PERFORMANCE AND HARDWARE COMPLEXITY ANALYSIS OF PROGRAMMABLE RADIO PLATFORM FOR MIMO OFDM BASED WLAN SYSTEMS

BY VIJAYANT BHATNAGAR

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Professor Predrag Spasojevic

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2008

ABSTRACT OF THE THESIS

Performance and Hardware Complexity Analysis of Programmable Radio Platform for MIMO OFDM based WLAN Systems

by Vijayant Bhatnagar

Thesis Director: Professor Predrag Spasojevic

Emerging wireless technologies and standards present a design space with multiple dimensions in terms of time, physical hardware space, and technology trends. Efficient evaluation of a desired combination of these dimensions to support multiple technologies and standards presents a significant challenge. We study the feasibility of a multiprotocol architecture without sacrificing the Quality of Service. An architecture facilitating such a mechanism can be implemented at different layers in the network stack with each layer offering a tradeoff between complexity and latency. Careful analysis of the physical layer reveals that most blocks of the transceiver can be reused for different protocols without significant architectural change. In addition to the feasibility analysis, we also identify common blocks in the network stack that could be possibly reused buying us significant hardware gains without sacrificing the aggregate system throughput. Our study presents the gate count complexity and the performance analysis of programmable radio architecture with the 802.11n (Draft 3.0) MIMO-OFDM based protocol stream and 802.11a OFDM based WLAN protocol stream. In this thesis, we demonstrate that multiple protocols can be supported using the same hardware under acceptable latency requirements. Complexity of the system in terms of gate count has been determined. It has been found that for shorter frame sizes, it is better to process less number of OFDM symbols at a time. However, for larger frame sizes, it is beneficial to process large number (four to eight) of

OFDM symbols at a time. Also, the minimum clock rate required to run the hardware, would vary depending upon the number of OFDM Symbols processed. The switching and multiplexing overhead of the programmable radio platform has also been investigated. Finally, our simulator is capable of evaluating bottlenecks, if any.

Acknowledgements

I would like to express my deep and sincere gratitude to my advisor, Dr. Predrag Spasojevic, Professor, Rutgers University and Prof. Zoran Miljanic, Visiting Professor, WINLAB. It was an unforgettable experience working on this topic under their supervision. Their understanding, encouragement and personal guidance has provided a strong base for the present thesis.

I am deeply grateful to my supervisor, Ms. Renu Rajnarayan and Mr. Khanh Le, Research Assistant, WINLAB, for their helpful and detailed feedback. I am really thankful for their immense support throughout this work. This thesis would not have been possible without their guidance.

I wish to express warm and sincere thanks to Dr. Dragan Samardzija for clearing my concepts on MIMO Technology. His extensive background in MIMO Research helped me to analyze this architecture from practical viewpoint. His extensive feedback on my work helped me to further explore the topic and come up with insightful results by doing simulations.

I wish to express my warm and sincere thanks to my wonderful friend Gautam Bhanage for his support, encouragement and guidance throughout my days at WINLAB.

I enjoyed the company and support of Tathagata Ray and WINLAB students including Sankar Subramaniam Salvady, Rajesh Mahindra, Tejaswy Hari and Zhibin Wu.

Dedication

This work is dedicated to my parents for their inspiration and motivation throughout my life.

"Whatever you do, whatever you eat, whatever you offer or give away, and whatever austerities you perform do that, O son of Kunti, as an offering to Me." - Shrimad BhagwadGita (Chapter 9,Verse 29) [1]

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BCC	Binary Convolutional Codes
CCA	Clear Channel Assessment
CLB	Configurable Logic Block
CP	Command Processor
CPU	Central Processing Unit
CDD	Cyclic Delay Diversity
CRC	Cyclic Redundancy Check
CSD	Cyclic Shift Diversity
FDG	Frame Delimiter Generator
FFT	Fast Fourier Transform
FPGA	Field-programmable Gate Array
FU	Functional Unit
GI	Guard Interval
HT	High Throughput
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LAN	Local Area Network
LDPC	Low Density Parity Check Codes
LTF	Long Training Field
MAC	Medium Access Control
MCS	Modulation And Coding Scheme
MIMO-OFDM	Multiple Input Multiple Output-Orthogonal Frequency Division Multiplexing
MMSE	Minimum Mean Squared Error
OFDM	Orthogonal Frequency Division Multiplexing
PE	Processing Engine
PLCP	Physical Layer Convergence Procedure
PPDU	PLCP Protocol Data Unit
PHY	Physical Layer
PU	Processing Unit
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
QBPSK	Quadrature Binary Phase Shift Keying
RMAP	Register MAP
STBC	Space Time Block Codes
STF	Short Training Field
TSP	Task Spawning Processor
UCM	Unit Control Module
WiNC2R	WINLAB Cognitive Radio Platform
WINLAB	Wireless Information Network Laboratory, Rutgers University.
WLAN	Wireless Local Area Networks

Chapter 1

Introduction

1.1 Motivation

For the present WLAN deployment and emerging 4G Communications standards, there is a need to build an infrastructure with different PHY, MAC, and Network layer technologies and that is capable of operating in different spectrum bands. Such an infrastructure necessitates a common architecture (*WiNC2R - WINLAB Network-centric Cognitive Radio Hardware Platform* [3]) for multi-protocol processing. One of the biggest advantages of this architecture would be to minimize the design effort and leave the matter of reconfigurability in the hands of the user. We need to allow some degree of flexibility for different radio specifications which would help in upgrading the architecture to meet new radio specifications with minimal effort and less amount of replacements. We have to explore the radio specifications that can benefit from this common architecture.

In essence, this architecture would require flexible physical and higher layer network processing and adaptability for per packet protocol switching. The flexibility, while an important asset, needs to be added with the marginal compromise in cost and performance in comparison to a fixed function hardware solution. This architecture would be an enormous tool for identifying and evaluating appropriate wireless technologies for the mobile devices. This architecture would also help in identifying pros and cons of the hardware vs. software implementation trade-off and understanding the limitations of the programmable CPU architecture in addressing the needs of emerging wireless technologies.

1.2 General Statement of the Problem

WiNC2R allows flexible processing in a specific application domain. Most of the key elements of the FPGA based architecture for a 802.11a receiver have been designed prior to this work [4] receiver has already been done [3]. The main problem was to reuse physical layer components designed for 802.11a [4] on WiNC2R and extend it to incorporate MIMO-OFDM based WLAN systems (*802.11n [5] compliant*). 802.11n protocol standard was chosen as a reference point for analysis, design and evaluation of MIMO-OFDM based WLAN systems since it is the next generation MIMO WLAN standard. Problems addressed by this thesis can be broadly divided into five major sub-problems:

1. There was a need to support all the features mandated by the standard. Hence, there was a need to explore all the PHY layer transmission and reception *minimum requirements* of the overall system.

2. Some of the PEs (PE is a group of communication blocks like interleaver, encoder etc. done on the basis of localized communication pattern and related functionality e.g. Encoder PE is a combination of encoder and interleaver) in the transmitter and receiver needed to be redesigned in order to be able to support MIMO-OFDM systems. This was accompanied by exploring different possible architectures for each PE including the Modulator, Encoder, FFT etc. The ensuing latency and complexity for each of the blocks had to be analyzed. The architectural tradeoffs and limitations had to be known for a given specification.

3. Some of the PEs were added because of the need for more sophisticated PHY layer transmission and reception techniques like cyclic delay diversity based channel estimation and MMSE based MIMO Detection.

4. All the command specifications designed for 802.11a compliant architecture had to be modified for improved utilization and, more generally, for upgrading the architecture.

5. Last but not the least, the overall architectural performance was simulated using MATLAB [1]. The simulation resuls are interpreted in terms of

   - Control Overhead Clock cycles
   - Optimal chunksize
   - Performance Factor (α)
   - PE Utilization

   This interpretation has been done for a given throughput specifications. Also, analysis had to be done for viewing system performance while supporting multi-protocol streams like 802.11a and 802.11n [5].

1.3 Prior Work

Prior work consisted of OFDM based WLAN implementation on WiNC2R architecture. This work was taken as a baseline for designing MIMO-OFDM based WLAN systems. It is assumed that UCM has enough capabilities to control processing of physical and higher layer radio protocols. Also, the 802.11a compliant architecture is being built using FPGAs. ASIC based implementation has been assumed as the focus of this work.

1.4 Contribution

The contribution to the thesis has basically been highlighted in section 1.2. They are rested in the following:

- Designing transmitter and receiver for MIMO-OFDM compliant system
- Complexity analysis for the design. We note that InCyte Lite tool [2] was used for getting the complexity of IP Cores for ASIC design.
- Performance analysis of the design for the present architecture.

---

[1] MATLAB is a registered product of MathWorks Inc.

[2] InCyte is a registered trademark of ChipEstimate.com

1.5 Thesis Organization

Rest of the thesis is organized as follows. Chapter 2 presents a detailed picture on the specifications required to be supported for 802.11a and 802.11n based WLAN systems. In Chapter 3, we give an overview of the radio processing platform and the design of PEs at the transmitter and the receiver. Command specifications would also be described for PEs. Chapter 4 explains the simulation strategy following a pipelining mechanism. Chapter 5 describes the simulation results. In Chapter 6, we draw conclusions for the design and some strategies for future work.

Chapter 2

PHY LAYER SPECIFICATION FOR A 802.11N BASED WLAN SYSTEMS

In this chapter we would explore briefly explore some of the characteristics of 802.11n architecture and hardware design individual components of the transceiver.

2.1 Minimum Requirements

Some of the PHY Layer features of 802.11n based systems are :

- LDPC Codes are optional and BCC codes are mandatory.
- GI of 400ns is optional and GI of 800ns is mandatory.
- Other features like beamforming, greenfield format are optional.
- 40 MHz operational bandwidth is optional. 20 MHz bandwidth compliance is mandatory.
- PPDU Format has been classified into Non-HT Format (compliant to legacy devices like 802.11a and 802.11g), HT-Format (no compatibility with legacy devices) and Mixed Format (conforms to both legacy and HT devices).
- Support of 20 MHz Non-HT Format and 20MHz HT Format with one and two spatial streams is mandatory at AP.
- 20 MHz Non-HT Format and 20 MHz HT Format with one spatial stream is mandatory at non-AP.
- Four pilot subcarriers are used in 20 MHz mode and six pilot subcarriers are used in 40 MHz mode. On the other hand there are 52 data subcarriers for 20 MHz channel and 108 subcarriers for 40 MHz channel.

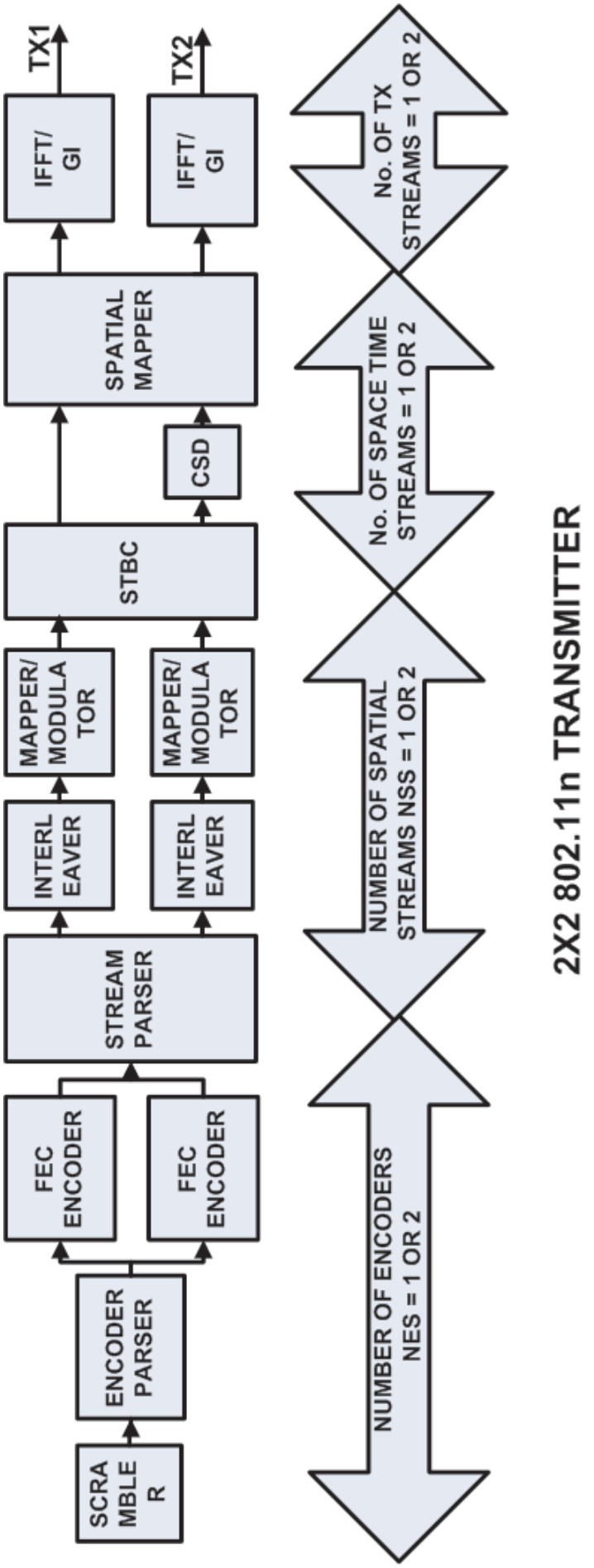

Figure 2.1: Generic 802.11n transmitter for 2X2 configuration

Generic 802.11n compliant transmitter has been represented in Figure2.1. The transmitter configuration has been restricted to 2X2 MIMO-OFDM systems.

2.2 Modulation and Coding Schemes

The modulation and coding schemes used for the design have been highlighted in Table 2.2.

Table 2.1: MCS Used

MCS	Mod1	Mod2	R	$N_{BPSCS}(i_{ss})$	N_{SD}	N_{SP}	N_{CBPS}	N_{DBPS}	Data Rate
0	BPSK	-	1/2	1	52	4	52	26	6.5
1	QPSK	-	1/2	2	52	4	104	52	13.0
2	QPSK	-	3/4	2	52	4	104	78	19.5
3	16-QAM	-	1/2	4	52	4	208	104	26.0
4	16-QAM	-	3/4	4	52	4	208	156	39.0
5	64-QAM	-	2/3	6	52	4	312	208	52.0
6	64-QAM	-	3/4	6	52	4	312	234	58.5
7	64-QAM	-	5/6	6	52	4	312	260	65.0
8	BPSK	BPSK	1/2	1	52	4	104	52	13.0
9	QPSK	QPSK	1/2	2	52	4	208	104	26.0
10	QPSK	QPSK	3/4	2	52	4	208	156	39.0
11	16-QAM	16-QAM	1/2	4	52	4	416	208	52.0
12	16-QAM	16-QAM	3/4	4	52	4	416	312	78.0
13	64-QAM	64-QAM	2/3	6	52	4	624	416	104.0
14	64-QAM	64-QAM	3/4	6	52	4	624	468	117.0
15	64-QAM	64-QAM	5/6	6	52	4	624	520	130.0
16	16-QAM	QPSK	1/2	6	52	4	312	156	39.0
17	64-QAM	QPSK	1/2	8	52	4	416	208	52.0
18	64-QAM	16-QAM	1/2	10	52	4	520	260	65.0
19	16-QAM	QPSK	3/4	6	52	4	312	234	58.5
20	64-QAM	QPSK	3/4	8	52	4	416	312	78.0
21	64-QAM	16-QAM	3/4	10	52	4	520	390	97.5

where,

R = Code Rate;

MCS = Modulation and Coding Scheme Index;

Mod1 = Modulation on First Stream;

Mod2 = Modulation on Second stream;

$N_{BPSCS}(i_{ss})$ = Number of coded bits per single carrier per spatial stream;

N_{SD} = Number of data subcarriers;

N_{SP} = Number of pilot subcarriers;

N_{CBPS} = Number of coded bits per OFDM Symbol;

N_{DBPS} = Number of data bits per OFDM Symbol;

Data Rate = Theoretical Data rate considering 800ns GI.

From the Table 2.2, three types of architectures Figure 2.2-Figure2.4 can be derived. We note that the number of encoders have been restricted to one.

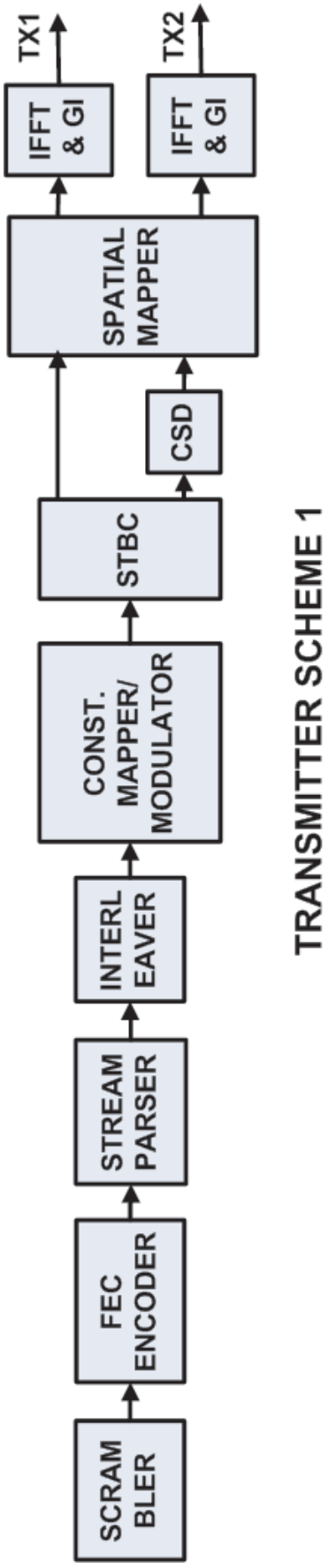

Figure 2.2: Transmitter scheme 1 for 2X2 configuration

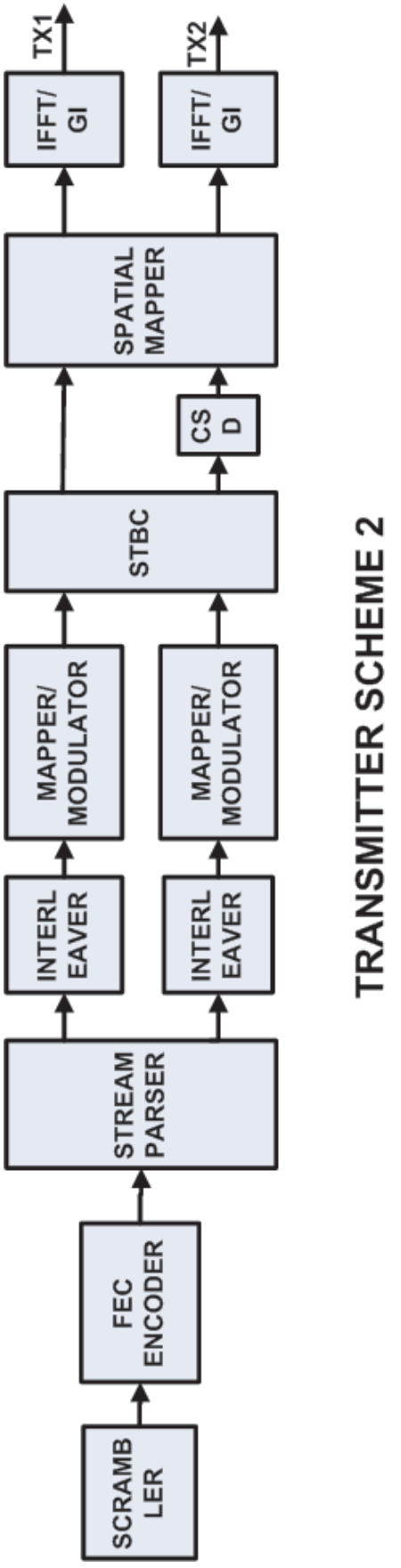

Figure 2.3: Transmitter scheme 2 for 2X2 configuration

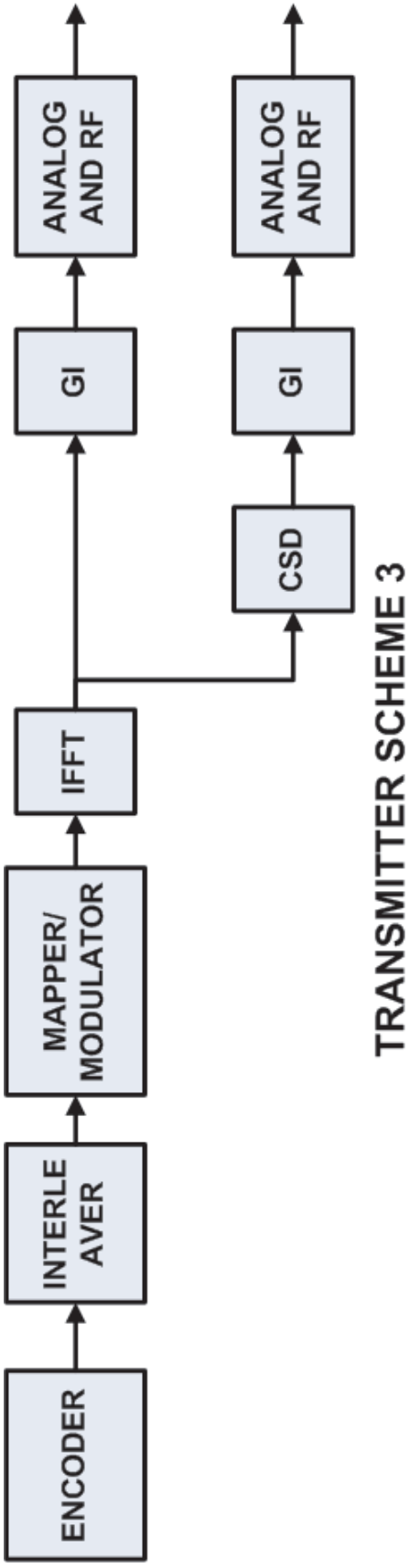

Figure 2.4: Transmitter scheme 3 for 2X2 configuration

2.3 Transmitter

The transmitter basically consists of the following blocks.

- *Encoder Parser* : It demultiplexes the scrambled bits among N_{ES} in round robin fashion; where N_{ES} is the number of encoders used. Since we have only one encoder block, encoder parser is removed from the three configurations.

- *Encoder* : It has been assumed that encoder is Binary Convolutional Code encoder with generator polynomials G_0=133 and G_1=171 with constraint length K = 7 .

- *Stream Parser* : It divides the outputs from the encoder to be sent to interleaver and mapper in *round robin fashion*.This sequence of bits sent to the interleaver is termed as *spatial stream*.

- *Frequency Interleaver*: It applies three levels of permutation in Interleaver. The first two levels of interleaver are exactly similar to that of legacy systems. The resultant of this interleaving is a *full interleaver*. We note that this block operates on serial data. Hence, it is anticipated that it will have large latency. We would analyze this block in much detail in subsequent chapters.

- *STBC* [6]: Since our MIMO model is 2X2, simple Alamouti scheme [7] would be used for STBC.

- *Spatial Mapper* It maps space time streams to transmit streams of the Transmitter. A matrix multiplcation is being performed. For 2X2 systems, this matrix multiplication simplifies to store and forward ofdm symbol procedure. If the number of space time streams is equal to the number of transmit stream, a simple one-to-one mapping is done.

- *CSD* : This is also termed as CDD. The OFDM symbol is cyclically shifted by different delay on each of the transmit streams. This leads to diversity in frequency domain since cyclic shift in time domain is equivalent to frequency shift in frequency domain. This concept applied on Long sequence of the Preambles would be used for estimating channel. [8], [9] and [10] have given an insight on CSD and how can it be used for better BER without increasing any complexities at the receiver.

- *FFT* : FFT block would be used for modulation of OFDM Symbol from frequency domain to time domain. FFT period defined in both the standards is 3.2us.

We note that PPDU encoding process will be exactly similar to that described in the 802.11n/802.11a standard [Section 20.3.4 of [5]].

2.3.1 Architectural Design of the Transmitter

- *Encoder, Stream Parser and Interleaver*: These blocks operate on data serially. Because of serial operation, these blocks have higher latency. There are two possible architectures for these blocks. BCC Encoder can be implemented using parallel architecture with reduced latency but at the cost of higher complexity. Stream Parser can be implemented by wiring specific bit input to specific bit output. Interleavers can also be designed using parallel architecture and serial architecture. We shall analyze both the architectures.As shown in Figure2.5, the distributed RAM is 1 bit wide with 1024 memory locations. The input address generator generates address for every bit on every clock cycle depending upon the type of protocol stream. We shall asssume that all the PE know which protocol is it operating on. This information is got through commands which would be described in detail in Chapter 3. This shall be analyzed in Chapter 3 in detail. The read and write operation alternate between the two RAMs A and B. Hence this kind of method is termed as *ping-pong buffer method*. This method has an advantage of lower complexity but higher latency. On the other hand, Figure 2.6 shows array of 1024 registers(1-bit wide). All the bits are fed at a time inside the registers using 16-bit wide or 32-bit wide data bus. At the output stage, these bits are read using 16-bit wide or 32-bit data bus. Though this method has latency of only one clock cycle for interleaving, the complexity of this kind of architecture is very high because of 1024 registers. We note that 1024 registers have been chosen in order to be able to accomodate largest chunk size of future technologies. For the present design, not all of 1024 registers shall be required.

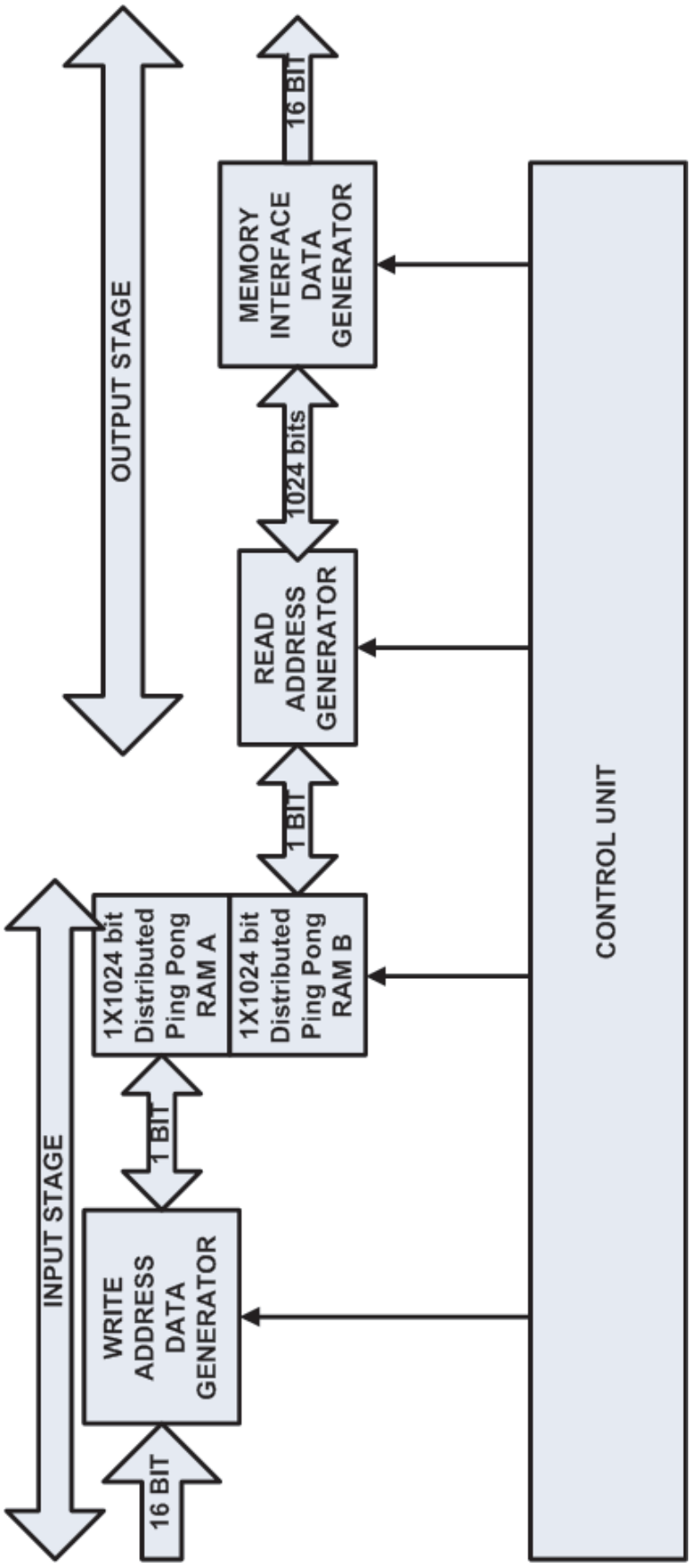

Figure 2.5: Implementation of Interleaver using Ping Pong Distributed RAM

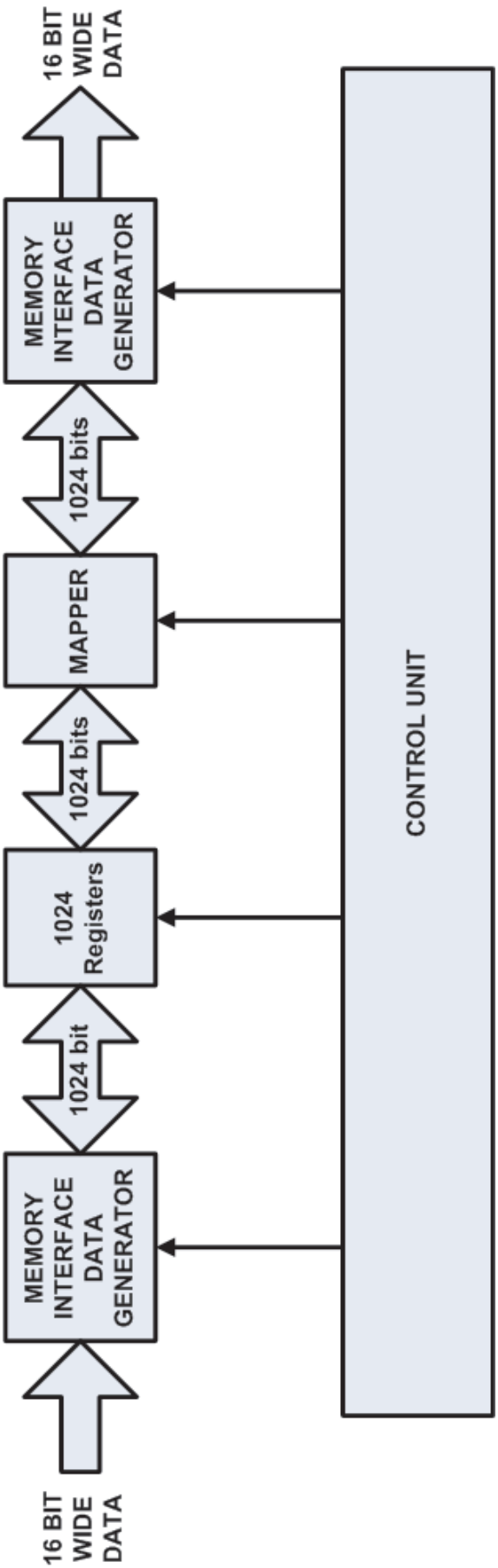

Figure 2.6: Implementation of Interleaver using Registers

- *FFT* : The FFT is another bulky block in terms of latency. Since programmable radio architecture operates on chunks (*chunk* means the number of OFDM symbols processed at a time. This could be one, two, four or eight.), FFT block has different latencies depending upon the chunk size and the FFT point. The latency of FFT depends on δ and χ where δ (equation 2.1)is the initial latency depending upon the type of architecture used and χ is the latency depending upon the chunksize. Hence, as the chunk size increases,total latency Π becomes nearly equal to χ. The effect of δ decreases. This is an important result for our analysis because programmable radio architecture splits the MAC data frame into chunks. Hence, if we operate on larger chunks, the latency would be minimal. Hoewever, this is not always the case. This case shall be further investigated in Chapter 5. We also note that FFT block used for the analysis has reconfigurable FFT Length. Further, the same FFT block can be used for IFFT operation. Since, WLAN based transceivers run on half-duplex mode, we can save considerable amount of logic gates by reusing the same FFT block for FFT/IFFT operation.

$$\Pi = \delta + \chi; \tag{2.1}$$

- *Cyclic Prefix Insertion* : Cyclic prefix insertion is handled by copying elements of RAM after taking the IFFT. This operation has very low latency.

- *Modulator* : This block is also called as Mapper in the literature. The latency of this block depends on the modulation block size and clock cycles required to read the data. Latency would also depend on the kind of architecture used. In this case wordsize of 32-bit would have a better performance over 16 bit wordsize. We assume that modulator block has all the information about the protocol stream under operation. This is further detailed in following chapters.

- *Preambles* : Legacy preambles of PPDU Header is similar to that in 802.11a compliant devices. However, the HT-preambles are cyclically shifted and modulated with QBPSK (i.e. on imaginary axis of the constellation) using 1/2 rate BPSK modulation. This is done so that the legacy devices do not *see* HT symbols. The HT-preambles consist of

HT-Signal,HT-STF and HT-LTF. HT-Signal field is used to *better understand the frame in terms of modulation, coding rate, bandwidth of the channel etc.*. HT-STF is used for timing acquisition and coarse frequency acquisition of HT compliant devices. HT-LTF at the receiver help in determining the number of spatial streams used by the transmitter. It is also used as sounding PPDUs for beamforming (optional feature not supported by programmable radio architecture).

2.4 Receiver

The MIMO OFDM receiver is more complex than transmitter in terms of gate level complexity. The latency requirements imposed upon by the MAC layer is more stringent hence, parallel processing is not feasible to full extent. A typical MIMO-OFDM based receiver block diagram has been shown in Figure 2.7.

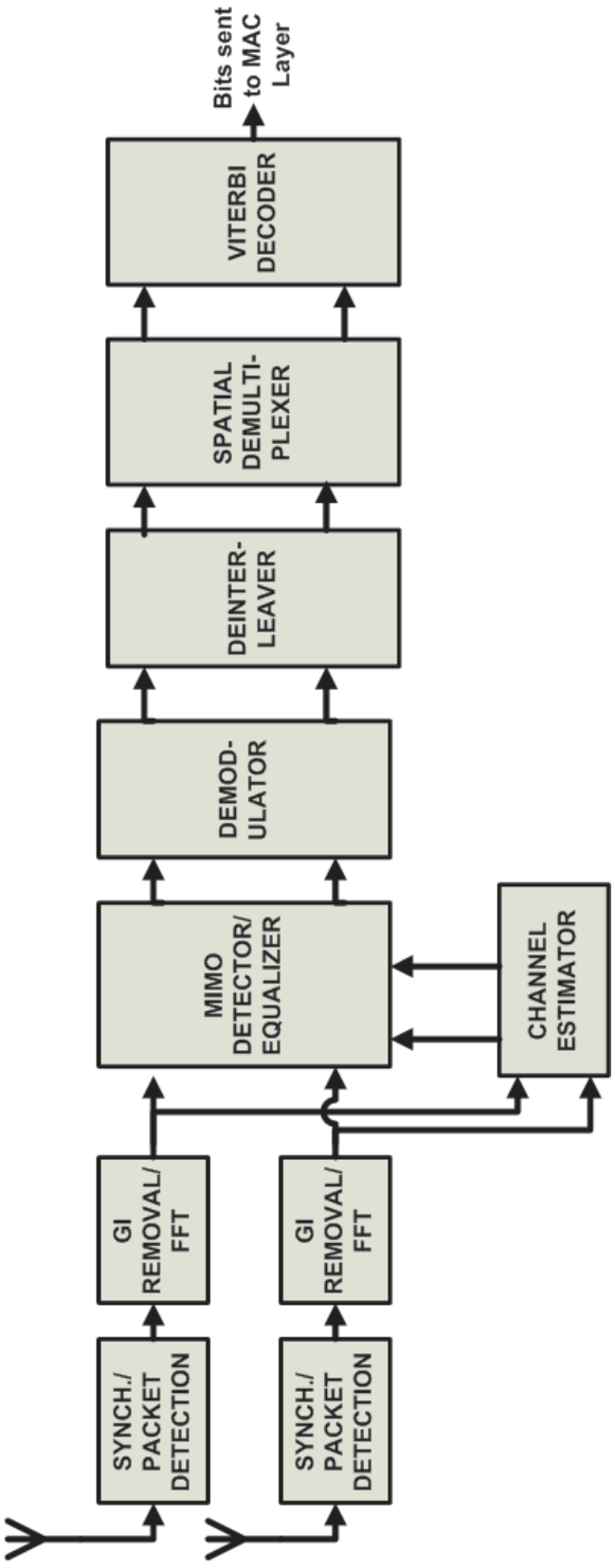

Figure 2.7: Typical block diagram of MIMO-OFDM based receiver.

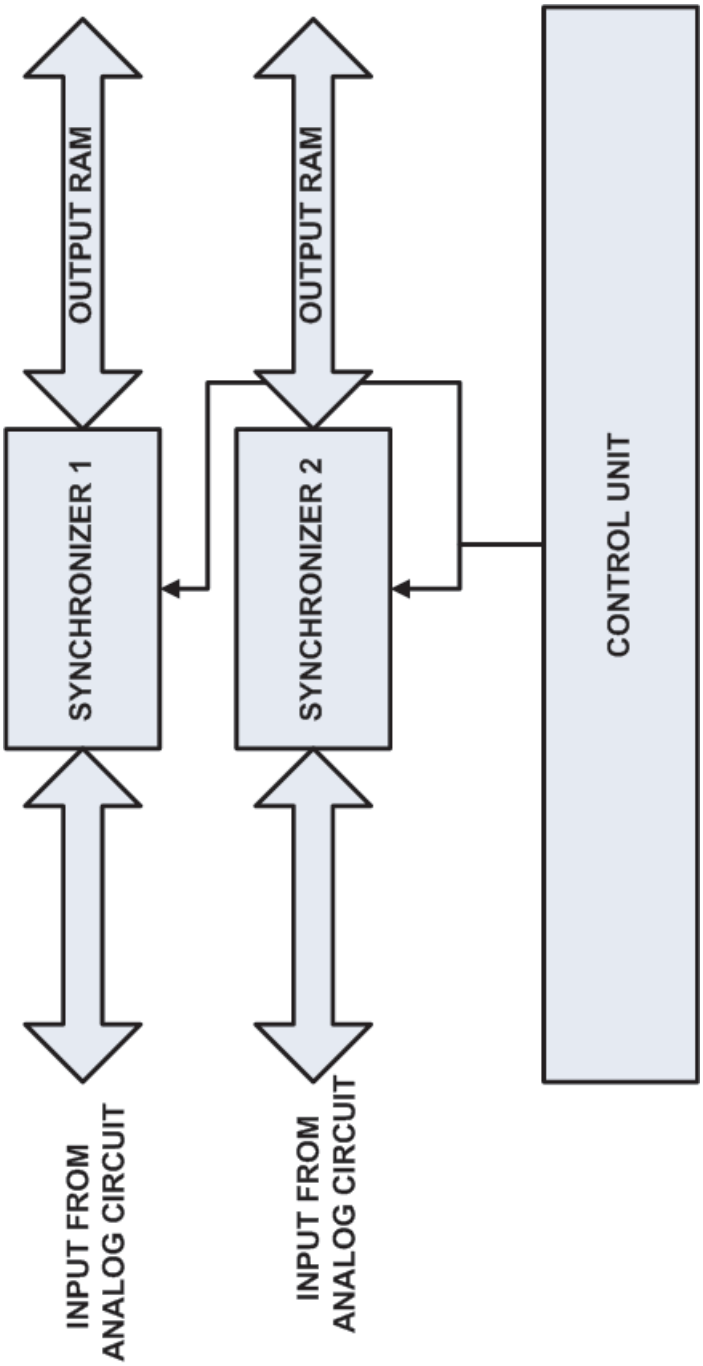

Figure 2.8: Typical Synchronizer block shared between the two streams.

These blocks have been further described in next subsection 2.4.1

2.4.1 Architectural Design of the Receiver

- *Synchronizer* :This block helps the receiver to get the timing information of the OFDM Symbol. Then frequency correction is done before FFT in order reduce the effect of inter-channel interference [11]. The Schmidl and Cox algorithm [12] based synchronizer has been developed for 802.11a based systems. There can be two different architectures possible for the synchronizer as shown in Figure2.8 and figure2.9.

  In the Figure2.8, there is only one synchronizer block shared among the two streams. The advantage of having such a block is to minimize gate level complexity. However, we would have to run this block twice the speed of normal clock rate for one synchronizer. For our analysis, we have assumed the second implementation of the synchronizer block shown in Figure2.9 owing to the fact that we don't have to run the receiver at higher clock rates. Secondly, our goal is to estimate feasibility of multiple protocol streams. Since the synchronizer block is in directly coupled with the analog circuit, the actual latency computation starts after synchronization is done. Hence, the design of this block has not been considered for the present analysis.

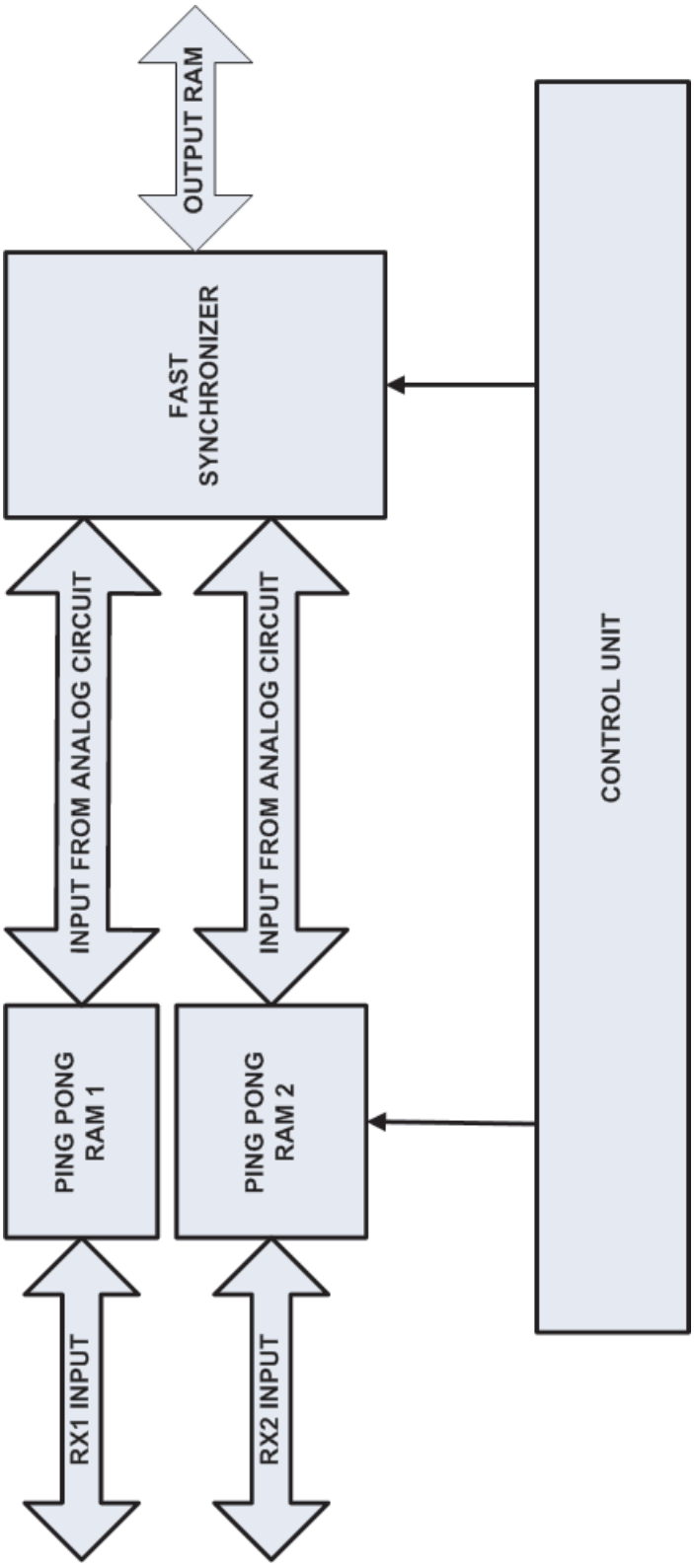

Figure 2.9: Dedicated synchronizer block for each stream.

- *Channel Estimation*: This block is used to estimate the channel coefficients. A number of techniques for channel estimation have been highlighted in [13], [14], [15] and [9]. However, these techniques cannot be applied to our architecture based on 802.11n because it has been assumed by them that transmitters maintain time domain orthogonality while transmitting training sequences (Long preambles) to the receiver. This is not the case in programmable radio architecture. We transmit long sequences after *cyclically shifting* the symbols in time domain. As a result of this we have orthogonality in frequency domain. Hence, we shall use the technique highlighted in [16]. A block diagram based on this scheme has been shown in Figure2.10 for one stream. We also note that this operation is done only once and it is assumed that *the channel response doesn't change for the rest of the frame.*

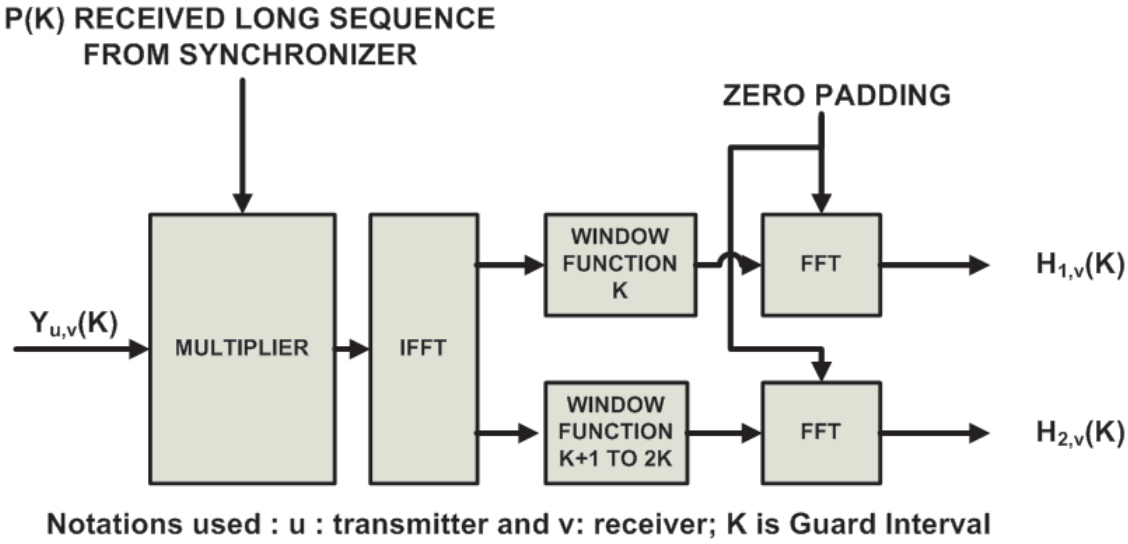

Figure 2.10: Typical Channel Estimation Scheme for MIMO-OFDM receiver. (Only one of the two possible streams has been shown in the figure).

- *MIMO Detector*: This block is also termed as equalizer because it nullifies the effect of channel and gives a good estimate of the transmitted symbol. Ideally the MAP rule offers the optimal performance. However, it has got a disadvantage of high complexity. For N transmitter system with modulation scheme Ω, where Ω is a modulation scheme like $\pm$ $\Omega \pm j\Omega$ for 4-QAM, the complexity is $|\Omega^N|$ since it has to search over all the symbols in codebook and find the minimum distance. We shall use MMSE linear receiver [17]

for our analysis. We note that zero-forcing based receiver has not been used (though it is much simpler than MMSE based receiver in terms of hardware complexity and latency) because it has got a drawback of enhancing noise power [17] [18]. Typically MMSE based equalizer is given by [17]

$$W = \frac{\rho}{M} H^{\dagger} (\frac{\rho}{M} HH^{\dagger} + N_0 I_N)^{-1} \tag{2.2}$$

where, $H^{\dagger}$ is the Moore-Penrose pseudoinverse of H [19].

W can then be used to linear estimate the transmitted symbol $\hat{a}$

$$\hat{a}_{LLSE} = Wr \tag{2.3}$$

where r is the received symbol.

Typical implementation of equalizer would be done using array of *MAC* blocks i.e. Multiplier-Adder-Cordic blocks. It can be easily found that the number of Multiplication, Addition and Cordics required to find equation 2.2 is 18 Complex multipliers, 17 Complex Adders and 1 Cordic square root block. The latency of 32-bit signed complex multiplier is approximately 5 clock cycles, 1 clock cycle for Adder and approximately 31 clock cycles for CORDIC to find square root [20]. When this operation is applied across all subcarriers, it becomes very complex in terms of gate count. Hence, *this block shall be reused for other subcarriers*.

- *Demodulator*: This block is similar to the Modulator block of transmitter. This block is capable of handling two streams at a time without much increase in latency and complexity.

- *Deinterleaver* : This block is exactly similar to that of Interleaver block of tranmsitter. This block has the capability of handling two streams at a time.

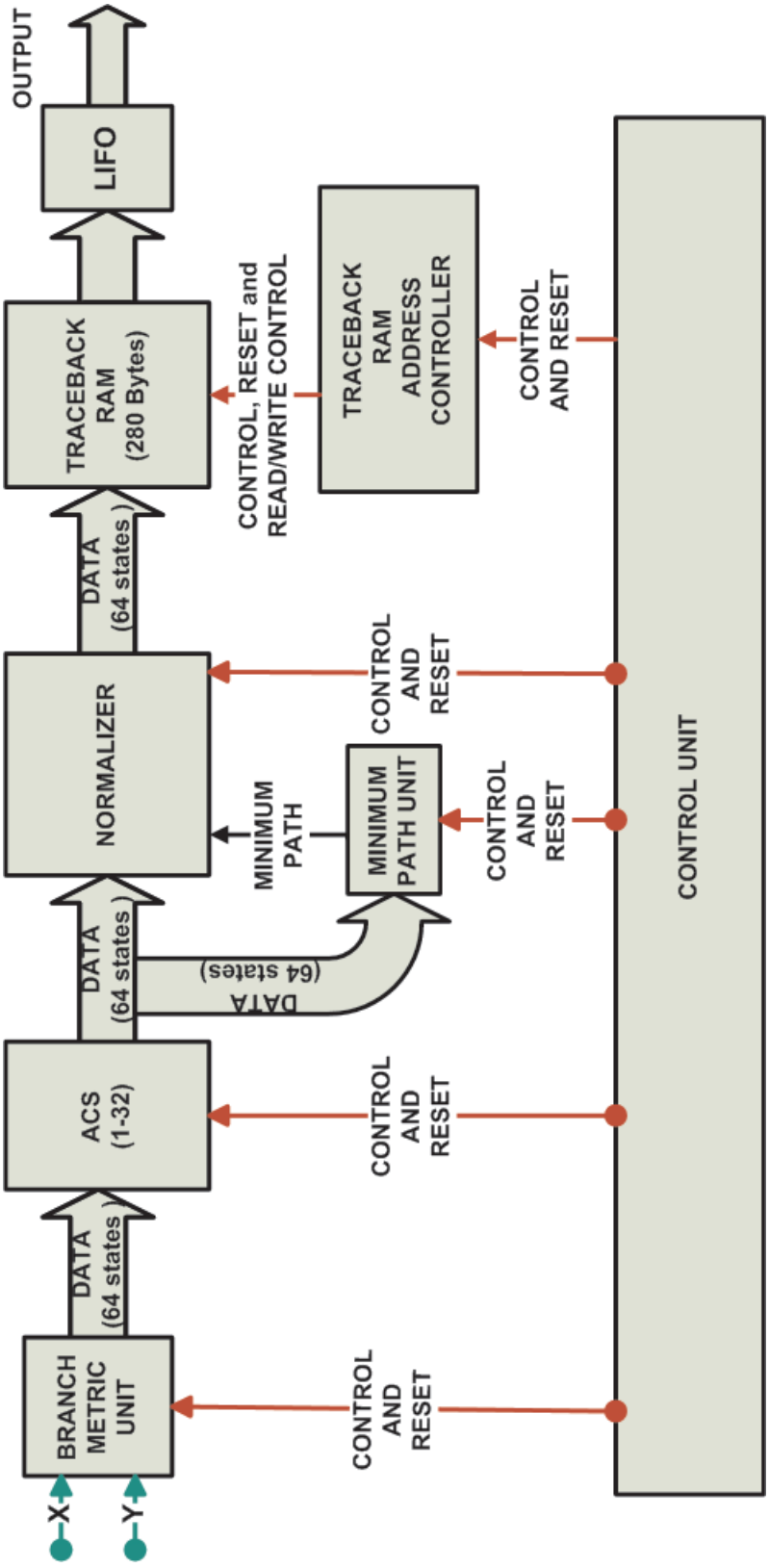

Figure 2.11: Top level block diagram of Soft Output Viterbi Decoder.

- *Viterbi Decoder*: This block decodes the bits generated by convolutional encoder. Traceback RAM method has been designed for operation of Viterbi Decoding. The architectural diagram of viterbi decoder is shown in Figure 2.11. Softdecision based decoding is done. It consists of following blocks :

    1. *Branch Metric Unit*: The branch metric unit calculates posteriori probabilities by estimating the distance with all possible transmitted codewords. An efficient Branch Metric Algorithm A has been developed for mapping the four level output of the Branch Metric Table to 64 rows of *ACS unit*.
    2. *ACS Unit* : This unit is called as Add Compare Select Unit. This unit compares all the possible path metrics of the trellis. The path metrics are computed knowing the information about the current state and the next state. For better performance and lesser latency, parallel implementation (64 such units) of this unit has been assumed.
    3. *Normalizer* : This unit calculates the minimum path metrics and subtracts it from all the branch of trellis. This helps to prevent the overflow problem encountered in hardware systems.
    4. *Traceback Unit* : This units consists of the RAM where all the possible path metrics are stored. This unit is considered as the heart of the Viterbi Decoder. This unit is most expensive with regards to complexity since it consists of four dual port RAM. For reference, the operation of these four RAM have been shown in Figure 2.12.

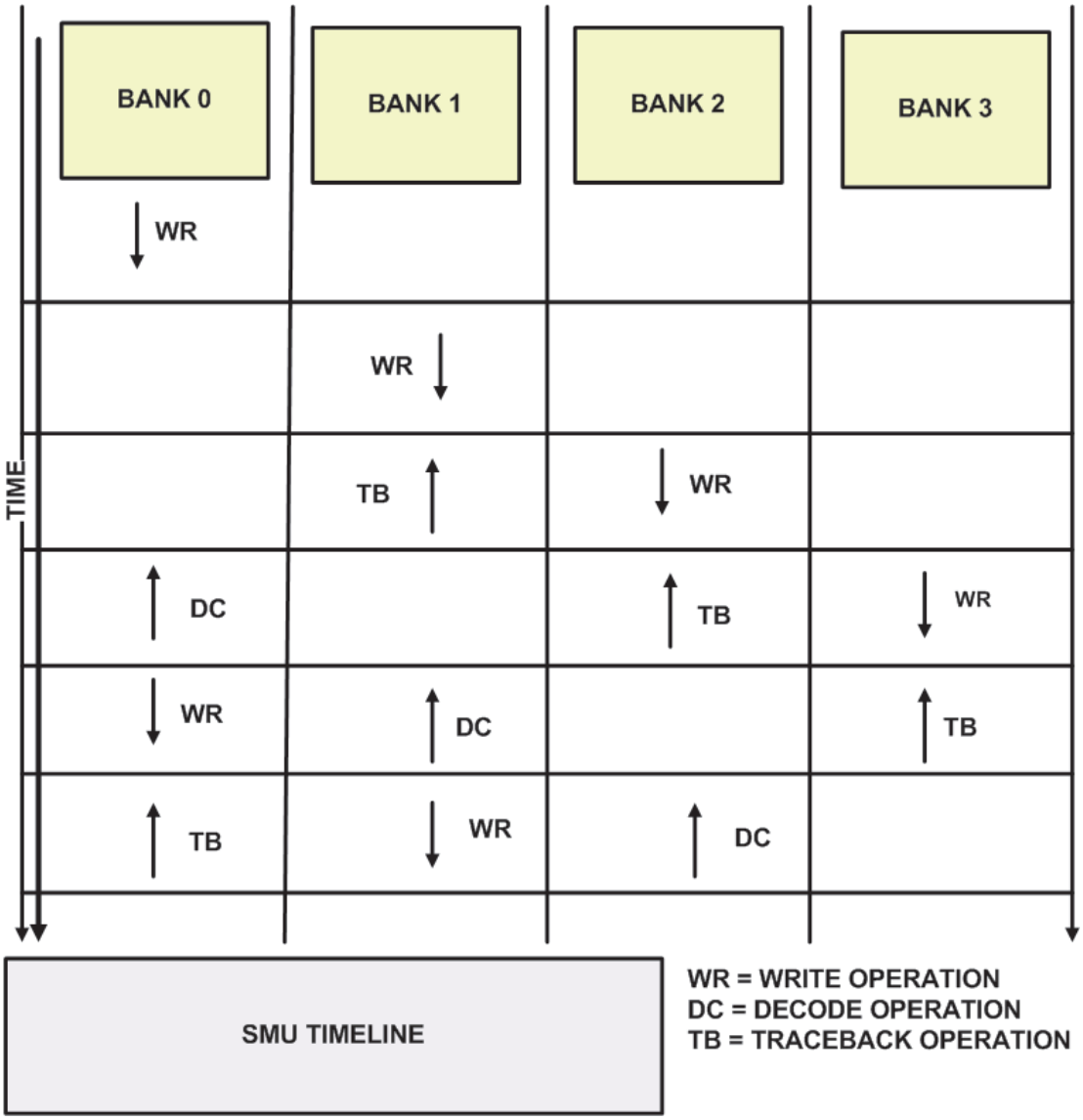

Figure 2.12: Pipelined operation of traceback RAM.

Viterbi decoder specification and design have been detailed in Appendix D.

- *FCS Unit* : This block calculates the Frame Check Sequence (CRC). Computing CRC doesn't involve much latency. The CRC implementation unit used here operates on 8bits in parallel. It discards the frame if an error is found in it.

Chapter 3

PROGRAMMABLE RADIO PLATFORM (WiNC2R)

3.1 Introduction to the Programmable Radio Platform

The Programmable radio platform consists of three blocks PE, system controller (UCM) and common blocks. Each PE with these common blocks is termed as *Functional Unit.* Common blocks are Memory associated with PE, Memory address interface, Command Processor and Task Manager. Each UCM of FU has global knowledge and intelligence for collaboration. Each block of FU is vividly described in later sections. On the whole, there exists a top level communication among FU. This top level communication is handled by UCM which is described in more detail below.

A block diagram describing the connection of these blocks is shown in Figure3.1

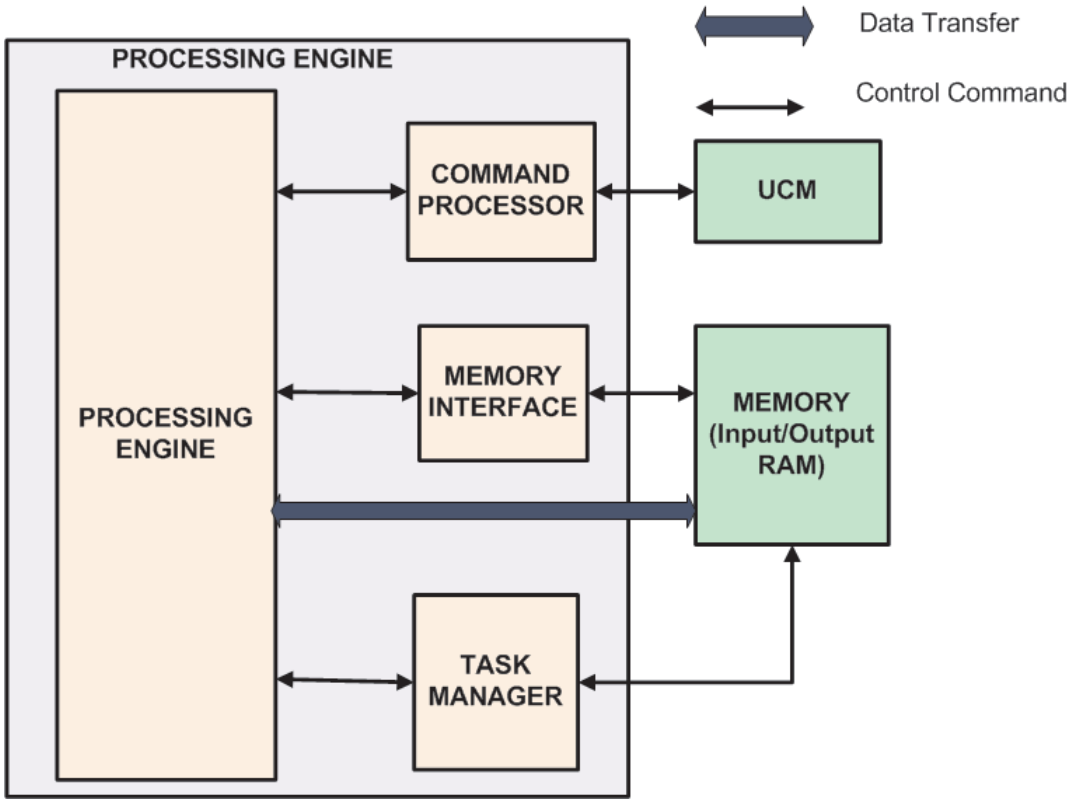

Figure 3.1: Block diagram of building blocks of programmable radio architecture.

All the blocks have been briefly described below:

- ***UCM*** : This is the heart of the whole architecture. The block mainly performs three functions :

  1. *Collaborating hardware modules* : It ensures that all the hardware modules work together in harmony. It maintains a database for keeping a record of each PE. It also makes sure that tasks like preemption, scheduling, queueing etc. is performed well.
  2. ***Controlled processing resources sharing*** : It makes sure that some of the resources like Memory and PEs are shared properly without any conflict. Also, the resources are made available whenever required.
  3. *Control mechanisms for sequencing operations between hardware modules* : It ensures that all the control mechanisms function properly. This is required because all the other blocks totally rely upon control commands. Hence, if something goes

wrong with these control signals, the architecture would collapse.

- ***PE*** : This block is the basic building block of the whole architecture. This block is designed in such a way that the UCM overhead is less. We note that UCM overhead is associated with the latency due to data transfer between PEs and some clock cycles for preprocessing and postprocessing of data. As a result of which some of the closely related blocks have been grouped in one PE Figures 3.2-3.4

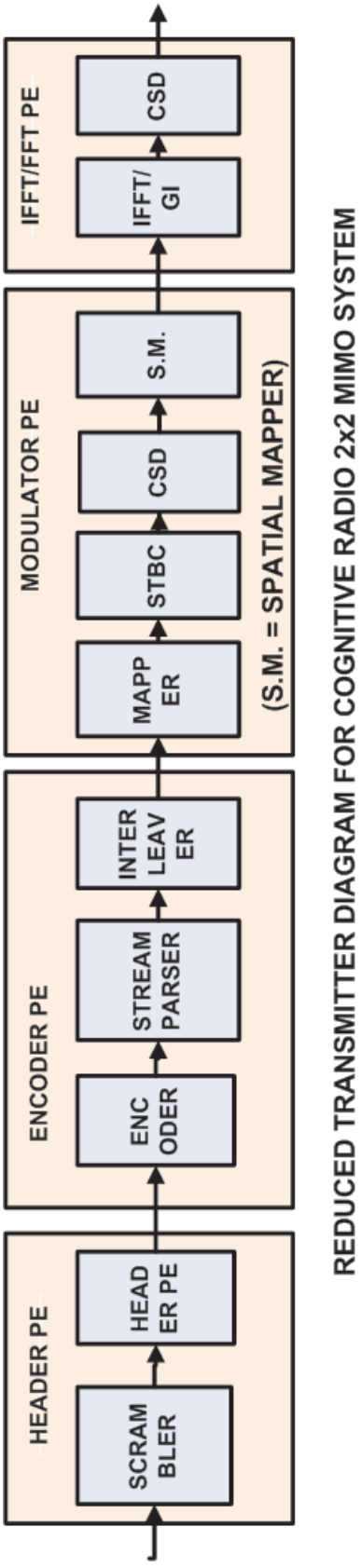

Figure 3.2: PE Level block diagram of transmitter.

Also, the encoder block splits one stream into two streams in 802.11n mode with two streams. However, it requires only one stream for legacy protocols. Figure 3.3.

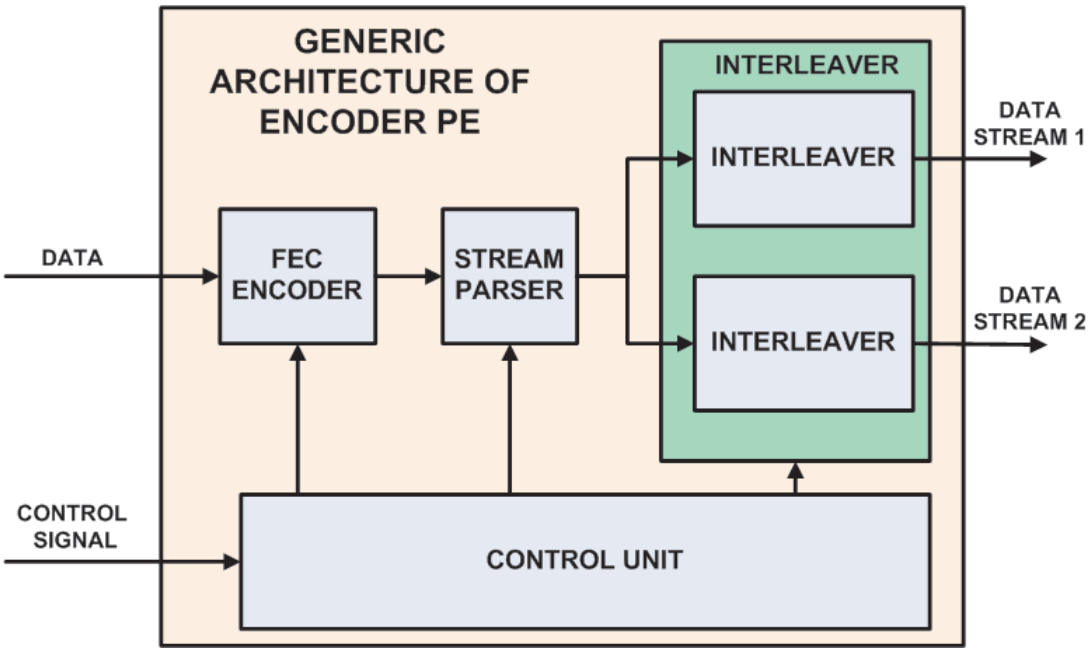

Figure 3.3: Encoder block diagram capable of supporting two streams.

The PE level block diagram for the receiver is shown in Figure3.4.

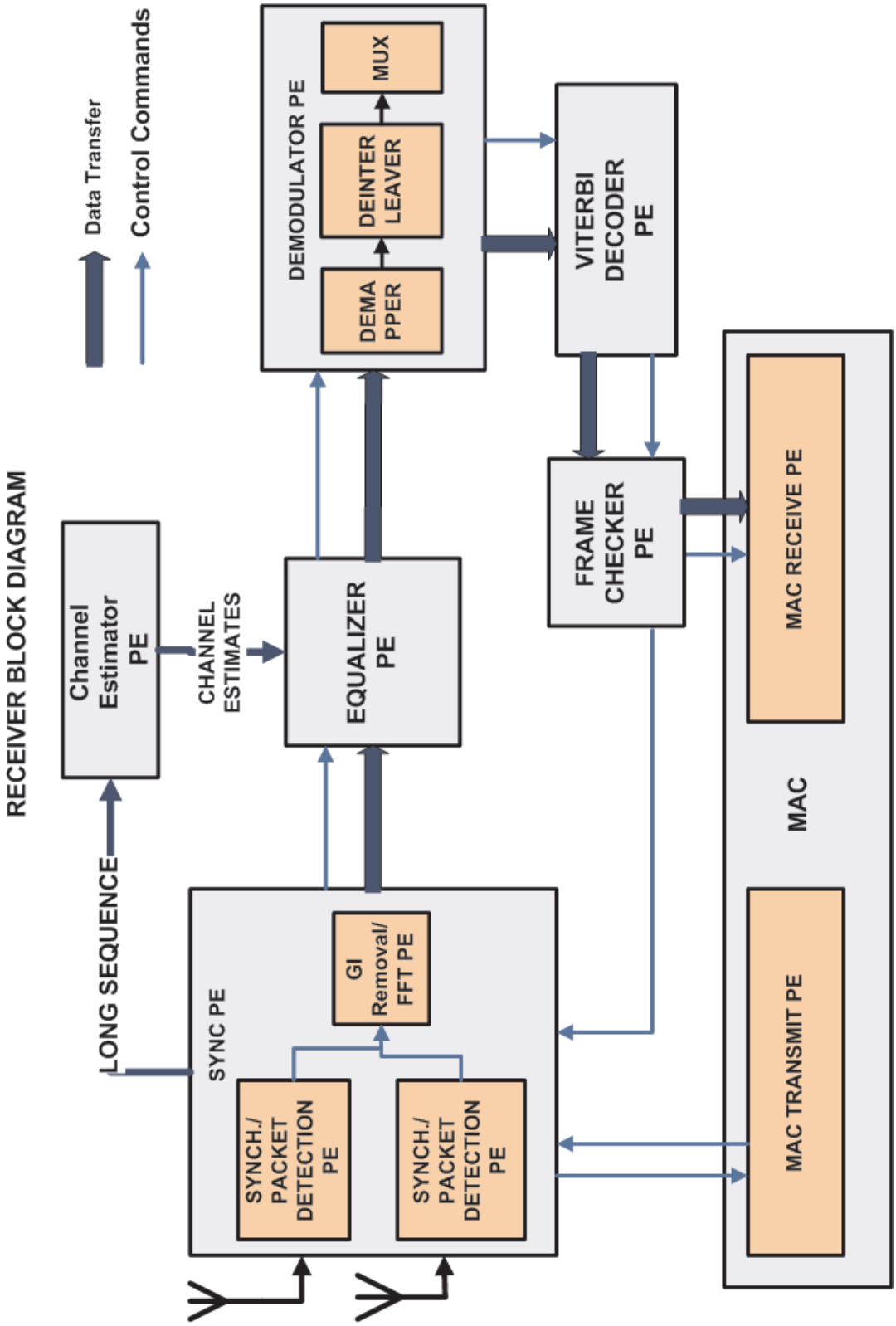

Figure 3.4: Block diagram of receiver showing all the PE with similar functionality being combined.

We see that blocks like Demodulator, Deinterleaver and Stream multiplexer have been combined to form one Modulator PE in order to reduce UCM Overhead. The effect of UCM Overhead has been analyzed in more detail in chatper 5

- ***Common Blocks***: The common blocks are *instantiated* with each PE depending upon the role of PE. These blocks help PE to

  1. communicate with UCM
  2. take appropriate action upon receiving a command from the UCM.
  3. handle errors during processing and hence help PE level debugging.
  4. generate addresses for input and output buffers associated with each PE.
  5. handle tasks generated by other PE. It also helps to dispatch new tasks depending upon the requirements.
  6. handle preemption of low priority tasks by high priority tasks.

  These common blocks are described in brief below:

  – *Command Processor* : This block decodes the command sent by the UCM block and generates an appropriate signal to the PE. Command flows and timing diagram have been analyzed in more detail in section .

  – *Memory Interface Module* : This block generates enable, start of frame and end of frame signal so that the data can be easily processed by underlying blocks.

  – *Task Manager* : It spawns new tasks either addressed to itself or addressed to some other PE.

3.2 The Programmable Command Flow and Timing Diagram

In this section we shall analyze the command flow across PE at the transmitter and the receiver.

3.2.1 Transmitter Command Flow

As shown in Figure3.5, as soon as the header PE gets a command, it splits the frames into chunks. There is a command associated with every chunk. The command includes all the PE

specific attributes for processing. A typical command has been shown in Appendix B. As soon

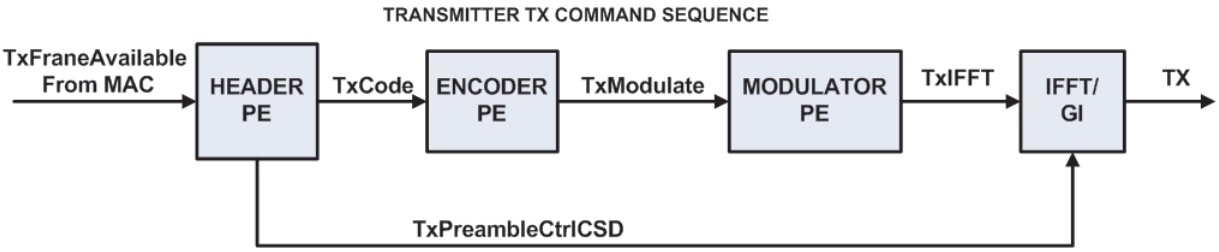

Figure 3.5: Block diagram showing the command flow of the transmitter block.

as Encoder PE receives command TxCode from the header PE, it first processes the chunk and then dispatches another command TxModulate for the Modulator PE. The modulator PE processes the chunk and dispatches command to IFFT block for modulation into time domain. When the PE sends a command, it also sends the data. The job of accomplishing data transfer is done by UCM block. It also does the bookkeeping required for data transfer. TXPreambleCtrlCSD specifies whether CSD should be inserted before the IFFT block or after it. This can be analyzed by referring to figures 2.2 - 2.4. The timing diagram associated with the above commands is shown in Figure 3.6 of subsection 3.2.2.

3.2.2 Transmitter Timing Diagram

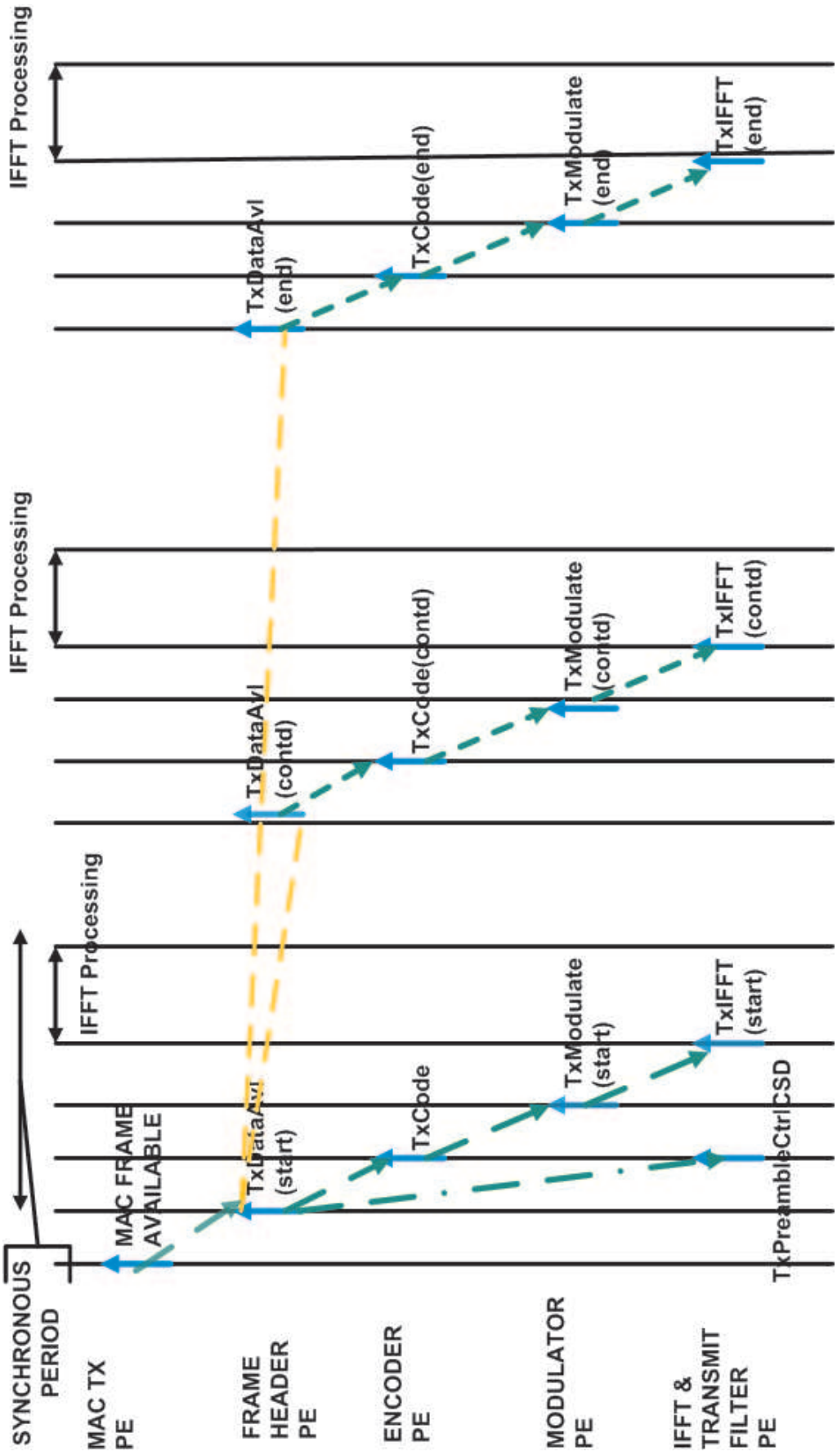

Figure 3.6: Timing diagram of transmitter commands.

Each transmitter splits the frame into chunks. Each chunk is processed in one synchronous time period. Allocation of synchronous period is done in such a way that the timing constraints of OFDM symbol are met. The analysis in Chapter 5 would consider these aspects. *We would not be finding the actual synchronous period but we would find the minimum clock rate essential to meet this period.*

3.2.3 Receiver Command Flow

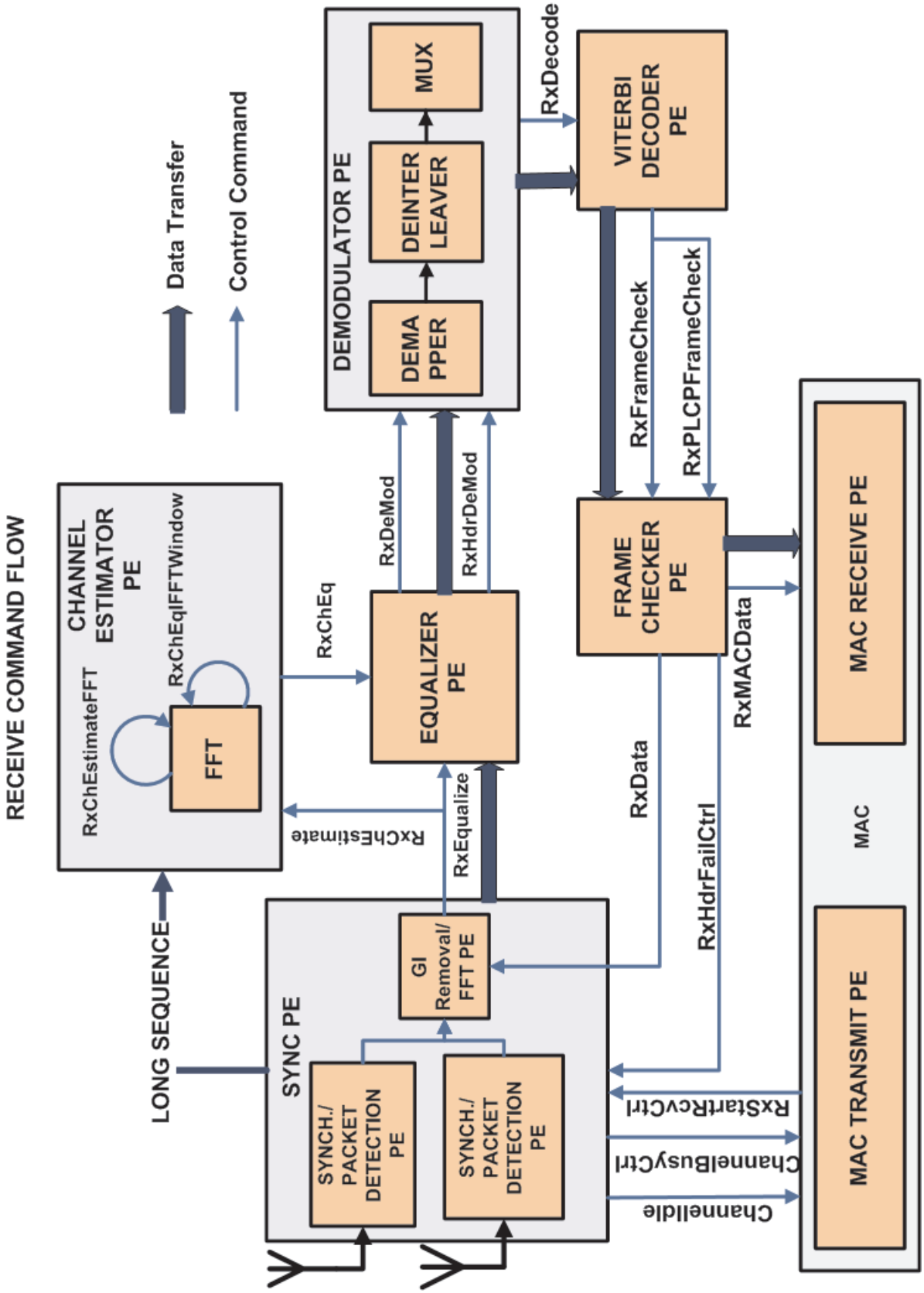

Figure 3.7: Block diagram showing the command flow of the receiver block.

The receiver command flow is quite similar to that of the transmit command flow but has few complications. The receiver first receives the header, runs CRC on the header and then analyzes the type of the frame i.e. modulation type, chunk size, coding rate etc. Hence, the timing requirements for the receiver block is more stringent. The processing has to be done before the interval of one OFDM symbol expires. This imposes a big constraint on the architectural design. This topic is more clearly discussed in chapter 5. As shown in Figure3.2.3, as soon as the synchronizer is done with its job, it sends long sequence to the Channel Estimator PE so that it can start this estimation. Once the Channel Estimator PE finishes finding the channel estimates, it sends a command RxChEq to the Equalizer PE. Once the channel estimates are got, the secondary path (consisting of Channel Estimator PE) is not traversed. The data is equalized from the stored channel estimates. Initially, the header reception is initiated by sending RxHdrDemod command. The modulator block works on the equalized data to send out bits to the Viterbi Decoder PE. The Viterbi Decoder PE decodes the data sends this data with the command RxPLCPFrameCheck to the FCS block. If the CRC check passes, this data is sent to the synchronizer block. The chunking is initiated by Frame Checker PE in the receiver as opposed to the header PE in the transmitter. Once all the attributes of the frame are received and decoded using the header, the normal operation of reception is done. The timing diagram has been shown in Figure3.8.We also note that if the CRC on the header fails, the FCS PE sends RxHdrFailCtrl command to the synchronizer PE to drop the frame.

3.2.4 Receiver Timing Diagram

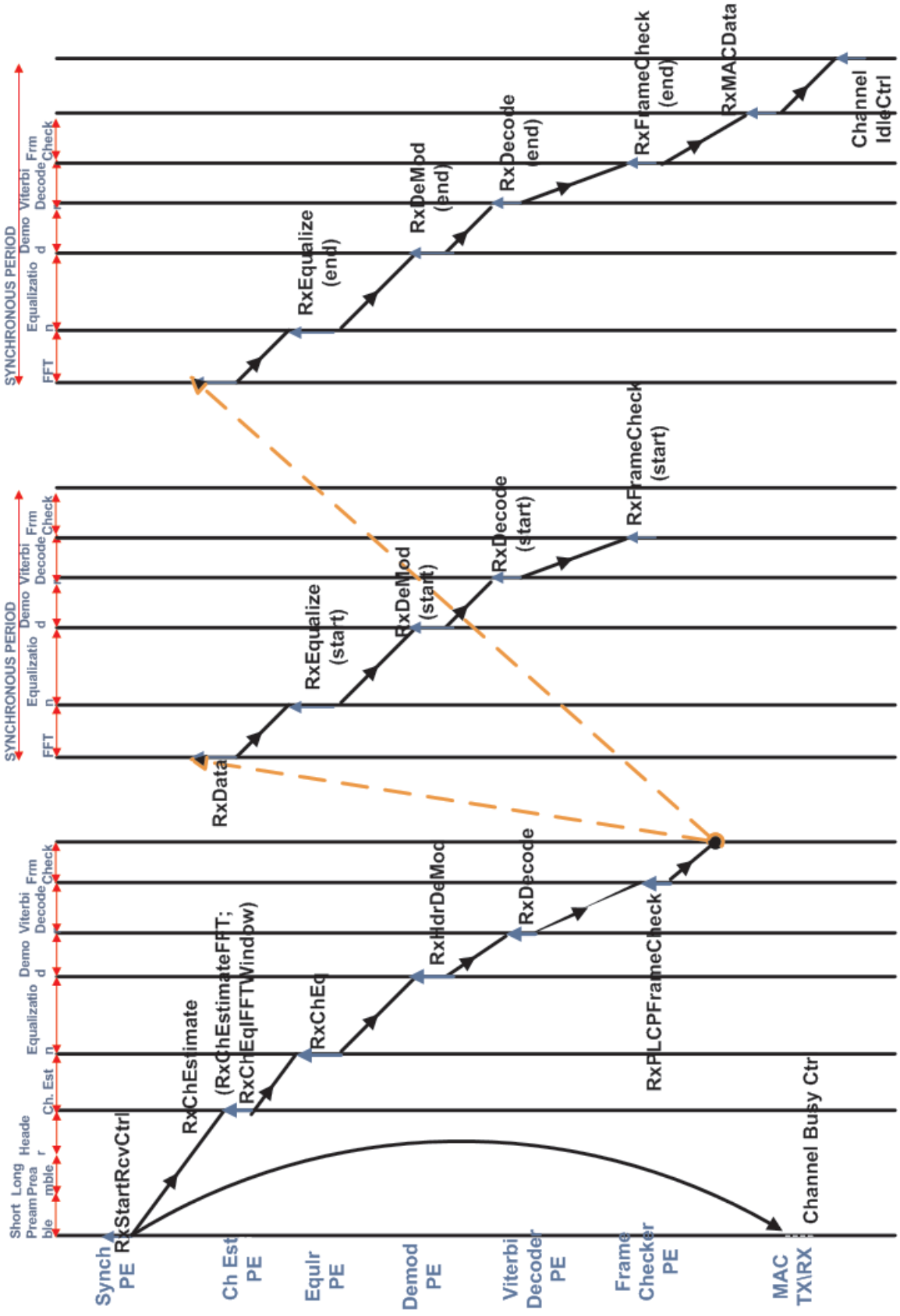

Figure 3.8: Timing diagram of receiver commands.

As shown in figure, the chunking process is initiated by the Frame Checker PE. This chunking is done every synchronous period. The task of Channel Estimation is done only once throughout the frame and it is assumed that the channel characteristics do not change over the rest of the data.

3.3 Half Duplex Operation of the PHY Layer

Since this transceiver runs on half duplex mode, either transmitter or receiver can be active at a time. If the transmitter is transmitting the frame, it sends TxStartCtrl (not shown in the figure) to the receiver block. This command shuts down the receiver. As soon as transmitter is done with its job, it sends another command TxEndCtrl to the receiver enabling it.

3.4 Interfacing with the MAC Layer

The MAC layer sends RxStartRcvCtrl command to enable the PHY Receiver. This command would be used like a bootup command. At the time of bootup, the MAC layer would specify the kind of receiver to be implemented i.e. the number of streams to be used by the receiver. At the time of bootup the CCA would be set to busy. Whenever the change of state of channel is determined (i.e. CCA could be busy or idle), the PHY layer then sends ChannelBusyCtrl or ChannelIdleCtrl command to the MAC layer. Once the reception of frame is finished, ChannelIdleCtrl Command is dispatched to the MAC Layer. We also note that once the whole frame is received, the CRC check sum is sent along with the command RxMACData command.

In the next section we shall analyze the simulator setup and strategies.

Chapter 4

EXPERIMENT SETUP: SIMULATION STRATEGY

In this section we shall describe the simulation setup, assumptions and models for analyzing transmitter and receiver.

4.1 Assumptions

- It is assumed that there are no queues. This assumption is valid because even though there are asynchronous and synchronous queues in the programmable radio platform, there isn't any mechanism to monitor the queues and prevent overflows. Also, in the present architecture for 802.11a, queues are not being used. But we note that the system performance can definitely be increased if we incorporate and queue handling mechanisms. In this case, we would also have to consider queueing mechanism overhead on top of UCM overhead. Further, in absence of queues, a PE can only transfer the task to the next PE if the next PE has completed its job. Otherwise, the PE will wait for the next PE to finish its job.

- The transmitter and receiver are analyzed independently. This assumption is valid because our transceiver is half duplex. Hence, we have to select the worst case scenarios for transmitter and receiver.

- We would not be analyzing MAC layer and UCM overhead of MAC layer because it is out of the scope of this work.

- We assume that the analog circuit is reconfigurable and is capable of performing Analog signal processing. This module is under development for the present architecture of WiNC2R. This assumption is important because we are dealing with multiple standards along with multiple specifications.

- We assume that *Peak to average ratio problem* is handled by the analog circuit since this topic is out of the scope of this thesis.

- Since puncturing and de-puncturing doesn't require much overhead, this has been neglected. This assumption is valid because puncturing blocks selects few bits from the output of $\frac{1}{2}$ rate convolutional encoder. It can be easily verified that the number of clock cycles for performing this operation is minimal.

- It has been assumed that the MAC layer is capable of transmitting and receiving mandatory features specified in 802.11n [5] and 802.11a [4] standard.

- For the compatibility with the legacy devices, we are assuming other protocol stream to be 802.11a compatible.

- For estimating the complexity of the architecture, InCyte [1] tool is being used for getting average gatecount of IP cores. This would help us to have a rough estimate of our architecture with respect to latency.

- Design of UCM block is not done. However, simulations have been done for testing feasibility of this block and providing bounds on UCM clock overhead.

- The simulation is not event driven simulation but based on database known to the PE.

- It is assumed that all the PE function in pipeline.

- For the legacy systems, only one of the two receiver antennas would be sufficient because the signal on the other antenna would be cyclically shifted version of that on the first antenna. Also, for mixed format or multiple streams we can dedicate one receiver per stream. However this design totally depends on the front end receiver design of the analog circuit. *For the present analysis, we assume that we have multiple streams available at the same time on both the synchronizer blocks. However, for legacy systems, we shall be using only one receiver antenna and the second receiver shall be switched off.*

[1] InCyte is a product of Cadence Design Inc.

4.2 Simulation Setup

In this section we shall understand the interfaces of the simulator in more detail.

4.2.1 Objective of simulations

1. For a given a set of (T_i, A_i) where T_i belongs to a set containing throughput and A_i belongs to a set containing architectures, the simulator would estimate gatecount (complexity), system throughput and per unit utilization. This would help us to know if we can incorporate some extra functionality within the architecture.
2. To reduce the UCM overhead by efficiently deciding upon the chunk size.
3. To estimate and reduce latency
4. To estimate gate count/CLBs for fpga.
5. To estimate minimum clock rate for a given architecture and given throughput requirements.

4.2.2 Input parameters to the simulator

1. ***Word size*** 16 bit or 32 bit.
2. ***Number of OFDM Symbols to be processed*** : One, two, four or eight OFDM symbols.
3. ***Frame Size*** : Size of frame per stream.
4. ***Protocol*** : Protocol in use of each stream
5. ***Flexibility of reuse*** : Flexibility on reuse of architectural components like MAC (Multiplier-Adder and Cordic), FFT etc.

4.2.3 Output of the simulator

The output of the simulator would be quantitative results that would gauge the performance of the architecture. The performance metrics would be the complexity (gate count). Depending on Latency and gate count, minimum clock rate would be computed.

4.2.4 Top Level Interface specification of the simulator

Typically top level interface specifications of the simulator would be :
(Total Latency, Gatecount) = $ncp\_simulator$(Wordsize, Number of OFDM Symbols, Framesize, Protocol in use, Reuse parameter, System level parameters) where,

- ***Wordsize*** : The word size of the architecture.
  - 0 : 16 bit wide word size.
  - 1 : 32 bit wide word size.
  - 2 : 64 bit wide word size (optional for future use. The present analysis is valid only for 16-bit and 32-bit wide words)
- ***Number of OFDM Symbols*** : This indicates how many OFDM symbols would be processed by each stream.
- ***Framesize*** : This would indicate the total number of data bytes to be transferred for each protocol.
- ***Reuse parameter*** : This parameter if set would imply to reuse the hardware components throughout the simulation. If reset, two components would be instantiated. This would lead to larger occupancy of area but less latency.
- ***System level parameters*** This has been described below.
  - ***Desired bit rate*** : Desired bit rate on each of the protocols to be tested.
  - ***Clock frequency (the base frequency)***. This is the base frequency that is used for all the standards. This is set to 20 MHz.
  - ***Clock multiplication factor*** : This is the multiplication factor of *Clock frequency* applied to target higher frequencies ASIC/FPGA design.
  - ***Number of streams*** : The total number of streams to be used for MIMO case only. For 802.11a case, this would be set to 0.
  - ***Transmitter configuration to be used***: The transmitter configuration to be used out of the three possible transmitters. This parameter would differentiate between the three possible transmitter configuration of 802.11n

- ***Total latency***: This would be the total estimated latency after adding individual latencies for each chunks of each protocol and UCM overhead latency Appendix D.

- ***Gatecount***: This would calculate the total gatecount depending on whether we are reusing components in our design. This would also depend on the transmitter configuration chosen.

4.3 Operation of the simulator

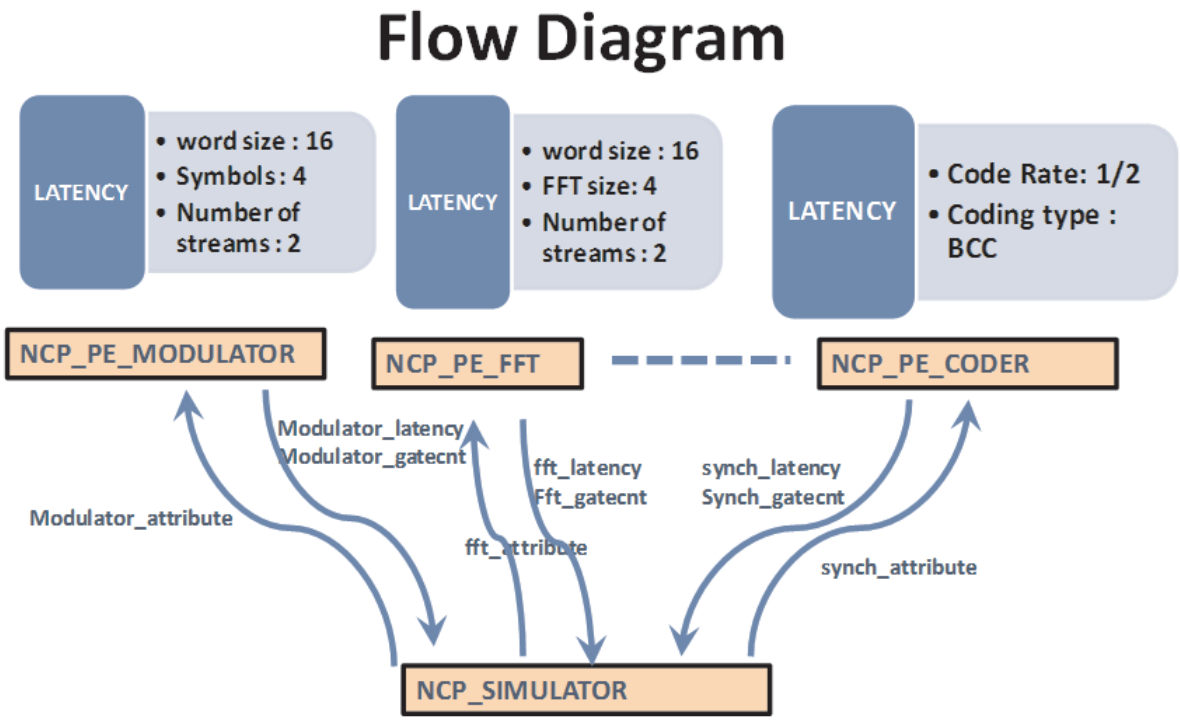

Figure 4.1: Diagram describing operation of the simulator.

As shown in the Figure4.1, the simulator decides which PE would be used based on configuration parameters written in the file. It then calls respective PE Interface. It also passes all the necessary attributes to respective PE interface. Each PE interface returns gate-count and latency depending upon the configuration. If the data size exceeds that of one chunk size, the simulator would divide the data stream into smaller chunks and then estimate the total latency. The same analysis is equivalently valid for the receiver also. The simulator estimates the total latency taking into account pipelining operation as shown in Figure 4.2.

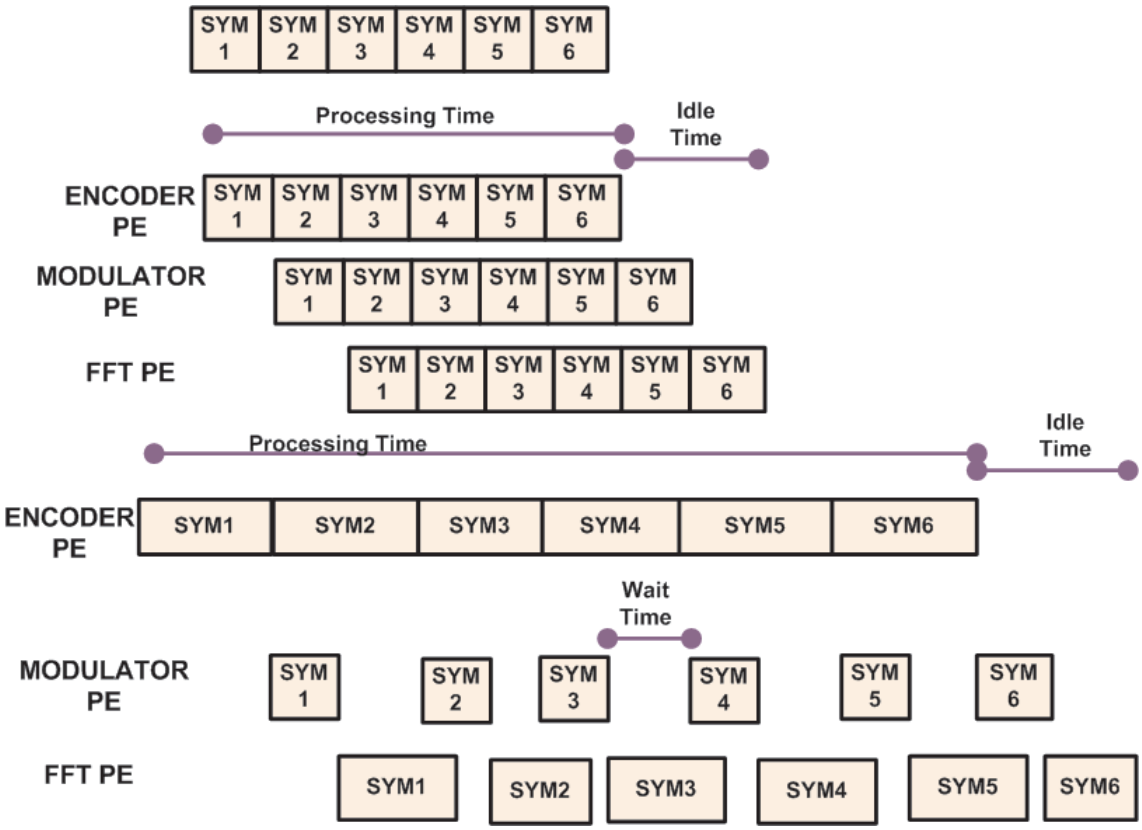

Figure 4.2: Diagram describing pipelining operation of the simulator.

As shown in the Figure4.2,*Idle time* is defined as the time during which the PE doesn't have any chunk for processing. *Processing Time* is defined as the time during which PE processes the incoming chunk.*Wait Time* is defined as the time during which the PE waits for its neighboring PE to finish off the job so that it can either hand over the task to the next PE or take over the task from the previous PE.

4.3.1 Estimation of latency

The latency estimation has been described in Appendix D. We would need to refer to the MCS table as shown in figure 2.2

4.4 Receiver Model

It has already been pointed out that receiver design is more complex than transmitter. Hence a model has been developed for analysis which is as described below. In this model, we assume

that a new chunk is available for PE as soon as it is done processing the chunk. In reality, this may not happen. This is clear from the Figure4.3 shown below. Practically there would be some $\triangle$delay because of UCM Overhead.

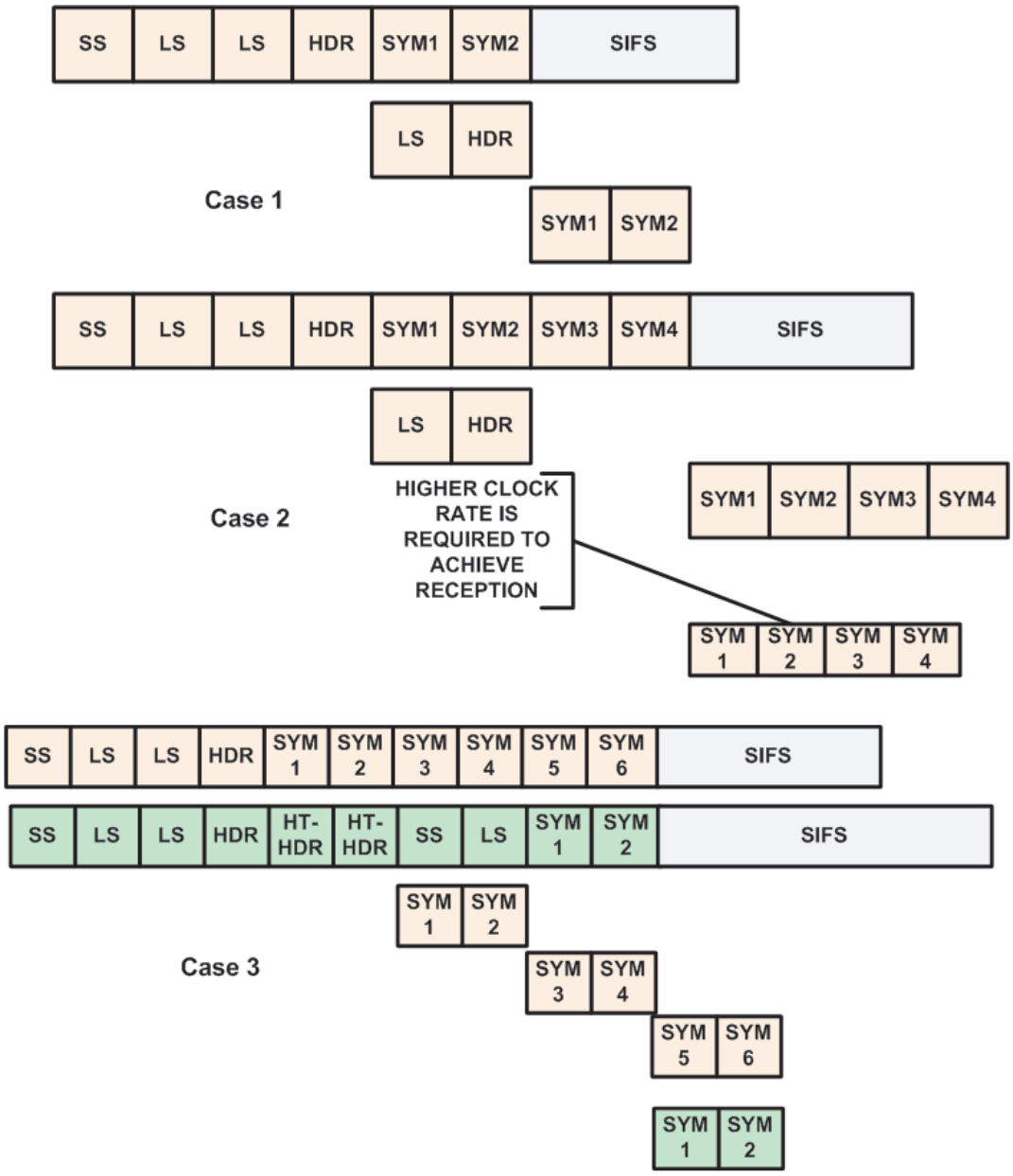

Figure 4.3: Diagram describing receiver model.

Now let us see understand each case shown in the Figure4.3.

1. ***Case 1*** : We have only one stream to process and we are processing two OFDM symbols in one chunk. As a result of this we are left with processing two OFDM symbols in SIFS period. We see that SIFS period is long enough to encompass the symbols. Hence, there would not be any problem in processing this kind of streams.

2. ***Case 2*** : We have only one stream to process and we are processing four OFDM symbols in one chunk. Since our chunk size is large, we need to first wait for symbols to get buffered and then process them. Also, we need to process these in only SIFS period. Hence, we have to run our hardware at a higher clock rate.

3. ***Case 3***: In this case, we have two streams (shown in green and pink color). As soon as we are done processing the first stream (802.11a pink colored), we have already buffered two ofdm symbols of 802.11a. We see that 802.11n has a longer header than 802.11a. As a result of which, while we haven't received 802.11n data and we are still processing 802.11n header, we can process 802.11a data. Since, at any time either of the two streams would be available, we can easily process the data. We can then assume that in the total time frame for transmitting both the streams, at any time we would be having *atleast* one chunk of either one of the streams. Also, since 802.11n stream has a longer *SIFS period*, we can easily prioritize streams during this period.

Conclusion: It is clear from the Figurethat when there are multiple streams, we can assume that there is a chunk available all the time. Hence, we need not wait for one stream to finish off its job. This is the basic idea behind this model. In other words, when we are processing stream A, data from stream B gets buffered and gets ready for processing during the next simulation cycle. This analysis would apply to multiple streams.

4.5 Usefulness of the simulator

The results of this simulator would help us in following ways :

- It would help us to estimate bounds on UCM Overhead.

- Also, it would help us in finding appropriate chunksize so that the architecture can support multiple streams.
- We can find the bounds on latency and required minimum clock rate.
- We can have rough estimate of complexity of architecture.

4.6 Limitations of the simulator

- This simulator is passive simulator. It doesn't calculate values after running the protocol. It works on values calculated for each chunk of individual protocols. Hence, estimation is done on the basis of pre-calculated values.
- Some of the estimates like gate-count are industry average approximation. True values have not yet been ascertained.
- UCM overhead has been calculated approximately by manual inspection of the present release. There is no information about this parameter. Hence, some simulations have been done for estimating the bounds on of this parameter. This fairly works okay because UCM overhead would be definitely less than individual PEs in terms of latency. However, gate count of UCM has not been estimated. This estimate would be available once the full implementation of programmable radio architecture is over.
- Place and routing estimation has not been covered in the simulations.

Chapter 5

EXPERIMENTAL RESULTS

In this section, we shall examine the simulation results. The simulator would be used to find

- To find the bounds of UCM Overhead.
- To illustrate the effects of different chunksizes on transmitter and receiver.
- To examine the complexity of different architectures of transmitter and receiver.
- To depict the effect of different chunksizes and framelength on effective power utilization.

5.1 UCM Overhead bounds

In this section we shall see the effect of UCM Overhead on the minimum clock rate for a given architecture and given framesize. This analysis would be used as an upper bound at the time of designing UCM for a given clock rate. Since, the UCM architecture has not been finalized yet, the results of this section would be useful for designing UCM architecture in the future.

5.1.1 UCM Overhead bounds for the transmitter

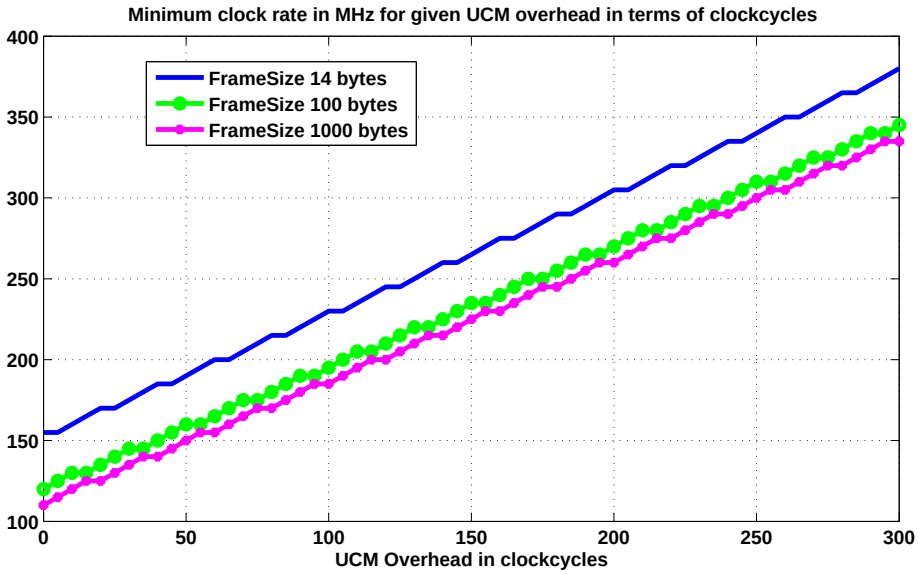

Figure 5.1: UCM Overhead bound for one stream input at the transmitter.

In the graph 5.1, we ran the simulation with chunk size equal to one and varying payload size. Only one stream of 802.11n with MCS equal to four was considered.

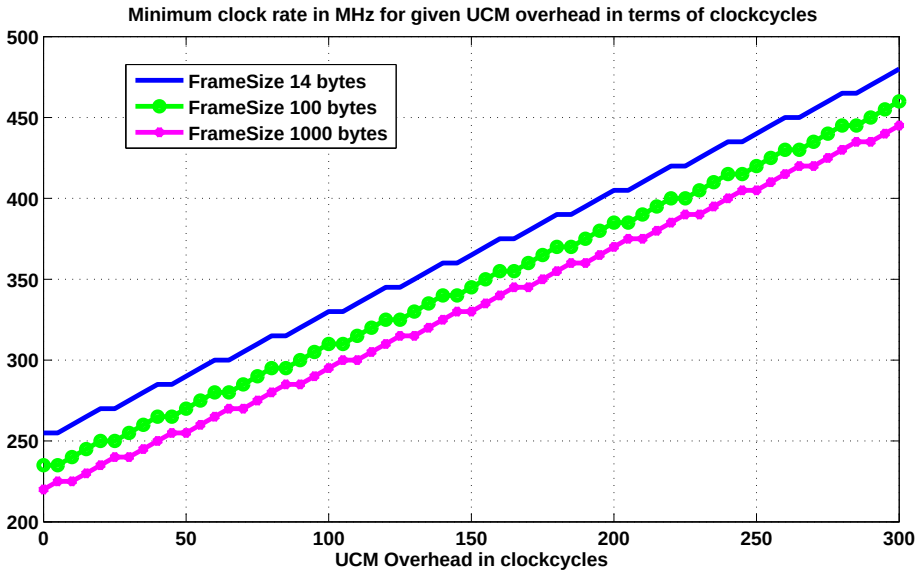

Figure 5.2: UCM Overhead bound for two stream input at the transmitter.

In the graph 5.2, we ran the simulation with chunk size equal to one and varying payload

size. Two streams of 802.11n and 802.11a with MCS equal to four was considered.

5.1.2 UCM Overhead bounds for the receiver

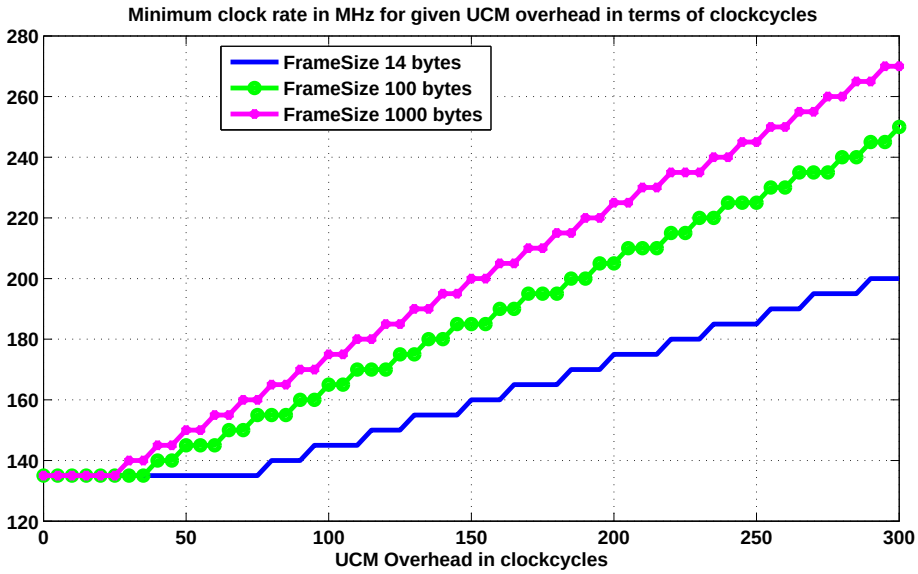

Figure 5.3: UCM Overhead bound for one stream input with chunksize equal to two at the receiver.

In the graph 5.3, we ran the simulation with chunk size equal to two ofdm symbols and varying payload size. Only one stream of 802.11n with MCS equal to four was considered.

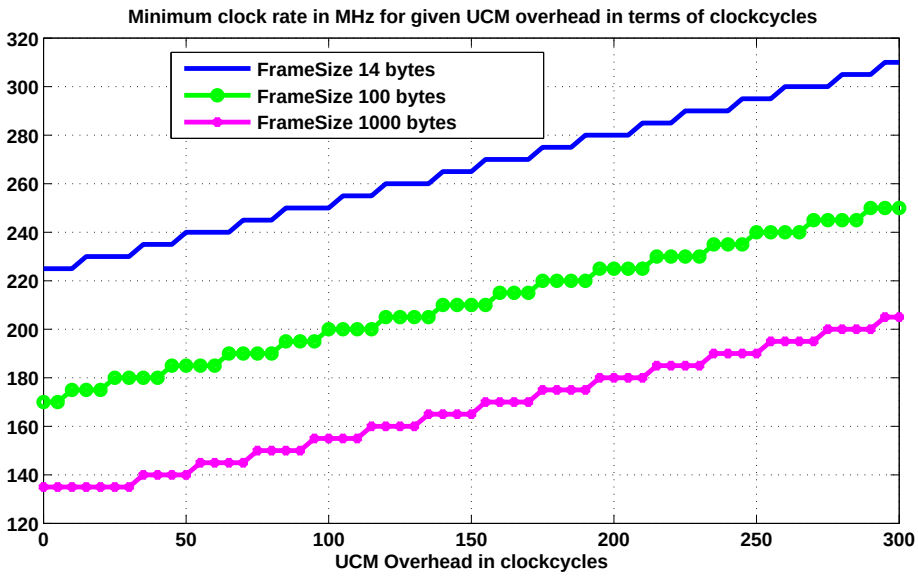

Figure 5.4: UCM Overhead bound for one stream input with chunksize equal to four at the receiver.

In the graph 5.4, we ran the simulation with chunk size equal to four ofdm symbols and varying payload size. Only one stream of 802.11n with MCS equal to four was considered.

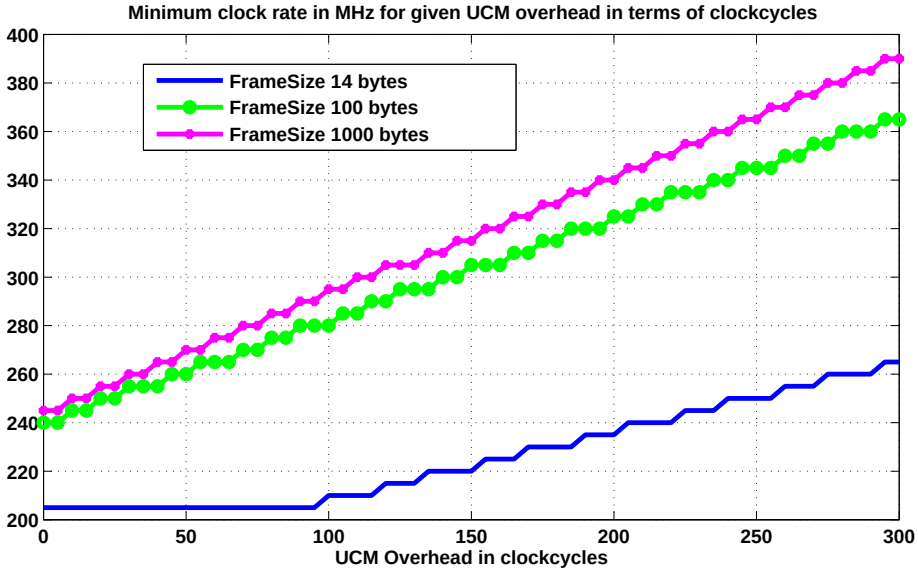

Figure 5.5: UCM Overhead bound for two stream input with chunksize equal to two at the receiver.

In the graph 5.5, we ran the simulation with chunk size equal to two ofdm symbols and varying payload size. Two streams of 802.11n and 802.11a with each MCS equal to four was

considered.

5.1.3 Conclusion

- We see that the graph of minimum clock rate versus UCM overhead is almost linear.
- For the transmitter, the slope is almost constant for all the payload size.
- For the receiver, the slope is high for higher payload size and low for smaller payload size. As a result of which, the UCM overhead should be as minimum as possible for higher throughputs.
- On the receiver side, as the payload size increases, the minimum clock rate increases. Hence, the UCM overhead should be less if we are operating with longer payload size.
- For supporting higher throughput, the UCM overhead should be small because of the linear relationship between the minimum clock rate and UCM overhead. *Hence, UCM architecture design is crucial for judging the throughput of programmable radio platform.*

5.2 Effects of Chunksize on Transmitter and Receiver design.

In this section we shall investigate the effect of changing chunksize on streams with different payload lengths. We shall also investigate the effect when there are multiple streams.

5.2.1 Effect of Chunksize on Transmitter Design

The effect of chunksize on transmitter is shown in Table 5.1. We note here that higher chunksize is not possible because 14 byte data fits in only one chunk.

Table 5.1: Typical minimum clock rates for the given chunksize and architecture with Payload size of 14 bytes.

Stream	**Chunksize**	**Minimum clock rate**
One 802.11n with MCS 4	1	190
Two 802.11n with MCS 4 each	1	290
802.11n and 802.11a with MCS 4 each	1	290

Now when the payload size increases, the effect of chunksize on minimum clock rate is demonstrated by Figures (5.6-5.7)

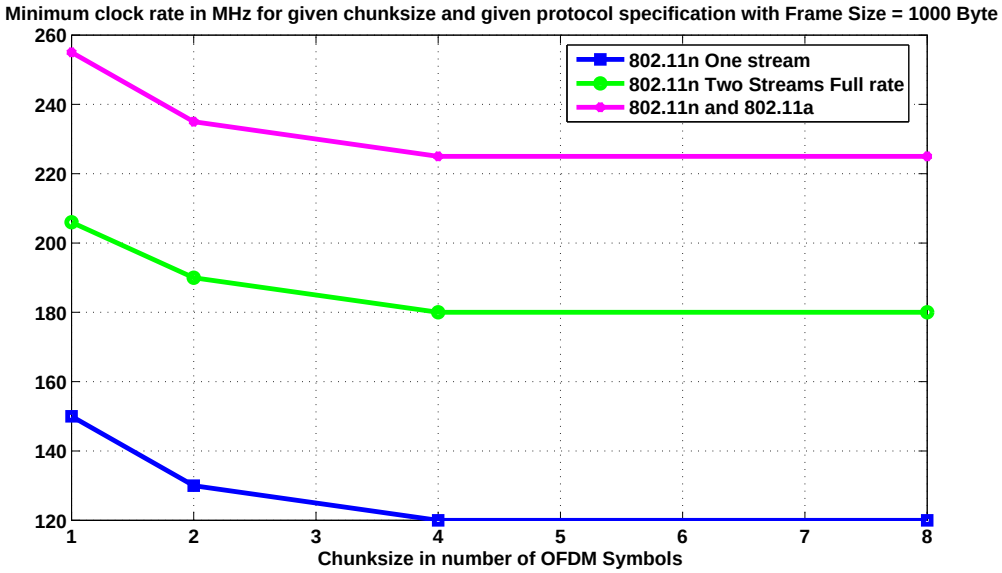

Figure 5.6: Effect of chunksize on minimum clock rate for payload size of 1000 bytes.

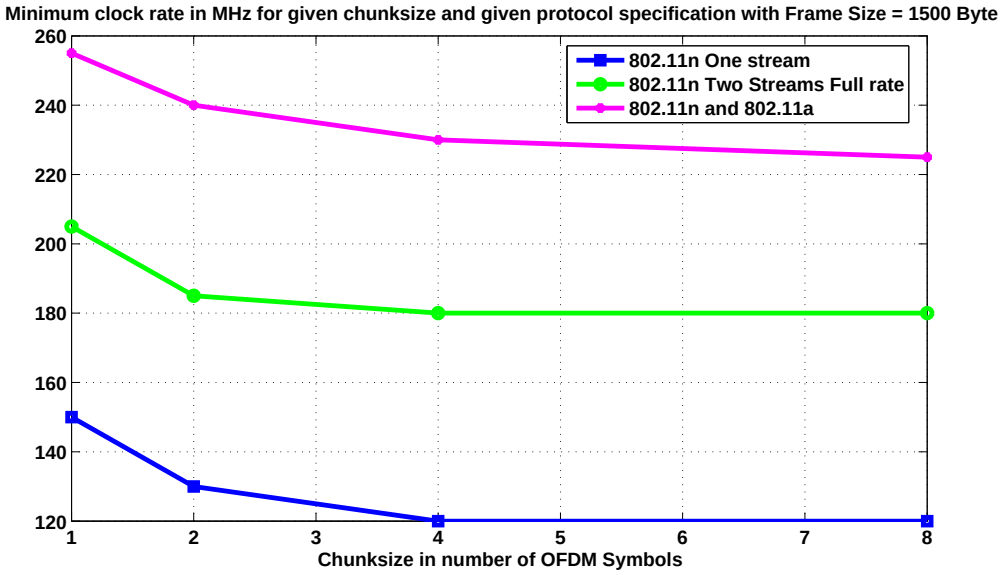

Figure 5.7: Effect of chunksize on minimum clock rate for payload size of 1500 bytes.

5.2.2 Analysis of Chunksize on Transmitter Design

From the figures (5.6-5.7) and Table 5.1, it is clear that we need to run the whole system at a higher clock rate if we wish to support higher data rate and multiple streams. Also, when the payload size is large, it is better to run the system with higher chunksize. This would have an added advantage of decreasing the minimum clock rate and improving the utilization of the PEs. Table 5.2 demonstrates the utilization of one of the PEs decreases as the chunksize increases. Hence, even if the minimum clock rate is same for different chunksize, the effective utilization of the PE is different. Although there is only a marginal difference (approximately 5-10% per chunk), this difference is considerably large when we consider a long stream. Also, from the Table 5.1, it is clear that when the payload size is small, it is better to operate the system with smaller chunksize. This is true because the added latency per chunksize decreases. in both cases (i.e. when the chunksize is small for smaller payloadsize and when the chunksize is large for a larger payloadsize)

Table 5.2: PE utilization of IFFT Block as a function of chunksize for payloadsize corresponding to Figure 5.6

Chunksize	**Minimum clock rate**	**PE Utilization in Percentage**
150	1	65.6746 %
130	2	41.8568 %
120	4	29.8507 %
120	8	23.6326 %

Also, when a chunk is processed by a PE, there are two kinds of latency that is involved :

- Θ is the latency required by each PE for initial setup. This latency doesn't depend on the chunksize. This is the minimum initial delay required to process the data. This can be thought of as initial delay of the FFT.

- γ is the latency required by each PE depending upon the chunksize. This could be the number of clockcycles required to read the chunk or write the chunk. This could even be processing time depending upon the chunksize.

The total latency is shown by the equation 5.1.

$$\tau = \Theta + \gamma \tag{5.1}$$

We see that when the chunksize increases, γ increases as compared to θ. This helps us to run the system at a lower clock rate because τ becomes approximately equal to γ. However, when the chunksize decreases, θ becomes comparable to γ. Also, θ is unavoidable. Hence, the total latency becomes higher. Now when we consider smaller payload, we cannot increase γ by using larger chunks because of the tight time constraints imposed to transmit an OFDM symbol. The only option is to decrease the chunksize and hence reduce γ.

5.2.3 Effect of Chunksize on Receiver Design

The effect of Chunksize is demonstrated by figures 5.8-5.11.

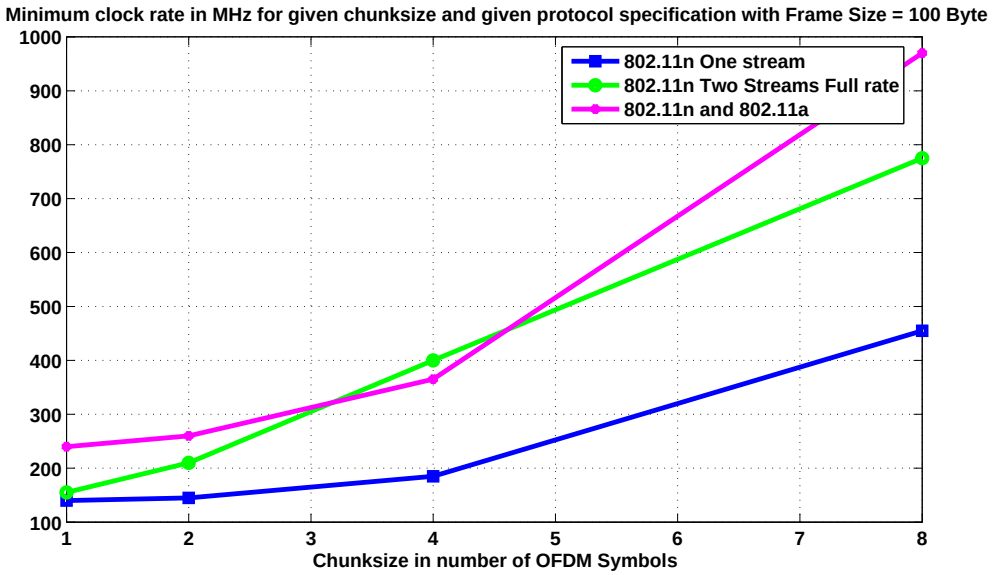

Figure 5.8: Effect of chunksize on minimum clock rate for payload size of 100 bytes.

In all the figures 5.8-5.11, three independent streams were considered :

- Only one 802.11n stream.
- One 802.11n stream and one 802.11a stream.
- Two 802.11n stream (two spatial streams.)

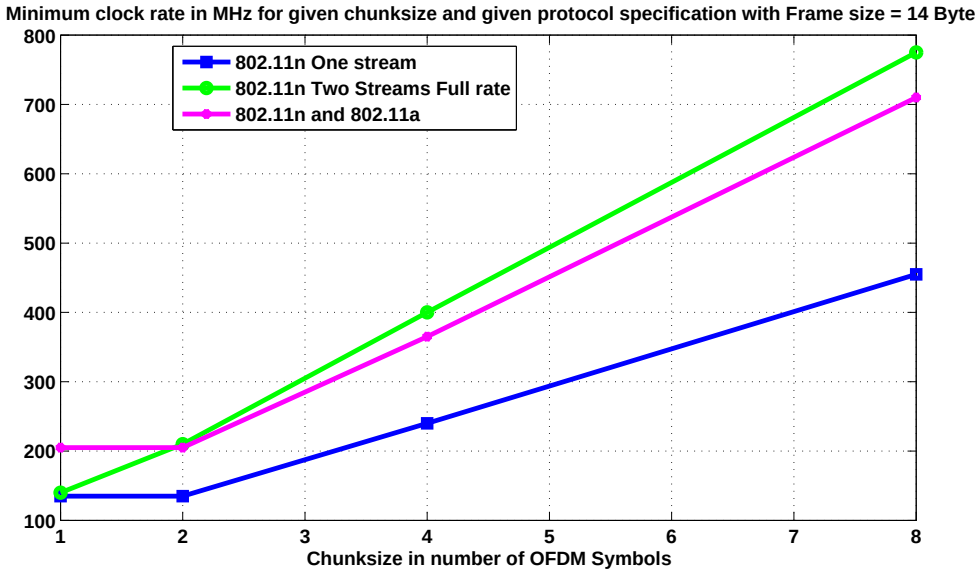

Figure 5.9: Effect of chunksize on minimum clock rate for payload size of 14 bytes.

In all the figures 5.8-5.11, the simulations were done using MCS 4 and chunksize was varied as one, two, four and eight.

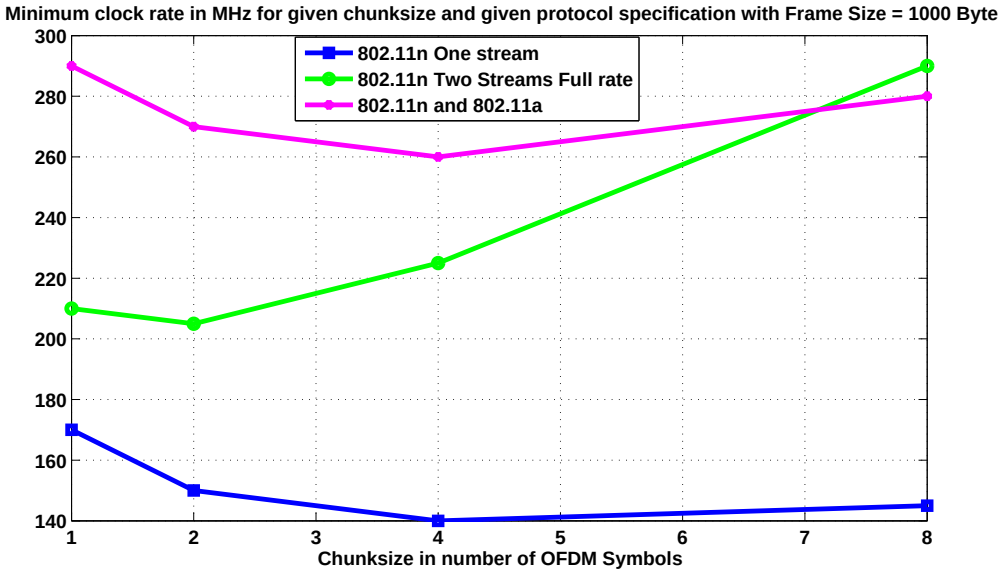

Figure 5.10: Effect of chunksize on minimum clock rate for payload size of 1000 bytes.

In all the figures 5.8-5.11, the UCM overhead was kept at 50 clockcycles.

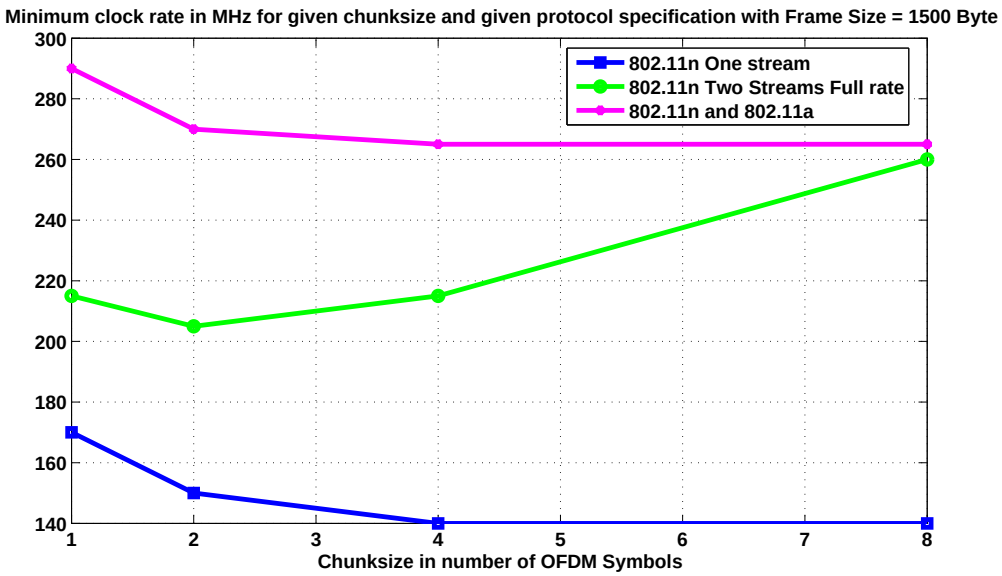

Figure 5.11: Effect of chunksize on minimum clock rate for payload size of 1500 bytes.

The architectural complexities corresponding to figures 5.8-5.11 have been analyzed in section 5.3.

5.2.4 Analysis of Chunksize on Receiver Design

The effect of chunksize is quite similar to that explained in subsection 5.2.2. The only difference is that for smaller payloadsize, we assume that the data is padded with zeros such that the data fits into the chunks. This is also possible at the transmitter. This has not been analyzed because this costs unnecessary power consumption because of padded zeros. However, at the receiver, we need to analyze it because we may have a frame with smaller payload and padded zeros. The results obtained for the receiver are quite similar to that of transmitter. There are few things that are worth noting from the Figure5.8-5.11.

Firstly, the minimum clock rate for two streams consisting of 802.11a and 802.11n is higher than that of two spatial streams of 802.11n (for higher payload size). This can be attributed to the fact that the number of bits in two 802.11n streams is higher than that of 802.11a and 802.11n streams for a given MCS because there are more number of subcarriers in 802.11n. Since the number of bits are higher, we get lesser latency per chunk i.e. θ in equation 5.1. Hence, the effective utilization of a chunk is higher. This doesn't mean that we should have

large chunksize because after certain point (called as the *knee-value*), the minimum clock rate doesn't decrease much. The effective utilization of the individual PEs decreases. Also, for smaller payload size, the effect is reverse of what has been explained. This is clear from the graphs 5.9 and 5.11.

There is one more concept worth noting. If we analyze Figure5.11, we see that when we have two spatial streams of 802.11n, the clock rate increases as the chunksize increases. This is true because in the SIFS interval, we have to process two 8 OFDM symbols (since we have two streams). Hence, the overhead increases by twofold. As a result of which, the minimum clock rate increases. Also, when the chunksize is small, the minimum clock rate increases due to the effect highlighted above.

5.2.5 Conclusion

Thus, when the payloadsize is large, we should use larger chunksize and when the payloadsize is small, smaller chunksize is advisable for transmitter and receiver. The minimum clock rate required to run the hardware would be less than that of different chunksize respectively.

5.3 Complexities of different architecture

In this section we shall present the total gate count for different architectures analyzed before. These gatecounts have been presented in Table 5.3 and Table 5.4.

Table 5.3: Approximate gatecount of TX design for different architectures of Table 5.1.

Stream	**Gatecount**
One 802.11n with MCS 4	49800
Two 802.11n with MCS 4 each	60600
802.11n and 802.11a with MCS 4 each	49800

Table 5.4: Approximate gatecount of RX design for different architectures of Table 5.1.

Stream	**Gatecount**
One 802.11n with MCS 4	750800
Two 802.11n with MCS 4 each	761600
802.11n and 802.11a with MCS 4 each	750800

The InCyte Lite tool [1] and [21], [22] was used to estimate the gatecount of individual PEs. The gatecount of modulator block was found by mutiliplying the total number of CLB on Xilinx FPGA for 802.11a by a factor of 10. These estimates are approximate and not accurate. The basic goal of finding these estimates is to have a rough idea about the area of the ASIC chip and its power utilization. These numbers do not include the gatecount of UCM block. This block is still under research and development phase. Hence, no results are available for this block.

Individual gatecounts of the transmitter and receiver have been highlighted in Table5.5.

Table 5.5: Approximate gatecount of Transmitter and Receiver.

Transceiver Block	**Gatecount**
Channel Equalizer	351000
Demodulator	800
Deinterleaver	10000
Viterbi Decoder	20000
FCS	35000
UCM	30000
Synchronizer	334000
Encoder	20000
Modulator	1600
IFFT/FFT	30000

5.4 Analysis of Complexities of different architectures

We see that the first and the third row of the Table5.4 and Table5.3 is the same because we are *reusing* the blocks for different streams. On the other hand, in order to accomodate two spatial processing we have to have additional hardware. This adds up to the complexity which is shown by second row of tables 5.4 5.3.

5.5 Study of power utilization of different architectures

In order to study the power utilization of different architectures, we would define a parameter α and name it as Performance factor (equation 5.3). The actual equation governing the power utilization is shown in 5.2 (equation 18.10 of [23]). For estimating the exact power utilization,

[1] InCyte is a registered trademark of ChipEstimate.com

we can multiply α by a number depending on fabrication technology and chip company i.e. $Energy_{gate}$, C_{load} and E_{gate}.

$$Power = fN(Energy_{gate} + 0.5V^2 C_{load})E_{gate} \tag{5.2}$$

where

- $Energy_{gate}$ is the average internal consumption for an equivalent gate (includes parasitic capacitance as well as short circuit currents [23]).
- C_{load} is the average capacitive load for an equivalent gate.
- E_{gate} is the average output activity for an equivalent gate per cycle.
- *N* is the gate equivalent count for the component.
- f is the clock rate or frequency of operation.
- *V* is the voltage of the *ASIC* circuit.

$$\alpha = clockrate * \{\sum_{i=1}^{\Xi}(Utilization_i)(Gatecount_i)\} \tag{5.3}$$

where

- Ξ is the total number of PE
- *Utilization* is the utilization of PE on the scale 0 to 1.
- *Gatecount* is the approximate gatecount of the PE corresponding to the desired configuration.

The results for the receiver have been shown in Table 5.7.

Power has been analyzed for both transmitter and receiver based on complexity of the hardware. The results for the transmitter have been shown in Table 5.6. There is one concept worth noting. When the payload size is small enough to fit in just one OFDM symbol, there is no added benefit in zero padding additional OFDM symbols. Hence, first row of each of the streams in Table 5.6 is left blank. At the receiver, we have analyzed OFDM symbol sizes two,

Table 5.6: Transmission chunksize for minimum power dissipation at transmitter

Streams	***Payload Size***	***Optimal Chunksize***	***Savings by factor***	***Optimal Chunksize compared with chunksize***
802.11n one stream	14	1	-	-
	100	4	1.8698	4 compared with 8
	1500	8	2.3027	8 compared to 1
802.11n two streams	14	1	-	-
	100	2	1.264	2 compared with 8
	1500	8	1.617	8 compared with 1
802.11n and 802.11a	14	1	-	-
	100	4	1.549	4 compared with 1
	1500	8	2.101	8 compared with 8

Table 5.7: Transmission chunksize for minimum power dissipation at receiver

Streams	***Payload Size***	***Optimal Chunksize***	***Savings by factor***	***Optimal Chunksize compared with chunksize***
802.11n one stream	14	1	15.4127	1 compared with 8
	100	1	88.83	1 compared with 8
	1500	8	1.2344	8 compared with 1
802.11n two streams	14	1	5.528	1 compared with 8
	100	1	8.8967	1 compared with 8
	1500	4	1.0063	4 compared with 1
802.11n and 802.11a	14	1	4.3	1 compared with 8
	100	1	4.68	1 compared with 8
	1500	8	1.1336	8 compared with 1

four and eight for payload size 14. The basic idea was to see the effect of padding additional symbols at the receiver when the payload size is small. The results are self explanatory. Zero padding doesn't benefit at all and is disastrous. The power dissipation rises by the factor of five to eighty. This is definitely not desirable. Similar analogy can be applied to the case of 802.11n two streams with datasize 1500 (sixth row of Table5.7). We also note that, *savings by factor* is estimated using the worst case OFDM symbol size, i.e. using four to eight OFDM symbols when payload size is small and vice versa.

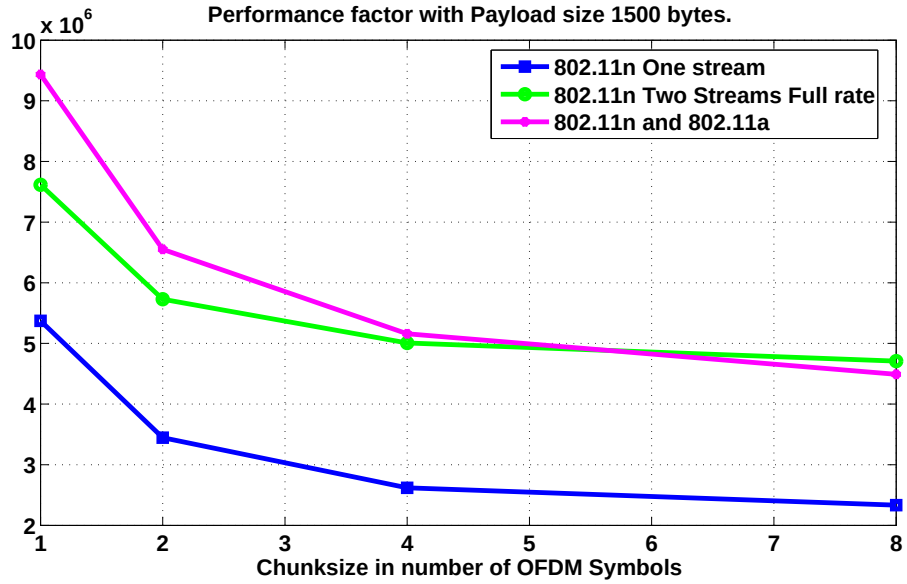

Figure 5.12: Performance factor (α) vs chunksize at the transmitter with payloadsize equal to 1500 bytes.

Three graphs have been shown for investigating the variation of α with chunksize. Figures 5.12 is straightforward. There exists a linear relationship between Figure 5.12 and 5.7.

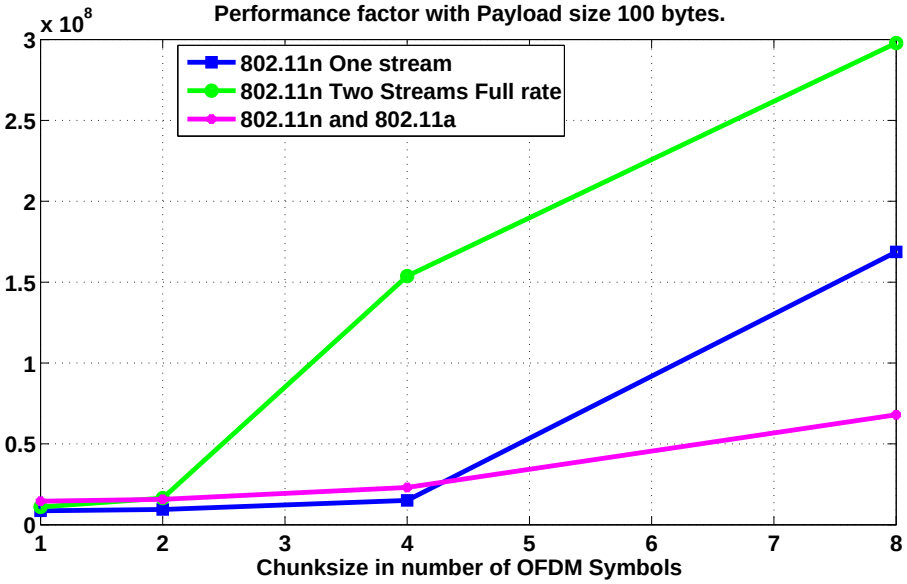

Figure 5.13: Performance factor (α) vs chunksize at the receiver with payloadsize equal to 100 bytes.

Also, in the Figure5.13, any value of chunksize greater than two is not feasible because of very high clockrate except for only one 802.11n stream. Also, a higher value of clockrate for

two 802.11n streams could be attributed to large gatecount.

However, there is an important concept in the Figure5.14. We see that as the chunksize increases to eight, the power dissipation doesn't decrease as anticipated. The reason is that the minimum clockrate for chunksize eight either increases (for two streams of 802.11n) or remains constant (for two stream of 802.11n and 802.11a and one stream of 802.11n; refer Figure 5.11) as compared to chunksize of four for receiver with payloadsize 1500 bytes. Also, the gatecount of two streams of 802.11n is greater than the other two data streams. This justifies the drastic increase of power for chunksize eight and knee value of four OFDM symbols.

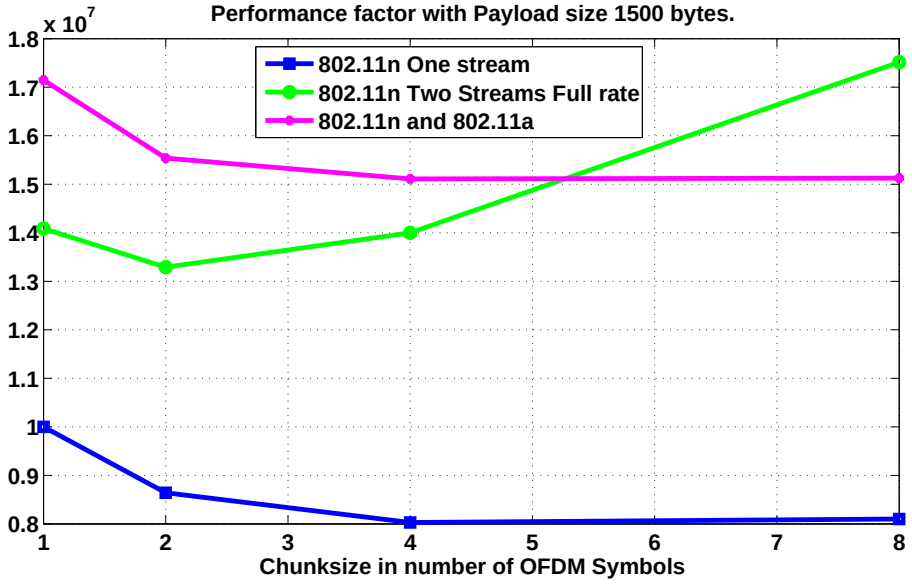

Figure 5.14: Performance factor (α) vs chunksize at the receiver with payloadsize equal to 1500 bytes.

5.6 PE Utilization

From the graph 5.15, it is clear that bottleneck of the system is demodulator PE at the receiver. Demodulator PE at the receiver includes interleaver and stream combiner which operate serially. Hence, the utilization of PE doesn't improve even if we increase the chunksize.

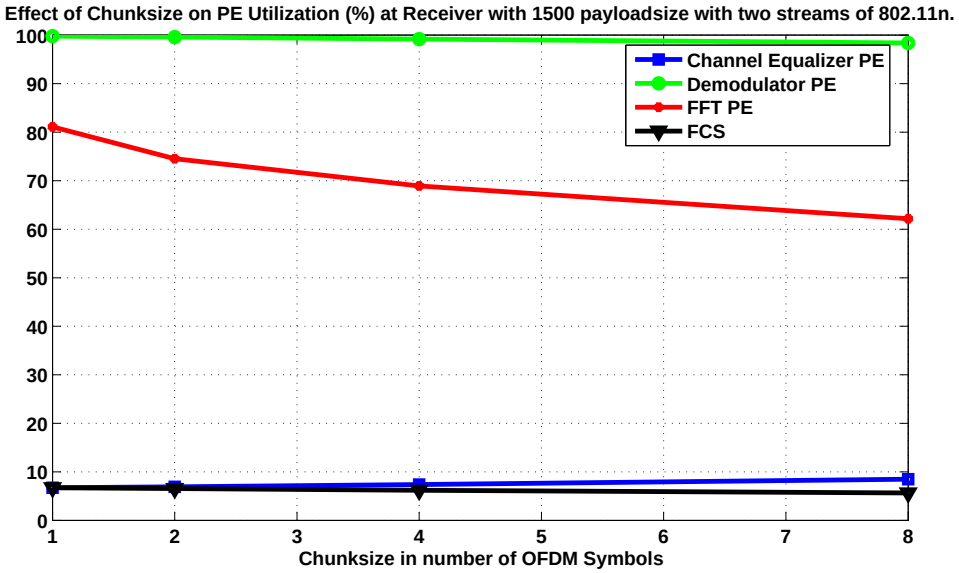

Figure 5.15: Variation of PE utilization with respect to chunksize at the Receiver with one 802.11n stream and payloadsize equal to 1500 bytes.

From the graph 5.16, it is clear that bottleneck of the system is encoder PE for the transmitter. Encoder PE at the transmitter includes interleaver, convolutional coder and stream parser. These blocks operate serially. Hence, it is advisable to go for parallel architectures for these blocks even if the gate count increases.

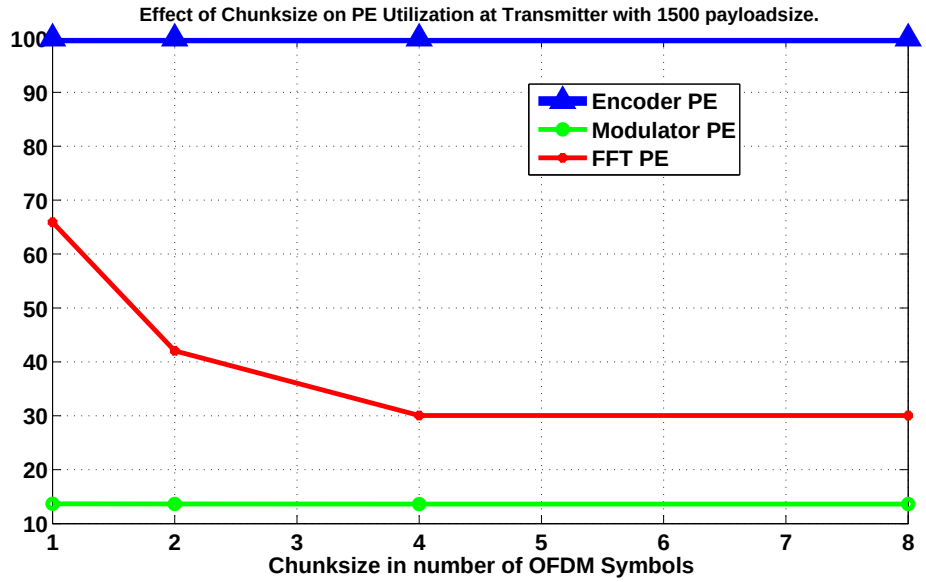

Figure 5.16: Variation of PE utilization with respect to chunksize at the Transmitter with two 802.11n stream and payloadsize equal to 1500 bytes.

5.7 Implementation of common blocks for PE

There are three basic blocks which were being simulated and implemented using VHDL. The block diagram showing the connection of these block is shown in Figure 5.17 . The design of these blocks were done by WiNC2R team. Only implementation of these blocks was done in this thesis.

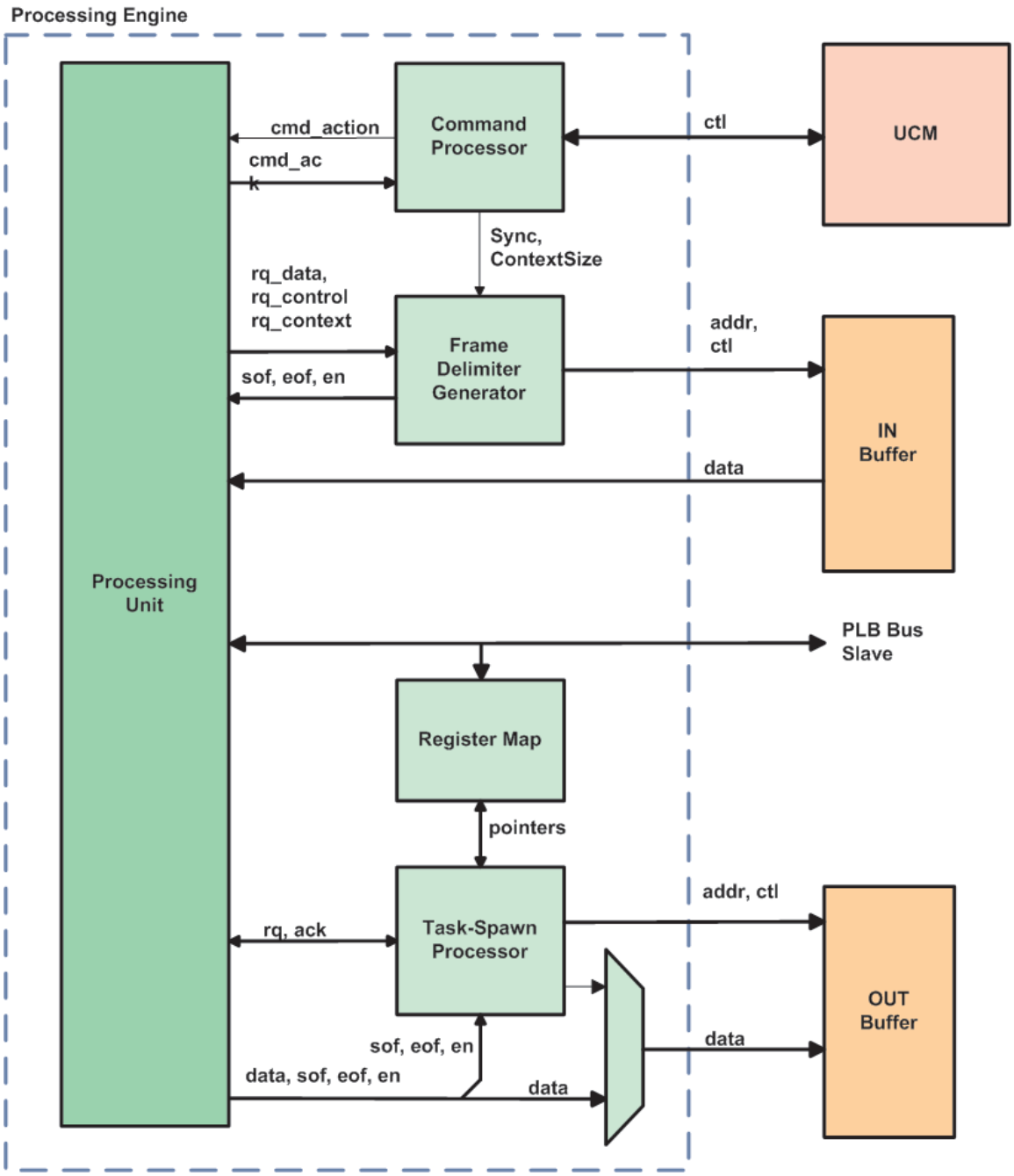

Figure 5.17: Top level diagram of common blocks [2].

The general PE Architecture consists of PU, CP, FDG, TSP and RMAP. Only CP and FDG would be discussed in this thesis.

The basic functionality of CP is to

- Provide interfacing to UCM
- Decode UCM commands (optional)
- setup PE common modules in preparation for command processing
- terminate current command cycle

The basic functionality of FDG is to

- Fetch data from Input Buffer
- Generate sof, eof and enb control signals for data synchronization

We note that sof, eof and enb signals help trigger data processing and synchronization. 'sof' goes high everytime there is a valid data frame for PE. Similarly, 'eof' goes high everytime last byte of data frame is read. 'enb' signal helps to know how many bytes are valid out of 32-bit dataword.

5.7.1 CP Block

The state machine diagram of CP block is shown in Figure 5.18. The idea behind this block is to decode a specific command and then give out a particular action signal to PU. The design of this block was done by WiNC2R team [2]. Only implementation was done in this thesis.

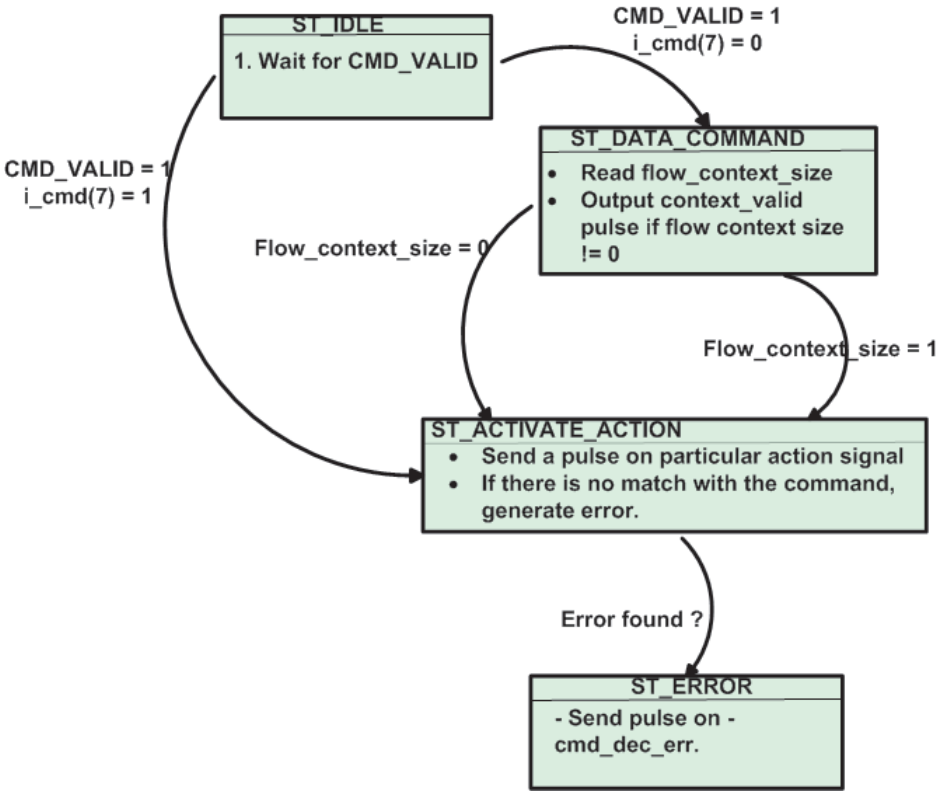

Figure 5.18: State Machine of Command processor.

5.7.2 FDG Block

The state machine diagram of the FDG block is shown in Figure 5.19. The idea behind this block is to be able to generate control signals alongwith the data. This would facilitate us to process the data in a better way.

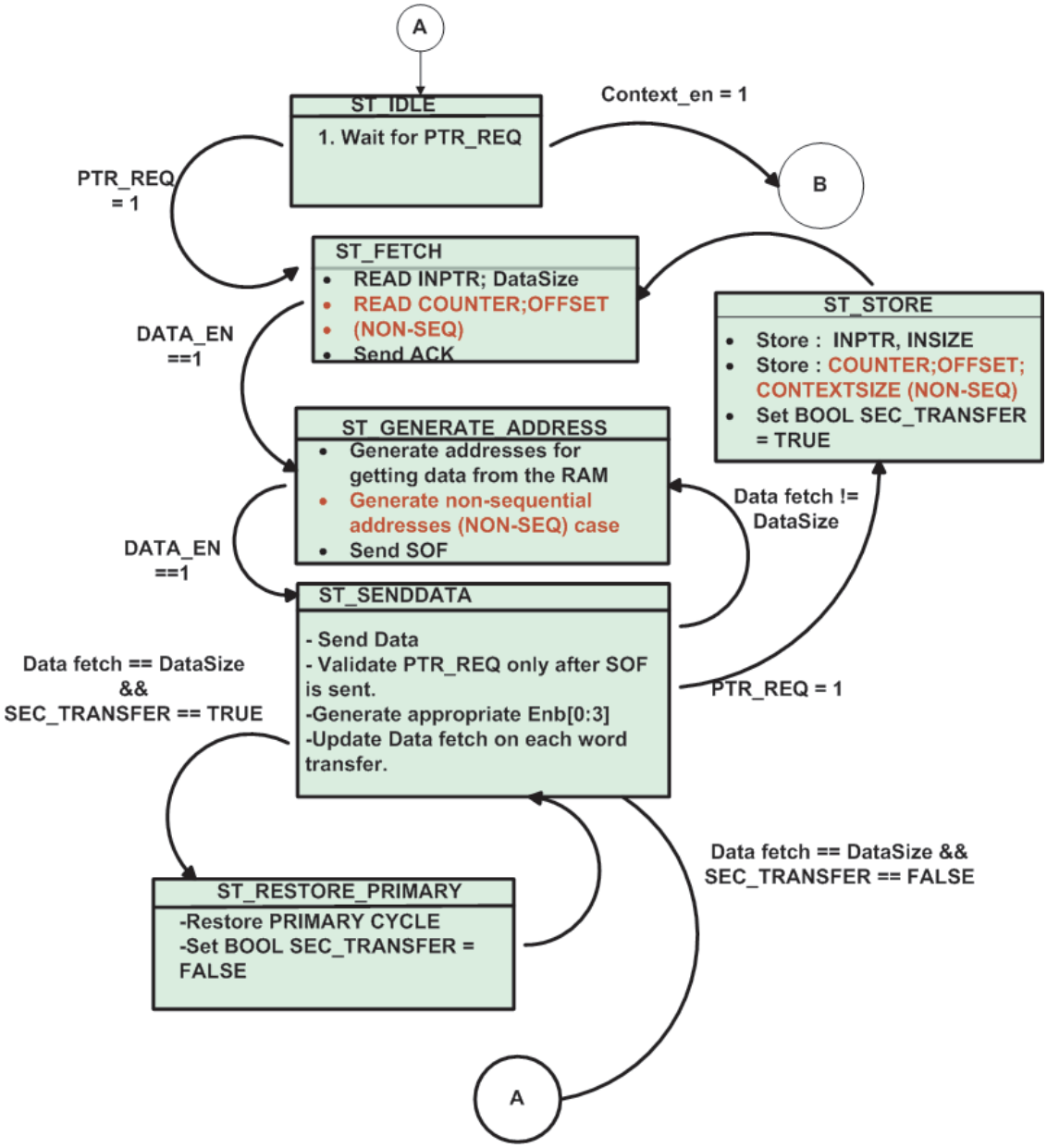

Figure 5.19: State Machine of FDG.

The state machine diagram of the context cycle of FDG block is shown in Figure 5.20. The idea behind this block is to be able to send contextual data when another command is in progress. Contextual data would be used to maintain state information between different streams.

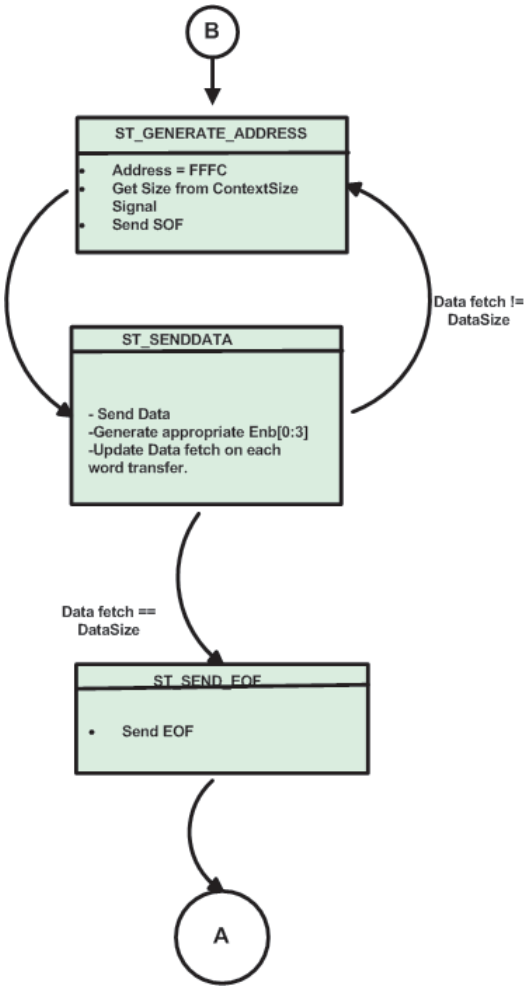

Figure 5.20: State Machine of context cycle of FDG.

5.7.3 FDG requirements

The FDG was required to support the following scenarios. Few of these requirements are actually error conditions meant to make the unit robust.

1. Sequential and non throttled cycle
2. Sequential and throttled cycle
3. Sequential, secondary and non throttled cycle
4. Non sequential and non throttled cycle
5. Non sequential and throttled cycle

6. Non sequential, secondary and non throttled cycle
7. Contextual cycle
8. Single word transfer
9. Sequential, secondary and throttled cycle
10. Non sequential, secondary and throttled cycle
11. Sequential and errorneous secondary cycle
12. Non sequential and errorneous secondary cycle
13. Non sequential and one word transfer
14. Non sequential and contextual transfer

The above terms have been defined as :

- ***Throttled Cycle*** : This means that the data transfer would be throttled i.e. transfer would be stalled for sometime and then resumed again. The FDG block would maintain state information about the transfer and hence resume it anytime.
- ***Secondary Cycle*** : This means that the FDG block would cater to another stream when one stream is already in progress. It is the job of FDG block to handle all the transfer and maintain the state information. The FDG block would also save the current state of the stream before switching to the new stream. As for e.g. when there is a data cycle already in progress, another control command could interrupt the data cycle (a.k.a. primary cycle). This control cycle would be termed as secondary cycle.
- ***Contextual Cycle*** : This cycle is used for transferring state information of one stream to the block. This would help to cater to high priority tasks.
- *Single word transfer* : Since one word could be either 16-bit or 32-bit, this cycle would transfer only one word and go back to the idle state.
- ***Sequential Cycle*** : In this the data would be read from contiguous locations of the memory.

- ***Non-sequential Cycle*** : In this cycle, the data would not be read from the contiguous locations but from the locations defined by the control signals. This would be helpful for removing the guard interval from the memory by changing the read and write locations.

- ***Errorneous cycle*** : These cycles have been designed and tested to make this block full proof. Since FDG is a common block, it is very essential to make this error free.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

- We should use different chunksizes for different payload size. Hence, our Chunksize should be reconfigurable for one frame.
- We should keep the UCM overhead as small as possible if we want to operate on streams with higher throughput.
- Parallel architecture should be used for Encoder PE at the transmitter.
- Channel estimation and MIMO detection block should be optimized to reduce the complexity at the receiver. Further, a common architecture should be designed for synchronizers of both the streams. This would help us to scale down power requirements.

6.2 Future Work

The future work could incorporate simulation of MAC layer in conjunction with the PHY layer. This would help us to have a bigger picture of the platform in terms of complexity, latency and UCM overhead. Also, presently there is no mechanism to monitor queue size. If this is incorporated, the minimum clock rate could be reduced further at an additional cost of queueing overhead. Hence, a good design would be to have variable length queues.

Further, a good analysis of this architecture would be to have dynamic simulator of the architecture which encompasses functionality to test different traffic models. It should also be able to simulate MAC layer.

References

[1] T. by His Divine Grace A.C. Bhaktivedanta Swami Prabhupada, "Bhagavad - Gita As It Is," e-book www.krishna.com.

[2] Z. Miljanic, "WiNC2R research team," WINLAB, Rutgers University.

[3] Z. Miljanic, I. Seskar, K. Le, and D. Raychaudhuri, "WINLAB Network Centric Cognitive Radio Hardware Platform - WiNC2R," in *Proceedings of International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, Orlando, Florida, 2007.

[4] "IEEE 802.11a Physical and MAC Layer Specifications," IEEE 802.11a, 1999.

[5] "IEEE 802.11n Draft 3.02 Physical and MAC Layer Specifications," IEEE 802.11n D3.02, 2007.

[6] V. Tarokh, H. Jafarkhani, and A. R. Calderbank, "Space-time block codes from orthogonal designs," *IEEE Transactions of Information Theory*, vol. 45, no. 5, pp. 1456–1467, 1999.

[7] S. M. Alamouti, "A simple transmit diversity technique for wireless communications," *IEEE Journal Selected Areas of Communications*, vol. 16, no. 8, pp. 1451–1458, 1998.

[8] G. Bauch and J. Malik, "Cyclic delay diversity with bit-interleaved coded modulation in orthogonal frequency division multiple access," *IEEE Transactions on Wireless Communications*, vol. 5, no. 8, pp. 2092–2100, 2006.

[9] M. Bossert, A. Huebner, F. Schuehlein, H. Haas, and E. Costa, "On Cyclic Delay Diversity in OFDM Based Transmission Schemes," in *Proceedings of International OFDM Workshop*, Hamburg, Germany, 2002.

[10] W. J. Hillery, T. Krauss, B. Mondal, T. Thomas, and F. Cook, "Finite Impulse Response Cyclic Shift Transmit Diversity for Broadband Mobile OFDM," in *Proceedings of IEEE Vehicular Technology Conference*, Baltimore, Maryland, 2007, pp. 511–515.

[11] A. R. Bahai and B. R. Saltzberg, *Multi-Carrier Digital Communications: Theory and Applications of OFDM*. New York: Kluwer Academic Publishers, 1999.

[12] T. Schmidl and D. Cox, " Robust frequency and timing synchronization for OFDM," *IEEE Transactions on Communications*, vol. 45, no. 12, pp. 1613–1621, 1997.

[13] G. L. Stuber, J. R. Barry, S. W. Mclaughlin, Y. G. Li, M. A. Ingram, and T. G. Pratt, "Broadband MIMO-OFDM Wireless Communications," *Proceedings of the IEEE*, vol. 92, no. 2, pp. 271–294, 2004.

[14] Y. Li, "Simplified Channel Estimation for OFDM Systems With Multiple Transmit Antennas," *IEEE Transactions on Wireless Communications*, vol. 1, no. 1, pp. 67–75, 2002.

[15] G. L. Stuber, J. R. Barry, S. W. Mclaughlin, Y. Li, M. A. Ingram, and T. G. Pratt, "Channel Estimation for OFDM Systems with Transmitter Diversity in Mobile Wireless Channels," *IEEE Journal on selected areas in Communications*, vol. 17, no. 3, pp. 461–471, 1999.

[16] L. Zhou and M. Nakamura, "Channel estimation of multiple transmit antennas for OFDM systems with cyclic delay preamble," in *Proceedings of Vehicular Technology Conference*, Berlin Germany, 2005, pp. 583–587.

[17] H. Bolcskei and A. J. Paulraj, *Multiple-input multiple-output (MIMO) wireless systems*. CRC Press, 2002.

[18] Y. Yapici, "V-BLAST/MAP: A New Symbol Detection Algorithm For Mimo Channels," Master of Science Thesis, Bilkent university, January 2005.

[19] G. Golub and C. Loan, "Matrix Computations," John Hopkins University Press, Baltimore, 1983.

[20] X. D. Board, "Xilinx Coregenerator Block - Multiplier, Adder and Cordic," LogiCORE Multiplier Generator v9.0.

[21] W. Xuejing, L. Liang, Y. Fan, R. Junyan, and H. Bo, "A novel synchronizer for ofdm-based uwb system on new preamble design," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Athens, Greece, 2007, pp. 1–5.

[22] A. Burg, S. Haene, D. Perels, P. Luethi, N. Felber, and W. Fichtner, "Algorithm and vlsi architecture for linear mmse detection in mimo-ofdm systems," in *IEEE International Symposium on Circuits and Systems*, Kos, Greece, May 2006, pp. 4102–4105.

[23] W.-K. Chen, "The VLSI Handbook," CRC Press LLC, Florida, 2000.

[24] G. C. Clark and J. B. Cain, "Error-Correction Coding for Digital Communications," Plenum Press, New York, August 1982.

[25] C. Langton, "Intuitive Guide to Principles of Communication," http://www.complextoreal.com/chapters/convo.pdf.

[26] G. Feygin and P. Gulak, "Survivor sequence memory management in viterbi decoders," in *IEEE International Sympoisum on Circuits and Systems*, Singapore, 1991, pp. 2967–2970.

[27] F. Angarita, M. J. Canet, T. Sansaloni, and J. Valls, "Architectures for the Implementation of a OFDM-WLAN Viterbi Decoder," *Springerlink*, vol. 52, no. 1, pp. 35–44, 2008.

[28] K. Chadha and J. R. Cavallaro, "A reconfigurable viterbi decoder architecture," in *Proceedings Of Asilomar conference on signals, systems and computers.*, California, 2001, pp. 66–71.

[29] H.-L. Lou, "Linear distances as branch metrics for viterbi decoding of trelliscodes," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Istanbul, Turkey, 2000, pp. 3267–3270.

Appendix A

Algorithm: Branch Metric Table Mapper.

Preconditions: $BM_0 - BM_3$ is available.

PostConditions :

Branch_Metric(i) where $i = 0 \rightarrow 63$ contains 2 bit branch metric indicating $BM_0(00)$, $BM_1(01)$, $BM_2(10)$ and $BM_3(11)$

Symbols:

χ : bit wise multiplication.

θ : Parity bit of an array

& : Concatenation

$\leftarrow$: Left hand side takes the value of Right hand side.

Ω : Binary Value

κ : Constraint Length of Convolutional code

Pseudocode:

$G0 \leftarrow 133$;

$G1 \leftarrow 171$;

for i in 0 to $2^{\kappa-1}$

do $b_0 \leftarrow \theta(\Omega(i))\chi(\Omega(G_0))$;

do $b_1 \leftarrow \theta(\Omega(i))\chi(\Omega(G_1))$;

$Branch\_Metric(i) \leftarrow b_1 \& b_0$;

step;

Appendix B

A sample command has been shown in Table

Table B.1: Typical command specification of TxDataAvl command

Bits	**Description**
$b_0 - b_3$	MCS
$b_4 - b_5$	Standard ID
$b_6 - b_{21}$	Frame Length
$b_{22} - b_{23}$	Channel ID
$b_{24} - b_{25}$	Channel Bandwidth
$b_{26} - b_{27}$	Channel Offset
b_{28}	Format
b_{29}	Number of Streams
b_{30}	GI
b_{31}	Coding Scheme Used

MCS values have been described in Table 2.2

Table B.2: Channel offset description

Channel Offset	**Description**
00	CH_OFF_20U [1]
01	CH_OFF_20L [1]
10	CH_OFF_40 [1]
11	NOT USED

Table B.3: Channel bandwidth description

Channel Bandwidth	**Description**
00	NON_HT_CBW40 [1]
01	NON_HT_CBW20 [1]
10	HT_CBW20 [1]
11	HT_CBW40

[1] Please refer to [5] for more details

Table B.4: Standard ID description

Standard ID	Description
00	802.11a
01	WiMAX
10	802.11n
11	NOT USED

Table B.5: Format description

Format	Description
Not Used	Non-HT
0	HT_Mixed_Format
1	HT_Greenfield_Format

Table B.6: Coding type

Coding Type	Description
0	BCC
1	LDPC

Table B.7: Guard Interval to be used (for future release)

Coding Type	Description
0	Use Short GI
1	Use Long GI

Table B.8: Stream description

Stream Number	Description
0	1 stream as input
1	2 stream as input

Appendix C

Viterbi Decoder

C.1 Introduction

The problem of decoding a convolutional code can be thought of as attempting to find a path through the trellis diagram by making use of some decoding rule. Ideally, one would like to choose a path which minimizes the actual number of symbol errors that would be made if one were to compare the encoder input with the decoder output. Although it is possible to conceive of a decoding algorithm based on this approach, attempts to do so have not resulted in hardware that can be easily implemented [24].

With block codes, it turns out to be more practical to attempt to choose the path which matches the received sequence as closely as possible; that is, to attempt to minimize the sequence error probability. While this procedure doesn't guarantee that the actual bit error rate will be minimized, one is guaranteed that making the sequence error rate small will also make the bit error rate small for all but pathologically bad codes. Any performance difficulties that result from this nonoptimality are of no practical importance.

The significance of the trellis viewpoint is that the number of nodes in the trellis does not continue to grow as the number of input symbols increases, but remains at 2k-1, where k is the constraint length of the code. This, of course, is because the redundant portions of the code tree have been merged. A consequence of this merging is that if at some point an incorrect path is chosen, it is possible that the correct path will merge with at a later time. At every stage of the trellis, a set of paths are discarded which balances the set of new paths chosen. This simple iterative process is called as Viterbi algorithm and is a special case of dynamic programming. A good example of how viterbi decoder works, has been described in detail in the online tutorial [25].

C.2 Decoding convolutional codes

There are basically two kinds of approach to decode convolutional codes.

C.2.1 Fano Algorithm

Sequential decoding was first proposed by Wozencraft. Thereafter, Fano improved the algorithm in terms of decoding efficiency. Sequential decoding allows forward and backward movement through the trellis. The decoder keeps track of its decisions each time it makes an ambiguous decision, it tallies it. If tally increases faster than some threshold value, decoder gives up that path and retraces the path back to the last fork where the tally was below the threshold. Detailed design can be found in [24] and [25].

C.2.2 Maximum Likelihood Algorithm

We shall concentrate on Viterbi decoding since errors occur infrequently and probability of error is small. While viterbi decoder extends and updates the metrics for all paths that can potentially be the best, a sequential decoder severely limits the number of paths that are actually updated.

C.3 Traceback and decode operation

[26] defines traceback operation as *This is one of the two read operations and consists of reading a bit and interpreting this bit in conjunction with the present state number as a pointer that indicates the previous state number (i.e. state number of the predecessor). Pointer values from this operation are not output as decoded values; instead they are used to ensure that all paths have converged with some high probability, so that actual decoding may take place. The traceback operation is usually run to a predetermined depth T, before being used to initiate the decode read operation (described next)* [26] also defines decode operation as *This operation proceeds in exactly the same fashion as the traceback operation, but operates on older data, with the state number of the first decode read in a memory bank being determined by the previously completed traceback. Pointer values from this operation are the decoded values and are sent to the bit-order reversing circuit. A decode read can serve as a dual decode and traceback read, this allows us to decode read multiple columns using one traceback read operation of T columns.*

The gist of these definition is that traceback operation is performed to ensure that we have reached optimum stage and if we decode the bits, we can be sure that decode process will not be faulty. There are basically two kinds of hardware architectures for these operations used in practice which is described in the next section.

C.4 Hardware Architectures for Viterbi Decoder

C.4.1 RegisterExchange Method

In the RE approach, a register is assigned to each state. The register records the decoded output sequence along the path from the initial state to the final state. This is shown in the above figures. At the last stage, the decoded output sequence is the one stored in the survivor path register, the register assigned to the state with the minimum PM. Since, the RE method does not need to be traced back, it is faster. However, the RE method does require the copying of all the registers at each stage [27].

C.4.2 Traceback Implementation

In the traceback method, we extract the decoded bits, beginning from the state with the minimum PM, S0. Beginning at this state and tracing backward in time by following the survivor path, which originally contributed to the current Path Metric, a unique path is identified. This is indicated in the figure shown above. While tracing back through the trellis, the decoded output sequence, corresponding to the traced branches, is generated in the reverse order [27].

C.4.3 Comparison of two architectures

Table C.1: Comparison of two architectures of Viterbi decoder

Traceback Method	**Register Exchange Method**
It is reconfigurable since all the operations are done using RAM [28]	It is not reconfigurable for different constraint lengths.
Latency is 2 to 3 times that of Register Exchange Method	Latency is lower and it is faster because of parallel implementation
Some of the implementations need higher clock rate but have low area requirements	Area requirements are higher since most of the work is done in hardware.
It is suitable for any constraint lengths	It is good for constraint lengths less than 5 but not advisable for constraint lengths greater than equal to 7.

C.5 Hardware Design of Viterbi Decoder

Let us first explore the entity level block diagram. Typical block diagram of Viterbi decoder is as shown in the Figure C.1. Each of the blocks have been explained in detail starting from Section C.6.

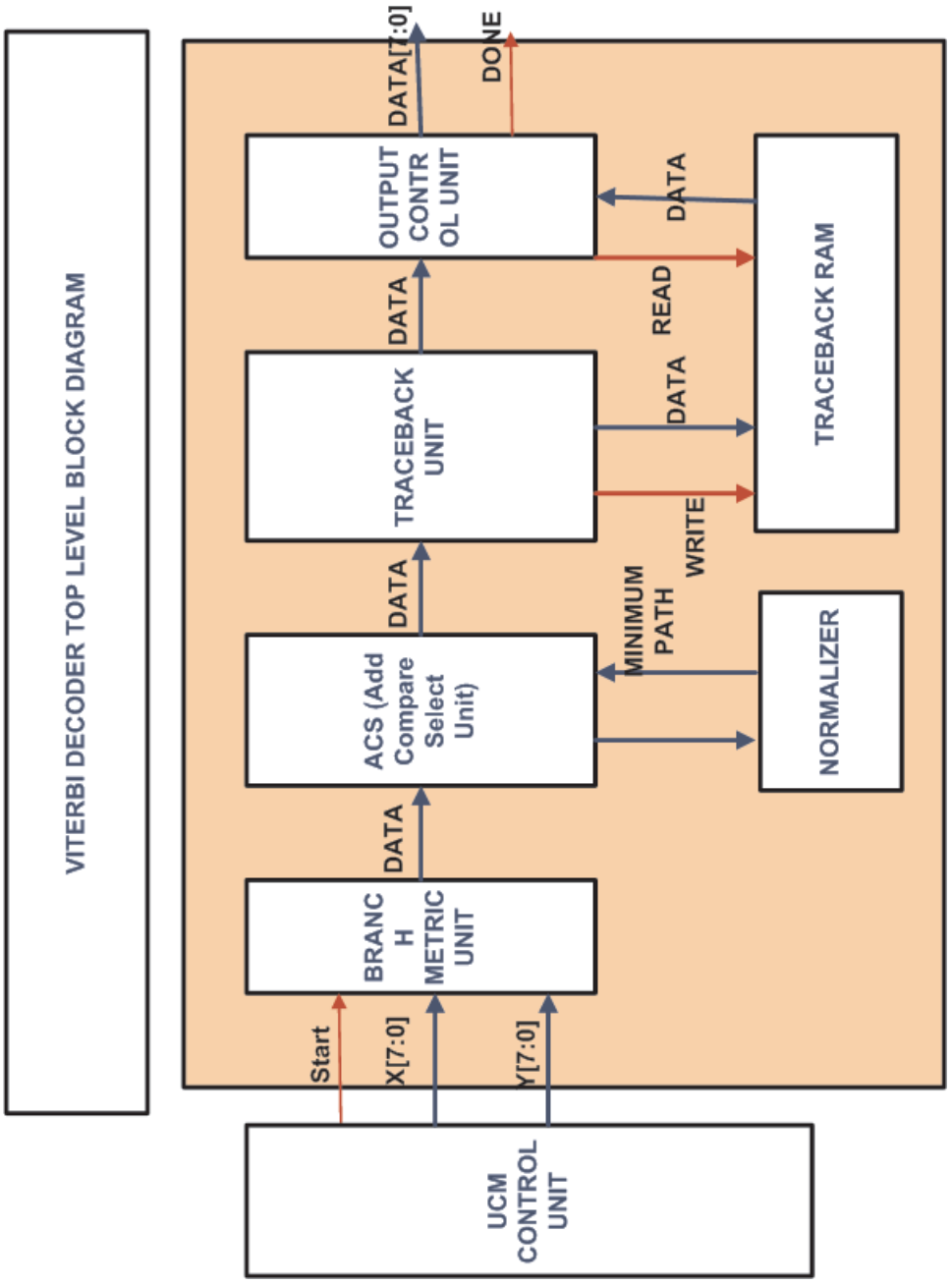

Figure C.1: Top level block diagram of viterbi decoder.

C.6 Viterbi Interface specifications

C.6.1 Overall Block Diagram

The overall block diaram has already been shown in Figure 2.11. The following are the specifications of a typical viterbi decoder.

- It is based on dynamic programming approach.
- It has constraint Length is 7 and code rate = 1/2.
- Generator polynomials of convolution encoder are G0 = 133 and G1 = 171.
- The viterbi decoder has been implemented using Traceback approach because of advantage of reconfigurability for different code rates. Traceback length has been chosen to be 35.
- Soft decision decoding is being done with 6 bit levels.
- Four Dual port distributed RAMs are used.

Table C.2: Comparison of two architectures of Viterbi decoder

Interface	**Signal**	**Number of bits**
Input	X_input	8
Input	Y_input	8
Output	Viterbi_out	8

C.6.2 Branch Metric Unit

The branch metric unit calculates posteriori probabilities by estimating the distance with all possible transmitted codewords. It has been proved that this distance can also be estimated linearly using adders and subtractors [29]. In this way, we have avoided calculation of Euclidean distance. Offset of +2 has been added to get rid of negative values of BM_0-BM_3. The figure has been shown in Figure C.2. Input output specifications have been shown in Table C.3

Table C.3: Comparison of two architectures of Viterbi decoder

Interface	**Signal**	**Number of bits**
Input	X_input	8
Input	Y_input	8
Output	BM_0	9
Output	BM_1	9
Output	BM_2	9
Output	BM_3	9

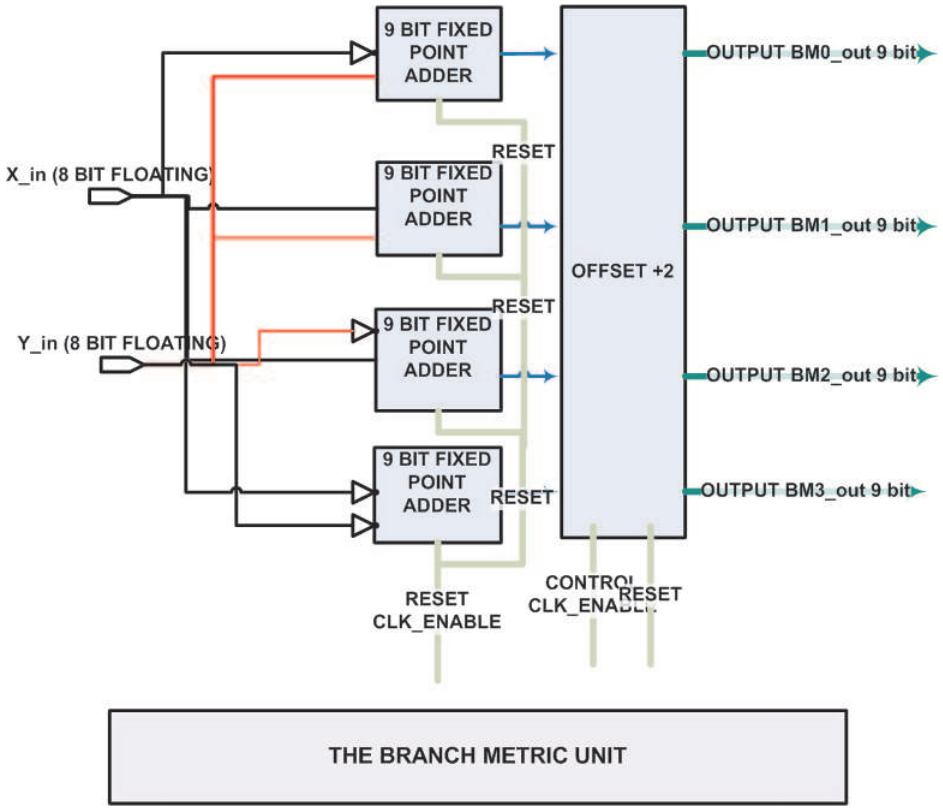

Figure C.2: Branch metric unit design

C.6.3 Branch Metric Table Generation Algorithm

Typically the four output from BMU $(BM_0 - BM_3)$ is fed as an input to ACS unit. There are 32 ACS units and each requires atleast two Branch metrics. For this, a branch metric table consisting of 64 rows is being generated. Each row acts like an input to the ACS unit. Each row references either $(BM_0 - BM_3)$. This algorithm is shown in Appendix A.

C.6.4 ACS Unit (Add-Compare-Select Unit)

Typical trellis structure of ACS Unit is as shown in Figure C.3

The state diagram of the trellis of Figure C.3 is shown in Table C.4.

Table C.4: State diagram of trellis shown in Figure C.3

	Output(NextState)	
States	Input = 0	Input = 1
00	00	10
01	00	10
10	01	11
11	01	11

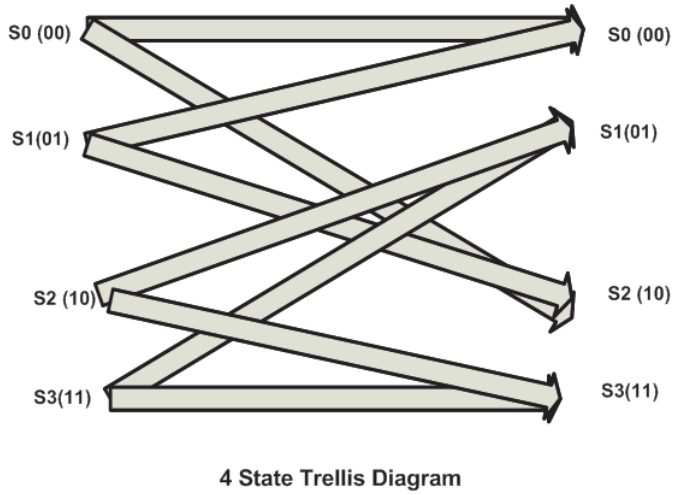

Figure C.3: Typical trellis of viterbi decoder

C.6.5 ACS Unit

Each of the 4 states on the RHS of trellis requires ACS operation. However, we see that states S_0 and S_2 have S_0 and S_1 as their previous state. The only difference between states S0 and S2 is that both have different BMs and PMs (Path Metrics). Hence, the two operations at S_0 and S_2 can be grouped into a single ACS unit. This operation is termed as Trellis reduction. PM is obtained from path metric register. We shall study this in the next subsection. An example has been shown in Figure C.4.

The block diagram of ACS unit is shown in Figure C.5 . The two ACS operations have been combined and put into one block. The four inputs to this block are the combination of two BMs and two PMs. The PM is 15 bit in length. [The reason for increase in the length of PM would be clear in next subsections. Actually, 6 clock cycles are required to compute minimum path metric. During these 6 clock cycles, the register length may increase by 1 bit because of addition operation. Hence, after 6 clock cycles, the register may increase by 6 bits. Therefore, total length of this block is equal to 9 bits + 6 bits.] The bits B_0 and B_1 are stored in Traceback unit in order to maintain the history of traversed trellis paths. This is further explained in later sections. The input output specification of ACS unit is shown in table

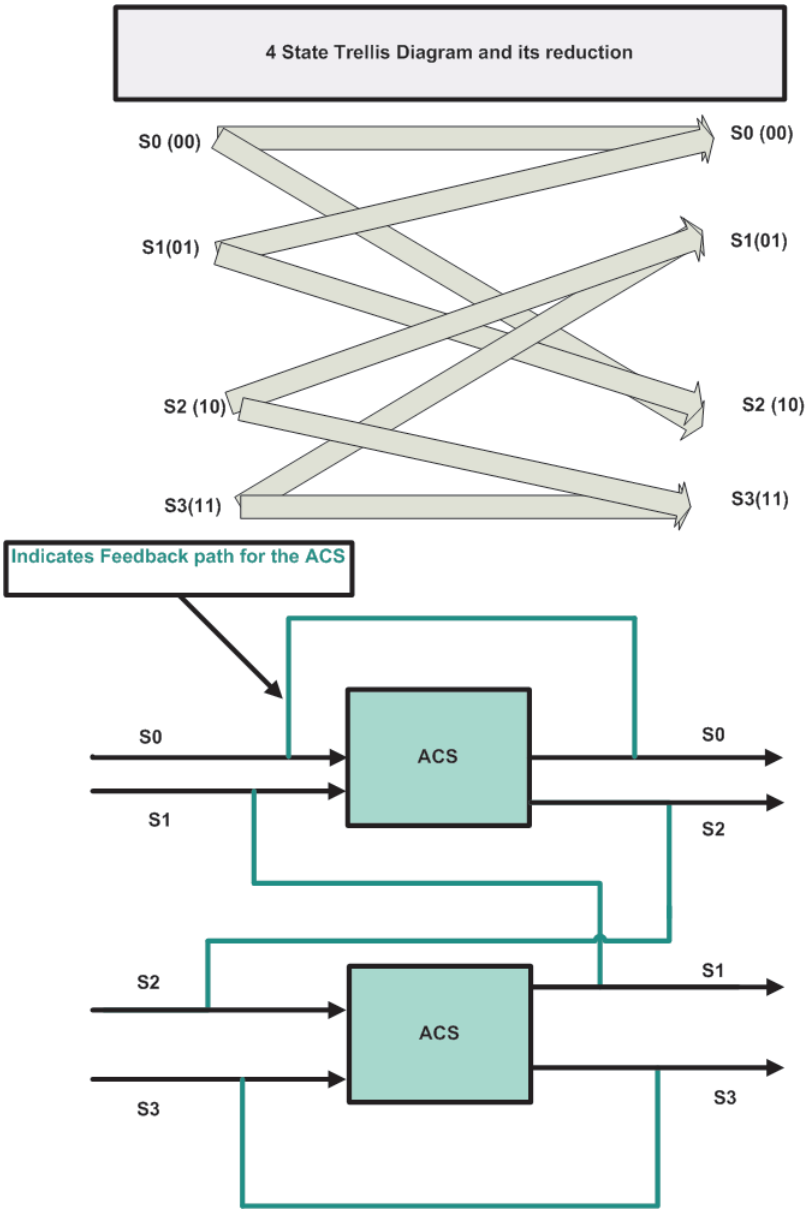

Figure C.4: Typical reduction of trellis diagram

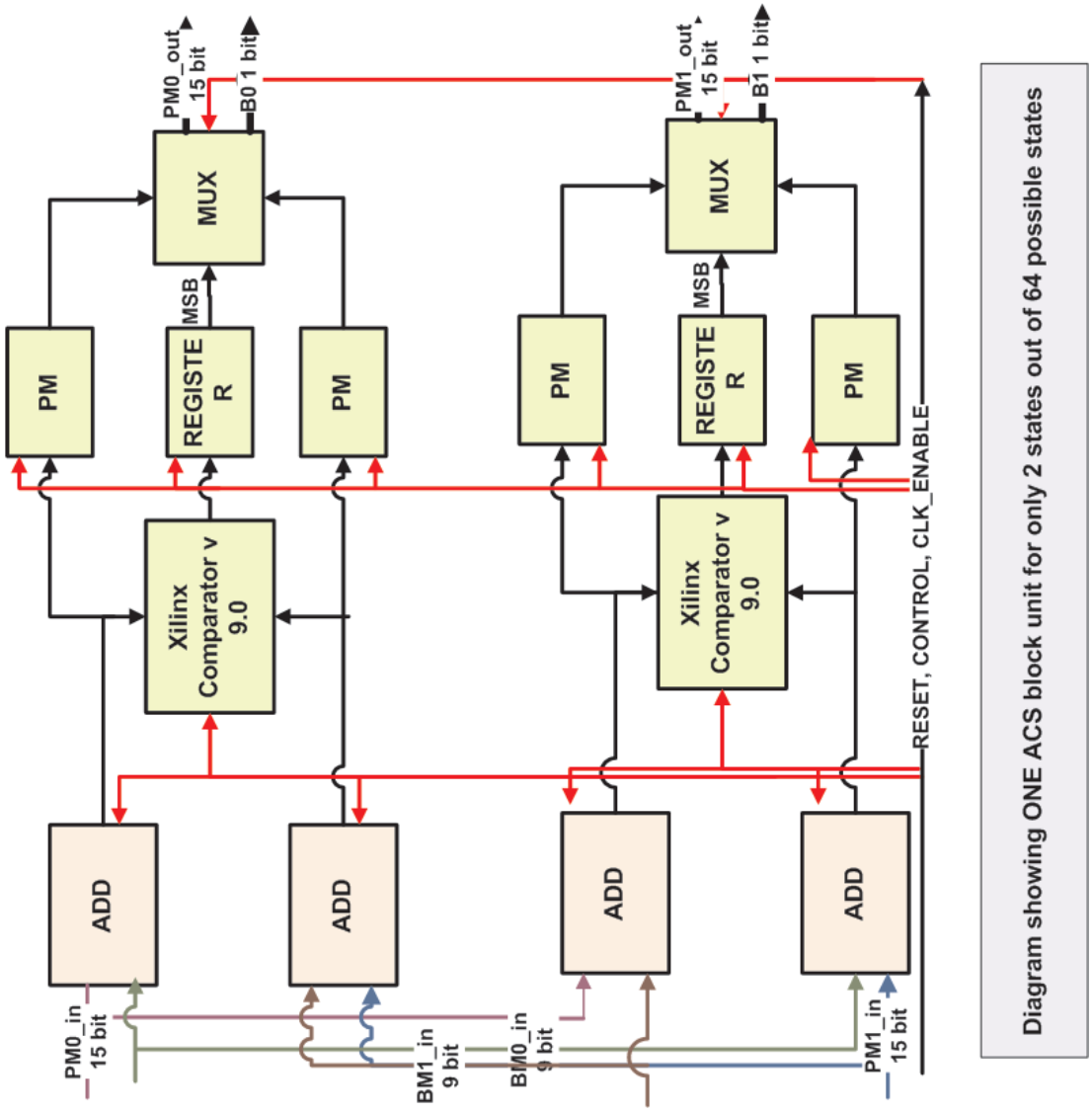

Figure C.5: Top level block diagram of ACS Unit.

Table C.5: ACS Unit specifications

	Signal	Number of bits
Input	BM_0 and BM_1	9
Input	PM_0 and PM_1	15
Output	PM_0_out and PM_1_out	15
Output	B_0 and B_1	1

C.6.6 Normalizer Unit

The normalizer block is used to prevent overflow. This block requires 6 clock cycles to compute the minimum of all 64 PM registers. Hence, the width of PM has been increased from 9 bits by 6 bits. The state diagram is as shown in the Figure C.7.

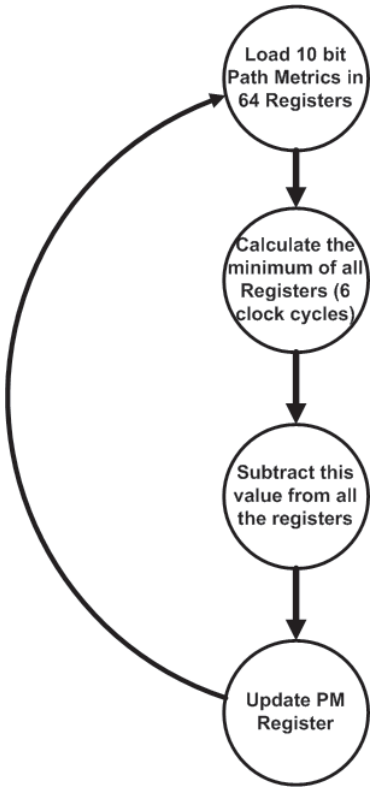

Figure C.6: State diaram of normalizer unit

The block diagram of Normalizer unit is as shown in Figure C.7.

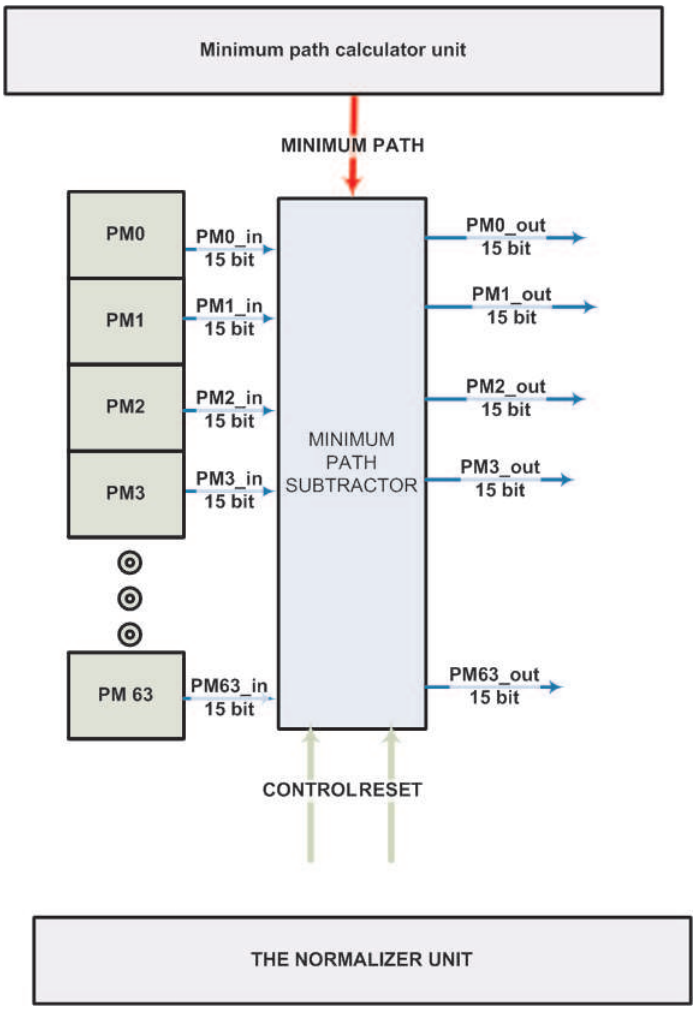

Figure C.7: Block diagram of normalizer unit

C.6.7 Minimum path calculator unit

The block diagram of minimum path calculator unit is shown in Figure C.8.This circuit calculates the minimum path. The Xilinx comparator circuit is being used. The output of this comparator circuit is being fed to the multiplexor. The multiplexor then selects the minimum of the two input PMs. This operation is then recursively repeated for subsequent PMs. This operation is done for 6 clock cycles processing 64, 32, 16, 8, 4 and 2 PMs respectively.

Table C.6: Minimum Path Calculation Unit specifications

MPCU	Signal	Number of bits
Input	PM_0 through PM_{63}	15
Output	MP_{in}	15

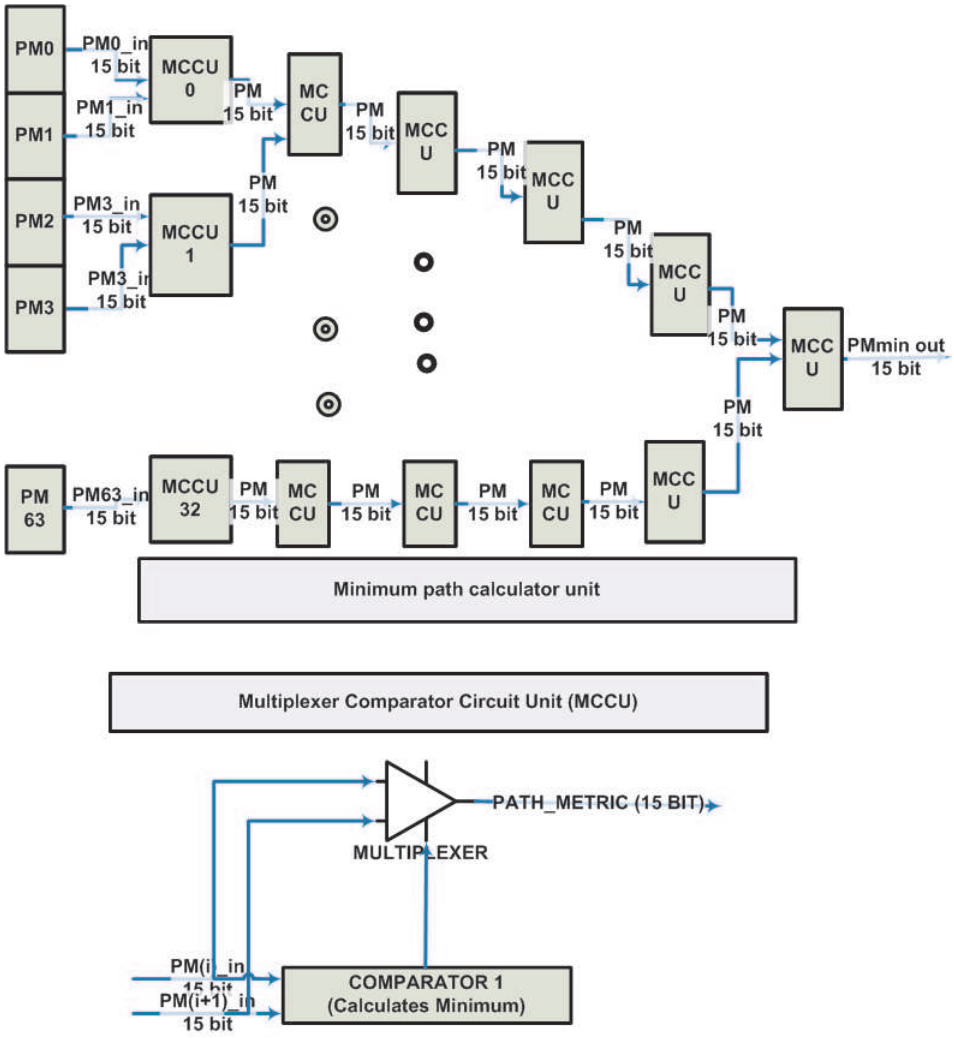

Figure C.8: Block diagram of minimum path calculator unit

C.6.8 Traceback unit

This can be considered the heart of whole Viterbi decoder. Since, the survivor path of the ACS unit is stored in Traceback RAM, this unit is also referred to as Survivor path unit. The output of ACS operation is written on to the memory. The memory is organized into 4 banks. Each bank is 64 X 35 bits wide. Subsections C.6.8 and C.6.8 describe the state diagram and timing operation of the four banks. The four banks help us to pipeline our traceback, decode and write operation. This results in higher throughput. All the four memories are dual port RAMs. Distributed RAMs are preferred over Block RAMs.

Table C.7: Minimum Comparator Circuit Unit specifications

MPCU	Signal	Number of bits
Input	PM_0_in through PM_{63}	15
Input	PM_1_in	15
Output	PM_out	15

State Machine Diagram of Traceback RAM

There are basically three cycles. Read (Decode or DC) cycle, Write Cycle and traceback cycle. Traceback is done to ensure that the states have converged to local minimum. After traceback is complete, decode operation is done. Traceback is done for 35 cycles. All the three operation are done in parallel to increase the throughput and decrease the latency. Traceback RAM is also called as Survivor Memory unit because it saves the survivor paths of the trellis. Typical state machine diagram of is shown in Figure

Timing diagram

This diagram shows input output behavior of all the four banks with respect to time. The three processes can be implemented using traditional VHDL 'processes'. After reading/writing/traceback, the pointer wraps to the beginning of the RAM ie the first memory location. The timing diagram has been shown in Figure 2.12

Traceback circuit

The traceback unit consists of three operations. Writing data to the memory, performing traceback and reading the data. This necessitates us to design a mechanism that would do three things:

- *Traverse the trellis in reverse direction during traceback operation* This can be further explained by taking an example of 4-state trellis shown in above sub-sections. If we see the trellis diagram, states S_0 or S_1 precedes S_2. Represented in binary form we can say that state 00 or 01 precedes 10. Also, given state 00, if the input is 0, the next state is 00. Given the state 00, if the input is 1, the next state is 10. Thus, we can go from state S_2 (10) to either 01 or 00 by left shifting the state metric; i.e. if we left shift 10 or 00, it becomes 0X. We need to now decide the 'X' (unknown) bit. This bit is received from the traceback RAM. [Recall that the survivor path of the two possible paths is stored in RAM]. Hence, if the survivor path is B_0 = 1 (say) the previous state from 10 is 01. If the survivor path is B_0 = 0 (say) the previous state from 10 is 00. Now, given the previous state, we choose one bit from the possible 64 bits corresponding to the 64 states using multiplexer. The select input to the multiplexer chooses the survivor path of the current state. (We

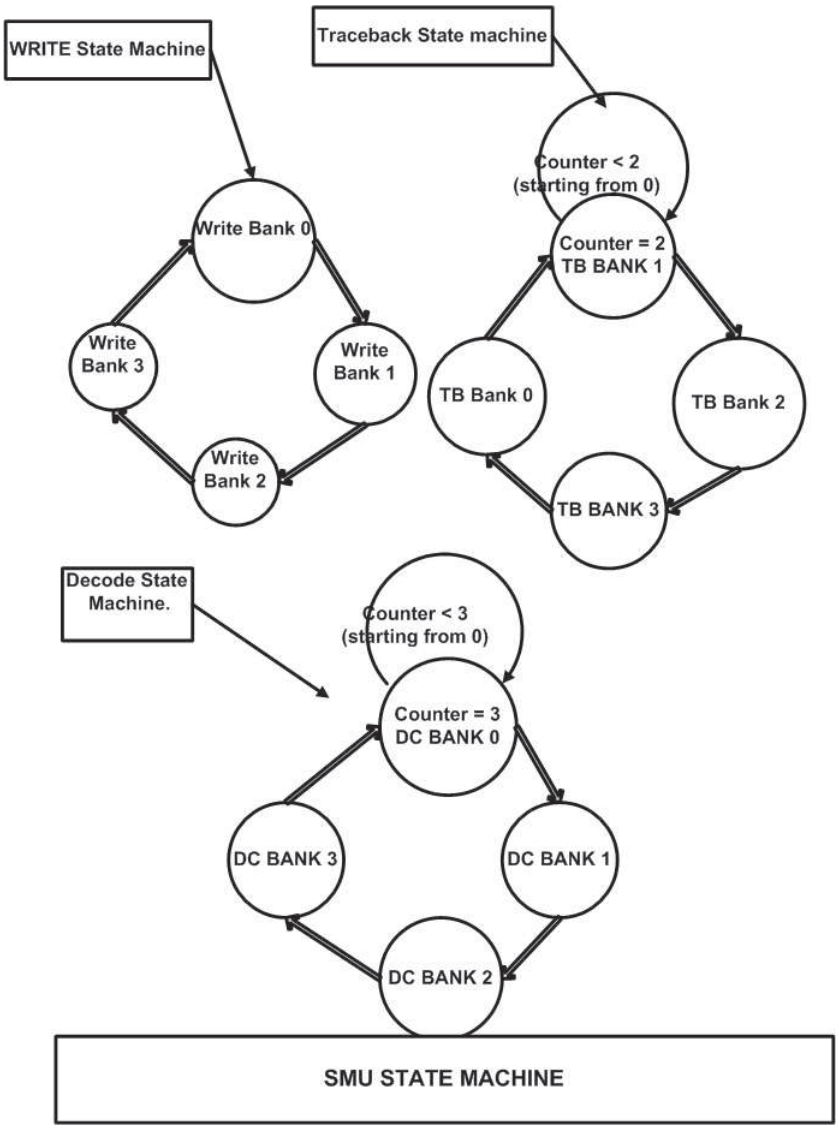

Figure C.9: State machine diagram of four banks

have assumed that we are in the process of decoding and 64 bit sequence are available as input to the multiplexer.)

- *Decode operation* We also need to decode the data. This decode operation is fairly simple. This can be easily explained by referring to 4-state trellis explained above. We see that given a zero input on any of the four states, the next state is either 00 or 01. Similarly, given a one input on any of the four states, the next state is either 10 or 11. Thus, we can easily determine the input if we examine the MSB of the present state. This concept can be applied to any number of states.

  Subsection C.6.8 shows arrangement of Memory Unit and the combiner/decombiner operation. Sub-section C.6.8 shows arrangement of Traceback and decode circuit.

- *Generating output* Since, we traverse our path in the reverse direction; bits are actually in the reverse order. In order to reverse it, we use a shift register coupled with the memory. The 8-bit output is first reversed by reading data from the register in the opposite direction. This 8-bit is then written to the memory starting from the last memory location. The data is then read from the top of the memory by the UCM.

Memory unit

This unit has been shown in Figure C.11.

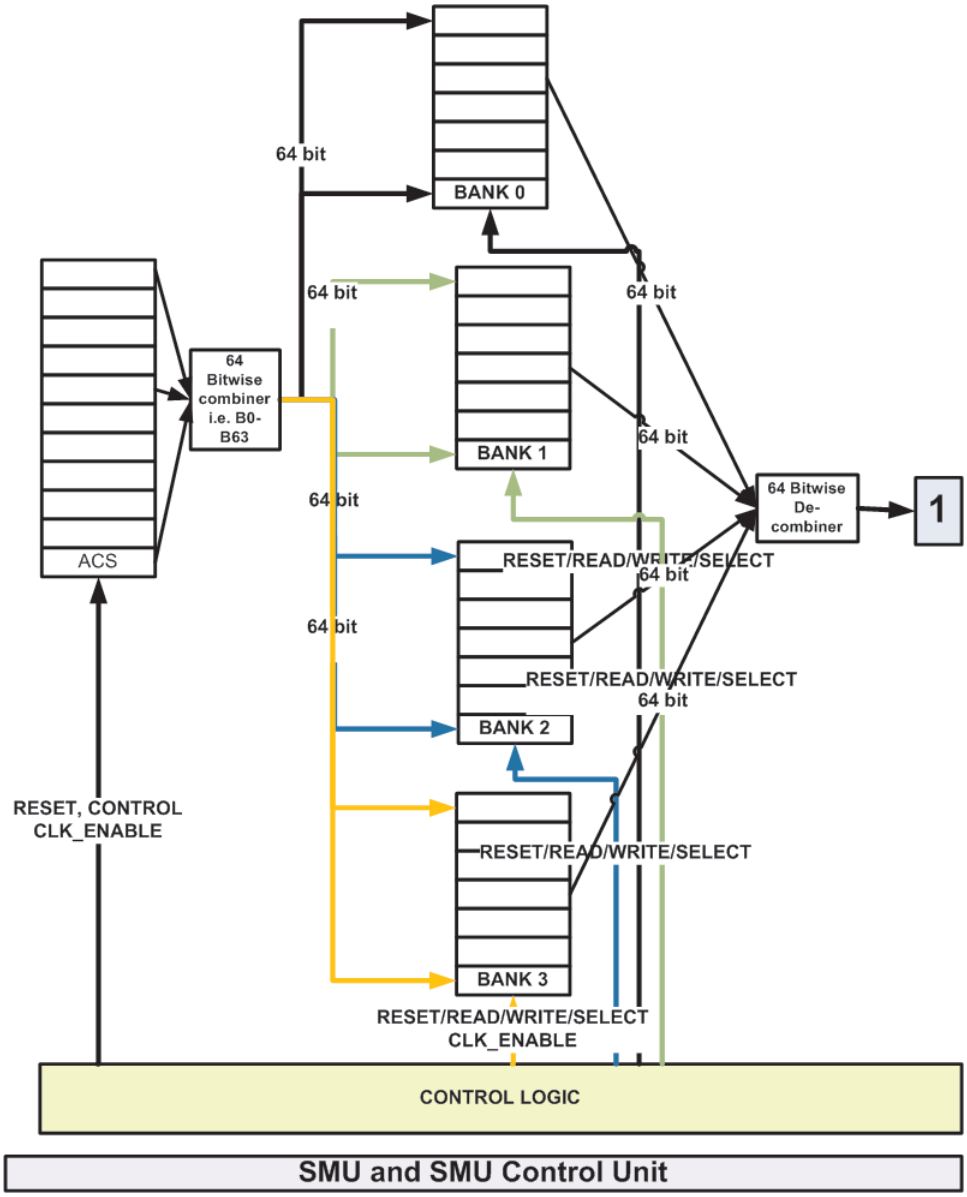

Figure C.10: Block diagram of SMU

Traceback output unit

This is the last stage of viterbi decoding operation. This unit has been shown in Figure C.11.

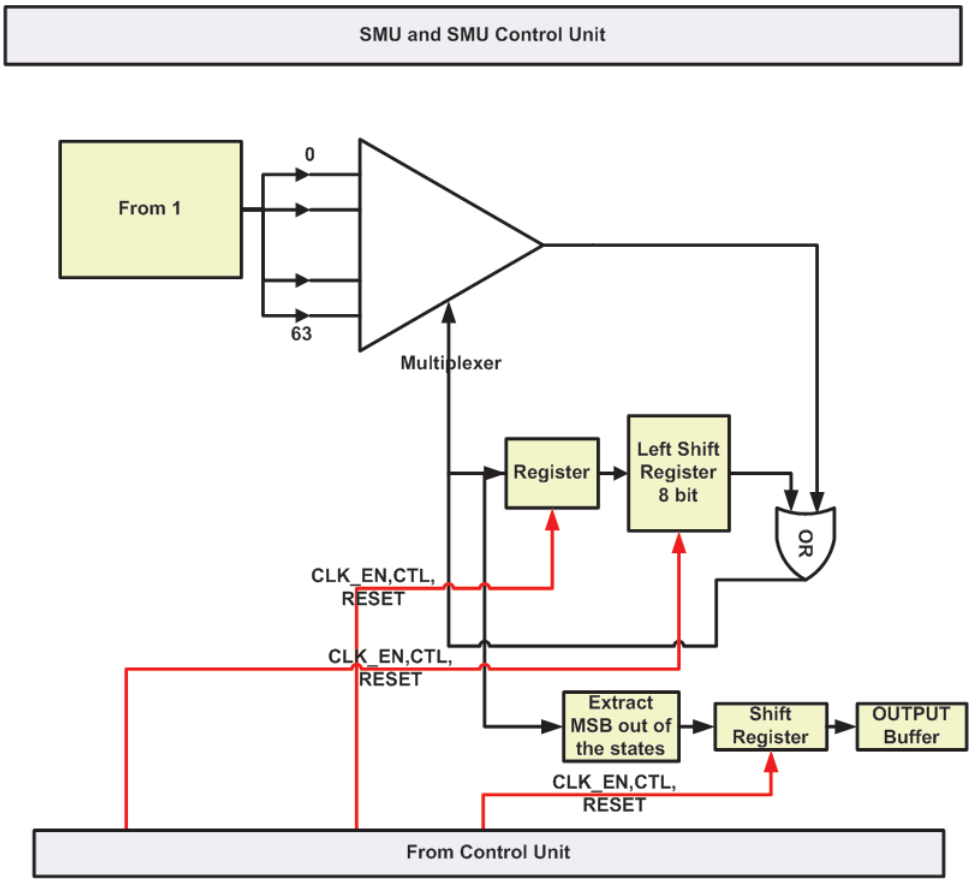

Figure C.11: Block diagram of SMU control unit

C.7 Vitebi Input/Output specification

C.7.1 Memory specifications

Total memory size that is required is equal to 64 X 35 X 4 = 1120 bytes.

C.7.2 Input output specifications

Input would be two 8 bit data where MSB would be used for signed arithmetic.

C.7.3 Latency specifications

- Latency due to Traceback Unit = 35 X 4 Cycles.

- Therefore, for 70MHz clock, latency would be 0.000002 seconds = 2 microseconds.
- The throughput in this case would be 70Mbps since on every clock cycle we would have one bit output.

C.8 Conclusion

We demonstrated the basic understanding of viterbi decoder. We also showed the different architectures and tradeoff between them. We also showed that traceback implementation best suits for applications like WLAN etc. Finally, hardware circuit was designed.

Appendix D

These tables show the latency estimates for each of the individual blocks. These estimates have been calculated using Xilinx.

1. **FFT**

Table D.1: FFT Latency estimation from Xilinx

Read	Write	Calculated latency from Xilinx	Latency after subtracting read cycles
64	206	241	321
128	160	468	628
256	320	852	1172

2. **Modulator**

   The number of read cycles would be equal to total number of bits per subcarrier (Table 2.2) / 16 for 16 bit wide data bus or total number of bits per subcarrier(Table 2.2) / 32 for 32 bit wide data bus.

   The number of write cycles would be equal to total number of subcarriers because each word would require atleast one clock cycle for writing.

3. **Encoder PE**

   We have assumed RAM based implementation for the interleaver. Since Interleaver, stream parser and encoder are included in one PE, there would be pipelining amongst them. Hence, the total latency β is calculated by equation D.1.

$$\beta = Readcycles + Writecycles + \varphi; \tag{D.1}$$

   where φ = N_{DBPS} (Table 2.2).

   Read cycles and write cycles depend on the size of data bus i.e. 16 bit or 32 bit.

4. **Channel Equalizer PE**

The latency of this block would totally depend on the number of multiplier, adder and cordic blocks used and their respective latency. The latency of these block is highlighted in Table D.2

Table D.2: Adder, Multiplier and Cordic Latency estimation from Xilinx

Block	Latency for performing its operation
Adder	1
Multiplier (16-bit)	5
Cordic	7

We note that Cordic is used for calculating square root. This would be required for calculating determinant of the matrix.

5. **Channel Estimation PE**

   Channel Estimation requires an IFFT and two FFTs. Hence the total latency would be calculated using the Table D.1.

6. **FCS PE**

   It has been assumed that FCS is a parallel 8 bit CRC. Hence the total latency would then depend on the modulation scheme and the number of output bits from the viterbi decoder.

It is assumed that blocks like puncturing/de-puncturing have latency of 2-3 clock cycles [20]. Hence, this is not considered for the analysis.