An ECA-Based ZigBee Receiver

Chen Zhang

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Electrical Engineering

Peter M. Athanas, Chair
Jeffrey H. Reed
Thomas L. Martin
Michael R. Buehrer

February 1, 2008
Blacksburg, Virginia

Keywords: ECA, Software Defined Radio,
ZigBee Receiver, Dataflow Machine

An ECA-Based ZigBee Receiver

Chen Zhang

ABSTRACT

Element CXI's Elemental Computing Array (ECA) delivers faster reconfiguration time and higher computational density than Field Programmable Gate Arrays (FPGAs) with similar computational power. It provides higher computational power than Digital Signal Processors (DSPs) with similar power consumption and price. It also utilizes a library-based graphical development environment promoting ease of use and fast development. In this thesis, the design and implementation of a ZigBee receiver on an Element CXI ECA-64 platform is presented. The ZigBee receiver is evaluated through simulations and implementation on an ECA device. During the design and implementation of the ZigBee receiver, some design experience and tips are concluded. The design methodology on the ECA is studied in detail to assure the implementation's correctness, since the methodology of the ECA is different from that of other platforms.

Acknowledgements

I would like to thank my advisor, Dr. Peter M. Athanas, for his continuing support, words of encouragement. I would also like to thank the rest of my committee, Dr. Reed, Dr. Buehrer and Dr. Martin, for their involvement in my thesis. I would also like to thank Mr. Zhong and Mr. Watson from Element CXI for their help understanding the intricacies of hardware implementations.

I could not have accomplished so much without my colleagues and friends at CCM Lab. I would like to specifically express my appreciation to Nicholas J. Macias, Stephen Craven, Alex Marschner and Justin Rice for their help on my research and thesis.

Finally, I would like to convey my gratitude to my wife and my parents for their support as well as their patient, persistent reminders to write this thesis.

Table of Contents

List of Figures

Glossary of Acronyms

ASIC	Application-Specific Integrated Circuit
AWGN	Additive Gaussian White Noise
BER	Bit Error Rate
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DSP	Digital Signal Processor
ECA	Elemental Computing Array
FIR	Finite Impulse Response
FPGA	Programmable Gate Array
GPP	General Purpose Processor
GUI	Graphic User Interface
HDL	Hardware Description Language
HLL	High-Level Languages
LSB	Least Significant Bit
LUT	Look-Up Table
MAC	Medium Access Control
MIPS	Million Instructions per Second
MSB	Most Significant Bit
OFDM	Orthogonal Frequency-Division Multiplexing
PHR	PHY Header
PHY	Physical Layout
PSDU	PHY Service Data Unit
RF	Radio Frequency
RSSI	Received-Signal-Strength Indicator
SDR	Software Defined Radio
SHR	Synchronization Header
SPD	Signal Processing Designer
SME	State Machine Element
WPAN	Wireless Personal Area Network

Chapter 1

Introduction

Software Defined Radio (SDR) systems have changed from obscurity to commercial viability a short time, as the result of rapid semiconductor technology development [1]. Currently, mainstream hardware platforms for SDR systems include General Purpose Processors (GPPs), DSPs, FPGAs, and Application-Specific Integrated Circuits (ASICs). Each of these platforms has its own advantages as well as drawbacks. GPPs are relatively easy to use because they are software-programmable with high-level languages, though with some cost to their computational power. DSPs intend to provide higher power efficiency than GPPs. Today's DSPs or GPPs alone cannot handle the complex algorithms at the required speeds necessary for SDR with reasonable power consumption [2]. ASICs provide the most optimized hardware implementation of an algorithm, albeit only for the specific applications for which they have been designed [3]. FPGAs offer a compromise of flexibility and computational power by using hardware reconfiguration to handle complex high-speed algorithms [2]; however, the Hardware Description Languages (HDLs) used for specifying an FPGAs configuration are generally not as easy to use as a high-level programming language. Moreover, the cost of FPGAs tends to be much higher than that of DSPs.

1.1 Motivation

The Elemental Computing Array architecture from Element CXI is a newly developed architecture providing a unique "electronic ecosystem" enabling software-defined hardware [4]. The ECA offers hardware flexibility and high computational power similar to an FPGA, but at lower cost and with higher power efficiency and

computational density than most FPGAs. Moreover, the ECA exhibits resiliency and rapid run-time reconfiguration; two unique features that give this chip great potential for use in SDR systems [5]. Hence it is a worthy objective to implement a software-defined radio system on the ECA. This is a challenge that no one has tried before.

In this thesis, a ZigBee receiver is implemented and tested on the ECA platform. Features of the ZigBee system such as low data rate, low cost, and low power consumption match the capabilities of the ECA platform very well.

1.2 Enumeration of Contributions

This thesis presents the first implementation of a radio system on the ECA platform. The primary contributions of this research are:

1. The implementation and verification of a ZigBee receiver on an ECA-64 chip: the first radio system implemented on such a chip.
2. The study of the design methodology of the ECA. Since the ECA is a dataflow machine, the design methodology is different from that used with FPGA and DSP designs. The methodology is studied in this thesis. The experience gained will be beneficial to future SDR system implementations, and will also help with improving the ECA hardware and design tools.
3. The design of essential functions: some essential functions, such as a control counter, a sequence detector, and an edge detector, are designed in a unique way.

1.3 Organization of Thesis

Following this introductory chapter, Chapter 2 provides the necessary background information on the SDR system, its hardware platforms, and an overview of the ZigBee system. Chapter 3 presents the characteristics, features, and benefits of the ECA architecture. Chapter 4 provides the features and benefits of the ECA development

tools. Chapter 5 describes the overall and detailed design of the ZigBee receiver on the ECA platform. Chapter 6 presents the gate-level implementations of the ZigBee receiver in the development tool of the ECA. In this chapter, the simulation results of the ZigBee receiver are also provided. Chapter 7 concludes with an overview and suggestions for future work.

Chapter 2

Prior Work

This chapter presents a brief overview of the SDR architecture, then discusses and compares the characteristics, benefits, and drawbacks of several different hardware platforms for SDR. It then introduces the ZigBee protocol and IEEE 802.15.4 standard, and describes the data transport architecture.

2.1 SDR and its hardware platforms

2.1.1 Software Defined Radios

The exact definition of an SDR is controversial, and no consensus exists about the level of reconfigurability need to qualify a radio as an SDR. The term "SDR" generally refers to a radio that derives its flexibility through software while using a static hardware platform [3].

Implementation of the ideal SDR system requires either digitization at the antenna, allowing complete flexibility in the digital domain, or the design of a completely flexible radio frequency (RF) front-end for handling a wide range of carrier frequencies and modulation formats [3]. However, in the real world, this second type of system can handle signal frequencies from 10 MHz to 10GHz, but may incur a significant-high cost. An intermediate version that supports several different waveforms within a certain frequency range is therefore more viable for commercial applications. For instance, a SDR system supporting ZigBee, Bluetooth, and IEEE 802.11 b/g can be a low-cost system [6].

2.1.2 Hardware Alternatives

The hardware platforms for an SDR system can be divided into three categories of digital hardware: GPPs/DSPs, FPGAs and ASICs. GPPs and DSPs represent the most generalized type of hardware, and can be programmed to perform various functions, while ASICs are the most specialized pieces of hardware and provide the best performance for the specific applications for which it has been designed. FPGAs offer a compromise in flexibility and performance between GPPs/DSPs and ASICs. The new ECA also fits the category of FPGAs, though with more new features.

Six key critical parameters for evaluating an SDR platform include: computational power; computational density; hardware flexibility; application flexibility; power consumption; and reconfiguration time [7]. In addition, total cost is also an important factor for the evaluation. Computational power is the most essential parameter. Figure 2-1 shows the approximated computational complexity of several types of radios. A platform can only support a radio system whose computational complexity does not exceed the maximum computational power of the platform. To a certain degree, computational density decides the computational power. Hardware flexibility denotes the ability to reconfigure the underlying hardware's configuration. Application flexibility defines the supporting range of applications. Power consumption is one of the most important parameters for mobile systems: the lower the power consumption, the longer the battery life. Reconfiguration time is important for those users who need to rapidly switch their transceiver between different radio systems. For a commercial SDR system, cost is another essential factor which has to be carefully considered.

GPPs/DSPs

GPPs and DSPs are similar to each other in having low computational power/density, excellent application flexibility, and single-clock-cycle reconfiguration times [7]. On each platform, programming can be accomplished with little difficulty since tasks can be described with High-Level Languages (HLLs) such as C/C++ and Java. However, while eight-core processes have appeared in Intel's processor road map, HLLs may make poor use of that potential parallelism [3].

DSPs provide better power efficiency than GPPs since DSPs lack some of the complex structures of GPPs. However, both of them can take advantage of newer manufacturing technology to reduce the power consumption and increase the computational power. For an instance, Intel's new 45-nanometer technology offers 20-percent improvement in transistor-switching speed, 30-percent reduction in transistor switching power, and 10 times reduction in transistor gate oxide leakage compared to 65-nanometer technology [8]. However, GPPs and DSPs still cannot handle high-end waveforms such as orthogonal frequency-division multiplexing (OFDM) with reasonable power consumption, even with multi-core and 45-nanometer technology [7]. The cost of GPPs and DSPs varies, and depends in general on their performance.

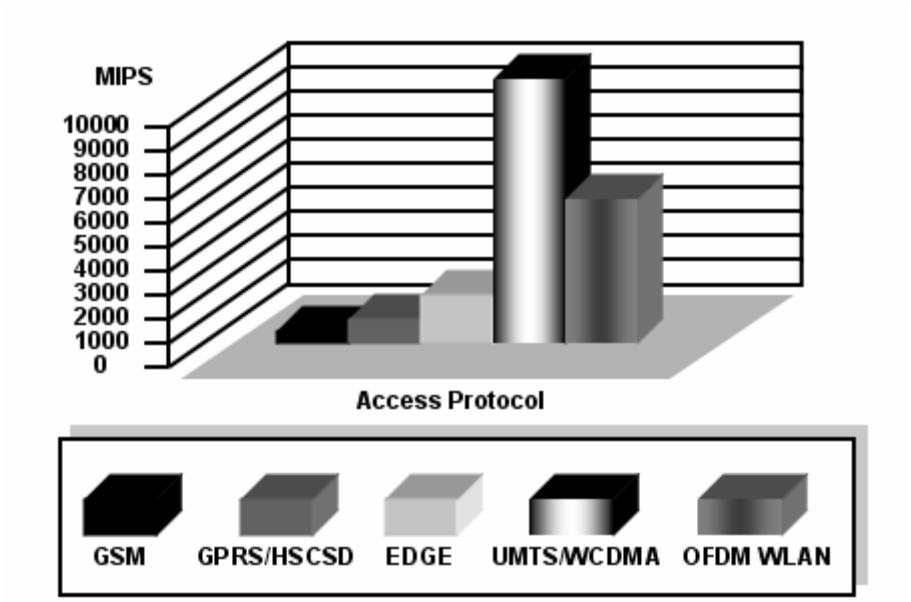

Figure 2-1. *Approximated Computational Complexity of Radio Systems* [7]

FPGAs

An FPGA is a semiconductor device containing both programmable logic components (called "configurable logic blocks") and programmable interconnects. Unlike a GPP or DSP, which has a fixed structure, the hardware structure inside an FPGA can be easily configured. Users can form their own structures in an FPGA to handle a specific application with optimal performance. Modern FPGAs also integrate

large on-die memory and DSP cores to boost performance to a new level. In some extreme applications, the performance of an FPGA can be hundreds of times the performance of a DSP. The impressive high-performance of FPGAs comes with some significant disadvantages. The first one is the cost: the price per unit can be over to $17,000 for the highest-performance devices [9]. The second disadvantage is the difficulty of application development. Hardware description languages, such as VHDL and Verilog, are used for programming FPGAs. The hardware design concepts used are totally different from those of C/C++ and Java. Hence it can be difficult for C/C++ programmers to adapt to VHDL or Verilog. In addition, most of the area within an FPGA is occupied by the interconnect fabric, while the actual logic components occupy only a small portion of the entire chip [10]. As the result, the computation density of FPGAs is lower than ASICs.

ASICs

Custom ASICs provide the highest computational power, computational density, and power efficiency. However, they alone cannot be the main processing unit for SDR systems, since they have fixed hardware structures with no post-manufacture configurability. The role of ASICs in SDR platforms is thus mainly as coprocessors for DSPs and GPPs. ASICs provide much lower recurring cost but significantly higher initial cost compared to FPGAs [7].

Elemental Computing Arrays

ECAs provide similar computational power but higher computational density than FPGAs, since the element-based (“coarse grained”) structure provides excellent parallelism for high computational power, while the simplified interconnection scheme saves more chip area for the actual logic components, thus resulting in higher computational density. Also, the power consumption and cost of the ECA are low enough to compare with that of DSPs [11].

2.2 ZigBee Overview

ZigBee is a low-power, relatively low data-rate, and low-cost wireless communications protocol for the next-generation WPAN. ZigBee specifies the physical layers (PHY) and medium access control (MAC) sub-layers based on IEEE 802.15.4, which is the short-distance wireless communication standard for the 2.4 GHz band. The ZigBee protocol, including the ZigBee platform specifications and ZigBee profile, are defined by the ZigBee Alliance [12].

2.3 IEEE 802.15.4 Standard Overview

IEEE 802.15 is the 15th working group of the IEEE 802 that specializes in Wireless Personal Area Network (WPAN) standards. Task Group 4 is working on the low data rate, low system complexity, low-cost, and long battery life standard. The first edition of the 802.15.4 standard was released in May 2003 [13].

IEEE 802.15.4 standard specifies a 250 Kbit/s data transfer rate at a 10-meter radium, and targets extremely low manufacturing and operation costs and technological simplicity, without sacrificing flexibility or generality [14]. The IEEE 802.15.4 standard employs an offset-QPSK modulation scheme, carrier-sense-multiple-access with collision-avoidance (CSMA/CA) and 16-ary orthogonal modulation technology. The CSMA/CA mechanism allows multiple ZigBee transceivers sharing the same frequency band to transmit signals. The 16-ary orthogonal modulation technology reduces the interference to the adjacent channel and provides the anti-interference ability to the ZigBee systems.

2.4 ZigBee Data Transport Architecture

ZigBee transfers data in frames which have four fundamental types - data, acknowledgment, beacon, and MAC command frames - providing a reasonable tradeoff

between simplicity and robustness [12]. Additionally, the ZigBee coordinators define a superframe structure that may be used. Within superframes, contention occurs between their limits and is resolved by CSMA/CA. From the point of view of the PHY, there are three parts forming a data frame in octets: a synchronization header (SHR) containing a preamble sequence and start of frame delimiter; a PHY header (PHR) containing the length of the frame; and a PHY service data unit (PSDU) containing all MAC sub-layer information and transmitted data [14]. The structure of the data frame is shown in Figure 2-2. The 4-octet (32-bit) preamble enables the receiver to achieve symbol synchronization.

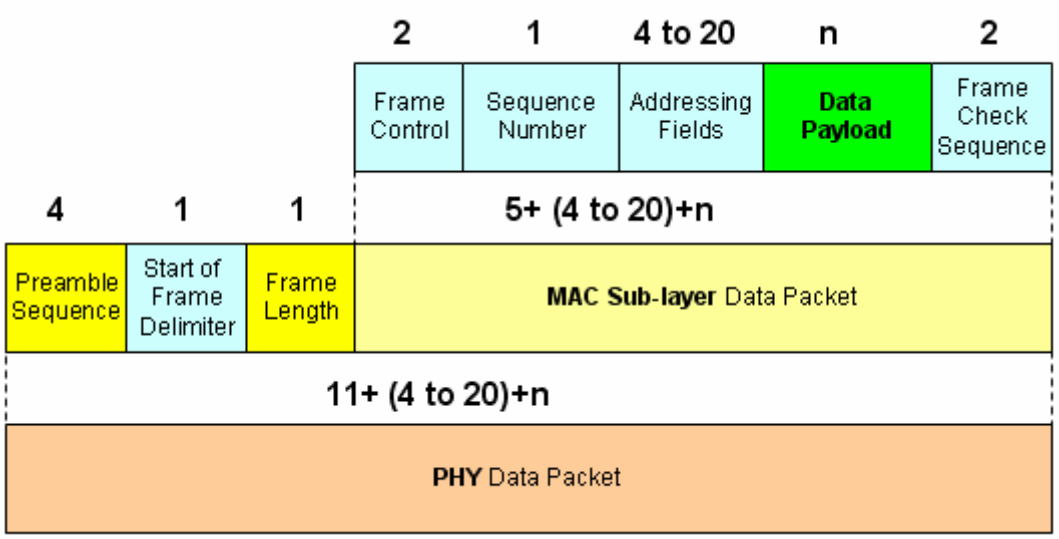

Figure 2-2. *Structure of the ZigBee Data Frame* [18]

2.5 Solutions for ZigBee

Since ZigBee only requires little computational power, it can be virtually implemented in all hardware platforms mentioned in Section 2.1. The ZigBee implementations on three hardware platforms are compared.

The first one is an ASIC-based commercial product, called XBee™, from Digi International. Figure 2-3 shows the product. This XBee is based on a CC2420 ASIC

chip from Texas Instruments [15]. The major features of the XBee are extremely low power consumption and low price ($19), which are the benefits of a high-volume ASIC implementation [15]. However, constrained by the fixed hardware structure, this product can only support the current ZigBee protocol.

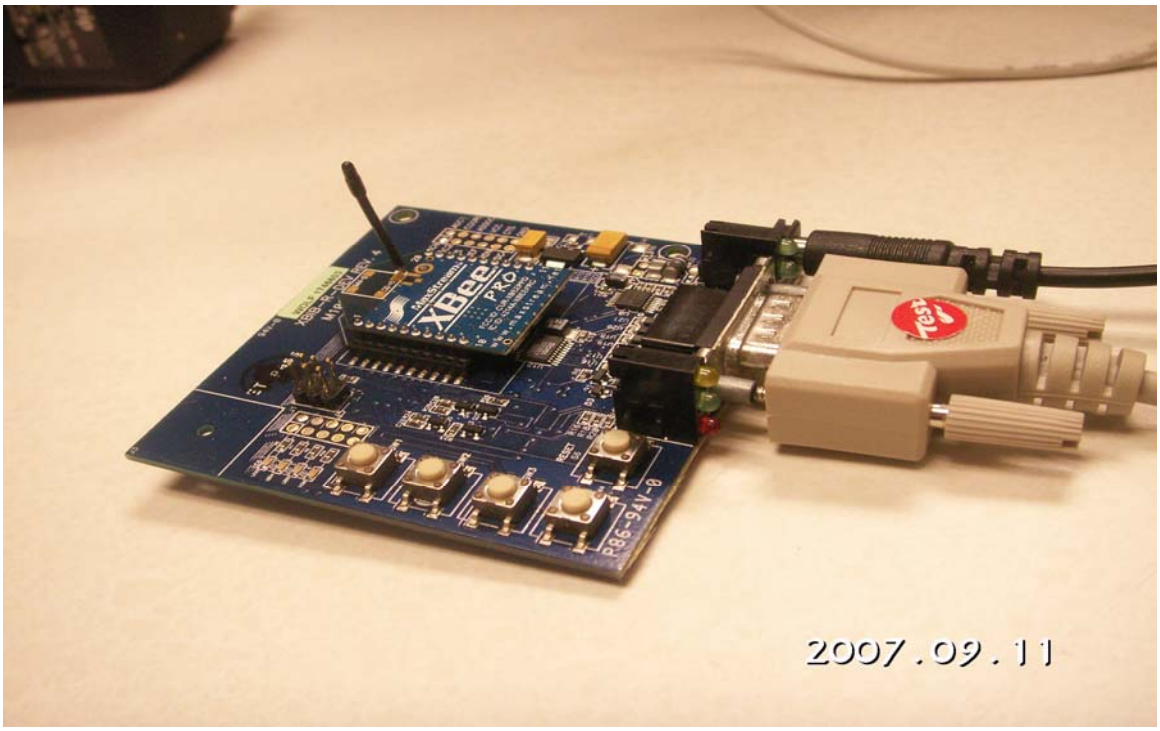

Figure 2-3. *XBee Transceiver from Digi International* [18]

The second implementation is also a commercial product, called PICDEM™ Z ZigBee demonstration board from Mirochip. A picture of the PICDEM Z board is shown in Figure 2-4. The features of this product are low power consumption, relatively low price ($267), and programmability because it is based on a PIC18LF4602 microcontroller, which can be viewed as an equivalent low-end DSP [16] [17]. Since it is programmable, it can support other low-end radios in the future. However, it cannot support high-end radios since the PIC18LF4620 microcontroller has limited computational power.

The last implementation is from Configurable Computing Machine Lab. A ZigBee receiver was implemented on the SIP SDR board from Harris [18]. The block diagram of the SIP SDR board is shown in Figure 2-5. This board is equipped with four Xilinx FPGAs and one Altera FPGA. This SDR board provides high computational

power that can support most radios. The ZigBee receiver on this board only occupies a small portion (11% slices, 6% flip-flops, and 7% LUTs) of one Xilinx Virtex-4 XC4VLX60 FPGA; however, the power consumption and price of this SDR board is also relatively high.

Figure 2-4. *PICDEM Z ZigBee Demonstration Board form Mirochip* [16]

There are no reported implementations of ZigBee transceivers on GPPs. When comparing these ZigBee alternatives, the advantages and disadvantages of each platform are clearly presented. In the next chapter, the structure and feature of the ECA is introduced.

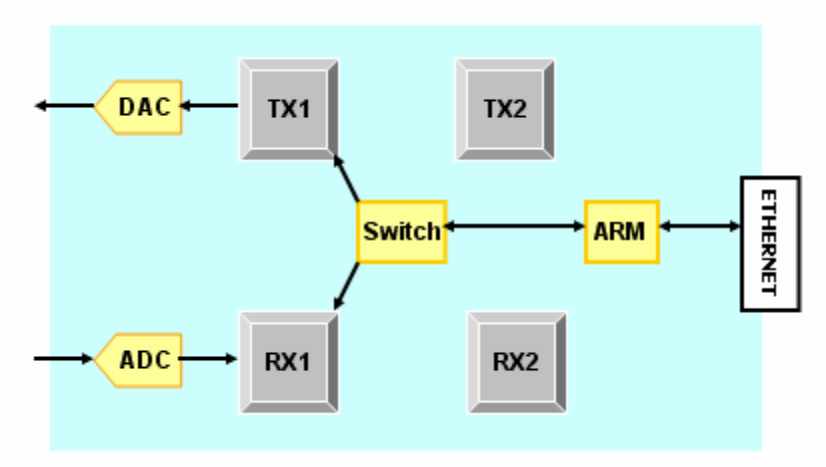

Figure 2-5. *Block Diagram of the SIP SDR Board* [18]

Chapter 3

ECA Hardware

3.1 ECA Hardware Overview

An elemental computing array is made up of a heterogeneous multiplicity of elements [5]. Each element is independently configurable to support several of operations. ECA-64 chips consist of 64 elements which are organized into four identical clusters. The structure of a cluster is shown in Figure 3-1. Each cluster is divided into four zones connected via through-queues. The elements and through-queues inside a zone are fully inter-connected and able to broadcast data simultaneously from all outputs. Through-queues also provide the local interconnection between clusters. Each cluster contains a message-manager providing the packet communication between any of the clusters, and the primary I/O communications of ECA [5].

The ECA is a dataflow computational model which is similar to the wormhole run-time reconfiguration model [19]. All operations inside of the ECA are data-driven (token-based) rather than clock-driven. A token is the 17th bit of a 16-bit data which represents a data. Each element executes an operation only when all input data (tokens) available. One advantage of the dataflow model is that users do not have to worry about the synchronization issue. To increase the efficiency of elements, each element can be shared by eight different data, called “contexts”, which virtually increasing the available element by eight times [5].

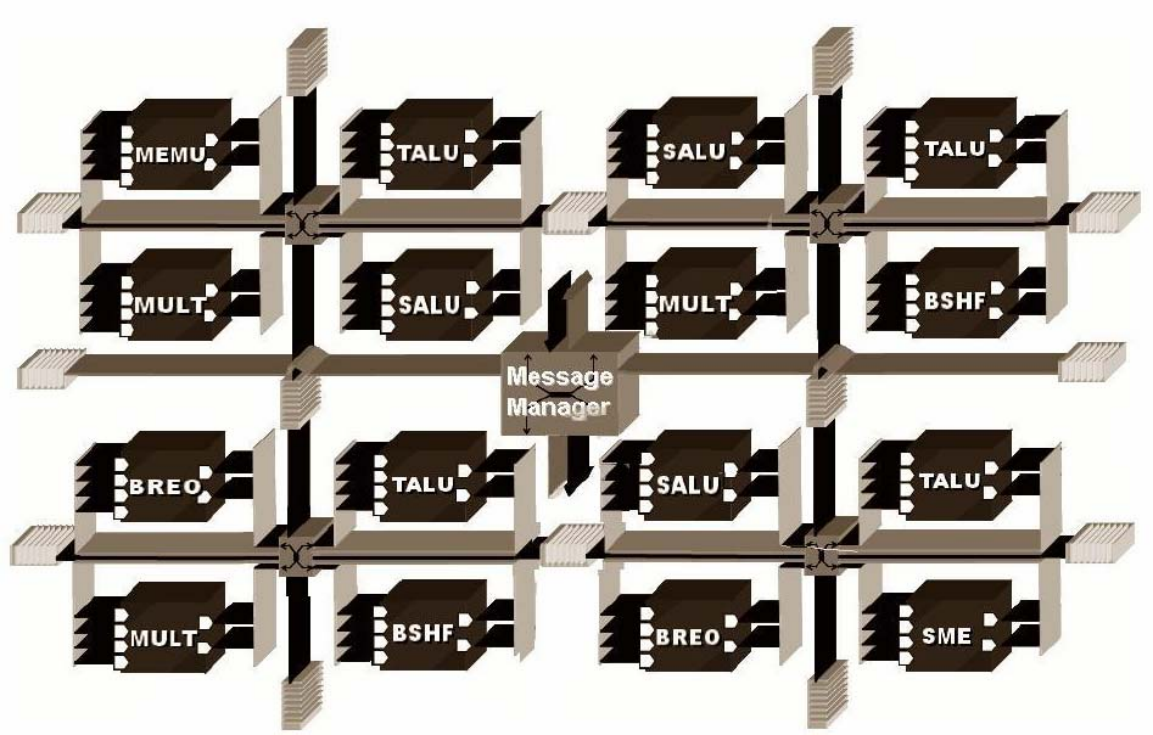

Figure 3-1. *Structure of the ECA Cluster* [5]

There are seven elements including MULT (multiplier), TALU (triple ALU), SALU (super ALU), BSHF (barrel shifter), BREO (bit reoderer), SME (state machine element), and MEMU (R/W memory). The MULTs provide the functions of dual 8-by-8 multiply accumulation, 16-by-16 multiply accumulation, and 32-bit multiply accumulation. Besides the regular logical and math functions like sorts, compares, ANDs, ORs, XORs, ADDs, SUBs, ABS, masking and detecting operation, the TALU can compute an add-compare-select operation within one clock cycle. Except the add-compare-select operation, the SALU can compute all other operations of the TALU. In addition, the SALU provides leading 1', leading 0', and 32-bit operations. BSHF is a barrel shifter that can execute 16-bit, 32-bit and 48-bit barrel shift, left/right/logical/arithmetic shifter, and circular operations. The BERO offers functions such as (un)packing, (de)interleaving, bit extraction, and simple conditionals. The SME is a sequential instruction-based element for executing arbitrary control code. The MEMU is a memory unit for data storage, look-up table (LUT), and random access.

3.2 Inherent Reliability of an Array of Elements

Since each type of the elements in the ECA chip has multiple copies, the loss of a single element due to a defect in the silicon, or long term aging phenomena, represents only a loss in capacity, not a failure of the total system. Extreme reliability can be achieved by taking advantage of kind of structure with the support of the development tools. Three significant features are gained through the ECA structure [5]:

- **Resiliency:** The ability of a system to continue to operate without disruption when sufficient computing resources exist, and to degrade gently while warning the user as resources becomes insufficient.

- **Robustness:** The ability of a system to tolerate to hard failures with minimal or no disruption when the system is continuously operating.

- **Elemental Testing:** The system can provide a self-test function without using external circuits. All elements within the ECA can test each other to determine the functional correctness of the other elements.

Chapter 4

ECA Development Environment

CoWare's Signal Processing Designer (SPD) and Element CXI's Crystal form a complete Development Environment for the ECA. The SPD provides a high-level graphical user interface (GUI) for quick start, and high-level simulations for function verification. Crystal converts SPD designs to the uuu files (which describes the physical hardware connection), and runs the low-level simulations. The ECA development model is shown in Figure 4-1.

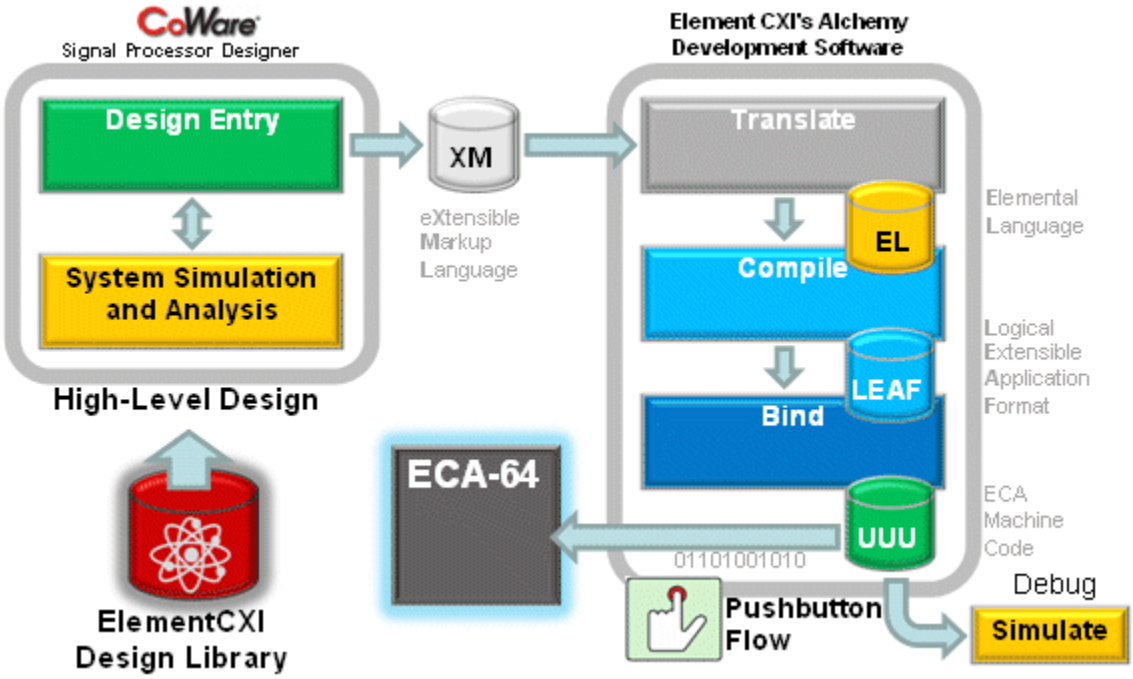

Figure 4-1. *ECA Development Model* [20]

4.1 CoWare SPD Overview

CoWare SPD, formerly Cadence's SPW, is a C-based modeling and simulation environment that facilitates structured modeling and model reuse across design teams [21]. Its efficient creation of complex DSP system models and extremely fast simulation makes Signal Processing Designer a good choice for multi-standard designs in the wireless and multimedia markets [21].

4.2 Crystal Overview

Crystal provides a graphical view of the physical layout of the ECA chip. Elements of the model are presented in an orbital layout using fractal principles to reveal several layers of architectural content simultaneously [22].

Crystal provides a convenient way to program the ECA chip, and test and verify the design through low-level simulations. The main functions provided by Crystal include:

- showing the virtual hardware layout of a design graphically;
- converting SPD design into uuu files that perform low-level functions;
- providing convenient methods to connect elements;
- running a low-level simulation for ECA designs; and
- applying various methods to import the test-bench data and output the simulated data.

4.3 Tools of Crystal

Though there are many useful tools in Crystal, only four were frequently used in the implementation of the ZigBee receiver. These tools are briefly described in the following sub-sections.

4.3.1 Main Window

The Main window presents the physical layout of the elements, and the connection and element utilizations of a design. The zoom-in/zoom-out function provides a convenient way to view a design at different scales, from a single context to the entire chip. Figure 4-2 shows the Main window of Crystal.

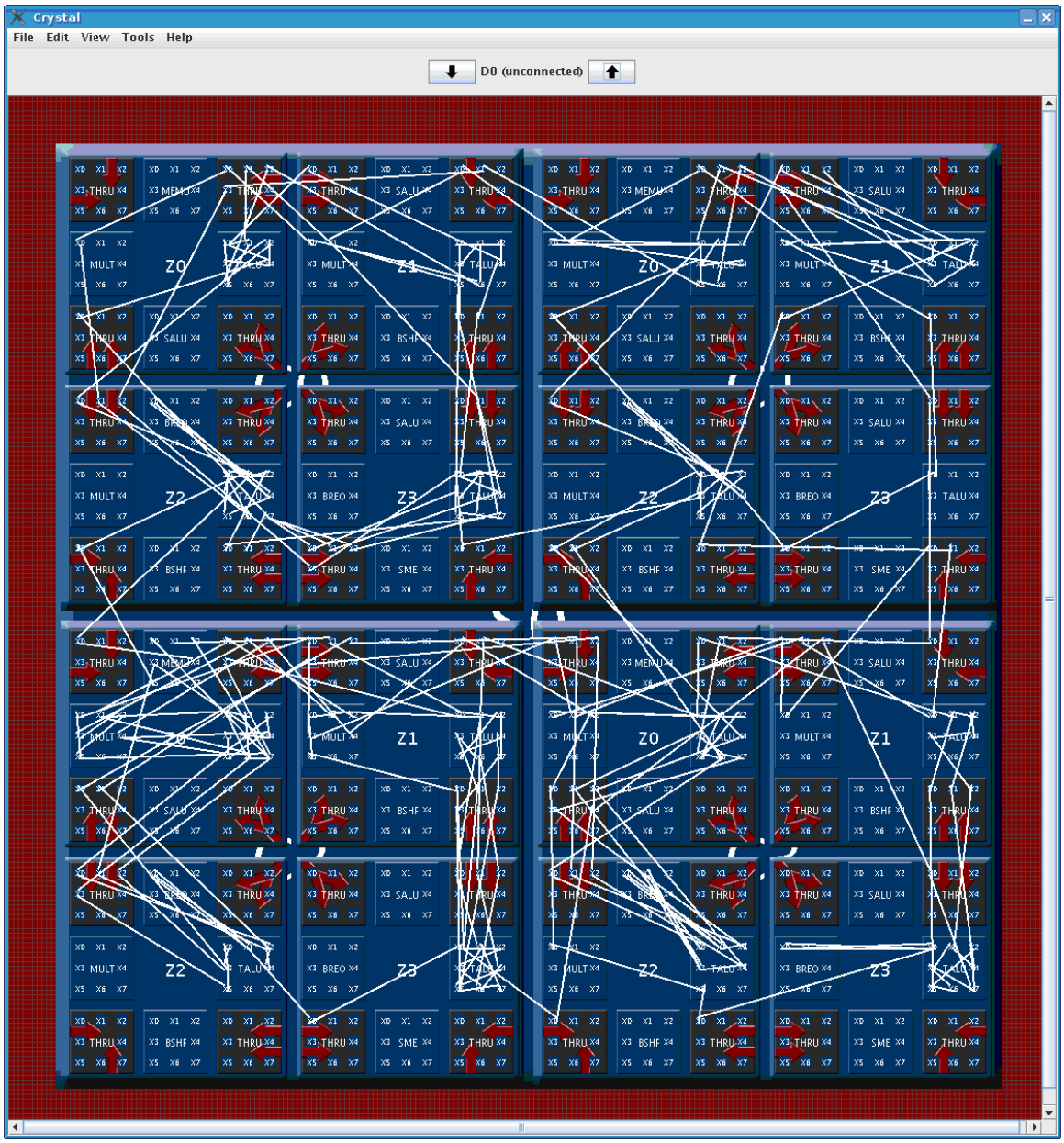

Figure 4-2. *Main Window of Crystal*

4.3.2 Logic Tool

The Logic tool is a separate window that displays the logical schematic of the computational design. A computation would typically have zero or more inputs and zero or more outputs. A designer can conveniently select a context, attach a test-bench data file to the input port, and set the output to dump data to a data file on a computer's file system. There are two kinds of views in this window. One is called the "physical view," which shows the sequence number of a context to help users locate this context in the main window. The other view is called the "logical view," which shows the function of each context. Figure 4-3 shows the window of the logic tool.

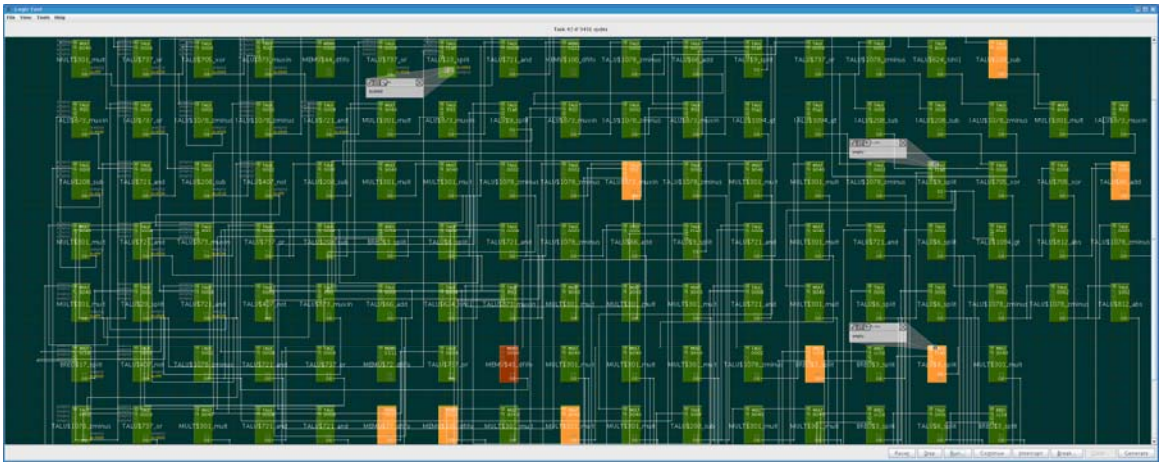

Figure 4-3. *Window of the Logical Tool*

4.3.3 Transaction Tool

The Transaction tool is also another selectable tool that displays the system transactions recorded in the event log of the simulator. Users can debug their design by analyzing those transactions. Figure 4-4 shows the window of the transaction tool.

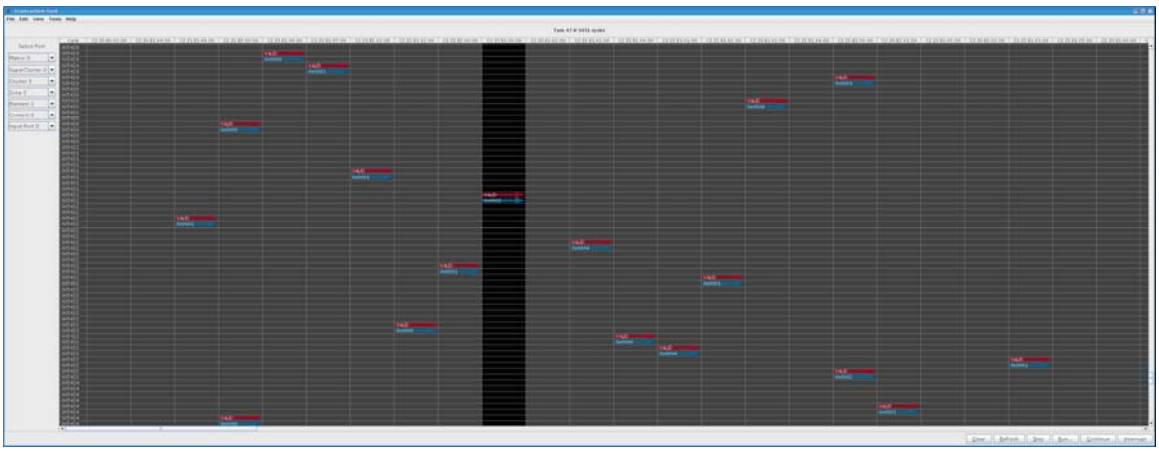

Figure 4-4. *Window of the Transaction Tool*

4.3.4 Simulator

Crystal is the front-end GUI for a separate simulator/server, which simulates the action of the Elemental chip, and communicates with a client via a TCP/IP socket using a documented protocol based loosely on HTTP [22].

Chapter 5

System Designs

In this chapter, the system design of the ZigBee system is discussed. The first section describes the system overview, while the remaining sections discuss each sub-module of the ZigBee system in detail.

5.1 System Design Overview

Since the ZigBee system is targeted to low-cost, low-data rate, low-power systems, simplicity is one of the important concerns in the design rules: the whole implementation can easily fit in an ECA-64 chip. However, some software limitations have been found in the latest release of the ECA development tools. All SALU elements (50% of the elements needed for the ZigBee system) cannot be accessed. This issue should be solved in the next release of the tools. Since only half of the resources are currently available for this thesis, two necessary simplifications were made to fit the ZigBee receiver into the usable half of the chip. One result of these simplifications is lower performance of the receiver.

The first simplification was to use the look-up-table method instead of the usual de-spreading method, which calculates the correlations of spreading sequence to de-spread, to recover the data. It may cause some performance degradation since the look-up-table method is hard decision and cannot fully take advantage of the spreading sequences. The second simplification was to use only the Q-channel signal to recover the data. Since half of chips are dumped, which means 50% power loss, it will cause 3dB performance loss. These simplifications are tolerable because the focus of this

thesis is mainly on the viability of the system implementation, and, in fact, the implemented ZigBee receiver is entirely functional. Furthermore, the performance loss can be easily recovered once all the resources on the ECA chip are available and a de-spreading module is used.

The block diagram of the overall ZigBee receiver design is depicted in Figure 5-1. A ZigBee system uses an offset-QPSK modulation scheme that requires two channels to transmit signals; hence, the implemented receiver has two channel inputs. Since each of the four data bits of an offset-QPSK symbol is mapped to a sequence of 32 *chips* and transmitted in the two channels at the transmitter-end, the first stage of the receiver is responsible for recovering the two chip-streams from the two-input signals. This is accomplished by two modules referred to as *chip-recoverers*.

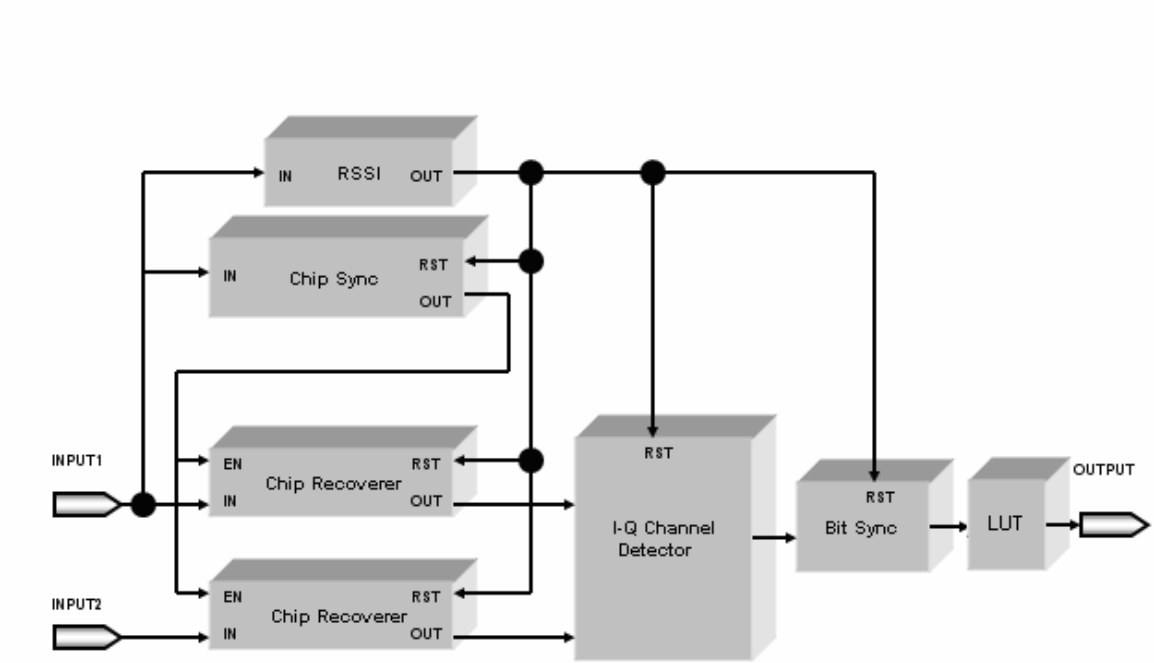

Figure 5-1. *Block Diagram of the Overall ZigBee Design*

The two chip-recoverers are controlled by a chip synchronizer for aligning the chips and a received-signal-strength indicator (RSSI) for detecting the presence of a valid input signal. Once the two chip-streams are recovered, an I-Q channel detector

determines the Q-channel chip stream in the two chip streams and locates the bit stream head.

To simplify this particular receiver implementation, the I-channel chip data is ignored and only the Q-channel chip data is used to extract the bit data by a bit-synchronizer module.

5.2 RSSI Design

The ZigBee system transfers data frame by frame. Since the ZigBee protocol specifies that receiver has to do synchronization at the beginning of each frame while it is receiving. Therefore, it is necessary to have a module to turn on and off the receiver to control the sync and re-sync. This module is called the RSSI module. When the strength of an incoming signal is higher than the preset threshold, this module turns on the entire system, and the sync modules try to synchronize the incoming signal. When the signal strength falls below the threshold, the system is turned off and the sync module is reset. The RSSI module does not care whether the incoming signal is a desired signal or not - other modules handle this task.

Figure 5-2 shows the block diagram of the RSSI module. The magnitudes of the latest incoming samples are summed, and the sum is compared with the threshold. If the sum is larger, the RSSI module turns the system on. Otherwise, it keeps the system off.

5.3 Chip-Synchronizer Design

When the RSSI module detects a strong signal and turns the receiver on, the chip-synchronizer starts to synchronize with one of the inputs. In this thesis, the period of each chip is fixed at four samples per chip, and a zero-crossing detection method is employed. If the chip-synchronizer detects two zero-crossings in a period of four

samples, it synchronizes with the signal and enables the chip-recoverers at the correct time.

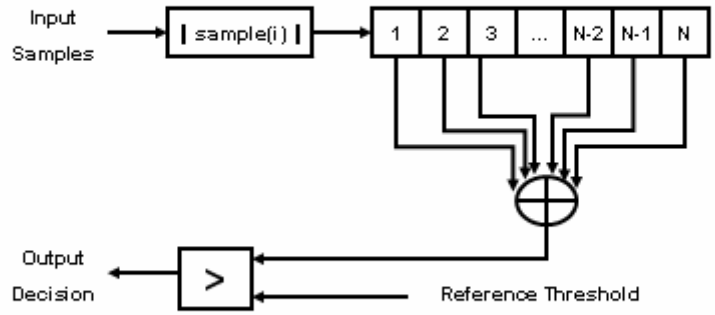

Figure 5-2. *Block Diagram of the RSSI Module*

Since input samples are in 2's complement format, a zero-crossing detector can easily find the zero-crossing point by comparing the most significant bits of every pair of neighboring samples. Once two zero-crossings are found within a period of four samples, a 2-input OR-gate with one feedback input holds it until the OR-gate is reset by the RSSI module. The block diagram of the chip-synchronizer is shown in Figure 5-3. The reset mechanism of this module is implemented by the two 2-input AND-gates feeding the OR-gate. When the reset signal from the RSSI module becomes low, the output of the chip-synchronizer also becomes low, which means the signal is unsynchronized.

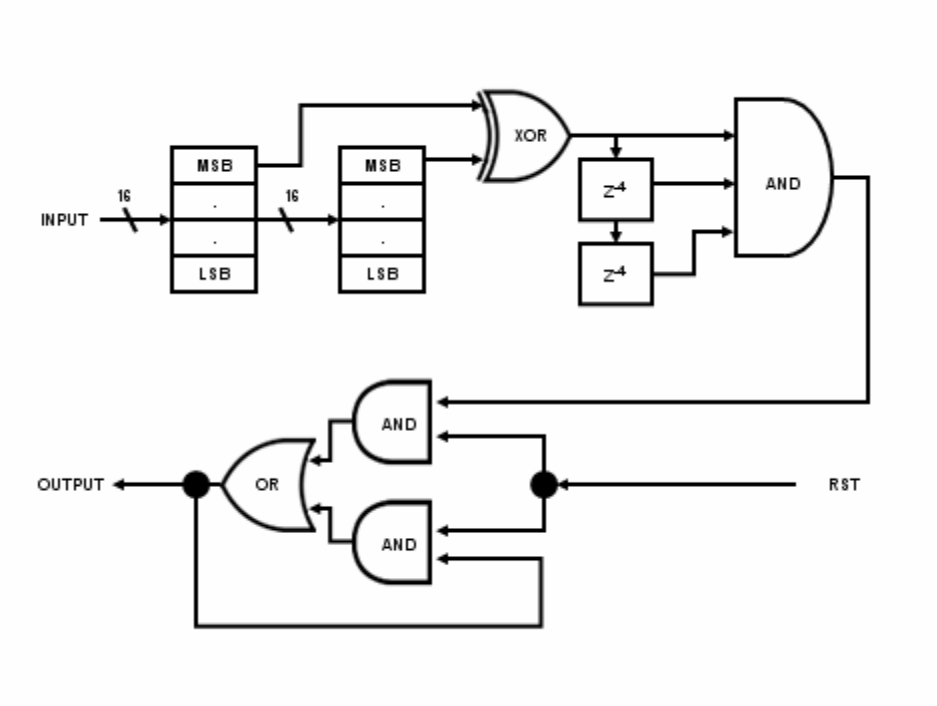

Figure 5-3. *Block Diagram of the Chip-synchronizer Module*

5.4 Chip-Recoverer Design

Once the chip-synchronizer locks onto the input signal, it enables the chip-recoverers. In the chip-recoverer shown in Figure 5-4, a 2-bit control counter is enabled by the enable signal from the Chip-synchronizer module. This control counter resets the accumulator after every four samples. Four consecutive samples are added in the accumulator, whose value is then compared to the constant value 0, to generate the output (which is the most confident decision). This mechanism is realized by a multiplexer controlled by the enable signal.

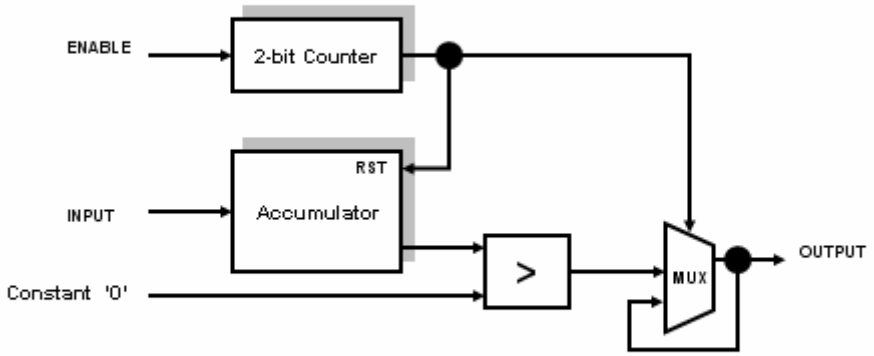

Figure 5-4. *Block Diagram of the Chip-Recoverer Module*

5.5 I-Q Channel Detector Design

The two recovered chip sequences are fed into the I-Q channel detector. This kind of synchronization method is a hard-decision method which has worse performance than a soft-decision method. However, the purpose of applying the spread spectrum technique in the ZigBee system is not to achieve higher performance but to reduce interference with the neighboring channels. Moreover, the spreading gain in ZigBee is only eight. Therefore, the performance difference between the hard-decision method and the soft-decision method is negligible.

Figure 5-5 shows the block diagram of the I-Q channel. This structure has two spreading-code detectors, one for each of the two inputs. Once one of the detectors finds the input chip sequence, the Q-channel data is then determined. The Q-channel is then output to the bit-synchronizer, and the I-channel is ignored (via the OR gate). This situation persists until the module is reset. The basic idea of the structure is that the first synchronized input is the Q-channel input, since each ZigBee data frame has a fixed 32-bit preamble of zeros for distinguishing the I-Channel and Q-Channel. Using a

sequence detector, each of the two detectors is looking for either the specific chip sequence “1101100111000010” or its inversion “0010011000111101.”

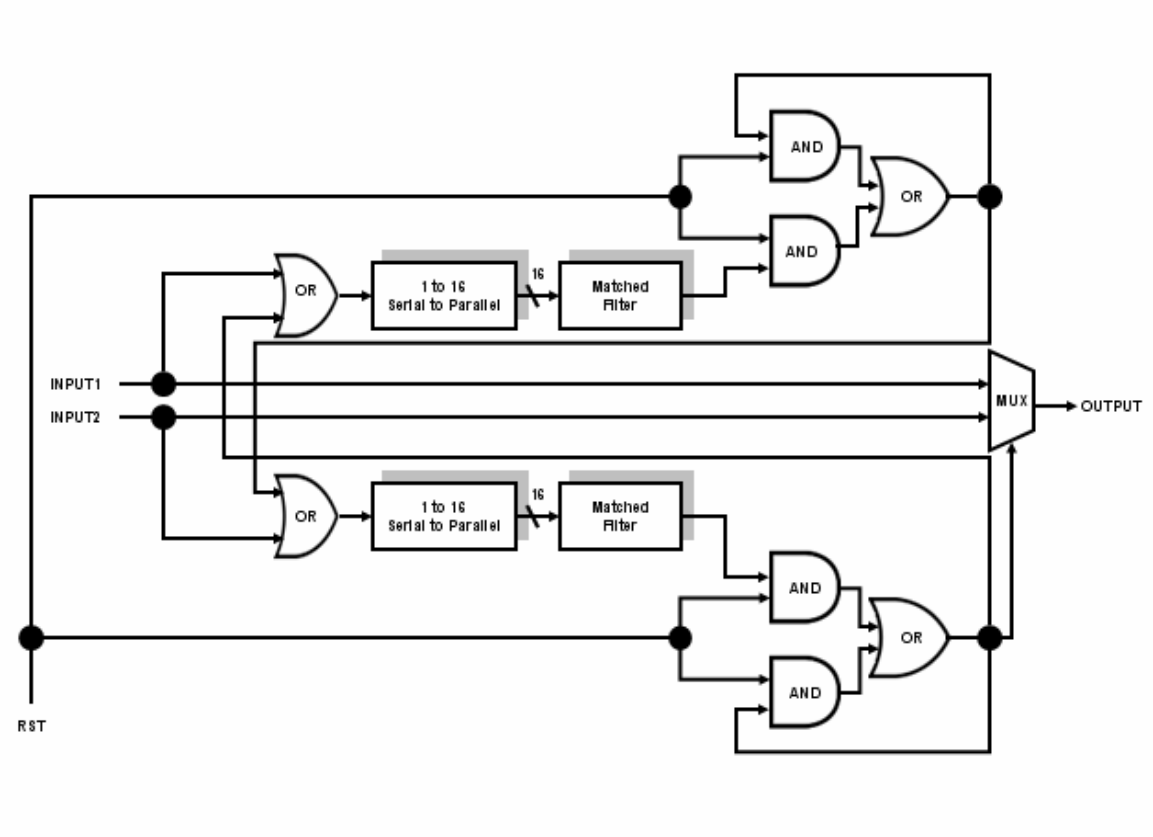

Figure 5-5. *Block Diagram of the I-Q Channel Detector Module*

5.6 Bit-Synchronizer Design

The bit-synchronizer has a similar structure to the I-Q channel detector. It provides three functions including frame-delimiter sequence detection, sequence inversion correction and bit synchronization. Since the input chip-stream could be inversed due to phase delays and the receiver has no side information about the inversion; this bit-synchronizer module is needed to detect whether the chip stream is inversed and correct the stream that is inverted. As shown in Figure 5-6, two sequence-detectors based on the matched filters seek the frame-delimiter sequence “0110011100001011” or

its inversion. If the frame-delimiter sequence is found, the input chip-sequence is not inversed. Otherwise, if the inversed frame-delimiter sequence is found, the input sequence is inversed and needs to be corrected. Once the frame-delimiter sequence or its inversion sequence is captured, the sequence detector enables the 4-bit counter at correct time to achieve the bit-synchronization.

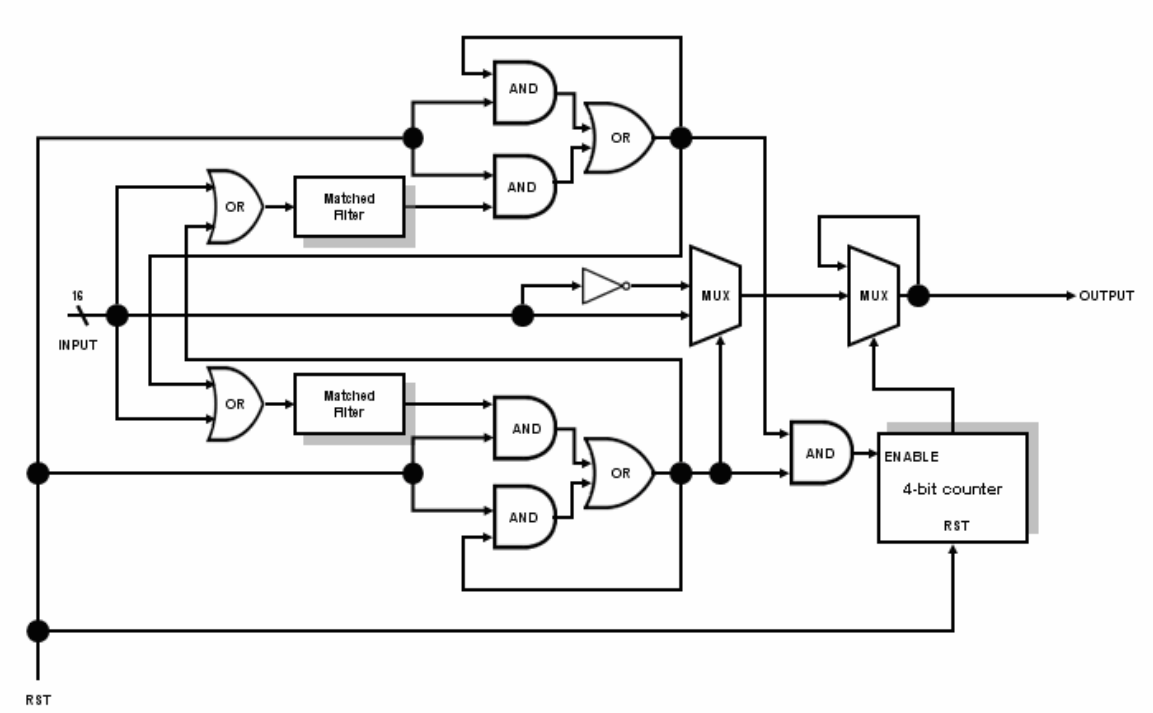

Figure 5-6. *Block Diagram of the Bit-synchronizer Module*

5.7 Symbol-Recoverer Design

The symbol-recoverer design is based on two lookup tables (LUTs) with 8-bit input and 256 conditions. The design goal of this symbol-recoverer module is to minimize the chip utilization while it keeps a reasonable error-correction capacity. Figure 5-7 shows the block diagram of the symbol-recoverer module.

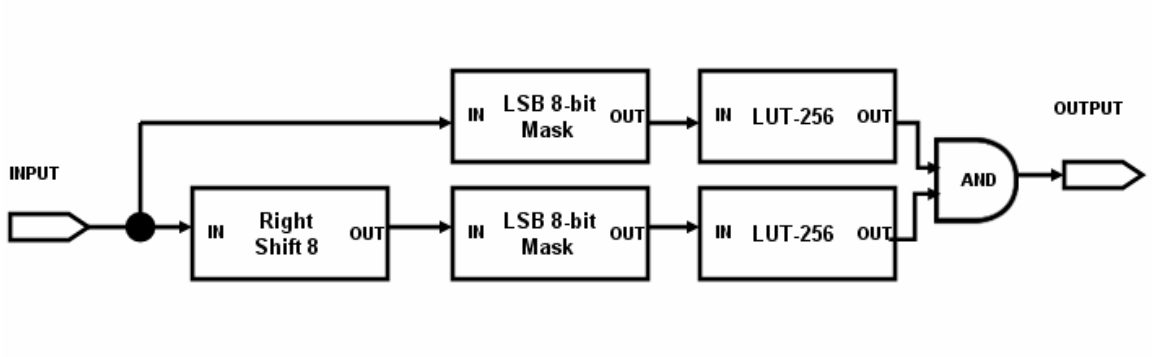

Figure 5-7. *Block Diagram of the Symbol-Recoverer Module*

The 16-bit inputs are split into eight most-significant-bits (MSBs) and eight least-significant-bits (LSBs) and fed into those two LUTs correspondingly. The correctness of the outputs of the LUTs depends on the number of bit (chip) errors of each 8-bit input. The output rule of the LUTs is when only one chip-error occurs, the error can be corrected and the outputs of the LUTs are the correct symbols; when two chip-errors occur, the outputs of the LUTs are set to "1111"; and when the chip-errors are more than two, the outputs of the LUTs could be "1111" or wrong symbols depending on whether the incorrect sequence looks like the other sequences representing the other symbols. Therefore, the output symbols are 100% correct if chip-errors are less than three. When chip-errors are more than two, the outputs still have certain probability to be correct symbols. For example, a 16-bit sequence representing symbol "1010" is fed into this symbol recoverer. There are two chip-errors in the most significant 8-bits and one chip-error in the least significant 8-bits. According to the LUT output rule mentioned above, the output of the LUT for the most significant 8-bit is "1111" and the output of the LUT for the least significant 8-bits is "1010". After the two outputs go through the AND gate, the final output is the correct symbol "1010". Actually, in such cases, even the chip-errors in the most significant 8-bit are more than two, the final symbol-output can be correct as long as the output of LUT for the most significant 8-bits is "1111" and there's no or only one chip-error in the least significant 8-bit.

The whole design of this ZigBee receiver aims at simplicity and resource efficiency. In the next chapter, the simulation results and detail implementation of the ZigBee receiver on ECA-64 are presented.

Chapter 6

Implementations and Simulation Results

The ZigBee design presented in Chapter 5 was implemented using the CoWare SPD design entry tool using element leaf modules. Since SPD provides a friendly drag-and-drop graphical user interface (GUI), the implementation and modification was an easy and efficient procedure. Each functional block in Figure 5-1 was implemented and tested individually before full system integration, implementation, and simulation. A sequence of samples of pseudo ZigBee signals with Additive Gaussian White Noise (AWGN) was generated using Matlab as the simulation source to test the BER performance of this ZigBee receiver. Some limitations in the initial release of Element CXI's Crystal simulator prevented the final performance verification from being completed. Hence, an alternative simulation method was employed to finish the BER performance verification.

6.1 CoWare SPD Implementations

Detailed implementations of functions such as control counters, accumulators, and sequence detectors are described in the first sub-section. In the next sub-section, thorough implementations of each functional module in Figure 5-1 are introduced. In the last sub-section, the complete ZigBee receiver implementation is presented.

6.1.1 Implementations of Functions

The first function is the control counter for counting the input tokens and resetting the accumulator in the chip-recoverer. The output of this counter has only two values, zero and one, and the output is fed to the reset ports of the accumulator. Figure 6-1 shows the implementation of the counter. It is a loop consisting of an OR-gate, an AND-gate, and an *n*-word-long shift register. The length of the shift register determines the counting period. When a logic '1' enters input to this loop through the enable port, the counter outputs a logic '1'. The enable signal must then return to a logic '0' to allow the counting to continue. The counter now acts as a circular buffer, recycling the initial logic '1' after several unit delays. An active-low reset signal removes the circling '1' at the AND-gate to disable and reset the counter.

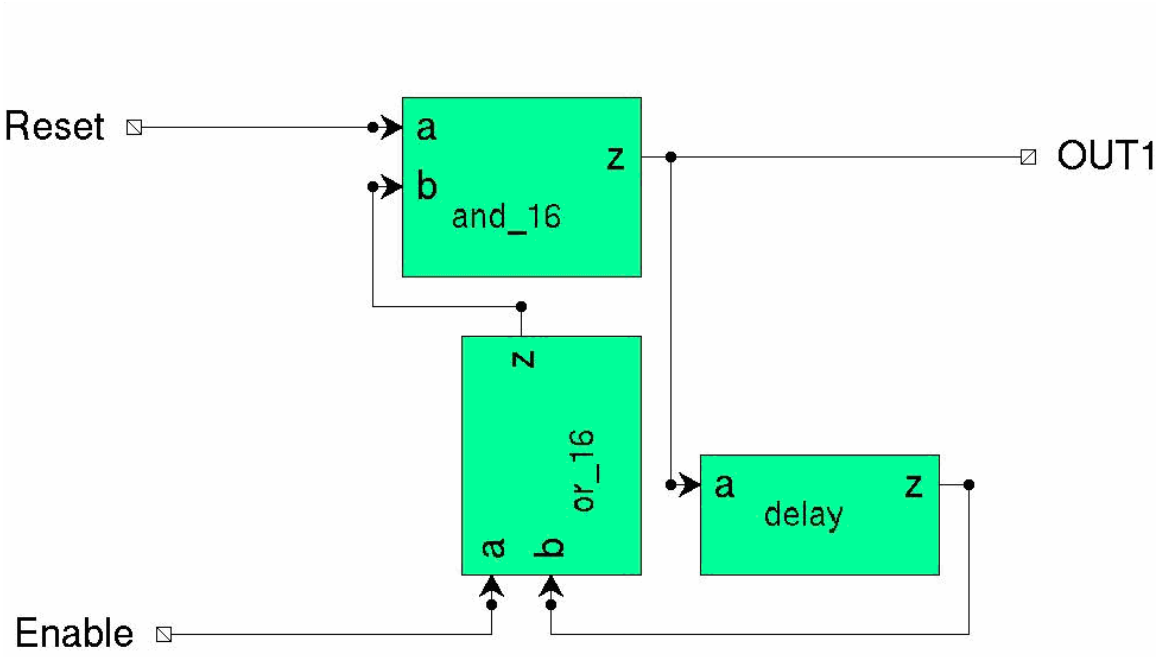

Figure 6-1. *Gate-Level Circuit of the Control Counter*

The enable signal from the output port is a constant one when the enable is active; hence, it cannot directly drive the control counter. An edge detector is needed for capturing the rising edge. The gate-level circuit of the edge detector is shown in Figure 6-2. This edge detector is formed by a one-token-delay unit and an XOR-gate. When

a rising edge or falling edge comes in, it outputs a logic one which drives the control counter. In this ZigBee receiver, a falling edge of the enable signal only comes at the moment of the reset of entire system by the RSSI module. Therefore, the detections of falling edges do not affect the control counter driven by this edge detector.

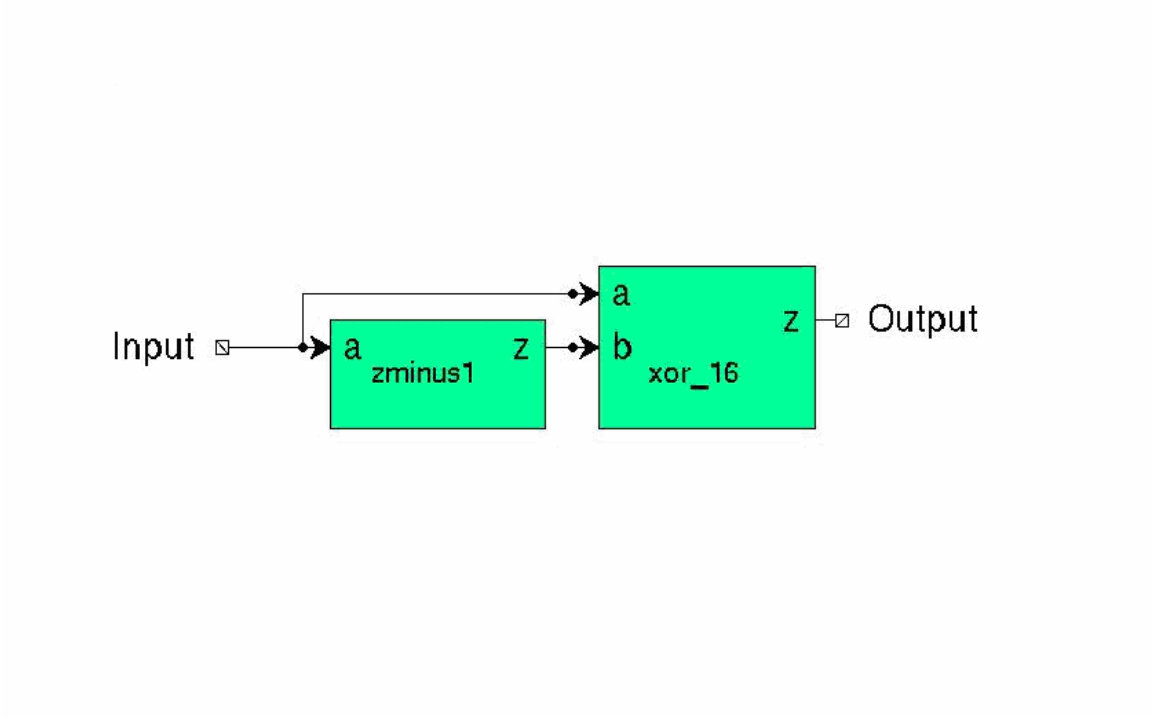

Figure 6-2. *Gate-Level Circuit of the Edge Detector*

Figure 6-3 shows the gate-level circuit of the accumulator. When the reset signal is active, the accumulator outputs the input value. Otherwise, the accumulator outputs the sum of the input value and the previous sum, which implements the accumulation operation.

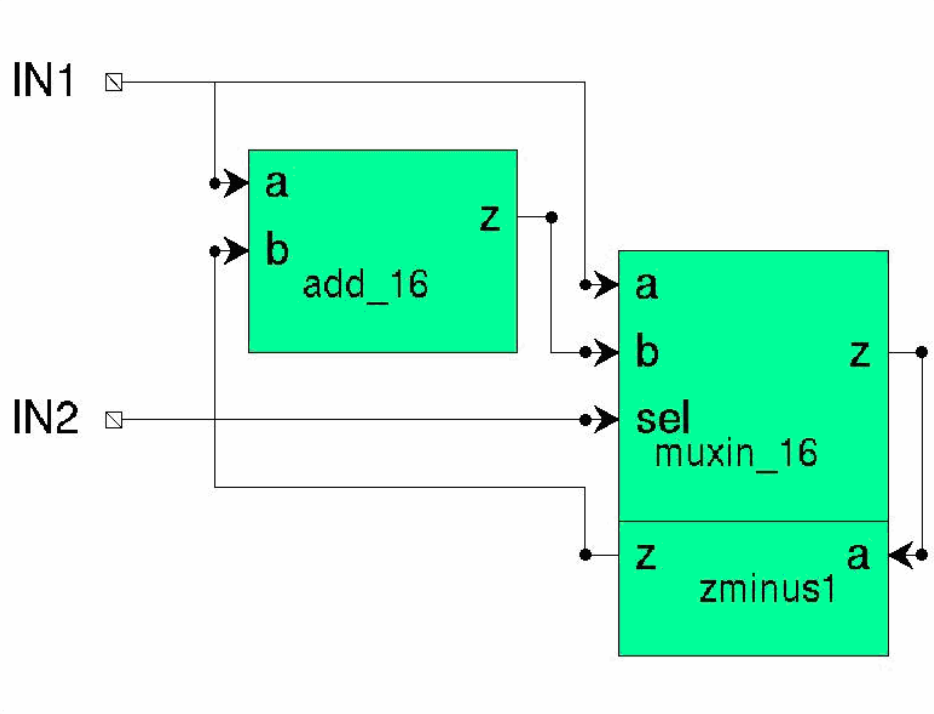

Figure 6-3. *Gate-Level Circuit of the Accumulator*

A latch is needed for selecting the desired data from an input data sequence; however, there is no latch unit in the element leaf-module library. Hence a simple implementation of a latch is designed using a two-input multiplexer. One of the inputs of the multiplexer was fed by the output to retain the data when the select-port input (equivalent to the enable input) is low. When the enable signal is high, the latch output is driven directly by the data input, as shown by Figure 6-4.

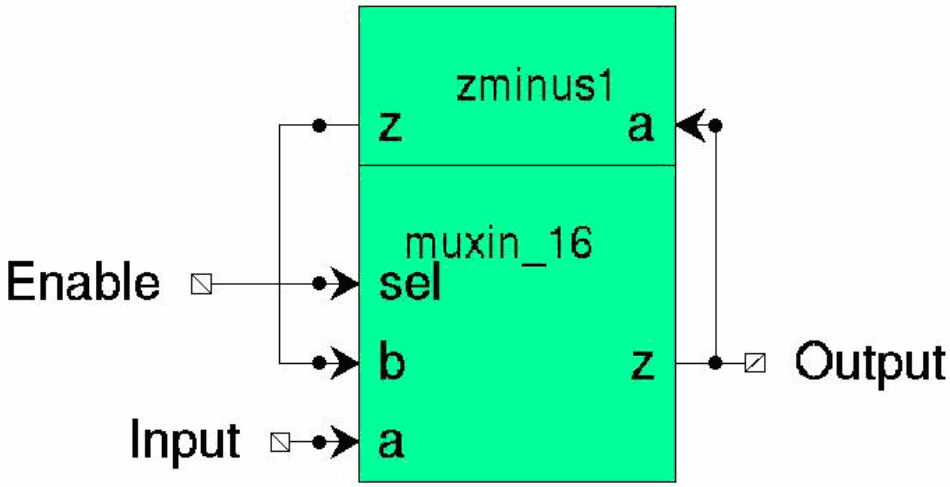

Figure 6-4. *Implementation of the Latch*

6.1.2 Implementations of ZigBee Modules

Following the design in Figure 5-1 in Chapter 5, these modules of the ZigBee receiver were implemented in the CoWare SPD. These gate-level implementations are shown in figures 6-5 through 6-9. Each implementation is individually converted to a uuu file and tested in Crystal to verify functional correctness. All modules successfully passed the test in Crystal.

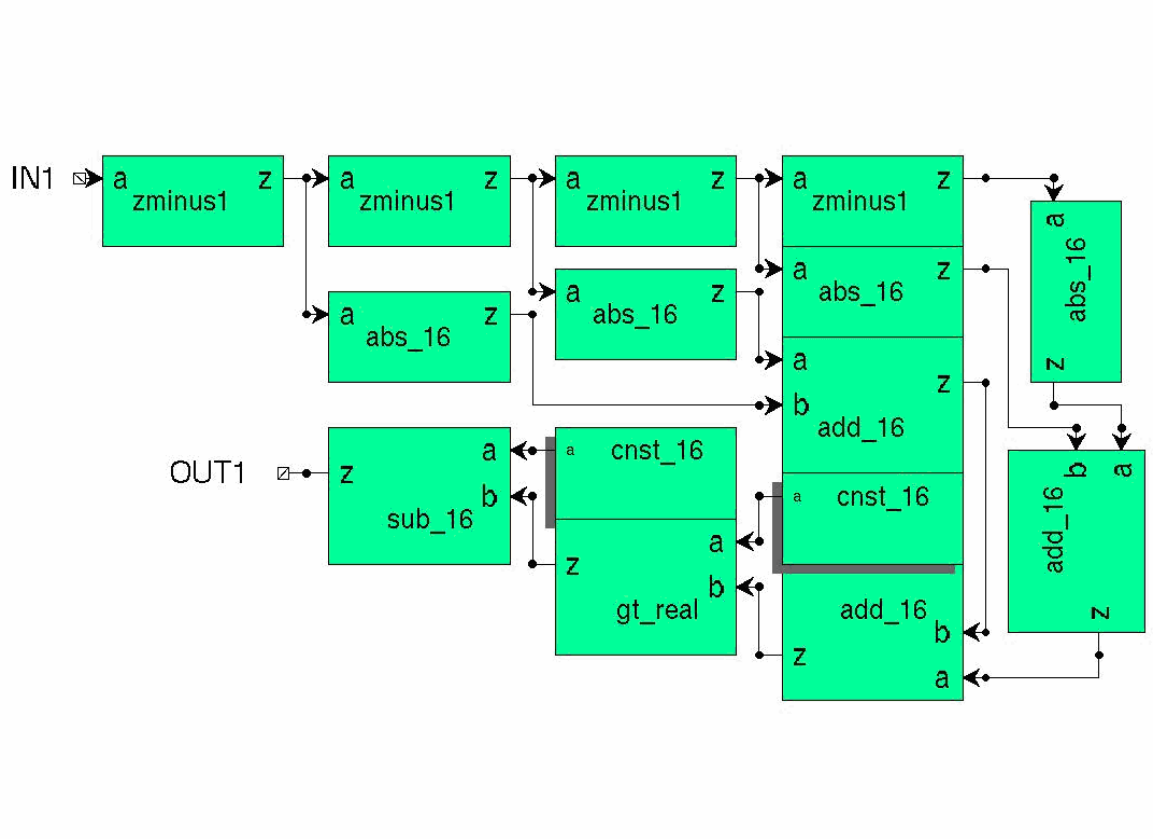

Figure 6-5. *Gate-Level Implementation of the RSSI Module*

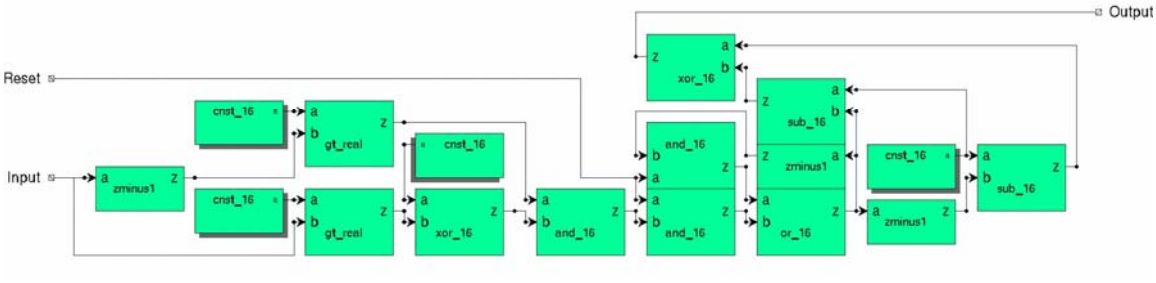

Figure 6-6. *Gate-Level Implementation of the Chip-Synchronizer Module*

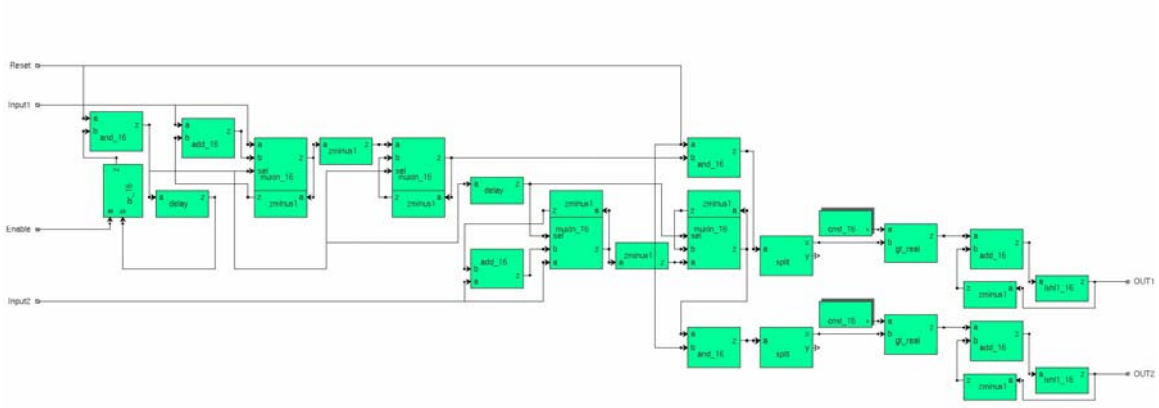

Figure 6-7. *Gate-Level Implementation of the Chip-Recoverer Module*

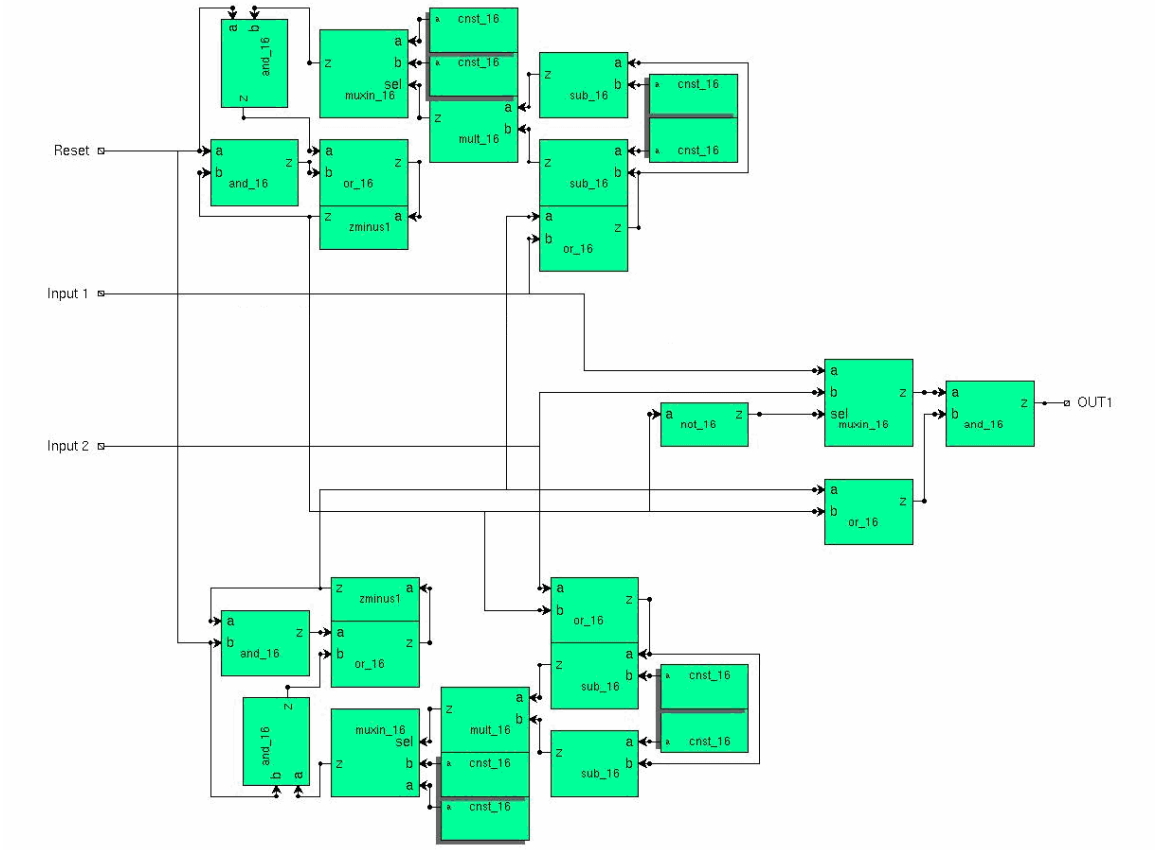

Figure 6-8. *Gate-Level Implementation of the I-Q Channel Detector*

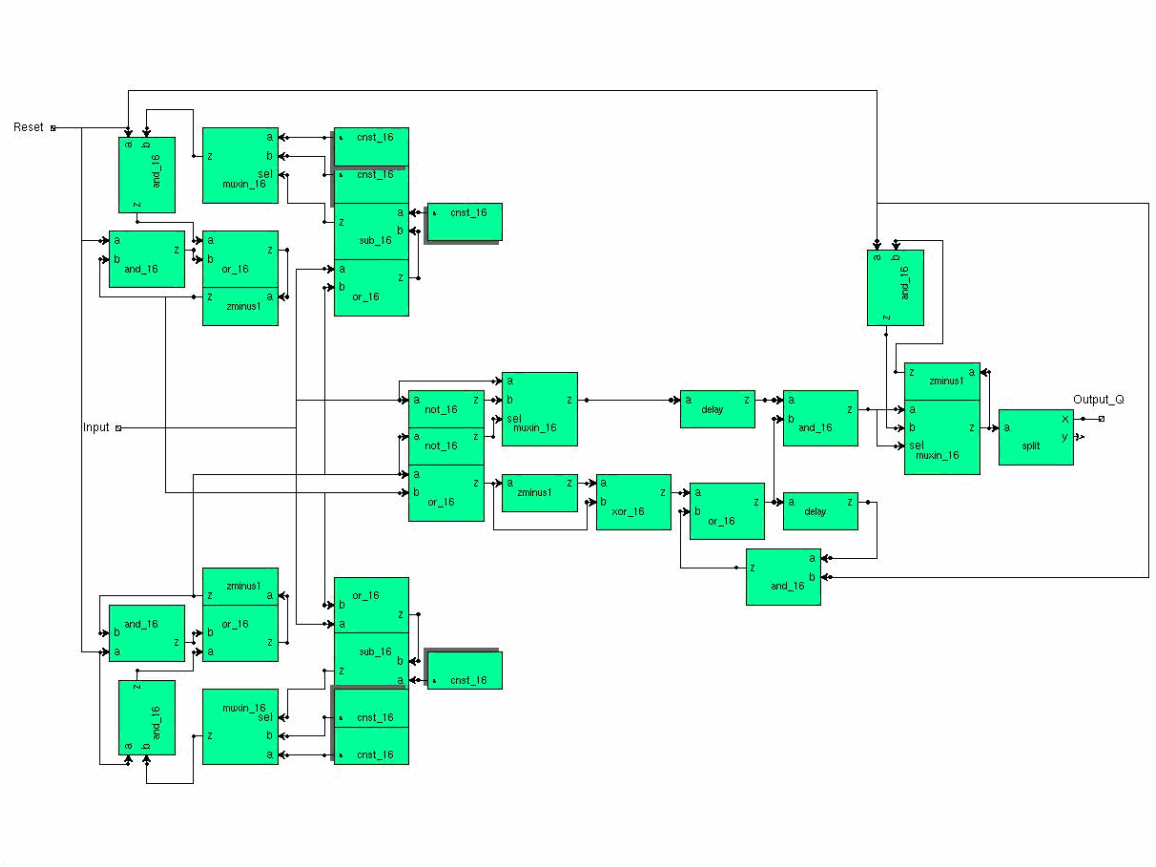

Figure 6-9. *Gate-Level Implementation of the Bit-Synchronizer Module*

6.1.3 Implementation of the ZigBee Receiver

Although all modules work correctly in Crystal simulation, the initial integrated system failed to function properly. The simulation data and transactions of this initial system were analyzed in the transaction tool of Crystal and the source of the problem was identified as an issue with the token mechanism. The reset signal, also known as the reset token from the RSSI module is fed to all of the other modules. When an output reset token splits into two tokens to be consumed, the one that goes directly to the bit-synchronizer module has to wait for the arrival of the other token which goes through the chip-sync module. Since the RSSI module cannot process the other input token and generate the new output reset token until the current output token is consumed, a large delay from the chip-sync module to the bit-synchronizer module results in a reduced

output-token rate as well as a low data processing rate that is much lower than the input data rate. As the result, the system halts. The solution to this issue is to insert a delay into the direct connection between the RSSI module and the bit-synchronizer module so that the system can work in a pipelined mode correctly.

Figure 6-10 depicts the complete implementation of the ZigBee receiver with the delay compensations. These delays are created by multipliers, delaying the data value by multiplying with a constant one.

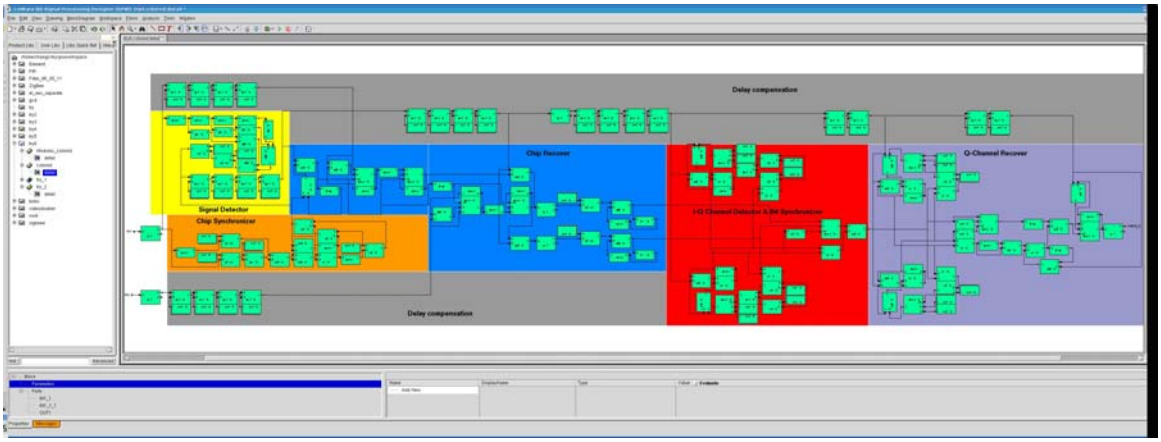

Figure 6-10. *Complete Implementation of the ZigBee Receiver*

6.2 Simulations and Results

6.2.1 Test-Bench Data Generation

The test-bench data for the input to the system simulation were generated in Matlab. The data generation procedure follows the bit-to-chip mapping rule specified by the IEEE 802.15.4 standard and the symbol-to-sample mapping rule of the offset-QPSK modulation [14]. Figure 6-11 shows the procedure of the test-bench data generation.

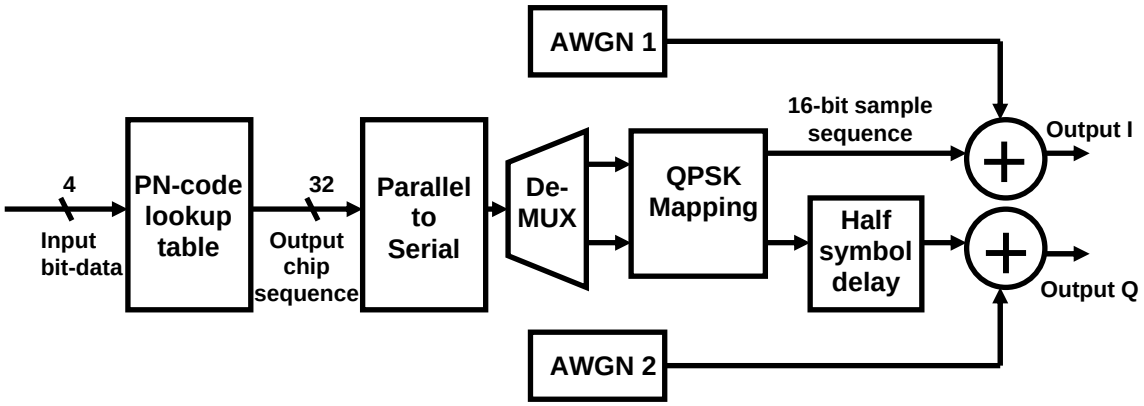

Figure 6-11. *Generation Procedure of the Test-Bench Data*

Each of 4-bit input data is mapped to a corresponding 32-chip sequence according to the PN-code table specified by the IEEE 802.15.4 standard [14]. This sequence splits into two sub-sequences (channels) and is fed into the QPSK mapping module. In the QPSK mapping module, each chip is mapped into a half-sine formed by four 16-bit samples. To produce the offset-QPSK signal, one of the two sample sequences is delayed by two sample periods. Finally, each sample sequence is added with an AWGN noise to simulate channel environment. Once the test-bench data is generated, the ZigBee receiver is ready to run simulations.

6.2.2 Functional Correctness Verifications

One limitation of Crystal simulator is that the simulation can run only 50,000 steps before the simulator experiences a memory overflow error. This length of simulation period is not long enough to finish the BER performance simulation but is long enough to verify the functional correctness of the ZigBee receiver. To accomplish the verification, the test-bench data frame is packed in a short length to assure the receiver can finish the reception with 50,000 steps. After the receiver passed this verification, a variety of non-ZigBee signals are fed into the receiver to make sure this receiver will not receive the undesired signal by a mistake. The results show this ZigBee receiver passed all verifications successfully.

6.2.3 Acquisition Performance Simulations

Two main factors of the spreading-sequence acquisition, probability of miss and probability of false alarm, are simulated to evaluate the performance of the acquisition. Figure 6-12 shows the acquisition performance of the ZigBee receiver. The probability of false alarm is pretty low since the receiver requires a whole sequence match to achieve the synchronization. The probability of a miss is significantly higher in low E_b/N_0 cases, but the probability of miss is acceptable in reasonable E_b/N_0 cases. Since the ZigBee systems are using acknowledgement frames to confirm the receptions of data frames, the transmitter will send the data frame second time if there is no acknowledge frame back from receiver. Hence the data will not be lost when a data frame loss happens, which will only increase the data transmission overhead and decrease the data transmission rate. A better sequence acquisition module can be implemented by using maximum likelihood estimation method with more cost of chip resources when SALU elements are available.

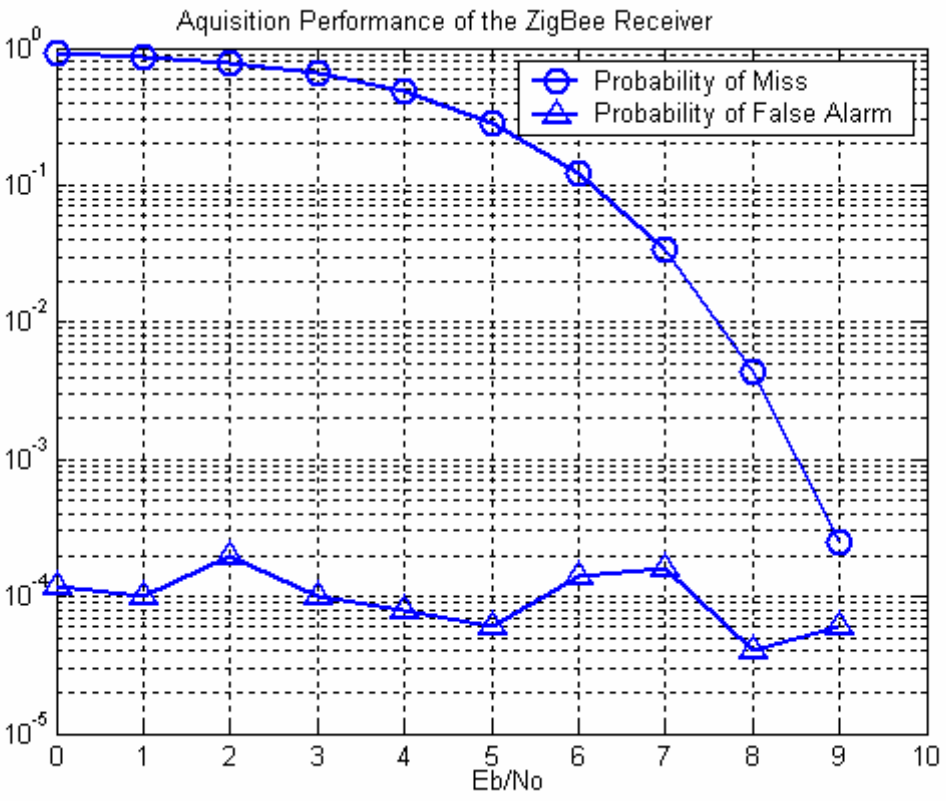

Figure 6-12. *Acquisition Performance of the ZigBee Receiver*

6.2.4 System Performance Simulations

To obtain the Bit Error Rate (BER) performance, an alternative simulation method is employed. Since the ECA is a token-based dataflow machine, functional blocks in Figure 5-1 are completely independent from the others and are freed from synchronization issues between these functional blocks. Therefore, these blocks can be viewed as black boxes. If a Matlab module and a functional block have the same input signals and produce the same outputs, this Matlab module can replace this functional block. Hence, the entire system can be simulated in Matlab equivalently.

Two kinds of performance simulations had been done in Matlab. One is the chip error rate test to verify if the offset-QPSK demodulator of the ZigBee receiver works correctly. The simulation result is shown in Figure 6-13. The performance curve matches the theoretical curve nearly perfectly.

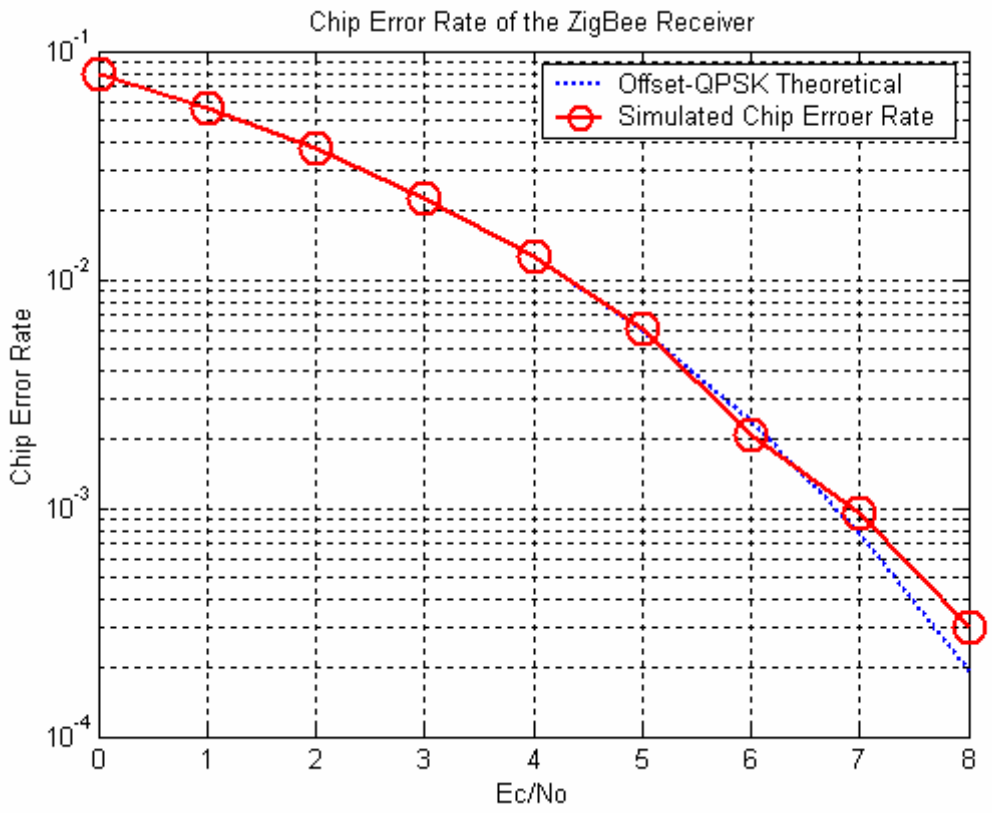

Figure 6-13. *Chip Error Rate of the ZigBee Receiver*

The other one is the BER test for the evaluation of the whole system performance. The Figure 6-14 shows the results. The baseline of this test is the BER performance of the theoretical 16-ary orthogonal system. The ideal case of the ZigBee system is simulated in MATLAB by using maximum likelihood estimation. Since the ZigBee receiver uses 16-ary quasi-orthogonal spreading sequence, the performance of the ideal ZigBee system is slightly worse than that of the 16-ary orthogonal system. The simulation result of the actual ZigBee system is about 4.5dB worse than the ideal one. The 3dB degradation of the 4.5dB is caused by the lost of the I-channel data, and the rest of 1.5dB degradation is the performance difference between the hard-decision and the soft-decision mentioned in Chapter 5. The simulation result matches the prediction made in Chapter 5.

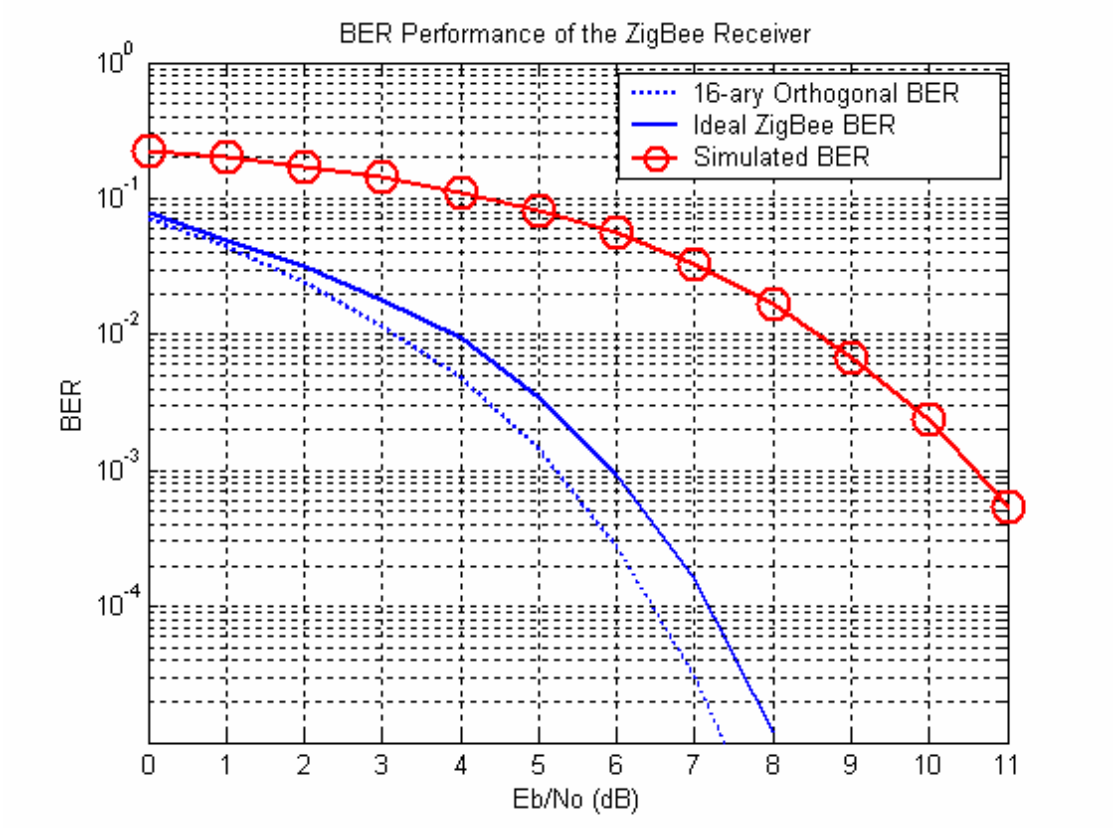

Figure 6-14. *BER Performance of the ZigBee Receiver*

6.3 Implementation Results

6.3.1 Chip Utilizations

The implementation report from Crystal is shown below:

Usage statistics:

Number of used BREOs in the design : 6 out of 64 (9%)
Number of used MEMUs in the design : 6 out of 32 (18%)
Number of used MULTs in the design : 24 out of 96 (25%)
Number of used TALUs in the design : 108 out of 128 (84%)

Placement statistics:

Number of contexts in Cluster 0 Zone 0 = 8
Number of contexts in Cluster 0 Zone 1 = 11
Number of contexts in Cluster 0 Zone 2 = 7
Number of contexts in Cluster 0 Zone 3 = 9
Number of contexts in Cluster 0 = 35
Number of contexts in Cluster 1 Zone 0 = 8
Number of contexts in Cluster 1 Zone 1 = 1
Number of contexts in Cluster 1 Zone 2 = 8
Number of contexts in Cluster 1 Zone 3 = 6
Number of contexts in Cluster 1 = 23
Number of contexts in Cluster 2 Zone 0 = 19
Number of contexts in Cluster 2 Zone 1 = 15
Number of contexts in Cluster 2 Zone 2 = 8
Number of contexts in Cluster 2 Zone 3 = 9
Number of contexts in Cluster 2 = 51
Number of contexts in Cluster 3 Zone 0 = 16
Number of contexts in Cluster 3 Zone 1 = 6
Number of contexts in Cluster 3 Zone 2 = 10
Number of contexts in Cluster 3 Zone 3 = 7
Number of contexts in Cluster 3 = 39

Cast return status: success

The first section containing the first four lines shows the utilizations of four different types of elements. The most-frequently-used element is the TALUs (92%), which performs the logical operations. 25% of the MULTs are used for delay compensation. 18% of the memory units implement the delay functions and the shift registers. The chip utilization of the ZigBee receiver is relatively low since all of the SALU elements remain unused.

6.3.2 Performance on Hardware

During the hardware testing of the ZigBee receiver, some issues of the hardware driver were discovered. The two main issues are unreliability of the data transmission from host computer to ECA chip, and the limitation of input data size. The first issue caused about 1dB degradation of the system performance, and the second matter complicated hardware BER testing. The input size is limited to 8192 16-bit samples per test, and the number of output bits is restricted to 512 bits. The input data have to be reloaded and the output data have to be processed manually for each test. Therefore, only four E_b/N_0 values are tested and there are around 10,000 test bits for each E_b/N_0 value. Figure 6-15 shows the performance of the ZigBee receiver on hardware. The performance curve is compared to the simulation curve in this figure. The 1dB degradation of the actual hardware BER performance is because of the errors occurred during the data transmission from host computer to the ECA chip as mentioned in the first driver issue.

These simulation and implementation results implied that this design and implementation of the ZigBee receiver is correct. In the next chapter, conclusions of design experience are made. The future mission on this ECA is specified.

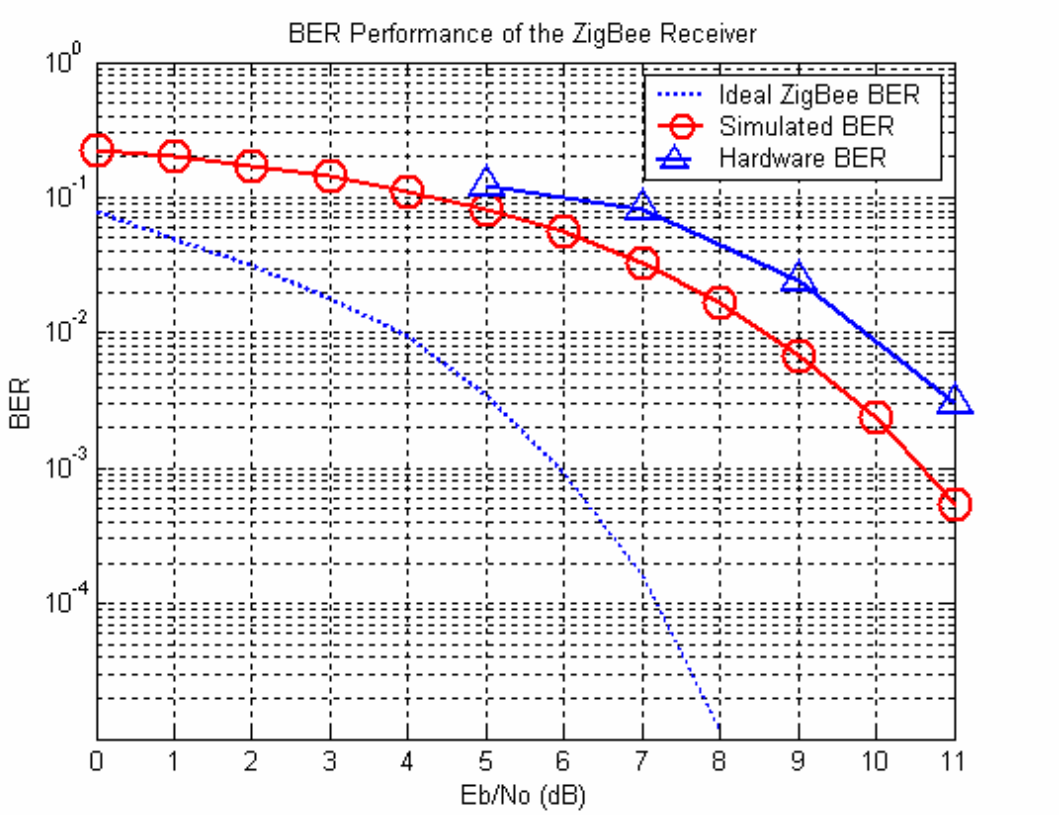

Figure 6-15. *BER Performance of the ZigBee Receiver on Hardware*

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The ZigBee is a simply and robust radio system. For some ASIC-based ZigBee products, the power consumption is as low as 60 miliwatts and the cost is as little as $20 [23]. A complete ZigBee receiver can even fit into half of an ECA-64 chip without significantly losing performance. That is the reason why ZigBee draws more and more attentions.

A comparison chart of all four hardware platforms is drawn in Figure 7-1. All comparison results drawn here only apply to the SDR applications and may not be true for other applications.

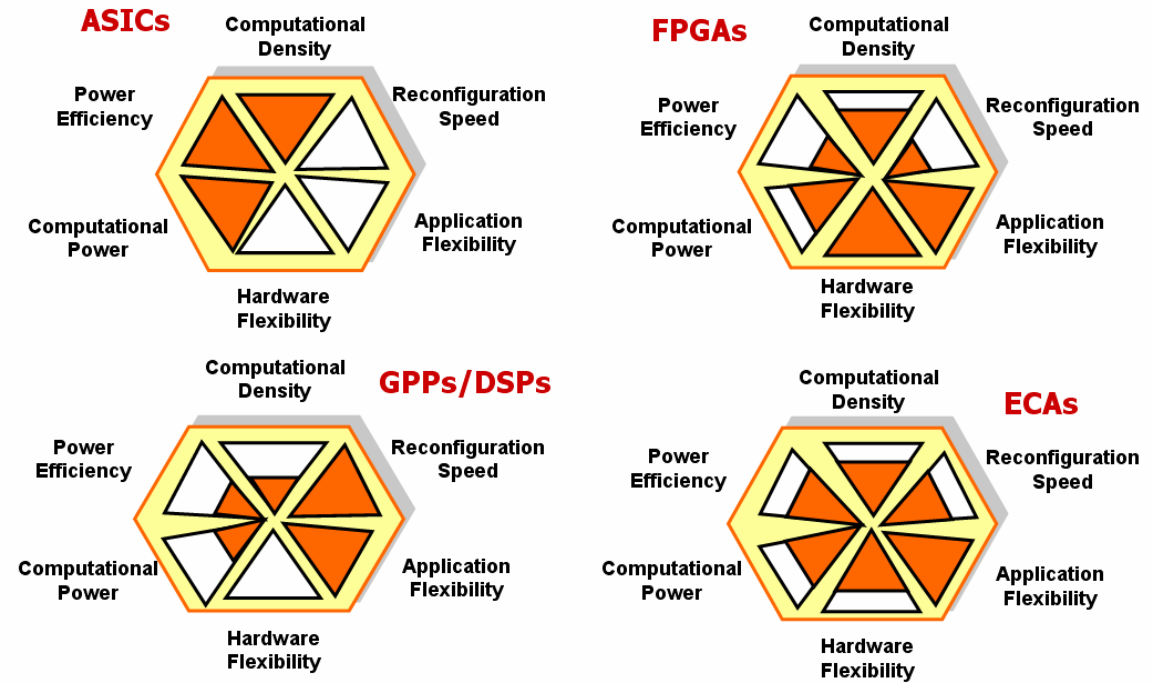

Figure 7-1. *Comparison of Four Platforms for SDR* [7]

The main difference in the hardware architecture between FPGAs and ECAs is that FPGAs are fine-grained and ECAs are coarse-grained. This difference provides FPGAs extreme flexibility for all applications while it provides more advantages such as faster reconfiguration speed and higher power efficiency. For an instance, ECAs only require one 16-bit configuration word to configure an element for forming an adder while FPGAs need much more than one 16-bit configuration word to configure about 10 configurable logic blocks and necessary inter-connections for an adder [12]. That is the reason why ECAs can reconfigure faster than FPGAs.

Although complex inter-connections provide the flexibility to FPGAs, they also cause some problems. One major problem is that inter-connections occupy a relatively large portion of FPGAs' chip area, which leads to relatively low computational density. Another problem is that inter-connections consume relatively high power and lead to relatively low power efficiency.

Comparing to GPPs/DSPs, ECAs have significant advantages for SDR applications. The only problem now is that only hundreds of engineers are doing ECA designs while millions of engineers are working on GPPs/DSPs. ASICs and ECAs have few overlap fields in SDR applications. They both have their own advantages and cannot simply replace each other.

Some lessons were also learned. As discussed in Chapter 6, if two or more inputs go through different data paths and subsequently merge together, it is better if all of the data paths have the same delay so that the system can work quickly and smoothly in a pipelined mode. Otherwise, the last-to-arrived input data may hold up all the other data at the merging end and slow down the entire system. Since the ECA is a data-flow machine, the design method is totally different from contemporary FPGA and DSP design-flows. For those who are used to working on FPGAs but interested in the ECA, a suggestion is to not rely on the experience from FPGAs too much while working on this platform, since some experience does not apply to this chip and may cause problems. Using knowledge from some classes like digital design can be a better way to proceed.

7.2 Future Work

So far, all work is done in digital domain, the next phase is to implement the RF front-end of the ZigBee receiver. The RF front-end can be based on a MAX2830 Zero-IF transceiver from Maxim which supports ZigBee and IEEE 802.11 b/g. A data acquisition card with a PCI interface is needed for interfacing the MAX2830 and the ECA platform.

When the ZigBee receiver system is working properly, implementing an IEEE 802.11 b/g (Wi-Fi) system on the ECA platform will be the next target. Although a Wi-Fi system is more complex than a ZigBee system, it is able to fit into the ECA chip with a proper design.

The final phase is to implement a smart radio which is able to distinguish incoming signals and load a corresponding receiver to handle the signal.

Bibliography

[1] W. H. W. Tuttlebe, "Software –defined radio: facets of a developing technology," *IEEE Personal Communications*, vol 6, pp. 38-44, April 1999.

[2] S. Srikanteswara, R. C. Palat, J. H. Reed and P. Athanas, "An overview of configurable computing machines for software radio handsets," *IEEE Communications Magazine*, pp 134-141, July 2003.

[3] J. H. Reed, *Software Radio A Modern Approach to Radio Engineering*, Prentice Hall, Upper Saddle River, NJ, 2002.

[4] Element CXI Inc., San Jose, CA, http://www.elementcxi.com.

[5] Element CXI Inc., San Jose, CA, *ECA-64 Advance Information*, Datasheet, July 2007.

[6] A. D. Stefano, G. Fiscelli, and C. G. Giaconia, "An FPGA-Based Software Defined Radio Platform for the 2.4GHz ISM Band," *Research in Microelectronics and Electronics*, pp.73-76, Jun 12-15, 2006.

[7] J. Reed, *Software Radio - Lecture 7: Hardware Issues*, class notes, Bradley Dept. of Electrical and Computer Engineering, Virginia Tech, November 11, 2007.

[8] Intel Inc, San Jose, CA, Introducing the 45nm Next-Generation Intel® Core Microarchitecture, White Paper, 2007.

[9] Digi-Key.com, http://search.digikey.com/scripts/DkSearch/dksus.dll?lang=en&site=US&keywords=XC5VLX330&x=18&y=24, Product Price List, January 2008.

[10] A. Doumar, H. Ito, "Design of switching blocks tolerating defects/faults in FPGA interconnection resources," *Defect and Fault Tolerance in VLSI Systems, 2000. Proceedings. IEEE International Symposium on*, pp. 134 - 142, 25-27 October, 2000.

[11] Element CXI Inc., San Jose, CA, *ECA-64 Device Architecture Overview*, Datasheet, July 2007.

[12] Wikimedia Foundation Inc, http://en.wikipedia.org/wiki/Zigbee

[13] Wikimedia Foundation Inc, http://en.wikipedia.org/wiki/IEEE_802.15.4.

[14] IEEE STD 802.15.4, *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, Institute of Electrical and Electronic Engineers, October, 2003.

[15] Digi International, Minnetonka, MN, http://www.digi.com/products/wireless/point-multipoint/xbee-series1-module.jsp.

[16] Micorchip, Chandler, AZ, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE &nodeId=1406&dDocName=en021925.

[17] W. A. Giovino, "Microcontrollers and DSPs - Will the Two Worlds Intersect?," http://www.microco ntroller.com /Embedded.asp?did=61.

[18] T. Meng, C. Zhang, P. Athanas, "AN FPGA-BASED ZIGBEE RECEIVER ON THE HARRIS SOFTWARE DEFINED RADIO SIP," *2007 Software Defined Radio Technical Conference*, November, 2007.

[19] R. Bittner, P. Athanas, "Wormhole Run-time Reconfiguration," *ACM fifth international symposium on Field-programmable gate arrays*, pp. 79-85, February 09-11, 1997.

[20] C. Maxfield, "Dynamically-reconfigurable Elemental Computing Arrays (ECAs) - Part 2 (Programming Model)," http://www.pldesignline.com/showArticle.jhtml?articleID=204701072.

[21] CoWare Inc., San Jose, CA, http://www.coware.com/products/signalprocessing.php.

[22] Element CXI Inc., San Jose, CA, *Crystal IDE*, Design Document, October 2007.

[23] M. Billoo, "How To Get Better Estimates and More Control over ZigBee Power Consumption," http://www.ddj.com/mobile/193104185, October 25, 2006.

[24] S. Xing, W.W.H. Yu, "FPGA Adders: Performance Evaluation and Optimal Design," *Design & Test of Computers, IEEE*, Vol 15, pp. 24-29, Jan.-March, 1998.

VITA

Chen Zhang was born in Shanghai, China. He received a B.E. degree of Telecommunications in Electrical Engineering from Xidian University in July of 2002. In August of 2006, he joined the Bradley Department of Electrical and Computer Engineering of Virginia Tech as a M.Sc. candidate. He started working in the Configurable Computing Laboratory, under the guidance of Dr. Athanas in January 2007, where he began researching configurable computing machine based software defined radios. His research interests include developing systems architectures needed to deploy software-defined radios on configurable computing machines. He will join Element CXI Inc., San Jose, CA to continue his research after graduation.