MASTER

Exploration of low-power viterbi decoders design for low-throughput application

Lu, C.

Award date:
2016

Link to publication
Exploration of Low-power Viterbi Decoders Design for Low-throughput Application

Master Thesis

Chunqiu Lu

Supervisors:
C.H. (Kees) van Berkel
Pepijn Boer
M.C.W. (Marc) Geilen

Eindhoven, July 2016
Abstract

The IEEE 802.11ah is a long range 802.11 Wireless Local Area Network (WLAN) standard. It aims to achieve a cost-effective and large scale wireless network for Internet of Things (IoT) applications. Convolutional encoding is an important process to provide error-correcting codes for the wireless transmission in a 802.11ah transmitter. This thesis work explores the digital integrated circuit (IC) design and implementation of a low-power and small-area convolutional decoder by the Viterbi Algorithm for the receiver to fit the low-throughput and low-power requirement of 802.11ah standard.

After investigating different architectures and algorithms to save the power and area, the promising designs are implemented with the Cadence digital IC design tool using the TSMC 40nm technology at 1.1V supply voltage. To fulfill the current project requirements, 5 Mb/s is the target throughput. The results of the Cadence front-end and back-end simulation are analyzed concerning the power and area and also are compared with the state-of-the-art design in the literature. Among all the architectures with the conventional Viterbi Algorithm, the optimal design is the fully-parallel architecture which has 247 uW power and 0.04 mm$^2$ area at 10 MHz clock. Among all implemented algorithms based on the fully-parallel architecture with the conventional Viterbi Algorithm, the pre-traceback (PTB) algorithm are the optimal with 269 uW power and 0.04 mm$^2$ area at 5 MHz clock rate. PTB can achieve the maximum throughput (40 Mb/s) among all designs with 40 MHz clock rate. So it is promising to be the optimal low-power design by the voltage and clock rate scaling. The power target based on the scaling results of state-of-the-art designs can be achieved by two selected implementations.

Keywords. 802.11ah, IoT, low-power, Viterbi Decoder, IC design
Preface

This document is the Master thesis of my graduation project for the M.Sc degree with a specialization in Embedded Systems at TU Eindhoven. The project began in November 2015 and finished in July 2016. It was carries out at IMEC in the Holst Center.

Firstly, I would like to express my sincere gratitude to my university supervisor Prof. C.H. van Berkel for the great guidance. His knowledge, patience and attitude to the science deeply impressed me. I would like to also thank Pepijn Boer as my company supervisor who gives me the chance to carry out my research at IMEC in the Holst Center. He guided me a lot during the research work and helped me improve my presentation and thesis writing. Also, my appreciation is also for dr. M.C.W. Geilen, the member of my examination committee, who gave me several valuable comments. In addition, I would like to appreciate my colleges in IMEC especially Evgeni and Benjamin for their help in the VLSI design. Without their help, I cannot smoothly work on my research. My gratitude is also for my friends and classmates during my work and study. Finally, I greatly appreciate the support from my parents and my girlfriend in the past years.

Lu Chunqiu
Eindhoven, August 2016
# Contents

<table>
<thead>
<tr>
<th>Contents</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>x</td>
</tr>
<tr>
<td>Acronyms</td>
<td>xi</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Background and contribution ................................. 1
1.2 IEEE 802.11ah and IoT Overview ................................ 2
  1.2.1 Base-band Processing Overview ................................ 2
  1.2.2 Binary Convolutional Encoder and Punctuate Overview ................ 3
1.3 Design Tools Overview ........................................ 4

## 2 Problem Definition

2.1 Motivation ................................................. 6
2.2 Problem .................................................. 6
2.3 Objectives ............................................... 7
2.4 Overview of literature study ................................ 8
2.5 Approach .................................................. 10

## 3 Viterbi Decoders

3.1 Introduction ............................................. 12
3.2 Branch Metric Unit ....................................... 13
  3.2.1 Maximum Likelihood Bit metrics .............................. 13
  3.2.2 log-likelihood ratio soft bit metrics ......................... 14
3.3 Path Metric Unit .......................................... 15
3.4 Survivor Memory Unit ...................................... 16
3.5 Simulation Platform ....................................... 17

## 4 Design and implementation of architecture

4.1 Branch Metric Unit ....................................... 19
  4.1.1 Quantization Scheme ...................................... 19
  4.1.2 Architecture Design ...................................... 21
4.2 Add-Compare-Select Unit ................................. 22
  4.2.1 Precision and Normalization Scheme ........................... 22
  4.2.2 Architecture Design ...................................... 23
4.3 Path Metric Unit .......................................... 24
  4.3.1 Architecture Design ...................................... 24
  4.3.2 Routing and Scheduler Design ................................. 25
  4.3.3 Analysis ................................................ 27
4.4 Survivor Memory Unit ...................................... 28

---

iv Exploration of Low-power Viterbi Decoders Design for Low-throughput Application
## CONTENTS

4.4.1 Register Exchange and Traceback Algorithm ................................. 28  
4.4.2 Architecture Design ................................................................... 29  
4.4.3 Choice of Memory ...................................................................... 32  
4.5 Overall design ............................................................................... 32  

5 Design of standard and adaptive algorithm ................................. 34  
5.1 Traceforward algorithm ................................................................. 34  
5.1.1 Introduction ............................................................................ 34  
5.1.2 Architecture design ................................................................. 35  
5.1.3 Analysis ................................................................................ 36  
5.2 Pre-traceback algorithm ................................................................. 38  
5.2.1 Principle and architecture design .............................................. 38  
5.2.2 Analysis ................................................................................ 40  
5.3 Dynamic traceback length with path prediction algorithm .......... 40  
5.3.1 Introduction and architecture design ........................................... 40  
5.3.2 Analysis ................................................................................ 41  

6 Implementation, results and analysis ........................................... 43  
6.1 Introduction ................................................................................. 43  
6.2 Specification ................................................................................ 43  
6.3 Implementation of different architectures .................................... 44  
6.3.1 Design .................................................................................. 44  
6.3.2 Back-end results and analysis .................................................. 45  
6.3.3 Comparison of front-end and back-end results ......................... 47  
6.4 Implementation of standard algorithm ........................................ 48  
6.4.1 Design .................................................................................. 48  
6.4.2 Back-end results and analysis .................................................. 48  
6.4.3 Comparison of front-end and back-end results ......................... 49  
6.5 Implementation of adaptive algorithm ........................................ 51  
6.5.1 Design .................................................................................. 51  
6.5.2 Back-end results and analysis .................................................. 51  
6.6 Comparison of all implementations .............................................. 52  

7 Conclusion and recommendations ................................................. 55  
7.1 Conclusion .................................................................................. 55  
7.2 Recommendations ....................................................................... 56  

Bibliography ....................................................................................... 57  

Appendix ............................................................................................ 61  

A Introduction to Viterbi Algorithm .................................................... 61  

B Conventional Viterbi Algorithm ....................................................... 62  

C Flow chart of dynamic truncation length and path prediction algorithm ...................................................................................... 63  

D Routing of fully parallel, shared factor 2 and 4 ................................ 64  

E Analysis of the front-end simulation results .................................. 66  
E.1 Front-end results and analysis of three architectures .................... 66  
E.2 Front-end results and analysis of two standard algorithms .......... 69  
E.3 Front-end results and analysis of the adaptive algorithm ............ 71  

Exploration of Low-power Viterbi Decoders Design for Low-throughput Application
CONTENTS

F Comparison of results by the front-end and back-end synthesis of different architectures 73

G Comparison of results by the front-end and back-end synthesis of two standard algorithms 75

H Comparison of results by the front-end and back-end synthesis of adaptive algorithms 77
## List of Figures

1.1 Simplified base-band processing flow of 802.11ah at transmitter and receiver sides . 2
1.2 Convolutional encoder (K=7) 3
1.3 An example of the trellis 4
1.4 Puncturing scheme to achieve coding rate 3/4 4
1.5 Cadence Digital IC design flow 5
2.1 Figure of Merit 8
2.2 Design space for the Branch Metric Unit and Path Metric Unit 9
2.3 Design space for Survivor Memory Unit (SMU) 10
3.1 Simplified base-band processing flow of 802.11ah at transmitter and receiver sides . 12
3.2 the BER performance with hard and soft decision decoding 15
3.3 Illustration of ACSU in a PMU 16
3.4 Truncation property [1] 17
3.5 Structure of the Viterbi Decoder 17
3.6 Simulation Platform based on Matlab 18
4.1 An example of a q=3 quantization 20
4.2 Comparison of different input quantization schemes with QPSK in the AWGN channel 20
4.3 Comparison of different input quantization schemes with 16QAM in the AWGN channel 21
4.4 Design of Branch Metric Unit 22
4.5 (a)2-way Add-Compare-Select Unit (ACSU) (b)Radix-2 trellis (c)Radix-2 ACSU [2] 24
4.6 Architecture of Path Metric Unit 25
4.7 An example of shared-4 architecture for 802.11ah standard 25
4.8 Local routing example 26
4.9 (a)Implementation of local routing (b) Example of the implementation 26
4.10 Control unit design 27
4.11 Critical path of a VD 28
4.12 VLSI design for register exchange algorithm (REA) [3] 29
4.13 the BER performance with different traceback length 30
4.14 the BER performance with different traceback start state scheme 31
4.15 Survivor Memory Unit architecture design 31
4.16 Saving percentage of memory access energy per bit 32
4.17 Overall architecture design 33
5.1 TF Example 35
5.2 (a)Design of Add-Compare-Select Unit (ACSU) for TF (b)Design of Processing Element (PE) 36
5.3 Memory access of sliding window TF algorithm [4] 36
5.4 Design of Add-Compare-Select Unit (ACSU) for sliding traceforward (TF) 37
5.5 Saving percentage of memory access energy per bit by TF 38
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6</td>
<td>PTB example</td>
<td>39</td>
</tr>
<tr>
<td>5.7</td>
<td>(a) Design of Add-Compare-Select Unit (ACSU) for PTB (b) Design of Processing Element (PE)</td>
<td>39</td>
</tr>
<tr>
<td>5.8</td>
<td>Illustration of the DTBL algorithm</td>
<td>41</td>
</tr>
<tr>
<td>5.9</td>
<td>Design of Survivor Memory Unit (SMU)</td>
<td>41</td>
</tr>
<tr>
<td>5.10</td>
<td>Memory access saving with the DTBL algorithm</td>
<td>42</td>
</tr>
<tr>
<td>6.1</td>
<td>Comparison of power distribution between three architectures by back-end synthesis</td>
<td>46</td>
</tr>
<tr>
<td>6.2</td>
<td>Comparison of power distribution of the PE between three architectures by back-end synthesis</td>
<td>46</td>
</tr>
<tr>
<td>6.3</td>
<td>Comparison of power distribution of minor modules between three architectures by back-end synthesis</td>
<td>46</td>
</tr>
<tr>
<td>6.4</td>
<td>Comparison of area distribution between three architectures by back-end synthesis</td>
<td>47</td>
</tr>
<tr>
<td>6.5</td>
<td>Comparison of area distribution of the PE between three architectures by back-end synthesis</td>
<td>47</td>
</tr>
<tr>
<td>6.6</td>
<td>Area distribution comparison of minor modules between three architectures by back-end synthesis</td>
<td>47</td>
</tr>
<tr>
<td>6.7</td>
<td>Comparison of power distribution between 2 standard algorithms by back-end synthesis</td>
<td>49</td>
</tr>
<tr>
<td>6.8</td>
<td>Comparison of power distribution of the PE between 2 standard algorithms by back-end synthesis</td>
<td>49</td>
</tr>
<tr>
<td>6.9</td>
<td>Power distribution comparison of minor modules between 2 standard algorithms by back-end synthesis</td>
<td>49</td>
</tr>
<tr>
<td>6.10</td>
<td>Comparison of area distribution between 2 standard algorithms by back-end synthesis</td>
<td>50</td>
</tr>
<tr>
<td>6.11</td>
<td>Comparison of area distribution of the PE between 2 standard algorithms by back-end synthesis</td>
<td>50</td>
</tr>
<tr>
<td>6.12</td>
<td>Area distribution comparison of minor modules between 2 standard algorithms by back-end synthesis</td>
<td>50</td>
</tr>
<tr>
<td>6.13</td>
<td>Comparison of power distribution between the DTBL and shared-2 by back-end synthesis</td>
<td>51</td>
</tr>
<tr>
<td>6.14</td>
<td>Power distribution of minor modules between the DTBL and shared-2 by back-end synthesis</td>
<td>51</td>
</tr>
<tr>
<td>6.15</td>
<td>Memory access saving with DTB algorithm based on shared-2 architecture</td>
<td>52</td>
</tr>
<tr>
<td>6.16</td>
<td>Comparison of area distribution of the PE between the DTBL and shared-2 by back-end synthesis</td>
<td>52</td>
</tr>
<tr>
<td>6.17</td>
<td>Comparison of energy per bit with maximum clock frequency</td>
<td>54</td>
</tr>
<tr>
<td>7.1</td>
<td>Final result</td>
<td>55</td>
</tr>
<tr>
<td>A.1</td>
<td>Viterbi Algorithm [5]</td>
<td>61</td>
</tr>
<tr>
<td>C.1</td>
<td>Illustration of (a) dynamic truncation length (b) path prediction algorithm [6]</td>
<td>63</td>
</tr>
<tr>
<td>E.1</td>
<td>Comparison of power distribution between three architectures by front-end synthesis</td>
<td>66</td>
</tr>
<tr>
<td>E.2</td>
<td>Comparison of power distribution of the PE between three architectures by front-end synthesis</td>
<td>67</td>
</tr>
<tr>
<td>E.3</td>
<td>Comparison of power distribution of minor modules between three architectures by front-end synthesis</td>
<td>67</td>
</tr>
<tr>
<td>E.4</td>
<td>Comparison of area distribution between three architectures by front-end synthesis</td>
<td>67</td>
</tr>
<tr>
<td>E.5</td>
<td>Comparison of area distribution of the PE between three architectures by front-end synthesis</td>
<td>68</td>
</tr>
<tr>
<td>E.6</td>
<td>Comparison of area distribution of minor modules between three architectures by front-end synthesis</td>
<td>68</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

E.7 Comparison of power distribution between 2 standard algorithms by front-end synthesis .................................................. 69
E.8 Comparison of power distribution of the PE between 2 standard algorithms by front-end synthesis ........................................ 69
E.9 Power distribution comparison of minor modules between 2 standard algorithms by front-end synthesis ........................................ 70
E.10 Comparison of area distribution between 2 standard algorithms by front-end synthesis .................................................. 70
E.11 Comparison of area distribution of the PE between 2 standard algorithms by front-end synthesis ........................................ 71
E.12 Area distribution comparison of minor modules between 2 standard algorithms by front-end synthesis ........................................ 71
E.13 Comparison of power distribution of the PE between the DTBL and shared-2 by front-end synthesis ........................................ 72
E.14 Power distribution comparison of minor modules between the DTBL and shared-2 by front-end synthesis ........................................ 72
E.15 Comparison of area distribution between the DTBL and shared-2 by front-end synthesis .................................................. 72
F.1 Comparison of power distribution with front-end and back-end synthesis .................................................. 73
F.2 Comparison of power distribution of PEs with front-end and back-end synthesis .................................................. 74
F.3 Comparison of area distribution with front-end and back-end synthesis .................................................. 74
F.4 Comparison of area distribution of PEs with front-end and back-end synthesis .................................................. 74
G.1 Comparison of power distribution with front-end and back-end synthesis .................................................. 75
G.2 Comparison of power distribution of PEs with front-end and back-end synthesis .................................................. 75
G.3 Comparison of area distribution with front-end and back-end synthesis .................................................. 76
G.4 Comparison of area distribution of PEs with front-end and back-end synthesis .................................................. 76
H.1 Comparison of area distribution between DTBL and shared-2 by front-end and back-end synthesis .................................................. 77
H.2 Comparison of area distribution between DTBL and shared-2 by front-end and back-end synthesis .................................................. 77
List of Tables

1.1 SIG MCSs for 1 MHz, $N_{ss} = 1$ .................................................. 3
3.1 Comparison of maximum likelihood and LLR soft bit metrics ............. 15
5.1 Comparison of decoding latency ..................................................... 38
6.1 Specification of three architectures ............................................. 45
6.2 Specification of 3 standard algorithms ....................................... 48
6.3 Comparison of all the implementations by back-end synthesis ............. 53
6.4 Comparison of all the clock frequency and memory size .................... 54
D.1 Scheduler in the local routing of PMU for (a) shared factor 2 and (b) shared factor 4 64
D.2 Global routing table of fully parallel architecture ............................ 65
D.3 Global routing table of (a) shared factor 2 and (b) shared factor 4 ........ 65
Acronyms

16-QAM  16-quadrature amplitude modulation.
ACS  Add-Compare-Select.
ACSU  Add-Compare-Select Unit.
ASIC  application-specific integrated circuit.
AWGN  additive white Gaussian noise.
BCC  binary convolutional coding.
BER  bit error rate.
BICM  Bit-Interleaved Coded Modulation.
BM  Branch Metric.
BMU  Branch Metric Unit.
BPSK  binary phase shift keying.
DC  decoding read.
DTBL  dynamic traceback length.
EDA  electronic design automation.
FILO  first-in last-out.
HDL  hardware-declare language.
IC  integrated circuit.
IoT  Internet of Things.
LLR  log-likelihood ratio.
MCS  modulation and coding scheme.
OFDM  orthogonal frequency-division multiplexing.
PE  Processing Element.
PM  Path Metric.
Acronyms

PMU  Path Metric Unit.
PTB  pre-traceback.
QPSK  quadrature phase shifting keying.
REA  register exchange algorithm.
RTL  register-transfer language.
S1G  sub 1 GHz.
SCM  standard cell-based memory.
SM  State Metric.
SMU  Survivor Memory Unit.
SNR  signal-to-noise ratio.
TB  traceback read.
TBA  traceback algorithm.
TF  traceforward.
TRR  traceback recursion rate.
TSMC  Taiwan Semiconductor Manufacturing Company.
ULP  ultra-low-power.
VA  Viterbi Algorithm.
VD  Viterbi Decoder.
VLSI  very-large-scale integration.
WLAN  Wireless Local Area Network.
WR  write-decisions.
Chapter 1

Introduction

1.1 Background and contribution

The Internet of Things (IoT) is one of the hottest fields in recent years. The world’s attention is attracted by its concept that establishes a smart network of household appliances, vehicles and other physical items. Inevitably, new theories and methods need to be developed to fulfill its requirements, thus the IEEE 802.11 ah standard are proposed to provide an efficient wireless network protocol in the application context of the IoT. IMEC at Holst Centre, Netherlands is researching ultra-low-power (ULP) design and development of the radio integrated circuit transceivers for the 802.11ah standard. This thesis work aims to design and implement low-power and small-area Viterbi Decoders to fit the requirement of the IEEE 802.11 ah standard and the application in the IoT field with the Taiwan Semiconductor Manufacturing Company (TSMC) 40 nm technology.

The contribution of this work is that it systematically studies the principle and integrated circuit (IC) design of Viterbi Decoders. Each component of the Viterbi Decoder (VD) is investigated and designed for low-power and small-area purpose. In addition, based on the literature study, the performance comparison is made among different architectures and algorithms. Then several promising designs are implemented by the Cadence tool. The simulation results concerning bit error rate (BER), power and area are compared and analyzed. The optimal architecture and algorithms are selected which completely satisfy the target.

The contents of this thesis are organized as the following. Chapter 1 introduces the general background and contribution of this thesis work as well as several important related concepts including 802.11ah, IoT and design tools. Chapter 2 briefly illustrates the motivation and problem definition to figure out the issues to be solved. In addition, the related literature is reviewed and the approach and detail objectives of this thesis work are described. Chapter 3 discusses the principle of Viterbi Decoders and introduces the Matlab simulation chain. In the chapter 4, the conventional very-large-scale integration (VLSI) design and implementation are described in detail. Chapter 5 discusses the theory and design of several novel algorithms to save power and area. The implementation of promising designs are introduced in Chapter 6. The simulation results are also illustrated, compared and analyzed in this chapter. The last chapter concludes all the work and lists several recommendations for the future work.
1.2 IEEE 802.11ah and IoT Overview

IEEE 802.11ah [7] is a new long range 802.11 Wireless Local Area Network (WLAN) standard, operating at sub 1 GHz (S1G) bands. Compared to the current IEEE 802.11 WLAN at 2.4 GHz and 5 GHz bands, the low frequency bands of the 802.11ah extend the limitation of the transmission range and it can achieve cost-effective and large scale wireless networks, as described in [8]. It can be used in various situations, such as the large scale sensor network, outdoor WiFi for the cellular traffic offloading.

The IoT is a network with physical devices which can collect, process and exchange data. Consisting with scattered sensors and actuators, important requirements for the communication protocol are low data rate, low-power and long-range. The characteristic of 802.11ah can support the concept of the IoT, thus 802.11ah standard is also regarded as an IoT standard.

1.2.1 Base-band Processing Overview

Figure 1.1 shows a simplified base-band processing flow of the 802.11ah at transmitter and receiver sides in the physical layer which omits the parts that is not concerned in this thesis work. The flow works as that, at the transmitter side, the source bits are firstly encoded and then punctuated at a certain pattern. The standard uses the Bit-Interleaved Coded Modulation (BICM) based on [9], therefore bits are bit-by-bit interleaved and then mapped into symbols, according to the constellation mappings in the standard. After that, the complex symbols are modulated by an orthogonal frequency-division multiplexing (OFDM) modulator and then transmitted to the receiver through the wireless communication channel. At the receiver side, the received bits are first processed by demodulation and de-interleaving. Then a decoder is used to correct the channel errors to obtain decoded bits.

According to the description in [7], the sub 1 GHz (S1G) physical (PHY) data sub-carriers are modulated (also demodulated) with binary phase shift keying (BPSK), quadrature phase shifting keying (QPSK), 16-quadrature amplitude modulation (16-QAM), 64-QAM and 256-QAM. The encoding and decoding apply the forward error correction (FEC) including the binary convolutional coding (BCC) and low-density-parity-check (LDPC) coding, with coding rates of 1/2, 2/3, 3/4 and 5/6. A list of modulation and coding scheme (MCS) is illustrated in Table 1.1 with the single spatial stream (\(N_{ss} = 1\)) and 1 MHz channel.
Table 1.1: S1G MCSs for 1 MHz, \(N_{ss} = 1\)

<table>
<thead>
<tr>
<th>MCS idx</th>
<th>Mod</th>
<th>Rate</th>
<th>Data rate (kbps)</th>
<th>8 us Gl</th>
<th>4 us Gl</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BPSK</td>
<td>1/2</td>
<td>300.0</td>
<td>1333.3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>QPSK</td>
<td>1/2</td>
<td>600.0</td>
<td>666.7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>QPSK</td>
<td>3/4</td>
<td>900.0</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>16-QAM</td>
<td>1/2</td>
<td>1200.0</td>
<td>1333.3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16-QAM</td>
<td>3/4</td>
<td>1800.0</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>64-QAM</td>
<td>2/3</td>
<td>2400.0</td>
<td>2666.7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>64-QAM</td>
<td>3/4</td>
<td>2700.0</td>
<td>3000</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>64-QAM</td>
<td>5/6</td>
<td>3000.0</td>
<td>3333.3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>256-QAM</td>
<td>3/4</td>
<td>3600.0</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>256-QAM</td>
<td>5/6</td>
<td>4000.0</td>
<td>4444.4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1/2 with 2x repetition</td>
<td>150.0</td>
<td>166.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.2.2 Binary Convolutional Encoder and Punctuate Overview

The binary convolutional coding (BCC) encoder of the IEEE 802.11 standard in [10] uses the industry-standard generator polynomials, \( g_0 = 133_{8} \) and \( g_1 = 171_{8} \) of rate 1/2, as shown in figure 1.2. The following equation 1.1 shows the encoding formula. The coding rate \( R \) is defined as the ratio of the number of source bits over the number of encoded bits. The memory of the encoder is 6 and the constraint length \( K \) is defined as the memory size plus 1, which is 7 in the standard.

\[
\begin{align*}
A & : b_{0,n} = d_{n-6} \oplus d_{n-5} \oplus d_{n-3} \oplus d_{n-2} \oplus d_n \\
B & : b_{1,n} = d_{n-6} \oplus d_{n-3} \oplus d_{n-2} \oplus d_{n-1} \oplus d_n
\end{align*}
\] (1.1)

Figure 1.2: Convolutional encoder (\(K=7\))

In addition, the decimal number represented by the encoder memory is defined as the state of the encoder at time \(n\) is, shown in the equation 1.2.

\[
s_n = \sum_{j=1}^{K-1} d_{n-j} \cdot 2^{K-j-1}
\] (1.2)

Therefore, the encoding process can be regarded as a state machine, which can be extended as a trellis graph. Figure 1.3 illustrates an example of the trellis, with a rate 1/2 and \(K=3\). Each array represents a state transfer and each column shows all the states at one encoding step.
In the 802.11ah standard, the higher coding rate is achieved by puncturing in a specific scheme, shown in figure 1.4 which takes rate 3/4 as an example. Firstly, an 8 bits source stream are encoded to 16 encoded bits and then the 4th and 5th (stolen bits) are punctuated at every 6 bits. At the receiver side, the received bit sequence is de-punctuated by inserting dummy bits in the same scheme.

1.3 Design Tools Overview

To achieve high performance, the digital application-specific integrated circuit (ASIC) design flow is applied and Cadence software is used as an electronic design automation (EDA) tool with the Taiwan Semiconductor Manufacturing Company (TSMC) 40 nm technology library. The figure 1.5 shows the digital IC design work flow.

After the functional and logical design, a set of register-transfer language (RTL) program by a hardware-define language (HDL) is written whose behavior shall be verified with the Cadence NCsim. Once the behavioural verification is finished, Cadence RTLcompiler is applied to logical
synthesize the design and generate a gate-level netlist together with the TSMC 40 nm library and user-defined constraints (timing, power, etc.). Then NCsim is used to verify the behaviour of the netlist and check whether its performance can satisfy constraints. The previous procedure is called front-end simulation, which focuses on the verification of the logic. The power consumption and area can be coarsely estimated at by PrimeTime and RTLcompiler respectively the end of the front-end simulation.

Cadence Encounter continues the following steps for the physical design, called the back-end simulation and more complicated factors including routing, physical dimension, delay are considered. The design steps contain floor-planning, placement, clock tree synthesis, route and post-route optimization. The layout is generated at the end. At this stage, a much more accurate power and area can be obtained.

Figure 1.5: Cadence Digital IC design flow
Chapter 2

Problem Definition

This chapter describes the motivation of this thesis work and application context of the target design in the 802.11 ah standard. Also, the problem definition is figured out to illustrate a clear objectives. The related literature is reviewed especially the low-power and small-area methods to provide possible solutions for the further study. The last section shows the approach in detail.

2.1 Motivation

IMEC at Holst Centre, Netherlands is researching the ultra-low-power (ULP) design and development of the radio integrated circuit transceivers for the 802.11ah standard, aimed at transmitting and receiving data from various sources with low power. The digital application-specific integrated circuit (ASIC) design flow with Taiwan Semiconductor Manufacturing Company (TSMC) 40 nm technology is used in the development to achieve high performance. In the 802.11ah standard, after the data is encoded, processed and transmitted, the received signal needs to be recovered at the receiver side by a series of operation. The convolutional decoding is an important process which corrects the error and decodes received bits. The assignment aims at investigating binary convolutional coding (BCC) decoders for the use in ULP radio ICs.

The 802.11ah standard aims to be applied in the long transmission range of the WLAN and achieve cost-effective and large scale wireless networks, especially for the IoT. The main requirements include low-power consumption and small-area due to the limited power and area capacity in the application context. In addition, the transceiver is required to support the modulation and coding scheme (MCS) idx 0 to 5 in Table 1.1, which should achieve around maximum 2.7 Mbps throughput. Furthermore, according to the standard [7], decoding by the Viterbi Algorithm/Decoder is recommended. Therefore, the general target is to design and implement a low-throughput, low-power, small-area IC of a Viterbi Decoder (VD) for the 802.11ah standard.

2.2 Problem

A soft-decision VD (aka soft decoder or soft decoding) use the reliability of each bit as the input sequence while a hard-decision VD (aka hard decoder or hard decoding) uses logical value of each bit as the input sequence. It has been known that a soft-decision VD achieve about 2dB more coding gain than a hard-decision VD. Thus the soft decoder is usually used for better bit error
rate (BER) performance and higher sensitivity of the received signal power. However, more than one bit is needed to represent the reliability of each input bit, compared to the logical value input. So the soft decoder operates more bits during the decoding and thus consumes more power and area which limits its usage in the IoT standard. Therefore the main problem is how to design a soft decoder that consumes low-power and small-area as much as possible with acceptable BER performance. More precisely, the following problems need to be solved in this thesis work.

1. Based on the previous work of soft decoders, how much is the expected power consumption with current 40nm technology? To achieve low-power and small-area in the similar application context, what kind of architectures and algorithms have been used before? How to innovate and implement those designs in the ICs to optimize the trade-off between area, power and BER?

   The previous work should be reviewed, focusing on the power performance. For a fair comparison, all the power will be scaled and normalized to energy per bit and an accurate target of the power shall be determined.

2. What kind of architectures and algorithms fits the requirement of the application context for this thesis work? Based on the requirement, suitable architectures and algorithms need to be proposed and compared in terms of the power and the area etc.

3. For a promising design plan, how to implement the IC and evaluate its power, area and BER by the software simulation? What is the accuracy of the results? What kinds of factors influence the simulation accuracy?

   Cadence tools should be used to implement the IC design. After the system behavioural description by the register-transfer language (RTL), the front-end synthesis is used for functional verification and evaluation of the area and power. Then after the floorplan placement and layout configuration, the back-end synthesis is used also for the physical verification and evaluation. The results of the front-end and back-end simulation should be compared and analyzed to evaluate the accuracy. Also, the area and power will be normalized and compared with that of the state-of-art design based on the previous design review.

2.3 Objectives

To answer the first problem to determine a fair target in the section 2.2 and determine an appropriate power target, this thesis work summarizes and scales previous IC designs of the VD. The equation to fairly scale the energy per bit is based on the scaling formula in [11], taking important parameters into account, shown in the Equation 2.1.

\[ E_{\text{norm}} = P_{\text{norm}} \times [TH \times N]^{-1} \times (0.7 \times 8 \text{ bit/} \omega_\gamma + 0.3 \times 128/L_{\text{SMU}}) \] (2.1)

\[ P_{\text{norm}} = P_D \times v^2 \] (2.2)

Equation 2.2 shows how to scale the power dissipation which considers the supply voltage scaling factor \( v \) (the ratio of the real supply voltage and the nominal supply voltage).
CHAPTER 2. PROBLEM DEFINITION

Figure 2.1 shows the figure of merit and predicts the trend based on the work in [2, 12, 13, 14, 15, 16, 17, 6, 18, 19, 20, 21, 22, 23, 24, 25, 26]. The bottom vertical axis is the CMOS feature size (nm) and the horizontal axis is the scaled energy per bit (nJ/bit/state). The upper vertical axis indicates the standard nominal supply voltage of the corresponding technology feature size. To predict the trend, two curves (black curve and red dashed curve) are drawn based on Equation 2.1 and 2.2 to cover all the scaled energy per bit in different CMOS technology and supply voltage. Therefore, the target design shall achieve a scaled energy per bit per state between 0.2 and 4 pJ/bit/state which is equivalent to 7 to 80 pJ/bit with 40 nm technology and nominal supply voltage. The equivalent power target is from 50 µW to 1000 µW. As performance results in [17, 26] are not reliable enough, the new lower bound (drawn by the red full line) is about 200 µW after removing those two points. Furthermore, according to the current receiver’s performance in IMEC at Holst Centre, to obtain a 2dB gain of the signal detection sensitivity, the receiver antenna needs to consume 400 µW more power. Therefore, at the same signal detection sensitivity, the overall receiver can consume less power if the soft decoder, which can achieve about 2 dB more coding gain than hard decoder, can consume less than 400 µW. So the target power is 200 µW to 400 µW. In addition, considering the Table 1.1 and possible support to more modulation and coding scheme (MCS) index in the future, the requirement throughput is 5 Mb/s.

Figure 2.1: Figure of Merit

2.4 Overview of literature study

To answer the first problem about the related work study in the section 2.2, this section introduces the overview of the literature study. Generally, a conventional VD mainly consists of a Branch Metric Unit (BMU), a Path Metric Unit (PMU) and a Survivor Memory Unit (SMU), which will be explained in detail in the next chapter. The work in [1, 27] presents a comprehensive analysis of different design aspects of a VD. The design space contains the three components of a VD at algorithmic, word, bit levels.

Starting from the input of a VD with the fixed-point arithmetic, the authors in [28] studies the quantization loss from the additive white Gaussian noise (AWGN) channel with the BPSK and QPSK modulation and proposed a new quantization scheme. The idea is an optimal trade-off between the quantization bit and BER performance loss. In addition, the work in [29] applies
log-likelihood ratio (LLR) to calculates the Branch Metric (BM) with a soft output demodulator. The work concludes that the LLR-based method is better compared with the conventional method (maximum likelihood bit metrics).

Given the BM, the PMU operates in the Add-Compare-Select (ACS) recursion which determines the throughput and clock frequency of the VLSI implementation. To increase the throughput, several conventional methods such as pipelining can be used. However, those methods can not be applied at the nonlinear data dependent nature of the ACS recursion. To conquer the bottleneck, [2] combines the calculation of $k$ steps into 1 steps, called radix-$2^k$ method, which accelerates the throughput $k$ times at the same clock rate. Furthermore, the IC design of a VD can even achieve Gb/s-level throughput in the work [30] by parallel block processing. However, as the throughput requirement in this thesis work is just a few Mb/s-level, state-parallel and even state-sequential architecture can be applied according to the work in [14] to achieve small-area and low-power. Furthermore, the work in [31, 32] modelled an efficient management architecture to schedule the computation of state metrics, allowing the mapping of the trellis onto an arbitrary number of ACSs with a 100% processor utilization.

Furthermore, due to continuous addition of the branch metrics in the PMU, the range of the state metrics is potentially unbounded which may causes buffer overflow. To solve the problem, [33] applies the modulo technique to normalize the state metrics and it is the most local and uniform approach compared with other techniques. Figure 2.2 shows a summary of the design space for the BMU and PMU.

![Design space for the Branch Metric Unit and Path Metric Unit](image)

The SMU updates and stores the decision-bits from the PMU and produce the decoded sequence. The work in [34] proposes a direct implementation architecture, called the register exchange algorithm (REA). Although the decoded latency is short in the REA, its large wiring area and high power consumption limits its application only within small number of states. In the application context of 802.11ah standard, another method called traceback algorithm (TBA) is an alternative and the work in [3] derives a generalized method for the survivor memory management to satisfy arbitrary requirements of the memory size and the number of memory access pointers which is called M-pointer method. Furthermore, for the low-power purpose, the traceforward (TF) and pre-traceback (PTB) algorithms are proposed in [35] to eliminate the traceback stage and reduce memory access frequency respectively.

The previous methods including the register exchange algorithm (REA), the traceback algorithm (TBA) as well as the TF and the PTB algorithms are called standard algorithm. To further reduce the power consumption, another category method called adaptive techniques are applied to achieve dynamic power consumption in different signal-to-noise ratio (SNR). For example, [36, 37] introduces two adaptive algorithms called T-algorithm and M-algorithm which reduce the number of states in the calculation based on the path metrics. Those techniques can
CHAPTER 2. PROBLEM DEFINITION

dynamically adjust the number of the working ACS recursion to save the power consumption at
the costs of arithmetic overheads and the BER performance. A predictive method is proposed
in [6] based on the fact that current survivor path has high possibility to re-merge to the previous
survivor path at high SNR. An extra unit is needed to buffer the previous path and check the oc-
currence of the re-merge. This method won’t result in performance loss and aim to save the power
by reducing the number of the memory access. The work in [38] derived a Scarce-State-Transition
(SST) algorithm to reduce the switching activity of a VD by adding a simple pre-decoder and
pre-encoder before the decoding process.

As described in [27], the benefits of the adaptive algorithms are quite dependent on the
channel and application context and all the published results are mostly based on the AWGN
channel. Experiments and simulation with accurate channel modelling in possible application
cases are necessary when applying those algorithms. Figure 2.3 shows a summary of the design
space for the SMU.

Figure 2.3: Design space for Survivor Memory Unit (SMU)

Furthermore, to update and store the decision-bits from the PMU, an extra memory block
is needed for the traceback algorithm (TBA). According to many designs such as [27], the static
random-access memory (SRAM) is an option. The work in [39] introduces a novel standard cell-
based memory (SCM) for the low voltage operation. Taking the TSMC 40nm technology as an
example which is the same as this thesis work. The standard cell-based memory is dramatically
better in the power at low voltages. To save the power, the voltage and clock frequency of the
radio ICs for the 802.11ah will be scaled down in the practice. So SCM is thus very competitive
with the power dissipation for the application context (∼1k bits size) of this thesis work, compared
to commercially available macro memories.

2.5 Approach

The primary approach used in this thesis work is the literature study, mathematical analysis and
software simulation with electronic design automation (EDA) tools. More precisely,

1. Firstly, formulate each step of the Viterbi Algorithm in the 802.11ah standard. Then build
a Matlab simulation chain to evaluate the BER at different SNR.

2. After studying various literature to research how those work design a low-power, small
area VD in terms of architecture and algorithm for the low-throughput application context,
select potential algorithms by mathematical analysis, simulate and evaluate the result in the Matlab chain.

3. Meanwhile, the performance of the VD in the literature including the power, area, throughput and IC design technology will be collected, normalized and compared to derive an appropriate target for this thesis work.

4. Once the results works well, design and implement the algorithm with the application-specific integrated circuit (ASIC) design flow and verify the BER performance by comparing the output with the Matlab simulator.

5. Finally, implement the selected designs by Cadence digital IC design tools. Evaluate and compare the performance and select the optimal design. Also, compare the performance with the target and the previous collected work.

The thesis work mainly focuses on the design of the VLSI implementation at algorithmic level. The performance, including the BER, the power and the area are simulated and obtained based on the AWGN channel.
Chapter 3

Viterbi Decoders

This chapter discusses the principle of a conventional VD in the mathematical perspective. The processes in each component, including the Branch Metric Unit (BMU), the Path Metric Unit (PMU) and the Survivor Memory Unit (SMU) are shown and modelled. A Matlab simulation chain for the further work is also introduced at the last section.

3.1 Introduction

Figure 3.1 illustrates the processing flow in a mathematical way. Consider a sequence (totally T bits) of source bits \( \{d_n\} \) is encoded to a sequence of bits \( \{b_{1,n}, b_{2,n}\} \) and \( n \) represents a random member of the sequence. After the puncture, the interleaving and the modulation, a sequence of symbols are generated \( \{c_n\} \). Assume the channel adds noise \( N \) to the symbol \( c \), so the received symbol \( y \) are

\[
y = c + N
\]  

(3.1)

An approach called Maximum Likelihood Sequence Estimation (MLSE) is applied to find the source bit sequence \( \hat{D}=\{\hat{d}_n\} \), which has most likely coded the received symbol sequence \( R=\{y_n\} \).

\[
\hat{D} = \arg\{\max_D P(D|R)\}
\]  

(3.2)

\(^1\)Bold represents complex value
By applying conditional probability density function (PDF) and logarithm, the formation can be expressed as

\[
\hat{D} = \arg \left\{ \max_{\hat{D}} \sum_{n=0}^{T-1} \log(P(\hat{d}_n|y_n)) \right\}
\]

where \( T \) is the length of the source bit sequence, \( y_n \) is the corresponding symbol(s) of \( d_n \)

Since to obtain the maximum likelihood sequence for the decoding, all the possible sequences should be traversed. This will lead to the exponential increment of number of the traversal and calculation with the length of the sequence. Therefore, to efficiently solve the problem, the principle of Viterbi Algorithm (VA) \([34, 40]\) is applied so that only the most likely path (survivor path) needs to be remained when two paths merge in one state.

In general, the decoding procedure is that, in step 1, at each decoding step of a trellis, calculate the probability that the transfer to a state generates the represented encoded bits by received symbols, \( \log(P(\hat{d}_n|y_n)) \). Then in step 2, for each state, add the probability to the accumulated probability of its predecessor states, compare them and select the optimal transfer and then accumulate the probability. Finally, in step 3, after a certain amount of steps, traceback from the optimal state by retrieving its previous path and decode. Appendix A explains the theory in detail.

Define the probability of the transfer to one state as Branch Metric (BM) and the unit to calculate it as Branch Metric Unit (BMU). Similarly, in step 2, define accumulated probability as Path Metric (PM), the whole calculating unit for each state as Path Metric Unit (PMU). Each calculating element is defined as Add-Compare-Select Unit (ACSU). The unit in step 3 for retrieving a path (called final survivor path) and decoding is defined as Survivor Memory Unit (SMU).

### 3.2 Branch Metric Unit

This section introduces two methods for calculation in the Branch Metric Unit (BMU), maximum likelihood bit metrics and log-likelihood ratio (LLR) soft bit metrics.

#### 3.2.1 Maximum Likelihood Bit metrics

The Branch Metric Unit (BMU) calculates the probability that the represented encoded bits by received symbols corresponds to a state transfer for each state. Also, because of the logarithm in equation 3.3, the probability of a state transfer is the sum of the probability of each single received encoded bit, which is defined as the bit metrics. In the 802.11ah standard, the application of the Bit-Interleaved Coded Modulation (BICM) requires complex implementations to calculate the probability according to description in [9].

The work in [41] proposed a decoding scheme to calculate the BM. Based on the modulation and coding scheme (MCS) in Table 1.1, assume that the in-phase and quadrature components of a received complex symbol \( y \) consists of \( m \) interleaved bits respectively. Define \( y = y_I + jy_Q \) as symbol components and \( \{y_{I,1}, \ldots, y_{I,m}, y_{Q,1}, \ldots, y_{Q,m}\} \) as the bit sequence of in-phase and quadrature components. For each symbol \( y \), the bit metric of each in-phase and quadrature bit \( y_{I,k}, y_{Q,k} \)
(k ∈ {1, ..., m}) need to be calculated. Only y_{I,k} is discussed here as y_{Q,k} is the same. For y_{I,k}, define \( S_{I,k}^{(0)} \) which comprises the corresponding source symbols \( c \) before the transmission with logical value 0 in position (I, k) and the complementary value is \( S_{I,k}^{(1)} \). So for one bit represented by y_{I,k}, the bit metric is

\[
m_d(y_{I,k}) = \log \{ \sum_{\alpha \in S_{I,k}^{(d)}} P(y|c = \alpha) \} = \max_{\alpha \in S_{I,k}^{(d)}} \log (P(y|c = \alpha)), \text{ where } d=0,1
\]

The equation 3.4 applies the approximation method from the work in [42] which results in negligible performance loss.

If the communication channel is the additive white Gaussian noise (AWGN) channel, the equation 3.4 can be written as

\[
m_d(y_{I,k}) = \frac{E_s}{N_0} \min_{\alpha \in S_{I,k}^{(d)}} \{ |y - \alpha|^2 \} + \ln \sqrt{\frac{E_s}{\pi N_0}}, \text{ where } d=0,1
\]

The pairs \((m_0, m_1)\) of each received encoded bit are sent to the Viterbi Decoder as an input sequence after calculation and interleaving.

### 3.2.2 log-likelihood ratio soft bit metrics

The obvious disadvantage of the maximum likelihood bit metric explained above is the large calculation complexity and the storage size. To solve the problem, the work in [29] applies the log-likelihood ratio (LLR) to calculate the bit metric.

The log-likelihood ratio (LLR) of \( b_{I,k} \) is defined as

\[
LLR(b_{I,k}) = \log \frac{P[y_{I,k} = 1|\mathbf{y}]}{P[y_{I,k} = 0|\mathbf{y}]} = \log \frac{\sum_{\alpha \in S_{I,k}^{(1)}} P(y|c = \alpha)}{\sum_{\alpha \in S_{I,k}^{(0)}} P(y|c = \alpha)} = \max_{\alpha \in S_{I,k}^{(1)}} P(y|c = \alpha) - \max_{\alpha \in S_{I,k}^{(0)}} P(y|c = \alpha)
\]

The idea of the work is to use a soft-output demodulator to generate a LLR of each bit to indicate the reliability and sends to a VD as an input sequence after de-interleaving. Furthermore, the work in [29] proves that the branch metric calculation by the equation 3.4 is equivalent to use the LLR and also proposes a simplified method to calculate the LLR. Two methods are shown in the Table 3.1.

It can be observed that the LLR soft bit metric is better which requires half of the input memory and has less calculation complexity. Therefore, it is applied in this thesis work. Furthermore, the BER performance difference is shown in Figure 3.2. Clearly, a soft-decision VD has around 2dB performance gain compared to hard-decision decoding which conforms to the result in the literature.
### Table 3.1: Comparison of maximum likelihood and LLR soft bit metrics

<table>
<thead>
<tr>
<th>Bit metric (decoded bit 0)</th>
<th>ML bit metrics</th>
<th>LLR soft bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_0$</td>
<td>$LLR$</td>
<td></td>
</tr>
<tr>
<td>Bit metric (decoded bit 1)</td>
<td>$m_1$</td>
<td>$-LLR$</td>
</tr>
</tbody>
</table>

#### 3.3 Path Metric Unit

The Path Metric Unit (PMU) adds the Branch Metric (BM) to the Path Metric (PM) of its predecessor states for each transfer to a given state, compare them and select the optimal transfer and then accumulate the probability to update its PM. In each trellis step $k$, a branch metric $\lambda_k^{(m,i)}$ is defined to denote the branch metric of the $m$-th branch leading to state $i$ at step $k$. Furthermore, the Path Metric (PM) represents the accumulation of the branch metrics of one specific path through the trellis of the sequence $\hat{B}$. $\gamma_k^{(m,i)}$ represents the path metric for a branch $m$ leading to state $i$ at step $k$, where $m \in \{0,...,M-1\}$ is the branch label among all the $M$ paths leading to state $i$.

In the PMU, a concept called state metrics $\gamma_{i,k}$ is defined to represent the selected PM for the survivor paths of the state $x_k = i$ at trellis step $k$. In addition, the path metrics $\gamma_k^{(m,i)}$ leading to state $x_k = i$ are calculated by adding the state metrics of predecessor states and the corresponding branch metrics. The predecessor state $x_{(k-1)}$ is decided by the state transition function $Z()$:

$$x_{k-1} = Z(m, i)$$  \hspace{1cm} (3.7)

Which illustrates one branch path $m \ (m \in 0,...,M-1)$ leading to state $x_k = i$. Therefore, the PM is calculated as
\[ \gamma_k^{(m,i)} = \gamma Z(m,i),k-1 + \lambda_k^{(m,i)} \]  

(3.8)

The State Metric (SM) is thus calculated as

\[ \gamma_{i,k} = \max \left\{ \gamma_k^{(0,i)}, \ldots, \gamma_k^{(M-1,i)} \right\} \]  

(3.9)

To finally allow the Survivor Memory Unit (SMU) to retrieve the maximum likelihood path and decode, decision-bits sequence generated by the Add-Compare-Select Unit (ACSU) of all states and all steps need to be stored. Decision-bits, \( d_{i,k} \), represents the bit shifted out from the state \( i \) predecessor at time step \( k \) after a state transfer.

Figure 3.3 shows an example of the operation in an ACSU of a PMU for the binary convolutional coding (BCC) in 802.11ah standard. To determine the State Metric (SM) of state 32, \( \gamma_{32,n} \), at step \( n \), the BM of two predecessors states, \( \lambda_{n}^{(0,32)} \) and \( \lambda_{n}^{(1,32)} \), calculated by BMU are firstly added to the corresponding SM, \( \gamma_{0,n-1} \) and \( \gamma_{1,n-1} \), to obtain the PM, \( \gamma_{n}^{(0,32)} \) and \( \gamma_{n}^{(1,32)} \). Assume \( \gamma_{n}^{(0,32)} > \gamma_{n}^{(1,32)} \), so after comparing the PM, \( \gamma_{n}^{(0,32)} \) is selected and thus the SM \( \gamma_{32,n} = \gamma_{n}^{(0,32)} \). Also, the decision bit is \( d_{32,n} = 0 \) for retrieving, as \( 0 = Z(d_{32,n}, 32) \).

Figure 3.3: Illustration of ACSU in a PMU

### 3.4 Survivor Memory Unit

The Survivor Memory Unit (SMU) is applied to retrieve a path and decode. Two problems arise in the practical decoding that firstly it is not usually the case that the end state is known in the continuous transmission, so how to determine the start state of a path retrieving should be solved. Second, the decoding latency and the size of SMU is proportional to the length of the trellis, it is not practical to use a memory whose length is the same as the source bit sequence. To solve the problem, the truncation property in the work [43, 44] is used to obtain approximate result by a fixed length of decision-bits memory at the cost of inappreciable performance losses and limited extra implementation effort.

In the Figure 3.4, \( N \) survivor paths merge at time instant \( k \) into a single state. For the trellis steps smaller than \( k-D \), the paths will most probably merge into a single path called final survivor path, thus it is possible to find the final survivor path starting from the optimal state at step \( k \) for decoding. The distance \( D \) is defined as survivor depth (truncation depth). In such case, for steps smaller than \( k-D \), the only one final survivor path can be determined after the processing at step \( k \). This will possibly resulting in a fixed latency of \( D \) trellis steps for decoding the continuous transmission. Furthermore, the survivor memory can be truncated and only a fixed number of decision bits needs to be stored, i.e. \( d_{i,j} \) where \( i \in \{0, \ldots, N-1\} \) and \( j \in \{k-D, \ldots, k-1, k\} \).
In addition, it is also possible to find the final survivor path from other random state at step k. According to the definition in [43, 44], at the step k, the procedure that the path with the optimal state metric is chosen to determine the final survivor is called best state decoding. Otherwise, it is called fixed state decoding if an arbitrary path is chosen. With the same survivor depth, the best state decoding performs better which needs more hardware costs to find the optimal state, compared to the fixed state decoding.

Therefore, the general structure of the Viterbi Decoder is shown in Figure 3.5. The Branch Metric Unit (BMU) calculates the branch metrics after receiving the channel symbols and the results value are input into the Path Metric Unit (PMU) for all states recursively. The Survivor Memory Unit (SMU) stores and retrieves the final survivor path by decision bits for the decoding.

3.5 Simulation Platform

To provide an appropriate experiment platform to assist the further study of a VD, a stable Matlab simulation platform has been built, which simulates the process at the transmitter and receiver sides for 802.11ah standard. The platform covers the main operation including encoding, punctuation, modulation, adding noise, demodulation, quantization and decoding. Figure 3.6 shows the simulation platform with the Matlab based on an example in [45].

The Matlab telecommunication toolbox is applied in the platform and the core process is the conventional Viterbi Algorithm. The appendix B shows the pseudocode. The variables in the simulation chain are the encoded rate, the punctuation pattern, the signal-to-noise ratio (SNR), the modulation schemes, the quantization bits of received symbols and the decoded algorithms. To prepare the IC design, the platform can be used to evaluate the BER performance for a given method and study the influence of different variables.
CHAPTER 3. VITERBI DECODERS

Figure 3.6: Simulation Platform based on Matlab
Chapter 4

Design and implementation of architecture

This chapter discusses in detail the VLSI design of three main components of a conventional Viterbi Decoder (VD) including the quantization, precision and normalization scheme. Necessary results are also shown to indicate the performance at different conditions. Low-power and small-area are the two main concerned factors and several trade-off are made based on the application context of this thesis work.

4.1 Branch Metric Unit

Branch Metric Unit (BMU) calculates the probability that the received bits are encoded by the state transfer to a given state and it is the smallest unit of a conventional VD concerning both the power and area. This section introduces the quantization of its input and design architecture.

4.1.1 Quantization Scheme

Section 3.2 introduces two methods to calculate the Branch Metric (BM) which are the maximum likelihood bit metric and the log-likelihood ratio (LLR) soft bit metric. Because of the advantage on the storage size and calculation complexity, the LLR soft bit metrics is chosen in this thesis work. The hardware complexity of a conventional VD in the IC design depends linearly on the number of bits of the Branch Metric (BM) and path Path Metric (PM) which are both the calculation results of the input. Therefore the it is necessary to minimize the number of bits of input and the bit error rate (BER) performance loss to reduce the power and area by the input quantization.

According to the work in [28], uniform quantization scheme is usually applied in the binary convolutional coding (BCC) decoding. Assume a sequence of the received signal is sent to a soft demodulator which outputs a sequence of unquantized LLR with a range $(-\infty, \infty)$ and $J_i \in \{\text{sequence of LLR}\}$. If a q-bit signed integer is used to represent $J_i$, so firstly map $J_i$ to interval $\pm \Delta/2, \pm 3\Delta/2, \ldots, \pm (2^{q-1})\Delta/2$, where $\Delta$ is the spacing. Each interval is represented by $0, \pm \Delta, \pm 2\Delta, \ldots, \pm (2^{q-1})\Delta$ which maps to $0, \pm 1, \pm 2, \ldots, \pm (2^{q-1})$ as the quantization value. Figure 4.1 shows a quantization scheme example where $q=3$. The idea of quantization is to efficiently extend the representation range while increasing the number of representation levels by choosing
appropriate q and $\Delta$. Therefore q determines the range and $\Delta$ determines the accuracy of the quantization.

![Quantization Example](image)

Figure 4.1: An example of a q=3 quantization

The work in [28] studies the uniform quantization and proposes an optimal quantization scheme where q=3 and the BER loss is only 0.14dB compared to unquantized input. The parameters of the binary convolutional coding (BCC) is the same as this thesis work such as the encoder and transmission channel except the modulation scheme, only BPSK and QPSK. As one symbol in the 16QAM can represent more encoded bits, if its energy/symbol is the same as QPSK, it can be expected more bits are needed to accurately represent the unquantized LLR. Also, to simplify the following design, a uniform quantization scheme necessary for all the modulation scheme that this thesis work supports. Simulation shall be done to evaluate the BER performance with different combination of q and $\Delta$ and all modulation schemes under different signal-to-noise ratio (SNR) in the additive white Gaussian noise (AWGN) channel.

Figure 4.2 and figure 4.3 shows the simulation results with QPSK and 16QAM. It can be observed that in QPSK, the simulator with the quantizer with q=4 and $\Delta=1.0$ is quite close to the unquantized LLR as input, maximum only 0.1 dB loss. For 16QAM, the BER performances using q=4, $\Delta=1.0$ and q=5, $\Delta=0.5$ are almost the same which are also closed to the BER performances using unquantized LLR. Therefore, to save the hardware complexity and easiness of following design, the uniform quantization scheme q=4, $\Delta=1.0$ is applied.
4.1.2 Architecture Design

In the 802.11ah standard, the binary convolutional coding (BCC) encoder rate is 1/2 and thus there exists 4 combinations of output bits, corresponding to 4 BMs for a state at each step. The LLR soft bit metrics is used to calculate the BM and Table 3.1 shows the calculation for bit metrics. Furthermore, for a q-bit signed integer input LLR, if the range is limited to be symmetrical, \([-2^{q-1}+1, ..., 2^{q-1}-1]\) and thus the range of the BM is \([-2^q+2, ..., 2^q-2]\) if two bit metrics are directly summed up.

Because of the properties of addition and multiplication of inequalities, the result of equation 3.9 won’t be changed if the BM \(\lambda_k^{(m,i)}\) in equation 3.8 is divided by 2 and offset by the maximum LLR, \(2^{q-1}\). The range of BMs will be \([0, ..., 2^q-2]\) which only requires q-bit unsigned branch metrics. So in this thesis work, the word-width of BM is 4. Compared to the usual q-bit LLR input which ranges from \(-2^{q-1}\) to \(2^{q-1}-1\) and requires \((q+1)\)-bit signed branch metrics, this transform scheme has less bit and can saves the complexity of following calculation. Figure 4.4 shows a straightforward design of the Branch Metric Unit (BMU) based on the work in [27]. The operation of division 2 is replaced by shifting. The two inputs are LLRs of received bits after the quantization and the BMU outputs four BM.

As a smallest unit in a conventional VD, the power saving techniques at circuit-level is expected to be trivial for the general power consumption. Therefore, a straightforward design is enough.
CHAPTER 4. DESIGN AND IMPLEMENTATION OF ARCHITECTURE

4.2 Add-Compare-Select Unit

As described in Section 3.3, the Path Metric Unit (PMU) applies the Add-Compare-Select (ACS) recursion, a system of non-linear recurrence equations, to calculate the state metrics with the input Branch Metric (BM). The characteristic of the recursion determines the achievable data and clock rate of a VLSI implementation of a conventional VD. Furthermore, due to continuous addition of the branch metrics, the range of the state metrics is potentially unbounded. This section shows design of the normalization scheme and the architecture design.

4.2.1 Precision and Normalization Scheme

The width of the Path Metric (PM) determines the precision of the PMU. The work in [46] found that the maximum differences $\Delta_{\gamma_{\max}}$ of the state metrics is

$$\Delta_{\gamma_{\max}} \leq \lambda_{\max} \cdot \log_2 N$$  \hspace{1cm} \text{(4.1)}

$\lambda_{\max}$ is the upper bound of the BM and $N$ is the number of states. The work in [2] derives a equation to calculate the number of bits

$$\gamma_{\text{bits}} = \lceil \log_2 (\Delta_{\gamma_{\max}} + k \cdot \lambda_{\max}) \rceil + 1$$ \hspace{1cm} \text{(4.2)}

$k$ is related to the radix-$2^k$ structure and it is 1 in this thesis work which will be explained in the next section. Therefore, the minimum width of the PM is 8 in this thesis work.

The normalization scheme of this thesis work directly applies the techniques in the work of [33] because it is the most local and uniform approach compared to contemporary approaches in the VLSI implementation. The scheme is considered to be local if it is operated within each ACSU.
without the information from others. If the ACSU will not be interrupted by the normalization, it is considered to be uniform.

Based on the bound in Equation 4.3, the work of [33] applies modular arithmetic normalization approach and the State Metric (SM) $\gamma_{i,k}$ is replaced by the normalized value

$$\gamma_{i,k} \equiv (\gamma_{i,k} + \frac{C}{2}) \mod C - \frac{C}{2} \text{ where } \Delta_{\text{max}} \leq \frac{C}{2}$$  \hspace{1cm} (4.3)$$

The modular arithmetic can be implemented by 2’s complement calculation, which is suitable for the VLSI design.

Furthermore, a modified comparator is also proposed in the work [33] to determine the selection of the SM. Consider a comparison between two SM, $\gamma_{1,i,k}$ and $\gamma_{2,i,k}$, define $z(\gamma_{1,i,k}, \gamma_{2,i,k})$ as the logical result of the comparison

$$z(\gamma_{1,i,k}, \gamma_{2,i,k}) = \begin{cases} 1, & \gamma_{1,i,k} \leq \gamma_{2,i,k} \\ 0, & \text{otherwise} \end{cases}$$  \hspace{1cm} (4.4)$$

Assume the width of SM is $p$ bits and let $\overline{\gamma_{1,i,k}} = (\overline{\gamma_{p}}, \ldots, \overline{\gamma_{0}})$ and $\overline{\gamma_{2,i,k}} = (\overline{\gamma_{p}}, \ldots, \overline{\gamma_{0}})$ be the two’s complement representation of two normalized SM. In addition, define $\hat{\gamma}_{i,k} \equiv \gamma_{i,k} \mod (C/2)$ which is the unsigned value of $\gamma_{i,k}$ if sign bit is ignored. Then the modified comparison rule is

$$z(\overline{\gamma_{1,i,k}}, \overline{\gamma_{2,i,k}}) = \overline{\gamma_{p}} \oplus \overline{\gamma_{p}} \oplus y(\hat{\gamma}_{1,i,k}, \hat{\gamma}_{2,i,k})$$

where is $y$ unsigned comparison function

It can be observed that with this scheme, the normalization process is executed within each Add-Compare-Select Unit (ACSU). Also, the implementation of this scheme will only replace the comparator in the convolutional ACSU, so the calculation will not be interrupted. Therefore, it is local and uniform. The VLSI design of this normalization scheme will be introduced in the next section.

### 4.2.2 Architecture Design

According to the definition of the Path Metric (PM) and State Metric (SM) in equation 3.8 and 3.9, Figure 4.5 (a) illustrates a direct design for the ACSU, called 2-way ACSU. Furthermore, with the precision and normalization scheme in section 4.2.1, adders are 2’s complement adders and the comparator implements the modified comparison rule.

To further clearly represent the state transition, the state $x_k$ at step $k$ can be written as $x_j$ and its predecessor is $ix$, where $i$ is the bit shifted out of the encoder, $j$ is the input bit and $x$ represents common bits. Take the transfer from state $100000$ to state $000001$ as an example. Because of the input bit $j=1$, the $i=1$ is shifted out and $x$ is $000001$. Following such way of the representation, the BM of the transfer from state $ix$ to $x_j$ is thus $\lambda_{\text{BM}}^R$. So the equation 3.8 and 3.9 can be written as equation 4.6. The SM of state $x0$ and $x1$ can be both calculated from the SM of common predecessor states $0x$ and $1x$, shown in Figure 4.5 (b), which is a well-known butterfly structure and it is also called radix-2 trellis. To reduce the complexity of the global routing, the work in [2, 32] thus designs the Processing Element (PE) to implement the butterfly structure with two 2-way ACSU inside, shown in Figure 4.5 (c). In such way, the global routing between
two ACSU is converted to the local routing, which reduces half of the global routing. The BM in the figure correspond to the four outputs in the design of the BMU in section 4.1.2, so each PE can integrate one BMU to eliminate its global routing.

\[ \gamma_{x1,k} = \max\{\gamma_{1x,k-1} + \lambda_{k}^{x}, \gamma_{0x,k-1} + \lambda_{k}'^{x}\} \]

where \( j + j' = 1 \) and \( j \in 0, 1 \) (4.6)

Furthermore, higher radix structure is developed in [2], which combines more steps in the PMU into one step. The chip fabrication result in [2] shows that the radix-4 design achieves an increasing of the throughput with a factor of 1.7 and the area with a factor of 2, so the complexity doesn’t increase linearly with the throughput especially in higher radix. Also, as the requirement of throughput is up to only Mb/s level, radix-2 is selected in this thesis work.

### 4.3 Path Metric Unit

The architecture design of the Path Metric Unit (PMU) has two main categories: the parallel block processing which is introduced in [30] and state-parallel recursive processing which are introduced in several work, such as [14]. The parallel block processing targets to achieve Gb/s level throughput and consumes much more area and power, therefore only the state-parallel design is considered in this thesis work, suitable for the application context. Several designs are firstly proposed and then a mathematical analysis is presented concerning the power and area of the VLSI design.

#### 4.3.1 Architecture Design

The general architecture of the PMU is illustrated in figure 4.6. Define the number of ACSU as \( P \), so the number of Processing Element (PE) is \( \frac{P}{2} \). The local routing determines the scheduling of the output SM in a PE and the global routing determines the interconnection of each PE. For the fully-parallel architecture, one step of the ACS recursion for all the states has to be calculated in a single cycle and the design of the local routing will be straightforward, as each radix-2 trellis has its own PE and all of the SM of this step can connect to its corresponding state as the input for the next step. If several radix-2 trellises share one PE and the recursion is sequentialized to several cycles, the design will be much more complex for storing and scheduling.
4.3.2 Routing and Scheduler Design

The work in [32] develops an efficient way to design routing and scheduling. Firstly, define the shared factor $\delta$ (power of 2) as the number of radix-2 trellis that shared one PE, equal to $\frac{N}{\delta}$. So one cycle recursion can be sequentialized to $\delta$ cycles. Such architecture is called shared-$\delta$ architecture.

The two input ports of one PE is defined as port 0 and 1. Then a binary form of a state $s$ can be seen as $[x,y,z]$ mapping to $[\text{Cycle},\text{PE},\text{Port}]$, which indicates that the SM of its next state is calculated in PE $y$ at cycle $x$ through port $z$. $x \in \{0,1,\ldots, (\delta-1)\}$ and $y \in \{0,1,\ldots, (\text{NO. of PE } -1)\}$. As radix-2 trellis is used in this thesis work, $z$ is either 0 or 1. With such scheme, the recursion calculation of each state can be directly mapped to a PE at certain cycles.

Figure 4.7 shows an example of shared-4 architecture for 802.11 ah standard. As the shared factor is 4, 4 radix-2 trellis share one of the total 8 PE and the whole recursion (called one round) calculation is completed in 4 cycles (4 slices). The total number of state is 64 and the binary form of each state can be divided as $[2$-bit,3-bit,1-bit] for scheduling. Taking state 17 as an example, the binary form is $(010001)_2$ and $x=(01)_2$, $y=(000)_2$, $z=(1)_2$. To calculate the SM of its next states, it should be calculated in slice 2 at the port 1 of PE 0. With such scheme, the calculation of 64 states are sequentialized to 4 cycles, 16 states each.
The output decision bits of each slice can be directly sent to a Survivor Memory Unit (SMU) and the SM needs to be scheduled as the input for the next round. For example, consider the output of PE 0 at slice 0 which is the SM of state 0 and 32. The SM of state 0 and state 32 should be the input of the PE 0 at slice 0 and slice 2 respectively. So the local routing is used to schedule the output of each slice. Figure 4.8 shows an example of the usage of the local routing in PE 0 for the shared-4 architecture. Each column on the left side represents the output of the trellis in PE 0 on each slice. With the local routing, all the SM is scheduled to fit the slice input of the next round. For example, the SM of state 0 and 8 is sent as the input for the slice 0 in the next round. A fixed global routing network can connect the output ports to its corresponding PE’s input port. The work in [32] also proves the correctness of such scheme and compared to other designs.

The work in [31] proposes the VLSI implementation of the local routing, shown in figure 4.9 (a). The design achieves the in-place storage so that only N number of the SM memory is needed (N is the number of states). Two index port, \(Idx_0\) and \(Idx_1\), are the reading indexes for current round and the writing indexes for the next round. The shift port indicates whether the reading and writing the SM in this slice should be exchanged. Figure 4.9 (b) illustrates how the local routing works in detail, taking PE 0 of the shared-2 architecture as an example. All the port input is shown. \(s^j_i\) represents the SM of the state \(s\) at slice \(i\), round \(j\). The matrix row represents the SM memory. The state in brackets is the input SM of current slice which is then overwritten by the output SM in bold of current slice. The advantage of this design is the in-place SM reading and writing method and simple control and indexing input, which can consume less power and save memory area. In addition, it can be observed that the input index (\(Idx_0\) and \(Idx_1\)) and shift input are in a cycling loop highlighted with a dashed red rectangle, therefore it is possible to schedule all PE local routing with an uniform control unit, shown in figure 4.10. The control unit can be removed if the architecture is fully parallel without any sharing.

---

**Figure 4.8: Local routing example**

**Figure 4.9: (a)Implementation of local routing (b) Example of the implementation**
4.3.3 Analysis

The variants of the state-parallel design have no BER performance loss and the only issue is the tradeoff between the power and area of the PMU. Define the unit area of the SM memory, the ACSU, the local and global routing network as $A_{SM\text{mem}}$, $A_{ACSU}$, $A_{local\text{routing}}$ and $A_{global\text{routing}}$ respectively. Define the power of the SM memory, the ACSU. Define the unit power of the local and global routing as $P_{SM\text{mem}}$, $P_{ACSU}$, $P_{local\text{routing}}$ and $P_{global\text{routing}}$. The trivial area and power of the BMU and the control unit is omitted. So the area and power of the PMU with state-parallel architecture is quantified in equation 4.7 and 4.8. As the target throughput is the same for every designs, the total power of the SM memory and ACSU shall be the close. Therefore, it can be inferred that the shared architecture has less area and power of global routing at the cost of more power of the local routing. The precise situation should be evaluated by accurate simulations.

\begin{align*}
A_{PMU} & \propto N * A_{SM\text{mem}} + \frac{1}{\delta} * N * A_{ACSU} + N * (A_{local\text{routing}} + \frac{1}{\delta} * A_{global\text{routing}}) \tag{4.7} \\
\text{where } \delta \text{ is shared-factor, } N \text{ is number of states}
\end{align*}

\begin{align*}
P_{PMU} & \propto P_{SM\text{mem}} + P_{ACSU} + N * (\delta * P_{local\text{routing}} + \frac{1}{\delta} * P_{global\text{routing}}) \tag{4.8} \\
\text{where } \delta \text{ is shared-factor, } N \text{ is number of states}
\end{align*}

Inevitably, to satisfy the requirement of the throughput, it is impossible to implement any arbitrary shared factor which is determined by the target throughput. The recursion process of the ACSU determines the throughput and figure 4.12 shows an example of critical path in dotted array. Assume the critical path is $t$ seconds, so the maximum clock rate is $\frac{1}{t}$ Hz. In the application context of 802.11ah, the maximum throughput is also $\frac{1}{t}$ bps. Assume the requirement is $T$ bps and thus the maximum shared factor is shown in equation 4.9.

\begin{align*}
\text{Max shared factor} = \left\lfloor \log_2 \left( \frac{1}{Tt} \right) \right\rfloor \tag{4.9}
\end{align*}
4.4 Survivor Memory Unit

Two main different algorithms are applied in the Survivor Memory Unit (SMU), the register exchange algorithm (REA) [34] and the traceback algorithm (TBA) [3]. This section introduces and compares those two methods concerning the area and power and also illustrates the VLSI design. Finally, the selected memory is illustrated.

4.4.1 Register Exchange and Traceback Algorithm

The register exchange algorithm (REA) directly computes the bit sequences of the survivor paths of each state based on the decisions bit generated by the PMU. As defined in the previous chapter, the decision-bit and input bits associated with the reconstructed path of state \( i \) at step \( n \) are defined as \( d_{i,n} \) and \( u_{i,n} \). Define the input bits associated with the \( m \)'th branch merging into state \( i \) as \( \bar{u}_{i,m} \).

Algorithm 1 shows the detail.

![Algorithm 1 Register Exchange Algorithm](Algorithm1.png)

Figure 4.12 shows a direct VLSI architecture design as an example. At every trellis step, nearly \( N*D \) bits need to be exchanged. Although the register exchange algorithm (REA) has short decoding latency, the large wiring area and high power consumption does not fit the application context of 802.11ah standard.

Instead of exchanging decision bits at every step/cycle, traceback algorithm (TBA) reconstructs the state sequence based on the decisions \( d_{i,n} \) to find the final survivor path after \( D \) (survivor depth) steps. Then the path is traced back \( M \) steps (called decoding depth) further to decode \( M \) bits that are associated with the final survivor path. This method can significant reduce the memory access bandwidth every step at the expense of higher latency. Algorithm 2 shows the detail.

The memory size should be at least \( (D+M)*N \) bits to traceback \( M \) steps after the path reconstruction of \( D \) steps. As traceback algorithm (TBA) has more benefits related to the area and power in the application context of 802.11ah standard, it is chosen in this thesis work.
Algorithm 2 Traceback Algorithm

```
procedure TB
    traceState = startState
    for t = n to n - D + 1 do
        traceState = Z(d_{traceState,t},traceState)
    ▷ Traceback

    for t = n - D to n - D - M + 1 do
        u_{traceState,t} = u_{traceState,d_{traceState,t}}
        traceState = Z(d_{traceState,t},traceState)
    ▷ Decoding
```

Survivor depth is also called traceback length (TBL).

4.4.2 Architecture Design

How to design an appropriate architecture to minimize the area and power consumption is an important issue. The work in [3] defines three basic operations in the traceback.

1. **traceback read (TB)**, which traces a path from a given state on the trellis for D steps.
2. **decoding read (DC)**, which outputs bits corresponding to the path being traced after the traceback read (TB).
3. **write-decisions (WR)**, which writes decision bits to the memory generated by the PMU.

Obviously, the speed of reading should be the same as the speed of writing to achieve 100% work efficiency and avoid possible buffer overflow. The work in [2] defines the traceback recursion rate (TRR) to indicate the ratio of the reading (traceback read (TB) and decoding read (DC)) and writing in the equation 4.10 assuming that the speed of TB and TB is the same. The work in [3] generalizes the situation and derives a set of equations to achieve the minimum memory size with different speed combinations of three operations.

\[
TRR = \frac{M + D}{D} \quad (4.10)
\]

In the practical VLSI design, the traceback recursion rate (TRR) is usually 2, meaning that M=D. In addition, D is around 5*K for the rate 1/2 according to the work in [34], where K is the constraint length. Based on the 802.11 standard and the VLSI design, D is set as 32 in this
thesis work for the rate 1/2. Figure 4.13 shows the BER performance in the simulation of different traceback length. Compared to the theoretical bound, the BER with traceback length 32 has up to 0.2dB performance loss.

![Figure 4.13: the BER performance with different traceback length](image)

In addition, concerning the start state of the traceback, figure 4.14 shows the BER performance with different configurations. It can be observed that the fixed state decoding has up to 1dB performance loss compared to best state decoding. The fixed state decoding can perform quite close to best state decoding if the traceback length is twice. Longer traceback length results in larger decoding latency, memory area and power. Therefore, considering the area and latency, best state decoding method is chosen. Furthermore, instead of multiple-pointer reading method which needs multiple separate memory blocks, single-pointer method is preferred in this thesis work because of the smaller area and lower power of a whole memory and the concision of the implementation. Also, during the decoding read (DC), bits are decoded in reverse order and thus a first-in last-out (FILO) module is needed to recover the right order. Therefore, based on the equations in [3], the minimum memory length is 3*D in the fully-parallel architecture. The memory width is N bits and the latency of decoding is 3*D.

Figure 4.15 illustrates the architecture design of the SMU plus the FILO. The SMU receives the decision bits from the PMU and write to the memory at the address from an address generator. During the traceback read (TB), the decision bits at a given address is read from the memory and the current state determines the bits to be selected with the multiplexer. Then a new state is reached with the selected bits. At the phase of decoding read (DC), the input bits associated with the current state are decoded and sent to the FILO. Once the FILO is full, the decoded bits are output in the correct order.

Define the energy per bit of reading(traceback read and decoding read) and writing is $E_{rd}$ and $E_{wr}$. Considering the shared architecture discussed in section 4.3, assume the number of bits per reading or writing is $N/\delta$ if the shared factor is $\delta$. Furthermore, to obtain the decision bit of one state, only one slice of decision bits is needed to read which will be explained in the implementation section. So the average energy $E_{ave}$ to decode M bits in the SMU by the conventional design is derived in the equation 4.11 for this thesis work. This also indicates that the energy consumption can also benefits from shared architectures by reducing $\frac{\delta-1}{\delta}$ reading operation. Figure 4.16 shows
Figure 4.14: the BER performance with different traceback start state scheme

Figure 4.15: Survivor Memory Unit architecture design

the saving percentage of memory access energy per bit by different shared factors at different ratio of writing and reading energy $\frac{E_{wr}}{E_{rd}}$. It can be observed that the saving is more significant when the ratio $\frac{E_{wr}}{E_{rd}}$ is high, up to 60% in the shared-8 architecture. Also when the ratio $\frac{E_{wr}}{E_{rd}}$ is high, the writing energy dominates the total memory access and the improvement increment will be less.

$$E_{ave} = \frac{2E_{rd} \cdot M \cdot N_{mem} + N \cdot E_{wr} \cdot M}{M} = 2E_{rd} \frac{N}{\delta} + N \cdot E_{wr}$$ (4.11)
4.4.3 Choice of Memory

Equation 4.11 indicates that the reading and writing energy directly determines the energy per bit for decoding and thus it is necessary to choose low-power and small area memory to satisfy the requirement of this thesis work. Many of the designs in the literature such as [2] select SRAMs because of its good characteristics of low-power, reliability and simplicity. However, IoT application context of this thesis work does not require high speed, so the supply voltage can be reduced to save power during the run time. According to the study in [39], many SRAMs such as Synopsys and Virage cannot work below 0.8V with the target CMOS technology of this thesis work, so application-specific memory would be an optimal choice.

The work in [39] introduced a novel dual-port standard cell-based memory (SCM) to overcome the issues of low-power. The document of the standard cell-based memory describes that it is dramatically better in power and leakage figures at low voltages and is competitive with the power dissipation for the application context of this thesis work, compared to commercially available macro memories. According to the equations in [3], this thesis work needs total 96*64 bits=6kbits of memory. The work in [39] evaluated an example in TSMC 40nm technology, which is the same as this thesis work. According to the results, at the similar memory size level of this thesis work, the dynamic dissipation of memory writing is nearly half of that with Virage in the typical case. In addition, in another experiment, the supply voltage can be scaled down to 0.16V. Concerning the area, it is almost the same as Virage with 6K memory size. As the ratio of writing and reading energy $\frac{E_{wr}}{E_{rd}}$ in the SCM and the commercial memory is different, the benefit of the energy (power) saving methods will be different according to equation 4.11. Therefore, SCM is a promising choice in this thesis work and detail performance should be evaluated by multiple simulation experiments.

4.5 Overall design

Summarizing all the components in the previous sections, figure 4.17 shows the overall VLSI architecture design of a conventional VD, consisting of the BMU, PMU, SMU and FILO. Trivial components including the control unit of the PMU are omitted.
Figure 4.17: Overall architecture design
Chapter 5

Design of standard and adaptive algorithm

This chapter introduces three algorithms to save power consumption of Viterbi Decoders. Firstly, two standard algorithms are explained which aim to reduce the memory access energy (power) at the costs of power and area in the Path Metric Unit (PMU) and are channel/application independent without any performance loss. In addition, one adaptive algorithm is introduced, called dynamic traceback length with path prediction algorithm which can also reduce the memory access energy but at the cost of power and area in Survivor Memory Unit (SMU). Its benefit is dependent on the channel/application and the power is dynamic with the signal-to-noise ratio (SNR). The power-saving benefits of all those three algorithms depend on the ratio of writing and reading operation energy of the memory. The VLSI designs and necessary analysis will also be explained in this chapter.

5.1 Traceforward algorithm

5.1.1 Introduction

As described in section 2, the conventional traceback algorithm has three basic operations, write-decisions (WR), traceback read (TB) and decoding read (DC). As the traceback recursion rate (TRR) is 2 in this thesis work, the decision bits are firstly generated and WR to the memory, TB and then DC, totally 1 writing operation (write-decisions) and 2 reading operation (traceback read and decoding read). However, the TB is used to find the possible start state of the final survivor path and does not directly decode bits, thus the work in [35] proposes an algorithm called traceforward (TF) to eliminate TB.

The main idea of the TF algorithm is that instead of TB, it directly finds the start state at the beginning of decoding read (DC). More precisely, in the conventional traceback algorithm, the start state of the final survivor path is found with the traceback operation and corresponding decision bit at each step. However, with the TF, at the beginning of decoding a block, each state is firstly initialized with the state itself, called traceback state, then at each step of the Add-Compare-Select Unit (ACSU) recursion, the traceback state is updated with its predecessor traceback state, called traceforward which imitates the traceback operation. After D steps, the current traceback state of the best state to start traceback read (TB) is thus the start state of the
CHAPTER 5. DESIGN OF STANDARD AND ADAPTIVE ALGORITHM

final survivor path. In such case, the decoding read (DC) can start directly. Define the traceback state associated with state i at step n as $u_{i,n}$. At the beginning of decoding a block, the traceback state is initialized with the state itself, so that $u_{i,n}=i$. Algorithm 3 shows the detail during the write-decisions (WR) at step $n+k$ (kD). Once the TF is finished, the decoding read (DC) phase can start from the $u_{s,(n+D)}$, if $s$ is chosen as the best state.

Algorithm 3 Traceforward Algorithm

```
procedure TF
    for state = 0 to $N-1$ do
        $u_{xj,(n+k+1)} = u_{ix,(n+k)}$
```

Figure 5.1 shows an example of the TF. $t_1$ is the beginning time to decode the bits, so the traceback state of each state is initialized with the state itself. During the WR from $t_1$ to $t_6$, the traceback states are updated with its predecessor traceback state during the ACSU recursion. At $t_6$, $s_1$ is selected as the best state to start TB. With conventional traceback, the start state of the DC is found by TB, shown in red array. With the TF, the traceback state of $s_1$ directly indicates that the start state of DC is $s_0$, shown in the dashed red array.

5.1.2 Architecture design

As the TF runs with the generation of decision bits, the design can utilize the structure of the 2-way ACSU described in figure 4.5. Figure 5.2(a) shows the new design of the ACSU for the TF and define $u_{x,n}$ as the traceback state of state x at time step n. The rest of design including the local and global routing follows the design of the SM. Figure 5.2(b) shows the design of the Processing Element (PE).

As the traceback recursion rate (TRR) is 2 and single-pointer method is preferred in this thesis work, the reading speed should be twice as the writing speed to balance the input and output of the system. As TB can be eliminated if the TF is applied, the TRR is still 2 when reading and writing has the same speed. According to the derivation in [3], the memory length is 3*D.
The work in [4] generalizes the TF algorithm and further reduce the needed memory length to just 2*D by introducing two TF operations at the same time, called sliding window TF algorithm. Figure 5.3 illustrates how it works which even and odd TF overlap to prepare for decoding D/2 bits every D/2 steps and only 2*D length of the memory is enough. Figure 5.4 illustrates a modified design of the ACSU to implement the sliding window TF algorithm. Compared to the figure 5.2, each state has two traceback states with the modified design, shown as \( u^{\text{even}}_{i,n} \) and \( u^{\text{odd}}_{i,n} \).

Concerning the power and area of the PMU, firstly define \( m \) as the TF factor, which defines the overlapped TF operations. \( M \) is 1 and 2 in the initial TF and sliding window TF algorithms.

### 5.1.3 Analysis

Obviously, the benefits of the TF are that it reduces the average reading energy of the memory per bit and also the decode latency. However, because of the TF recursion, the power and area of the Path Metric Unit (PMU) will increase because of the overhead of the local and global routing as well as the memory of traceback state.

Concerning the power and area of the PMU, firstly define \( m \) as the TF factor, which defines the overlapped TF operations. \( M \) is 1 and 2 in the initial TF and sliding window TF algorithms.
CHAPTER 5. DESIGN OF STANDARD AND ADAPTIVE ALGORITHM

Figure 5.4: Design of Add-Compare-Select Unit (ACSU) for sliding traceforward (TF)

respectively. Based on the equations 4.7 and 4.8, define the unit area of traceback state memory as $A_{TF,mem}$. Define the power of traceback state memory as $P_{TF,mem}$. The trivial area and power of the FILO is omitted. So the area and power of the PMU with state-parallel architecture is quantified in equation 5.1. As the local and global routing of the TF is the same as the SM, therefore the original power and area increase $m$ more times. The precise situation should be evaluated by accurate simulations.

$$A_{PMU} \propto N \cdot A_{SM,mem} + m \cdot N \cdot A_{TF,mem} + \frac{1}{\delta} \cdot N \cdot A_{ACS} + (m + 1) \cdot N \cdot (A_{local,routing} + \frac{1}{\delta} \cdot A_{global,routing})$$

where $N$ is number of states and $m$ is the TF factor

$$P_{PMU} \propto P_{SM,mem} + m \cdot P_{TF,mem} + P_{ACSU} + (m + 1) \cdot N \cdot (\delta \cdot P_{local,routing} + \frac{1}{\delta} \cdot P_{global,routing})$$

where $N$ is number of states and $m$ is the TF factor

Concerning the reduction of average energy of the memory access per bit, based on the equation 4.11, equation 5.3 shows the calculation with the fully parallel architecture. Figure 5.5 shows the saving percentage of memory access energy per bit by initial TF algorithm compared to conventional traceback algorithm (TBA). As the energy reduction comes from the elimination of the TB, the lower the writing and reading energy ratio $E_{wr} / E_{rd}$, the higher the saving percentage, up to around 35% in the figure. Furthermore, the saving will be more if sliding window TF is applied as both $E_{rd}$ and $E_{wr}$ is less with 2*D memory.

$$E_{ave} = E_{rd} + E_{wr}$$

Concerning the latency, table 5.1 shows the comparison between three algorithms. Therefore, the TF algorithm has less energy of memory access per decoded bit and latency at the cost of more power and area in the PMU and the overall power will be less if the former saves more power than the cost of the latter.
5.2 Pre-traceback algorithm

5.2.1 Principle and architecture design

In the conventional Viterbi Decoder (VD), a single memory access operates (reading and writing) single step decision bits. The work in [35] proposes a pre-traceback (PTB) algorithm which combines several write-decisions (WR) to one step. During the decision bit generation, instead of writing a single step decision bit of a state at every step, the PTB algorithm allows to write and read multiple steps decision bits at a single step. One traceback operations can thus traceback multiple steps. To clearly illustrates the traceback procedure, define the state x at step n and its predecessor state y at step n-∆ as \( s_y, (n-\Delta) \) and \( s_{x,n} \). \( d_{x,n}^\Delta \) is the combined decision bits of \( \Delta \) steps. \( \Delta \) is called the PTB factor. Equation 5.4 shows how it tracebacks.

\[
\text{s}_{y,(n-\Delta)} = \text{Z}(d_{x,n}^\Delta, s_{x,n}) = d_{x,n}^\Delta (s_{x,n} \gg \Delta) \quad (5.4)
\]

To achieve the multiple-steps traceback, the generation and writing of multiple-step decision bits \( d_{x,n}^\Delta \) is needed. Define the combined decision bit associated with the reconstructed path of state i at step n as \( d_{i,n} \) and define the decision bits of state i at step n as \( d_{i,n} \). Consider the decision bits generation between \( k\Delta \) and \( (k+1)\Delta \) (k is integer). At the step \( k\Delta \), \( d_{i,(k\Delta)} \) are all initialized with 0. Algorithm 4 shows the following operation in this interval and the new combined decision bits are consists of the combined decision bits of the predecessor state and new generated decision
bit. At the last step \((k+1)\Delta\), \(\bar{d}_{i,(k+1)\Delta}\) is output to a Survivor Memory Unit (SMU).

**Algorithm 4** Pre-traceback Algorithm

```plaintext
procedure PTB
    for state = 0 to N - 1 do
        \(d_{xj,(t+1)} = d_{ix,t}d_{state,(t+1)}\)

\[\text{// transfer from state } ix \text{ to next state } xj = \text{ state}\]
```

Figure 5.6 shows an example of the PTB factor 2. Left figure shows the decision bits of two steps in the conventional algorithm and right figure shows decision bits with the PTB algorithm. In the example, the PTB algorithm combines the 2 steps decision bits generation and generate decision bits every 2 steps.

![Figure 5.6: PTB example](image)

Similar as the design of the TF, the original ACSU can also be directly utilized in the PTB. Figure 5.7(a) shows the new design. \(d'_{i,n}\) is the selected result of decision bits of two predecessor states which is the new combined \(\bar{d}_{i,n}\) together with the current \(d_{i,n}\). Figure 5.7(b) shows the design of PE. The rest of the design including the local and global routing follows the design of SM except that that the decision bits output should also connect to the SMU as the conventional design.

![Figure 5.7: (a)Design of Add-Compare-Select Unit (ACSU) for PTB (b)Design of Processing Element (PE)](image)
5.2.2 Analysis

The benefit of the PTB is that it reduces the memory access frequency to $\Delta$ times so that the clock frequency and supply voltage can be reduced to save power. In addition, with the original clock frequency and supply voltage, the PTB will reduce the energy per decoded bit of the memory access. Specifically, define the overhead energy to write and read a separate bit is $E_{\text{overhead}}$ which relates to the multiplexer and control. Assume that $E_{\text{overhead}}$ is constant for the same size memory regardless of its length and word width. The conventional algorithm needs write-decisions (WR), traceback read (TB) and decoding read (DC) to decode a bit, so the overhead energy per decoded bit is $3\times E_{\text{overhead}}$. If $\Delta$ bits are accessed once, the average overhead energy per bit is $\frac{3}{\Delta}\times E_{\text{overhead}}$, reducing the total average energy per bit. Equation 5.5 compares the energy per decoded bit of conventional traceback and the PTB. Obviously, the drawback is the same as the TF, resulting in more power and area of overhead in the Path Metric Unit (PMU). The total power will be reduced if the saving power is more than the overhead.

$$E_{\text{ave}} = 2 \times E_{\text{rd}} + E_{\text{wr}} + 3 \times E_{\text{overhead}}$$

$$E_{\text{ave}} = 2 \times E_{\text{rd}} + E_{\text{wr}} + \left(\frac{3}{\Delta}\right) \times E_{\text{overhead}}$$

5.3 Dynamic traceback length with path prediction algorithm

5.3.1 Introduction and architecture design

The work in [17, 6] proposes an adaptive algorithm, called dynamic traceback length algorithm. The main idea of the algorithm is that the survivor depth(traceback length) is usually configured for the worst case, $5\times K$, so that the traceback path can merge to the final survivor path for decoding, but the decoder does not always work in the worst case. Instead, when the SNR is high, the traceback path will quickly merge to the final survivor path only after a few steps, therefore there is a possibility that the current traceback path can re-merge to the previous path. In that case, if an extra buffer is applied for the previous path, the memory reading to decode current block can be eliminated. The work in [17, 6] proposes an adaptive algorithm, called dynamic traceback length algorithm. The main idea of the algorithm is that the survivor depth(traceback length) is usually configured for the worst case, $5\times K$, so that the traceback path can merge to the final survivor path for decoding, but the decoder does not always work in the worst case. Instead, when the SNR is high, the traceback path will quickly merge to the final survivor path only after a few steps, therefore there is a possibility that the current traceback path can re-merge to the previous path. In that case, if an extra buffer is applied for the previous path, the memory reading to decode current block can be eliminated.

Furthermore, the work in [6] also introduces a path prediction algorithm based on the idea that the best state of each step has higher probability to be in the final survivor path. So during the WR operation, the best state of each step is selected by the comparison tree and is added to the prediction buffer if it is reachable from the previous best state. Figure 5.8 illustrates the idea of the algorithm. The two dashed red rectangles show the possible traceback path re-merge of the current block to the previous block. Figure 5.8 shows the flow chart of two ideas. The path prediction executes during the WR and dynamic traceback length executes during the TB and DC. For simplicity, the whole algorithm is called dynamic traceback length (DTBL) algorithm.

To implement the DTBL algorithm, an extra buffer is needed for path prediction and buffering previous traceback path. To save the buffer size, only decision bits are buffered which calculates the buffer state at each step. Figure 5.9 illustrates the design of the SMU for the DTBL algorithm.
5.3.2 Analysis

As the DTBL only buffers the traceback path from the previous block decoding, there will not exist any BER performance compared to the conventional VD which is verified by Matlab simulations. Another significant benefit of the DTBL is that it reduces the memory reading with the buffer, which depends on the level of the SNR. Define the saving percentage as \( \sigma \) which is related to...

![Figure 5.8: Illustration of the DTBL algorithm](image1)

![Figure 5.9: Design of Survivor Memory Unit (SMU)](image2)
the SNR (EbN0), the equation 5.6 quantifies the energy per bit of memory access based on the fully parallel architecture. Figure 5.10 shows the simulation result to indicate the memory saving percentage of different SNR in the additive white Gaussian noise (AWGN) channel. It can be observed that the reduced memory reading percentage can go up to 95% when EbN0 = 6dB. Also, the total memory access energy per decoded bit of different $E_{wr}/E_{rd}$ ratio is drawn, up to 65% saving when $E_{wr}/E_{rd}$ is 1 and EbN0 is 6dB. However, to buffer the previous traceback path and predict the path, extra components are added, resulting in more power and area. Similar as the previous algorithms, the general system can consume less power when the benefit exceeds the costs.

$$E_{ave} = (1-\sigma)^2 E_{rd} + E_{wr}$$  \hspace{1cm} (5.6)

Figure 5.10: Memory access saving with the DTBL algorithm
Chapter 6

Implementation, results and analysis

6.1 Introduction

This chapter describes the implementation of IC designs of three architectures, two standard algorithms and one adaptive algorithms, which would possibly reduce the power consumption and area. The architectures contain fully parallel, shared-factor 2 and 4. Those algorithms contain traceforward, pre-traceback and dynamic traceback length and path predication. Furthermore, to fairly compare the area and power, control variate method is applied so that architectures are all based on the same conventional algorithm and algorithms are implemented with the same architecture. The criteria to evaluate the low-power performance of a design is energy/bit which is the ratio of power and throughput. As the throughput of all the designs are the same, so it is equivalent to evaluate power to indicate the performance. The results from front-end and back-end simulations including area and power are presented as well as necessary theoretical analysis. An optimal implementation is selected at the end concerning the area and power for the application context of this thesis work.

All designs are implemented firstly by the Verilog to model the behaviour and then is verified by the NCsim. Then RTLcompiler is applied for logical front-end synthesis to generate gate-level netlist with the CMOS library as well as estimated area size. With the PrimeTime, coarse power consumption is obtained. In the front-end simulation, the standard cell-based memory is substituted temporarily by register files. After evaluating the results, only promising designs are selected for back-end simulations. In the back-end synthesis, Cadence Encounter is applied, including floor-planning, placement, clock tree synthesis, route, post-route optimization and layout generation. Then PrimeTime is applied again to estimate the power with the parasitic extracted from the layout.

6.2 Specification

The general specification for the Viterbi Decoder for the 802.11ah standard implemented are

1. K =7(64 states), R=1/2 convolutional decoder,
2. Generator polynomials, $g_0=133_8$ and $g_1=171_8$,
3. 4-bit soft decision input log-likelihood ratio (LLR),
4. 4-bit Branch Metric (BM) and 8-bit Path Metric (PM),
5. Best state decoding,
6. Traceback algorithm with survivor depth (traceback length) $D=32$ and decoding depth $M=32$,
7. Throughput 5 Mb/s,
8. Dual-port standard cell-based memory (SCM),
9. TSMC CMOS 40nm technology with 1.1V supply voltage.

The other code rate with the punctuation is omitted in this thesis work and only QPSK is considered as an example of the modulation scheme when evaluating the BER performance.

### 6.3 Implementation of different architectures

This section discusses the implementation of three architectures, fully-parallel, shared factor 2 and 4 architectures. The idea of designs, front-end and back-end results concerning the power and area as well as analysis are explained in this section.

#### 6.3.1 Design

Before the implementation, it is important to derive the maximum Processing Element (PE) shared factor by equation 4.9. According to the post timing report generated by the Cadence RTL compiler, the critical path is the Add-Compare-Select (ACS) operation which is roughly 18000 ps and thus the maximum clock frequency is 55Mhz. As the requirement throughput is 5Mb/s, the maximum shared factor is 8 which indicates that 2, 4 and 8 are feasible in this thesis work. Therefore, fully-parallel, shared-factor 2 and 4 are to be implemented to investigate the power and area performance.

The design of the Branch Metric Unit (BMU) follows the discipline in chapter 4 which applies 4-bit uniform quantization scheme for the LLR input. The modular arithmetic normalization approach is used in the Add-Compare-Select Unit (ACSU) of the Path Metric Unit (PMU). The number of the PE of the fully parallel, shared-factor 2 and 4 architectures are 32, 16 and 8 respectively. The local and global routing as well as the scheduling follows the section 4.3 and appendix D shows the detail configurations. An extra control unit is implemented in the shared architecture to control the local routing of the PE. In addition, a comparison tree module is added to find the best state for the start of the traceback read (TB).

To implement the Survivor Memory Unit (SMU), as described in section 4.4, the traceback algorithm is chosen in this thesis work with the single-pointer reading method and the best state decoding scheme. As traceback recursion rate (TRR) is 2, the reading speed (traceback read (TB) and decoding read (DC)) should be twice the writing speed. One column represents 64 decision bits in this thesis work. In the fully parallel architecture, decision-bits of all states are generated every cycle and the writing speed is 1 column/cycle. So the reading speed requires 2 columns/cycle. One simple solution to fulfill the required TRR is to generate and write a column of decision bits.
at one cycle and suspend for the next cycle. So the writing speed is 1 column per 2 cycles and a single reading pointer can achieve 1 column/cycle reading speed which fulfills the required TRR. Based on the work in [3], the memory length is 3*D, so the memory size is 96 length with 64 bits of each column, totally 6 kbits. To achieve a 5Mb/s throughput, the clock frequency should be set as 10 MHz.

In the shared-factor 2 architecture, as two radix-2 trellis shares one PE, it takes 2 cycles (slices) to generate all decision-bits and half column is written per cycle. So the writing speed is thus 1 column per 2 cycles. As the decision bits of slice 0 are from state 0 to 15 and state 32 to 47 and the rest decision-bits are from slice 1, so the second most significant bit (MSB) of a traceback state determines whether its decision bit is from slice 0 or slice 1. During the traceback read (TB) and decoding read (DC), it only takes one cycle to read the target decision bit for a given state. In such case, the reading speed can be regarded as 1 column/cycle which conforms to the required TRR. Therefore the clock frequency is 10 MHz and the memory configuration is 192 length with 32 bits of each column.

Similarly, to design the SMU of shared factor 4 architecture, the second and third MSB bits of the traceback state determine which memory column to be read for its decision bits. So the reading speed can also achieve 1 column/cycle. As the writing are 1 column per 4 cycles, 1 column per 2 cycles reading speed is sufficient to fulfill the required TRR. The clock frequency is 20 MHz and the memory configuration is 384 length with 16 bits of each column. Table 6.1 lists the configuration of clock frequency and memory size of three architectures.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Clk freq (MHz)</th>
<th>Memory size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully parallel</td>
<td>10</td>
<td>96 × 64 bits</td>
</tr>
<tr>
<td>Shared factor 2</td>
<td>10</td>
<td>192 × 32 bits</td>
</tr>
<tr>
<td>Shared factor 4</td>
<td>20</td>
<td>384 × 16 bits</td>
</tr>
</tbody>
</table>

Table 6.1: Specification of three architectures

6.3.2 Back-end results and analysis

After the implementation, the front-end results are shown in appendix E.1 with necessary analysis. In the back-end synthesis, the standard cell-based memory (SCM) is used and figure 6.1 illustrates the comparison of power distributions by the back-end synthesis. The total power of the fully-parallel, shared-2 and shared-4 architecture are 247uW, 272uW and 339uW. Similar to situations by front-end synthesis, the PMU and memory dominates the general power, however, the proportion of PMUs is over half, up to 60% because of the usage of the low-power standard cell-based memory. Figure 6.2 indicates the power distribution of the PE of three architectures. The result are similar to figure E.2, showing that the power of local routing is linear to the shared factor and power of all the ACSU is constant. Furthermore, by equation E.1, the power of global routing can be calculated and figure 6.3 shows the results with the comparison tree and control unit. The result is also similar to figure E.3 and it shows that the main difference is the power of the control unit as higher shared architecture need more complex scheduling. The power of traceback unit are also nearly the same.

Concerning the memory power, based on the equation 4.11, equation 6.1 shows the theoretical power, assuming the energy per cycle of reading and writing is $E_{rd}$ and $E_{wr}$ and the frequency of reading and writing is $f_{rd}$ and $f_{wr}$. The calculation results of the power of the fully parallel, shared factor 2 and 4 are 67 uW, 59 uW and 59 uW respectively, which conforms to the ratio of the obtained results. Furthermore, the total memory power saving of shared factor 2 and 4 architectures compared to the fully parallel architecture is 12% and 13%. The ratio of the energy/cycle
CHAPTER 6. IMPLEMENTATION, RESULTS AND ANALYSIS

Figure 6.1: Comparison of power distribution between three architectures by back-end synthesis.

Figure 6.2: Comparison of power distribution of the PE between three architectures by back-end synthesis.

Figure 6.3: Comparison of power distribution of minor modules between three architectures by back-end synthesis.

of standard cell-based memory writing and reading is around 5 and this result conforms to the figure 4.16, considering that the $E_{wr}/E_{rd}$ ratio is higher in the longer memory.

$$P_{mem} = P_{leak} + E_{rd} \cdot f_{rd} + E_{wr} \cdot f_{wr} \tag{6.1}$$

Figure 4.6, 6.5 and 6.6 show the area distribution. The distribution situations are nearly the same as results by the front-end synthesis in the appendix E.1, so the explanation is omitted.

It can be concluded that the power and area distribution by the back-end synthesis is similar to that by front-end synthesis. Also, the benefits of the memory access power saving by shared architectures conform to the theoretical analysis.
6.3.3 Comparison of front-end and back-end results

To understand the estimation accuracy of the front-end and back-end synthesis, appendix F illustrates the power and area comparison. Except the memory, the overall situation is that the power results of back-end synthesis are higher than front-end synthesis. Front-end focuses more on the functional design and verification which estimates the possible physical characteristics to generate more accurate results. Back-end synthesis focuses on the practical physical design including placement and routing (P&R), clock tree synthesis (CTS) and parasitics extraction, so the results are more accurate because it is closer to the reality. PR plans the physical dimension and wire inter-connections and CTS adds buffer to synchronize all the clock signals which dominates the extra power by back-end synthesis. Generally, the overall power estimation difference is around 14% to 23% and the PE power difference is 10% to 15%, so it means that the error comes more from the other parts such as the routing, buffers and parasitics.

About the area distribution, the value and proportion estimation of the front-end and back-end synthesis is quite close, up to 8% difference. The main reason is that as the throughput and
clock frequency is far below the maximum value, so the loose optimization is enough to satisfy the timing constraints. It can be expected that the area in the back-end will increase when the clock frequency approaches the maximum point.

6.4 Implementation of standard algorithm

In this section, two algorithms are implemented, including traceforward (TF) and pre-traceback (PTB) algorithms. The TF has an variant, called the sliding window TF (sliding TF) which is also implemented. The design, results and analysis are explained in this section.

6.4.1 Design

The implementations are all based on the fully parallel architecture. The design of the PMU for the TF, sliding TF and PTB algorithm follow the design of figure 5.2, 5.4 and 5.7 respectively. The PTB factor $\Delta$ is 2 in this section. The rest of the components in the PMU follows the design in fully parallel architecture.

Concerning the implementation of the SMU, the traceback units are the same as the fully parallel architecture and only difference is the memory configuration. The memory length of the TF and sliding TF are 3*D and 2*D with 64-bit width as described in the section 5.1. For the PTB algorithm, as the PTB factor $\Delta=2$, the decision bits of every 2 cycles are thus 128 bits and the memory length and width are 48 and 128 bits. In addition, concerning the clock rate, as traceback recursion rate (TRR) is 2 and traceback read (TB) operation of TF-type algorithm is eliminated, reading and writing can have the same speed and thus the clock frequency is 5 MHz to satisfy the requirement of 5 Mb/s. For the PTB algorithm, the reading and writing speed are 2 and 1 columns/cycle, satisfying TRR is 2, so the clock frequency is also 5 MHz. Table 6.2 shows the clock frequency and memory size of three implementations.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Clk freq(MHz)</th>
<th>Memory size</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTB</td>
<td>5</td>
<td>48 $\times$ 128 bits</td>
</tr>
<tr>
<td>Sliding TF</td>
<td>5</td>
<td>64 $\times$ 64 bits</td>
</tr>
<tr>
<td>TF</td>
<td>5</td>
<td>96 $\times$ 64 bits</td>
</tr>
</tbody>
</table>

Table 6.2: Specification of 3 standard algorithms

6.4.2 Back-end results and analysis

After the implementation, the front-end results are shown in appendix E.2 with necessary analysis. Concerning the back-end results, figure 6.7 shows the power distribution results which also contains the baseline fully parallel architecture with conventional Viterbi Algorithm (called fully parallel in the following). The total power of the PTB and TF is 269 and 268 respectively and their PMU power is 7.5% and 35% more than fully parallel. Figure 6.8 shows the power distribution of the PE, the main part of power in the PMU. The total PE power of the PTB and TF is 115uw and 145 uW. Concerning the ACSU power, PTB and TF has 3%, 23% more compared to fully parallel, which is closed to the front-end synthesis. The local routing and SM memory power has 14% and 55% more than fully parallel of 2 algorithms, so the TF has about 4 times increment than the PTB, higher than the theoretical value 3. Figure 6.9 shows the comparison of minor modules. The global routing of the TF and PTB consume higher power compared to the fully parallel as
the interconnection is more complex. Also, the global routing results are all much higher than the front-end result.

![Power distribution comparison with backend synthesis](image)

**Figure 6.7:** Comparison of power distribution between 2 standard algorithms by back-end synthesis

![Power distribution comparison of PE](image)

**Figure 6.8:** Comparison of power distribution of the PE between 2 standard algorithms by back-end synthesis

![Power distribution comparison of minor modules](image)

**Figure 6.9:** Power distribution comparison of minor modules between 2 standard algorithms by back-end synthesis

To interpret the **memory** power, equation 6.1 is applied and the result is 60 uW and 58 uW of the PTB and TF. Its ratio is lower than the measured value and also it conforms that they save power compared to fully parallel. According to the result, the TF saves 15% power which exactly conforms to the calculation in 5.5 as the ratio of the energy/cycle of standard cell-based memory writing and reading is around 5. However, as the overhead in the PMU is larger than the saving power of memory reading, the overall power of the TF is still higher than the fully parallel architecture.

The **area distribution** of the whole system and the PE are shown in figure 6.10, 6.11 and 6.12. Similar to the simulation in section 6.3, the back-end result of area is closed to the front-end result explained in appendix E.2, so the explanation is omitted.

It can be concluded that the back-end result conforms to the front-end results, which increase 20% to 30% of power and around -5% of area. The measured results also correspond to the theoretical calculation.
CHAPTER 6. IMPLEMENTATION, RESULTS AND ANALYSIS

6.4.3 Comparison of front-end and back-end results

The front-end and back-end results are also compared in this section, shown in appendix G. Similar to the situation in section 6.3, because of routing, buffers and parasitics in back-end process, there exists a difference between the front-end and back-end results and the overall power estimation difference go up to 20% to 30% of 2 standard algorithms, bigger than previous three architectures. This is because the 2 standard algorithms, especially the TF has much more global routing.

About the area distribution, the proportion of each part is the same between front-end and back-end simulations but the front-end overestimate the overall area with around -5% difference. However, the PE area distribution is closed, showing that the over estimation comes from the global routing.

Therefore, comparing the front-end and back-end results, the tool will have more error in the power estimation and overestimate the area when there is more global routing.
6.5 Implementation of adaptive algorithm

This section introduces the implementation of dynamic traceback length with path prediction algorithm. The design, front-end and back-end simulation results and analysis compared to shared-2 architecture are also explained in this section.

6.5.1 Design

The implementation of the dynamic traceback length (DTBL) algorithm are based on the share factor 2 (called shared-2) architecture so the design of the PMU is the same. The design of the SMU follows the figure 5.9 and the clock frequency is 5MHz to achieve 5Mb/s throughput. The memory size is also the same as shared-2 architecture, $192 \times 32$ bits.

6.5.2 Back-end results and analysis

After the implementation, the front-end results are shown in appendix E.3 with necessary analysis. The power distribution of back-end results are shown in figure 6.13 which also contains the baseline shared-2 architecture with conventional Viterbi Algorithm (called shared-2 architecture directly in the following). The PMU power is similar as the shared-2 architecture because the designs are the same. For the power of the traceback unit, the DTBL consumes 38 uW and 58 uW at 0 dB and 6 dB. Figure 6.14 illustrates the power distribution of minor modules. As the comparison tree power is 10 uW and 26 uW, the buffer control power is around 17 uW and 22 uW respectively. The reason is that higher SNR result in more calculation about the traceback path buffer and prediction.

![Figure 6.13: Comparison of power distribution between the DTBL and shared-2 by back-end synthesis](image)

![Figure 6.14: Power distribution of minor modules between the DTBL and shared-2 by back-end synthesis](image)

Concerning the memory power, figure 6.15 shows the memory access power saving based on shared-2 architectures with different ratio of $E_{wr}/E_{rd}$. In this thesis work, $E_{wr}/E_{rd}=5$, thus the
power saving goes from 8% at 0 dB to 16% at 6 dB. The measured result shows that memory power saving 8.4% and 21% which exactly conforms to the theoretical result. Therefore, at EbN0=0 dB, the DTBL saves 7 uW power of memory reading at the cost of 23 uW extra power of the traceback unit, at EbN=6 dB, the DTBL saves 20 uW power of memory reading at the cost of 43 uW extra power.

Figure 6.15: Memory access saving with DTB algorithm based on shared-2 architecture

Figure 6.16 shows that the extra area is 1800 um² of the traceback unit. Appendix H compares the front-end and back-end results and situation is almost the same as 6.3.3. The difference of power at front-end and back-end simulations is around 15% while the area estimation is almost the same. Detail is not discussed here.

Figure 6.16: Comparison of area distribution of the PE between the DTBL and shared-2 by back-end synthesis

It can be concluded that with the standard cell-based memory, the DTBL cannot save power at such context because of the cost of the comparison tree and buffer control.

### 6.6 Comparison of all implementations

To summarize all the implementations concerning the power and area, table 6.3 show the comparison by the back-end synthesis compared to the baseline architecture. The ACSU and local routing...
and SM memory are the main components of the PMU. As power is the first priority, among the three architectures, fully-parallel implementation is the optimal. Although shared architecture can save up to 15% area, the increment power is up to 37% because of the local routing and SM memory. It can be expected that shared factor 8 architecture consumes more power with less area which does not fit the power requirement.

Comparing the PTB and TF, the PTB is the optimal, as it has the same power with only 4% area overhead, compared to 13% overhead of TF. The main extra power of PTB comes from the memory and local routing and SM memory which are also the source of extra area. The TF eliminates the traceback read (TB) operations, resulting in less power at traceback unit and memory and more area at the PMU especially the local routing and SM memory, around 134%.

The DTBL algorithm does save the power consumption of memory, up to 21% compared to shared-2 architecture and only 5% extra area. However, it consumes up to 267% more power in the traceback unit. It can be concluded that this implementation cannot be applied to save power. Possible solution is that reduce the power of comparison tree which is the main part of overhead. Concerning the PTB, TF and DTBL, as the $E_{wr}/E_{rd}$ ratio is around 5 in the standard cell-based memory, its benefit of power saving is thus not obvious and the possible solution is to apply different memories with lower $E_{wr}$ and $E_{wr}/E_{rd}$ ratio.

<table>
<thead>
<tr>
<th></th>
<th>fully-parallel</th>
<th>shared-2</th>
<th>shared-4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>power</td>
<td>area</td>
<td>power</td>
</tr>
<tr>
<td>PMU Traceback unit</td>
<td>0%</td>
<td>0%</td>
<td>28%</td>
</tr>
<tr>
<td>Mem</td>
<td>0%</td>
<td>0%</td>
<td>-12%</td>
</tr>
<tr>
<td>ACS Local routing+SMs</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
</tr>
<tr>
<td>Overall</td>
<td>0%</td>
<td>0%</td>
<td>10%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>fully-parallel</th>
<th>PTB</th>
<th>TF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>power</td>
<td>area</td>
<td>power</td>
</tr>
<tr>
<td>PMU Traceback unit</td>
<td>0%</td>
<td>0%</td>
<td>8%</td>
</tr>
<tr>
<td>Mem</td>
<td>0%</td>
<td>0%</td>
<td>-3%</td>
</tr>
<tr>
<td>ACS Local routing+SMs</td>
<td>0%</td>
<td>0%</td>
<td>14%</td>
</tr>
<tr>
<td>Overall</td>
<td>0%</td>
<td>0%</td>
<td>9%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>shared-2</th>
<th>DTBL(0 dB)</th>
<th>DTBL(6 dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>power</td>
<td>area</td>
<td>power</td>
</tr>
<tr>
<td>PMU Traceback unit</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
</tr>
<tr>
<td>Mem</td>
<td>0%</td>
<td>0%</td>
<td>146%</td>
</tr>
<tr>
<td>Overall</td>
<td>0%</td>
<td>0%</td>
<td>6%</td>
</tr>
</tbody>
</table>

Table 6.3: Comparison of all the implementations by back-end synthesis

Table 6.4 illustrates the standard and maximum clock frequency as well as memory size of all the back-end implementations. Among all the designs, the **fully parallel architecture** with the conventional Viterbi Algorithms is selected as the optimal design, which consumes least power. Although the **PTB** consumes a little bit more power compared to fully-parallel architecture, it can achieve the maximum throughput (40 Mb/s) among all designs with 40 MHz clock rate. So it is promising to be the optimal low-power design by the voltage and clock rate scaling and it is thus selected as another optimal algorithm among all implemented algorithms. Furthermore, figure 6.17 compares the energy per bit between nominal and maximum clock frequency by back-
end simulations. Obvious, all the energy per bit decrease when frequency increases because the utilization ratio of clock cycle is higher. Among all the measured data, the conventional fully parallel architecture and the PTB are optimal.

Furthermore, to compare the front-end and back-end synthesis, the difference of the power estimation in this thesis work is 10% to 30% while the difference of area estimation is only around 5%. Also, based on the results, the influence of the global routing of different designs to power and area can be omitted.

<table>
<thead>
<tr>
<th></th>
<th>fully-parallel</th>
<th>shared-2</th>
<th>shared-4</th>
<th>PTB</th>
<th>TF</th>
<th>DTBL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Clk (MHz)</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Max clk (MHz)</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>Max throughput (Mlb/s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem size</td>
<td>96*64bits</td>
<td>192*32bits</td>
<td>38*16bits</td>
<td>48*128bits</td>
<td>96*64bits</td>
<td>192*32bits</td>
</tr>
</tbody>
</table>

Table 6.4: Comparison of all the clock frequency and memory size

![Comparison of energy/bit with different clk rate](image)

Figure 6.17: Comparison of energy per bit with maximum clock frequency
Chapter 7

Conclusion and recommendations

7.1 Conclusion

This thesis work presents an investigation to design and implement low-power and small-area IC of Viterbi Decoders for low-throughput applications and 802.11ah standard with TSMC 40nm technology. The idea is to separately investigate different architectures and different algorithms to reduce power and area based on the mathematical analysis and literature study. Simulation results show that the implementation of the fully-parallel architecture with the conventional Viterbi Algorithm, 247 uW and 0.04 mm$^2$, and the pre-traceback (PTB) algorithm, 269 uW and 0.04 mm$^2$, are the optimal designs among all the work. With 1.1V voltage supply, the clock rate is respectively 10 MHz and 5 MHz to both achieve a 5 Mb/s throughput. Compared to the power target (200 to 400 uW) in the section 2.3, two selected optimal implementations achieve the goal, shown in figure 7.1.

![Figure 7.1: Final result](image)

The main contributions of this thesis work are in the following
CHAPTER 7. CONCLUSION AND RECOMMENDATIONS

1. Investigate the related algorithm in the literature to reduce power and area for the application context in this thesis work and quantify a detail target,

2. Build up a simulation chain to evaluate the BER, power and area performance by Matlab and Cadence tools,

3. Propose and implement IC solutions for each components of low-power and small-area designs,

4. Design and implement 3 architectures and 3 algorithms to reduce power and area,

5. Compare and investigate the simulation accuracy of Cadence front-end and back-end synthesis concerning the area and power,

6. Evaluate and analyze the power and area as well as BER performance by Matlab simulation and Cadence tools. Among all the designs, the fully parallel architecture and pre-traceback (PTB) algorithms are the optimal implementation with the least power and acceptable area.

7.2 Recommendations

Further research work for possible improvements is listed as the following

1. Continue investigate the work in [36, 47, 38] about scarce-state-transition (STT) and T-algorithm to reduce the power of the Path Metric Unit (PMU) at the cost of extra area and BER performance loss.

2. Reduce the power of the bottleneck part of implemented designs. For example, the comparison tree is the power bottleneck of the dynamic traceback length (DTBL) algorithm, scarce-state-transition algorithm can be applied to find the best state at each step which consumes less power.

3. Combine frequency and voltage scaling to further reduce the power of optimal designs while satisfying the throughput requirement because the supply voltage of the standard cell-based memory can be reduced down to 0.2V which will lead to a tremendous power saving.
Bibliography


Appendix A

Introduction to Viterbi Algorithm

Consider a Hidden Markov model (HMM) $M$ with $m$ states. Given an observed sequence $X=X_1, \cdots, X_n$, find the most likely sequences of states $s=s_1, \cdots, s_n$ that generated the sequence $X$.

As described in [5], the Viterbi Algorithm (VA) solves the problem as the following. Define $\gamma_j(i)$ as the probability of the most probable sequence of states $s=s_1, \cdots, s_{j-1}$, ending in state $i$ i.e. $s_j=i$ and $\delta_j(i)$ corresponds to the state where the probability is derived. $t_k(i)$ represents the transition probability from state $k$ to state $i$ and the $e_i(X_j)$ defines the emission probability of $X_j$. Therefore, the algorithm can be divided into two phases. In the first phase, at every step, for each step, $\gamma_j(i)$ is calculated as the maximum probability of the state path ending in state $j$, derived from $\gamma_{j-1}(i)$ and the corresponding state $i$ is stored. In the second phase, back-tracing phase, the sequence of states in the most probable state path is reconstructed by starting from the state $s_n$ with the maximum probability.

```
Viterbi(M, X)
1: for i = 1 to m do {Initialization}
2: $\gamma_0(i) = \pi_j$
3: $\delta_0(i) = 0$
4: end for
5: for j = 1 to n do {First phase}
6: for i = 1 to m do
7: $\gamma_j(i) = \max_{k=1}^{m} \gamma_{j-1}(k) t_k(i) e_i(X_j) \{\text{Recurrence}\}$
8: $\delta_j(i) = \arg \max_{k=1}^{m} \gamma_{j-1}(k) t_k(i) e_i(X_j)$
9: end for
10: end for
11: $s_n = \arg \max_{k=1}^{m} \gamma_n(k)$
12: for $i = n - 1$ to 1 do {Second phase (back-tracing)}
13: $s_i = \delta_{i+1}(s_{i+1})$
14: end for
15: return $s$
```

Figure A.1: Viterbi Algorithm [5]
Appendix B

Conventional Viterbi Algorithm

Algorithm 5 Conventional VD algorithm

Require: rec_signal, trunc_length

1: procedure VD
   2: \[ \text{block_length} \leftarrow \text{LENGTH}(\text{rec_signal})/N \] \( \triangleright \) N is 2, NO. of output
   3: \[ \text{path_memory} \leftarrow \text{int}[\text{NO \_ST} \times \text{trunc\_length}] \] \( \triangleright \) NO\_ST is 64, NO. of states
   4: \[ l \leftarrow 0 \]
   5: \[ \text{sum\_metric}[] \leftarrow 0 \]
   6: for \( l + 1 < \text{block\_length} \) do \( \triangleright \) evolve all trellis, PMU
   7: \[ \text{rec\_bit}0 \leftarrow \text{rec\_signal}[l*N] \]
   8: \[ \text{rec\_bit}1 \leftarrow \text{rec\_signal}[l*N+1] \]
   9: \[ s \leftarrow 0 \]
   10: \[ \text{temp\_sum\_metric}[] \leftarrow 0 \]
   11: for \( s++ < \text{NO\_ST} \) do
      12: \[ \text{branch\_state}0 = \text{Prev\_state}(s, 0) \]
      13: \[ \text{branch\_state}1 = \text{Prev\_state}(s, 1) \]
      14: \[ \text{zero\_metric, one\_metric} = \text{BMU}(s, \text{rec\_bit}0, \text{rec\_bit}1, \text{branch\_state}0, \text{branch\_state}1) \]
      15: \[ \text{temp\_zero\_metric} \leftarrow \text{sum\_metric}[s] + \text{zero\_metric} \]
      16: \[ \text{temp\_one\_metric} \leftarrow \text{sum\_metric}[s] + \text{zero\_metric} \]
      17: if \( \text{temp\_zero\_metric} < \text{temp\_one\_metric} \) then
         18: \[ \text{temp\_sum\_metric}[s] \leftarrow \text{temp\_zero\_metric} \]
         19: \[ \text{path\_memory}[s*\text{block\_length}] \leftarrow 0 \]
      20: else
         21: \[ \text{temp\_sum\_metric}[s] \leftarrow \text{temp\_one\_metric} \]
         22: \[ \text{path\_memory}[s*\text{block\_length}] \leftarrow 1 \]
      23: \[ \text{sum\_metric}[] \leftarrow \text{temp\_sum\_metric}[] \]
   24: \[ \text{min\_metric\_state} = \text{MIN}(\text{sum\_metric}) \] \( \triangleright \) traceback from min metric, SMU
   25: \[ t \leftarrow l - 1 \]
   26: for \( t-- > 1 - \text{trunc\_length} \) do
      27: \[ \text{min\_metric\_state} = \text{Prev\_state}(\text{min\_metric\_state}, \text{path\_memory}\slash \text{min\_metric\_state}) \]
      28: \[ \text{dec\_bits}[t] = \text{get\_input}(\text{min\_metric\_state}) \] \( \triangleright \) decode last state
   30: return \text{dec\_bits}
Appendix C

Flow chart of dynamic truncation length and path prediction algorithm

This appendix illustrates the flow chart of dynamic truncation length and path prediction algorithm.

Figure C.1: Illustration of (a) dynamic truncation length (b) path prediction algorithm [6]
Appendix D

Routing of fully parallel, shared factor 2 and 4

This appendix shows the global and local routing configuration of three different architectures. Table D.1 shows the scheduler in the local routing of PMU for shared factor 2 and 4. The index 0, 1 and shift control wire corresponds to the interface in figure 4.9 (a).

<table>
<thead>
<tr>
<th>Slice</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Index 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Reading shift</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table D.1: Scheduler in the local routing of PMU for (a) shared factor 2 and (b) shared factor 4

Table D.2 and D.3 illustrate the global routing table for fully-parallel, shared factor 2 and 4 architectures. The nomination of \text{pe}_x.in/out.y indicates the connection of input/output port \( y \) of processing element(PE) \( x \).
### APPENDIX D. ROUTING OF FULLY PARALLEL, SHARED FACTOR 2 AND 4

#### Table D.2: Global routing table of fully parallel architecture

<table>
<thead>
<tr>
<th>Connection</th>
<th>Connection</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe0_in_0</td>
<td>pe0_out_0</td>
<td>pe8_in_0</td>
</tr>
<tr>
<td>pe0_in_1</td>
<td>pe1_out_0</td>
<td>pe8_in_1</td>
</tr>
<tr>
<td>pe1_in_0</td>
<td>pe2_out_0</td>
<td>pe8_in_0</td>
</tr>
<tr>
<td>pe1_in_1</td>
<td>pe3_out_0</td>
<td>pe8_in_1</td>
</tr>
<tr>
<td>pe2_in_0</td>
<td>pe4_out_0</td>
<td>pe10_in_0</td>
</tr>
<tr>
<td>pe2_in_1</td>
<td>pe5_out_0</td>
<td>pe10_in_1</td>
</tr>
<tr>
<td>pe3_in_0</td>
<td>pe6_out_0</td>
<td>pe11_in_0</td>
</tr>
<tr>
<td>pe3_in_1</td>
<td>pe7_out_0</td>
<td>pe11_in_1</td>
</tr>
<tr>
<td>pe4_in_0</td>
<td>pe8_out_0</td>
<td>pe12_in_0</td>
</tr>
<tr>
<td>pe4_in_1</td>
<td>pe9_out_0</td>
<td>pe12_in_1</td>
</tr>
<tr>
<td>pe5_in_0</td>
<td>pe10_out_0</td>
<td>pe13_in_0</td>
</tr>
<tr>
<td>pe5_in_1</td>
<td>pe11_out_0</td>
<td>pe13_in_1</td>
</tr>
</tbody>
</table>

#### Table D.3: Global routing table of (a) shared factor 2 and (b) shared factor 4

<table>
<thead>
<tr>
<th>Connection</th>
<th>Connection</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe0_in_sm0</td>
<td>pe0_out_sm0</td>
<td>pe8_in_sm0</td>
</tr>
<tr>
<td>pe0_in_sm1</td>
<td>pe1_out_sm0</td>
<td>pe8_in_sm1</td>
</tr>
<tr>
<td>pe1_in_sm0</td>
<td>pe2_out_sm0</td>
<td>pe9_in_sm0</td>
</tr>
<tr>
<td>pe1_in_sm1</td>
<td>pe3_out_sm0</td>
<td>pe9_in_sm1</td>
</tr>
<tr>
<td>pe2_in_sm0</td>
<td>pe4_out_sm0</td>
<td>pe10_in_sm0</td>
</tr>
<tr>
<td>pe2_in_sm1</td>
<td>pe5_out_sm0</td>
<td>pe10_in_sm1</td>
</tr>
<tr>
<td>pe3_in_sm0</td>
<td>pe6_out_sm0</td>
<td>pe11_in_sm0</td>
</tr>
<tr>
<td>pe3_in_sm1</td>
<td>pe7_out_sm0</td>
<td>pe11_in_sm1</td>
</tr>
<tr>
<td>pe4_in_sm0</td>
<td>pe8_out_sm0</td>
<td>pe12_in_sm0</td>
</tr>
<tr>
<td>pe4_in_sm1</td>
<td>pe9_out_sm0</td>
<td>pe12_in_sm1</td>
</tr>
<tr>
<td>pe5_in_sm0</td>
<td>pe10_out_sm0</td>
<td>pe13_in_sm0</td>
</tr>
<tr>
<td>pe5_in_sm1</td>
<td>pe11_out_sm0</td>
<td>pe13_in_sm1</td>
</tr>
<tr>
<td>pe6_in_sm0</td>
<td>pe12_out_sm0</td>
<td>pe14_in_sm0</td>
</tr>
<tr>
<td>pe6_in_sm1</td>
<td>pe13_out_sm0</td>
<td>pe14_in_sm1</td>
</tr>
<tr>
<td>pe7_in_sm0</td>
<td>pe14_out_sm0</td>
<td>pe15_in_sm0</td>
</tr>
<tr>
<td>pe7_in_sm1</td>
<td>pe15_out_sm0</td>
<td>pe15_in_sm1</td>
</tr>
</tbody>
</table>

Table D.2: Global routing table of fully parallel architecture

Table D.3: Global routing table of (a) shared factor 2 and (b) shared factor 4
Appendix E

Analysis of the front-end simulation results

This appendix compares and analyzes the front-end simulation results of all the implementations.

E.1 Front-end results and analysis of three architectures

Figure E.1 shows the comparison of the power distribution between those three architectures. The total power of the fully-parallel, shared-2 and shared-4 architecture are 156uW, 230uW and 413uW. The PMU and memory consumes around half of the total power, dominating the overall consumption which conforms to the power distribution in several literature. Concerning the PMU, figure E.2 shows the detail power distribution. The SM memory in the fully parallel architecture accounts for most of the power of the sum power of the local routing and SM memory. So the power of the local routing of shared-2 and shared-4 architectures can be calculated by subtracting it and results are 31uW and 71uW. The ratio of the result, 0.43, closed to the theoretical value 0.5 of equation 4.8 indicating that the power of local routing linearly scales with the shared factor.

Concerning the power of the global routing, equation 4.8 shows it is inverse proportional to the shared factor $\delta$. The power of the global routing $P_{\text{global routing}}$ can be calculated by subtracting the power of the PE $P_{\text{PE}}$, control units and other sub-modules $P_{\text{minor modules}}$ from the PMU $P_{\text{PMU}}$, shown in equation E.1. Figure E.3 shows calculation results as well as the power of the control unit and comparison tree. As the power of the global routing is closed and tiny for three architectures, the global routing power can be omitted in such situation. Furthermore, the power of the control unit increases with higher shared factor, as the SM memory needs more complex
APPENDIX E. ANALYSIS OF THE FRONT-END SIMULATION RESULTS

Figure E.2: Comparison of power distribution of the PE between three architectures by front-end synthesis

The power of traceback unit are nearly the same after excluding the power of comparison unit. It can also be observed that by sequentialization, the power (also energy per bit) of the ACSU increases slightly.

\[ P_{\text{global routing}} = P_{\text{PMU}} - P_{\text{PE}} - P_{\text{minor modules}} \]  
(E.1)

Figure E.3: Comparison of power distribution of minor modules between three architectures by front-end synthesis

The area distribution of three architectures are shown in figure E.4. The total area of fully-parallel, shared-2 and shared-4 architecture are 43600 um^2, 37700 um^2 and 37500 um^2. Generally, the memory area dominates the overall size, as it is self-defined register file without much optimization. Because the totally number of bits of three memories are the same, the area are thus quite similar. Furthermore, because of the sequentialization, the size of the PMU is reduced for higher shared factor as expected.

Figure E.4: Comparison of area distribution between three architectures by front-end synthesis

Figure E.5 illustrates the area distribution of the PE. The linear relation between the total ACSU size and shared factor is clearly shown, which is roughly 1:1.9:3.7. The sum area of the
local routing and SM memory in the fully parallel architecture are dominated by the SM memory. As the total memory of the SM are the same for different architectures, the local routing area of shared architectures can be calculated by subtracting the area of the SM memory. The calculation results of shared 2 and 4 are similar and this result conforms to the equation 4.7 indicating that the area of total local routing is constant for shared architecture.

![Figure E.5: Comparison of area distribution of the PE between three architectures by front-end synthesis](image)

For the area of global routing $A_{\text{global routing}}$, it can be obtained by subtracting the area of the PE $A_{\text{PE}}$, control units and other sub-modules $A_{\text{minor modules}}$ from the PMU $A_{\text{PMU}}$, shown in equation E.2. The results are shown in figure E.6 with the comparison tree and control unit. The calculation results of the global routing is trivial, so its area can be omitted in front-end synthesis. Concerning the area of the comparison tree, the ratio of the fully parallel, shared factor 2 and 4 is about 4:2:1. The reason is that the comparison tree computes the a column of the SM of one cycle, so its size is linear to the number of PE. So the theoretical ratio of the comparison tree area is 4:2:1 which explains the result.

$$A_{\text{global routing}} = A_{\text{PMU}} - A_{\text{PE}} - A_{\text{minor modules}}$$  \hspace{1cm} (E.2)

![Figure E.6: Comparison of area distribution of minor modules between three architectures by front-end synthesis](image)

Therefore, based on the simulation results of front-end synthesis, concerning the PMU, it can be concluded that the shared architecture has less area at the cost of more power. The extra power comes from the local routing which is linear to the shared factor. Also, both the power and area of the global routing can be omitted. However, the benefit of area is less significant when the shared factor increases because of the existence of the SM memory and local routing area.
E.2 Front-end results and analysis of two standard algorithms

Figure E.7 illustrates the comparison of power distribution between 3 standard algorithms as well as a conventional VD with fully parallel architecture (called fully parallel in the following). The total power of the PTB, sliding TF and TF are 145 uW, 200 uW and 179 uW and their PMU power is 11%, 63% and 38% more than fully parallel architecture. Figure E.8 shows the detail power distribution of the PE, compared with fully parallel.

Concerning the ACSU power, PTB, TF and sliding TF has 6%, 23% and 30% more compared to fully parallel, which are the overhead in the ACSU. Concerning the local routing and SM memory power, which is mainly SM memory, the three algorithms has 26%, 74% and 157% more. The reason is that the PTB only needs 2*2 more bits memory of each the PE for ∆=2 while the TF and sliding TF need 2*6 and 2*2*6 more bits (each traceback state has 6 bits). So theoretically based on the power modelling of equation 5.1, the extra power of the SM memory in the TF and sliding TF shall be 3 and 6 times more than the PTB, which conforms to the practical value 2.8 and 6.

Furthermore, by equation E.1, the power of the global routing can be calculated and figure E.9 shows the results with the comparison tree and control unit. The results are all around 4 uw, indicating that the extra power can be omitted for the three algorithms. As the traceback frequency of sliding TF algorithm is twice the other algorithms, its comparison tree consumes more power to generate the best state to start the traceback.

Concerning the power of the memory, the results are 51, 32, 44 and 41 of the fully parallel, PTB, TF and sliding TF respectively. Equation 5.5 explains why PTB has less memory power than fully parallel, as it has less overhead. In addition, according to equation 5.3, TF-type algorithms
APPENDIX E. ANALYSIS OF THE FRONT-END SIMULATION RESULTS

reduce one reading operation of each decoded bit, therefore it has less power consumption. As sliding TF only needs $2^D$ memory compared to TF, it consumes less power, 41 uW.

The area distribution of three algorithms compared to the fully parallel architecture are shown in figure E.10. The total memory size is 45400 um$^2$, 47600 um$^2$ and 42800 um$^2$ of the PTB, sliding TF and TF respectively. The memory area of the PTB and TF are the same as they have the same number of bits while sliding TF has around $\frac{2}{3}$ proportion memory area.

Concerning the area of the PMU, figure E.11 illustrates the area distribution of the PE. Compared to fully parallel, the ACSU of the PTB, TF and sliding TF has 1%, 9% and 17% more area, which is the overhead area. The SM memory has 38%, 129% and 225% more area, so TF and sliding TF has 3.3 and 5.9 times than the PTB, conforming to the equation 5.1, as the extra area scales linearly with the memory size. Also, figure E.12 shows the global routing size, which is 200, 600 and 600 um$^2$ of the PTB, TF and sliding TF. As the TF-type needs wider interconnections, it has more area of global routing. However, as it only accounts for 1% of the total area, the influence can be omitted.

Therefore, based on the front-end simulation results, all three algorithms have less power consumption of the memory at the costs of more area and power in PMUs. Similar to the simulation of different architectures in the previous section, the power and area of global routing can also be omitted. In addition, the PTB is the optimal design concerning the power with a slight more area compared to fully parallel design. Although the sliding TF has smallest area, the power is the highest, more than 38% compared to the PTB. Thus it is not necessary to continue the back-end simulation for the sliding TF.
APPENDIX E. ANALYSIS OF THE FRONT-END SIMULATION RESULTS

Figure E.11: Comparison of area distribution of the PE between 2 standard algorithms by front-end synthesis

Figure E.12: Area distribution comparison of minor modules between 2 standard algorithms by front-end synthesis

E.3 Front-end results and analysis of the adaptive algorithm

As the DTBL is an adaptive algorithm, its power depends on the channel signal-to-noise ratio (SNR). So the simulation is done under two channel EbN0, 0 dB and 6 dB. The power distribution is shown in the figure E.13. As expected, the power of the PMU of the DTBL at 0dB and 6dB is nearly the same as conventional shared-2 architecture. Concerning the memory consumption, as the memory in the front-end simulation is self-defined register files, the power saving cannot work in such situation.

As the extra control unit is added in the traceback unit, its power is dependent on the SNR, which can be verified by the measured power. Set traceback unit power of shared-2 as the baseline, the DTBL has 130% and 307% more power at 0 dB and 6 dB respectively. The power of traceback unit consists of comparison tree and other control parts and the DTBL has more control parts to manage the buffer. Figure E.14 shows the power of minor modules. The power of the comparison tree is 5 uW, 14 uW and 35 uW of the shared-2, DTBL at 0 dB and 6 dB, therefore excluding those power, the common control part is 9 uW and the buffer control unit is around 18 uW and 22 uW of the DTBL at 0 dB (low SNR) and 6 dB (high SNR). According to the figure 5.10, when EbN0=6dB, the reduced memory reading percentage goes to nearly 100%, which indicates that its buffer control power is maximum among all the cases. Therefore, the maximum extra power of the DTBL is 43 uW consisting of comparison tree, 35 uW and buffer control, 8 uW and the power saving benefit will be positive when the saving memory reading power is larger than 43 uW. Back-end results will testify the benefit.
APPENDIX E. ANALYSIS OF THE FRONT-END SIMULATION RESULTS

Figure E.13: Comparison of power distribution of the PE between the DTBL and shared-2 by front-end synthesis

Figure E.14: Power distribution comparison of minor modules between the DTBL and shared-2 by front-end synthesis

Figure E.15 shows the area distribution. As expected, the extra area 3300 $\mu$m$^2$ of buffer control unit and buffer comes from the traceback unit.

Figure E.15: Comparison of area distribution between the DTBL and shared-2 by front-end synthesis
Appendix F

Comparison of results by the front-end and back-end synthesis of different architectures

This appendix compares the power and area distribution of fully-parallel, shared factor 2 and 4 architectures from the front-end and back-end simulations.

![Comparison of power distribution with front-end and back-end synthesis](image)[Figure F.1: Comparison of power distribution with front-end and back-end synthesis]
APPENDIX F. COMPARISON OF RESULTS BY THE FRONT-END AND BACK-END SYNTHESIS OF DIFFERENT ARCHITECTURES

Figure F.2: Comparison of power distribution of PEs with front-end and back-end synthesis

Figure F.3: Comparison of area distribution with front-end and back-end synthesis

Figure F.4: Comparison of area distribution of PEs with front-end and back-end synthesis
Appendix G

Comparison of results by the front-end and back-end synthesis of two standard algorithms

This appendix compares the power and area distribution of TF and PTB algorithms.

Figure G.1: Comparison of power distribution with front-end and back-end synthesis

Figure G.2: Comparison of power distribution of PEs with front-end and back-end synthesis
APPENDIX G. COMPARISON OF RESULTS BY THE FRONT-END AND BACK-END SYNTHESIS OF TWO STANDARD ALGORITHMS

Figure G.3: Comparison of area distribution with front-end and back-end synthesis

![Comparison of area distribution(100 um^2)](image)

Figure G.4: Comparison of area distribution of PEs with front-end and back-end synthesis

![Comparison of PE area distribution(100 um^2)](image)
Appendix H

Comparison of results by the front-end and back-end synthesis of adaptive algorithms

This appendix compare the power and area distribution between DTBL and shared-2 architecture by front-end and back-end synthesis.

Figure H.1: Comparison of area distribution between DTBL and shared-2 by front-end and back-end synthesis

Figure H.2: Comparison of area distribution between DTBL and shared-2 by front-end and back-end synthesis