6.111 Introductory Digital Systems Laboratory

FINAL PROJECT

CDMA Forward Channel Receiver

Authors:
Zackary Anderson
Russell Ryan

TA:
David Wentzloff

Abstract

IS-95A Code-Division Multiple-Access (CDMA) is a prevalent RF modulation scheme and protocol that is used by many cellular telephone providers. The following project uses software radio to inspect forward control channel data of an IS-95A network. The system communicates serially with an RF front-end called the USRP. A Field Programmable Logic Array (FPGA) processes USRP data in order to retrieve Sync channel data. The system demodulates and decodes the signal, automatically discovers the current pseudonoise sequence, derepeats and deinterleaves data, performs error correction with a Viterbi filter, and then relays the processed data to a computer for reporting.

May 17, 2007

Contents

List of Figures

1 Introduction

Many modern cell phone providers use a form of RF modulation known as Code-Division Multiple-Access (CDMA) to support several mobile users on the same frequency band. One standard, known as IS-95A, is used by many popular networks such as Verizon. This project aims to design a digital system capable of acquiring the signal of an IS-95A base station and decoding the system-synchronization information it broadcasts. With this information, it is possible to extend the system to incorporate interception of voice traffic. To accomplish the proposed goals, a full forward-channel receiver was designed and built.

1.1 CDMA Technical Overview

CDMA is a type of Direct-Sequence Spread Spectrum (DSSS) modulation which allows the multiplexing of many different channels of data on a single frequency with only 1.25MHz bandwidth. DSSS modulation allows the transmission of data by 'spreading' it across the bandwidth using two 2^{15} bit-long pseudo-random noise sequences known as Short PN Codes. These PN codes are used to encode both the In-Phase (I) and Quadrature (Q) components of the signal using Phase Shift Keying (PSK).

The short PN codes modulate data at a chipping rate of 1.228 million chips-per-second (cps). Chips refer to "bits" of *modulated* data, several of which equate to a single data bit. At this rate, the sequences repeat once every 26.7ms. Each base station is identified uniquely by its 64-bit starting offset into each short PN sequence. This allows up to 512 base stations to transmit on the same CDMA frequency.

Separate data channels on a base station are supported by spreading each channel with mutually orthogonal, binary sequences known as Walsh Codes. Since all the channels are orthogonal to each other, one can select the desired data merely by despreading the data with the appropriate Walsh Code.

For a mobile station to acquire a lock on a base station, it must synchronize its short PN code generators to the target base station's generators. To help facilitate this, each

base station transmits a 'pilot' channel, which contains nothing but zeroes as its base-band data. A mobile station attempts to decode this channel repeatedly, trying new shifts of the short PN codes until it decodes the transmitted zeroes. Since a zero-Walsh code is used for the Pilot channel, the transmitted Pilot channel looks like the PN sequence being used to modulate it.

The base station also transmits three other types of channels; Paging, Sync, and Traffic. The sync channel contains various pieces of information required to decode both the Paging (another type of Control channel) and Traffic (voice) channels. It also contains various parameters of the base station, such as the System ID of the Provider.

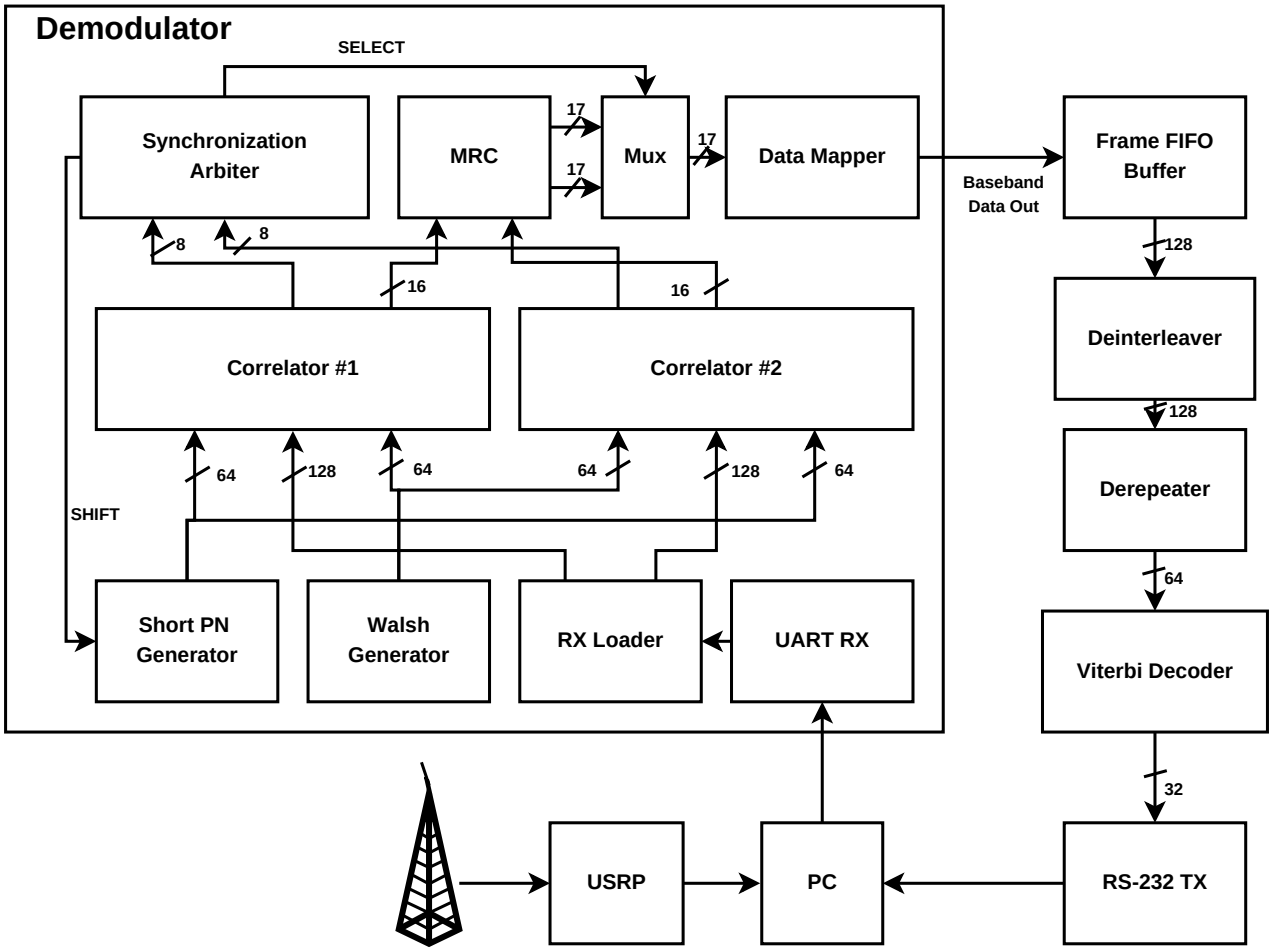

Figure 1: An overall block diagram of the IS-95A Forward Channel Receiver.

1.2 Design Overview

The design of the IS-95A receiver is divided into three parts: Data Acquisition, Demodulation, and Processing. To acquire signal data, a Universal Software Radio Peripheral (USRP) is employed. This device streams In-Phase and Quadrature signals over USB to a PC, which in turn, transmits the data to the FPGA over a high-speed serial connection.

Once data arrives at the FPGA from the PC, the demodulation process begins, as is depicted in Figure 1. The data, which is oversampled at the Nyquist rate, is passed to two correlators which attempt to verify a lock on the pilot channel. A synchronization arbiter determines whether the correlation presented by either correlator is valid. If neither are valid, it signals to the short PN code generator to shift forward a chip in the PN sequence, and the process repeats. Once a correlator acquires a lock, the system attempts to receive data on the Sync channel.

The data from the correlator undergoes Maximal Ratio Combining (MRC), and is buffered for the length of a Sync channel frame. The data is deinterleaved and derepeated before it is processed by the Viterbi decoder, which decodes the convolutionally encoded data, producing the frame payload. The result is transmitted back to the PC via serial, where pertinent information is parsed and displayed to the user.

2 Design Implementation

2.1 Demodulation Module

The very first step in data processing is demodulation. Before any further processing can occur on the data coming from the RF-frontend, it must be demodulated. Data enters the FPGA in quadrature-format at 2.456 Mbs (twice the chipping rate). This data is demodulated using a number of different sub-modules in order to produce a base-band data output at a rate of 19.2 kbs.

The forward-link for an IS-95A system is Quadrature Phase Shift Keyed (QPSK).

Demodulation begins with unsynchronized in-phase (I) and quadrature-phase (Q) values from the USRP device. To avoid timing problems, Nyquist contraints are met by sampling the frequency at twice the chipping rate. This gives two data points for every transmitted chip. This data is treated as being interleaved, that is, the demodulation module sends every other I,Q sample to one of two correlators. At least one of these two correlators will obtain a lock when the correct PN sequence is discovered. The first correlator to obtain a lock on the Pilot channel is therein used exclusively until an unsynchronized state is reached again.

The overall concept behind the demodulation block is as follows:

1. Transmit 64 independent, interleaved samples to each correlator such that each correlator works on every other sample. This is just enough samples to successfully demodulate a single bit of baseband data.
2. Generate Walsh index 32 and the short in-phase PN sequence and short quadrature-phase PN sequence.
3. Accumulate samples XORed with the appropriate PN and Walsh codes (over a period of 64 chips).
4. Wait until the Synchronization Arbiter measures an exceeded threshold on the accumulated Pilot channel data.
5. If synchronized, then use Maximal Ratio Combining (MRC) to combine synchronized I and Q data. If not synchronized, then shift PN generator sequences.
6. Map MRC output to a bit of data.

Figure 1 illustrates the interactions between different modules in the demodulation block.

2.1.1 Correlator

The correlator module is at the very heart of the demodulation procedure. It accumulates the combination of the PN sequence, Walsh code for the given channel, and input data over the period of 64 chips (one full data bit). The following equation, where I and Q are iterated over 64 chips, governs the general IS-95A accumulator:

$$I = \sum_{i=0}^{63}(PN_I(i) * WALSH_n[i] * DATA_I) + (PN_Q(i) * WALSH_n[i] * DATA_Q)$$

$$Q = \sum_{i=0}^{63}(PN_I(i) * WALSH_n[i] * DATA_Q) - (PN_Q(i) * WALSH_n[i] * DATA_I)$$

The above equation assumes values are in polar format where a binary "0" is represented as 1 and a binary "1" is represented as -1. In implementation, a simple mux executes this conversion. Since the described system listens to two channels, the above equation is used for both the Pilot and Synchronization channels, with merely a different Walsh code (n) for each channel. Since the Pilot channel uses an all "1" polar Walsh code, multiplication does nothing and is therefore omitted from the code.

After 64 chips have been summed, the accumulated value is latched to the output and stays valid until the next 64 chips have been correlated with the PN sequence and Walsh code.

2.1.2 Synchronization Arbiter

The Synchronization Arbiter is a feedback system that employs two correlators and the controllable PN sequence generator to match the local PN sequence with the transmitting base station. In addition, it chooses the most accurate correlator and forces the system to exclusively use the chosen correlator for data acquisition. If the system should drift over time and lose its synchronization, the Arbiter will re-initiate this process.

In order to synchronize with the base station, the Arbiter must know how to identify a synchronized state. This is accomplished by comparing the Pilot channel accumulator to a large, preset parameter. When this threshold is met, the system assumes coordination.

This policy works because the pseudonoise sequence (in polar form) averages to zero. The received signal also averages to zero. By multiplying the two unsynchronized signals, a very low number will accumulate over time. The Pilot channel is constantly sending a stream of data comprised of zeros. After modulation, this is merely the PN sequence that is transmitted. When the receiver is in perfect synchronization with the base sta-

tion sequence, the four (I and Q for each participant) PN sequences will be aligned. Multiplying these identical PN sequences together will yield a very high number on the accumulator. The correlator output literally goes from low amplitude noise to a constantly peaking high value. This makes it very clear to the Arbiter that a particular correlator has synchronized.

The Arbiter at this point sets the multiplexer between the Maximal Ratio Combiner and the Data Mapper to receive data from the appropriate correlator.

In order to accomplish this alignment of PN sequences, the current sequence is shifted forward by one extra bit each 64-chip period if the correlator does not reach the preset threshold. By shifting forward one extra bit, the mobile station quickly hits the correct portion of the looping PN sequence the base station is using.

Since the PN sequence used is the same for the other channels, synchronizing on the Pilot channel also synchronizes reception on the Sync channel.

2.1.3 Maximal Ratio Combining

The maximal ratio combining block takes the synchronized and correlated I and Q values for the Sync channel and produces a single output. Since the short PN sequence has been synchronized with the modulated data coming into the FPGA, the mathematics work out such that the synchronized, QPSK modulated channels can be multiplied together and then summed.

The following equation is used in the Maximal Ratio Combining portion of the demodulator block to combine synchronized I and Q components:

$$MRC = (Pilot\_I_a * Sync\_I_a) + (Pilot\_Q_a * Sync\_Q_a)$$

where Pilot_I, Sync_I, Pilot_Q, and Sync_Q are all the accumulated outputs given by the correlator module.

This MRC algorithm is performed twice: once for each correlator. A multiplexer controlled by the Synchronization Arbiter selects which MRC output ties into the Data

Mapper block.

2.1.4 Data Mapper

The Data Mapper transforms the 17-bit signed input from the MRC module into a single bit output. This is done based on the sign of the value. The module was designed based on the assumption that a positive value maps to a "1" and a negative or zero value to a "0."

2.1.5 Walsh Generator

The Walsh Generator provides the unique 64-bit code associated with a given channel. There are a total of 64 Walsh codes, all of which are orthogonal. This allows the code to select a single channel. The two codes currently used by the described system are $WALSH_0$ and $WALSH_{32}$ for the Pilot and Sync channels, respectively. Since the system was designed and built to scale into a full CDMA receiver system, the Walsh generator is able to produce all 64 codes. Careful study of the 64x64 Walsh table reveals a simple pattern:

$$WALSH_n[i] = ((\alpha[0] \oplus \alpha[1]) \oplus (\alpha[2] \oplus \alpha[3])) \oplus (\alpha[4] \oplus \alpha[5])$$

where $\alpha[j]$ is the jth bit of

$$\alpha = i\&n$$

and where n is the Walsh index and i is the ith bit of the Walsh sequence.

The Walsh Generator uses this equation and cycles through each of the 64 bits for a given Walsh code and then latches the output to a register that stays valid for the next 64 cycles.

2.1.6 Short PN Sequence Generator

The short PN sequence generator generates two, 2^{15}-bit long pseudo-random noise sequences. One sequence is known as the In-Phase short PN Code and the other is known as the Quadrature Short PN Code. Both sequences repeat after 2^{15} digits. These are used as described in §1.1 to multiplex the operation of many base stations over a single frequency using DSSS. The two PN sequences are generated using the following generator polynomials, respectively:

$$PN_I(t) = x^{15} + x^{13} + x^9 + x^8 + x^7 + x^5 + 1$$

$$PN_Q(t) = x^{15} + x^{12} + x^{11} + x^{10} + x^6 + x^5 + x^4 + x^3 + 1$$

To implement each PN code generator, two 15-bit Linear Feedback Shift Registers (LFSR) are used. For each LFSR, if x^n is a term in the generator polynomial, then the n'th bit of the register is used as a tap for generating the next bit in the sequence. Both polynomials are 'maximal' which means that they will cycle through all $2^n - 1$ possible values of the shift register, except for when the register is in a state of all zeroes.

The short PN generator module provides buffered outputs of the last 64-bit wide frame of bits generated. This output is held at its value until the next 64 bits of the PN_I and PN_Q sequence have been generated. As described in §2.1.2, a shift input is provided by the Synchronization Arbiter which indicates that the two PN sequences must be shifted forward by an extra bit. In this case, the normal LFSR shift process is simply performed twice.

2.2 Serial Interface Modules

A serial interface provides for both data input and output from the FPGA. A high-speed UART receiver is used to facilitate USRP data acquisition. A standard RS-232 interface is used to transmit decoded data to a computer for parsing and display.

2.2.1 RS-232 Transmitter

The RS-232 transmitter interfaces with the labkit DB9 port for communication from the FPGA to a computer. Serial transmission involves transmitting a stop bit of zero, serially sending eight bits of data, and then raising the line for a stop bit. The line then idles high while no data is present.

Serial transmission is accomplished via a state machine that alternates between an idle state with a high output, the start bit with a low output, and the transmit state which ties the output to the lower order bit of a latched input. On each clock in the transmit state the latched input is right shifted, causing the next bit to be transmitted on the line. This progresses until eight bits are sent, and then state switches to the idle state. Since this transition to idle is at least one clock period long, it accomplishes the stop (high) bit restraint.

In order to transmit at the correct frequency, the module is clocked on a 9600 Hz clock for 9600 bps transmission. Since the sync channel outputs decoded data at a rate of 1200 bps, this is more than enough speed.

2.2.2 UART Receiver

While the design for the UART receiver is similar to the transmitter in many ways, it is slightly more complex. A UART receiver has to be able to synchronize with the asynchronous transmitter. The described system accomplishes this by oversampling the UART line at sixteen times the baud-rate. When the line is first detected going low, the system switches to the **CENTER** state. This waits half a baudrate in order to align with the center of the bit transmission. The line is then sampled again to verify that it is still low. If not, the state machine switches back to **IDLE**, assuming there was a false alarm. If it is indeed a start bit, the state machine then switches to **STALL**, where it waits for one baudrate, at which point it should be in the center of the first transmitted bit. It samples the bit and then stalls again. The **SAMPLE** state shifts the read bit into a byte-long register. The **STALL** and **SAMPLE** state transitions repeat for eight

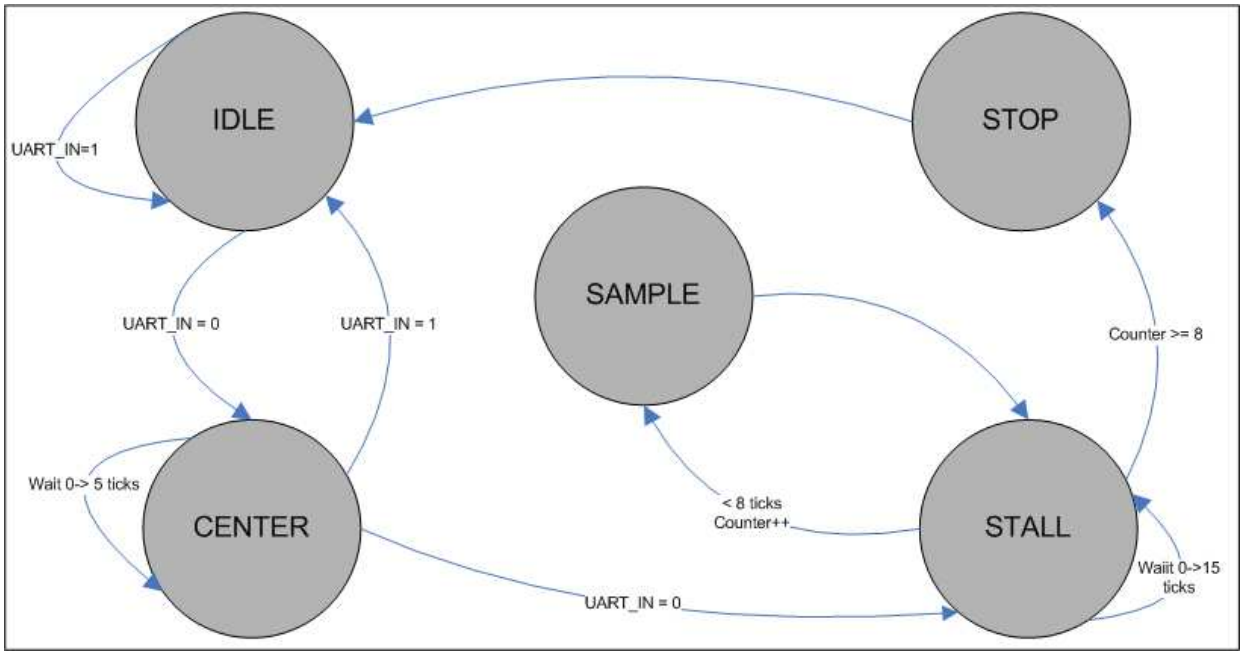

Figure 2: A state transition diagram of the UART Receiver Module.

data bits, at which point it switches to a **STOP** state. This is no more than a delay at which point the system triggers the *byte_ready* indicator. State immediately returns to **IDLE** and the module waits for the entire process to occur again.

2.3 Deinterleaving and Derepeating Module

Data transmitted on an IS-95A system is repeated and then interleaved after convolutional encoding. Repetition on the Sync channel involves duplicating every transmitted bit (i.e. "1010" yields "11001100"). Interleaving then spreads these bits so as to minimize the chance that a burst of noise will completely distort a transmitted bit. The interleaving on the transmit side involves a lookup table that maps each input bit to a particular output bit. Interleaving, and thus deinterleaving, only operates on an entire frame of data. The lookup table is published with the IS-95A standard.

In order to undo these two transmit steps, the Deinterleaver/Derepeater module first buffers an entire frame of data. Between the demodulator block and the deinterleaver block, this involves a 128-bit first-in-first-out (FIFO) buffer. The buffer shifts new data

into a 128-bit wide temporary register and then latches the temporary register to a registered output every frame.

Once a full frame is ready to be deinterleaved and derepeated, the FIFO buffer signals the Deinterleaver/Derepeater module. The Deinterleaver/Derepeater module then loads each bit from the register, starting from the smallest, into the corresponding bit-position of a temporary output register. This is accomplished by addressing the output register with the output from a read-only memory (ROM). This ROM is pre-loaded with a .COE file that contains a custom deinterleaving table. This table maps inputs to appropriate outputs, but while doing so it scraps repeated bits by assigning them to the same location. By doing this, the module can deinterleave and derepeat the 128-bit input in 128 clock cycles, and output a 64-bit value. This value is then passed to the Viterbi Decoder for further processing.

2.4 Viterbi Decoder Module

On an IS-95A forward channel, the data of the Sync, Traffic, and Paging channels are all convolutionally encoded with a rate $\frac{1}{2}$ encoder with a constraint length $\mathbf{K} = 9$. The half data rate means that for every bit input into the encoder, two bits are generated. This convolutional encoding step is performed to achieve Forward Error Correction of the datastream. Convolutional encoding allows the receiving mobile station to detect and correct errors that occur during transmission, and is paramount to the quality of IS-95A transmissions.

The Viterbi Decoder is a maximum-likelihood detector which given a frame of size $2n$ bits, provides the most-likely series of n bits input into the convolutional encoder which generated that sequence given that multiple errors could have occured in the transmission process. In this implementation, it decodes an encoded Sync channel frame of 64 bits and produces the decoded 32 bit value of the frame.

In hardware, the decoding consists of three main processes: Branch Metric Calculation, Path-Metric Update, and Path Traceback.

2.4.1 Branch Metric Calculation

A $K = 9$ Viterbi Decoder consists of 256 internal states for every input bit processed. Branch metrics are simply measures of the hamming code distance between any combination of two bits with two input bits. The semantics of branch metric calculation can be pre-computed and stored as parameters in each of the 256 states. Each state is modeled as an 'Add-Compare-Select' (ACS) Unit, and is instantiated with the pre-computed parameters.

2.4.2 Path-Metric Update

The path metric update process is performed by 256 parallel ACS units. This process is designed to compute the maximum path metric for every path through the tree.

The ACS units take the input path metric from two incident edges of previous ACS states and combine both with the next two input bits. Whichever incident node's branch metric result is bigger becomes the branch metric output of the current state. The path with the largest path metric is thus calculated with a greedy dynamic programming algorithm. As the path is decided, the previous state of each node is recorded in registers, as well as a 'hard decision', or bit value for the current state.

2.4.3 Path Traceback Processing

After the path metric step finishes, the decoder enters traceback mode. The first step of this mode is to calculate the maximum path metric for the final set of 256 ACS states. This is accomplished by building a 256 input 7-bit maximum value detector out of cascaded 2-input 7-bit maximum value detectors, named pm_compare modules.

Once the maximum end state is determined, the traceback begins. Bits of the output sequence are filled from high to low as the traceback proceeds from the last stage to the first. At each state, the stored 'hard decision' is output as the bit value for that position, and the previous state information is used to traverse backward to the previous stage. The decoding is complete when the traceback finishes, and the decoder raises its *done*

signal to indicate that the processing stage is done.

2.5 PC Frontend

The purpose of the PC frontend module is to provide RF data to the IS-95A demodulator, and to receive the processed data from the demodulator after the process is finished. RF data is acquired using a Universal Software Radio Peripheral (USRP) with a daughterboard designed to receive on frequencies used in IS-95A.

2.5.1 Radio Data Acquisition

The USRP is connected to the PC via USB. The open source GNURadio project is used on a Linux platform to communicate and stream data from the USRP to the PC. A Python script initializes the USRP and tunes it an IS-95A forward channel frequency. The script then receives the RF In-Phase and Quadrature signals from the USRP at a $2x$ oversampling rate, re-formats them, and transmits them to the FPGA via the UART serial interface described in §2.2.

2.5.2 Data Post-processing and Display

A receiver that synchronizes to a base station and receives Sync channel data is of little use without some method of communicating with the user. After a frame of data is Viterbi-decoded, the 32 bits are sent to a computer via RS-232. The computer buffers this frame and the next two frames of data to acquire a single synchronization channel super-frame. This twelve-byte super-frame may contain several parameters such as a universal coordinated time, the network and system identifiers, long PN code offset, and several parameters indicating the type of message that is being transmitted.

A Python script buffers and then parses this super-frame, displaying to the user Sync channel data as it arrives. Since different types of messages are constantly transmitted over the Sync channel, the PC front-end is regularly displaying new pieces of data to the user.

3 Testing and Debugging

3.1 General Methodology

The implementation of a working forward-channel receiver is highly complex and dependent on a number of independent modules a) being designed based on sound assumptions, and b) being correctly implemented. Ensuring that individual components can not only be thoroughly tested using test-bench software, but also on the actual FPGA, is critical.

Three major simplification factors are used to cope with the inherent complexity of the receiver:

1. extensive ModelSim computer-end simulation to test theoretical functionality,
2. a "serial test-bench" which allows components to process data on the FPGA and dump the results to a terminal, and
3. a highly modular design.

Since the described CDMA receiver is primarily a signal processing system, all testing essentially comes down to placing input and reading output. A piece of serial-interface verilog was written early on during the project. This allows for a very easy tie in for testing individual components. With this code, one can test components with a computer terminal. The highly-modular nature of each component in the signal processing datastream further allows for such localized testing.

It became readily apparent early on in the project that a working implementation does not mean a *correct* implementation. With the original standard unavailable until the very last moments of implementation, several sources had to be consulted. Unfortunately, not all the literature on the IS-95A standard is consistent. In some cases, the authors of this paper were forced to discriminate between conflicting books. Having a wealth of information available proved to be absolutely neccessary.

3.2 Design Issues

3.2.1 Lack of Information

The largest problem with the implementation of the receiver system is lack of information about the IS-95A standard. The document itself is available for a very steep price - infeasible for a project of this scale. There are many books available on the subject, but they either contain conflicting information, or are unclear about the specifics of implementing the system.

The scarcity of information thus seriously complicated the debugging process. When testing a given module, it's difficult to know whether a problem is occurring due to error in implementation, or incorrect information about the theory of the process.

3.2.2 Demodulation Uncertainty

Most literature on IS-95A is not implementation-based, but rather, theory based. Conflicting literature on how the system works, misused nomenclature, and an overall scarcity of information made the problem especially difficult to tackle. By studying various partial implementations (mostly from graduate/PhD theses), a coherent picture finally came together. While in the end, the demodulation dilemma was tackled, the lack of information regarding the modulation/demodulation specifics made the process far slower. Towards the immediate end of implementation, Matlab was of significant help in testing theories on how to demodulate the CDMA traffic. The Simulink IS-95A block-set was indispensable.

3.2.3 Deinterleaver Timing

The original implementation of the Deinterleaver was giving consistent, but incorrect output. Modeling the module in ModelSim and inspecting local registers revealed that the timing delay on the ROM was not being taken into account. Figure 3 shows the timing constraints that had to be met. By giving the ROM time to produce the correct

output before using the value to index a bus, the timing issue was solved and the module started to give correct results.

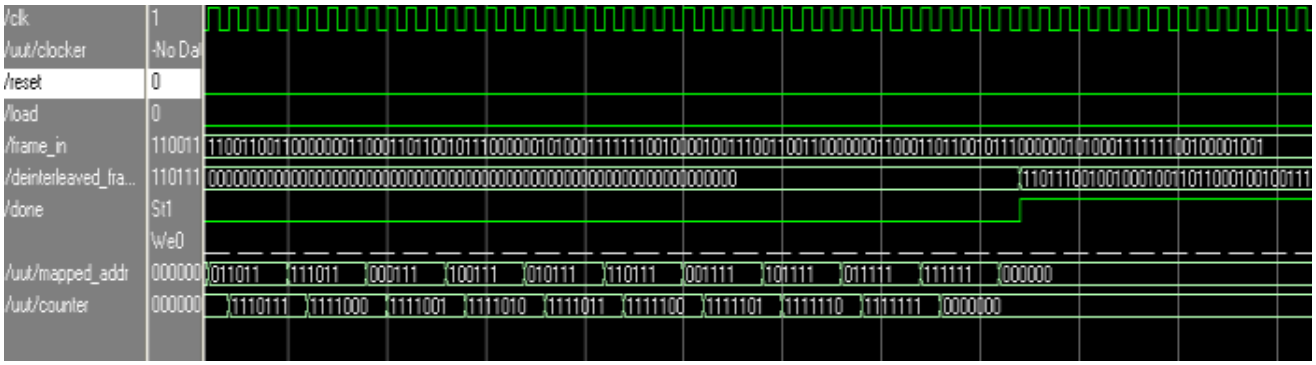

Figure 3: A ModelSim simulation of the Deinterleaver module processing data. "Mapped_addr refers to the ROM output.

3.2.4 Short PN Code Generator

The short PN code generator is verified in simulation by using a ModelSim test bench. The test bench clocks the module indefinitely and displays its buffered output. The output is verified with the output of a Matlab short PN code generator included in the CDMA Reference Blockset. To verify the correct hardware operation of the short PN code generator, the serial interface described in §2.2 is used to transmit the output of the short PN code generator over serial to a PC. It's output can then be verified with the Matlab-generated short PN sequences.

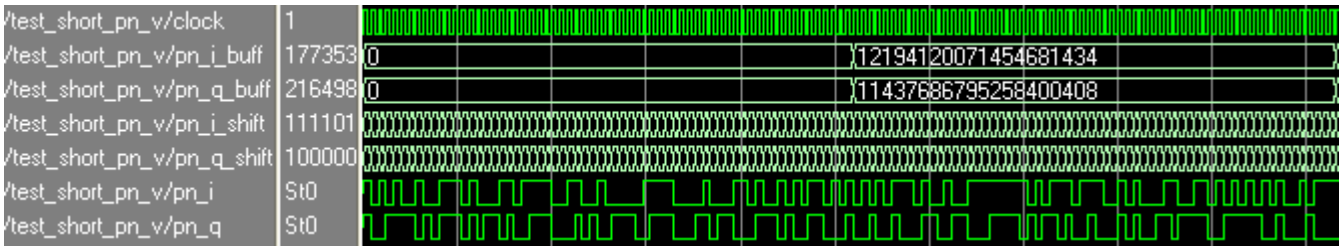

Figure 4: A ModelSim simulation of the Short PN Code Generator generating the PN_I and PN_Q sequences.
The *pn_i* and *pn_q* signals are the latest generated bit at each timestep, and the *pn_i_buff* and *pn_q_buff* signals are set to a new 64-bit value every time 64 bits are generated.

3.2.5 Viterbi Decoder Synthesis

The Viterbi Decoder module instantiates over 512 separate sub-modules, of which nearly 80% have complex interconnections. As a result, the Xilinx synthesizer was unable to synthesize the decoder in a reasonable amount of time (less than ninety minutes). This fact, combined with time constraints, prohibited a non-stalling synthesis of the module to occur.

4 Conclusion

The major blocks of the IS-95A standard were successfully implemented on an FPGA. Most of these blocks were independently tested to verify proper operation according to the standard. In retrospect, it has become apparent that acquiring specific implementation information early on is of crucial importance. This allows a fleshed-out design to be made early on without the necessity of modifying components later. In figuring out how to tackle the problems each block provided, it also became apparent that serious parallelism could be strongly exploited on an FPGA. Many of the methods presented in this project could be further optimized to run at extremely fast speeds compared to microprocessor-architecture counterparts. The authors plan to continue development on this project and eventually be able to intercept voice traffic.

Acknowledgements

We would like to thank David Wentzloff for our helpful discussions about demodulation. We were essentially going into this project blind, and spent at least as much time figuring out how to tackle our design as we did implementing our design. Our discussions regarding spread-spectrum cleared up many of our questions and saved us a good deal of time. In addition, we would like to thank the entire 6.111 course staff for keeping the lab open late and for constantly ignoring the fact that we were generally the last people to leave as the lab was being cleared out.