Development and Implementation of Physical Layer Kernels for Wireless Communication Protocols

by

Vamsi Reddy Chagari

A Thesis Presented in Partial Fulfillment
Of the Requirements for the Degree
Master of Science

Approved July 2016 by the
Graduate Supervisory Committee:

Chaitali Chakrabarti, Chair
Hyunseok Lee
Umit Ogras

ARIZONA STATE UNIVERSITY

August 2016

ABSTRACT

Historically, wireless communication devices have been developed to process one specific waveform. In contrast, a modern cellular phone supports multiple waveforms corresponding to LTE, WCDMA(3G) and 2G standards. The selection of the network is controlled by software running on a general purpose processor, not by the user. Now, instead of selecting from a set of complete radios as in software controlled radio, what if the software could select the building blocks based on the user needs. This is the new software-defined flexible radio which would enable users to construct wireless systems that fit their needs, rather than forcing to use from a small set of pre-existing protocols.

To develop and implement flexible protocols, a flexible hardware very similar to a Software Defined Radio (SDR) is required. In this thesis, the Intel T2200 board is chosen as the SDR platform. It is a heterogeneous platform with ARM, CEVA DSP and several accelerators. A wide range of protocols is mapped onto this platform and their performance evaluated. These include two OFDM based protocols (WiFi-Lite-A, WiFi-Lite-B), one DFT-spread OFDM based protocol (SCFDM-Lite) and one single carrier based protocol (SC-Lite). The transmitter and receiver blocks of the different protocols are first mapped on ARM in the T2200 board. The timing results show that IFFT, FFT, and Viterbi decoder blocks take most of the transmitter and receiver execution time and so in the next step these are mapped onto CEVA DSP. Mapping onto CEVA DSP resulted in significant execution time savings. The savings for WiFi-Lite-A were 60%, for WiFi-Lite-B were 64%, and for SCFDM-Lite were 71.5%. No savings are reported for SC-Lite since it was not mapped onto CEVA DSP.

Significant reduction in execution time is achieved for WiFi-Lite-A and WiFi-Lite-B protocols by implementing the entire transmitter and receiver chains on CEVA DSP. For instance, for WiFi-Lite-A, the savings were as large as 90%. Such huge savings are because the entire transmitter or receiver chain are implemented on CEVA and the timing overhead due to ARM-CEVA communication is completely eliminated. Finally, over-the-air testing was done for WiFi-Lite-A and WiFi-Lite-B protocols. Data was sent over the air using one Intel T2200 WBS board and received using another Intel T2200 WBS board. The received frames were decoded with no errors, thereby validating the over-the-air-communications.

DEDICATION

To my family for their support

ACKNOWLEDGMENTS

I would first like to express my sincere gratitude to my advisor and committee chair, Dr. Chaitali Chakrabarti for giving me an opportunity to work on this exciting project and for overseeing my research. I am grateful to my committee members, Dr. Hyunseok Lee and Dr. Umit Ogras for their help, guidance, insightful comments and feedbacks. I would also like to thank Dr. Dan Bliss for providing valuable ideas and support. I thank my fellow project mate Ganapati Bhat for all his help in this project. I would also like to thank the other team members, Sharanya Srinivas, Eric Torkildson, Shunyao Vincent Wu, Tom Mcgiffen, and Prashant Sharma. Finally, I would like to express my profound gratitude to my parents for providing me with unfailing support and continuous encouragement through the process of researching and writing this thesis.

This thesis work has been supported by Google through the Center for Wireless Information Systems and Computational Architectures (WISCA).

TABLE OF CONTENTS

Page

LIST OF TABLES ix

LIST OF FIGURES xi

CHAPTER

1 INTRODUCTION 1

1.1 Software Defined Radio 1

1.2 Flexible Protocols 3

1.2.1 Proposed Method 3

1.2.2 Protocol Recommendation Engine (PRE) 4

1.2.3 Hardware Recommendation Engine (HRE) 4

1.3 Thesis Contributions 5

1.3.1 WiFi-Lite-A Protocol 6

1.3.2 WiFi-Lite-B Protocol 7

1.3.3 SCFDM-Lite Protocol 8

1.3.4 SC-Lite Protocol 9

1.4 Organization 9

2 BACKGROUND 10

2.1 Intel Transcede T22xx SOC Architecture 10

2.2 CEVA DSP XC323 Architecture 10

2.3 Wireless Protocols 12

2.3.1 WiFi-Lite 12

2.3.2 SCFDM-Lite 18

2.3.3 SC-Lite 20

3 WiFi-Lite-A Protocol 22

3.1 Protocol Overview 22

3.1.1 Transmitter 22

3.1.2 Receiver 23

3.2 WiFi-Lite-A Implementation on ARM 24

3.2.1 Experimental Results 25

3.3 WiFi-Lite-A Implementation on ARM and CEVA DSP 27

3.3.1 ARM-CEVA Interface 27

3.3.2 Experimental Results 28

3.4 WiFi-Lite-A Fixed-Point Implementation on CEVA DSP 31

3.4.1 Experimental Results – CEVA DSP SDK 32

3.4.2 Experimental Results – CEVA DSP Hardware 33

3.5 WiFi-Lite-A Implementation using Ettus Radio 37

3.5.1 Experimental Results 38

CHAPTER Page

3.6 Implementation Using Intel T2200 for TX & RX – Real Time Processing 40

3.6.1 Experimental Results 40

4 WiFi-Lite-B Protocol 43

4.1 Protocol Overview 43

4.1.1 Transmitter 43

4.1.2 Receiver 44

4.2 WiFi-Lite-B Implementation on ARM 45

4.2.1 Experimental Results 46

4.3 WiFi-Lite-B Implementation on ARM and CEVA DSP 48

4.3.1 Experimental Results 48

4.4 WiFi-Lite-B Fixed-Point Implementation on CEVA SDK 51

4.4.1 Experimental Results 51

4.5 Implementation Using Intel T2200 for TX and RX – Real Time Processing.... 55

4.5.1 Experimental Results 55

5 SCFDM-Lite 57

5.1 Protocol Overview 57

5.1.1 Transmitter 57

5.1.2 Receiver 58

CHAPTER Page

5.2 SCFDM-Lite Implementation on ARM ... 59

5.2.1 Experimental Results ... 60

5.3 SCFDM-Lite Implementation on the ARM and CEVA DSP ... 62

5.3.1 Experimental Results ... 62

6 SC-Lite ... 66

6.1 Protocol Overview ... 66

6.1.1 Transmitter ... 66

6.1.2 Receiver ... 66

6.2 SC-Lite Implementation on ARM ... 67

6.2.1 Experimental Results ... 68

7 CONCLUSIONS ... 69

7.1 Summary ... 69

7.2 Future Work ... 69

LIST OF TABLES

Table Page

1. WiFi-Lite-A Protocol Configuration 23

2. WiFi-Lite-A Transmitter Implementation on ARM: Timing Profile 25

3. WiFi-Lite-A Receiver Implementation on ARM: Timing Profile............................... 26

4. Scaling Factor for CEVA FFT Library Functions 28

5. WiFi-Lite-A Transmitter Implementation on ARM+CEVA: Timing Profile 29

6. WiFi-Lite-A Receiver Implementation on ARM+CEVA: Timing Profile.................. 30

7. WiFi-Lite-A Transmitter Implementation on CEVA SDK: Timing Profile................ 32

8. WiFi-Lite-A Receiver Implementation on CEVA SDK: Timing Profile 32

9 WiFi-Lite-A Transmitter Implementation on CEVA DSP: Timing Profile 33

10 WiFi-Lite-A Receiver Implementation on CEVA DSP: Timing Profile.................... 34

11. WiFi-Lite-A Transmitter Profiler Results - ARM+CEVA vs CEVA DSP 35

12. WiFi-Lite-A Receiver Profiler Results - ARM+CEVA vs CEVA DSP.................... 36

13. WiFi-Lite-B Protocol Configuration... 44

14. WiFi-Lite-B Transmitter Implementation on ARM: Timing Profile......................... 46

15. WiFi-Lite-B Receiver Implementation on ARM: Timing Profile 47

16. WiFi-Lite-B Transmitter Implementation on ARM+CEVA: Timing Profile 48

17. WiFi-Lite B Receiver Implementation on ARM+CEVA: Timing Profile 49

18. WiFi-Lite-B Transmitter Implementation on CEVA SDK: Timing Profile.............. 52

19. WiFi-Lite-B Receiver Implementation on CEVA SDK 52

20. WiFi-Lite-B Transmitter Profiler Results - ARM+CEVA vs CEVA SDK............... 53

21. WiFi-Lite-B Receiver Profiler Results - ARM+CEVA vs CEVA SDK 54

Table Page

22 SCFDM-Lite Protocol Configuration 58

23. SCFDM-Lite Transmitter Implementation on ARM: Timing Profile 60

24. SCFDM-Lite Receiver Implementation on ARM: Timing Profile............................. 61

25. SCFDM-Lite Transmitter Implementation on ARM+CEVA: Timing Profile 62

26. SCFDM-Lite Receiver Implementation on ARM+CEVA: Timing Profile................ 64

27. SC-Lite Transmitter Profiler Results 68

28. SC-Lite Receiver Profiler Results.. 68

LIST OF FIGURES

Figure Page

1. Overview of the Protocol Development Kit 4

2. CEVA XC323 Hardware Architecture 11

3. WiFi-Lite Protocol: Block Diagram of Transmitter 13

4. WiFi-Lite Protocol: Block Diagram of Receiver... 16

5. SCFDM-Lite Protocol: Block Diagram of Transmitter... 18

6. SCFDM-Lite Protocol: Block Diagram of Receiver 19

7. SC-Lite Protocol: Block Diagram of Transmitter.. 20

8. SC-Lite Protocol: Block Diagram of Receiver 21

9. WiFi-Lite-A Protocol: Block Diagram of Transmitter 23

10. WiFi-Lite-A Protocol: Block Diagram of Receiver... 24

11. WiFi-Lite-A Implementation – TX and RX Loopback 25

12. WiFi-Lite-A Profiling Results for ARM Implementation 26

13. WiFi-Lite-A TX Execution Time Values - ARM vs ARM+CEVA.......................... 29

14. WiFi-Lite-A RX Execution Time Values - ARM vs ARM+CEVA.......................... 30

15. WiFi-Lite-A CEVA SDK Implementation 31

16. WiFi-Lite-A: Over the Air Experiment Using Ettus Radio and Intel WBS 37

17. WiFi-Lite-A GNURadio Transceiver Block Diagram... 38

18. WiFi-Lite-A GNU Radio Implementation - Frame Detection Results....................... 39

19. WiFi-Lite-A – GNU Radio QPSK Constellation Diagram.. 39

20. Intel T2200 WBS Dual Frequency Setup 40

21. WiFi-Lite-A Over the Air Implementation - Intel T2200 WBS Boards..................... 41

Figure Page

22. WiFi-Lite-A Over-the-Air Frame Detection Algorithm Result 42

23. WiFi-Lite-A Carrier Frequency Offset Algorithm Result 42

24. WiFi-Lite-B Protocol: Block Diagram of Transmitter 44

25. WiFi-Lite-B Protocol: Block Diagram of Receiver 45

26. WiFi-Lite-B Protocol Implementation - TX and RX Loopback 46

27. WiFi-Lite-B Profiling Results for ARM Implementation 47

28. WiFi-Lite-B TX Execution Time Values - ARM vs ARM+CEVA 49

29. WiFi-Lite-B RX Receiver Time Values - ARM vs ARM+CEVA 50

30. WiFi-Lite-B CEVA SDK Implementation 51

31. WiFi-Lite-B Over the Air Implementation - Intel T2200 WBS Boards 55

32. WiFi-Lite-B – 16 QAM Constellation Diagram at the Receiver 56

33. SCFDM-Lite Protocol: Block Diagram of Transmitter 58

34. SCFDM-Lite Protocol: Block Diagram of Receiver 59

35. SCFDM-Lite Protocol Implementation - TX and RX Loopback 60

36. SCFDM-Lite Profiling Results for ARM Implementation 61

37. SCFDM-Lite TX Execution Time Values - ARM vs ARM+CEVA 63

38. SCFDM-Lite RX Execution Time Values - ARM vs ARM+CEVA 64

39. SC-Lite Protocol: Block Diagram of Transmitter 66

40. SC-Lite Protocol: Block Diagram of Receiver 67

41. SC-Lite Protocol Implementation – Experimental Setup 67

1 INTRODUCTION

1.1 Software Defined Radio

Historically, wireless communication devices have been developed to process a specific waveform. Most of the older radios were single function systems. For instance, a first generation cellular phone sent only voice using GSM standard. In contrast, a modern cellular phone supports LTE, WCDMA(3G) and 2G standards. The user is not required to flip a switch to access each network; the selection is controlled by software running on the phone (Grayver, 2012).

Now, instead of selecting from a set of complete radios as in software controlled radio, what if the software could select the building blocks. For example, the software could select a particular modulation block wherein the software has to configure details of the modulator, such as choose between QPSK or QAM modulator, to map bits to symbols. This is the new software-defined radio (SDR) which can be reprogrammed for functionally as desired. In contrast, a software-controlled radio is limited to functionality explicitly included by the designers (Lee H. L., 2005).

The main characteristic of a SDR is its ability to provide great flexibility in software to support different waveforms with low power consumption. The definition from wireless innovation forum (formerly SDR forum) states: A software-defined radio is a radio in which some or all of the physical layer functions are software defined (SDR forum). In a broad sense, software defined means that different waveforms can be supported by modifying the software or firmware but not changing the hardware.

SDR standardization has been progressing for many years. The Software Communications Architecture (SCA) standard developed by the US Army and Space Telecommunication Radio System (STRS) standard developed by NASA define robust and powerful infrastructures for flexible radios. There are currently many software reconfigurable radio communications equipment's being deployed in the field by the military and NASA organizations (Grayver, 2012). There are also a few commercial mobile multi-standard terminals (VANU base stations) which support many waveform standards like GSM, EDGE, W-CDMA, HSDPA, WiFi. There are two popular open-source SDR platforms, namely GNURadio and OSSIE, which are being used in academia and industry to develop and implement a wide range of protocols (Ramacher, 2007).

SDR architectures come in all flavors – from custom hardware to reconfigurable array to DSP assisted architectures to hybrid SIMD architectures (Lin, et al., 2006). While custom hardware can reach the requirements (throughput and power) of a single protocol, it is not the solution when multiple protocols have to be supported. A very important aspect of a multiple radio system is programmability. However, programmability comes with increase in power consumption. As software flexibility increases, the power consumption of a chip increases. To meet the high computational requirements of SDR with low power budgets heterogeneous architectures have been proposed, these architectures have good programmable flexibility with moderate power consumption and is the SDR architecture of choice.

There are several commercial solutions that provide moderate programmable flexibility with low power consumption. Some vendors like Mercury, Morphics, Quicksilver and Televersal designed hardware solutions that provided moderate flexibility in mapping the PHY layer algorithms onto the hardware (Ramacher, 2007). Other vendors designed hardware solutions based on DSP-centered and accelerator-assisted architectures to provide the highest degree of flexibility in mapping PHY layer algorithms onto the hardware. Examples include Intel T2200 Wireless Base Station (Berkeley Design Technology, Inc., 2012) which has two ARM cores, two DSP cores, four MAP coprocessors and other accelerators (FEC, Turbo/Viterbi decoder), Sandbridge SB3011

(Glossner, et al., 2007) which has Quad-DSP complex cores (8-way multithreaded), one ARM processor and, Infineon (Ramacher, 2007) which has one ARM processor, four SIMD DSP cores, and other accelerators (FIR Filter and Turbo/Viterbi decoder). These heterogeneous architecture solutions helped designers to derive SDR solutions with low power for a wide range of protocols.

1.2 Flexible Protocols

The next revolution of the radio technologies "5G and internet of things" has started. While 5G focuses on higher data rates, IoT targets many aspects, such as greater access to network, network reliability, and communication diversity. As user needs keep evolving, the need for different kind of communication protocols increases. Instead of developing a rigid protocol for the current needs/requirements, which may change in future, our idea is to develop a flexible protocol, which enables users to construct wireless systems that fit their needs, rather than forcing to use from a small set of pre-existing protocols.

1.2.1 Proposed Method

To aid the development and implementation of new or modified protocols, a protocol development kit (PDK) is proposed as shown in Figure 1. PDK is designed to help reduce the cost of defining, developing, testing and implementing the protocols. PDK consists of a protocol recommendation engine which recommends a protocol based on the user inputs, a hardware recommendation engine which interacts with the protocol recommendation engine to converge to a protocol that can be implemented on the target hardware and finally implementation of the protocol on a hardware platform. In this study, the hardware platform is the Intel T2200 board which consists of two ARM cores, two CEVA DSP cores, a few accelerators (Viterbi decoder/FEC) and a few Multi-Purpose Advanced Processors (Berkeley Design Technology, Inc., 2012).

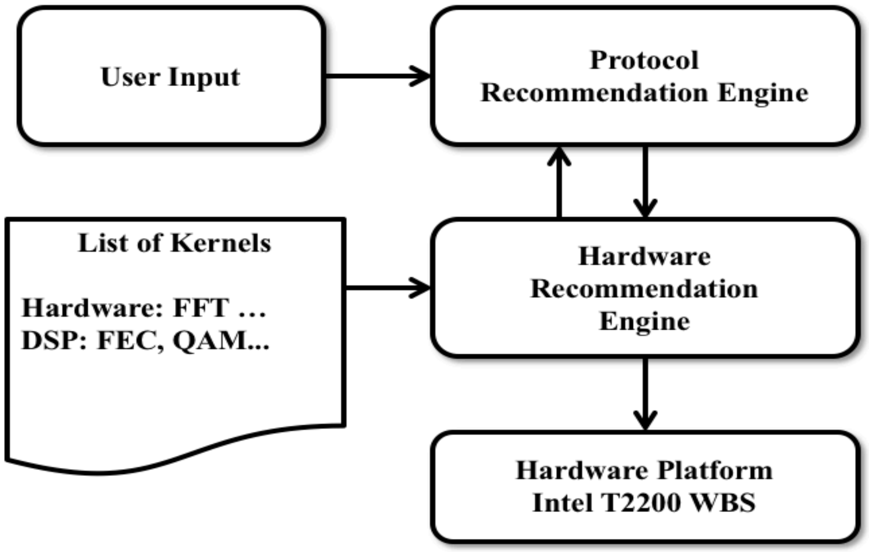
User Input
Protocol Recommendation Engine
List of Kernels
Hardware: FFT …
DSP: FEC, QAM...
Hardware Recommendation Engine
Hardware Platform Intel T2200 WBS

Figure 1. Overview of the Protocol Development Kit

1.2.2 Protocol Recommendation Engine (PRE)

The PRE recommends a protocol using the user inputs and system requirements. The user parameters include operational environment, user density, link length and system requirements which include bandwidth constraints, network topology, available frequency bands, data rate requirements, antenna specifications, maximum transmit power, latency requirement, etc. The PRE recommends the set of most suitable protocols based on the user and systems inputs. These protocols could be similar to existing protocols such as WiFi, LTE, SCFDM, SC or could be brand new. The details of the recommended protocols are given to the hardware recommendation engine.

1.2.3 Hardware Recommendation Engine (HRE)

After getting the protocol details from PRE, HRE checks to see whether it can support the implementation of the specific protocol. If not, HRE sends feedback to PRE asking to request the user to modify the inputs. HRE selects the best possible protocol from the list of the protocols that

PRE has recommended and allocates resources to process the different kernels. Each kernel in the protocol can run on a general purpose processor - ARM, or on a vector processing unit (VPU) - CEVA DSP, or on a dedicated hardware accelerator - FEC. The HRE does the resource assignment based on timing or memory constraints.

1.3 Thesis Contributions

In this study a wide variety of wireless protocols are selected for implementing on the Intel platform. These include WiFi, an OFDM based WLAN system, mostly used for indoor communications, SCFDM which is an OFDM-based system used in cellular communications such as LTE-uplink, and a Single Carrier-based system that is used for low power and low data rate communications. The performance of the protocols is evaluated on a heterogeneous computing platform, namely, Intel Transcede T2200 Wireless Base Station board. This platform consists of two ARM cores (Cortex A9), two DSP cores (XC-323), a few accelerators (Viterbi decoder/FEC) and a few Multi-Purpose Advanced Processors (MAP). A summary of the tasks that were undertaken is as follows.

i. Protocol implementation on only ARM, where all the transmitter and receiver blocks are mapped to ARM.
ii. Protocol implementation on ARM+CEVA platforms, where FFT, IFFT, and Viterbi decoder are mapped onto CEVA DSP and the remaining blocks are mapped onto ARM.
iii. Protocol implementation on only CEVA, where all the transmitter blocks and receiver blocks are mapped onto CEVA DSP to overcome the communication overhead between ARM and CEVA.
iv. Over-the-air testing using Intel Transcede boards or GNURadio and Ettus Radios.

1.3.1 WiFi-Lite-A Protocol

WiFi-Lite-A is an Orthogonal Frequency Division Multiplexing (OFDM) based WLAN system that is used for indoor communications. WiFi-Lite-A protocol uses 64 subcarriers that are modulated by quadrature phase shift keying (QPSK) followed by convolutional coding with a coding rate of ½ (IEEE Standards association, 2012).

First, WiFi-Lite-A protocol transmitter and receiver PHY layer kernels were developed and implemented on ARM. In the transmitter chain, IFFT took 1.6ms, which is 95% of the total 1.7ms execution time. Similarly, for the receiver chain, FFT took 29%, channel estimation and equalization took 31%, and Viterbi decoder took 28% of the total execution time. These three blocks together accounted for 88% of the total execution time of 7.8ms. This prompted us to implement the IFFT, FFT, Viterbi decoder blocks onto the CEVA DSP; the rest of the blocks in transmitter and receiver chains were still mapped on ARM. Such a mapping resulted in reduction in the execution time of IFFT block from 1.6ms to 0.6ms, and the reduction in the transmitter chain execution time reduction from 1.7ms to 0.69ms. Similarly, in the receiver chain, FFT block time reduced from 2.2ms to 0.6ms, channel estimation block reduced from 2.4ms to 0.8ms and decoder block reduced from 2.1ms to 0.5ms. Overall, the receiver chain execution time reduced by 62% from 7.8ms to 2.9ms.

Next, WiFi-Lite-A protocol PHY layer kernels were implemented on CEVA DSP SDK and on CEVA DSP hardware. On SDK this resulted in reduction of the transmitter execution time (without preamble block) from 638μs to 7.6μs and reduction of the total receiver execution time from 2.9ms to 76.84μs. The spectacular reduction in the execution time was achieved because the CEVA DSP SDK does not exactly emulate the hardware. On CEVA DSP hardware, this resulted in reduction of the transmitter execution time from 638μs to 86.8μs and reduction of the total receiver execution time from 2.9ms to 0.11ms. The significant reduction in the execution time was achieved because of reduced context switching and memory overhead due to running the complete transmitter and receiver chains on the CEVA DSP hardware.

A GNURadio WiFi-Lite-A protocol transmitter model was developed to do over-the-air communication using Ettus radios (N210). The Intel T2200 WBS board was used to decode the received frames. Finally, over-the-air transmission was demonstrated using Intel T2200 WBS boards for both transmitter and receiver.

1.3.2 WiFi-Lite-B Protocol

WiFi-Lite-B is also an Orthogonal Frequency Division Multiplexing (OFDM) based WLAN system. It also uses 64 subcarriers as in WiFi-Lite-A. However, here quadrature amplitude modulation (QAM) is used followed by convolutional coding with a coding rate of 2/3 (IEEE Standards Association, 2012).

First, WiFi-Lite-B protocol transmitter and receiver PHY layer kernels were developed and implemented on ARM. In the transmitter chain, IFFT took 1.6ms, which is 94% of the total 1.7ms execution time. Similarly, for the receiver chain, FFT took 19%, channel estimation and equalization took 21%, and Viterbi decoder took 50% of the total 11.6ms execution time. These three blocks together accounted for 90% of the execution time. Next, IFFT, FFT, Viterbi decoder blocks were implemented on the CEVA DSP; the rest of the blocks in transmitter and receiver chains were still mapped on ARM. Such a mapping resulted in reduction in the execution time of IFFT block from 1.6ms to 0.6ms, and the reduction in the transmitter chain total execution time reduction from 1.7ms to 0.72ms. Similarly, in the receiver chain, FFT block time reduced from 2.2ms to 0.65ms, channel estimation block reduced from 2.4ms to 0.8ms and decoder block reduced from 5.7ms to 0.6ms. Overall, the receiver chain execution time reduced by 72% from 11.6ms to 3.2ms.

Next, WiFi-Lite-B protocol PHY layer kernels were implemented on CEVA DSP SDK. This resulted in reduction of the transmitter execution time (without preamble block) from 638µs to 10.5µs and reduction of the total receiver execution time from 3.2ms to 82.6µs. The significant reduction in the execution time is achieved by reducing context switching and memory overhead due to running the complete transmitter and receiver chains on the CEVA DSP. Finally, over-the-air transmission was demonstrated using Intel T2200 WBS boards for the both transmitter and receiver.

1.3.3 SCFDM-Lite Protocol

SCFDM-Lite is an Orthogonal Frequency Division Multiplexing (OFDM) based system with a DFT mapper, which utilizes single carrier modulation (SC), DFT-spread orthogonal frequency multiplexing, and frequency domain equalization. It is mostly used in cellular communications such as LTE-uplink. SCFDM-Lite protocol uses 64 DFT subcarriers and 128 FFT subcarriers that are modulated by quadrature phase shift keying (QPSK) and forward error correction coding (convolutional coding) with a coding rate of ½ (3GPP a global initiative, 2004).

First, SCFDM-Lite protocol transmitter and receiver PHY layer kernels were developed and implemented on ARM. In the transmitter chain, IFFT took 4.6ms and FFT took 2.3ms of the total 7ms execution time. These two blocks accounted for 98% of the execution time. Similarly, for the receiver chain, FFT took 43%, IFFT took 19%, channel estimation and equalization took 19%, and Viterbi decoder took 15% of the total time. These four blocks together accounted for 96% of the total execution time of 11.2ms. This prompted us to implement the IFFT, FFT, Viterbi decoder blocks onto the CEVA DSP; the rest of the blocks in transmitter and receiver chains were still mapped on ARM. Such a mapping resulted in the execution time of the transmitter chain reducing from 7ms to 1.2ms. Similarly, in the receiver chain, the reduction in the FFT block, IFFT block, and decoder block execution time resulted in 60% reduction in total execution time, from 11.2ms to 4.3ms.

1.3.4 SC-Lite Protocol

SC-Lite is a single carrier based system used for low power and low data rate communications. SC-Lite protocol uses Reed Solomon encoder to encode the bits and uses BPSK modulation to map the bits to symbols.

SC-Lite protocol transmitter and receiver PHY layer kernels were developed and implemented on ARM. In the transmitter chain, Reed-Solomon encoder took 30µs, which is 81% of the total 37µs execution time. Similarly, for the receiver chain, Reed-Solomon decoder took 40µs, which is 75% of the total 53µs execution time.

1.4 Organization

The remainder of the thesis is organized as follows. In Chapter 2, WiFi-Lite, SCFDM-Lite and SC-Lite protocols transmitter and receiver blocks, and also the architecture details of CEVA DSP XC323 and Intel Transcede T22xx series SOC are described. In Chapter 3, implementation of WiFi-Lite-A protocol on ARM and CEVA DSP platforms are described. In Chapter 4, implementation of WiFi-Lite-B protocol on ARM and CEVA DSP platforms are described. In Chapter 5, implementation of SCFDM-Lite protocol on ARM and CEVA DSP platforms are described and in chapter 6, implementation of SC-Lite protocol on ARM platform are described. Chapter 7 concludes the thesis.

2 BACKGROUND

2.1 Intel Transcede T22xx SOC Architecture

Intel Transcede T22xx-series SOC contain three primary processing blocks, a dual-core ARM Cortex-A9 cluster, running at 1 GHz with a NEON fixed/floating point SIMD unit, two CEVA-XC323 cores, operating at 750MHz, each consisting of a vector DSP core and an SIMD vector unit with an 8-issue VLIW instruction set that delivers up to 8 GIPS per core, and four MAP4 (MindSpeed Application DSP) cores, each core containing four SIMD vector units (with each unit capable of 1 GMAC/sec performance for 24x16 array math). In addition, it also includes forward error correction function blocks, chip-rate processing correlator (used in WCDMA), and several security protocol accelerators. These blocks connect to each other, as well as to numerous special-function cores, via a combination of AXI (advanced extensible interface) and AHB (advanced high-performance bus) AMBA interconnect fabric (Berkeley Design Technology, Inc., 2012).

Intel Transcede T22xx provides several I/O bus options, dual JESD207 radio interfaces (CMOS and LVDS), PCI Express (single and quad) clusters, serial and reduced gigabit media independent interfaces (SGMII & RGMII), universal subscriber identity module (USIM), USB, UART, JTAG, I^{2}C, SPI and multiple GPIOs (Berkeley Design Technology, Inc., 2012).

2.2 CEVA DSP XC323 Architecture

CEVA XC323 is a fully programmable fixed point DSP processor architecture with a unique mix of VLIW and vector capabilities. It has two vector processing units (SIMD engine), each unit operates on 256-bit vector register. It supports up to 8 simultaneous instructions (8-Way VLIW) and also supports non-vectorized data, control, and ANSI-C operations. CEVA DSP has powerful computation capabilities which support 32 16x16-bit MAC operations, 64 arithmetic operations and over 200 16-bit operations per cycle. It also has many coprocessor units which allow efficient 1ow power implementation of transceiver algorithms, like MLD MIMO detector, Fast Hadamard Transform, DFT, FFT, Viterbi decoding, LLR processing and HARQ combining. It provides a high flexible SIMD programming model with intra-vector permutation capabilities and optimized modem instruction sets including high precision instruction set architecture (ISA) (CEVA DSP, 2012).

CEVA XC323 has a memory subsystem, which includes coupled memories(TCM), caches, AXI system interfaces, APB interface, DMA controller, message queues, emulation and profiling modules. It also has Power Scaling Unit (PSU), which can be used to achieve significant energy savings.

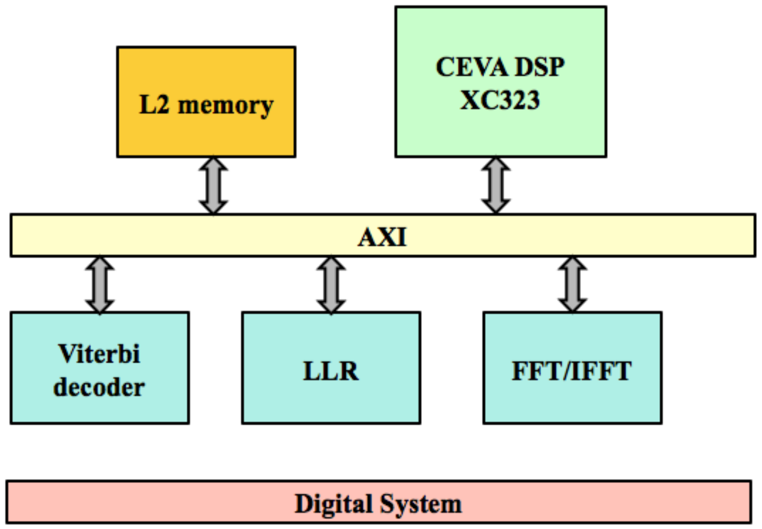

Figure 2. CEVA XC323 Hardware Architecture

CEVA's development environment or CEVA-Toolbox provides all the software and hardware tools to the programmer to develop any specific application. CEVA-Toolbox provides a Software Development Framework, which includes a complete set of development, debug and optimization tools, which can be operated and configured via the integrated development environment (IDE) using a GUI (CEVA DSP, 2012).

CEVA also provides an Integrated Development Environment (IDE) which includes a powerful compiler that facilitates software development without the need for a programmer to master architecture-specific details. The compiler supports the CEVA VEC-C language extensions for vector processors, enabling the entire architecture to be programmed in C-level language (CEVA DSP, 2012).

2.3 Wireless Protocols

The goal is to develop and implement a wide range wireless protocols on the Intel Transcede T2200 Wireless Base Station. For this study, WiFi, an OFDM based WLAN system, mostly used for indoor communications, SCFDM, an OFDM-based system used in cellular communications such as LTE-uplink, and a Single Carrier based system used for low power and low data rate communications, are selected.

2.3.1 WiFi-Lite

The transmitter and receiver blocks used in the WiFi-Lite protocol are described in Sections 2.3.1.1 and 2.3.1.2, respectively.

2.3.1.1 Transmitter

The key blocks in the transmitter chain of our WiFi-Lite protocol are presented in Figure 3, followed by short description of each of the blocks.

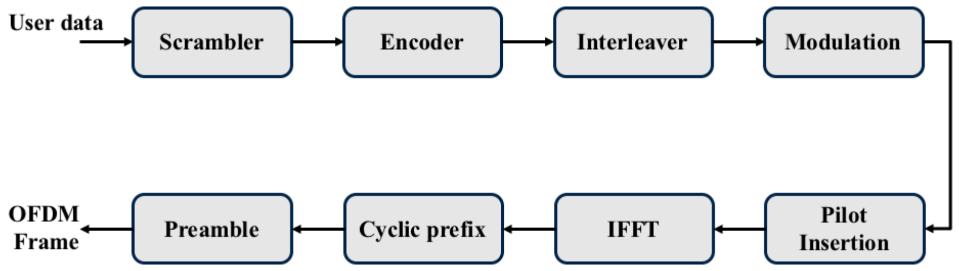

Figure 3. WiFi-Lite Protocol: Block Diagram of Transmitter

2.3.1.1.1 Scrambler

Scrambler is a unit that transposes or inverts signals or encodes a message at the transmitter to make it unintelligible at the receiver. It facilitates the work of timing recovery circuits, automatic gain control and other adaptive circuits of the receiver by reordering sequences such that there are no long sequences consisting of consecutive zeros and ones. It is also used for energy dispersal of the carrier and reducing intercarrier signal interference (Unnikrishnan & Sunil, 2011).

2.3.1.1.2 Encoder

The encoder is used to convert data from one format to another format for better and reliable transmission. It adds redundant bits to the payload which helps the receiver to decode the bits in case of severe channel conditions (Proakis & Masoud, 2007) .

2.3.1.1.2.1 Viterbi/Convolutional Encoder

A convoutional encoder, encodes the entire data stream, into a single codeword. It maps information to code bits sequentially by convolving a sequence of information bits with generator sequences. It can be implemented easily using linear feedback shift register. A convolutional code is specified by n, k, and K parameters, where n is the number of outputs, k is the number of inputs, K is a contraint length of the convolutional code, k/n is the coding rate which determines the number of data bits per coded bit (Proakis & Masoud, 2007). The sliding nature (convolutional of the encoder over the data) of the convolutional codes enables us to use a time-invariant trellis, which supports maximum likelihood soft decision decoding.

2.3.1.1.3 Interleaver

Interleaving is a technique commonly used in communication systems to overcome correlated channel noise such as burst errors. The interleaver rearranges input data such that consecutive data are spaced apart. At the receiver end, the interleaved data are arranged back into the original sequence by the de-interleaver. A linear random interleaver is used in this protocol, which only does data permutation.

2.3.1.1.4 Modulation

Modulation is the process of varying one or more properties of a periodic waveform, called the carrier signal, with a modulating signal that typically contains information to be transmitted (Proakis & Masoud, 2007).

2.3.1.1.4.1 Quadrature Phase-shift Keying

QPSK is a digital modulation scheme that conveys data by changing (modulating) the phase of a reference signal (the carrier wave). QPSK maps two bits into a symbol and can be visualized using the constellation diagram.

2.3.1.1.4.2 Quadrature Amplitude Modulation

QAM is a digital modulation scheme that conveys data by changing the amplitude of a reference signal (carrier waves) using amplitude-shift keying. QAM can map four bits into a symbol and can be visualized using the constellation diagram.

2.3.1.1.5 Pilot Insertion

Pilots are the known data symbols inserted at fixed locations in each OFDM symbol. They are used to estimate the channel behavior. In our implementation, sixteen pilots are used for each OFDM symbol. The positions of the pilot symbols in each OFDM symbol are fixed. Therefore, it is easy to extract the pilots at the receiver.

2.3.1.1.6 IFFT/FFT

Data bits, which are modulated as complex symbols, are mapped to the subcarriers which are orthogonal to each other using the IFFT. The Fourier transform converts a signal from its original domain (time or space) to its representation in the frequency domain and vice versa. The Fast Fourier Transform (FFT) rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. As a result, it reduces the complexity of computing the Discrete Fourier Transform (DFT) from O (n^2) to O (n log n), where n is the data size (Rader & Brenner, 1976).

2.3.1.1.7 Cyclic Prefix/Guard Interval

The cyclic prefix adds the tail end of each symbol to the front end. This is done to eliminate the intersymbol interference and to make the system robust to multipath fading. The length of the cyclic prefix is based on the IEEE standard.

2.3.1.1.8 Preamble

The preamble is a complex data inserted at the head of every OFDM frame to achieve proper synchronization at the receiver. In this protocol, the preamble is used for packet and frame detection. FFT modulated short and long preamble symbols (10 short symbols and 2 long symbols) are used as per the IEEE standards (802.11a) (IEEE Standards association, 2012).

2.3.1.2 Receiver

In this section, the blocks in the receiver chain of the WiFi-Lite protocol are described. Figure 4 provides the block diagram of the receiver system.

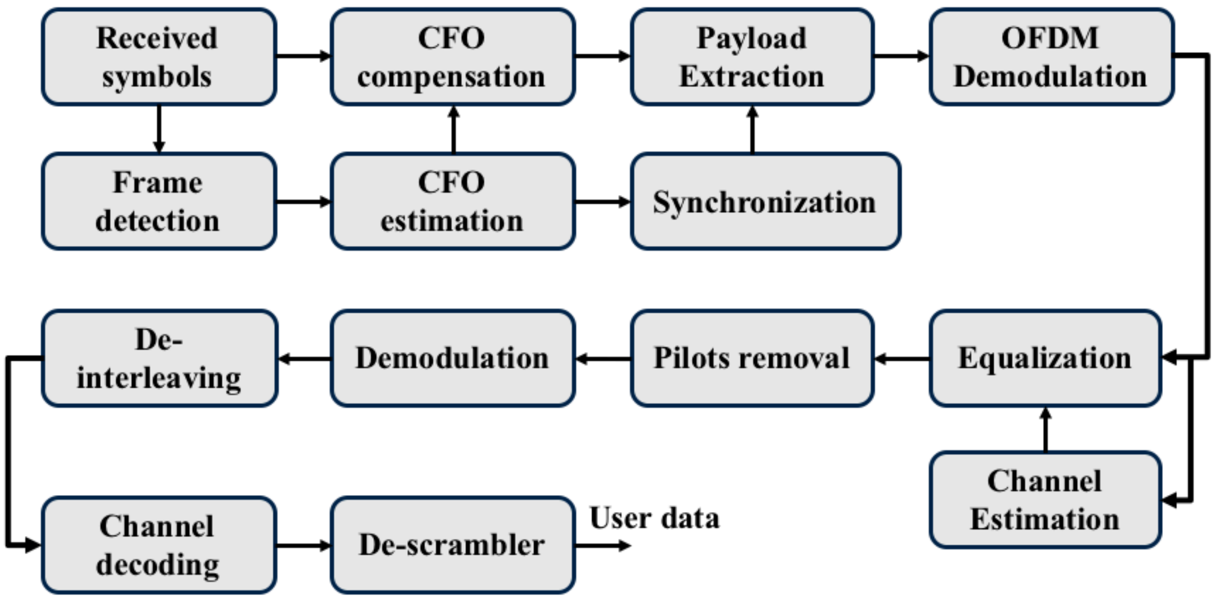

Figure 4. WiFi-Lite Protocol: Block Diagram of Receiver

2.3.1.2.1 Frame Detection

Synchronization is a necessary task for any digital communication system to detect the packets and frames at the receiver. Without accurate synchronization algorithms, it is not possible to reliably receive the transmitted data. To achieve proper synchronization, current WLAN standards uses preamble. Our frame detection algorithm uses the short preamble in the OFDM frame and performs autocorrelation to detect the packet. It then performs cross-correlation of the long preabmle sequence with the data samples to detect the start of the frame (Heiskala, 2001).

2.3.1.2.2 Channel Estimation and Equalization

Channel estimation and equalization algorithms are used in the receiver to remove the channel effects. Of the existing algorithms, first, the zero forcing algorithm was implemented which does channel estimation by doing a linear interpolation on the estimated pilots to form a channel response (H matrix). Since this algorithm did not suppress the channel effects completely, Least Square Estimation (LSE) algorithm was used (Cai & Giannakis, 2004). Channel equalization is achieved by dividing the received OFDM symbol with the estimated channel coefficients in the time domain, which nullifies the channel effect.

2.3.1.2.3 Demodulation

Demodulation is used to recover the information bits from the modulated symbols. Soft demodulator algorithm was used, which provides the reliability information of the bits. This reliability information is used by the Viterbi decoder to make better decisions.

2.3.1.2.3.1 QPSK Demodulation

QPSK soft demodulator which calculates the distance from the received symbol to the four possible constellation points (symbols in 2D space – I & Q samples) was implemented and the distance was used to make the decision to select an appropriate boundary region.

2.3.1.2.3.2 QAM Demodulation

QAM demodulator which calculates the distance from the received symbol to all the possible constellation points was implemented. It uses the log-likelihood ratio (LLR) algorithm to make the decision to select an appropriate boundary region.

2.3.1.2.4 Decoder

Decoder is used to decode the bitstream that has been encoded by an encoder at the transmitter.

2.3.1.2.4.1 Viterbi Decoder

Viterbi Decoder is used to decode the bitstream that has been encoded by a convolutional encoder. It is based on the Viterbi algorithm, which does maximum likelihood decoding (Viterbi, 1967).

2.3.2 SCFDM-Lite

The transmitter and receiver blocks used in SCFDM-Lite protocol are described in Sections 2.3.2.1 and 2.3.2.2 respectively.

2.3.2.1 Transmitter

The transmitter blocks used in SCFDM-Lite protocol are described here; the corresponding block diagram is shown in Figure 5. Most of the blocks are the same as WiFi-Lite protocol. The blocks that are different are only described in this section.

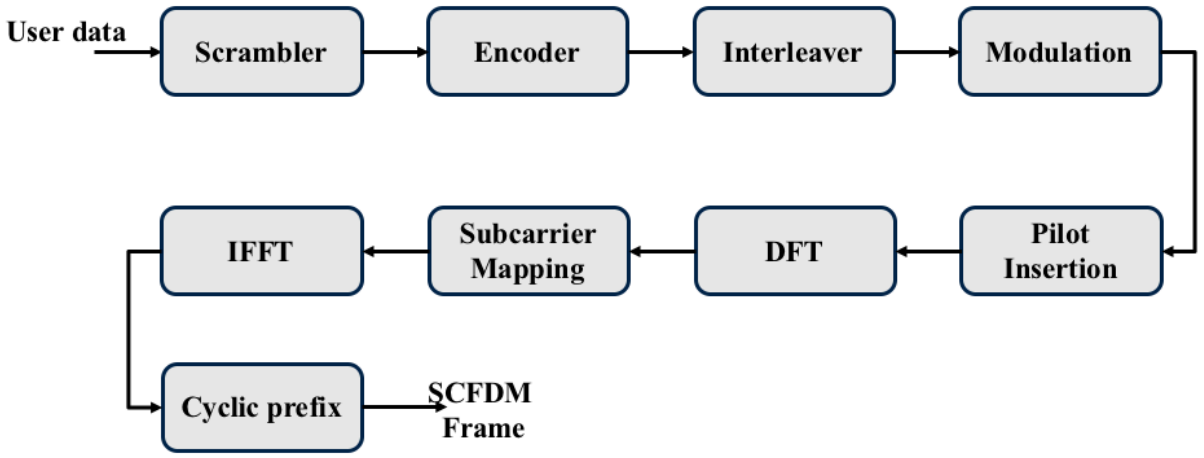

Figure 5. SCFDM-Lite Protocol: Block Diagram of Transmitter

2.3.2.1.1 Subcarrier Mapping

The subcarrier mapping assigns N-point DFT complex outputs values as the amplitude of some of the selected subcarriers in M-point IFFT, where M is greater than N. Subcarrier mapping can be classified into two types: localized mapping and distributed mapping. Localized mapping, was used in this protocol, where, the DFT output is mapped to a subset of consecutive sub-carriers (3GPP a global initiative, 2004).

2.3.2.2 Receiver

In this section, the receiver blocks used in our SCFDM-Lite protocol are described. Most of the blocks are the same as WiFi-Lite protocol, the only new block here is subcarrier de-mapping. Figure 6 gives the block diagram of the receiver.

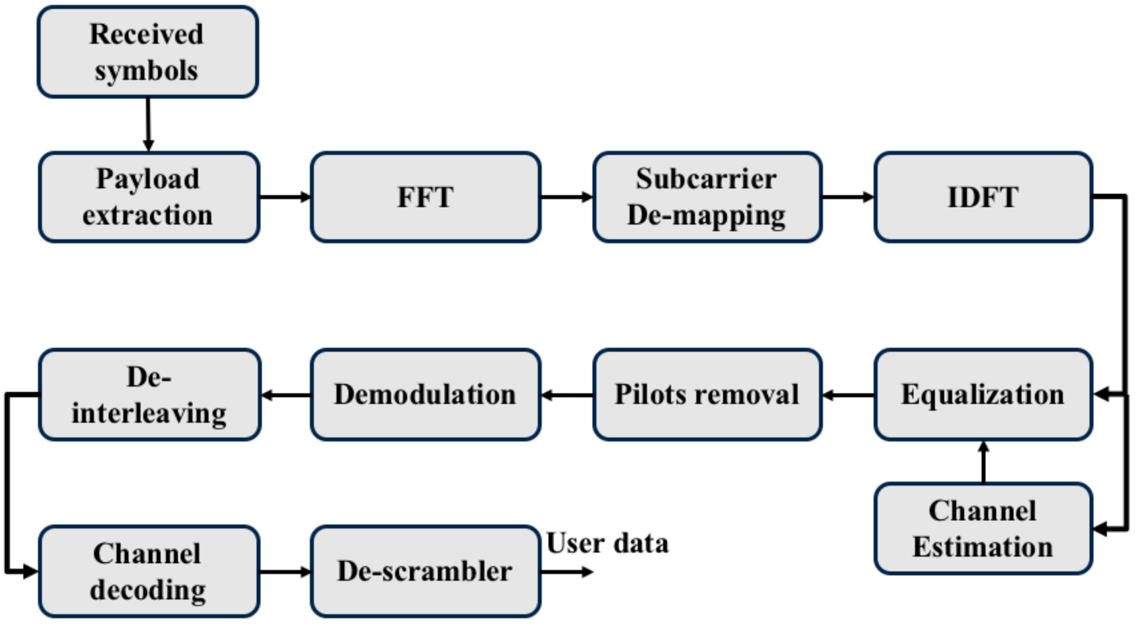

Figure 6. SCFDM-Lite Protocol: Block Diagram of Receiver

2.3.2.2.1 Subcarrier De-mapping

The subcarrier de-mapping de-maps the consecutive sub-carriers from an M-point FFT output and gives it to an N-point IDFT block, where M is larger than N.

2.3.3 SC-Lite

In this section, the transmitter and receiver blocks used in our SC-Lite protocol are described.

2.3.3.1 Transmitter

The transmitter block diagram of the SC-Lite protocol is described in Figure 7. The Scrambler block is reused from WiFi-Lite protocol; the encoder and modulation blocks are new.

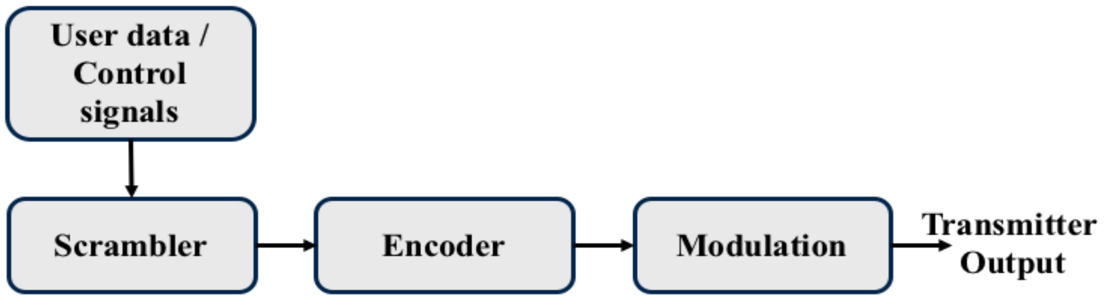

Figure 7. SC-Lite Protocol: Block Diagram of Transmitter

2.3.3.1.1 Reed-Solomon Encoder

Reed-Solomon (RS) encoder takes a block of digital data and adds extra redundant bits, it takes k data symbols of s bits each and adds parity sumbols to make an n symbol codeword. A Reed-Solomon code is a type of error-correcting linear block code, specified as (n,k) with s-bit symbols. The error capability of a RS code depends on the minumum distance which is n-k (Lin & Costello, 2004).

2.3.3.1.2 Binary Phase Shift Keying Modulation

BPSK is a digital modulation scheme that conveys data by changing the phase of a reference signal (the carrier wave). BPSK can only map 1 bit into a symbol.

2.3.3.2 Receiver

The receiver block diagram of our SC-Lite protocol is described in Figure 8. The descrambler block is reused from WiFi-Lite protocol; the Reed Solomon decoder and demodulation blocks are new.

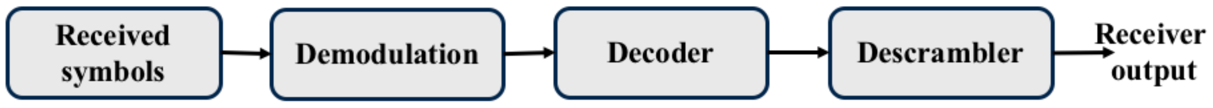

Figure 8. SC-Lite Protocol: Block Diagram of Receiver

2.3.3.2.1 BPSK Demodulation

Our BPSK demodulator makes decisions based on the reference point. If the value is below the reference point, then the demodulator assumes that the data bit as zero and if it is greater than the reference point, then it assumes that the data bit as one.

2.3.3.2.2 Reed-Solomon Decoder

Reed-Solomon (RS) Decoder is used to decode the symbols that has been encoded by a RS encoder. It consists of syndrome computation, key equation solver (KES), and Chien search and error evaluator (CSEE) units. (Minsky, 2010).

3 WiFi-Lite-A Protocol

In this chapter, the implementations of WiFi-Lite-A protocol on ARM and CEVA DSP platforms are described. An overview of the protocol is given in Section 3.1. This is followed by protocol implementation on ARM in Section 3.2, protocol implementation on ARM+CEVA in Section 3.3, protocol implementation on CEVA DSP in Section 3.4, over-the-air protocol implementation using Ettus Radio in Section 3.5, and over-the-air protocol implementing using Intel T2200 WBS boards in Section 3.6.

The work in Section 3.3.1 is done in collaboration with Ganapati Bhat, the work in Section 3.5 is done in collaboration with Dr. Hyunseok Lee, and the work in Section 3.6 is done in collaboration with Dr. Hyunseok Lee and Ganapati Bhat.

3.1 Protocol Overview

WiFi-Lite A is an Orthogonal Frequency Division Multiplexing (OFDM) based system similar to IEEE 802.11 standard. Our WiFi-Lite system uses 64 subcarriers that are modulated by quadrature phase shift keying (QPSK) followed by forward error correction (FEC) coding (convolutional coding) with a coding rate of ½.

3.1.1 Transmitter

At the transmitter, all the data bits undergo the following PHY layer signal processing. Data bits are scrambled by a linear non-additive scrambler using LFSR and then encoded by the convolutional encoder with a constraint length of 7 and code rate of ½. The bits are then interleaved by the linear random interleaver using LFSR, followed by QPSK modulation, pilot insertion, IFFT of width 64, cyclic prefix addition, and then by a long and short preamble (please refer to 2.3.1.1 section for transmitter block description). Figure 9 gives the block diagram of the transmitter chain for WiFi-Lite-A.

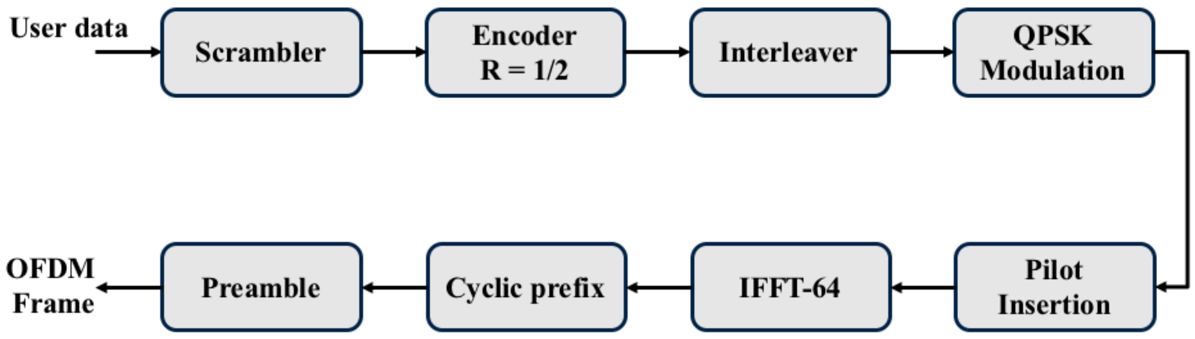

Figure 9. WiFi-Lite-A Protocol: Block Diagram of Transmitter

Protocol Configuration	Values
No of input data bits per OFDM symbol	40
No of data subcarriers per OFDM symbol	48
No of pilot subcarriers	16
FFT/IFFT size	64
Convolutional encoder	(2,1,8) Mask0 – 0133 Mask1 – 0171 Mask2 – 0165
Modulation	QPSK
Cyclic Prefix	16 symbols
Preamble	322 symbols

Table 1. WiFi-Lite-A Protocol Configuration

3.1.2 Receiver

At the receiver, the frame detection unit performs correlation of the received data samples with the known preamble sequence to detect a packet and also to detect the start of each frame. Once the frame is detected successfully, the cyclic prefix is removed, and the OFDM frame is given to the FFT-64 unit for converting the samples from time domain to frequency domain. This is followed by channel estimation and equalization to estimate and remove the channel effect. Next pilots are removed from the frame and sent to QPSK demodulator to extract the bits from the symbols. The de-interleaved data is given to the Viterbi decoder to remove the redundant bits and to rectify the errors bits. The bits are then descrambled to get the exact payload (please refer to 2.3.1.2 section for receiver blocks description). Figure 10 gives the block diagram of the receiver chain for WiFi-Lite-A.

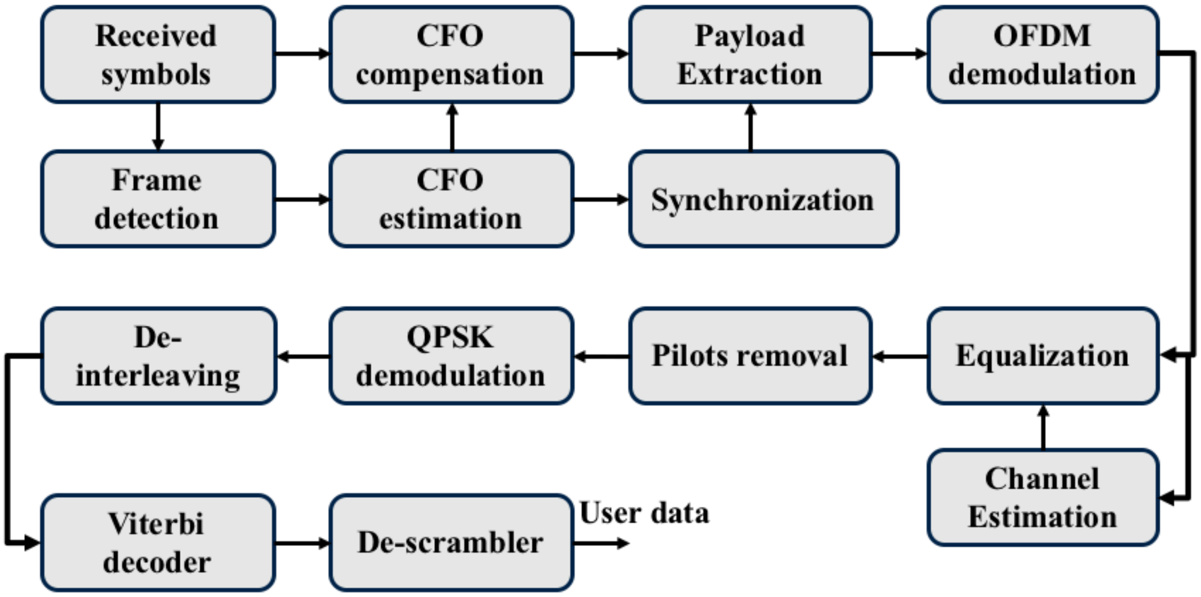

Figure 10. WiFi-Lite-A Protocol: Block Diagram of Receiver

3.2 WiFi-Lite-A Implementation on ARM

First, the WiFi-Lite-A transmitter and receiver chains were implemented on ARM. While the protocol development is easy using MATLAB, it was still developed entirely in C, because the MATLAB to C language converter was not very good. Figure 11 describes the experimental setup. The input to the transmitter chain is a sequence of 40 bits. These are converted into OFDM symbols. Transmitter chain output data is saved to a file. The channel model (Rayleigh fading) corrupts the data and saves the data to another file. The receiver chain reads the data from a file and decodes the payload. In our implementation the receiver chain successfully decodes the OFDM symbol with zero bit error rate.

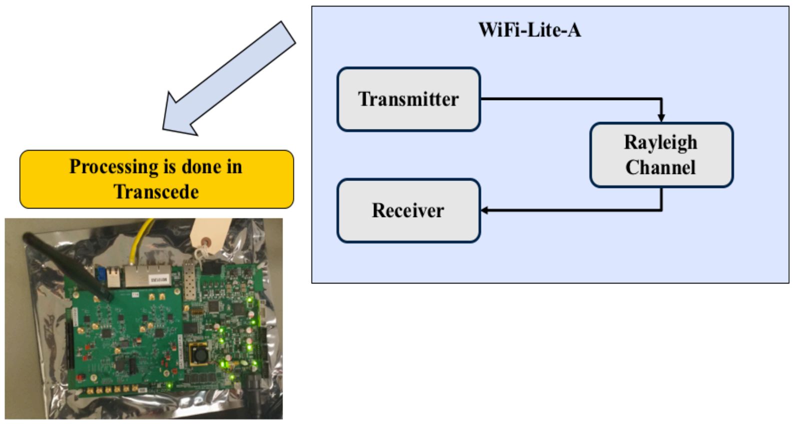

Figure 11. WiFi-Lite-A Implementation – TX and RX Loopback

3.2.1 Experimental Results

The timing performance of the different blocks in transmitter and receiver are calculated using the following way. Time stamps are introduced before and after each function call and the difference in the time is calculated to estimate the execution time of each block. Table 2 describes the execution time of the major functional blocks for the transmitter. The transmitter chain took 1.7ms with 95% of the time being spent on computing the 64 point IFFT.

Function	Time(μs)
Scrambler	2
Encoder	15
Interleaver	4
QPSK Modulation	12
Pilot Insertion	4
64-IFFT	1650
Cyclic Prefix	3
Preamble	54
Total	1744

Table 2. WiFi-Lite-A Transmitter Implementation on ARM: Timing Profile

Function	Time(μs)
64-FFT	2296
Channel Estimation & Equalization	2434
Pilot Removal	5
QPSK Demodulation	455
De-Interleaver	428
Decoder	2180
De-Scrambler	2
Total	7800

Table 3. WiFi-Lite-A Receiver Implementation on ARM: Timing Profile

Table 3 lists the execution time of the major functional blocks of the receiver chain. The results show that channel estimation and equalization, FFT, and decoder blocks took most of the computational time. These three blocks all together took 6.9ms, which is 88% of the total time.

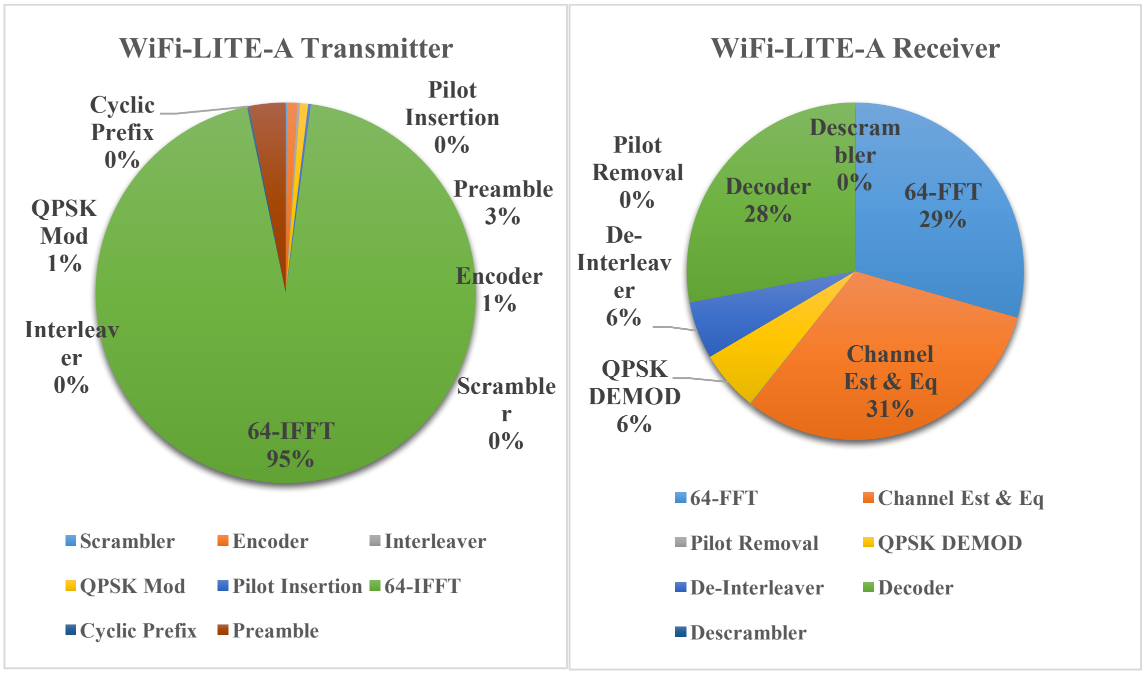

Figure 12. WiFi-Lite-A Profiling Results for ARM Implementation

Figure 12 shows the pie chart of the transmitter and receiver timing profiles. In the transmitter, IFFT took 95% of the total execution time. In the receiver, channel estimation and equalization block took 31%, FFT took 29%, Viterbi decoder took 28% of the execution time.

Since the FFT, IFFT, and decoder algorithms took most of the processing time, so these algorithms are mapped onto the CEVA DSP and the timing performance evaluated.

3.3 WiFi-Lite-A Implementation on ARM and CEVA DSP

In order to map some of the blocks, namely the IFFT, FFT, and Viterbi decoder blocks onto the CEVA DSP, and map the remaining blocks onto ARM, the interface between ARM and CEVA had to be designed.

3.3.1 ARM-CEVA Interface

The communication between ARM and CEVA was implemented using the polling mechanism. It utilizes a generic function whose arguments include the task ID, pointer to an input buffer, pointer to an output buffer and length of the input data. Task ID is the name of the library function, which CEVA has to call during runtime. Our function takes the input buffer, does type casting to the respective data type and does a memory copy to the shared memory. Once the function maps the buffers to the respective data types, it makes the Task Control Block (TCB) status ready so that CEVA can read the data and execute the specific function. Once CEVA is done executing the function, it copies back the result from shared memory to the output buffer, so that the next block in the chain can read the data and proceed with the execution. At the end of this process, the function changes the Task Control Block (TCB) flag status to completed.

CEVA DSP is a fixed point processor, so CEVA soft libraries are in fixed point. Since all computations in ARM are done in floating point, data handling has to be done while passing the data from the floating point buffer to the fixed point buffer. The output data of a floating point function is multiplied by a scalar value; the scalar value is decided based on the data types of the function input and output buffers. CEVA DSP libraries expect 16-bit values (in C language – short data type), so by considering the overflow cases, the data is multiplied by 1000 to get better results. In spite of the scaling, there was some loss in precision, as expected.

CEVA FFT soft library functions give scaled outputs, which have to be handled to get the correct results. CEVA FFT library function outputs were compared with the MATLAB function outputs by passing a known input data to find the exact scalar value. Table 4 shows the scalar values for some of the FFT libraries.

CEVA FFT Library	Scalar Value
FFT64	Output is scaled by 4
IFFT128	Output is scaled by 8
FFT128	Output is scaled by 16

Table 4. Scaling Factor for CEVA FFT Library Functions

3.3.2 Experimental Results

The experimental setup is the same as shown in Figure 11. A sequence of 40 bits is processed by the transmitter chain, corrupted by Rayleigh fading channel and decoded successfully by the receiver chain. DSP and communication libraries (C & ASM) provided by CEVA are used to improve baseband timing performance. Specifically, the FFT and Viterbi decoder libraries (ASM) are mapped onto CEVA DSP. This mapping resulted in significant reduction in the computational time of both the transmitter and receiver chains. The transmitter chain took 0.69µs compared to 1.7ms when implemented on only ARM and the receiver chain took 2.9ms compared to 7.8ms when implemented on only ARM.

Function	Time(µs)
Scrambler	2
Encoder	15
Interleaver	4
QPSK Modulation	12
Pilot Insertion	4
64-IFFT	601
Cyclic Prefix	3
Preamble	51
Total	692

Table 5. WiFi-Lite-A Transmitter Implementation on ARM+CEVA: Timing Profile

The profiling results in Table 5 show that the IFFT block took most of the transmitter chain computational time. The IFFT block execution time is reduced to almost 60% when compared to the only ARM implementation (Table 2).

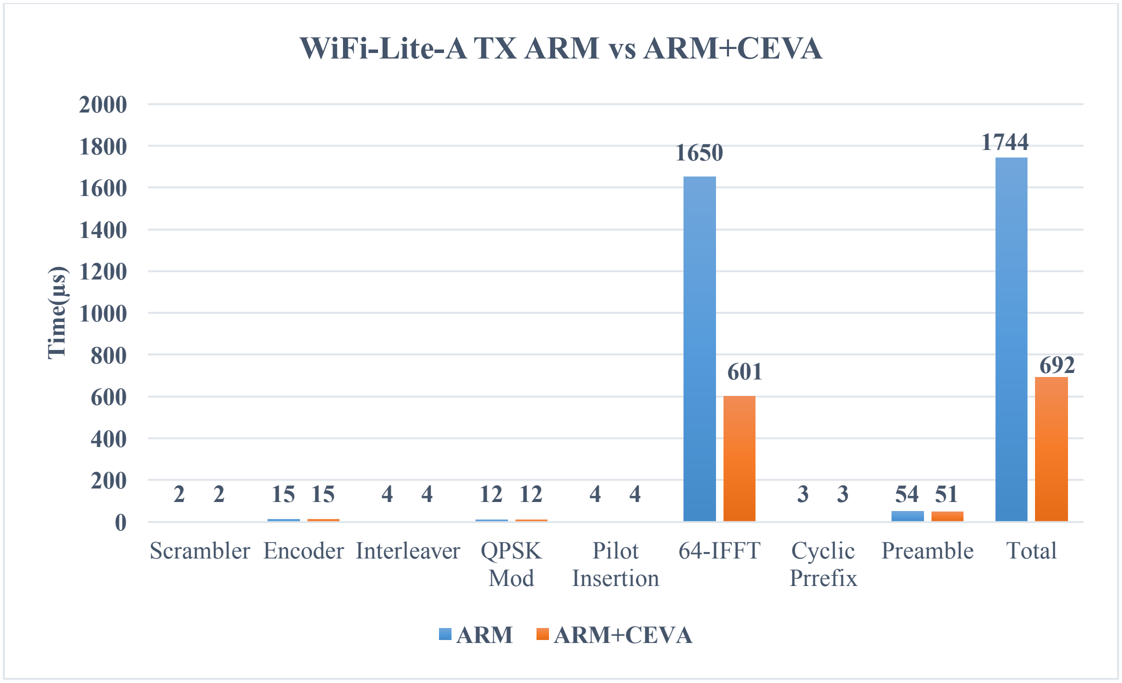

Figure 13. WiFi-Lite-A TX Execution Time Values - ARM vs ARM+CEVA

Figure 13 compares the execution times of the transmitter chain implemented using ARM with that using ARM+CEVA. Other than the IFFT block which was mapped onto CEVA, the other blocks took the same time since they are implemented on ARM. The reduction in the IFFT block timing translated to almost 60% reduction in the transmitter chain execution time.

Function	Time(μs)
64-FFT	645
Channel Estimation & Equalization	855
Pilot Removal	5
QPSK Demodulation	450
De-Interleaver	420
Decoder	576
De-Scrambler	5
Total	2956

Table 6. WiFi-Lite-A Receiver Implementation on ARM+CEVA: Timing Profile

The profiling results in Table 6 show that for the receiver chain, the channel estimation and equalization, FFT, and decoder blocks took most of the computational time. When compared to the ARM only implementation, decoder, FFT, and channel estimation block execution time reduced significantly. Overall, the execution time of the ARM+CEVA implementation reduced from 7.8ms to 2.9ms.

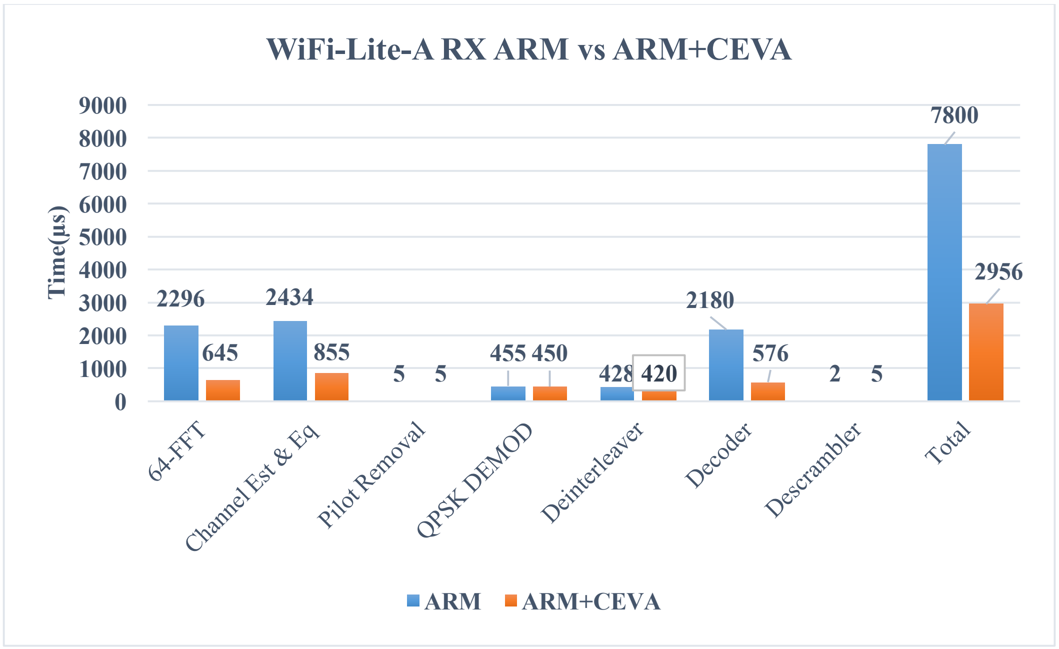

Figure 14. WiFi-Lite-A RX Execution Time Values - ARM vs ARM+CEVA

Figure 14 compares the execution times of the receiver chain implemented using ARM with that using ARM+CEVA. The results show that the FFT block and decoder took significantly lower time on CEVA DSP than on ARM. The channel estimation and equalization block also took less time in ARM+CEVA implementation, since the FFT used in the channel estimation is implemented on the CEVA DSP. The timing of the other blocks in the receiver chain remain the same since they are implemented on ARM in both cases. The total execution time of a receiver chain for ARM+CEVA implementation is reduced by almost 62% compared to the execution time for only ARM implementation.

3.4 WiFi-Lite-A Fixed-Point Implementation on CEVA DSP

Next, all blocks of the transmitter and receiver chains were implemented on the CEVA DSP. This was done to avoid the communication overhead and to reduce the memory overhead between ARM and CEVA DSP. Figure 15 describe the setup. Transmitter chain output data is saved to a file, the channel model (Rayleigh fading) corrupts the data and saves the data to a file. The receiver chain reads the data from a file and decodes the payload. At first, all processing is done in CEVA SDK and then all the blocks are mapped onto CEVA DSP hardware.

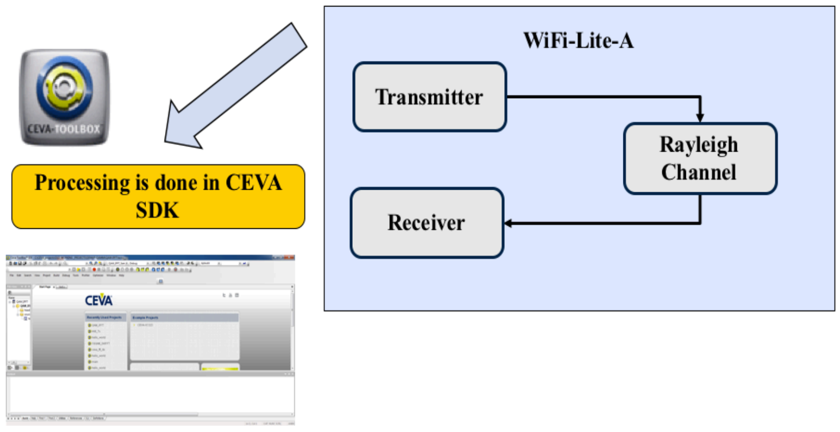

Figure 15. WiFi-Lite-A CEVA SDK Implementation

3.4.1 Experimental Results – CEVA DSP SDK

The protocol was implemented on CEVA SDK and the execution time for each block was calculated. Tables 7 and 8 include the timing results of each block in the transmitter and receiver, respectively.

Function	Cycles	Time(µs)
Scrambler	214	0.3
Encoder	2297	3
Interleaver	610	0.8
QPSK Modulation	2119	2.8
Pilot Insertion	497	0.6
IFFT	70	0.09
Total	5807	7.6

Table 7. WiFi-Lite-A Transmitter Implementation on CEVA SDK: Timing Profile

From Table 7, the two computationally expensive operations in the transmitter chain are the encoder which took 3µs, and QPSK modulation which took 2.8µs. Together they account for 76% of the total execution time. The IFFT block took only 0.09µs which is significantly lower compared to the ARM+CEVA implementation.

Function	Cycles	Time(µs)
FFT	70	0.1
Channel Est & Eq	48469	64.6
Pilot removal	932	1.2
QPSK Demodulation	33	0.04
De-Interleaver	5496	7.3
Viterbi Decoder	2490	3.3
De-Scrambler	214	0.3
Total	57704	76.8

Table 8. WiFi-Lite-A Receiver Implementation on CEVA SDK: Timing Profile

The profiling results of the receiver chain presented in the Table 8 show that channel estimation and equalization algorithm took 64.6µs out of the total 76.8µs. While this is very large, the CEVA SDK implementation takes significantly less time compared to the ARM+CEVA implementation (see Table 6). Also, FFT, QPSK Demodulation, and decoder blocks only took 0.1µs, 0.04µs, and 3.3µs respectively, which is very low when compared to the ARM+CEVA implementation results.

CEVA SDK timing results are significantly lower since SDK runs on the general purpose processor (Intel Xeon) and so it does not exactly emulate the cost of inter-process communication and interrupts. It also does not consider the exact size of the memory available on the hardware. Since SDK does not provide accurate estimates of the execution time on real hardware, next we implement the protocol on CEVA DSP hardware.

3.4.2 Experimental Results – CEVA DSP Hardware

All the transmitter and receiver blocks are mapped onto CEVA DSP hardware and the execution time for each block was calculated. Tables 9 and 10 include the timing results of each block in the transmitter and receiver, respectively.

Function	Cycles	Time(µs)
Scrambler	498	1.9
Encoder	8371	33.4
Interleaver	1644	6.5
QPSK Mod	1646	6.5
Pilot insert	1302	5.2
IFFT-64	58	0.2
Cyclic Prefix	1664	6.6
Preamble	6534	26.1
Total	21717	86.8

Table 9 WiFi-Lite-A Transmitter Implementation on CEVA DSP: Timing Profile

From Table 9, the two computationally expensive operations in the transmitter chain are the encoder which took 33µs, and preamble block which took 26µs. Together they account for 69% of the total execution time. The IFFT block took only 0.2µs which is significantly lower compared to the ARM+CEVA implementation but slightly higher compared to CEVA SDK implementation.

Function	Cycles	Time(µs)
FFT-64	57	0.2
Channel Est & Eq	20350	81.4
Pilot removal	2608	10.4
QPSK De-mod	66	0.2
De-Interleaver	3104	12.4
Decoder	1167	4.6
Descrambler	466	1.8
Total	27818	111.2

Table 10 WiFi-Lite-A Receiver Implementation on CEVA DSP: Timing Profile

The profiling results of the receiver chain presented in the Table 10 show that channel estimation and equalization algorithm took 81.4µs out of the total 111.2µs. While this is very large, the CEVA DSP hardware implementation takes significantly less time compared to the ARM+CEVA implementation (see Table 6). Also, FFT, QPSK Demodulation, and decoder blocks only took 0.2µs, 0.2µs, and 4.6µs respectively, which is very low when compared to the ARM+CEVA implementation results.

Function	Time(µs) ARM+CEVA	Time(µs) CEVA hardware
Scrambler	2	1.9
Encoder	15	33.4
Interleaver	4	6.5
QPSK Modulation	12	6.5
Pilot Insertion	4	5.2
64-IFFT	601	0.2
Cyclic Prefix	3	6.6
Preamble	51	26.1
Total	638	86.8

Table 11. WiFi-Lite-A Transmitter Profiler Results - ARM+CEVA vs CEVA DSP

Table 11 compares the execution times of the transmitter chain implemented using ARM+CEVA and CEVA DSP hardware. The IFFT block took significantly lower time on CEVA DSP when compared to ARM+CEVA results. While in both cases, the IFFT was implemented on CEVA, in ARM+CEVA implementation there is a wrapper function which handles the polling mechanism between CEVA DSP and ARM and adds to the timing complexity. In comparison, in the CEVA DSP implementation, the complete transmitter chain is implemented on CEVA DSP and there is no need for any wrapper function. Of the remaining blocks, QPSK modulation and preamble blocks took much lower time on CEVA DSP compared to ARM+CEVA implementation where these blocks are implemented on ARM. In contrast, the encoder, interleaver, and cyclic prefix blocks took more time on CEVA DSP when compared to ARM+CEVA implementation, where these blocks are implemented on ARM. This is possibly because, the ARM compiler optimized the scalar codes better. The total execution time of the transmitter chain for CEVA DSP implementation is reduced by almost 86% compared to the transmitter chain execution for ARM+CEVA implementation.

Function	Time(µs) ARM+CEVA	Time(µs) CEVA hardware
64-FFT	645	0.2
Channel Estimation & Equalization	855	81.4
Pilot Removal	5	10.4
QPSK Demodulation	450	0.2
De-Interleaver	420	12.4
Decoder	576	4.6
De-Scrambler	5	1.8
Total	2380	111.2

Table 12. WiFi-Lite-A Receiver Profiler Results - ARM+CEVA vs CEVA DSP

Table 12 compares the execution times of the receiver chain implemented using ARM+CEVA and CEVA DSP. The IFFT, decoder and QPSK demodulation blocks took significantly lower time on CEVA DSP when compared to ARM+CEVA implementation. While these blocks are implemented on CEVA in both implementations, the ARM+CEVA implementation has wrapper function which handles the polling mechanism between CEVA DSP and ARM. In contrast, the CEVA DSP hardware implementation, has no wrapper function because the complete receiver chain is implemented on CEVA DSP. The de-scrambler and channel estimation and equalization blocks took much lower time on CEVA DSP compared to ARM+CEVA implementation where these blocks also are implemented on ARM. The reduction in channel estimation was expected since the FFT block in channel estimation was mapped on CEVA DSP of the remaining blocks, the pilot removal block took more time on CEVA DSP when compared to ARM+CEVA implementation, where the block was implemented on ARM, because the ARM compiler possibly optimized the scalar code better. The total execution time of a receiver chain for CEVA DSP hardware implementation is reduced by almost 95% compared to the receiver chain execution for CEVA+ARM implementation. In summary, a 30x - 80x reduction in the execution time is achieved by running the complete receiver and transmitter chain on the CEVA DSP hardware.

3.5 WiFi-Lite-A Implementation using Ettus Radio

GNURadio was used as a platform for over-the-air communication. The transmitter chain (GUI blocks – signal processing blocks) for the WIFI-Lite-A protocol was developed using the GNURadio OOT model. Ettus radio (N210) is used to transmit and receive the I and Q samples as shown in Figure 16. At the receiver, data samples from the ADC are saved into a buffer and sent to the Intel Transcede board for decoding.

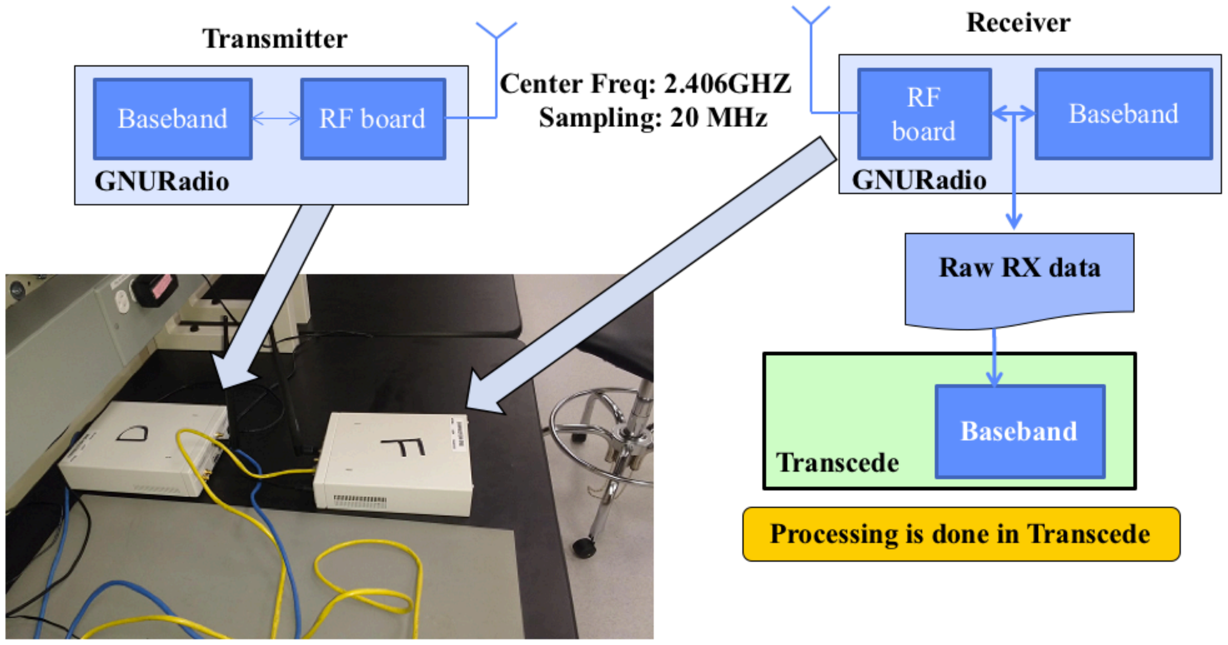

Figure 16. WiFi-Lite-A: Over the Air Experiment Using Ettus Radio and Intel WBS

GNURadio OOT model provides an excellent framework to create the individual blocks. The key issues in the development were handling input and output buffers of each block, monitoring the length of the data in the buffers and scheduling block processing. The input buffer of each block was monitored. If the data in the input buffer is greater than zero, then the block starts processing the data and gives it to the next block in the chain, as in cut through implementation. The other ways such as store and forward mechanism was also implemented (storing one complete packet/frame in the input buffer of the block and then allowing the block to process), but observed buffer overflow, underflow, and output mismatch issues, so the cut through based mechanism was finally used.

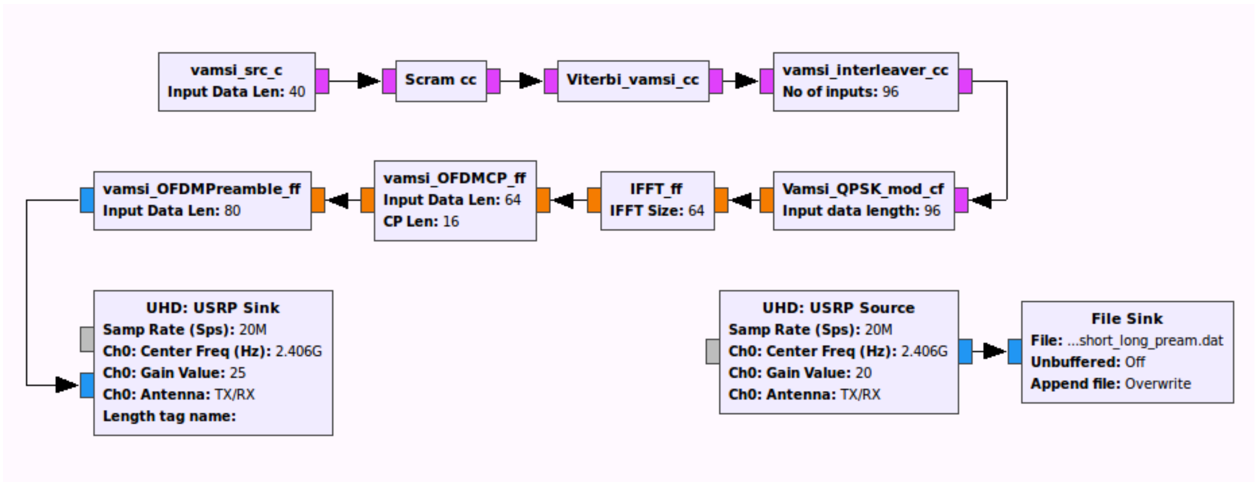

Figure 17. WiFi-Lite-A GNURadio Transceiver Block Diagram

3.5.1 Experimental Results

To verify that the transmitter and receiver chain functioned correctly, 40 bits of payload data were transmitted and received (over the air data samples) using Ettus Radios as shown in Figure 17. At the receiver, data samples are received and saved into a file and the dump file is sent to Transcede board for decoding. Our receiver algorithm successfully decoded the dump file with zero bit error rate (BER).

Figure 18 explains the synchronization result of the frame detection algorithm. Here, the x-axis represents the number of samples and y-axis represents the amplitude of the samples. The red color shows the received data samples. The green color line shows the correlation result of the data samples with the short preamble, the correlation peak (green color peak) is the starting point of the packet. The blue color shows the correlation result of the data samples with the long preamble; the correlation peak (blue color peak) is the starting point of the OFDM frame.

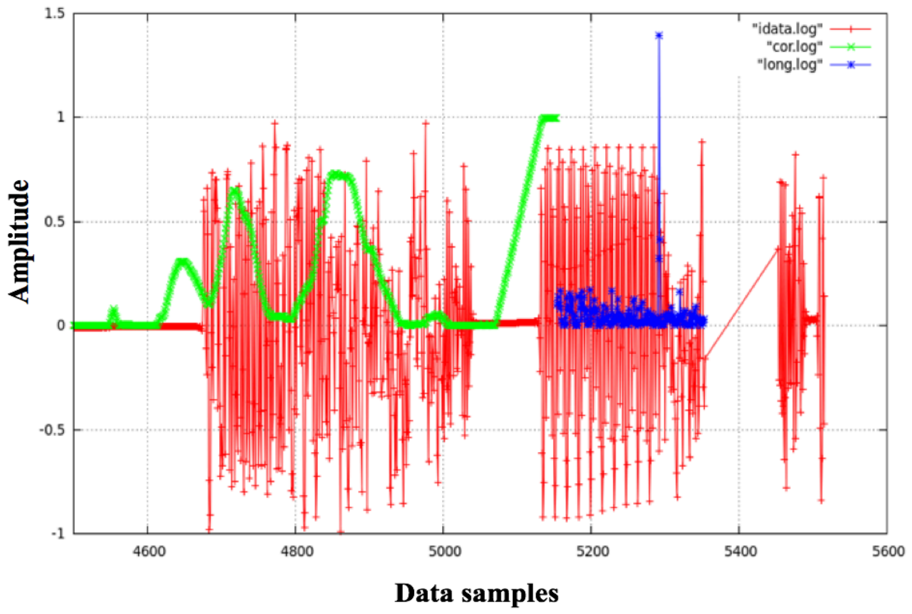

Figure 18. WiFi-Lite-A GNU Radio Implementation - Frame Detection Results

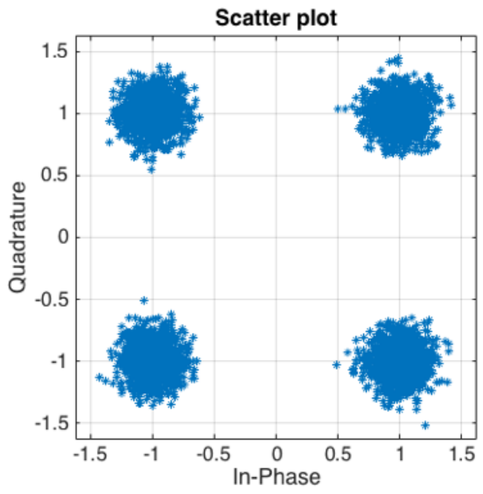

Figure 19. WiFi-Lite-A – GNU Radio QPSK Constellation Diagram

Figure 19 shows the constellation diagram of the symbols before the QPSK demodulation block. It shows that our channel estimation and equalization algorithm removed the channel effect in the OFDM symbol.

3.6 Implementation Using Intel T2200 for TX & RX – Real Time Processing

Two Intel T2200 WBS boards are used: one as a transmitter, and the other as a receiver to communicate data over the air using two different frequencies to communicate. Specifically, frequency 2635Mhz is used to transmit data from board A to board B and frequency 2675Mhz is used to transmit data from Board B to Board A as shown in Figure 20. The processing on the T2200 boards was done using only ARM.

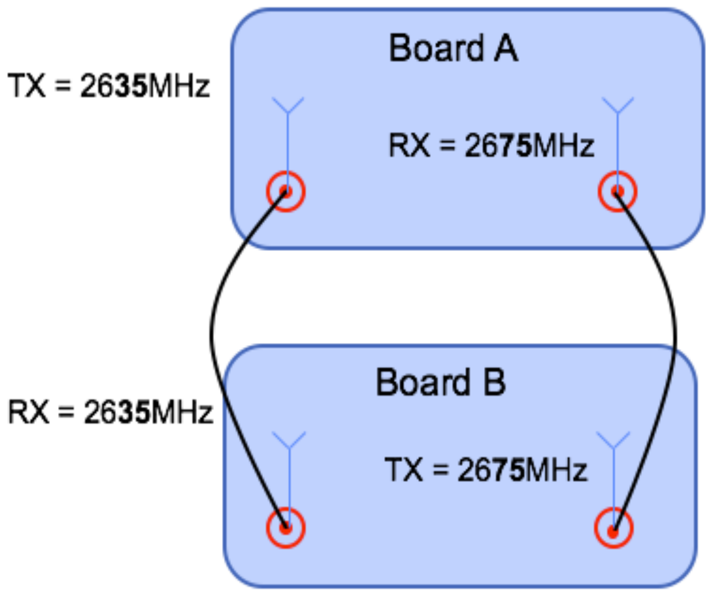

Figure 20. Intel T2200 WBS Dual Frequency Setup

3.6.1 Experimental Results

The RF cards are configured as shown in Figure 20. Each board can transmit and receive data simultaneously. A fixed string of 124 bytes long data is sent per OFDM frame (40 bits per OFDM symbol) from one board to another and vice versa.

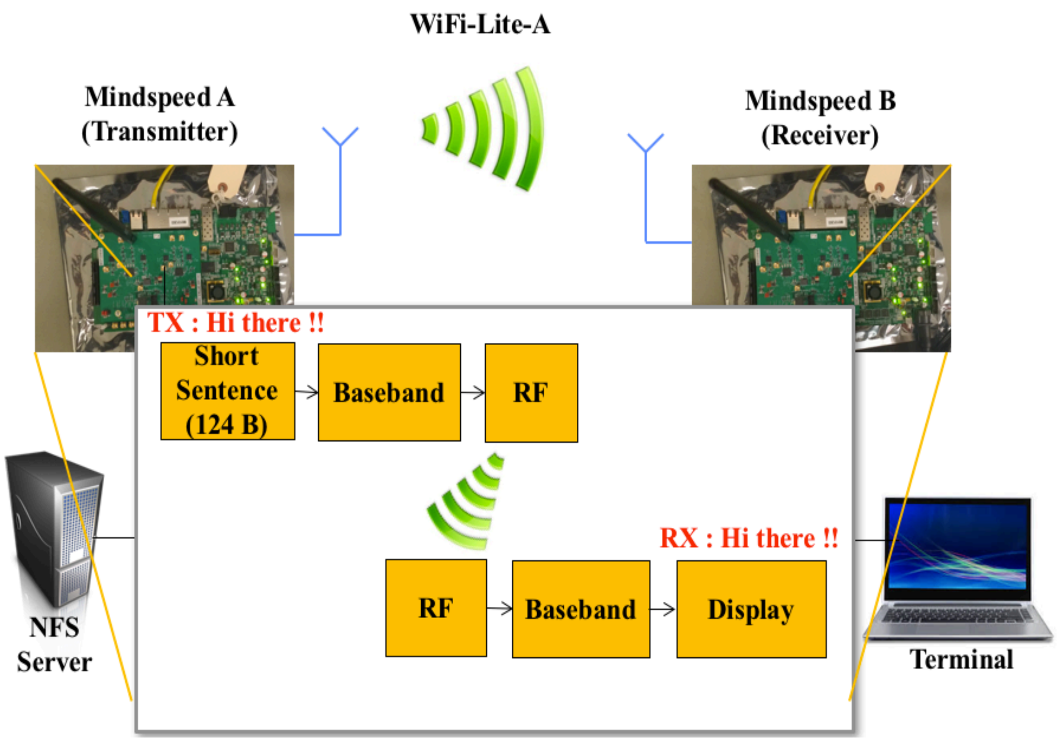

Figure 21. WiFi-Lite-A Over the Air Implementation - Intel T2200 WBS Boards

The data (124 bytes long) was decoded successfully with zero BER. The receiver algorithms and RF configurations worked perfectly. Figure 22 shows the synchronization result of our frame detection algorithm. The x-axis represents the number of samples and y-axis represents the amplitude of the samples. The red color shows the received data samples. The green color line shows the correlation result of the data samples with the short preamble, the correlation peak (green color peak) is the starting point of the packet. The blue color shows the correlation result of the data samples with the long preamble. The pink color shows the payload data samples. Therefore, the frame detection algorithm successfully detected the packet and the payload.

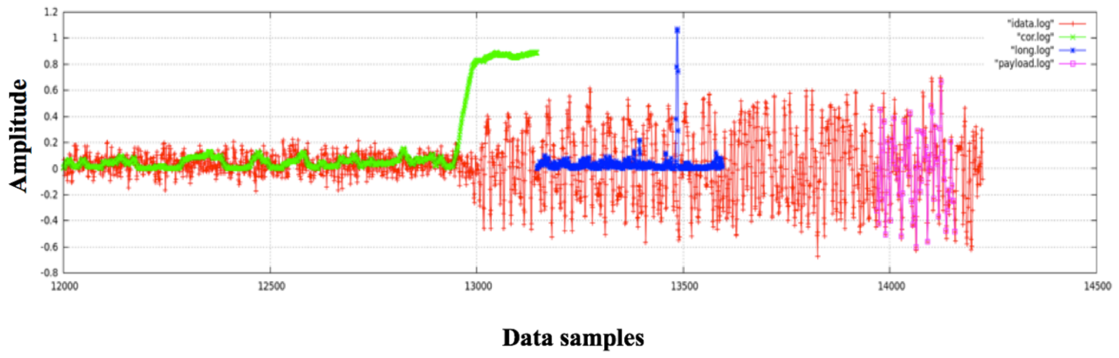

Figure 22. WiFi-Lite-A Over-the-Air Frame Detection Algorithm Result

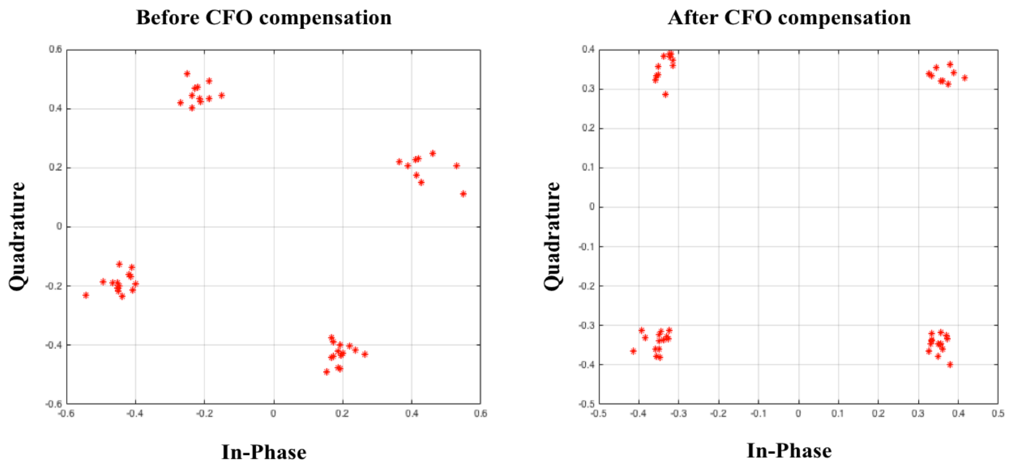

Figure 23. WiFi-Lite-A Carrier Frequency Offset Algorithm Result

Figure 23 describes the constellation diagram before and after CFO compensation. The x-axis of Figure 23 shows the in-phase component and y-axis shows the quadrature components of the samples. Here autocorrelation is performed on the short preamble sequence at the receiver to find the phase shift. The algorithm successfully corrected 20-degree phase shift in the symbols.

4 WiFi-Lite-B Protocol

In this chapter, the implementations of WiFi-Lite-B protocol on ARM and CEVA DSP platforms are described. An overview of the protocol is given in Section 4.1, followed by protocol implementation on ARM in Section 4.2, protocol implementation on ARM+CEVA in Section 4.3, protocol implementation on CEVA SDK in Section 4.4 and over-the-air protocol implementation using Intel T2200 WBS boards in Section 4.5. The work in Section 4.5 is done in collaboration with Dr. Hyunseok Lee.

4.1 Protocol Overview

WiFi-Lite B is an Orthogonal Frequency Division Multiplexing (OFDM) based system similar to IEEE 802.11 standard. Our WiFi-Lite system uses 64 subcarriers that are modulated by quadrature amplitude modulation (QAM) followed by forward error correction (FEC) coding (convolutional coding) with a coding rate of 2/3.

4.1.1 Transmitter

At the transmitter, the data bits are first scrambled by a linear non-additive scrambler using LFSR and then encoded by the convolutional encoder with a constraint length as 7 and code rate of 2/3. The bits are then interleaved by the linear random interleaver using LFSR, followed by QAM modulation, pilot insertion, IFFT of width 64, cyclic prefix addition, and then by a long and short preamble (please refer to 2.3.1.1 section for transmitter block description). Figure 24 describes the block diagram of the WiFi-Lite-B transmitter.

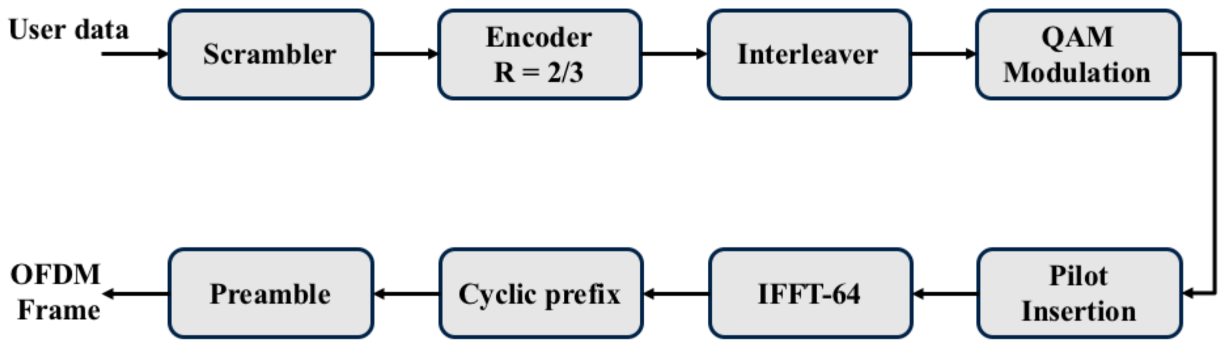

Figure 24. WiFi-Lite-B Protocol: Block Diagram of Transmitter

Protocol Configuration	Values
No of input data bits per OFDM Symbol	136
No of data subcarriers per OFDM Symbol	48
No of pilot subcarriers	16
FFT/IFFT width	64
Convolutional Encoder	(3,2,8) Mask0 – 0133 Mask1 – 0171 Mask2 – 0165
Modulation	QAM
Cyclic Prefix	16 symbols
Preamble	322 symbols

Table 13. WiFi-Lite-B Protocol Configuration

4.1.2 Receiver

At the receiver, the frame detection unit performs correlation of the received data samples with the known preamble sequence to detect a packet and also to detect the start of each frame. Once the frame is detected successfully, the cyclic prefix is removed, and the OFDM frame is sent to the FFT-64 unit for converting the samples from time domain to frequency domain. This is followed by channel estimation and equalization to estimate and remove the channel effect. Next pilots are removed from the frame and sent to QAM demodulator to extract the bits from the symbols. The de-interleaved data is given to the Viterbi decoder to remove the redundant bits and to rectify the errors bits. The bits are then descrambled to get the exact payload (please refer to 2.3.1.2 section for receiver block description). Figure 25 describes the block diagram of the WiFi-Lite-B receiver.

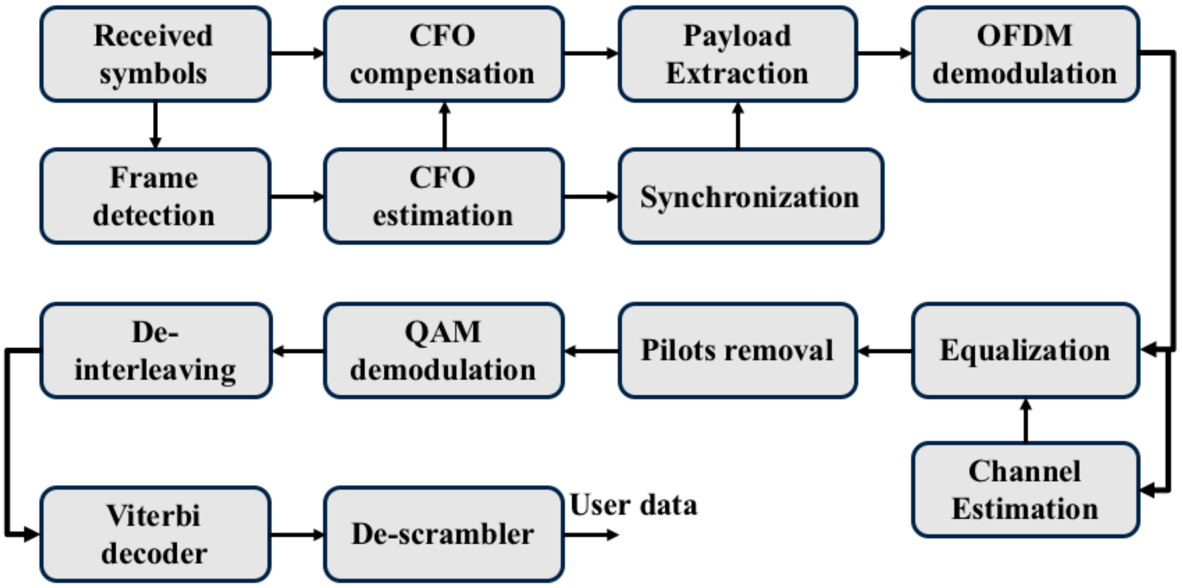

Figure 25. WiFi-Lite-B Protocol: Block Diagram of Receiver

4.2 WiFi-Lite-B Implementation on ARM

First, the transmitter and receiver chains were implemented on ARM. The protocol was developed entirely using the C language, because the MATLAB to C language converter was not very good. Figure 26 describes the experimental setup. The input to the transmitter chain is a sequence of 136 bits. These are converted into OFDM symbols. Transmitter chain output data is saved to a file, the channel model (Rayleigh fading) corrupts the data and saves the data to another file. The receiver chain reads the data from a file and decodes the payload. In our implementation, the receiver chain successfully decoded the OFDM symbol with zero bit error rate.

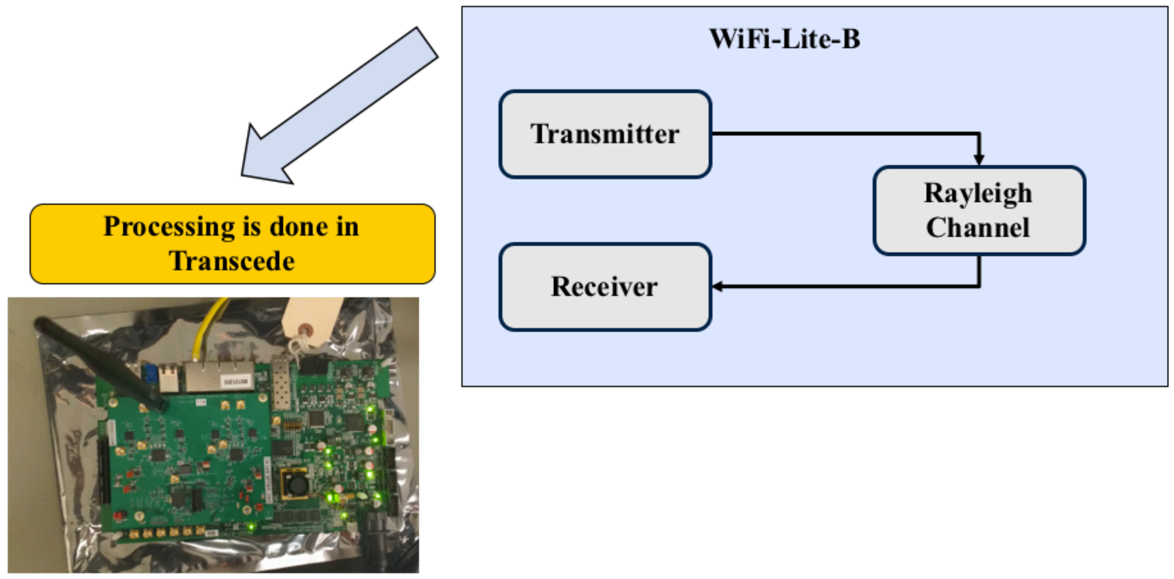

Figure 26. WiFi-Lite-B Protocol Implementation - TX and RX Loopback

4.2.1 Experimental Results

The timing performance of the different blocks in transmitter and receiver was evaluated. Time stamps were introduced before and after each function call and the difference in the time was calculated to estimate the execution time of each block. Table 14 describes the execution time of the major functional blocks for the transmitter. The transmitter chain took 1.7ms with 94% of the time being spent on computing the 64 point IFFT.

Function	Time(μs)
Scrambler	5
Encoder	41
Interleaver	6
16Qam Modulation	8
Pilot Insertion	3
64-IFFT	1598
Cyclic Prefix	4
Preamble	41
Total	1706

Table 14. WiFi-Lite-B Transmitter Implementation on ARM: Timing Profile

Function	Time(µs)
64-FFT	2250
Channel Equalization and Estimation	2459
Pilot Removal	5
16 QAM demodulation	645
De-Interleaver	488
Viterbi Decoder	5767
Descrambler	5
Total	11619

Table 15. WiFi-Lite-B Receiver Implementation on ARM: Timing Profile

Table 15 lists the execution time of the major functional blocks of the receiver chain. The results show that channel estimation and equalization, FFT, and decoder blocks take most of the computational time. These three blocks took 10.4ms, which is 90% of the total time.

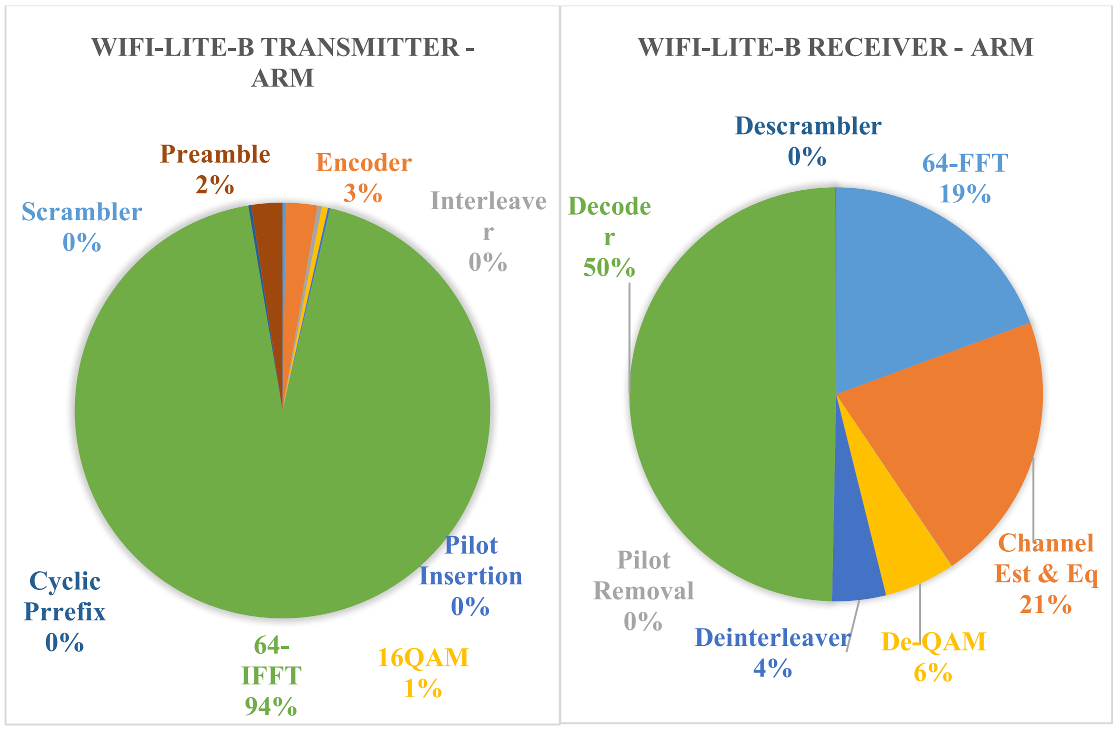

Figure 27. WiFi-Lite-B Profiling Results for ARM Implementation

Figure 27 shows the pie chart of the transmitter and receiver timing profiles. In the transmitter, IFFT took 94% of the total execution time. In the receiver, channel estimation and equalization block took 21%, FFT took 19%, and the Viterbi decoder took 50% of the execution time. Since the FFT and decoder algorithms took most of the processing time, these algorithms were mapped onto the CEVA DSP and their timing performance re-evaluated.

4.3 WiFi-Lite-B Implementation on ARM and CEVA DSP

In order to map the IFFT, FFT, and Viterbi decoding algorithms onto the CEVA DSP, and map the remaining algorithms onto ARM, a polling mechanism was used to communicate between ARM and CEVA (refer to Section 3.3.1 for ARM-CEVA interface details).

4.3.1 Experimental Results

The experimental setup is the same as shown in Figure 26. A sequence of 136 bits is processed by the transmitter chain, corrupted by Rayleigh fading channel and decoded successfully by the receiver chain. DSP and communication libraries (C & ASM) provided by CEVA are used to improve baseband timing performance. Specifically, the FFT and Viterbi decoder libraries (ASM) are mapped onto CEVA DSP. This mapping resulted in significant reduction in the computational time of both the transmitter and receiver chains. The transmitter chain took 0.72µs compared to 1.7ms for the only ARM implementation and the receiver chain took 3.2ms compared to 11.6ms for the only ARM implementation.

Function	Time(µs)
Scrambler	5
Encoder	41
Interleaver	6
16Qam Modulation	8
Pilot Insertion	3
64-IFFT	620
Cyclic Prefix	4
Preamble	41
Function	728

Table 16. WiFi-Lite-B Transmitter Implementation on ARM+CEVA: Timing Profile

The profiling results in Table 16 show that the IFFT block took most of the transmitter chain computational time. The IFFT block execution time is reduced to almost 61% when compared to the only ARM implementation (Table 14).

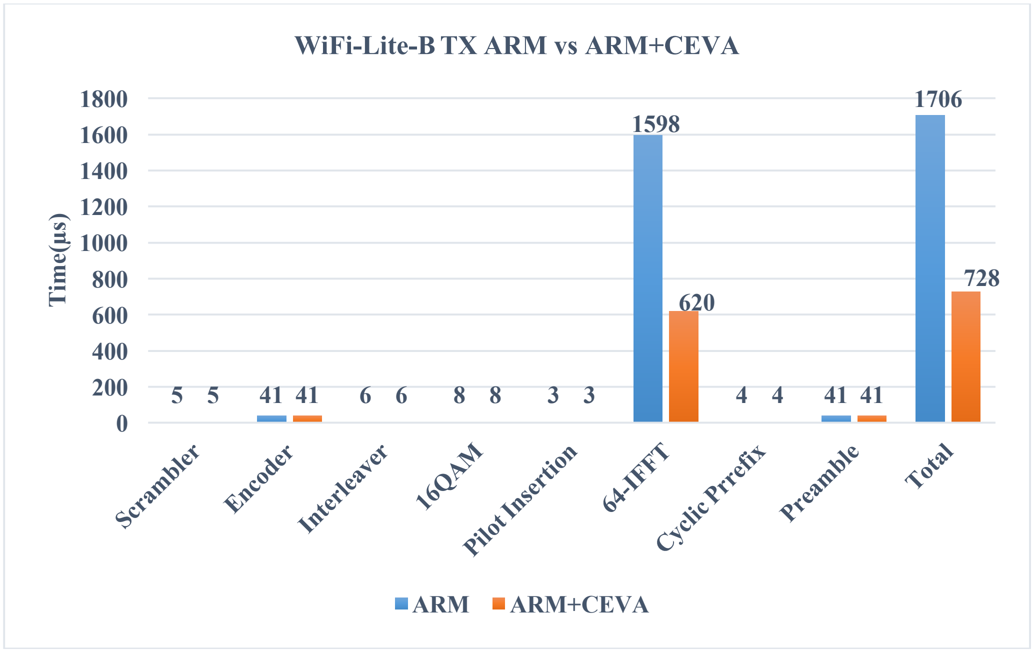

Figure 28. WiFi-Lite-B TX Execution Time Values - ARM vs ARM+CEVA

Figure 28 compares the execution times of the transmitter chain implemented using only ARM with that using ARM+CEVA. Other than the IFFT block which was mapped onto CEVA, the other blocks took the same time since they are implemented on ARM. The reduction in the IFFT block timing translated to almost 61% reduction in the transmitter chain execution time.

Function	Time(µs)
64-FFT	650
Channel Equalization and Estimation	870
Pilot Removal	5
16 QAM demodulation	646
De-Interleaver	488
Viterbi Decoder	611
Descrambler	5
Total	3275

Table 17. WiFi-Lite B Receiver Implementation on ARM+CEVA: Timing Profile

The profiling results in Table 17 show that for the receiver chain, the channel estimation and equalization, FFT, and decoder blocks took most of the computational time. When compared to the only ARM implementation, decoder, FFT, and channel estimation block execution time reduced significantly. Overall, the execution time of the ARM+CEVA implementation reduced from 11.6ms to 3.2ms.

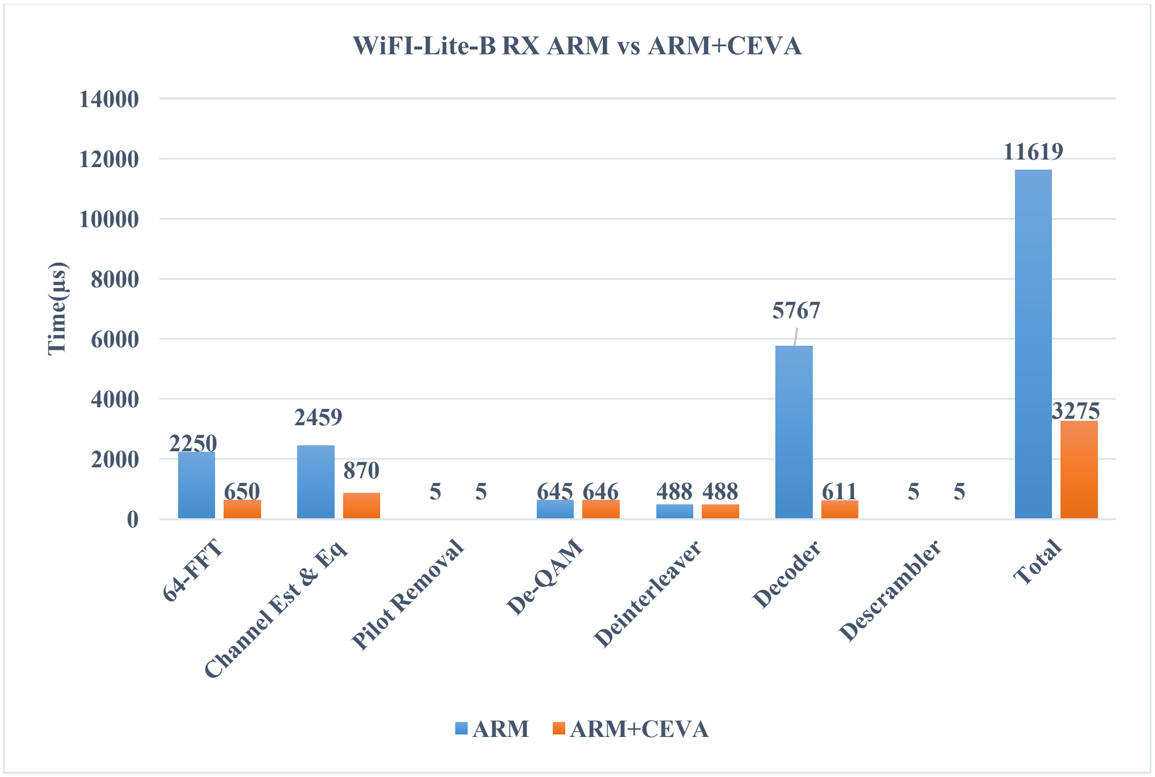

Figure 29. WiFi-Lite-B RX Receiver Time Values - ARM vs ARM+CEVA

Figure 29 compares the execution times of the receiver chain implemented using only ARM with that using ARM+CEVA. The results show that the FFT block and decoder took significantly lower time on CEVA DSP than on ARM. The channel estimation and equalization block also took less time in ARM+CEVA implementation, since the FFT used in the channel estimation is implemented on the CEVA DSP. The timing for the other blocks in the receiver chain remain same since they are implemented on ARM in both cases. The total execution time of the receiver chain for ARM+CEVA implementation is reduced by almost 71% compared to the receiver chain execution time for only ARM implementation.

4.4 WiFi-Lite-B Fixed-Point Implementation on CEVA SDK

Next, all blocks of the transmitter and receiver chains were implemented on the CEVA SDK. Figure 30 describe the setup. Transmitter chain output data is saved to a file, the channel model (Rayleigh fading) corrupts the data and saves the data to a file. The receiver chain reads the data from a file and decodes the payload. All processing is done in CEVA SDK and so the transmitter chain does not include the preamble block.

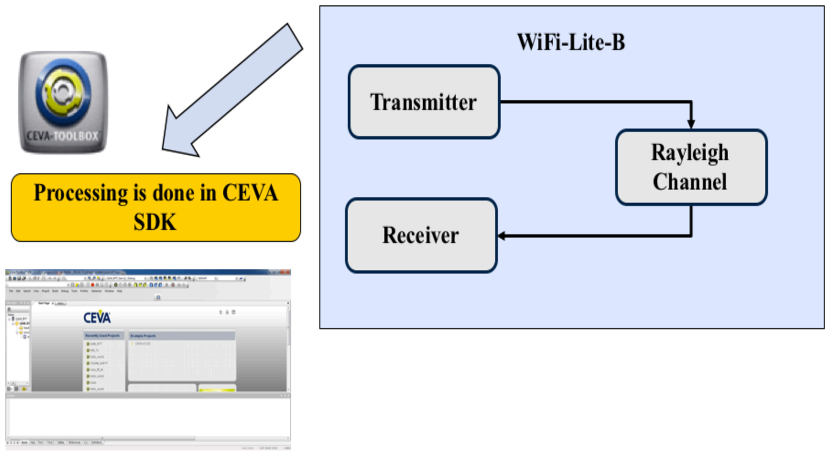

Figure 30. WiFi-Lite-B CEVA SDK Implementation

4.4.1 Experimental Results

The protocol was implemented on the CEVA SDK and the execution time was calculated for each block. Tables 18 and 19 include the timing results of each block in the transmitter and receiver, respectively.

Function	Cycles	Time(µs)
Scrambler	694	0.9
Encoder	4577	6.1
Interleaver	1286	1.7
QAM Modulation	878	1.1
Pilot Insertion	497	0.6
IFFT	70	0.09
Total	8002	10.5

Table 18. WiFi-Lite-B Transmitter Implementation on CEVA SDK: Timing Profile

From Table 18, the two computationally expensive operations in the transmitter chain are the encoder which took 6.1µs and interleaver which took 1.7µs. Together they account for 74% of the total execution time. The IFFT block took only 0.09µs which is significantly lower compared to the ARM+CEVA implementation.

Function	Cycles	Time(µs)
FFT	70	0.09
Channel Estimation	42570	56.7
Pilot removal	932	1.2
QAM Demodulation	60	0.08
De-Interleaver	11322	15.
Viterbi Decoder	7048	9.3
De-Scrambler	274	0.3
Total	62276	82.6

Table 19. WiFi-Lite-B Receiver Implementation on CEVA SDK

The profiler results of the receiver chain presented in the Table 19 show that channel estimation and equalization algorithm took 56.7µs out of the total 82.6µs. While this is very large, the CEVA SDK implementation takes significantly less time compared to the ARM+CEVA implementation (see Table 15). Also, FFT, QAM Demodulation, and decoder blocks only took

0.09μs, 0.08μs, and 9.3μs respectively, which is very low when compared to the ARM+CEVA implementation results.

Function	Time(μs) ARM+CEVA	Time(μs) CEVA
Scrambler	5	0.9
Encoder	41	6.1
Interleaver	6	1.7
16-QAM Modulation	8	1.1
Pilot Insertion	3	0.6
64-IFFT	620	0.09
Total	683	10.5

Table 20. WiFi-Lite-B Transmitter Profiler Results - ARM+CEVA vs CEVA SDK

Table 20 compares the execution times of a transmitter chain implemented using ARM+CEVA and CEVA SDK. The results show that the scrambler, encoder, interleaver, QPSK modulation and pilot insertion blocks took much lower time on CEVA SDK when compared to ARM+CEVA implementation where these blocks are implemented on ARM. The IFFT block took significantly lower time on CEVA SDK when compared to ARM+CEVA implementation. While the FFT block was implemented on CEVA in both implementations, the spectacular reduction in time is because the SDK doesn't exactly emulate the hardware. For instance, it does not take into account memory size or latency. The total execution time of a transmitter chain for CEVA SDK implementation is reduced by almost 98% compared to the transmitter chain execution for ARM+CEVA implementation.

Function	Time(μs) ARM+CEVA	Time(μs) CEVA
64-FFT	650	0.09
Channel Estimation & Equalization	870	56.7
Pilot Removal	5	1.2
16-QAM Demodulation	646	0.08
De-Interleaver	488	15
Decoder	611	9.3
De-Scrambler	5	0.3
Total	3275	82.6

Table 21. WiFi-Lite-B Receiver Profiler Results - ARM+CEVA vs CEVA SDK

Table 21 compares the execution times of a receiver chain implemented using ARM+CEVA and CEVA SDK. The results show that the pilot removal, de-interleaver, de-scrambler and channel estimation & equalization blocks took much lower time on CEVA SDK compared to ARM+CEVA implementation where these blocks are implemented on ARM. The IFFT, decoder and QPSK demodulation blocks took significantly lower time on CEVA SDK when compared to ARM+CEVA implementation. While these blocks are implemented on CEVA in both implementations, reduction in time is because the SDK doesn't exactly emulate the hardware. For instance, it does not take into account memory size or latency. The total execution time of a receiver chain for CEVA SDK implementation is reduced by almost 97.4% compared to the receiver chain execution for CEVA+ARM implementation. Overall, there is a 30x - 80x reduction in the execution time achieved by running the complete receiver and transmitter chain on the CEVA DSK.

4.5 Implementation Using Intel T2200 for TX and RX – Real Time Processing

Two Intel T2200 WBS boards are used: one as a transmitter, and the other as a receiver to communicate data over the air using the same RF configuration as WiFi-Lite-A protocol (refer to Figure 20). The processing on the T2200 boards was done using only ARM.

4.5.1 Experimental Results

Figure 31 shows the experimental setup. A fixed string of length 424 bytes corresponding to one OFDM frame (136 bits per OFDM symbol) is sent from one board to another and vice versa.

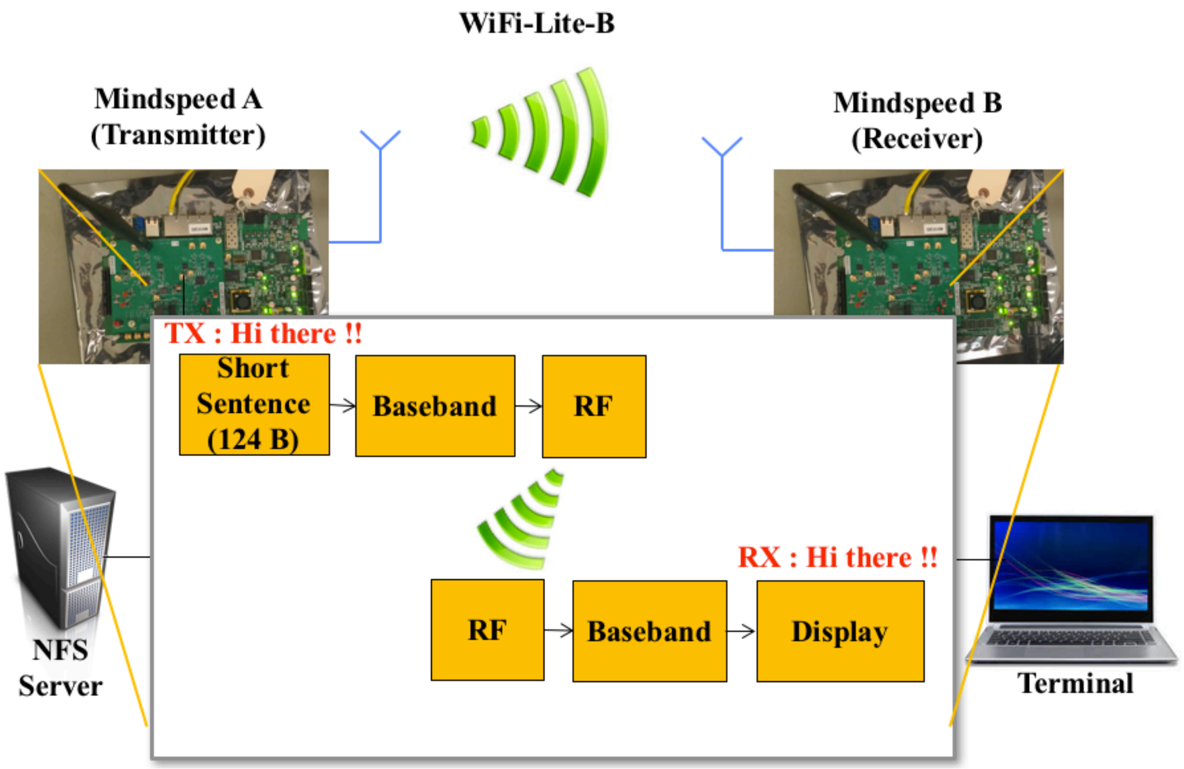

Figure 31. WiFi-Lite-B Over the Air Implementation - Intel T2200 WBS Boards

The data (424byte) was decoded successfully with zero BER. Our receiver algorithms and RF configurations worked perfectly. Figure 32 shows the constellation diagram of the symbols before the 16-QAM demodulation block, the diagram shows that the channel estimation and equalization algorithm removed the channel effect in the OFDM symbol.

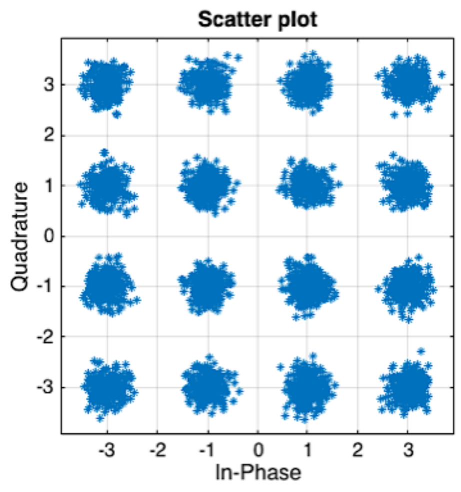

Figure 32. WiFi-Lite-B – 16 QAM Constellation Diagram at the Receiver

5 SCFDM-Lite

In this chapter, the implementations of the SCFDM-Lite protocol on ARM and CEVA DSP platforms are described. An overview of the protocol is given in Section 5.1, followed by protocol implementation on ARM in Section 5.2, and protocol implementation on ARM+CEVA in Section 5.3.

5.1 Protocol Overview

SCFDM-Lite is an Orthogonal Frequency Division Multiplexing (OFDM) based system with a DFT mapper, which utilizes single carrier modulation (SC), DFT-spread orthogonal frequency multiplexing, and frequency domain equalization. Our SCFDM-Lite system uses 64 DFT subcarriers and 128 FFT subcarriers that are modulated by quadrature phase shift keying (QPSK), and forward error correction (FEC) coding (Convolutional coding) with a coding rate of ½. Our protocol is implementation is based on the LTE Uplink standard.

5.1.1 Transmitter

At the transmitter, all the data bits undergo the following PHY layer signal processing. Data bits are scrambled by a linear non-additive scrambler using LFSR and then encoded by the convolutional encoder with a constraint length of 7 and code rate of ½. The bits are then interleaved by the linear random interleaver using LFSR, followed by QPSK modulation, pilot insertion, DFT of width 64, subcarrier mapping, IFFT of width 128, and then by a cyclic prefix addition (please refer to 2.3.1.1 and 2.3.2.1sections for transmitter block description).

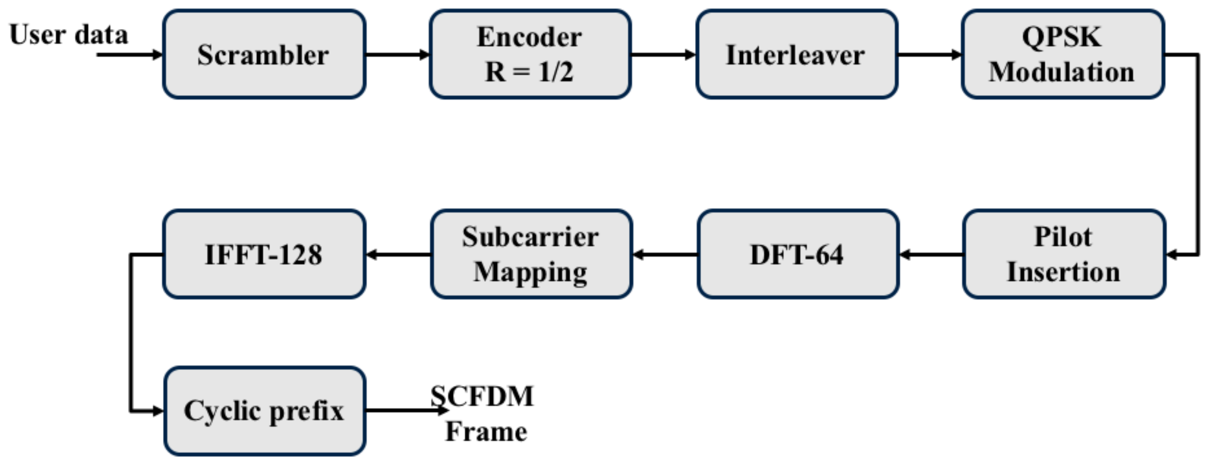

Figure 33. SCFDM-Lite Protocol: Block Diagram of Transmitter

Protocol Configuration	Values
No of input data bits per OFDM Symbol	40
No of data subcarriers per OFDM Symbol	48
No of pilot subcarriers	16
FFT/IFFT width DFT/IDFT width	128 64
Convolutional Encoder	(2,1,8) Mask0 – 0133 Mask1 – 0171 Mask2 – 0165
Modulation	QPSK
Cyclic Prefix	16 symbols

Table 22 SCFDM-Lite Protocol Configuration

5.1.2 Receiver

At the receiver, the cyclic prefix is removed, and the OFDM frame is given to the FFT-128 unit for converting the symbols from time domain to frequency domain. This is followed by subcarrier de-mapping and IDFT-64/IFFT-64 unit for converting the symbols from frequency domain to time domain. Then the symbols are given to channel estimation and equalization to estimate and remove the channel effect. Next pilots are removed from the frame and sent to QPSK demodulator to extract the bits from the symbols. The de-interleaved data is given to the Viterbi decoder to remove the redundant bits and to rectify the errors bits. The bits are then descrambled to get the exact payload.

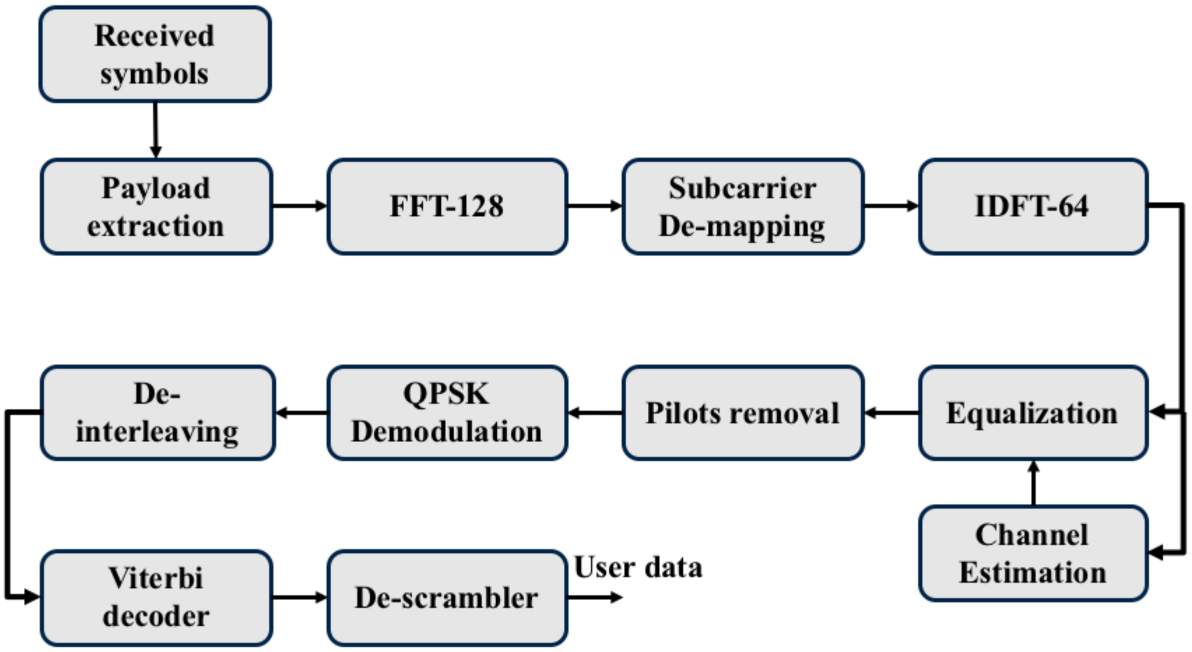

Figure 34. SCFDM-Lite Protocol: Block Diagram of Receiver

5.2 SCFDM-Lite Implementation on ARM

First, the transmitter and receiver chains were implemented on ARM using the C language. Figure 35 describes the experimental setup. The input to the transmitter chain is a sequence of 40 bits. These are converted into OFDM symbols. Transmitter chain output data is saved to a file, the channel model (Rayleigh fading) corrupts the data and saves the data to another file. The receiver chain reads the data from this file and decodes the payload. In our implementation the receiver chain successfully decodes the OFDM symbol with zero bit error rate.

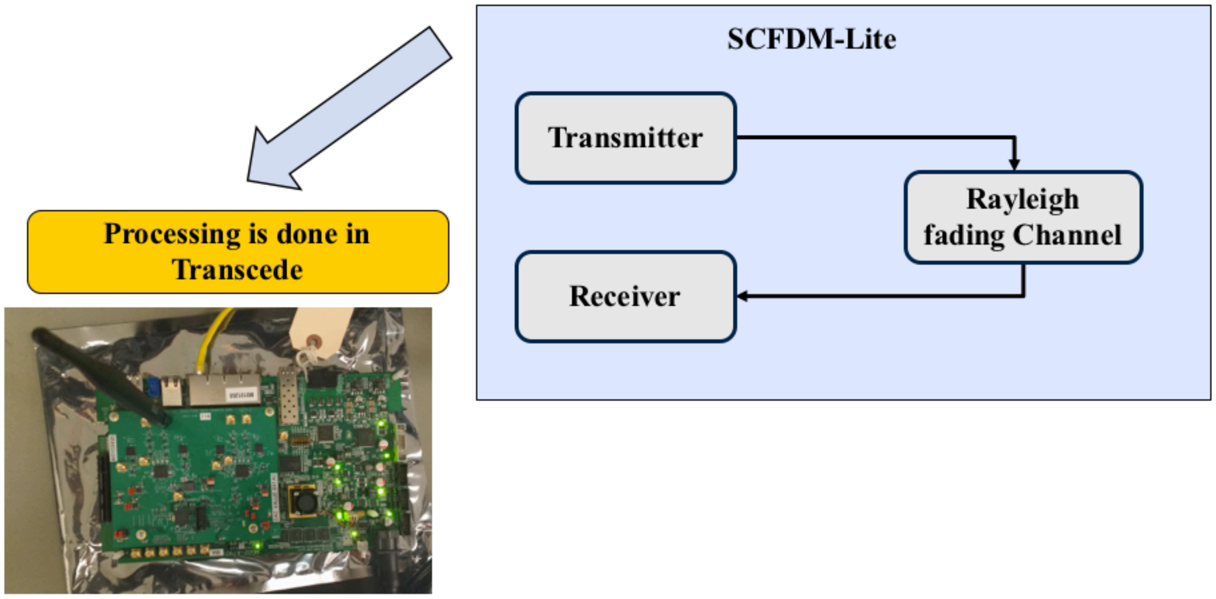

Figure 35. SCFDM-Lite Protocol Implementation - TX and RX Loopback

5.2.1 Experimental Results

The timing performance of the different blocks in transmitter and receiver was evaluated. Table 23 describes the execution time of the major functional blocks for the transmitter. The transmitter chain took 7ms with a 93% of the time being spent on computing the 128 point IFFT and 64 point FFT/DFT.

Function	Time(μs)
Scrambler	3
Encoder	15
Interleaver	4
QPSK Mod	10
Pilot Insert	4
64-DFT	2329
128-IFFT	4673
Cyclic Prefix	7
Total	7045

Table 23. SCFDM-Lite Transmitter Implementation on ARM: Timing Profile

Function	Time(µs)
128-FFT	4779
64-IDFT	2148
CH EST & EQ	2165
Pilot Rem	5
QPSK Demodulation	465
De-Interleaver	16
Decoder	1636
Descrambler	2
Total	11216

Table 24. SCFDM-Lite Receiver Implementation on ARM: Timing Profile

Table 24 lists the execution time of the major functional blocks of the receiver chain. The results show that channel estimation and equalization, FFT, and decoder blocks take most of the computational time. These three blocks took 10.7ms, which is 95% of the total time.

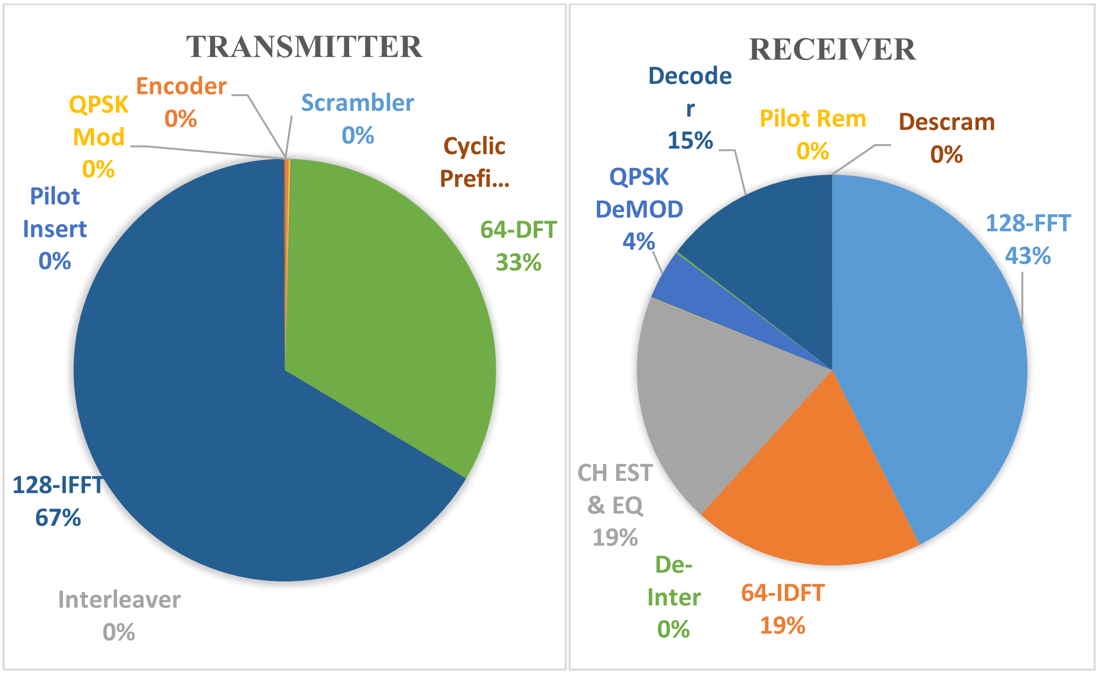

Figure 36. SCFDM-Lite Profiling Results for ARM Implementation

Figure 36 shows the pie chart of the transmitter and receiver timing profiles. In the transmitter, IFFT took 67% and FFT/DFT took 33% of the total execution time. In the receiver, channel estimation and equalization block took 19%, FFT took 43%, IFFT/IDFT took 19%, and

Viterbi decoder took 15% of the execution time. Since the FFT/DFT, IFFT/IDFT and decoder algorithms took most of the processing time, these algorithms were mapped onto the CEVA DSP.

5.3 SCFDM-Lite Implementation on the ARM and CEVA DSP

In order to map the IFFT, FFT, and Viterbi decoding algorithms onto the CEVA DSP, and map the remaining algorithms onto ARM, the interface between the ARM-CEVA interface was utilized (refer to Section 3.3.1 for ARM-CEVA interface details).

5.3.1 Experimental Results

The experimental setup is the same as shown in Figure 11. A sequence of 40 bits is processed by the transmitter chain, corrupted by Rayleigh fading channel and decoded successfully by the receiver chain. DSP and communication libraries (C & ASM) provided by CEVA are used to improve baseband timing performance. Specifically, the FFT and Viterbi decoder libraries (ASM) were mapped onto CEVA DSP. This mapping resulted in significant reduction in the computational time of both the transmitter and receiver chains. The transmitter chain took 1.2ms compared to 7ms when implemented on only ARM and the receiver chain took 4.3ms compared to 11.2ms when implemented on only ARM.

Function	Time(µs)
Scrambler	3
Encoder	15
Interleaver	5
QPSK Mod	12
Pilot Insert	4
64-DFT	585
128-IFFT	608
Cyclic Prefix	10
Total	1242

Table 25. SCFDM-Lite Transmitter Implementation on ARM+CEVA: Timing Profile

The profiling results in Table 25 show that the IFFT and FFT/DFT blocks took most of the transmitter chain computational time. The IFFT and FFT/DFT blocks execution time is reduced to almost 86% and 72.7%, respectively, when compared to the only ARM implementation (Table 20).

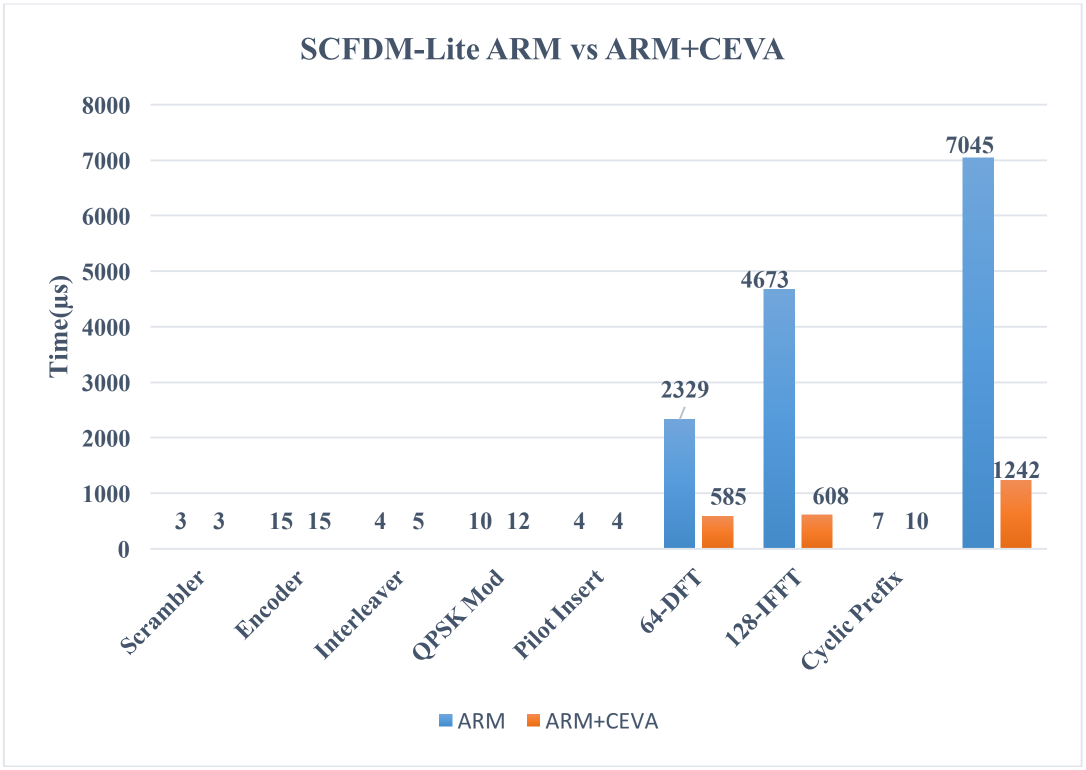

Figure 37. SCFDM-Lite TX Execution Time Values - ARM vs ARM+CEVA

Figure 37 compares the execution times of the transmitter chain implemented using ARM with that using ARM+CEVA. Other than the IFFT and FFT/DFT blocks which were mapped onto CEVA, the other blocks took the same time since they are implemented on ARM. The reduction in the IFFT block timing translated to almost 82% reduction in the transmitter chain execution time.

Function	Time(µSeconds)
128-FFT	563
64-IDFT	517
CH EST & EQ	2218
Pilot Rem	5
QPSK Demodulation	483
De-Interleaver	10
Decoder	589
Descrambler	3
Total	4388

Table 26. SCFDM-Lite Receiver Implementation on ARM+CEVA: Timing Profile

The profiling results in Table 26 show that for the receiver chain, the channel estimation and equalization block took most of the computational time. When compared to the only ARM implementation, FFT, IFFT/IDFT and decoder blocks execution time reduced significantly. Overall, the execution time of the ARM+CEVA implementation reduced from 11.2ms to 4.3ms.

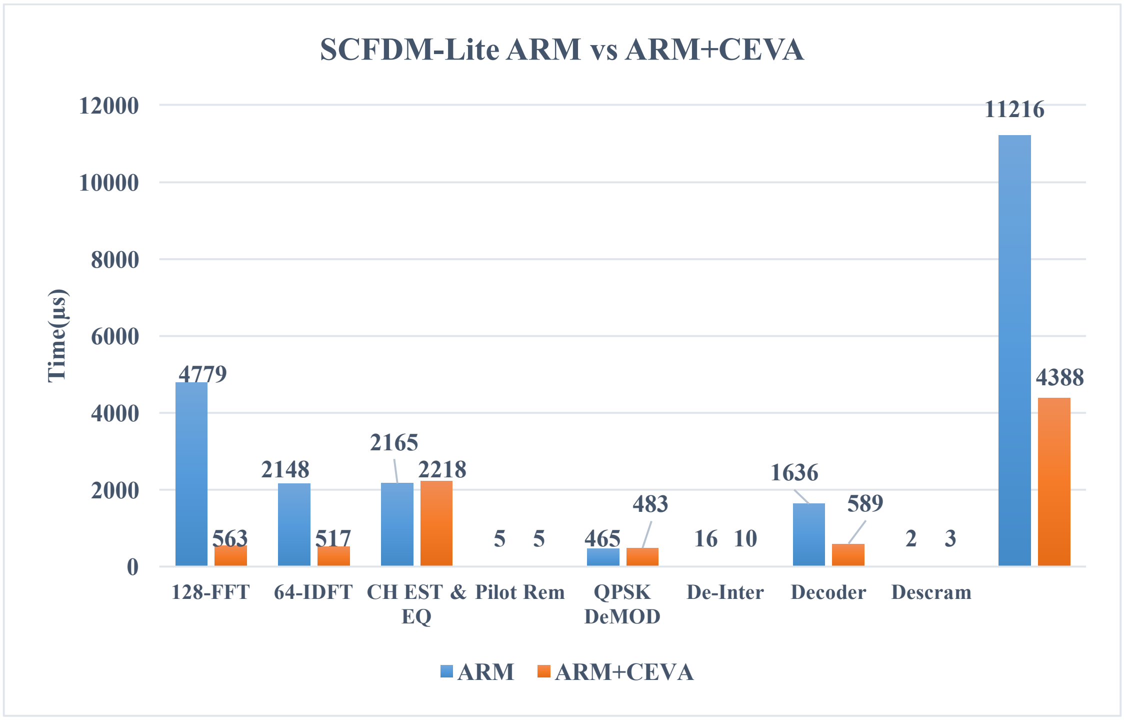

Figure 38. SCFDM-Lite RX Execution Time Values - ARM vs ARM+CEVA

Figure 38 compares the execution times of the receiver chain implemented using ARM with that using ARM+CEVA. The results show that the FFT block, IFFT/IDFT block and decoder took significantly lower time on CEVA DSP than on ARM. The other blocks values in the receiver chain remain same since they are implemented on ARM. The total execution time of a receiver chain for ARM+CEVA implementation reduced by almost 60% compared to the execution time for only ARM implementation.

6 SC-Lite

In this chapter, the implementation of the SC-Lite protocol on ARM platform is described. An overview of the protocol is given in Section 6.1, and protocol implementation on ARM in given in Section 6.2.

6.1 Protocol Overview

SC-Lite is a Single Carrier based system. Our SC-Lite system uses Reed Solomon encoder to encode the bits and uses BPSK modulation to map the bits to symbols.

6.1.1 Transmitter

At the transmitter, all the data bits undergo the following PHY layer signal processing. Data bits are scrambled by a linear non-additive scrambler using LFSR, encoded by the convolutional encoder with a (132,100) configuration. The bits are then modulated by a BPSK modulator (please refer to 2.3.3.1 section for transmitter blocks description). Figure 39 describe the block diagram of the SC-Lite transmitter.

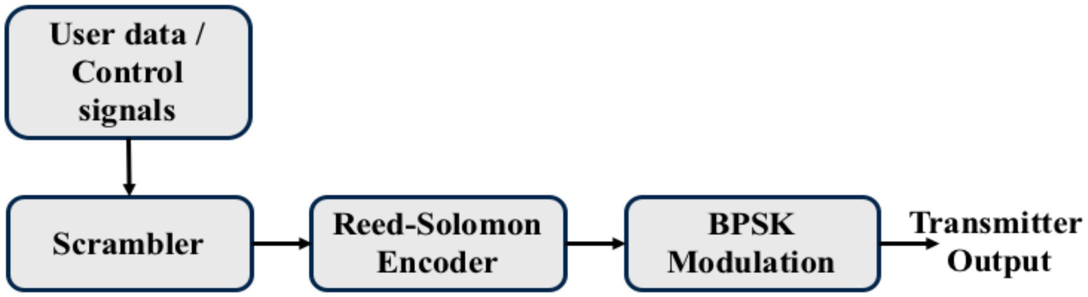

Figure 39. SC-Lite Protocol: Block Diagram of Transmitter

6.1.2 Receiver

At the receiver, the symbols are given to BPSK demodulator to extract the bits from the symbols. The data bits are given to the Reed-Solomon decoder to remove the redundant bits and to rectify the errors bits. The bits are then descrambled to get the exact payload (please refer to 2.3.3.2 section for receiver block description). Figure 40 describes the block diagram of the SC-Lite receiver.

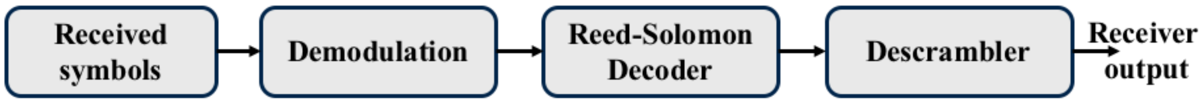

Figure 40. SC-Lite Protocol: Block Diagram of Receiver

6.2 SC-Lite Implementation on ARM

The protocol is developed using the C language and implemented on ARM in the Intel T2200 WBS board. Figure 41 describes the experimental setup. The input to the transmitter chain is a sequence of 40 bits. These are converted into SC symbols. Transmitter chain output data is saved to a file, the channel model (AWGN) corrupts the data and saves the data to a file. The receiver chain reads the data from a file and decodes the payload. Receiver chain successfully decodes the symbol with zero bit error rate.

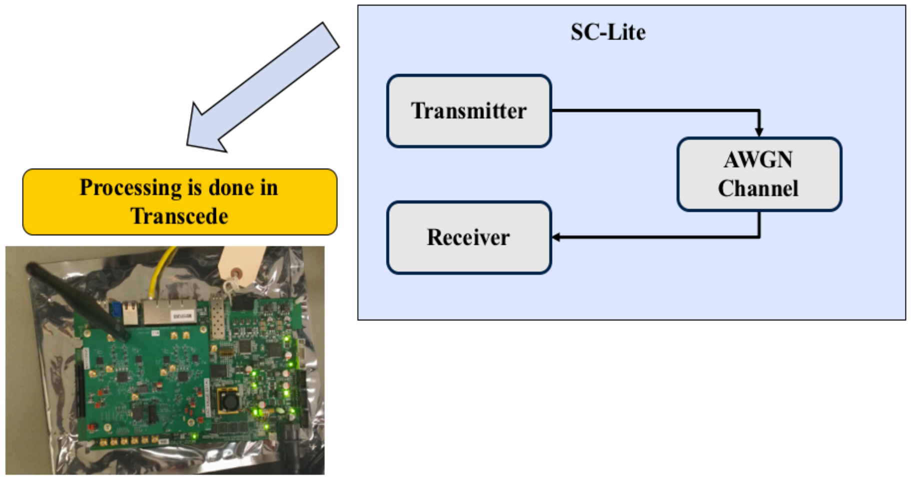

Figure 41. SC-Lite Protocol Implementation – Experimental Setup

6.2.1 Experimental Results

The timing performance of the different blocks in transmitter and receiver is evaluated. Table 27 describes the execution time of the major functional blocks for the transmitter. The transmitter chain took 37μs with 81% of the time being spent on the encoder.

Function	Time(μs)
Scrambler	3
Reed Solomon Encoder	30
BPSK Modulation	4
Total	37

Table 27. SC-Lite Transmitter Profiler Results

Function	Time(μs)
BPSK Demodulation	10
RS Decoder	40
De-scrambler	3
Total	53

Table 28. SC-Lite Receiver Profiler Results

Table 28 lists the execution time of the major functional blocks of the receiver chain. The results show that the receiver chain took 53μs, with 75% of the time being spent on the RS decoder.

7 CONCLUSIONS

7.1 Summary

This thesis focused on developing and implementing a wide variety of wireless protocols on a heterogeneous computing platform. Specially, OFDM-based protocols such as WiFi-Lite-A, WiFi-Lite-B, a DFT-spread OFDM based protocol such as SCFDM-Lite, and a single carrier based protocol such as SC-Lite, were mapped onto Intel T2200 board. The transmitter and receiver blocks of all the protocols were first mapped onto ARM. The IFFT, FFT, and Viterbi decoder blocks took most of the computational time and so, next, IFFT, FFT, and Viterbi decoder blocks were mapped onto CEVA DSP and the remaining blocks were mapped onto ARM. The timing results of CEVA+ARM implementation showed 60%, 64%, and 71.5% savings compared to an only-ARM implementation for WiFi-Lite-A, WiFi-Lite-B, and SCFDM-Lite protocols, respectively. Such a reduction was due to mapping the time intensive computational units on CEVA DSP hardware. Further savings in baseband processing was achieved by implementing all the transmitter and receiver blocks on CEVA DSP. The timing results showed almost 90% savings, for WiFi-Lite-A protocol. Such large savings were due to reduction in the context switching and memory overhead since the interaction between ARM and CEVA DSP was minimized. Finally, over-the-air transmission communication was demonstrated for WiFi-Lite-A and WiFi-Lite-B protocols using Intel T2200 WBS boards for both the transmitter and receiver.

7.2 Future Work

In the near future, the WiFi-Lite-B transmitter and receiver blocks will be mapped onto CEVA DSP hardware. The challenges include:

i. Integrating the necessary source files and linking the libraries into a single executable file, which will be used to boot the CEVA DSP.

ii. Code and data section address alignment using the linker file. CEVA SDK doesn't align the address as needed by XC323 DSP, so external linker file should be used to align the address for the code and data sections.

iii. Conversions of floating point algorithms to fixed point algorithms of transmitter and receiver blocks.

In the long term, the goal is to study other communication (standard as well as non-standard) protocols and map them onto the CEVA DSP hardware. An important step is to reduce the computation time of the transmitter and receiver chain. This can be done by implementing the computationally expensive kernels such as channel estimation, and frame detection algorithm in VEC-C language (CEVA DSP specific Vector C language). Without this step, multiple packets cannot be processed in real time.

REFERENCES

3GPP a global initiative. (2004, Janurary 10). *The Mobile Broband Standard LTE.* Retrieved July 1, 2016, from 3GPP : http://www.3gpp.org/technologies/keywords-acronyms/98-lte

Akyildiz, I., Lee, W., Vuran, M., & Mohanty, S. (2006). NeXt generation/dynamic spectrum access/cognitive radio wireless networks: a survey. *Computer networks, 50(13)*, 2127-2159.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4), 2347-2376.

Arslan, H. (. (2007). *Cognitive radio, software defined radio, and adaptive wireless systems (Vol. 10).* Berlin: Springer.

Berkeley Design Technology, Inc. (2012, Feburary 23). *Picochip and Mindspeed: Former Competitors Unite to Address Wireless Spectrum Needs*. Retrieved June 24, 2016, from An inside look at DSP Technology: http://www.bdti.com/InsideDSP/2012/02/23/Mindspeed

Bliss, D. W., & Govindasamy, S. (2013). *Adaptive Wireless Communications: MIMO Channels and Networks.* Cambridge University Press.

Bloessl, B., Segata, M., Sommer, C., & Dressler, F. (2013). An IEEE 802.11 a/g/p OFDM Receiver for GNU Radio. *In Proceedings of the second workshop on Software radio implementation forum (pp. 9-16). ACM.*

Bluethgen, H., Grassmann, C., Raab, W., Ramacher, U., & Hausner, J. (2004). A programmable baseband platform for software defined radio. A programmable baseband platform for software-defined radio. In *Proceedings of SDR FORUM.*

Cai, X., & Giannakis, G. B. (2004). Error Probability Minimizing Pilots for OFDM With M-PSK Modulation Over Rayleigh-Fading Channels. *IEEE Transactions on Vehicular Technology*, *53*(1), 146-155.

CEVA DSP. (2012, January 20). *CEVA-XC323.* Retrieved June 24, 2016, from ceva-dsp: http://www.ceva-dsp.com/CEVA-XC323

Grayver, E. (. (2012). *Implementing software defined radio.* Springer Science & Business Media.

Gupta, U., Korrapati, S., Matturu, N., & Ogras, U. Y. (2016). A generic energy optimization framework for heterogeneous platforms using scaling models. *Microprocessors and Microsystems, 40, 74-87.*

Heiskala, J., & Terry Ph D, J. (2001). *OFDM wireless LANs: A theoretical and practical guide*. Sams.

IEEE Standards association. (2012). *802.11: Wireless LANs.* Retrieved from standards.ieee.org: http://standards.ieee.org/about/get/802/802.11.html

Intel . (2014, January 10). *Leading-Edge SMALL CELL Solutions .* Retrieved June 24, 2016, from Transcede product family brief : http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/transcede-product-family-brief.pdf

Intel Corp. (n.d.). *LTE / Dual-Mode Femtocell SoC.* Retrieved from http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/transcede-product-family-brief.pdf

Kelley, B. (2009, October). Software defined radio for broadband OFDM protocols. *SMC 2009. IEEE International Conference* on *Systems, Man and Cybernetics,* pp. 2309-2314.

Lee, H. (2007). *A baseband processor for software defined radio terminals.* (Doctoral Dissertation, University of Michigan).

Lee, H., Chakrabarti, C., & Mudge, T. (2010). A low-power DSP for wireless communications. *IEEE transactions on very large scale integration (VLSI) systems*, 18(9), 1310-1322.

Lin, S., & Costello, D. J. (2004). *Error Control Coding: Fundamentals and Applications.* India: Pearson Eduation.

Woh, M., Lin, Y., Seo, S., Mahlke, S., Mudge, T., Chakrabarti, C., ... & Flautner, K. (2008, November). From SODA to scotch: The evolution of a wireless baseband processor. In *2008 41st IEEE/ACM International Symposium on Microarchitecture* (pp. 152-163). IEEE.

Lin, Y., Lee, H., Woh, M., Harel, Y., Mahlke, S., Mudge, T., & Flautner, K. (2007). SODA: A high-performance DSP architecture for software-defined radio. *IEEE MICRO*, 27(1), 114-123.

Minsky, H. (2010, Jan 10). *rscode* . Retrieved 2016, from rscode sourceforge: http://rscode.sourceforge.net

Parhi, K. K. (2001). *Digital Signal Processing for Mutlimedia Systems.* Minnesota, Minnesota, USA: Marcel Dekker.

Perahia, E., & Stacey, R. (2013). *Next Generation Wireless LANS: 802.11 n and 802.11 ac.* Cambridge University press.

Proakis, J., & Masoud, S. (2007). *Digital Communications.* San Diego, California, USA: McGraw-Hill Education.

Rader, C. M., & Brenner, N. M. (1976). A New Principle for Fast Fourier Transformation. *IEEE Acoustics, Speech & Signal Processing*, 24(3), 264-266.

Ramacher, U. (2007). Software-defined radio prospects for multistandard mobile phones. *IEEE Computer*, 40(10), 62-69.

SDR forum. (n.d.). *SDRF Congnitive Radio Definitions* . Retrieved from sdrforum: http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-06-R-0011-V1_0_0.pdf

Singh, D., Tripathi, G., & Jara, A. J. (2014, March). A survey of Internet-of-Things: Future vision, architecture, challenges and services*, 2014 IEEE World Forum on Internet of things (WF-IoT), pp. 287-292.*

Ulversoy, T. (2010). Software Defined Radio: Challenges and Opportunities. *IEEE Communications Surveys & Tutorials*, 12(4), 531-550.

Unnikrishnan, S., Surve, S., & Bhoir, D. (Eds.). (2011). *International Conference on Advances in Computing, Communication and Control, ICAC3 2011, Mumbai, India, January 28-29, 2011.* Springer.

Viterbi, A. J. (1967). Error Bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, *13*(2), 260-269.