Implementing a RAKE Receiver for Wireless Communications on an FPGA-based Computer System

Ali M Shankiti
Motorola, SPS
Mansfield, MA, 02048
ali.shankiti@motorola.com

Prof. Miriam Leeser
Dept of Electrical and Computer Eng.
Northeastern University
Boston, MA 02115
mel@ece.neu.edu

ABSTRACT

RAKE receivers are widely used in the wireless communications industry. Currently, custom VLSI is the most popular implementation. Programmable and reconfigurable logic implementations are becoming more attractive because of their flexibility and due to technology advancements. We have implemented a RAKE receiver on an Annapolis Wildforce board with four Xilinx 4000 family chips for a total of 100,000 gate equivalents. Our system is able to implement a RAKE receiver for underwater data communication systems that works in real time. We also investigate mapping a RAKE receiver to a Virtex chip for real-time atmospheric wireless communication.

Keywords

FPGA, RAKE receiver, wireless communications.

1. INTRODUCTION

In wireless communications, transmitted signals arrive at the receiver through different paths. This is because the signals get reflected and refracted off of the physical obstacles in their way and through the atmosphere. This problem is known as multi-path in the wireless industry, and it stands as the most challenging problem in recovering the transmitted signal. A RAKE receiver rakes through a given time-window and searches for different delayed versions of the *same* signal. It then employs a variety of techniques to use the different images to decode the signal, rather than using a single image.

FPGA '2000 Monterey CA USA

We have implemented a RAKE receiver on an Annapolis Microsystems Wildforce Board with four Xilinx XC4028 processing elements. Our RAKE receiver both demodulates and despreads the incoming signal. It operates in real time for underwater communication. We have also investigated implementing a RAKE receiver for voice reception through the atmosphere. A Virtex chip from Xilinx Corporation has both the capacity and the speed to implement a RAKE receiver for voice applications.

Below, we discuss related work in mapping signal processing algorithms to FPGAs. In the next section, we present the techniques used to implement the wireless transmitter and the RAKE receiver. In Section 3 we present the receiver design and implementation constraints. In Section 4 we discuss our design flow and implementation of the RAKE receiver. Finally, we discuss mapping a RAKE receiver to a Xilinx Virtex chip for real-time voice reception.

1.1 Related Research

There has been increased interest in implementing signal processing and communications algorithms on FPGAs. Cummings and Haruyama [1], for example, discuss the emergence of "FPGAs in the Software Radio", and compare FPGA implementations to DSPs and ASICs. Their comparison is qualitative. It does not give particular performance comparisons, but rather focuses on the advantages of FPGAs in communications products, namely flexibility and performance advantages due to concurrency.

Other researchers present quantitative data showing the advantages of using FPGAs in signal processing systems. Peterson and Hutchings [2] conducted performance measurements of individual multipliers mapped to FPGAs, DSP processors and ASICs. They found that FPGAs can provide an order of magnitude better performance than DSP processors and can even approach ASIC performance levels. The authors point out that most DSP systems are limited by their multiplication performance. The authors use the FPGAs to perform DSP algorithms, and compare their performance to that of the C5X DSP processors from Texas Instruments. In addition to individual multipliers, a 20-tap FIR filter and a Fast Fourier Transformer (FFT) are used as benchmarks. They conclude that the only systems that performed worse than the DSP processor are those using only a single FPGA-based multiplier to perform the entire

filter loop in the FIR implementation. In order for FPGAs to obtain a performance improvement over DSP processors, extensive specialization and concurrency must be employed. Our RAKE receiver implemented on a Wildforce board, currently uses only one multiplier per chip. Each finger occupies one chip. Parallelism is realized by using multiple FPGAs for multiple fingers running concurrently. We are investigating a RAKE receiver implemented on a Virtex chip, in which case there are many more opportunities for parallelism.

Other researchers have implemented complete systems in FPGAs, and compared them to both ASIC and DSP implementations. Moeller and Martinez [3] have implemented a radar front-end processor in a Virtex chip and concluded that a filter implemented on a Xilinx Virtex chip can meet ASIC performance. Graham and Nelson [4] have implemented sonar beamforming on a board of Xilinx based FPGAs, and compared the performance to the same algorithm implemented on a Sharc DSP from Analog Devices. Their findings show that multi-chip FPGA implementations can outperform multi-chip DSP implementations as a result of flexibility in FPGA architectures, good communications, the ability to match the implementation to the algorithm, and concurrent processing.

The work most closely related to ours [5] uses a Xilinx FPGA to build a matched filter for a spread spectrum communications system. The authors explain how a matched filter can be used as a means of establishing synchronization in Direct Sequence Spread Spectrum (DSSS) communications. A matched filter is essentially a sub-design of the RAKE receiver. It despreads the received signal without demodulating it. A filter matched to a specific pseudo random (PN) sequence has an impulse response equal to the PN sequence reversed in time. The input is the receiver data; the output will peak when the data is matched to the filter. This indicates that the data and the code sequence are synchronized. A block diagram of the matched filter is shown in Figure 1. Decoded signals for both I (inphase) and Q (quadruture) channels are generated. To decode a signal, the matched filter simply multiplies the input by the PN coefficients A_0 through A_{n-1}. The results are added every T word delay. A hit signal is asserted when the magnitude is greater than a user-specified threshold. This design was implemented on two Xilinx 4005 FPGA chips. The main resource used in the matched filter circuit is the add/subtract module.

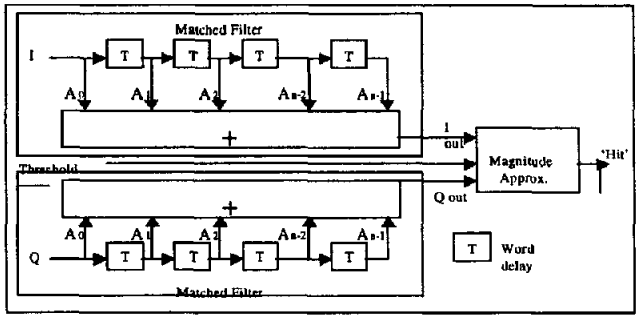

Figure 1: Block diagram of the matched filter detector

Note that no multiplier is required in this filter because the PN sequence elements are either 1 or –1. In our approach, we demodulate as well as despread the signal, so our implementation is more complex, requiring multipliers as well as add/sub modules. The authors are able to run their bit-parallel design with data speeds as high as 17 MHz

2. SPREAD SPECTRUM MINI-TUTORIAL

Every band-limited channel has a maximum data rate at which it can be used to transmit data reliably. Spread spectrum communication systems use a bandwidth W much greater that the information bit rate R, to spread the power of the transmitted signal over the wide bandwidth of the transmission channel. There are two common techniques to accomplish this, Frequency Hopping Spread Spectrum (FHSS), and Direct Sequence Spread Spectrum (DSSS). In the first technique, the channel is divided into equal sub-channels. The transmitter uses a pseudo-random sequence to hop between those channels while sending data. The later technique, which is the one used in this project, cross correlates a pseudo random sequence called the direct sequence with the information sequence and sends the wide-band product over the entire channel.

Both techniques result in making transmitted signals appear similar to random noise and difficult to demodulate by receivers unless they can produce that same random sequence. In short, spread spectrum systems are used to combat and suppress noise, and to hide the signal by transmitting it at low power. The following block diagram illustrates the basic elements of a spread spectrum communication system with a binary information sequence entering the channel encoder, and at the output of the channel decoder.

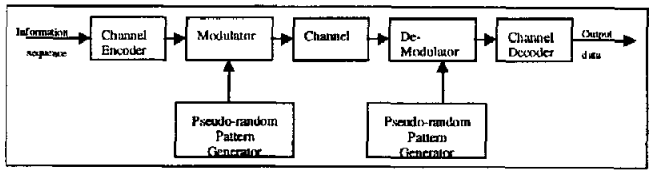

Figure 2: Spread Spectrum digital communication system

2.1 Important Properties of Direct Sequence

The pseudo random sequence used in DSSS possesses unique properties. It has a unity auto-correlation and a minimal cross correlation. Let $p(n)$ be the random sequence vector. Then, $E\{p(n)p^*(n)\}$ is the expected value of the dot product of $p(n)$ with itself, or in other words, the average value of that product. This product is called the auto-correlation of $p(n)$. In a multi-user environment, each user will have a unique $p(n)$ sequence. $E\{p_1(n)p_2^*(n)\}$ is the cross correlation between $p_1(n)$ and $p_2(n)$. A DSSS transmitter n, cross-correlates the random sequence $p_n(n)$ with the information sequence $I(n)$ and sends it over the channel. At the receiver side, $p_n(n)I^*(n)+noise$ is detected. The DSSS despreader, cross-correlates the received signal with the locally generated $p_n(n)$. The mathematical equation that represent this are shown below:

$$E\{[p_n(n)I^*(n)+noise]p_n^*(n)\} = E\{[p_n(n)I^*(n)]p_n^*(n)\} + E\{noise \times p_n(n)\}$$
$$E\{[p_n(n)I^*(n)]p_n^*(n)\} + E\{noise \times p_n(n)\} = E\{[p_n(n)I^*(n)]p_n^*(n)\} + noise$$
$$E\{[p_n(n)I^*(n)]p_n^*(n)\} + noise = E\{p_n(n)p_n^*(n)\}E\{I(n)\} + noise$$

but

$$E\{p_n(n)p_n^*(n)\} = 1$$

hence

$$E\{[p_n(n)I^*(n)]p_n^*(n)\} + noise = E\{I(n)\} + noise = I(n) + noise$$

Note that if the receiver is using the wrong sequence $p_m(n)$, then:

$$E\{p_n(n)p_m^*(n)\} = 0$$

hence

$$E\{[p_n(n)I^*(n)]p_m^*(n)\} + noise = noise$$

This prevents receivers from demodulating the signal unless it was intended for them.

3. MODULATION AND CODING IN DSSS COMMUNICATIONS

Modulation and *coding* are two ways to reject added noise in a wireless communication environment. Nowadays, we can choose from different modulation schemes according to our needs. Direct Sequence Spread Spectrum (DSSS) is a relatively new technique that allows the transmitter to spread the transmitted signal over the entire given spectrum. Differential coding enables us to use non-coherent detection at the receiver, thus simplifying its hardware by eliminating the need for a phase locked loop (PLL). Both DSSS and differential coding are described below.

3.1 Direct Sequence Spread Spectrum (DSSS)

To transmit an information bearing sequence (IBS) of ones and zeros over a wireless channel, the IBS must be modulated. As a result of modulation, many bits are transmitted on the channel that correspond to one bit of information in the IBS. In DSSS, a Pseudo-random Noise (PN) sequence consisting of a number of chips is used to spread the bandwidth of the information bearing sequence over a given spectrum. At the transmitter side, the IBS is multiplied by the PN sequence. At the receiver side, the received signal with added noise is despread by multiplying it again by the same PN sequence, as shown in Figure 3. This allows us to recover the IBS, and at the same time, filter the noise by spreading it at the receiver side. Note that DSSS allows for multiple access by using different PN codes for different users [6, 7].

In Figure 4, a spectral analysis of the modulated signal is plotted. The first figure shows the IBS spectral density before multiplying by the PN sequence. The middle figure shows the spread IBS after multiplying, and the narrow band noise in its center. The last figure shows the recovered IBS after multiplying it again by the PN sequence, and the spread noise that has been multiplied only once. This weakens the effect of the noise by distributing its power across a wide spectrum.

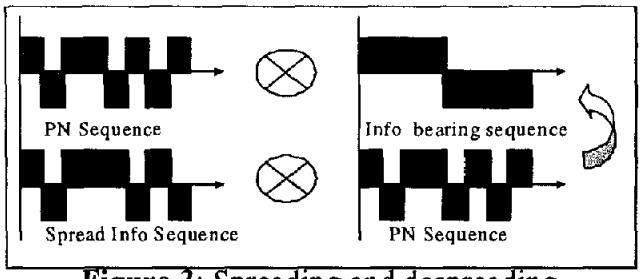

Figure 3: Spreading and despreading

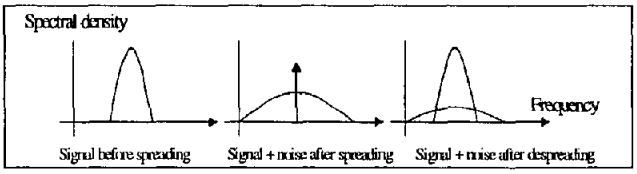

Figure 4: Noise reduction using DSSS

3.2 Differential Coding

Differential coding means that the receiver will only need to track the phase difference between two consecutive symbols (non-coherent detection), instead of tracking the phase of each received symbol by comparing it to a reference signal (coherent detection). In this wireless system, we assume the Doppler effect is negligible. That is because the Doppler shift between two consecutive chips is small. The information will effectively be stored in the phase difference between two consecutive symbols. If a is the information bearing sequence, then $a(n) = d(n) \times d(n-1)$, and the modulated signal is

$$y(n) = d(n)e^{j\theta(n)}$$

where $d(n)$ is the coded digital sequence that goes to the modulator, and θ is the carrier phase.

At the receiver side, two consecutive received symbols are multiplied to recover one transmitted information bit as follows:

$$y(n)y^*(n-1) = d(n)d(n-1)e^{j\theta(n)}e^{-j\theta(n-1)} + noise$$
$$but \quad \theta(n) \approx \theta(n-1)$$
$$r(n) = d(n)d(n-1)$$

where $r(n)$ is the received sequence.

3.3 Transmission Process

The transmitter consists of three modules. The *Source Encoder* receives the information bit-stream. The output is differentially coded. The *Channel Encoder* uses DSSS techniques to overcome the effects of noise and interference introduced by the channel. Every bit is spread by multiplying it with the spreading sequence. In our case, 25 chips are the result of spreading 1 bit. The *Digital Modulator* serves as the interface to the communication channel. It has a clock frequency of 4 kHz. Every chip will be represented by 16 samples. This means that every IBS bit requires that 400 samples be transmitted.

4. RAKE RECEIVER SYSTEM DESIGN OVERVIEW

The complete design consists of a transmitter, a channel model and a receiver as shown in Figure 5. We are only implementing the receiver, but we were required to simulate all three in order to model the design. We used MATLAB® to simulate the design of the transmitter and multi-path channel. We also simulated the receiver and used this as the specification for our design. In the next section we discuss the receiver design and tradeoffs in more detail, and then present the Annapolis Wildforce Board that was used to implement the receiver.

4.1 Receiver Design

Figure 5 gives an overview of the whole system, and details of the implementation of the RAKE receiver. We simulated the receiver in MATLAB and used this as the specification for our receiver design. This receiver can have as many fingers as desired. Each finger will simply receive a delayed version of the transmitted signal. A constant delay is placed between the different fingers. If a path does not exist at a certain finger, corresponding to a certain delay, then the despreading process will produce noise instead of a meaningful signal [8,9].

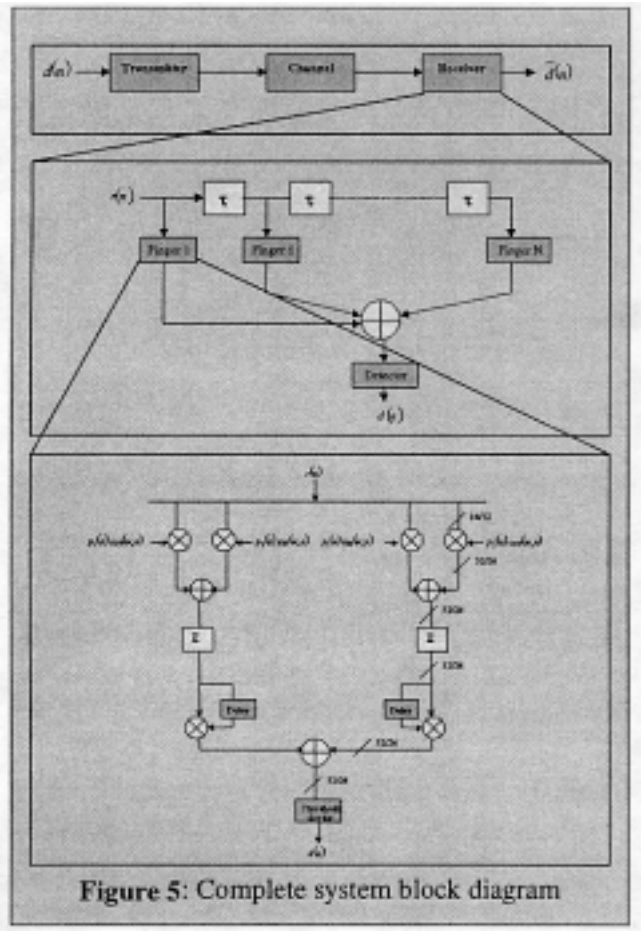

Figure 5: Complete system block diagram

This noise is filtered out by the final detector as shown in the receiver block diagram in Figure 5. Please note that in the figure, the notation 32/24 means that the result is 32 bits, and only 24 bits are used.

4.2 Algorithm Design

Our receiver is designed for use with an underwater, acoustic communication system. For this application, the number of bits that need to be decoded in one second is 400. This important performance number varies widely depends on the application. If this receiver is to be used in mobile phones for example, it will have to be much faster. A quick look at the algorithm shown in Figure 6 indicates that the main loop will iterate 400 times in order to decode one bit. At a rate of 4000 samples/second, the hardware has 250 microseconds to execute one iteration. In a single loop, 12 multiplications are required.

This number is significantly reduced by noticing that the variables INPHASE and QUADRATURE depend only on the sampling frequency and the carrier frequency. Therefore they can be pre-calculated and stored in memory. Also, since the variables Pn1 and Pn2 always have the values 1 or –1, four multiplication operations (lines 12 and 13) can be replaced by simpler operations.

This brings down the total number of multiplications to two, which can be calculated serially or in parallel. The two multiplications outside the loop (lines 15 and 16) could potentially share the hardware with the inner-loop multiplications. The

```
1 SPECIFICATIONS:
2 SAMPLE_PER_CHIP = 16;
3 CHIP_PER_BIT = 25;
4 SAMPLE_FREQ = 40000;
5 CARRIER_FREQ = 12000;
6 SAMPLE_PER_BIT = SAMPLE_PER_CHIP *
CHIP_PER_BIT;

7 FOR n FROM 1 TO SAMPLE_PER_BIT LOOP
8 READ Pn1 EVERY (SAMPLE_PER_CHIP) TIMES;
9 READ  Pn2 EVERY (SAMPLE_PER_CHIP) TIMES;
10  INPHASE = COS (n/(SAMPLE_FREQ) * 2 * PI *
CARRIER_FREQ);
11 QUADRATURE = SIN (n/(SAMPLE_FREQ) * 2 * PI *
CARRIER_FREQ);
12 BRANCH_1 = BRANCH_1 + r(n) * [Pn1 * INPHASE + Pn2
* QUADRATURE];
13 BRANCH_2 = BRANCH_2 + r(n) * [Pn1 *
QUADRATURE - Pn2*INPHASE];
14 END FOR LOOP;

15 BRANCH_1 = BRANCH_1 * PREVIOUS_BRANCH_1;
16 BRANCH_2 = BRANCH_2 * PREVIOUS_BRANCH_2;
17 FINGER=BRANCH_1 + BRANCH_2;
18 IF FINGER > THRESHOLD THEN PASS;
19 ELSE FINGER = 0;
```

Figure 6: Algorithm

algorithm, as mentioned before, was tested using MATLAB. MATLAB is a very efficient computational language. Coupled with its signal processing toolbox, it becomes a powerful means to verify signal-processing algorithms before attempting synthesis.

MATLAB code looks very different than the behavioral code listed previously. One reason is the MATLAB dot operator (.x) that facilitates one line vector operations such as multiplication and addition instead of using a loop. The multiplication loop in Figure 6 has the MATLAB representation shown in Figure 7.

MATLAB code is generally shorter than behavioral VHDL since in VHDL, the loops must all be made explicit. MATLAB is a succinct way to express the algorithms used in our receiver.

The main disadvantage of MATLAB from a hardware modeling point of view, is that MATLAB uses very wide precision arithmetic operations. In hardware, by comparison, these operations are performed in fixed point with the minimum number of bits required in order to keep the area used small. Note that our MATLAB code is purely algorithmic, and cannot be directly translated to hardware.

```
7 PN1 = [ 1 1 1 .....1] // each number repeated
(SAMPLE_PER_CHIP) times//
8 PN2 = [ -1 -1 -1.....1] // each number repeated
(SAMPLE_PER_CHIP) times//
9 n = [1 2 .. Sample_per_bit]
10 Inphase = cos(n/(SAMPLE_FREQ) .x 2 x PI x
CARRIER_FREQ
11 Quadrature = SIN(N/(SAMPLE_FREQ) .x 2 x PI x
CARRIER_FREQ
```

Figure 7: MATLAB sample code

4.3 Annapolis Wildforce™ Board

A Xilinx® based FPGA board from Annapolis Microsystems, shown in Figure 8, is used to realize this receiver. This board consists of four XC4028EX-3HQ240 programmable elements (PEs), and one XC4028EX-3HQ304 controller. Each finger is mapped on one PE. The final result, the bit decoded by the receiver, is communicated by the controller to the host PC. The detector and the threshold device can be mapped on any of the four PEs.

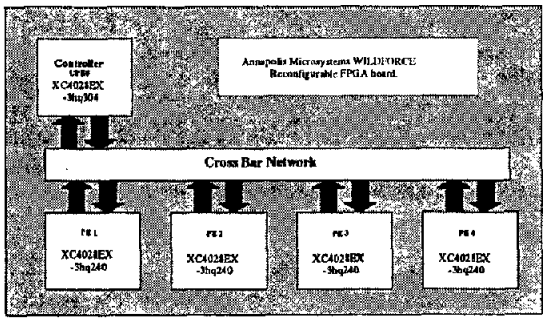

Figure 8: Annapolis FPGA board

5. DESIGN FLOW

As stated previously, we began our receiver design with simulation in MATLAB® in order to verify functionality. A fixed-point library, written in the C programming language, was used to simulate the actual hardware implementation. At this point the overall design of the receiver and the bit-width of the operations were determined. Next, we used Synplicity design tools to map our design to the Annapolis Wildforce board. Xilinx placement and routing tools are used to route the design for final downloading to the FPGA board. A block diagram of the system design and synthesis process is shown in Figure 9.

Since the receiver could not possibly fit on a single chip, a distributed implementation strategy had to be employed. A decision was taken to configure one finger on each FPGA chip. That meant that one finger had to occupy no more than 1024 CLBs. The downside was that we could fit only one multiplier on one chip, the upside side was that this approach made our design more modular. Recall that two multiplications are executed in each iteration. We used one multiplier to serialize these operations. We were able to use this multiplier for the outer loop multiplications as well.

Our final design uses 22 cycles and occupies close to 1000 CLBs. The clock speed was chosen to be 1 MHz. The throughput is equal to 4000 samples per second or 4 kHz. Delays between fingers were implemented using a 5-bit counter, where the finger will ignore the incoming data bits until the counter reaches a preset number equal to the number of samples per bit.

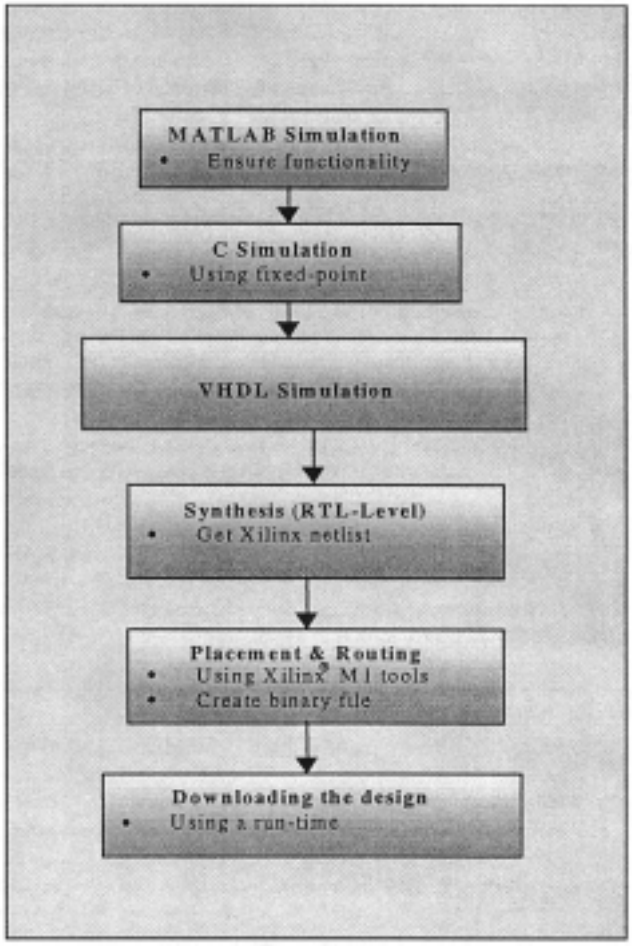

Figure 9: Design flow block diagram

To test the design, a number of bits were modulated and spread using the transmitter routine in MATLAB. The bits were then passed through the channel simulator which acted as a 3-path

channel. Finally the output of the channel was passed as the input to the receiver on the FPGA board. The output of the receiver was compared with the transmitted bits and verified to be the same.

6. DISCUSSION

We have presented the design for an underwater RAKE receiver that his been implemented on an Annapolis Wildforce Board and runs in real time. The receiver both despreads and demodulates the incoming signal. In this section we discuss improvements we would like to see in the design process, and current work in implementing a RAKE receiver for voice communication over the air on a Virtex chip.

The translation process of the MATLAB code to VHDL significantly slows down our design flow. This is because of the inherent differences between the two languages. MATLAB on one hand is a language specifically designed for modeling signal processing with special constructs for vector operations, while VHDL is a concurrent hardware modeling language which requires all loops, interfaces, and signal types to be made explicit. As a result, the translation process between the two languages is not trivial. What is needed is a design tool that would translate the MATLAB code either VHDL, or directly to an FPGA implementation. Other researchers are currently working on this problem [10, 11]. These approaches currently require the MATLAB description to be specified closer to hardware than the algorithmic approach we currently use.

We have used a member of the Xilinx 4000 family to implement this receiver. Since then, Xilinx has come up with far more dense chips. For example, the Xilinx XCV50 Virtex chip can hold up to 25 RAKE fingers as opposed to one using the XC4028EX. To perform real time voice wireless communications using direct sequence spread spectrum techniques, a more involved analog interface will have to be adopted. A typical air-interface carrier frequency is 1900 MHz. The signal is extracted from the carrier frequency in the analog domain mixing it with the output of a variable crystal oscillator (VCO) running at the carrier frequency. The output will then be fed to an FPGA which will despread it then demodulate the inphase and quadrature parts. Multipath still exists in such an environment, and an FPGA-based RAKE receiver is necessary to increase the reliability of this wireless system. The system architecture of such a receiver would be the same as the one described in this paper.

A typical example of an air-interface wireless standard is the IS-95 standard protocol for CDMA mobile radio communications in North America. Channel bandwidth is 1.25 MHz. A maximum data rate of 9.6 kb/s per user is spread over the channel bandwidth, a total spreading factor of 128. This is equivalent to 1.2288 Mchips/sec or 128 chips/bit. The transmitter simply spreads the data then modulates it before transmitting it. The receiver will have to first demodulate the signal then despread it. Since the channel is 1.25 MHz wide, the receiver will have to process 1.25 million samples each second. Each sample involves two multiplications, inphase and quadrature. A simple calculation reveals that the receiver will have to perform 2.5 million multiplications each second. An FPGA running at 10 MHz could do the job provided more than one multiplier is used. For example if two multipliers were used, each will have 8 cycles to perform a multiplication. This design should be easily implementable on a Virtex chip, while still fitting several fingers per chip. The IS-95 standard recommends three or more fingers for the receiver.

7. CONCLUSION

In this paper, we have discussed the design and implementation of a RAKE receiver. The design process requires numerous decisions before committing to a particular RTL implementation. Prior to synthesis, the receiver algorithm was designed both for accuracy and for efficient hardware implementation. Synplicity synthesis tools were used to implement the design on an Annapolis Wildforce Board.

8. ACKNOWLEDGMENTS

The RAKE receiver design was developed by Dr. M. Stojanovic and Ethem Sozer. We would like to thank Shantanu Tarafdar for implementing our fixed-point library. We are grateful to Synplicity for their donation of software.

9. REFERENCES

[1] Cummings, M. and Haruyama, S., "*FPGA in the Software Radio,*" in IEEE Communications Magazine, pp. 108-112, February, 1999.

[2] Peterson, R. and Hutchings, B., "*An Assessment of the Suitability of FPGA-Based Systems for use in Digital Signal Processing,*" 5th International Workshop on Field-Programmable Logic and Applications, pp. 293-302, August 1995, Oxford, England.

[3] Moeller, T. J. and Martinez, D. R., "*Field Programmable Gate Array Based Radar Front-End Digital Signal Processing,*" in Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, April, 1999.

[4] Graham, P. and Nelson, B. "*FPGA-based Sonar Processing,*" in ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp. 201-208, February 1998.

[5] Mathews, T., Gibb, S., Turner, L., Graumann, P. and Fattouche, M., "*An FPGA Implementation of Matched filter Detector for Spread Spectrum Communications systems,*" the 7th International Workshop on Field-Programmable Logic and Applications, in *Lecture Notes in Computer Science 1304*, pp. 365-373, Springer Verlag, Sept 1997.

[6] Proakis, John G. *Digital Communications,* McGraw-Hill, New York, 1995.

[7] Rappaport, Theodore S. *Wireless Communications,* Prentice-Hall, 1996.

[8] M.Stojanovic and Z. Zvonar, "*Multichannel processing of broadband multiuser communication signals in shallow water acoustic channels,*" IEEE Journal of. Oceanic Engineering, pp. 156-166, April 1996.

[9] E.M. Sozer et. al., *"Direct Sequence Spread Spectrum Based Modem for Under Water Acoustic Communication and Channel Measurements,"* Proceedings of OCEANS'99, Seattle, WA, Nov. 1999.

[10] P. Fiore, C. Myers, J. Smith, E. Pauer, *"Rapid Implementation of Mathematical and DSP Algorithms in Configurable Computing Devices"*, in Configurable Computing: Technology and Applications, Proceedings SPIE Vol. 3526, November, 1998.

[11] A. Nayak, A. Jones, et al., *"Library Functions in Reconfigurable Hardware for Matrix and Signal Processing Operations in MATLAB,"* Proceedings 11th IASTED Parallel and Distributed Computing and Systems Conference (PDCS'99), Cambridge, MA, Nov. 1999.