic top level in Viewdraw (*txradio_plessey.1*). The VHDL design was implemented with a state machine that contains seven states, as illustrated below in Figure 3-11. In addition to the state machine, the VHDL design contains a 7-bit counter that is used to implement the delays needed when programming the Plessey radio. The VHDL design for the Plessey interface also drives the antenna select and the radio standby signals. The schematic top level for the Plessey transmit interface for the basestation instantiates the VHDL design and routes the outputs of the design to the pins on the RF Xilinx that control the Plessey radio. The transmit serial data line is also brought into the schematic from the Transmit Control block and is passed out to the serial data transmit pin for the Plessey. The transmit clock from the mult-timer module is also buffered in the Plessey top level schematic with a global clock buffer before it is used to drive the VHDL design.

Table 3-4. I/O for the Viewlogic Schematic of the Plessey Interface on the Basestation

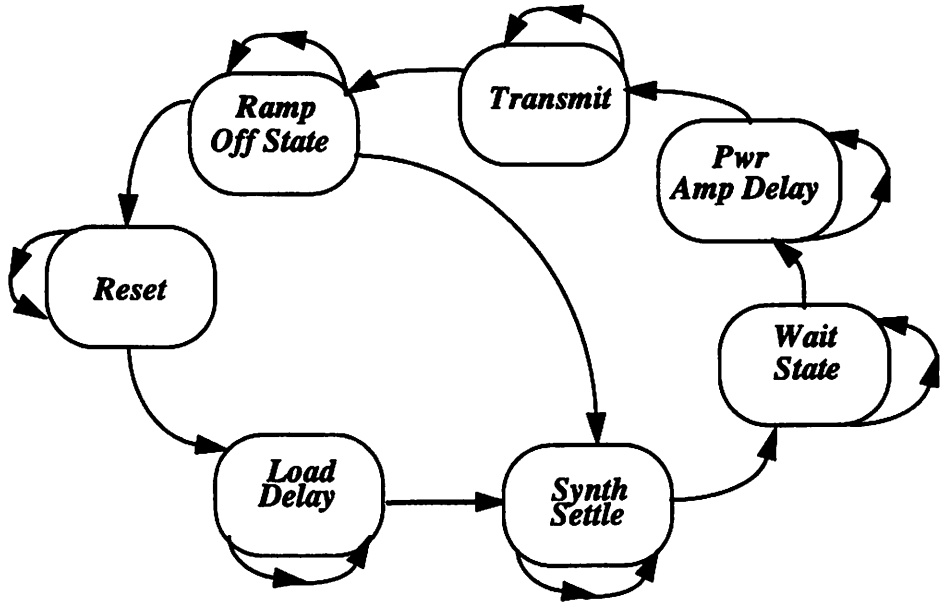
Signal Name	I/O	Description
TX_CLKIN	I	Transmit Clock from Multi-timer block
RESET	I	Internal System Reset. Controlled by RF Xilinx register file.
HOP_RESET	I	Hop Reset. Controlled by RF Xilinx register file.
TX_REQUEST	I	Maps to REQUEST in the VHDL block.
TX_FREQ[15:7]	I	Not used.
TXFREQ[6:0]	I	Maps to *channel_in* in the VHDL block.
TX_DOUT	I	Serial Transmit Data. Originates in Transmit Control Block and is routed to the Plessey Transmit Data output pin on the RF Xilinx.
TXCNTL[7:0]	I	Not used.
PL_ANTSEL	O	Output to the Plessey Radio. Antenna Select. Maps to *antsel* in Plessey Transmit Controller VHDL block.
PL_POW_LEVEL	O	Output to the Plessey Radio. Power Level Select. Maps to *power_level* in Plessey Transmit Controller VHDL block.
PL_TX_RX	O	Output to the Plessey Radio. Transmit/Receive Mode Select. Maps to *tx_rx* in Plessey Transmit Controller VHDL block.
PL_PA_OFF	O	Output to the Plessey Radio. Power Amplifier On/Off Select. Maps to *pa_off* in Plessey Transmit Controller VHDL block.
PL_STDBY	O	Output to the Plessey Radio. Standby. Maps to *stdby* in Plessey Transmit Controller VHDL block.
PL_LOADB	O	Output to the Plessey Radio. Active Low Load Line. Maps to *loadb* in Plessey Transmit Controller VHDL block.
PL_SD[6:0]	O	Output to the Plessey Radio. Input Data Bus for Programming the Plessey Radio. Maps to *channel_out* in Plessey Transmit Controller VHDL block.
PL_TXD	O	Serial Transmit Data to the Plessey Radio. From the Transmit Control Block.
TX_CLKOUT	O	Buffered TX_CLKIN. Used to drive Plessey Transmit VHDL design and the Transmit Control Block.
TX_GRANT	O	Maps to GRANT in the VHDL block.

Table 3-5. I/O for the VHDL Design of the Plessey Interface on the Basestation

VHDL Signal Name	I/O	Description
clk	I	Input Clock to the Plessey Radio Interface on the Basestation. Driven by a buffered transmit clock from the multi-time module.
RESET	I	Internal System Reset. Controlled by RF Xilinx register file (XILCNTL[7]).
HOP_RESET	I	Hop Reset. Controlled by RF Xilinx register file.
REQUEST	I	Transmit Request. Controlled by RF Xilinx Transmit Control Block.
xmit_pow_level	I	Transmit Power Level. Output directly to the output pin on the RF Xilinx. 0-Low Power Mode (Receive/Idle) 1-High Power Mode (Transmit)
channel_in[6:0]	I	Transmit Frequency for Plessey Radio. From the RF Xilinx register file.
channel_out[6:0]	O	Transmit Frequency for Plessey Radio. Latched from *channel_in*. Output directly to the output pin on the RF Xilinx using the SD[6:0] bus described in the Viewlogic schematic.
loadb	O	Active low load line used to load the Plessey Radio with the Transmit Frequency. Output directly to the output pin on the RF Xilinx.
stdby	O	Standby signal for the Plessey Transmit Radio. The radio should always be on standby when this interface is used, hence this value should always be set to one. Output directly to the output pin on the RF Xilinx.
pa_off	O	Power Amplifier on/off Select. When this signal is zero, the power amplifier on the Plessey is turned off. Conversely, when this signal is one, the power amplifier is turned on in preparation for transmission. Output directly to the output pin on the RF Xilinx.
tx_rx	O	Receive/Transmit mode select for the Plessey. Should be set to Transmit for this interface block. Output directly to the output pin on the RF Xilinx. 0 - Transmit 1 - Receive
power_level	O	Power Level to the Plessey Radio. Output directly to the output pin on the RF Xilinx. 0 - Low Power Mode (Idle Mode) 1 - High Power (Transmit Mode)
antsel	O	Antenna Select. Originally intended to help provide antenna diversity, this signal is statically driven to a value of zero, corresponding to a choice of Antenna 1. Any future work on antenna diversity would dynamically control this signal. Output directly to the output pin on the RF Xilinx.
GRANT	O	Transmit Grant. This signal is an input to the Transmit Control block. It is asserted by the Plessey Interface while the radios are in transmit state, i.e. are in the process of actually transmitting the data.

State Machine Description

Figure 3-11. State Diagram for Plessey Transmit Controller



Reset

In the *Reset* state, the state machine waits for two consecutive resets. These resets are driven by the ARM kernel in the *iprf.c* file. The first reset can be either a hop reset or a system reset, and the second reset in the series must be a hop reset. When this condition occurs, the state machine clears the delay counter and advances to the *Load Delay* State.

Load Delay

In the *Load Delay* state, the state machine waits until the delay counter reaches the value specified in order to allow the proper time elapse after reset. In this design, the load delay value is 2 cycles of a clock with a frequency of approximately 1 MHz (using the transmit clock from the multi-timer). The load delay is approximately 2 μs. When the counter reaches this value, it is resets the delay counter to zero, lowers the active low load line, and outputs the desired transmit frequency to the data input bus of the radio (*channel_out*[6:0] internal to the interface block, SD[6:0] in the higher level schematic). The state machine also clears the internal reset signal that was used to detect the first reset in the *Reset* Stare. The state machine then advances to the *Synth Settle* State.

Synth Settle

In the *Synth Settle* State, the active low load line is raised, and the radio is allowed to settle for approximately 80 μs. This delay is counted by using the delay counter, in the same manner as it was used in the *Load Delay* State. The settling time is 85 cycles in the present design. In this state, the radio is in low power mode, the power amplifier is turned off on the radio, and the transmit is disabled. When the settling time has elapsed, the delay counter is reset to zero, and the state machine advances to the *Wait* State.

Wait State

In the *Wait* State, the state machine waits for either a reset (system or hop) or a request to transmit from the Transmit Control Module of the RF Xilinx. The transmit power mode for the radio is switched from low power to high power in this state, i.e. The one bit output signal *pwr_level* is driven to the value of the input signal *xmit_pwr_level.* The *xmit_pwr_level* signal is driven externally by the 5 V power supply, VDD. If a reset condition occurs, the state machine will return to the *Reset* state. However, if a transmit condition occurs, i.e. the Transmit Control Module raises the TX_REQUEST line, then the state machine advances to the *Power Amp Delay* state.

Power Amp Delay

In the *Power Amp Delay* State, the state machine enables the transmitter by selecting the transmit mode of the radio (as opposed to the receive mode). This is accomplished by writing a zero to the *tx_rx* output signal. The state machine then delays the transmission of data by 2 μs. This delay is implemented by using the delay timer to count the value for the Power Amp Delay. The value used in the design is 2, since each cycle is approximately 1 μs. When the counter reaches the Power Amp Delay value, the delay counter is cleared, and the state machine advances to the *Transmit* state.

Transmit

In the transmit state, the power amplifier in the radio is turned on by writing a one to the *pa_off* output signal. The transmitter stays enabled and the GRANT output is driven

high while in this state. The state machine stays in this state until the REQUEST signal is lowered. When the REQUEST signal is driven low, the state machine advances to the *Ramp Off* state.

Ramp Off

In this state, the GRANT signal is driven low, and the delay counter counts 5 cycles in order to turn the power amplifier in the radio off and wait for the decay. If the hop reset is asserted, the state machine returns to the *Reset* state, otherwise the state machine returns to the *Synth Settle* State. When in the *Synth Settle* state, the radio is switched back to a low power mode.

3.1.6. Transmit Interface for the Proxim Radio

The Transmit interface for the Proxim radio on the pad is considerably simpler than the Plessey's transmit interface. The loading of the Proxim's configuration registers is performed serially and is under software control in the ARM kernel, *iprf.c*. The kernel programs the RF Xilinx registers that control the load signal, the data strobe, the serial configuration data, the receive/transmit mode select, the standby signal, and the ready signal for the Proxim. The kernel also handles the timing requirement for the programming of the Proxim radio. The Proxim radio generates its own transmit clock, so on the pad the transmit clock from the multi-timer module is not used.

In order to write to the Proxim's registers, the ARM kernel must serialize the data to be written. The Proxim must be initialized while not in standby mode. Standby (TX_FREQ[4]) is a low power, reset mode for the Proxim. While the Proxim is not in standby, the kernel then takes each bit of the serial data and writes it to the serial configuration data register of the RF Xilinx (TXFREQ[2]). Then two writes are performed to toggle the data strobe (TXFREQ[1]). When the entire word is strobed into the Proxim, the *load* signal (TXFREQ[0]) is asserted. This procedure of strobing in serial data and then asserting *load* is continued until all registers in the Proxim have been programmed. In addition to programming the Proxim with serial data, the receive/transmit mode select (TXFREQ[3]) signal must be set to transmit mode. After the Proxim has been programmed, the ready signal (TXFREQ[5]) should be asserted. When the ready signal is high

and a transmit request (TX_REQUEST) arrives from the Transmit Control Block, the TX_GRANT line is asserted.

The Proxim Transmit Radio interface for the Pad is specified entirely by a Viewlogic schematic (*txradio_proxim.1*), and the I/O for the schematic block can be seen below.

Table 3-6. I/O for the Viewlogic Schematic of the Proxim Interface on the Pad

Signal Name	I/O	Description
TX_CLKIN	I	Not used.
RESET	I	Not used.
HOP_RESET	I	Hop Reset. Controlled by RF Xilinx register file.
TX_REQUEST	I	Originates from the Transmit Control Block. ANDed with the Proxim_Ready signal to generate TX_GRANT output.
TX_FREQ[15:6]	I	Not used.
TXFREQ[5]	I	Proxim Ready signal. Should be asserted after the transmit frequency has been loaded into the Proxim.
TXFREQ[4]	I	Proxim Standby. Active low. When this signal is low, all digital electronics are reset, and the radio is put into its low power state.
TXFREQ[3]	I	Proxim TX/RX select. When this signal is high, the radio's transmitter is enabled.
TXFREQ[2]	I	Proxim Serial Data.
TXFREQ[1]	I	Proxim Serial Data Strobe
TXFREQ[0]	I	Proxim Sync Load. When this signal is high, a write of the accumulated serial data in the radio is performed to the internal registers of the Proxim.
TX_DOUT	I	Serial Transmit Data. Originates in Transmit Control Block and is routed to the Proxim Transmit Data output pin on the RF Xilinx.
TXCNTL[7:0]	I	Not used.
TX_CLKOUT	O	Transmit Clock for the Transmit Control Block. Driven by an external pin by the Proxim's Transmit clock.
TX_GRANT	O	Asserted when the Proxim is Ready and TX_REQUEST is high.

3.1.7. Receive Interface for the Plessey Radio

Overview

The pad uses the Plessey radio as the receive radio. In order to use the Plessey radio in receive mode, it must first be programmed to a particular receive frequency whose value is contained in the RF Xilinx register file. After the receive frequency is programmed, the Plessey enters a receive mode that terminates only when a hop reset occurs and a packet is

not being received. The Plessey Transmit Interface block is responsible for the programming of the Plessey and implements the appropriate delays required.

The interface to the Plessey radio is designed in VHDL (*plessey_r.vhd*) with a schematic top level in Viewdraw (*rxradio_plessey.1*). The VHDL design was implemented with a state machine that contains four states, as illustrated below in Figure 3-12. In addition to the state machine, the VHDL design contains a 7-bit counter that is used to implement the delays needed when programming the Plessey radio. The VHDL design for the Plessey interface also drives the antenna select, receive/transmit mode select, standby, and low/high power mode select. The schematic top level for the Plessey receive interface for the pad instantiates the VHDL design and routes the outputs of the design to the pins on the RF Xilinx that control the Plessey radio. The receive serial data line from the Plessey is output from the schematic to the Receive Control block. The receive clock from the mult-timer module is also buffered in the Plessey top level schematic with a global clock buffer before it is used to drive the VHDL design.

In addition to the Plessey Receive Interface VHDL design, the top level schematic of the Plessey Receive Block instantiates two other VHDL designs, the RSSI controller and the Clock Recovery module. The RSSI controller takes the serial input from the Plessey receive radio, and with the help of an A/D converter on the board, outputs the RSSI measurement for the receive data packet. The RSSI system is used for both Plessey and Proxim designs and is discussed in detail in Section 3.1.9. on page 69.

Since the Plessey does not have its own clock recovery circuitry, a clock recovery system had to be implemented. Serial data is received from the Plessey radio and is directly input to this system. The recovery algorithm uses digital interpolation to determine the optimal sampling point. It also includes a closed-loop update that allows tracking of small frequencies. The clock recovery algorithm will be discussed later in this section.

Table 3-7. I/O for the Viewlogic Schematic of the Plessey Interface on the Pad

Signal Name	I/O	Description
RXCLK_IN	I	Receive Clock from Multi-timer block
RESET	I	Internal System Reset. Controlled by RF Xilinx register file. Maps to reset in the Plessey Receive Controller design.
HOP_RESET	I	Hop Reset. Controlled by RF Xilinx register file (XILCNTL[7]). Maps to HOP_RESET in the Plessey Receive Controller VHDL design.
RX_FREQ[15:7]	I	Not used.
RXFREQ[6:0]	I	Maps to *channel_in* in the Plessey Receive Controller VHDL design.
RXCNTL[7:0]	I	Not used.
RXP	I	Received packet signal. This signal is generated by the Receive Control Block and set high whenever a frame sync matches on the incoming received data. RXP stays high during the duration of the reception of the packet, when it is then set low. Maps to RXP of the Plessey Receive Controller VHDL design.
RA17	I	Serial RSSI Data Input from the Plessey Radio via the A/D Converter
CR_IN	I	Serial Receive Data Input from the Plessey Radio.
PL_ANTSEL	O	Output to the Plessey Radio. Antenna Select. Maps to *antsel* in Plessey Receive Controller VHDL block.
PL_POW_LEVEL	O	Output to the Plessey Radio. Power Level Select. Maps to *power_level* in Plessey Receive Controller VHDL block.
PL_TX_RX	O	Output to the Plessey Radio. Transmit/Receive Mode Select. Maps to *tx_rx* in Plessey Receive Controller VHDL block.
PL_PA_OFF	O	Output to the Plessey Radio. Power Amplifier On/Off Select. Maps to *pa_off* in Plessey Receive Controller VHDL block.
PL_STDBY	O	Output to the Plessey Radio. Standby. Maps to *stdby* in Plessey Receive Controller VHDL block.
PL_LOADB	O	Output to the Plessey Radio. Active Low Load Line. Maps to *loadb* in Plessey Receive Controller VHDL block.
PL_SD[6:0]	O	Output to the Plessey Radio. Input Data Bus for Programming the Plessey Radio. Maps to *channel_out* in Plessey Receive Controller VHDL block.
RSSICLK	O	Output Clock for the A/D Converter. Used to Strobe in the converted RSSI Measurement Data. Maps to *ser_clk* of RSSI Controller Block.
RSSICNVT	O	Active low Convert Signal that issues a start of conversion command to the A/D converter. Maps to *convt_l* in the RSSI Controller Block.
RX_READY	O	Maps to ready in the Plessey Receive Controller VHDL block.
RSSI_VALID	O	Active high signal indicating that the value in the RSSI register is the current RSSI measurement. Used to generate an RSSI interrupt in the IRQ module.Generated by the RSSI Controller VHDL design.
RSSI[11:0]	O	RSSI measurement data. Generated by the RSSI Controller VHDL design and output to the ARM interface module of the RF Xilinx.
RX_DIN	O	Serial Receive Data. Originates from Plessey radio input to RF Xilinx. with a destination of the Receive Control Block.

RXCLK_BUF	O	Buffered RXCLK_IN
RXCLK_OUT	O	Clock used to drive the Receive Control Block. Derived from feeding RXCLK_IN through the Clock Recovery Block.
RX_READY	O	Maps to ready in the Plessey Receive Controller VHDL block.
RSSI_VALID	O	Active high signal indicating that the value in the RSSI register is the current RSSI measurement. Used to generate an RSSI interrupt in the IRQ module.Generated by the RSSI Controller VHDL design.
RSSI[11:0]	O	RSSI measurement data. Generated by the RSSI Controller VHDL design and output to the ARM interface module of the RF Xilinx.
RX_DIN	O	Serial Receive Data. Originates from Plessey radio input to RF Xilinx. with a destination of the Receive Control Block.

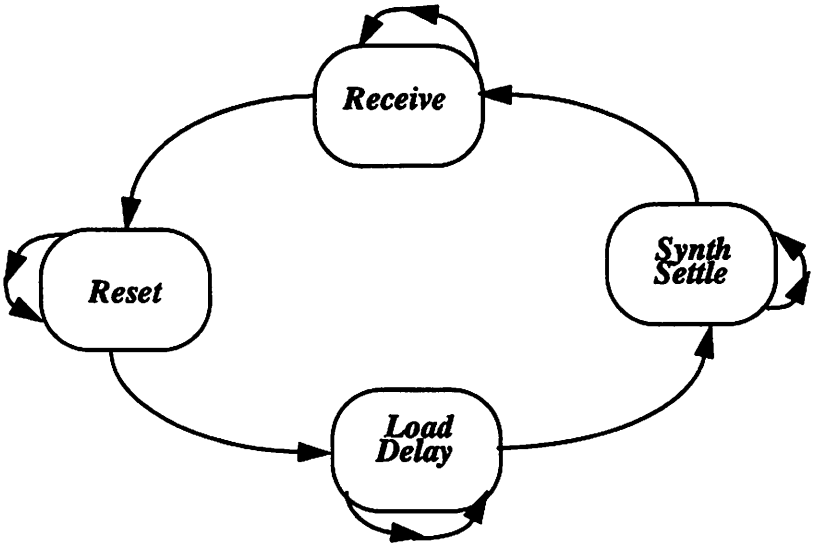
Table 3-8. I/O for the VHDL Design of the Plessey Radio Interface on the Pad

VHDL Signal Name	I/O	Description
clk	I	Input Clock to the Plessey Radio Interface on the Basestation. Driven by a buffered receive clock from the multi-time module.
reset	I	Internal System Reset. Controlled by RF Xilinx register file.
HOP_RESET	I	Hop Reset. Controlled by RF Xilinx register file.
RXP	I	Received packet signal. This signal is generated by the Receive Control Block and set high whenever a frame sync matches on the incoming received data. RXP stays high during the duration of the reception of the packet, when it is then set low.
stdby	O	Standby signal for the Plessey Receive Radio. The radio should always be on standby when this interface is used, hence this value should always be set to one. Output directly to the output pin on the RF Xilinx.
pa_off	O	Power Amplifier on/off Select. When this signal is zero, the power amplifier on the Plessey is turned off. Conversely, when this signal is one, the power amplifier is turned on in preparation for transmission. Output directly to the output pin on the RF Xilinx.
tx_rx	O	Receive/Transmit mode select for the Plessey. Should be set to Receive for this interface block. Output directly to the output pin on the RF Xilinx. 0 - Transmit 1 - Receive
power_level	O	Power Level to the Plessey Radio. Output directly to the output pin on the RF Xilinx. 0 - Low Power Mode (Idle/Receive Mode) 1 - High Power (Transmit Mode)
antsel	O	Antenna Select. Originally intended to help provide antenna diversity, this signal is statically driven to a value of zero, corresponding to a choice of Antenna 1. Any future work on antenna diversity would dynamically control this signal. Output directly to the output pin on the RF Xilinx.
ready	O	Active high signal that indicates the Plessey radio is ready to receive data.

channel_in[6:0]	I	Receive Frequency for Plessey Radio. From the RF Xilinx register file.
channel_out[6:0]	O	Receive Frequency for Plessey Radio. Latched from *channel_in*. Output directly to the output pin on the RF Xilinx using the SD[6:0] bus described in the View-logic schematic.
loadb	O	Active low load line used to load the Plessey Radio with the Receive Frequency. Output directly to the output pin on the RF Xilinx.
stdby	O	Standby signal for the Plessey Receive Radio. The radio should always be on standby when this interface is used, hence this value should always be set to one. Output directly to the output pin on the RF Xilinx.
pa_off	O	Power Amplifier on/off Select. When this signal is zero, the power amplifier on the Plessey is turned off. Conversely, when this signal is one, the power amplifier is turned on in preparation for transmission. Output directly to the output pin on the RF Xilinx.
tx_rx	O	Receive/Transmit mode select for the Plessey. Should be set to Receive for this interface block. Output directly to the output pin on the RF Xilinx. 0 - Transmit 1 - Receive
power_level	O	Power Level to the Plessey Radio. Output directly to the output pin on the RF Xilinx. 0 - Low Power Mode (Idle/Receive Mode) 1 - High Power (Transmit Mode)
antsel	O	Antenna Select. Originally intended to help provide antenna diversity, this signal is statically driven to a value of zero, corresponding to a choice of Antenna 1. Any future work on antenna diversity would dynamically control this signal. Output directly to the output pin on the RF Xilinx.
ready	O	Active high signal that indicates the Plessey radio is ready to receive data.

State Machine Description for Plessey Receive Controller

Figure 3-12. State Diagram for Plessey Receive Controller



Reset

In the *Reset* state, the state machine waits for two consecutive resets. These resets are driven by the ARM kernel in the *iprf.c* file. The first reset can be either a hop reset or a system reset, and the second reset in the series must be a hop reset. When this condition occurs, the state machine clears the delay counter and advances to the *Load Delay* State.

Load Delay

In the *Load Delay* state, the state machine waits until the delay counter reaches the value specified in order to allow the proper time elapse after reset. In this design, the load delay value is 2 cycles of a clock with a frequency of approximately 1 MHz (using the receive clock from the multi-timer). The load delay is approximately 2 μs. When the counter reaches this value, it is resets the delay counter to zero, lowers the active low load line, and outputs the desired receive frequency to the data input bus of the radio (*channel_out*[6:0] inside the interface block, SD[6:0] in the higher level schematic). The state machine also clears the internal reset signal that was used to detect the first reset in the *Reset* Stare. The state machine then advances to the *Synth Settle* State.

Synth Settle

In the *Synth Settle* State, the active low load line is raised, and the radio is allowed to settle for approximately 80 μs. This delay is counted by using the delay counter, in the same manner as it was used in the *Load Delay* State. The settling time is 85 cycles in the present design. In this state, the radio is in low power mode, the power amplifier is turned off on the radio, and the transmit is disabled. When the settling time has elapsed, the delay counter is reset to zero, and the state machine advances to the *Receive* State.

Receive

In the *Receive* state, the *ready* line is pulled high. The Plessey radio stays in this receive mode until a hop reset occurs and a latched version of RXP (*start_rcv*) goes low, i.e. reception of the packet has completed. When this occurs, the state machine advances to the *Reset* State.

Table 3-9. I/O for the VHDL Design of the Clock Recovery Module

VHDL Signal Name	I/O	Description
clk	I	Input Clock to the Plessey Radio Interface on the Basestation. Driven by a buffered receive clock from the multi-time module.
reset	I	Internal System Reset. Controlled by RF Xilinx register file.
data_in	I	Serial input data from the Plessey Receive Radio
rxsync_enb	I	Receive Sync Enable. Maps to RXP in the schematic. Clock recovery is only enabled when a packet is being received.
clock_out	O	Recovered Received Clock. Passed to Receive Control Block.
data_out	O	Recovered Serial Receive Data. Passed to Receive Control Block.

Clock Recovery Module

The clock recovery module is necessary only for the Plessey receive radio, because the Proxim provides its own clock recovery. The design was performed in VHDL and can be found in *clk_recovery.vhd*. This algorithm is designed to be tolerant to spurious noise, as well as frequency offsets that can occur when the transmit and receive frequencies drift. This drift is due to limited precision in the oscillators during receipt of long packets.

A free running phase counter (*phase_c*) is started in order to provide a reference point for the phase0 mark used in the clock recovery. The input data stream *data_in* is first oversampled 10 times by a sample clock with a 10% duty cycle. The oversampled data is shifted into the 10-bit wide *data_reg* shift register. Another register called *clock* is also 10-bits wide, and it stores a 1 in the location of the bit that should be sampled in the *data_reg* shift register.

An accumulator (*sum*) is used to obtain data for the clock recovery decision. Data in the accumulator is changed if the data_in value and the data point ten samples ago are different. Whenever a 1 is shifted into the *data_reg* shift register, and a 0 is shifted out, the *sum* register is incremented. Likewise, if a 0 is shifted in, and a 1 is shifted out, *sum* is decremented. If a 1 is shifted in and out (or a 0 is shifted in and out), then the sum is unchanged. In this fashion, the integral of the samples can be obtained.

Before reception of a packet occurs, a new sampling point is determined (i.e. the bit in the *data_in* shift register that corresponds to the output data) whenever the accumulator reaches a value greater than the *largest* value provided the largest value of the accumulator is at least 6. (The maximum value the accumulator can reach is 10.) Whenever this condition occurs, the value in the accumulator (*sum*) is latched to become the new *largest* value, and the sample phase is latched from the phase counter (*phase_c*).

If the data stream becomes a stream of zeroes, the algorithm enters a low power mode. The sample phase is not changed but the *largest* value is set to 6, which basically causes the sampling point to be recalculated. In practice, the packets transmitted over the InfoPad wireless length are not long enough to encounter the problem with frequency drift that the low power mode addresses. A reduced complexity clock recovery system could be implemented that only requires the calculation of the sampling point once during the entire system operation.

The clock recovery algorithm outputs the sampled data (*data_out*) and the sample clock (*clk_out*), which is ultimately passes through to the Receive Chip.

3.1.8. Receive Interface for the Proxim Radio

The receive interface for the Proxim radio on the basestation is considerably simpler than the Plessey's receive interface. The loading of the Proxim's configuration registers is performed serially and is under software control in the ARM kernel, *iprf.c*. The kernel programs the registers in the RF Xilinx that control the load signal, the data strobe, the serial configuration data, the receive/transmit mode select, the standby signal, and the ready signal for the Proxim. The kernel also handles the timing requirement for the programming of the Proxim radio. The Proxim radio generates its own receive clock, so on the basestation the receive clock from the multi-timer module is not used.

In order to write to the Proxim's registers, the ARM kernel must serialize the data to be written. The Proxim must be initialized while not in standby mode. Standby (RX_FREQ[4]) is a low power, reset mode for the Proxim. While the Proxim is not in standby, the kernel then takes each bit of the serial data and writes it to the serial configuration data register of the RF Xilinx (RXFREQ[2]). Then two writes are performed to toggle the data strobe (RXFREQ[1]). When the entire word is strobed into the Proxim, the *load* signal (RXFREQ[0]) is asserted. This procedure of strobing in serial data and then asserting *load* is continued until all registers in the Proxim have been programmed. In addition to programming the Proxim with serial data, the receive/transmit mode select (RXFREQ[3]) signal must be set to receive mode. After the Proxim has been programmed, the ready signal (RXFREQ[5]) should be asserted.

The Proxim Receive Radio interface for the Pad is specified entirely by a Viewlogic schematic (*rxradio_proxim.1*), and the I/O for the schematic block can be seen below.

Table 3-10. I/O for the Viewlogic Schematic of Proxim Interface on the Basestation

Signal Name	I/O	Description
RX_CLKIN	I	Not used.
RESET	I	Not used.
HOP_RESET	I	Hop Reset. Controlled by RF Xilinx register file.
RX_FREQ[15:6]	I	Not used.
RXFREQ[5]	I	Proxim Ready signal. Should be asserted after the receive frequency has been loaded into the Proxim.
RXFREQ[4]	I	Proxim Standby. Active low. When this signal is low, all digital electronics are reset, and the radio is put into its low power state.
RXFREQ[3]	I	Proxim TX/RX select. When this signal is low, the radio is in receive mode.
RXFREQ[2]	I	Proxim Serial Data.
RXFREQ[1]	I	Proxim Serial Data Strobe
RXFREQ[0]	I	Proxim Sync Load. When this signal is high, a write of the accumulated serial data in the radio is performed to the internal registers of the Proxim.
RX_DIN	I	Serial Receive Data. Originates from the Proxim Radio.
RXCNTL[7:0]	I	Not used.
RX_CLKOUT	O	Transmit Clock for the Transmit Control Block. Driven by an external pin by the Proxim's Transmit clock.

3.1.9. RSSI

The Plessey and Proxim radios each have an analog RSSI (Reduced Signal Strength Indicator) output that is sent through an A/D converter on both the pad and the basestation. The RF Xilinx provides the RSSI clock and the start of conversion signal to the A/D and accepts the serial input of the digitized RSSI data for further processing. The RSSI is mainly used in the mobility feature of the pad in order to determine to which cell the pad should be assigned when operating over a region containing more than one cell.

3.1.9.1. Initiating an RSSI Measurement

The RF Xilinx generates a "start of conversion" signal (RSSI_START) by latching the rising edge of the received packet (RXP) signal. The RXP signal is generated by the Receive Control Module of the RF Xilinx. Hence, an RSSI measurement is performed by the A/D converter at the start of every received packet. This circuitry is implemented with discrete components in a Viewdraw schematic that is instantiated in the Receive Radio Interface Module. This RSSI_START signal is one input to the RSSI Control Module, a

module that is described in VHDL and whose purpose is to control the reading of serial data from the A/D converter.

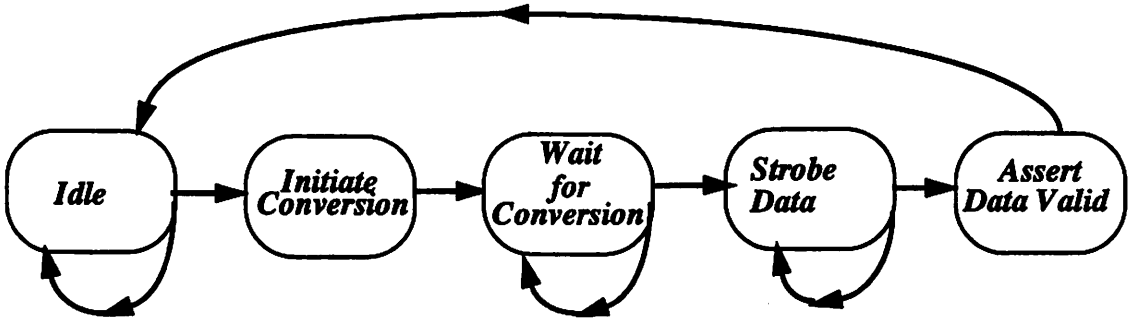
3.1.9.2. Control of RSSI Serial Data

As was mentioned in the last section, the reading of serial data from the A/D converter is controlled by the RSSI Control Module. This module is described in VHDL (*ad_cntl.vhd*) and is instantiated in the Viewlogic schematic in the Receive Radio Module (*rxradio_plessey.1*, *rx_radio_proxim.1*). The VHDL implements a state machine which generates the appropriate control signals to the A/D.

Table 3-11. I/O for the RSSI Controller

VHDL Signal Name	I/O	Description
Ext_Req	I	Active high external request for an RSSI interrupt. This is one of the vestigial remains of the original Xilinx design that is no longer used. Originally it allowed for the host to initiate an RSSI measurement at times other than the start of a received packet. The host wrote to a register in the Xilinx register file to control this. Now this input is grounded in the Viewlogic schematic, although it is still implemented in the VHDL.
Ser_Data_In	I	Serial data from the A/D converter that constitutes the measurement of the RSSI signal from the receive radio.
Start_Read	I	Also RSSI_START. This active high signal causes the RSSI controller to initiate the start of an RSSI measurement.
Reset	I	System Reset.
Clk	I	Receive Clock
Data_Valid	O	Active high signal to indicates when the data in the RSSI measurement register is valid. Once a measurement is completed, this signal is set high and remains high until the start of the next measurement.
Data_Out[11:0]	O	RSSI Measurement value.
Ser_Clk	O	Also RSSI_CLK. This is the output from the RSSI Controller to the A/D Converter, and it is used to strobe data from the A/D. The RSSI controller latches the Ser_Data_In signal on the falling edge of this clock.
Convt_l	O	Also RSSI_CONVT. This is an active low signal to the A/D converter to instruct it to begin an A/D conversion on the RSSI signal.

Figure 3-13. State Diagram for RSSI Controller



***Idle* State**

When the RSSI controller is in the *Idle* state, the serial clock and convert signals to the A/D converter are inactive. The RSSI controller monitors the RSSI_START signal which will advance the state machine to the *Initiate Conversion* State.

***Initiate Conversion* State**

When the RSSI_START (Read_Start) signal goes high, the RSSI_CONVERT signal is pulsed low for one cycle (1 μs) and is output to the A/D converter. The RSSI_VALID signal, which indicates whether or not the data in the RSSI register is a valid measurement, is lowered to zero during the conversion process, thereby indicating that the measurement in the RSSI measurement register is invalid until the present conversion is completed.

***Wait for Conversion* State**

The RSSI Controller then raises the RSSI_CONVERT line and idles for 7 cycles (7 μs) while waiting for the A/D converter to finish performing the conversion.

***Strobe Data* State**

When these 7 cycles have elapsed, the RSSI_CLK is pulsed high for one cycle and pulsed low for 1 one cycle. The data is latched when the clock is low and is fed into a 16-

bit shift register, MSB first. This process continues for a total of sixteen cycles, one cycle for each bit of the RSSI measurement from the A/D converter.

***Data Valid* State**

When the entire measurement has been obtained, the RSSI_CLOCK signal is deactivated and left at zero, and the RSSI_VALID signal is asserted. The RSSI Controller then returns to its idle state to await the next activation of the RSSI_START signal.

3.1.9.3. Methods for Reading the Resultant RSSI Measurement

There are two separate methods for obtaining the RSSI Measurement from the RF Xilinx. The first method saves the RSSI values to a register in the ARM interface module of the RF Xilinx. An interrupt is generated by the IRQ block whenever the RSSI_VALID signal is raised. Whenever the interrupt occurs, this value is written to the Status Register in the RF Xilinx to indicate that an RSSI interrupt has occurred, as opposed to a timer interrupt. When the host receives this interrupt, it can read the RSSI register in the RF Xilinx register file. The RSSI registers are implemented in VHDL (*armif.vhd*), while the interrupt circuitry for the RSSI is in the Viewlogic schematic form (*irqblock.1*).

Alternatively, another method is implemented in VHDL (*rxpath_ecc.vhd*) so the host can send an RSSI Measurement Packet down to the pad or basestation. This packet is of a particular data type called RSSI_TYPE (presently type 0x04). This type is hard-coded into the Receive data path of the RF Xilinx. When the third byte of the packet (the pad alias and sequence number must be included for this to work properly) corresponding to the data type arrives, the Receive Data Path performs a check to see if the data packet is of the RSSI packet type. If a match occurs, the data from the RSSI register (*armif.vhd*) will be packaged into a return packet also of the RSSI packet type and retransmitted across the wireless link. The Receive data path will inject the RSSI Measurement into the received packet, overwriting the previous data in the third and fourth bytes of the data payload. This alternative for RSSI measurement was implemented in order to support antenna diversity.

3.1.10. Receive Data Path

The receive control block is at the heart of the receive data path. It is responsible for formatting the received data from the radios, decoding the forward error correction, and checking the hardware CRCs. This block is described in VHDL (*rxpath_ecc.vhd*) and is instantiated in a Viewlogic schematic (*rxctrl_ecc.1*). An XBLOX component is also instantiated within the VHDL file (*rxpath_ecc_decode.1*). The Receive Control Data Path continually shifts data from the radios into its Input Data Shift Register. When the data in the shift register matches the frame sync, the Main Control State Machine is activated, and the RF Xilinx decodes the data via an external EPROM lookup table, performs a serial to parallel conversion on the data, and generates the Receive packet signal (RXP) and receive strobes for the RX Chip to use in order to latch the bytes of data from the bus. The RF Xilinx also calculates the CRC on the incoming data and checks to make sure it matches the previously calculated CRC. The RF Xilinx continues receiving data until the RX Chip asserts the End of Packet signal (RX_EOP).

The I/O for the Viewlogic schematic is described below. The Viewlogic schematic also includes the Rx Kick circuitry described in Section 3.1.11, as well as I/O to the EPROM that serves as a lookup table for the error correction.

Table 3-12. I/O for the Viewlogic Schematic of the Receive Control Block

Signal Name	I/O	Description
F_SYNC[15:0]	I	Maps to RXSYNC in RF Xilinx register file. Frame sync for the receive data packets.
RESET	I	System Reset controlled by RF XIlinx register file.
RSSI_VAL[11:0]	I	RSSI Measurement value supplied by Receive Radio Module
RX_CRC	I	Receive CRC. When this signal is asserted by the RX Chip, the RF Xilinx identifies the current byte as a CRC value.
RX_DIN	I	Receive Serial Data from the Receive radio
RX_ECC[1:0]	I	Error Correction level for Received data payload - from RX Chip
RX_EOP	I	Receive End of Packet signal from RX Chip. Indicates to the RF Xilinx that all bytes of the packet have been received, and the end of the packet has occurred.
RXCLK_OUT	I	Buffered Clock from the RXCLK provided by the multi-timer module
RX_KICK	I	Resets the RXP line whenever an infinite length packet is suspected. Controlled by the RX Kick register in the RF Register file. Discussed in Section 3.1.11.
ROM_DATA[7:0]	I	Decoded receive data from the external EPROM lookup table.
ROMADDR[16:0]	O	Address to the external EPROM used to decode the receive data

RX_DOUT[7:0]	O	Received data output to the RX Chip
RXS	O	Receive data strobe. Strobe high for each byte of received data to indicate that the RX Chip can latch the data value.
RXP	O	Receive packet signal. Indicates to the RX Chip that a packer is being received.
RXP2	O	Buffered RXP for use by the Receive Radio Module
DISTNOW	O	When this signal is asserted, the bits from the external EPROM that indicate the distance (i.e. the number of errors in a block) of the raw data from the decoded data can be latched in order to calculate an error count. This signal is not currently used, but is available for future implementations.

Table 3-13. I/O for the VHDL design of the Receive Data Path

Signal Name	I/O	Description
FRAME[15:0]	I	Maps to RXSYNC in RF Xilinx register file. Frame sync for the receive data packets. Maps to F_SYNC in Viewlogic schematic.
RESET	I	System Reset controlled by RF XIlinx register file and RX Kick module.
RSSI_VAL[11:0]	I	RSSI Measurement value supplied by Receive Radio Module
RXCRC	I	Receive CRC. When this signal is asserted by the RX Chip, the RF Xilinx identifies the current byte as a CRC value.
DIN	I	Receive Serial Data from the Receive radio. Maps to RX_DIN in Viewlogic schematic.
RXECC[1:0]	I	Error Correction level for Received data payload - from RX Chip
EOP	I	Receive End of Packet signal from RX Chip. Indicates to the RF Xilinx that all bytes of the packet have been received, and the end of the packet has occurred. Maps to RX_EOP in Viewlogic schematic.
CLK	I	Buffered Clock from the RXCLK provided by the multi-timer module. Maps to RXCLK_OUT in Viewlogic schematic.
RomData[7:0]	I	Decoded receive data from the external EPROM lookup table.
RomAddr[16:0]	O	Address to the external EPROM used to decode the receive data
DOUT[7:0]	O	Received data output to the RX Chip. Maps to RX_DOUT in Viewlogic schematic.
RXS	O	Receive data strobe. Strobe high for each byte of received data to indicate that the RX Chip can latch the data value.
RXP	O	Receive packet signal. Indicates to the RX Chip that a packer is being received.
distNow	O	When this signal is asserted, the bits from the external EPROM that indicate the distance (i.e. the number of errors in a block) of the raw data from the decoded data can be latched in order to calculate an error count. This signal is not currently used, but is available for future implementations.

The interface timing signals for the RX Chip/RF Xilinx interface can be seen in the following Figure 3-4.

Figure 3-14. RX Bus Control Signals

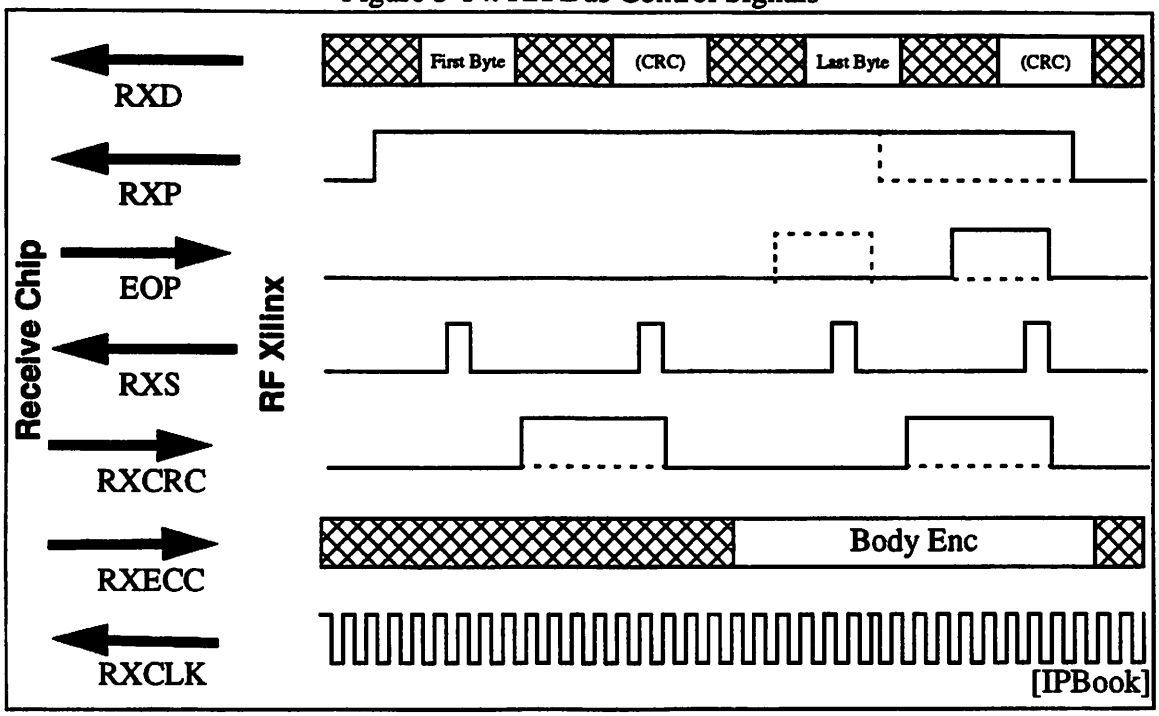
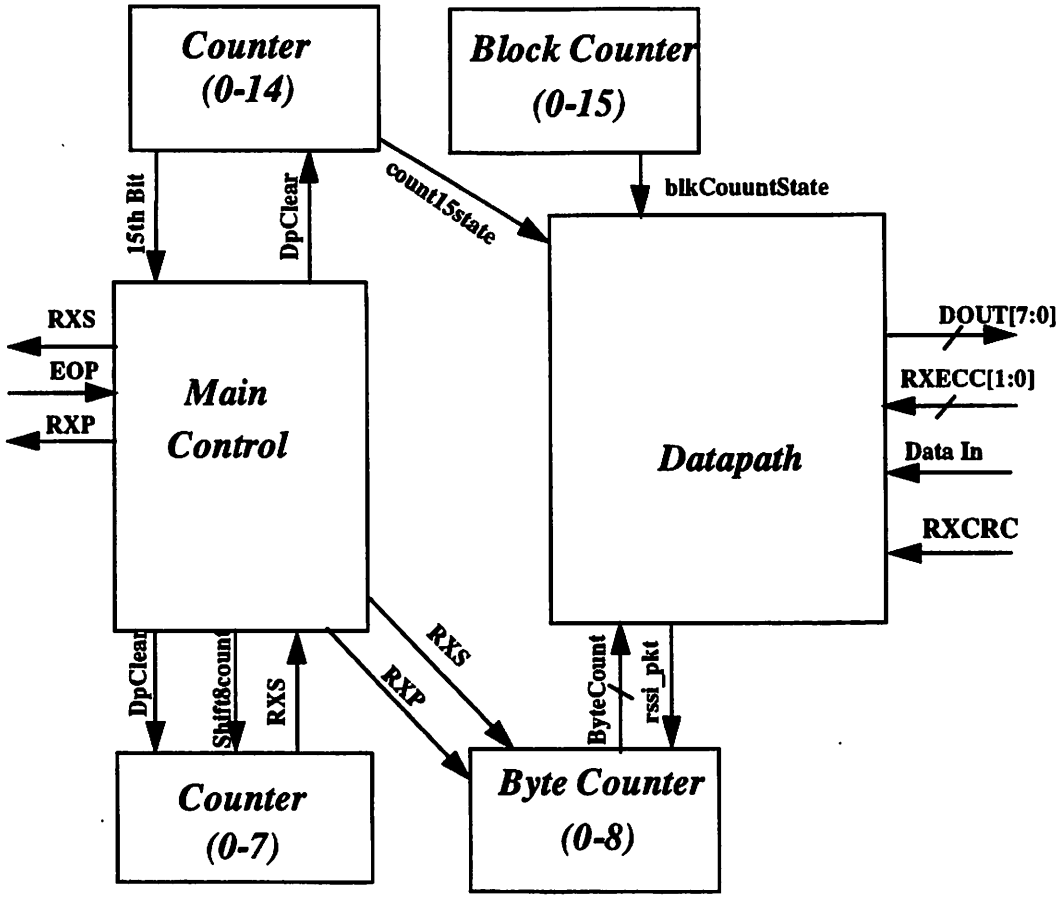


Figure 3-15. Decoder High Level Schematic

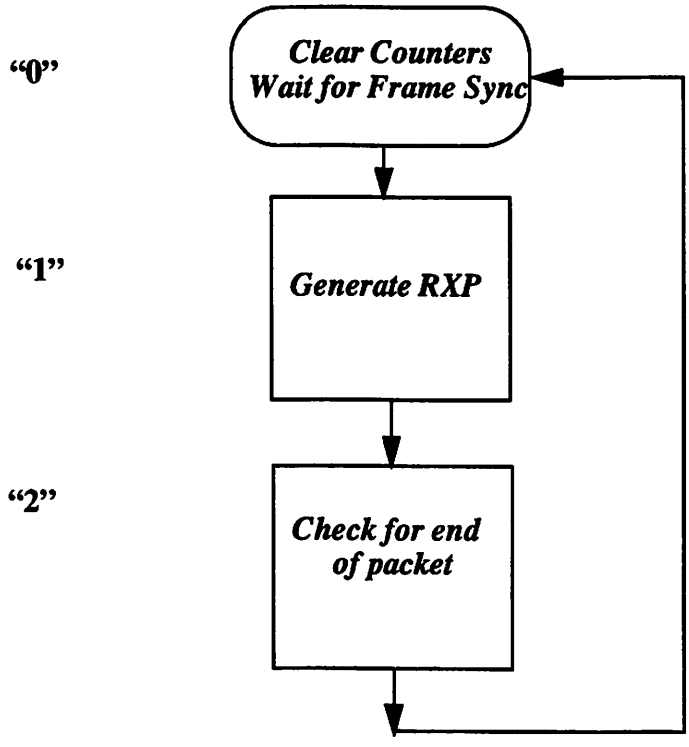


3.1.10.1. Control Logic

Main Control State Machine

The receive control block is driven by a state machine with three states called *Main Control*, which basically controls the signals to the RX Chip interface, i.e RXS and RXP. The simplified state diagram is presented below, and each state is discussed in more detail in the following section.

Figure 3-16. State Machine for the Main Control



State "0"

The state machine clears the counters in the design with the *dpClear* signal. If the frame sync has been identified, the state machine sets the *setRxActive* signal high and advances to *State "1"*.

State "1"

Once in *State "1"*, the state machine asserts the RXP line to indicate to the RX Chip that a packet is being received. When the reception of the current 15 bit block is completed, the state machine advances to *State "2"* to check to see if more data should be received.

State "2"

The state machine continues to assert RXP and verifies that there is still data to receive, i.e. *rxActive* is not zero. If there is additional data to receive, the machine starts in this state; however, if there is no further data, the state machine returns to state zero where the RXP line is de-asserted.

Control Counters

The Receive Control has several counters that help to implement the control of the receive data path. These are the count8, the count15, byte, and block counters. Each counter is implemented as a state machine in VHDL and is described separately below.

Count8

The count8 counter is a 2 bit up-counter that is initialized to 0 by the control logic in the Main Control state machine. The counter increments whenever the *shift8count* signal (also from the Main Control state machine) is asserted, and a flip flop latches the counter value at the rising edge of the transmit clock. When the counter reaches 7, the receive strobe (RXS) signal is asserted, and the counter wraps around to 0 and sets RXS back to zero.

Count15

The count15 counter is a 4 bit up counter that is initialized to 2 by the control logic in the Main Control state machine (*dpClear*, i.e. Data Path Clear). The count15 counter is necessary because the encoded data is received in 15 bit blocks. The counter is incremented every receive clock, and a flip flop latches this value. Before each increment, however, a line balance signal is set within this state machine. The line balance pattern is ultimately exor'ed with the input data to remove the DC balance applied to the transmitted data signal. Other control signals are also determined by the count15 counter. After the count15 has

counted down fourteen states (to a count value of 13), the *End14bit* signal is generated. This signal helps synchronize when to enable and disable decoding. When the counter reaches the value of 14, the counter generates the *End15bit* signal and wraps around to begin at 0. When the count15 value equals 0, the Input Data Shift Register is loaded (*loadInDatReg* = 1). When the count15 value equals 14, the Error Correction Register is loaded with the value for the decoding error correction level (*loadEccReg* = 1). When count15 equals 1 or 9, the address for the external EPROM is loaded into the ROMOut Register (*loadROMOutReg* = 1).

Byte Counter

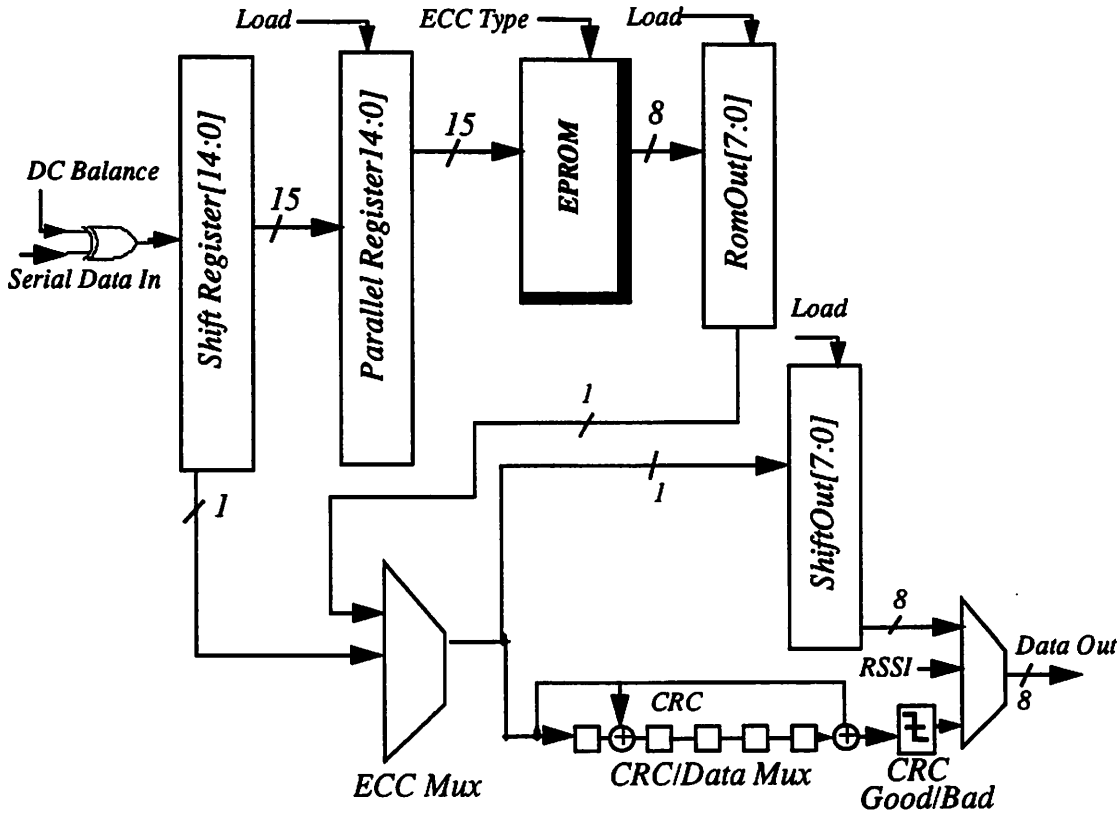
The byte counter is used to determine when insert the RSSI measurement if the received packet is of type RSSI_PACKET. This counter is a 4 bit up counter that counts from 0 to 8. It is cleared to zero whenever the RXP signal is asserted.

Block Counter

The block counter is used to determine when to switch from the header decoding level to the decoding level prescribed by the RXECC signal. The block counter is an up-counter that counts from 0 to 15. The counter increments whenever the *loadEccReg* signal is asserted. If the block counter is less than or equal to 7, the error correction level is set to maximum error correction. Otherwise it is set to the value in the RXECC register. When the counter reaches 15, it does not wrap back around to 0. Instead it stays locked at 15, unless the counter is reinitialized to zero by the Main Control state machine via the *dpClear* (Data Path Clear) signal.

3.1.10.2. Receive Data Path

Figure 3-17. High Level View Of Receive Data Path



DC Balance

The Serial data input from the receive radios in exor'ed with the line balance pattern generated by the count8 module. This removes the line balance added by the transmit data path. The output is passed to the Input Data Shift Register (*inputdataReg*).

Input Data Shift Register

The Input Data Shift Register is used to accumulate the data from the serial data input. It is 15 bits wide which corresponds to the 15 bit blocks used in the error correction encoding. The Input Data Shift Register has two outputs. The first is a serial data stream that is used for data without encoding and as an input to the CRC calculation module. The second is a 15 bit parallel input to a 15 bit register (*parallIn*) that is used to latch the 15 bit

value. This latching occurs whenever the *loadInDataReg* signal is asserted. This signal is asserted whenever the count15 counter reaches a value of zero.

Parallel Data Register

This register is loaded according to the previous section with a 15-bit block of input data. This value becomes the lower 15 bits of the address used to index the external EPROM. The upper two bits of the address are determined by which error correction level is chosen. This will be discussed in the following section on the Receive Decoding Component.

Receive Decoding Component

The Receive Decoding Component is instantiated as a black box in the VHDL file, and is specified by a Viewdraw schematic that uses XBLOX modules, most notably an internal PROM, not to be confused with the external EPROM that functions as the FEC look up table. The error correction level and the value of the count15 counter are input to an internal PROM whose contents are described by the *rx_decode_ecc.mem* file. Based upon the inputs to the PROM, the two upper address lines to the external EPROM are generated (*lutAddr*), as well as a data signal (*infobit*) that acts as a flag to signal when bits of data are present, as opposed to the added redundancy.

External EPROM LUT

The external EPROM is a AM27C010 EPROM that is programmed with the *epromdata.src* file. The EPROM is has a 16-bit address. The EPROM can be thought to have four "pages" of data that are indexed by the error correction type. Only two pages are actually needed. A third page is mapped to non-error corrected data, while a fourth is undefined. The data output is a 5 bit data value (DataOut[7:3]) accompanied by a 3 bit indicator (DataOut [2:0])of how many errors were corrected in the decoding of that block. If more than 1 error is present in medium error corrected data, this 3-bit value will be invalid. In the case of the maximum error correction, if more than 3 errors are present, the 3-bit value will also be invalid. This is due to the fact that if a greater number of errors than the code

can correct occurs in a block, an entirely incorrect code word may be selected. The contents of the EPROM were generated by Kathy Lu in support of her master's thesis.

EPROM Output Shift Register

When the *loadRomOutReg* signal is asserted, the EPROM output shift register loads the 8-bit data output from the external EPROM. When the load signal is not asserted, the EPROM output shift register begins shifting data out of its serial port with a destination of the ECC Mux.

ECC Mux

The Error Correction level is set to maximum when decoding one of the first eight bytes of data. Otherwise, the error correction level is set to the value specified by the RXECC signal.

If the error correction level is uncoded, the ECC Mux passes the serial input data from the Input Data Shift Register on to the next stage of the datapath: the CRC and *ShiftRegOut* shift register. If one of the error correction levels is selected, the serial output from the EPROM Output Shift Register is instead passed on to the CRC and *ShiftRegOut* shift register.

Output Shift Register

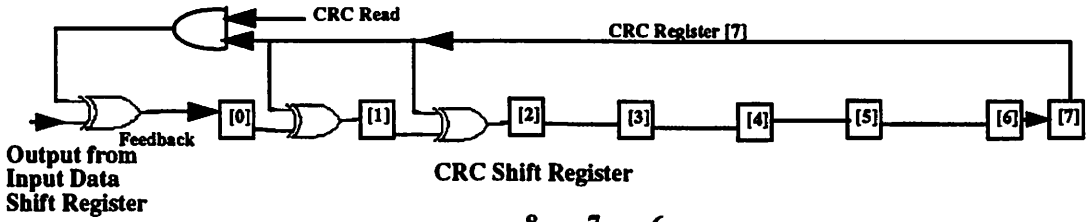
Whenever *shift8count* is asserted, i.e. the current byte has been processed and relayed to the RX Chip, the output shift register begins shifting in the next byte of data. This data originates from the ECC Mux.

CRC Algorithm

When the CRC Clear signal (*dpClear*) is asserted by the *Main Control* state machine, the CRC Shift Register is cleared to a value of 0. The CRC is implemented with an 8-bit shift register. The feedback signal for the CRC is generated by exor'ing the MSB of the CRC shift register with the input serial data, provided the CRC is not being transmitted. If the CRC value is being transmitted, the CRC feedback signal is disabled. The

CRC is calculated by writing the feedback signal to the LSB of the shift register, writing the exor of the LSB and the feedback signal to the second bit in the shift register, and writing the exor of the feedback and the second bit of the shift register to the third bit of the shift register. Each time a new bit of data is shifted out of the input shift register, the CRC shift register updates its values. Whenever the RXCRC line is raised, the contents of the CRC register are examined. If any of the bits is a 1, the CRC is invalid. An eight bit value is generated to correspond to valid (0x00) or invalid (0xff). This byte is then sent to the output mux.

Figure 3-18. CRC



$$g(x) = x^8 + x^7 + x^6 + 1$$

Threshold for CRC and the Output Mux

The Output Mux selects between the CRC valid byte, the received data, and the RSSI data. When RXS is strobed, the RXCRC line is examined to see if it has been pulsed high by the RX Chip. If the RXCRC line is high, the *CRCbad* signal is checked. If the CRC is invalid, 0xff is relayed in the data stream to the RX Chip. However, if the CRC is valid, 0x00 is relayed in the data stream to the RX Chip. This is true for both the header and payload CRCs. The RX Chip only raises the RXCRC signal for the header and body CRCs if the data type to be received is configured to include CRCs.

If the RXCRC signal is not high, the data type is examined to determine whether or not the data is an RSSI packet. If the packet is of type RSSI, the third and fourth bits of the payload are replaced with the value from the RSSI measurement from the receive radio.

The remaining bytes of the packet are untouched. If the packet is not an RSSI packet, then the received data bytes are passed through to the RX Chip.

3.1.11. RX chip Work-around

With the Receive Control Block is the RX Chip work around. It is implemented with a Viewlogic schematic (*rx_unstick.1*). It is necessary to recover from a possible receiver chip lockup. If a packet received over the radio link has a length of 0xffff or if the Receiver EOP signal is not sent to the RF Xilinx, the receive state machine in the Xilinx can lock up in the receiving state indefinitely. The ARM has to periodically call a routine with a period longer than the longest possible packet (every second or so should be fine). This activates a state machine in the Xilinx which detects if the RX_KICK register has changed twice or more while RXP has stayed TRUE. If so, the receive state machine is reset, and the unit continues as before. This condition often occurs at start-up, but is usually not seen during operation. A sample of the ARM code used to perform this function can be seen below.

```
void iprf_UnstickRXP(void)
{
  *XILCNTL = RF_UNSTICK_RXP_MASK | _xil_RunMask;
  *XILCNTL = _xil_RunMask;
}
```

The I/O for the RX Kick module can be seen below.

Table 3-14. I/O for the Viewlogic Schematic of the Receive Control Block

Signal Name	I/O	Description
RXCLK_OUT	I	Buffered Clock from the RXCLK provided by the multi-timer module
RXP2	I	Buffered RXP for use by the Receive Radio Module
RESET	I	System Reset controlled by RF XIlinx register file.
RX_KICK	I	Resets the RXP line whenever an infinite length packet is suspected. Controlled by the RX Kick register in the RF Register file.
NewReset	O	System Reset or RX Kick if infinite length packet is suspected

3.1.12. Interrupt Handling

The RF Xilinx interrupts the ARM in two cases. The first case is whenever a timer event occurs, and the second case is when an RSSI measurement is ready. A register (XILSTAT) stores which event causes the interrupt to occur, and the ARM is responsible for reading this register and deciding which event to service. The interrupt remains high until the ARM clears it by writing to the appropriate bit in the XILCMD register.

3.1.13. General Purpose Timer

The Xilinx contains a general purpose timer that is a 16-bit up counter which generates an interrupt whenever its terminal count is reached. The timer is implemented by a Viewdraw schematic (*irqblock.1*) that uses an XBLOX module for an internal counter. The XBLOX modules are used for the counter in order to help provide a tighter constraint on the routing of time critical data paths. The timer is driven by a 1 MHz clock that is generated in the multi-timer module. The terminal count for this 16-bit up counter is loaded from the 16 bit Timer Register in the Xilinx when any of the following three conditions occur:

1. Reset
2. ARM processor updates the Upper Byte of the Timer Register
3. The counter has reached the terminal value

There is also a fourth vestigial condition called an external load that is implemented in the timer module, but is not used in the Xilinx design. This provides for some external event controlling when the terminal count is updated from the Timer Register. The load signal is generated by a logical OR of these events, and a one cycle load pulse is generated to enable the data input port on the XBLOX counter module.

Whenever the terminal condition is reached, i.e. whenever the counter counts up to the value in the Timer Register, an active high timer interrupt to the host is generated. Since there is only one interrupt line to the host to support all events in the Xilinx that might cause an interrupt, whenever the timer interrupt occurs, the Timer Interrupt Status Register is updated. The host can then read this status register to determine which event has caused the external interrupt. The interrupt stays high until it is cleared. The interrupt is cleared

when the host writes to the Timer Interrupt Acknowledge register or when a reset condition occurs.

3.2. The ARM kernel

3.2.1. Register Mapping

The ARM kernel was modified so that the register reads and writes corresponded to the registers in the EDAC Xilinx. A complete listing of basestation, pad, and wired register descriptions can be seen in the Appendix, Section 6.1. on page 108. This register remapping resided mostly in modifying the masks and register addresses in the include files of the kernel, however several changes were made in the ARM code in order to accommodate the EDAC Xilinx register mappings. In an attempt to maintain one design that was currently undergoing development in addition to the EDAC integration, *ifdef* commands were used to delineate the "non-ECC" code from the "ECC" code. In this manner, the flow control and other developments to the kernel that were compatible with the original Xilinx could continue, and the EDAC Xilinx was kept current. Relevant files for register mapping changes are *rfxil.h*, *rfxil_masks.h*, and *rfxil_h.s*.

3.2.2. Writing to the RF Xilinx Registers

Most of the registers in the RF Xilinx are write only registers. The ARM kernel uses mask variables in order to shadow the value of each register write. In addition, bit masks are maintained in an include file that correspond to writing to only a particular bit in a register without overwriting all the other bits. When writes to the registers are performed, the ARM kernel ORs the bit mask of the value to be written with the shadow variable of the register. Hence, only the bit that is intended to be changed is altered in the register write. This is especially important when clearing interrupts, avoiding resetting the entire system, etc.

For example, in the include file *rfxil_masks.h*, the bit mask for the system reset is defined. Reset is controlled by the first bit in the RF Xilinx Control Register (XILCNTL):

```
#define RF_XIL_RESET_MASK 0x01   //bit 0 of XILCNTL
```

A shadow variable is declared in *iprf.c* in order to shadow the XILCNTL register. Each register in the RF Xilinx that contains separate functionalities per bit or groups of bits needs a shadow register. The current design shadows the control register, the transmit control register, and the receive control register. The shadow variable can be seen below:

```
static int _xil_RunMask;
```

When a write to the Reset bit in the XILCNTL register is desired, the kernel ors the RESET_MASK with the _xil_RunMask and then writes this value to the XILCNTL register. In this manner, other bits like the Loopback bit which is also in the same register, are not inadvertently cleared. An example is seen below:

```
*XILCNTL = RF_XIL_RESET_MASK | _xil_RunMask;
```

3.2.3. Reading the RF Xilinx Registers

Reading the RF Xilinx registers is accomplished in assembly code in the *ihandler.s* file. Whenever an interrupt occurs, this code reads the RF Xilinx Status (XILSTAT) register to determine which event has triggered the interrupt. If the interrupt is an RSSI interrupt, then the XILRSSIMSB and XILRSSILSB registers are read. The C code in the kernel can only access the RSSI value through a call to the RSSI handler.

3.2.3.1. Resolving Interrupts

As was discussed in the previous section, the assembly code is responsible for resolving the interrupts. The Xilinx Status register contains two bits, one for the timer and one for the RSSI. Whenever the Xilinx asserts its interrupt line, the appropriate event designator is set in the Status register (XILSTAT).

3.2.3.2. Acknowledging Interrupts

When an interrupt is received, it is acknowledged by writing to the appropriate acknowledge bit in the XILCMDREG register. Again, the two events that can generate an interrupt are the timer and the RSSI.

3.2.3.3. Obtaining an RSSI Value

An RSSI interrupt is generated on every received packet. In order to obtain the RSSI value, it is only necessary to call the kernel's interrupt service routine (*kernel_RegisterISR*) and provide it with the interrupt mask for the RF Xilinx interrupt.

3.2.3.4. Reset

The active high reset for the RF Xilinx is maintained by an internal register that has to be set by the ARM. A register write to the RF Xilinx reset register (XILCNTL[0]), causes an internal system reset in the RF Xilinx. This write operation is performed in *iprf.c*.

3.2.3.5. Loopback Mode

The RF Xilinx supports a loopback mode which enables testing of the InfoPad hardware without sending data through the radios. Transmitted data is simply routed back through the RF Xilinx to the Receive ASIC, bypassing the radios. Loopback mode is activated by setting the RF Xilinx loopback register (XILCNTL[2]). This write operation is performed in *iprf.c*.

3.2.3.6. Plessey Interface

The ARM must write the programming data for the Plessey Radio to the RF Xilinx. The Plessey must be programmed with the transmit or receive frequency, depending on whether the configuration is for the pad or the basestation. The transmit and receive frequencies are programmed by writing to XILTXF and XILRXF, respectively. This programming is performed in *iprf.c*.

3.2.3.7. Proxim Interface

The Proxim interface is not as simplified as the Plessey interface. The Proxim has several tasks in the ARM associated with it. In order to set up the uplink or downlink, the ARM executes a sequence of writes to the XILTXF or XILRXF registers. Each bit of these register (when programming the Proxim) has a different meaning. These signals are discussed in detail in Section 3.1.6 and Section 3.1.8. This programming is performed in *iprf.c*.

Proxim Register (XILTXF or XILRXF)

Figure 3-19. Proxim Register Bit Definitions

Bit position							
7	6	5	4	3	2	1	0
		Ready	Stdby	Tx/Rx	Syn-Data	Syn-Clk	SynLd

- Ready: Set high when the ARM has completed the programming of the Proxim Radio.
- Standby: Set low when the radio is being programmed.
- Tx/Rx: Set to 1 for Transmit, 0 for Receive
- Syn_Data: Serial data input for programming the divider word into the Proxim
- Syn_Clk: Data strobe for each bit of the divider word
- SynLd: Load signal asserted when the divider word has been completely written

Proxim Initialization

- Take Proxim out of standby mode by writing a 1 to the active low Standby bit.
- Write the reference divider word to the register in order to program the frequency for the radio. This is a serial write. Each bit that is written must be accompanied by a write to the Strobe bit in the register. When the entire word has been strobed in, the ARM writes a 1 to the Load bit in the register.
- Set the Rx/Tx option on the Proxim radio.
- Enable the radio by writing a 1 to the Ready bit.

3.2.4. Using the Timer

The terminal count for the timer is set by writing to the XILTIMERLSB and XILTIMERMSB registers. These writes occur in *timer.c*. The timer will generate an interrupt whenever this terminal value is reached. The timer will then wrap around to zero. It is important to remember that the timer register is write only.

3.2.5. Controlling the RX Chip Work Around

In order to keep the RX chip from locking up, the *iprf_UnstickRXP* function (in *iprf.c*) is called every 10 seconds. The timer handler function is set up in the *rx_Init* func-

tion (in *iprx.c*) in order to set this 10 second periodicity. The *iprf_UnstickRXP* function writes a 1 to the *RX Kick* bit of the Xilinx Control Register (XILCNTL). This kicking action ensures that if an infinite length packer is received, the system will not remain locked up. See "RX chip Work-around" on page 83.

3.3. Debugging Procedure

3.3.1. Simulation of Timer with Software

At the onset of this project, the EDAC Xilinx did not support an internal timer. In the initial phase of testing, timer controlled events were replaced by software looping that estimated the duration of the original timer specification. A timer was added to the EDAC Xilinx after the first stage of integration had been accomplished. The timer controlled events in the kernel were then reinstated.

3.3.2. Self-Test Infrastructure

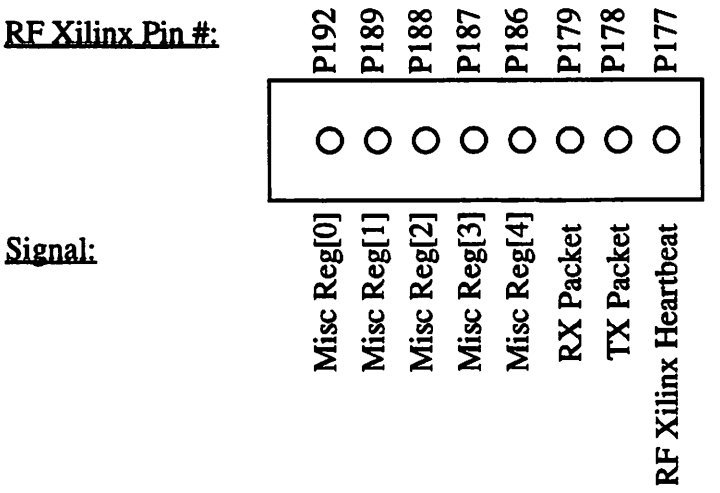
In order to aid the integration process, it was important to add a test function to the kernel. This self test was activated by a switch in the command line, and once activated, it caused data packets to be transmitted and received. This simple loopback test could be used to verify basic functionality of the system, i.e. functioning receive and transmit packet signals. After the self-test was integrated, the next step was to modify the kernel to support the new Xilinx register.

The self test that was added to the ARM kernel allowed the basestation and pad to transmit and receive the same data and verify that the data was not corrupted. This was a good litmus test for functionality, as it enabled print statements to be added to the kernel as the testing and debugging progresses. When the loopback test worked, the transmission and reception of a data packet had been verified. At first, CRC and ECC were kept deactivated, but eventually they too were included in the loopback test.

3.3.3. LEDs

There are eight LEDs on the InfoPad circuit board that could be used for debugging purposes.

Figure 3-20. Diagnostic LEDs



The LEDs consist of a heartbeat controlled by the RF Xilinx, a receive packet LED, a transmit packet LED, and 5 LEDs controlled by the register in the RF Xilinx register file. The programmability of the LEDs provides for a more flexible testing environment. User level code in the ARM can be used to control the five LEDs for debugging or status purposes.

3.3.4. Probe of Internal Xilinx Signals

The internal signals of the EDAC Xilinx design were brought out to test pins on the board during critical parts of the debugging process. These signals were probed with an oscilloscope or the DAS in order to determine where a state machine may have locked up, the presence of the clock, and to ensure that the chip was not stuck in a reset state. Signals could be brought out in one of two ways. If the signal was not synthesized into an intermediate signal of a CLB, it could be probed using XACT's EditLCA tool. This helped to verify problems in routing. In particular, the pin for the RSSI input from the A/D would sometimes not route automatically, so the EditLCA was used to re-route hard to place signals as well as add the external probes that could be used to minimally modify the design routing. Using EditLCA also saved the enormous compile time required for modifying the

design. Unfortunately, some signals were unreachable from the LCA file. Many of the counters were optimized to the extent that the original signal could not be accessed for probing. It was then necessary to bring out certain signals by hand within the Viewlogic schematics and resynthesize the entire design.

3.3.5. Gateway Test

The final test of the system was to boot the pad and basestation, and to then start the gateway enabling the X-terminal link to the workstation. When this was accomplished, the new EDAC Xilinx appeared to exhibit the same functionality as the original RF Xilinx. The user was able to use the pen input successfully, view QuickTime text/graphics movies, as well as take advantage of the other features of InfoPad. Although this was a subjective measure, it did indicate that some degree of integration had occurred.

4 Analysis

4.1. Channel Statistics

Analysis on the channel statistics was performed by using user level code in the ARM kernel in order to transmit and receive packets with known data. If errors were present in the received packets, these errors were printed to the screen via a serial link and logged. During normal operation of the InfoPad system in the laboratory environment, the presence of errors in the packet payload was minimal to the extent that forward error correction was not necessary. These results can be seen in Table 4-1. The test included measurements from 10,000 received packets (with 20-byte payload) on both the uplink and the downlink.

Table 4-1. Statistics From System Test During Normal Operation

	% of Packets with Incorrect Body CRCs [1]	% of Incorrect Payload Bits [2]
Downlink		
No ECC	0	0
15,11,1 ECC	0	0
15,5,3 ECC	0	0
Uplink		
No ECC	0.02%	0.00125%
15,11,1 ECC	0	0
15,5,3 ECC	0	0

[1] Out of 10,000 packets received, the percent of the received packets with invalid body CRCs and valid header CRCs.

[2] Out of 10,000 * 20 * 8 bits received in the payload of the packets, the percent of those bits with errors

In order to obtain a better idea of how forward error correction could potentially increase the reliability of packet transfer, interference was added by operating two basestations and pads at the same frequency, and observing the error rates as forward error correction levels were varied on the payload.

4.1.1. Interference Test Description

Each pad and basestation transmitted 10,000 packets with 20 byte payloads. The experimental basestation/pad pair transmitted payloads containing all zeroes. The interference basestation/pad pair transmitted a 0x0002 in each 2 bytes of its payload. The experimental basestation/pad pair were connected by serial cables to the workstation in order to record whenever errors were received. The contents of the received packet were printed to the screen if errors were noted by the test program running on the pad and basestation. The interference and experimental systems were programmed with different frame syncs to eliminate synchronization problems. It was difficult to perform a completely controlled experiment due to the lack of exact knowledge about the injected errors. Other interference was possible in the laboratory environment due to blocking and various other sources of random noise. Hence, this aim of this experiment was to see if forward error correction could in fact improve reliable reception under adverse conditions. It is important to note that this experiment is introducing interference through jamming with another basestation/pad pair operating at exactly the same frequencies.

4.1.2. Test Results

Table 4-2. Statistics from System Test with Added Interference

	% of Packets with Incorrect Body CRCs [1]	% of Incorrect Payload Bits [2]
Downlink		
No ECC	67%	1.7%
15,11,1 ECC	57%	2.2%
15,5,3 ECC	10%	0.17%
Uplink		
No ECC	0.84%	0.013%
15,11,1 ECC	3.1%	0.16%
15,5,3 ECC	7.5%	0.40%

[1] Out of 10,000 packets received, the percent of the received packets with invalid body CRCs and valid header CRCs.

[2] Out of 10,000 * 20 * 8 bits received in the payload of the packets, the percent of those bits with errors

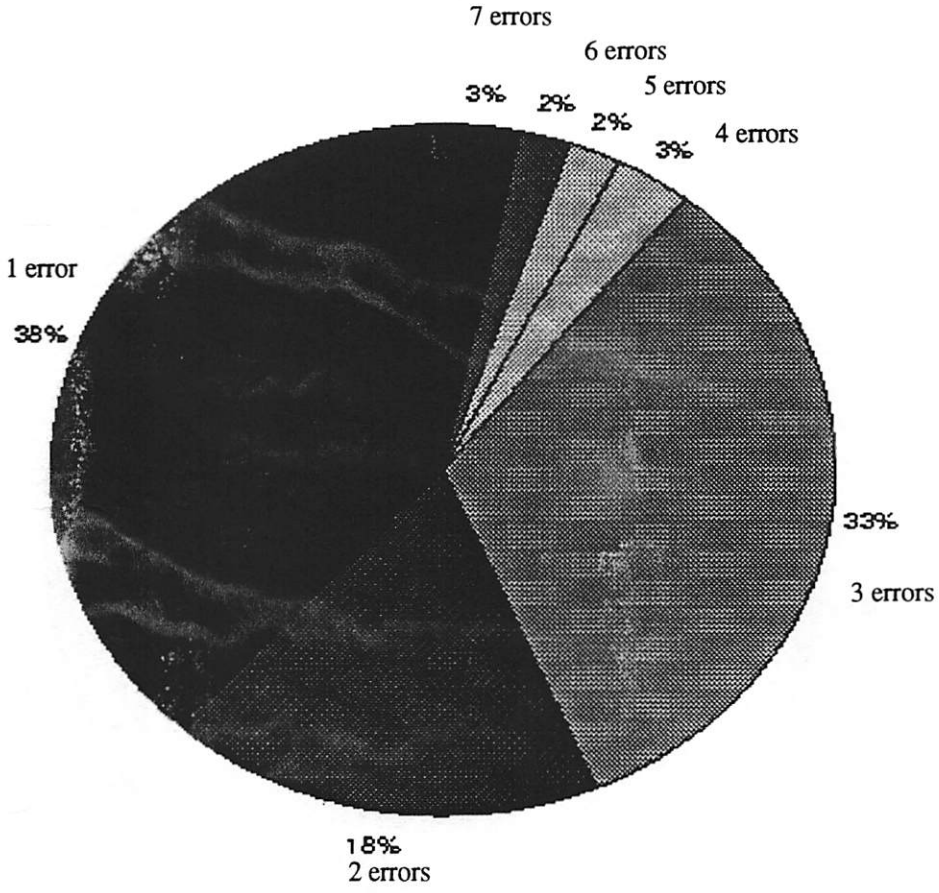
The uplink experienced very few errors on the non error corrected test, so it is difficult to conclude anything from the received data on the basestation. The medium and maximum levels of error correction appear to provide worse performance, but with such a small sample of errors, this is quite possibly the effects of random burst errors. On the other hand, the downlink experienced quite a few errors (67% of Body CRCs invalid) in the non error corrected case, giving a much better sample for comparison. The maximum level of error correction provided quite a significant improvement over no error correction. The medium error correction level in fact performed worse on the average than having no error correction. This will be discussed below. One must also remember that the errors were somewhat random in nature, and more errors could have been introduced in a one time interval than in another.

One possible explanation for the high percentage of errors on the downlink (and very few on the uplink even operating under high interference conditions) is the fact the uplink uses a direct sequence spread spectrum radio, while the downlink uses a hopping spread spectrum radio with hopping disabled. Since the hopping is disabled on the down-

link, most of the benefits of spread spectrum are lost, hence the high error rate under jamming conditions.

In order to provide a better understanding of the nature of errors in this test, a histogram of the data was performed to examine the burstiness of the channel using the non-error corrected case. Pie charts were also compiled to see the how many 15-bit blocks contained certain numbers of errors. In bursty channels, short error correcting codes can rarely improve reliability in communication. The medium error correcting code fails if more than one error in a 15 bit block occurs, while the maximum error correcting code fails if more than three errors occur in a 15 bit block. The histogram below charts the probability of an error occurring in each of the 14 bits following a bit error. Note: Bit errors only contribute to the histogram once, hence only the first bit error in a cluster is used to start the histogram measurement, i.e. each the starting point for the 15 bit block used in a histogram does not re-initialize with every bit error.

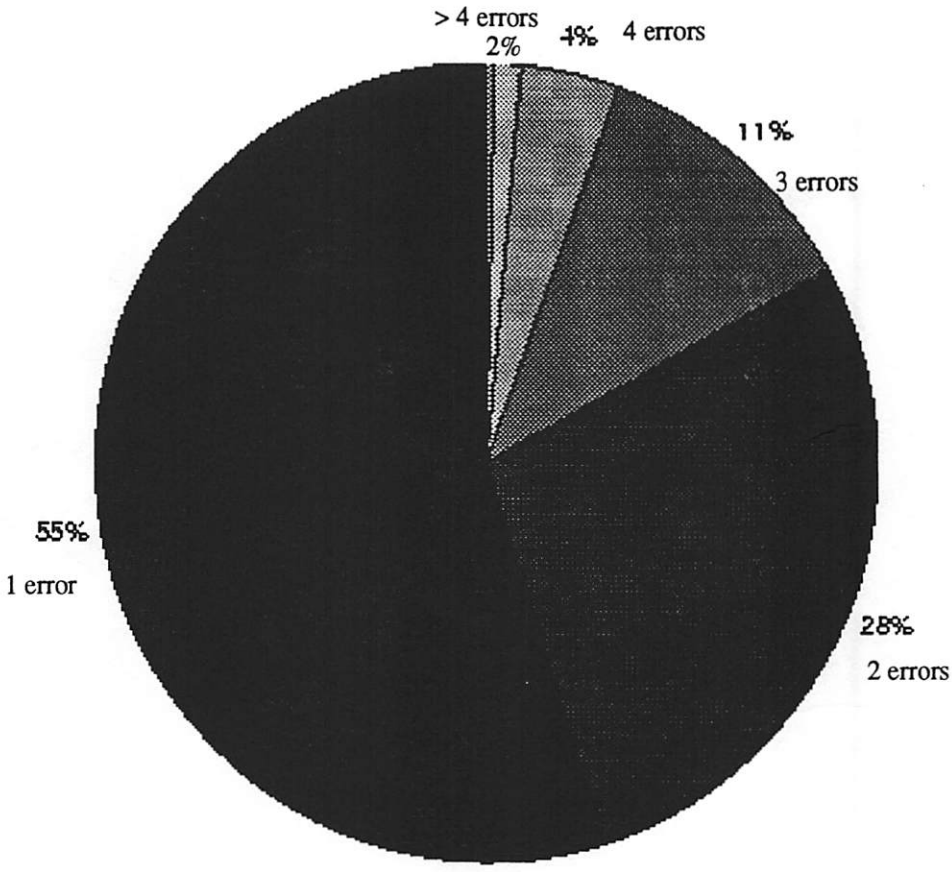
Figure 4-1. Breakdown of Burst Length in 15 Bit Blocks



This chart show the percentage of 15 bit blocks in the experiment that contain various numbers of errors. It is useful to see how many errors appear in a 15 bit block in order to determine what level of error correction could be effective. A 15 bit block is defined as a beginning with an error and continuing until 15 bits are counted. The next 15 bit block begins with the occurrence of the next error.

From this pie chart, it appears that medium error correction would be beneficial 38% of the time, while maximum error correction would be beneficial 89% of the time. The data in Table 4-2 does not confirm this, however, but it could be the result of a small sample of actual errors. The errors are also time varying, as someone walking into a room, a power supply activating, etc., can introduce noise.

Figure 4-2. Breakdown of Burst Length in 15 bit Blocks

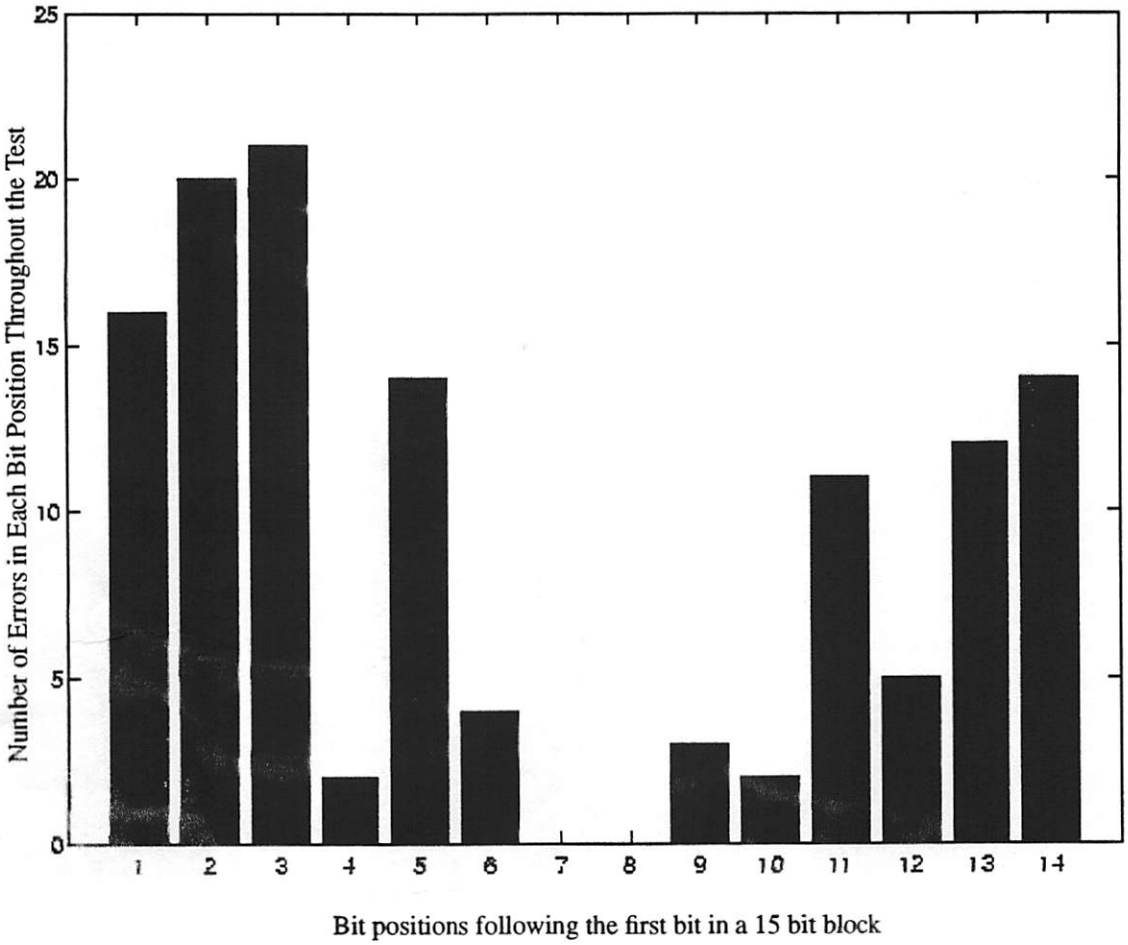


This chart show the percentage of 15 bit blocks in the experiment that contain various numbers of errors. It is useful to see how many errors appear in a 15 bit block in order to determine what level of error correction could be effective. A 15 bit block is defined as a beginning with an error and continuing until 15 bits are counted. The next 15 bit block begins with the occurrence of the next error.

From this chart, one can see that 55% of the 15 bit blocks in the test contained only 1 error, while 94% of the blocks contained 3 errors or less. With this data, one can see that the maximum level of error correction seems quite appropriate for this test, while medium error correction would work in approximately half of the cases. When medium error correction is used when burst errors are present, medium error correction can actually intro-

duce errors to the data stream by incorrectly decoding the incoming data when a received word has more errors than the error correcting code can accommodate. This appears to be the case apparent in the statistics of Table 4-2.

Figure 4-3. Histogram of UpLink Errors

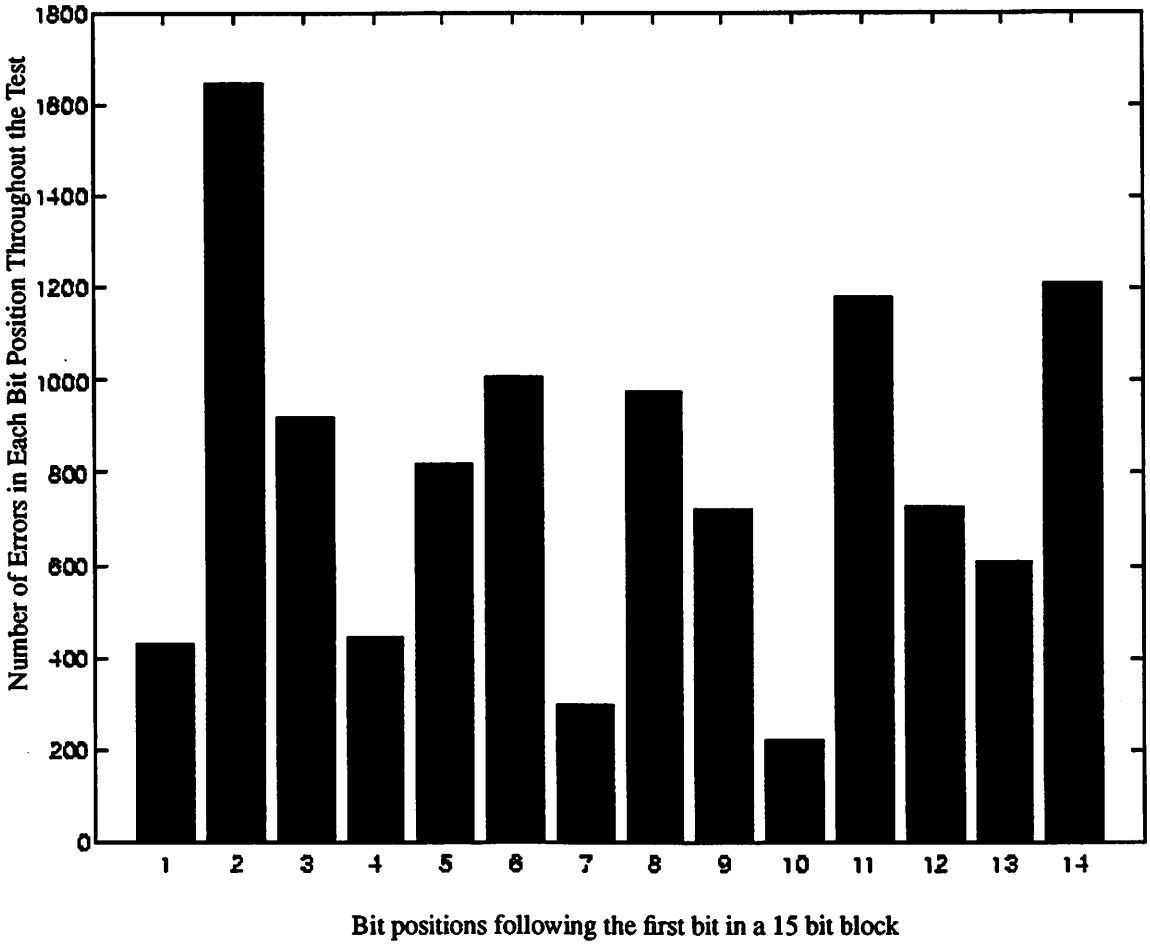


This histogram shows the distribution of bit errors in 15 bit blocks given that the first bit in the block is an error (uplink). This is a histogram of the following 14 bits in the 15 bit block.

Given one bit error, the histogram above (Figure 4-3) shows the number of occurrences of errors in the subsequent 14 bits of the block on the uplink. This data was accumulated with the interference test setup described above over 10,000 packets. One can see that the errors are quite bursty in nature. Given this data, it appears that even the maximum

level of error correction would not be sufficient, because following the first error of a 15 bit block, it appears very likely that the first three bits following that error will also have errors. When the burst of errors exceeds lengths of 3 errors, problems occur with the decoding when using short BCH codes without interleaving.

Figure 4-4. Histogram of Downlink Errors



This histogram shows the distribution of bit errors in 15 bit blocks given that the first bit in the block is an error (downlink). This is a histogram of the following 14 bits in the 15 bit block.

The errors on the downlink appear to be extremely bursty in nature. From this data (Figure 4-4), the most striking data is the fact that given an error at the beginning of a 15 bit block, it is very likely that the second bit following that error will also contain an error.

This again indicates that medium error correction would be less suitable than maximum error correction in this case.

4.2. Area

Table 4-3. Xilinx Resource Usage for Each Design

Totals	F/G Function Generators (out of 800)	H Function Generators (out of 400)	CLB Flip-flops (out of 800)
Pad	669	109	400
Pad (no EDAC)	491	76	388
Basestation	635	105	362
Base (no EDAC)	507	85	338
Wired	550	94	327

Each module in the RF Xilinx design occupies a certain number of function generators and CLB flip-flops. A breakdown of the modules is shown in the table below to illustrate the relative complexity of each module in the RF Xilinx. Some modules are common to all three designs, but as the mapping varies slightly for even the same module, only the largest value is included in the table below. For instance, the Multi-timer module occupies 22 F&G function generators in the pad design, but 23 in the basestation and wired designs. Hence, the largest value is reported in the table below.

Table 4-4. Xilinx Resource Usage by Module

Module	F & G Function Generators	H Function Generators	CLB Flip-flops
Receive Data Control	148	30	108
Trasmit Data Control	165	53	60
Plessey Transmit Radio Interface	37	9	25
Plessey Receive Radio Interface	116	18	101
Proxim Transmit Radio Interface	0	0	0
Proxim Receive Radio Interface	44	6	38
Wired Mode Transmit Module	0	0	0

Module	F & G Function Generators	H Function Generators	CLB Flip-flops
Wired Mode Receive Module	44	8	43
ARM Interface	110	4	83
Timer/Interrupts	24	0	20
Multi-timer Module	23	3	25
LED Control	5	0	4

The main difference of the EDAC Xilinx design from its predecessor is the addition of the FEC and CRC data paths. For a comparison, resource usage for the RF Xilinx transmit/ receive datapath without error correction and CRCs is shown in Table 4-5.

Table 4-5. Xilinx Resource Use for Tx/Rx Data Path without EDAC

Totals	F/G Function Generators	H Function Generators	CLB Flip-flops
Transmit Data Control	55	14	43
Receive Data Control	84	5	65

Hence, adding CRCs and the two levels of forward error correction impacted the RF Xilinx design by using 4 times the function generators and 1.4 times the flipflops on the transmit path, and 2 times the function generators and 1.7 times the flipflops on the receive path.

4.3. Power

An estimate of the power consumption of the RF Xilinx design was performed. The first step in performing the analysis involved providing power to the basestation with a power supply equipped with an ammeter in series. The system was booted through the serial link with the RF Xilinx design in the test mode. The board consumed 5 V at .8 A while this design was operational regardless of the degree of error correction selected. Next the board was booted without the RF Xilinx design being initialized, and the board consumed 5 V at .7 A. Hence, the RF Xilinx consumes approximately 0.5 W.

Table 4-6. Power Measurements

Test	Voltage	Current	Power
Basestation without RF Xilinx design	5 V	0.7 A	3.5 W
Basestation with RF Xilinx design	5 V	0.8 A	4.0 W
RF Xilinx design alone (calculated from previous measurements)	5 V	0.1 A	0.5 W

5 Conclusion

5.1. Summary

The InfoPad system provides an excellent research platform for characterizing trade-offs in error correction techniques. Not only is the error correction logic implemented with an FPGA providing flexibility for design change, the InfoPad infrastructure allows characterization of the wireless link. This master's project included an integration of a new FPGA design that added error correction and detection into the system. In addition to system integration issues, a QOS protocol was also added to the InfoPad system to allow round trip communication from host to pad and back. This protocol enables the system to feedback information about the amount of errors encountered by the system in order to determine what error correction level is most appropriate for the channel.

Although an error correction system may be a viable option for the InfoPad system, significant improvement was only observed with the maximum error correction level on the downlink channel. While this is an improvement, it is difficult to justify using these short forward error correction codes that can only correct small bursts of errors. Without interleaving, short codes are largely ineffective for correcting burst errors, and the additional bandwidth required can be prohibitive. It is recommended that more analysis be performed to determine the most prevalent burst lengths encountered by the system, and an error correction code should be designed with that in mind. During normal system operation, very few errors were encountered, indicating that error correction is unnecessary. However, mobility and operation of the pad far from the basestation could warrant the error correction, making a scheduler a critical part of the system.

5.2. Future Directions

Further research can be performed using the InfoPad as a test platform in order to characterize the nature of noise bursts in indoor wireless environments. More extensive measurements could be taken, with future emphasis on the performance of the CDMA radio with different levels of error correction. By measuring the channel and acquiring real data that reports the actual errors encountered in the system, this data can be used for stimulus in simulation of different error correction techniques. This could lead to improved knowledge of the actual effect of error correction on the system, which is far more valuable than simulating with a Gaussian noise signal.

In addition to using the InfoPad as a measurement platform, the system feedback protocol introduced in Chapter 2 can be used as hooks to a variable QOS scheduler. As the number of users in a cells increases with the addition of the CDMA radio, bandwidth and power consumption, scheduling will become increasingly important to the successful operation of the InfoPad system.

Bibliography

[ESP] "Implementing CRCs." Embedded Systems Programming. 1992.

[Le95] Le, M.T.; Burghardt, F.; Seshan, S.; Rabaey, J. InfoNet: The networking infrastructure of InfoPad. COMPCON'95. Technologies for the Information Superhighway, San Francisco, CA, USA, 5-9 March 1995).

[LinCostello] Lin and D. Costello, Error Control Coding. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[Ylu93] Yuming Kathy Lu. "Error-control coding with applications for the infopad : research project", Master's Project at UC Berkeley. 1993..

[Ylu96] Yuming Lu; Brodersen, R. "Unified power control, error correction coding and scheduling for a CDMA downlink system." Proceedings IEEE INFOCOM '96, San Francisco, CA, USA, 24-28 March 1996). Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 1996. p. 1125-32 vol.3.

[InfoPad] InfoPad Book. Internal document. Description of InfoPad system. 1995. ~infopad/IP2/arm/IP2.book.

[Ples94] DE6003 Digtial Radio Transceiver Specification. GEC Plessey Semiconductors. Scotts Valley, CA. Jan 1994.

[Prox92] RDA Series Reference Manual. Proxim, Inc. Mountain View, CA. Oct 12, 1992.

[Prox90] RXA Series Reference Manual. Proxim, Inc. Mountain View, CA. Oct 1, 1990.

[Xil1] XACT Viewlogic Interface Users Guide. Xilinx. April 1994.

[Xil2] XACT Libraries Guide. Xilinx. April 1994.

[Xil3] Xilinx User Guide and Tutorials. Xilinx. 1991.

[Xil4] The Programmable Logic Data Book. Xilinx. 1994.

6 Appendix

6.1. Pinout for RF Xilinx at Board Level

Pin #	Init	Oper	Pin #	Init	Oper	Pin #	Init	Oper
1	NC	NC	2	GND	GND	3	NC	NC
4	A16	EDT1	5	A17	O	6		RXEOP
7		RXCRC	8	TDI	RXENCA	9	TCK	RXENCB
10		RXCLK	11		RXP	12		(I/O)
13		(I/O)	14	GND	GND	15		CADDR3
16		RXS	17	TMS	RXD7	18		CADDR2
19		I/O	20		I/O	21		CADDR1
22		RXD6	23		RXD5	24		CADDR0
25	GND	GND	26	VCC	VCC	27		RXD4
28		RXD3	29		RXD2	30		RXD1
31		I/O	32		I/O	33		RXD0
34		DICLK	35		DIDIN	36		DIDONE
37	GND	GND	38		(I/O)	39		(I/O)
40		EIN0	41		EIN1	42		EIN2
43		EIN3	44		EIN4	45		EIN5
46		EIN6	47		EIN7	48	M1	--
49	GND	GND	50	M0	--	51	NC	NC
52	NC	NC	53	NC	NC	54	NC	NC
55	VCC	VCC	56	M2	--	57		DIDOUT
58	HDC	DIRST	59		DIENB	60		ICLK
61		IDIN	62	LDC	IRST	63		IDONE

Pin #	Init	Oper	Pin #	Init	Oper	Pin #	Init	Oper
64		IDOUT	65		(I/O)	66		(I/O)
67	GND	GND	68		IENB	69		TXENCA
70		TXENCB	71		TXP	72		I/O
73		I/O	74		TXS	75		TXD7
76		TXD6	77	/INIT	--	78	VCC	VCC
79	GND	GND	80		TXD5	81		TXD4
82		TXD3	83		TXD2	84		RSSICLK
85		RSSICNVT	86		TXD1	87		TXD0
88		TXCRC	89		NRST	90	GND	GND
91		(I/O)	92		(I/O)	93		RA0
94		RA1	95		RA2	96		RA3
97		RA4	98		RA5	99		RA6
100		RA7	101	GND	GND	102	NC	NC
103	DONE	--	104	NC	NC	105	NC	NC
106	VCC	VCC	107	NC	NC	108	/PROG	--
109	D7	D7	110	(PGCK3)	BIT_CLK	111		RA8
112		RA9	113	D6	D6	114		RA10
115		RA11	116		RA12	117		(I/O)
118		(I/O)	119	GND	GND	120		RA13
121		RA14	122	D5	D5	123	/CS0	/CCS
124		RA15	125		RA16	126		RA17
127		RB0	128	D4	D4	129		RB1
130	VCC	VCC	131	GND	GND	132	D3	D3
133	/RS	/RS	134		RB2	135		RB3
136		RB4	137		RB5	138	D2	D2
139		RB6	140		RB7	141		RB8
142	GND	GND	143		(I/O)	144		(I/O)
145		RB9	146		RB10	147	D1	D1
148	RDY	INTOUT	149		RB11	150		RB12
151	D0	D0	152	DOUT	O	153	CCLK	--
154	VCC	VCC	155	NC	NC	156	NC	NC

Pin #	Init	Oper	Pin #	Init	Oper	Pin #	Init	Oper
157	NC	NC	158	NC	NC	159	TDO	--
160	GND	GND	161	A0/WS	/WS	162	A1	EDC1
163		EDC0	164		EDC2	165	A2/CS1	--
166	A3	EDC3	167		RB13	168		RB14
169		(I/O)	170		(I/O)	171	GND	GND
172		RB15	173		RB16	174	A4	EDC4
175	A5	EDC5	176		RB17	177		TST0
178		TST1	179		TST2	180	A6	EDC6
181	A7	EDC7	182	GND	GND	183	VCC	VCC
184	A8	EDC8	185	A9	EDC9	186		TST3
187		TST4	188		TST5	189		TST6
190	A10	EDC10	191	A11	EDC11	192		TST7
193		MCLK	194	GND	GND	195		(I/O)
196		(I/O)	197		I/O	198		I/O
199	A12	EDC12	200	A13	EDC13	201		I/O
202		I/O	203	A14	EDC14	204	A15	EDT0
205	VCC	VCC	206	NC	NC	207	NC	NC
208	NC	NC	209			210		

6.2. Pinout of RF Xilinx Functional Design

Pin #	I/O	Name	Description
4	O	ROMADDR[16]	Address to the Lookup Table in the EPROM
6	I	RXEOP	End of Received Packet Signal
7	I	RXCRC	Receive CRC Enable
8	I	RXENCA	Receive Error Correction Level Select (RXECC[1})
9	I	RXENCB	Receive Error Correction Level Select (RXECC[0})
10	O	RXCLK2	Receive Data Clock
11	O	RXP	Receive Packet Signal
15	I	CADDR[3]	Address line from the ARM
16	O	RXS	Receive Data Byte Strobe
17	O	RXD[7]	Received Data
18	I	CADDR[2]	Address line from the ARM
21	I	CADDR[1]	Address line from the ARM
22	O	RXD[6]	Received Data
23	O	RXD[5]	Received Data
24	I	CADDR[0]	Address line from the ARM.
27	I	RXD[4]	Received Data
28	O	RXD[3]	Received Data
29	O	RXD[2]	Received Data
30	O	RXD[1]	Received Data
33	O	RXD[0]	Received Data
40	I	ROMDATA[0]	Data from the Lookup Table EPROM
41	I	ROMDATA[1]	Data from the Lookup Table EPROM
42	I	ROMDATA[2]	Data from the Lookup Table EPROM
43	I	ROMDATA[3]	Data from the Lookup Table EPROM
44	I	ROMDATA[4]	Data from the Lookup Table EPROM
45	I	ROMDATA[5]	Data from the Lookup Table EPROM
46	I	ROMDATA[6]	Data from the Lookup Table EPROM
47	I	ROMDATA[7]	Data from the Lookup Table EPROM
69	I	TXENCA	Transmit Error Correction Level Select (TXECC[1])
70	I	TXENCB	Transmit Error Correction Level (TXECC[0])
71	I	TXP	Transmit Packet Signal
75	I	TXD[7]	Transmit Data
76	I	TXD[6]	Transmit Data
80	I	TXD[5]	Transmit Data
81	I	TXD[4]	Transmit Data
82	I	TXD[3]	Transmit Data

Pin #	I/O	Name	Description
83	I	TXD[2]	Transmit Data
84	O	RSSICLK	Data Strobe for the A/D converter used to obtain RSSI data
85	O	RSSICNVT	RSSI Conversion Start Signal. Initiates a conversion in the A/D.
86	I	TXD[1]	Transmit Data
87	I	TXD[0]	Transmit Data
88	I	TXCRC	Transmit CRC Enable
100	I	PR_TXCLK	Proxim Radio Transmit Clock
109	I/0	D[7]	ARM Data input/output bus
113	I/0	D[6]	ARM Data input/output bus
114	O	PR_DATA	Serial Transmit Data for the Proxim Radio
115	O	PR_SYNLD	Active Low Load Line for the Proxim Radio
116	O	PR_SYNDATA	Serial Data for Programming the Proxim Radio
120	I	D0IN[3]	Input to the D0IN3 register which is not implemented in this design.
121	O	PR_TX_RX	Proxim Radio Transmit/Receive Mode Select
122	I/O	D[5]	ARM Data input/output bus
123	I	/CS0	Chip Select for the RF Xilinx from the ARM
124	O	PR_SYNCLK	Data Strobe for Programming the Proxim Radio
125	O	PR_STDBY	Proxim Standby Signal
126	I	RA17	Serial Input RSSI Measurement Data from A/D Converter
127	O	PL_POW_LEVEL	Plessey Radio Power Level Select
128	I/O	D[4]	ARM Data input/output bus
129	O	PL_ANTSEL	Plessey Antenna Select
132	I/O	D[3]	ARM Data input/output bus
133	I	/RS	Read Strobe from the ARM. Not used in this design.
134	O	PL_TX_RX	Plessey Transmit/Receive Mode Select
135	I	CR_DIN	Serial Receive Data from the Plessey Radio
136	O	PL_PA_OFF	Plessey Power Amplifier On/Off Select
138	I/O	D[2]	ARM Data input/output bus
140	O	PL_TXD	Plessey Transmit Data
141	O	PL_STDBY	Plessey Standby signal
145	O	PL_SD[0]	Plessey Programming Data
146	O	PL_SD[1]	Plessey Programming Data
147	I/O	D[1]	ARM Data input/output bus
148	O	INTOUT	Interrupt to the ARM for Timer and RSSI events
149	O	PL_SD[2]	Plessey Programming Data
150	O	PL_SD[3]	Plessey Programming Data
151	I/O	D[0]	ARM Data input/output bus

Pin #	I/O	Name	Description
161	I	WS	Write Strobe from the ARM
162	O	ROMADDR[1]	Address to the Lookup Table in the EPROM
163	O	ROMADDR[0]	Address to the Lookup Table in the EPROM
164	O	ROMADDR[2]	Address to the Lookup Table in the EPROM
166	O	ROMADDR[3]	Address to the Lookup Table in the EPROM
167	O	PL_SD[4]	Plessey Programming Data
168	O	PL_SD[5]	Plessey Programming Data
172	O	PL_SD[6]	Plessey Programming Data
173	O	PL_LOADB	Plessey Load Line
174	O	ROMADDR[4]	Address to the Lookup Table in the EPROM
175	O	ROMADDR[5]	Address to the Lookup Table in the EPROM
176	O	TG_RESET	Text Graphics Active Low Reset. Used for Debugging Purposes.
177	O	LED[0]	Heartbeat signal for LED (D301) on the board
178	O	LED[1]	Transmit Packet signal driving LED (D302) on the board
179	O	LED[2]	Receive Packet signal driving LED (D303) on the board
180	O	ROMADDR[6]	Address to the Lookup Table in the EPROM
181	O	ROMADDR[7]	Address to the Lookup Table in the EPROM
184	O	ROMADDR[8]	Address to the Lookup Table in the EPROM
185	O	ROMADDR[9]	Address to the Lookup Table in the EPROM
186	O	LED[3]	Programmable LED (D304) on the board. Driven by XIL-MISC[4] in the RF Xilinx register file.
187	O	LED[4]	Programmable LED (D305) on the board. Driven by XIL-MISC[3] in the RF Xilinx register file.
188	O	LED[5]	Programmable LED (D306) on the board. Driven by XIL-MISC[2] in the RF Xilinx register file.
189	O	LED[6]	Programmable LED (D307) on the board. Driven by XIL-MISC[1] in the RF Xilinx register file.
190	O	ROMADDR[10]	Address to the Lookup Table in the EPROM
191	O	ROMADDR[11]	Address to the Lookup Table in the EPROM
192	O	LED[7]	Programmable LED (D308) on the board. Driven by XIL-MISC[0] in the RF Xilinx register file.
193	I	CLK20	20 MHz System Clock
199	O	ROMADDR[12]	Address to the Lookup Table in the EPROM
200	O	ROMADDR[13]	Address to the Lookup Table in the EPROM
203	O	ROMADDR[14]	Address to the Lookup Table in the EPROM
204	O	ROMADDR[15]	Address to the Lookup Table in the EPROM

6.3. Register Descriptions

The basestation, pad, and wired designs of the RF Xilinx have slightly different register configurations. Hence, the register files for each design are described in separate sections.

6.3.1. Basestation Registers

Table 6-1. Register 0x00 - XILRXSA, XILSTAT

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Least Significant Byte of Receive Sync Word							
R						RSSI Interrupt Mask	Timer Interrupt Mask	

Table 6-2. Register 0x01 - XILRXSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Most Significant Byte of Receive Sync Word							
R	Least Significant Byte of RSSI Value							

Table 6-3. Register 0x02 - XILTXSA, XILRSSIMSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Least Significant Byte of Transmit Sync Word							
R	Most Significant Byte of RSSI Value							

Table 6-4. Register 0x03 - XILTXSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Most Significant Byte of Tx Sync Word (XILTXSB)							

Table 6-5. Register 0x05 - XILRXFB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W			Rx Frequency Register (Proxim Control)					
			Ready	Stdby	Tx/Rx	Syn-Data	Syn-Clk	SynLd

Table 6-6. Register 0x07 - XILTXFB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W		Tx Frequency Register (Plessey Control)						

Table 6-7. Register 0x10 - XILCNTL

R/W	Bit position							
	7	6	5	4	3	2	1	0
W				TG Reset	RX Kick	Loop-back	Multi-Timer Reset	Sys Reset

Table 6-8. Register 0x11 - XILRXC

R/W	Bit position							
	7	6	5	4	3	2	1	0
W				Rx Clock Divider Word				

Table 6-9. Register 0x12 - XILTXC

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Hop Reset		Tx Clock Divider Word					

Table 6-10. Register 0x13 - XILTIMERLSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Least Significant Byte of Timer							

Table 6-11. Register 0x14 - XILTIMERMSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Most Significant Byte of Timer							

Table 6-12. Register 0x15 - XILCMDREG

R/W	Bit position							
	7	6	5	4	3	2	1	0
W						RSSI Ack	Timer Ack	

Table 6-13. Register 0x17 - XILMISCREG

R/W	Bit position							
	7	6	5	4	3	2	1	0
W				LED Register				

6.3.2. Pad Registers

Table 6-14. Register 0x00 - XILRXSA, XILSTAT

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Least Significant Byte of Receive Sync Word							
R						RSSI Interrupt Mask	Timer Interrupt Mask	

Table 6-15. Register 0x01 - XILRXSB, XILRSSILSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Most Significant Byte of Receive Sync Word							
R	Least Significant Byte of RSSI Value							

Table 6-16. Register 0x02 - XILTXSA, XILRSSIMSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Least Significant Byte of Transmit Sync Word							
R	Most Significant Byte of RSSI Value							

Table 6-17. Register 0x03 - XILTXSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Most Significant Byte of Tx Sync Word							

Table 6-18. Register 0x05 - XILRXFB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W			Rx Frequency Register (Plessey Control)					

Table 6-19. Register 0x07 - XILTXFB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W			Tx Frequency Register (Proxim Control)					
			Ready	Stdby	Tx/Rx	Syn-Data	Syn-Clk	SynLd

Table 6-20. Register 0x10 - XILCNTL

R/W	Bit position							
	7	6	5	4	3	2	1	0
W				TG Reset	RX Kick	Loop-back	Timer Reset	Sys Reset

Table 6-21. Register 0x11 - XILRXC

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Hop Reset			Rx Clock Divider Word				

Table 6-22. Register 0x12 - XILTXC

R/W	Bit position							
	7	6	5	4	3	2	1	0
W			Tx Clock Divider Word					

Table 6-23. Register 0x13 - XILTIMERLSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Least Significant Byte of Timer							

Table 6-24. Register 0x14 - XILTIMERMSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Most Significant Byte of Timer							

Table 6-25. Register 0x15 - XILCMDREG

R/W	Bit position							
	7	6	5	4	3	2	1	0
W						RSSI Ack	Timer Ack	

Table 6-26. Register 0x17 - XILMISCREG

R/W	Bit position							
	7	6	5	4	3	2	1	0
W				LED Control Register				

6.3.3. Wired Registers

Table 6-27. Register 0x00 - XILRXSA, XILSTAT

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Least Significant Byte of Receive Sync Word							
R						RSSI Inter-rupt Mask	Timer Inter-rupt Mask	

Table 6-28. Register 0x01 - XILRXSB, XILRSSILSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Most Significant Byte of Receive Sync Word							
R	Least Significant Byte of RSSI Value							

Table 6-29. Register 0x02 - XILTXSA, XILRSSIMSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Least Significant Byte of Transmit Sync Word							
R	Most Significant Byte of RSSI Value							

Table 6-30. Register 0x03 - XILTXSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Most Significant Byte of Tx Sync Word							

Table 6-31. Register 0x10 - XILCNTL

R/W	Bit position							
	7	6	5	4	3	2	1	0
W				TG Reset	RX Kick	Loop-back	Timer Reset	Sys Reset

Table 6-32. Register 0x11 - XILRXC

R/W	Bit position							
	7	6	5	4	3	2	1	0
W				Rx Clock Divider Word				

Table 6-33. Register 0x12 - XILTXC

R/W	Bit position							
	7	6	5	4	3	2	1	0
W			Tx Clock Divider Word					

Table 6-34. Register 0x13 - XILTIMERLSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Least Significant Byte of Timer							

Table 6-35. Register 0x14 - XILTIMERMSB

R/W	Bit position							
	7	6	5	4	3	2	1	0
W	Most Significant Byte of Timer							

Table 6-36. Register 0x15 - XILSTAT

R/W	Bit position							
	7	6	5	4	3	2	1	0
W						RSSI Ack	Timer Ack	

Table 6-37. Register 0x17 - XILMISC

R/W	Bit position							
	7	6	5	4	3	2	1	0
W				LED Control Register				

6.4. Location of Design Files

The entire design is placed on the infopad account at *~infopad/IP2/rfxil_ecc*. Under this directory is a makefile that will make the wired, basestation, and pad designs simply by typing "*make all.*" A *README* file is also present that includes a description of files and directories. This report and the master's report of Kathy Lu are also included in the *doc* subdirectory. A simulation (in ViewSim) of the transmit and receive data path with injected errors is also included in this directory under the subdirectory *rxtx_sim.*

6.5. Design Libraries

It is important to recall that these designs were compiled with the XC4000 libraries. If the Xilinx libraries are upgraded, these libraries should be archived to maintain compatibility with the old design in the event that the design is not upgraded to the new libraries.