

國立交通大學

電信工程學系

博士論文

用於高傳輸率渦輪碼之交錯器設計

Inter-Block Permutation Interleaver
Design for High Throughput Turbo Codes

研究生：鄭延修

指導教授：蘇育德

中華民國九十六年十月

用於高傳輸率渦輪碼之交錯器設計

Inter-Block Permutation Interleaver Design for High Throughput Turbo Codes

研究生：鄭延修

Student: Yan-Xiu Zheng

指導教授：蘇育德

Advisor: Yu Ted Su

國立交通大學

電信工程學系



A Dissertation

Submitted to Institute of Communication Engineering
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in
Communication Engineering
Hsinchu, Taiwan

2007 年 10 月

用於高傳輸率渦輪碼之 交錯器設計

研究生：鄭延修

指導教授：蘇育德

國立交通大學電信工程研究學系



渦輪碼以其優異的性能獲得通訊界的青睞，但為達到較佳的效能，渦輪碼需要進行較多次的遞回運算並搭配較長的交錯器，也因此造成較長的解碼延遲。因此我們提出一種系統化的交錯器設計流程去解決解碼延遲與解碼效能之間的兩難。我們的設計考量代數特性與硬體限制。從代數特性的觀點來看，此設計利用較短交錯器去建構較長的交錯器以保持較佳的碼距特性，我們所提出的交錯器還可額外滿足高解碼率與平行解碼的硬體限制，其中包括避免記憶體衝突，有限的網路複雜度以及較簡單的記憶體控制線路。所提出的交錯器有較簡單的代數形式，也允許較有彈性的平行度並較容易適應各種不同的交錯器長度。就算並非應用於平行解碼，在相同的交錯器長度下，本設計亦提供較佳的碼距特性。

我們將區塊間交錯重排交錯器分成方塊式與串接式，針對兩者我們為重量為二的輸入序列推導碼重邊界，此推導亦給了我們設計區塊間重排交錯器的參考依據，我們另外證明為了達成好的碼重性質並避免記憶體競爭的代數性質。針對方塊式區塊間重排交錯器，我們提出記憶體配置函數去描述與提供具彈性的解碼器平行度與支援高基數後驗機率解碼器。網路導向設計概念解決了平行解碼架構下網路複雜度的問題。我們亦提出有效率的交錯器設計流程去做大範圍的交錯器設

計。我們亦用一個超大型積體電路設計去展現本設計確可同時兼顧高速與低複雜並提供較好的錯誤率。

串接式區塊間交錯排列交錯器是針對導管型解碼架構而設計，此架構非常適合高解碼率應用但須付出複雜度的代價。為了得到複雜度與解碼率的最佳折衷點，我們提出了一個動態結構。我們處理其所面對的解碼排程與記憶體控制問題，我們亦介紹了一種新穎的結合冗餘檢測碼與正負號檢測的終止機制，在一個較佳的解碼排程，記憶體控管與終止機制下，我們可以減少硬體複雜度並在較短的平均解碼延遲下達成更好的錯誤率。

為了描述各種遞回式解碼排程與分析其特性，我們發展了一個圖形工具稱為多階層元素圖，基於這個新工具，我們推導了可提供較佳錯誤率與使用較少記憶體的新解碼排程，基於完整性，我們亦展出非規則性打洞樣式去提供更好的錯誤率。



Inter-Block Permutation Interleaver Design for High Throughput Turbo Code

Student: Yan-Xiu Zheng

Advisor: Yu T. Su

Institute of Communications Engineering

National Chiao Tung University

Abstract

With all its remarkable performance, the classic turbo code (TC) suffers from prolonged latency due to the relatively large iteration number and the lengthy interleaving delay required to ensure the desired error rate performance. We present a systematic approach that solves the dilemma between decoding latency and error rate performance. Our approach takes both algebraic and hardware constraints into account. From the algebraic point of view, we try to build large interleavers out of small interleavers. The structure of classic TC implies that we are constructing long classic TCs from short classic TCs in the spirit of R. M. Tanner. However, we go far beyond just presenting a new class of interleavers for classic TCs. The proposed inter-block permutation (IBP) interleavers meet all the implementation requirements for the parallel turbo decoding such as memory contention-free, low routing complexity and simple memory addressing circuitry. The IBP interleaver has simple algebraic form; it also allows flexible degrees of parallelism and is easily adaptable to variable interleaving lengths. Even without high throughput demand, the IBP design is capable of improving the distance property with increased equivalent interleaving length but not the decoding delay except for the initial blocks.

We classify the IBP interleavers into block and stream ones. For both classes we derive codeword weight bounds for weight-2 input sequences that give us important

guidelines for designing good IBP interleavers. We prove that the algebraic properties required to guarantee good distance properties satisfying the memory contention-free requirement as well. For block IBP interleavers, we propose memory mapping functions for flexible parallelism degrees and high-radix decoding units. A network-oriented design concept is introduced to reduce the routing complexity in the parallel decoding architectures. We suggest efficient interleaver design flows that offer a wide range of choices in the interleaving length. A VLSI design example is given to demonstrate that the proposed interleavers do yield high throughput/low complexity architecture and, at the same time, give excellent error rate performance.

The stream-oriented IBP interleavers are designed for the pipeline decoding architecture which is suitable for high throughput applications but has to pay the price of large hardware complexity. In order to achieve optimal trade-off between hardware complexity and decoding throughput, a dynamic decoder architecture is proposed. We address the issues of decoding schedule and memory management and introduce the novel stopping mechanisms that incorporate both CRC code and sign check. With a proper decoding schedule, memory manager and early-stopping rule, we are able to reduce the hardware complexity and achieve improved error rate performance with a shorter average latency.

In order to describe various parallel and pipeline iterative decoding schedules and analyze their behaviors, we develop a graphic tool called multi-stage factor graphs. Based on this new tool we derive a new decoding schedule which gives compatible error rate performance with less memory storage. For completeness, we show some irregular puncturing patterns that yield good error rate performance.

論 文 致 謝

感謝指導教授蘇育德這幾年的指導與對論上的繕改，讓這本論文不至遺漏。也感謝口試委員趙啟超、林茂昭、韓永祥、楊谷章、蘇賜麟、鍾嘉德、王忠炫、陳伯寧教授對論文的建議，讓本論文更為完整。

在此亦對由李鎮宜，張錫嘉教授與其所領導的 OCEAN 團隊至上感謝，能夠跟貴團隊合作打造渦輪碼晶片的晶片，讓我的設計得以實現，深感榮幸。

最後亦感謝工研院由蕭昌龍領軍的 TW4G 團隊，在其中與貴團隊的合作讓本論文內容得以更貼近工業界的需求。

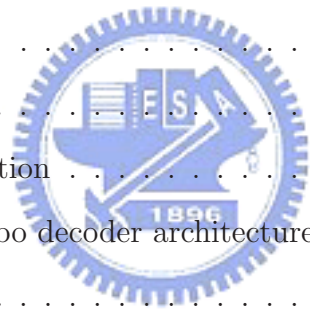


Contents

Chinese Abstract	i
English Abstract	iii
Acknowledgements	v
Contents	vi
List of Figures	xii
List of Tables	xix
Glossary	xx
1 Introduction	1
1.1 Turbo decoding	1
1.2 Performance analysis and graph codes	3
1.3 Low latency/high performance interleavers	4
1.4 Statement of purpose: main contributions	7
1.5 Existing interleavers as instances of IBP interleaver	9
1.6 Overview of chapters	10
2 Fundamentals	13
2.1 Digital communication system	14
2.1.1 Discrete memoryless channel model	15



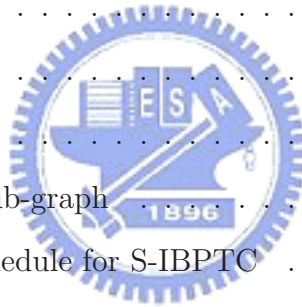
2.1.2	Mapper and de-mapper	16
2.1.3	Error control system	17
2.2	Convolutional code	17
2.2.1	Mathematical notations	17
2.2.2	Encoder	18
2.2.3	State space, state diagram and trellis representation	20
2.2.4	Termination	22
2.2.5	Soft output decoding algorithm for convolutional code	24
2.3	Turbo code	29
2.3.1	Encoder	30
2.3.2	Decoder	31
2.4	Factor graph	33
2.5	Convergence analysis	35
2.5.1	EXIT chart	35
2.5.2	Density evolution	37
2.6	High throughput turbo decoder architecture	37
2.7	Notations	38
2.7.1	Definitions	38
3	Inter-block permutation interleaver	40
3.1	Inter-block permutation turbo code	41
3.2	Inter-block permutation interleaver	42
3.3	IBP properties	45
3.3.1	First property: invariant permutation	46
3.3.2	Second property: periodic permutation	47
3.4	Constraints on the intra-block permutations	50
3.4.1	TP-IBPTC	51
3.4.2	TB-IBPTC	53



3.4.3	C-IBPTC	54
3.5	TB-IBPTC bounds of codeword weights for weight-2 input sequences . .	55
3.5.1	The achievable weight-2 lower bound	56
3.5.2	Analytical results	60
4	Block-oriented inter-block permutation interleaver	63
4.1	The parallel turbo decoder architecture and memory contention	64
4.2	Block-oriented IBPTC	66
4.2.1	B-IBP interleaver	67
4.2.2	Parallelization method in the B-IBP manner	71
4.2.3	Parallelization method in the reversed B-IBP manner	75
4.2.4	Generalized maximal contention-free and intra-block permutation	77
4.2.5	High-radix APP decoder and intra-block permutation	80
4.3	Network-oriented interleaver design	82
4.3.1	Network-oriented B-IBP design	83
4.3.2	Butterfly network	83
4.3.3	Barrel shifter network	88
4.4	B-IBP interleaver supports variable information length	89
4.4.1	Shortening and puncturing	90
4.4.2	Pruning	91
4.4.3	Comparison between shortening and pruning	91
4.5	An interleaver design	94
4.5.1	Interleaver description	94
4.5.2	Comparison to 3GPP LTE QPP	95
4.6	Implementation	96
4.7	Simulation results	98
4.7.1	The interleaver design	98
4.7.2	Shortening and puncturing	100

4.7.3	Separate and continuous encoding	101
5	Stream-oriented inter-block permutation interleaver	106
5.1	Stream-oriented IBP interleaver and the associated encoding storage . . .	107
5.2	Stream-oriented IBPTC encoding and the associated storage	108
5.3	Pipeline decoder and the associated message-passing on the factor graph	110
5.4	Bound and constraints modification for S-IBP interleaver	111
5.5	Codeword weight upper-bounds of stream-oriented IBPTC	114
5.5.1	The upper-bound for weight-2 input sequences	116
5.5.2	The upper-bound for weight-4 input sequences	118
5.5.3	Interleaving gain comparison	121
5.6	Stream-oriented IBP	121
5.7	Modified semi-random interleaver	121
5.8	Simulation Results	123
5.8.1	Covariance and convergence behavior	124
5.8.2	Error probability performance	126
6	Dynamic IBPTC decoder and stopping criteria	134
6.1	IBP turbo coding system with stopping mechanism	135
6.1.1	System model	135
6.1.2	Iterative decoder with variable termination time	137
6.1.3	Graphical representation of an IBPTC and CRC codes	141
6.2	Dynamic decoder and the associated issues	141
6.2.1	Dynamic decoder	142
6.2.2	Decoding delay	143
6.2.3	Memory contention and decoding schedule for multiple ADUs . .	146
6.2.4	Memory management	148
6.3	Multiple-round stopping tests	152

6.3.1	A general algorithm	153
6.3.2	T1.m: the m -round CRCST	154
6.3.3	T2.m: the m -round SCST	155
6.3.4	T3.m: the m -round hybrid stopping test (MR-HST)	156
6.3.5	Genie stopping test	156
6.4	Simulation results	157
7	Multi-stage factor graph	163
7.1	Multi-stage factor graph	164
7.1.1	LDPC code	165
7.1.2	S-IBPTC	169
7.2	Multi-stage factor sub-graph	172
7.2.1	LDPC code	173
7.2.2	S-IBPTC	174
7.2.3	Discussion	175
7.3	Causal multi-stage sub-graph	176
7.4	A memory-saving schedule for S-IBPTC	179
7.5	Simulation results	181
8	Conclusions	183
A	Proof of Lemma 3.6	185
B	Proof of Lemma 3.7	187
C	Proof of Theorem 3.3	188
D	Proof of Theorem 5.10	190
E	Puncturing Patterns	197



Bibliography

198

About the Author

217



List of Figures

1.1	An inherent IBP structure can be found in most practical interleavers. . .	10
2.1	(a) The block diagram of a generic digital communication system; (b) the block diagram of a simplified channel model; (c) the block diagram of a discrete memoryless channel model.	14
2.2	(a) The controller canonical form; (b) The observer canonical form. . . .	18
2.3	(a) The encoder associated with $G(D) = (D + D^2, 1 + D + D^2)$; (b) state diagram; (c) trellis segment.	21
2.4	(a) The block diagram of turbo code encoder; (b) the block diagram of turbo code decoder.	30
2.5	An example of a turbo code factor graph.	34
2.6	The parallel turbo decoder architecture.	38
2.7	a) The block diagram of a decoding module for one iteration; (b) The block diagram of the pipeline decoder.	39
3.1	The block diagram of inter-block permutation turbo code encoder.	41
3.2	(a) Inter-block permutation interleaver; (b) Reversed inter-block permutation interleaver (c) Sandwich inter-block permutation interleaver. . . .	43
3.3	Partition of equivalence classes; $L = 66, T_c = 9$	56
3.4	Set mapping; $N_1 = 3, N_2 = 6$ and $n = 8$	59
3.5	The weight 2 lower bound for the Scrambling function $\frac{1+D^2}{1+D+D^2}$	61
3.6	The weight 2 lower bound for the Scrambling function $\frac{1+D+D^3}{1+D^2+D^3}$	62

3.7	The weight 2 lower bound for the Scrambling function $\frac{1+D^2+D^3+D^4}{1+D+D^4}$	62
4.1	(a) The block diagram of the parallel turbo decoder architecture with parallelism degree 4; (b) memory contention; (c) memory contention-free.	65
4.2	The block diagram of a block-oriented IBPTC encoder.	67
4.3	An example of block-oriented IBP interleaving.	68
4.4	(a) The memory mapping for a B-IBP interleaver composed of 6 blocks to support parallelism degree 6; (b) the merged memory mapping for a B-IBP interleaver composed of 6 blocks to support parallelism degree 3.	74
4.5	The reversed memory mapping for the B-IBP interleaver composed of 7 blocks to support parallelism degree 3.	76
4.6	The block diagram of the asymmetric parallel turbo decoder architecture with 4 memory banks and 3 APP decoders.	76
4.7	(a) Two connected trellis segments referring to Fig. 2.3 (c); (b) The merged trellis segment for the radix-4 APP decoder.	81
4.8	An example of butterfly network-oriented turbo code decoder architecture with parallelism degree 8.	84
4.9	Multiple streams decoding.	86
4.10	An example of barrier shifter network-oriented turbo code decoder architecture with parallelism degree 8.	89
4.11	Weight-2 and 4 error events for a turbo code.	92
4.12	Influence of the weight-2 error events for both shortening and pruning strategy.	93
4.13	Frame error rate comparison for our implementation.	97
4.14	Chip photo [112].	98
4.15	The B-IBP interleaver vs. 3GPP Rel'6 interleaver with the length ranging from 40 to 1000 bits.	100

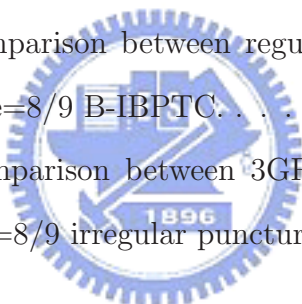
4.16	The B-IBP interleaver vs. 3GPP Rel'6 interleaver with the length ranging from 1000 to 6144 bits.	101
4.17	The B-IBP interleaver vs. 3GPP LTE QPP interleaver with the length ranging from 40 to 1000 bits.	102
4.18	The B-IBP interleaver vs. 3GPP LTE QPP interleaver with the length ranging from 1000 to 6144 bits.	103
4.19	The shortening and puncturing effect for the B-IBP interleaver with the length ranging from 40 to 1000 bits.	103
4.20	The shortening and puncturing effect for the B-IBP interleaver with the length ranging from 1000 to 6144 bits.	104
4.21	The comparison between the separate and continuous encoding for code rate $1/2$	104
4.22	The comparison between the separate and continuous encoding for code rate $3/4$	105
5.1	(a) Conventional inter-block permutation ($S = 1$); (b) Storage saving inter-block permutation ($S = 1$).	109
5.2	(a) The block diagram of an S-IBPTC decoding module for one iteration; (b) The block diagram of the S-IBPTC pipeline decoder.	111
5.3	The time diagram of the pipeline decoder with 4 APP decoders or $I_{\max} = 2$.	112
5.4	A factor graph representation for an S-IBPTC encoded system with the S-IBP span 1.	113
5.5	Partition of equivalence classes into subsets; $L = 68, \Lambda = 27, T_c = T_s = 3$.	117
5.6	Pre- and post-interleaving nonzero coordinate distributions of weight-4 input sequences that result in low-weight S-IBPTC codewords.	119
5.7	Covariance between a priori information input and extrinsic information output.	125

5.8	Exit chart performance of the S-IBPTC and the classic TC at different E_b/N_0 's.	126
5.9	SNR evolution chart behavior of the S-IBPTC and the classic TC at different E_b/N_0 's.	127
5.10	A comparison of covariance of bit-level and symbol-level IBP.	128
5.11	BER performance of the S-IBPTCs with interleaver delay ≈ 800 , block size $L = 402$ and interleaver span $S = 1$ and the classic TCs with block sizes $L = 400, 800$	129
5.12	BER performance of the S-IBPTCs with interleaver delay ≈ 800 , block size $L = 265$ and interleaver span $S = 2$ and the the classic TCs with block sizes $L = 400, 800$ are also given.	130
5.13	BER performance of S-IBPTCs and the classic TC with interleaver delay 1320 and the 3GPP interleaver.	131
5.14	BER performance of S-IBPTCs and the classic TC with interleaver delay 1320 and the modified semi-random interleaver.	132
5.15	Influence of the interleaver span on the BER performance for various S-IBPTCs with interleaver delay 1320.	132
5.16	BER comparison of S-IBPTCs and the 3GPP defined turbo code of various block sizes.	133
5.17	A comparison of S-IBPDTC applying bit-level and symbol-level IBP with DTC using both Log-MAP and MAX Log-MAP APP decoders.	133
6.1	The block diagram of the proposed VTT-APP decoder applied IBP turbo coding system.	136
6.2	A graph representation for a CRC and S-IBPTC encoded system with interleaving span $S = 1$	138
6.3	The block diagram of IBPTC dynamic decoder.	142

6.4	A comparison of exemplary decoding schedules for classic TC and S-IBPTC when decoding 7 blocks with 2 iterations (four decoding rounds). The numbers in the two rectangular grid-like tables represent the order the APP decoder performs decoding.	144
6.5	A multiple zigzag decoding schedule for an S-IBPTC with the span $S = 1$.	147
6.6	A joint memory management and IBPTC decoding procedure.	150
6.7	Flow chart of a general m -round stopping test.	155
6.8	Block error rate performance of various stopping tests; no memory constraint; $D_{\max} = 30$ DRs.	158
6.9	Average APP DR performance of various stopping tests; $D_{\max} = 30$ DRs, no memory constraint.	159
6.10	The effect of memory constraint and management on the block error rate performance. Curves labelled with infinite memory are obtained by assuming no memory constraint; “fixed DRs” implies that no early stopping test is involved.	160
6.11	Average APP DR performance for various decoding schemes and conditions. Curves labelled with infinite memory are obtained by assuming no memory constraint; “fixed DRs” means no early-stopping condition is imposed.	161
6.12	Block error rate performance of a classic TC using various STs; $L = 800$ bits and $D_{\max} = 30$ DRs.	162
6.13	The effect of various STs on the average APP DR performance of a classic TC with $L = 800$ and $D_{\max} = 30$ DRs.	162
7.1	(a) Factor graph representation of an LDPC code; (b) the grouped factor graph; (c) node grouping.	167
7.2	Multi-stage factor graph for conventional BP algorithm.	168
7.3	Multi-stage factor graph for horizontal-shuffled BP.	169

7.4	Multi-stage factor graph for the new scheduled BP which reduces cycle effect.	170
7.5	Factor graph representation of a CRC- and S-IBPTC-coded communication link.	171
7.6	(a) Node grouping; (b) grouped factor graph.	172
7.7	A multi-stage factor graph for the S-IBPTC pipeline decoding schedule. .	173
7.8	A multi-stage factor graph for the new S-IBPTC decoding schedule. . . .	174
7.9	(a) The multi-stage factor sub-graph associated with Fig. 7.9; (b) the multi-stage factor sub-graph associated with Fig. 7.3; (c) the multi-stage factor sub-graph associated with Fig. 7.4; (d) the initial multi-stage factor sub-graph associated with Fig. 7.4.	175
7.10	(a) The multi-stage factor sub-graph extracted from Fig. 7.7; (b) the multi-stage factor sub-graph extracted from Fig. 7.8.	176
7.11	(a) The sub-graph of the MSFG shown in Fig. 7.7; (b) the sub-graph of the MSFG shown in Fig. 7.8; (c) the causal multi-stage sub-graph associated with the MSFG shown in Fig. 7.7; (d) the causal multi-stage sub-graph associated with the MSFG shown in Fig. 7.8.	177
7.12	(a) Padded virtual nodes retain the regularity of the CMSSG for the beginning nodes on the MSFG; (b) padded virtual nodes retain the regularity of the CMSSG for the last nodes on the MSFG.	178
7.13	A memory saving schedule for the S-IBPTC.	179
7.14	The beginning stages on the partial MSFG associated with Fig. 7.13. . .	180
7.15	BER performance as a function of SNR for three decoding schedules. . .	182
7.16	Average APP decoding round number performance as a function of SNR for three decoding schedules.	182
E.1	Frame error rate comparison between regular and irregular puncturing patterns for code rate=3/4 3GPP Rel'6 turbo code.	199

E.2	Frame error rate comparison between regular and irregular puncturing patterns for code rate= $3/4$ B-IBPTC.	199
E.3	Frame error rate comparison between 3GPP Rel'6 turbo code and B-IBPTC for code rate= $3/4$ irregular puncturing pattern.	200
E.4	Frame error rate comparison between regular and irregular puncturing patterns for code rate= $4/5$ 3GPP Rel'6 turbo code.	200
E.5	Frame error rate comparison between regular and irregular puncturing patterns for code rate= $4/5$ B-IBPTC.	201
E.6	Frame error rate comparison between 3GPP Rel'6 turbo code and B-IBPTC for code rate= $4/5$ irregular puncturing pattern.	201
E.7	Frame error rate comparison between regular and irregular puncturing patterns for code rate= $8/9$ 3GPP Rel'6 turbo code.	202
E.8	Frame error rate comparison between regular and irregular puncturing patterns for code rate= $8/9$ B-IBPTC.	202
E.9	Frame error rate comparison between 3GPP Rel'6 turbo code and B-IBPTC for code rate= $8/9$ irregular puncturing pattern.	203



List of Tables

3.1	(α, β) for some RSC codes.	60
4.1	Throughput and latency analysis for 8 APP decoders	87
4.2	Throughput and latency analysis for 16 APP decoders	88
4.3	Parallelism degree corresponding to various data lengths K and the supported number of interleavers.	94
4.4	Butterfly B-IBP sequences and the corresponding generator polynomials	95
4.5	Double prime interleaver parameters	96
4.6	Comparison to different turbo decoder designs	99
5.1	S-IBP Algorithm	122
E.1	Puncturing patterns for code rate=3/4.	198
E.2	Puncturing patterns for code rate=4/5.	198
E.3	Puncturing patterns for code rate=8/9.	198

Glossary

3GPP	Third Generation Partnership Project
3GPP LTE	Third Generation Partnership Project Long Term Evolution
ADU	A Posteriori Probability Decoding Unit
APP	A Posteriori Probability
ARP	Almost Regular Permutation
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
B-IBP	Block-Oriented Inter-Block Permutation
B-IBPTC	Block-Oriented Inter-Block Permutation Turbo Code
BCH	Bose-Chaudhuri-Hocquenghem
BCJR	Bahl-Cocke-Jelinek-Raviv
BER	Bit Error Rate
BLER	Block Error Rate
C-IBPTC	Continuous Inter-Block Permutation Turbo Code
CE	Cross Entropy
CP	Cyclic Prefix
CRC	Cyclic Redundancy Check
CRC-SC ST	Cyclic Redundancy Check-Sign Check Stopping Test
CRCST	Cyclic Redundancy Check Stopping Test
CMSSG	Causal Multi-Stage Sub-Graph
D-IBPTC	Discontinuous Inter-Block Permutation Turbo Code
DWW	Decoding Window Width
DPSK	Differential Phase Shift Keying
DR	Decoding Round
DRP	Dithered Prime Permutation
DSP	Digital Signal Processing
DTC	Duo-Binary Turbo Code
DVB-RCS	Digital Video Broadcast-Return Channel Satellite
DVB-RCT	Digital Video Broadcast-Return Channel Terrestrial
EXIT	Extrinsic Information Transfer
ESD	Extended Stopping Decision
FER	Frame Error Rate
IBP	Inter-Block Permutation
IBPTC	Inter-Block Permutation Turbo Code

ID-BICM	Iterative Decoding Bit-Interleaved Coded Modulation
IN	Iteration Number
ISI	Inter-Symbol Interference
LDPC	Low Density Parity Check
Log-MAP	Logarithmic Maximum a Posteriori
MAP	Maximum a Posteriori
MAX Log-MAP	Maximum Logarithmic Maximum a Posteriori
MIMO	Multiple Input Multiple Output
MMSE	Minimum Mean Square Error
MR-HST	Multiple-Round Hybrid Stopping Test
MRST	Multiple-Round Stopping Test
MSFG	Multi-Stage Factor Graph
MSFSG	Multi-Stage Factor Sub-Graph
MU	Memory Unit
OFDM	Orthogonal Frequency Division Modulation
QAM	Quadrature Amplitude Modulation
QPP	Quadratic Permutation Polynomial
RDR	Repeated Decoding Round
RS	Reed-Solomon
RSC	Recursive Systematic Convolutional Code
S-C-IBPTC	Stream-Oriented Continuous Inter-Block Permutation Turbo Code
S-IBP	Stream-Oriented Inter-Block Permutation
S-IBPDTC	Stream-Oriented Inter-Block Permutation Duo-Binary Turbo Code
S-IBPTC	Stream-Oriented Inter-Block Permutation Turbo Code
S-TB-IBPTC	Stream-Oriented Tail-Biting Inter-Block Permutation Turbo Code
S-TP-IBPTC	Stream-Oriented Tail-Padding Inter-Block Permutation Turbo Code
SC	Sign Check
SCST	Sign Check stopping Test
SNR	Signal to Noise Ratio
SOVA	Soft Output Viterbi Algorithm
SRID	Single Round Interleaving Delay
ST	Stopping Test
SV	Soft Value
SVST	Soft Value Stopping Test
SWAPP	Sliding-Window a Posteriori Probability
TB-IBPTC	Tail-Biting Inter-Block Permutation Turbo Code
TC	Turbo Code
TCM	Trellis Coded Modulation
TP-IBPTC	Tail-Padding Inter-Block Permutation Turbo Code
VTT	Various Termination Time
VTT-APP	Various Termination Time-a Posteriori Probability

Chapter 1

Introduction

The invention of turbo codes by Berrou *et al.* in the early 1990s [13] has ignited a revolution within the coding research community. The underlying *turbo principle* has since crossed the borderline of coding theory and made far-reaching impacts on various scientific disciplines like communications, computer science, statistics, physics and bioinformatics, to name just a few.

1.1 Turbo decoding

The now classic turbo code is composed of two identical short recursive systematic convolutional codes and a random-like interleaver. A message sequence and its interleaved version are separately encoded by two component encoders and the resulting codeword consists of the original sequence (systematic part) and two parity sequences. Each parity sequence along with the systematic part or its interleaved version forms a conventional convolutional codeword.

The original decoder proposed by Berrou *et al.* has a serial structure with identical a posteriori probability (APP) decoders. Maximum a posteriori probability (MAP) algorithm [8] is iteratively applied to decode each convolutional code and produce reliability estimates about the systematic bits. The reliability estimate, which is universally called extrinsic information now, generated by one APP decoder is interleaved or de-interleaved and then passed on to the other APP decoder for use as the a priori information needed

in its MAP decoding. Such a turbo-like iterative feedback decoding procedure divides the formidable task of maximum likelihood (ML) decoding of the complete codeword into much simpler subtasks of computing the log-likelihood ratio (extrinsic information) associated with each systematic bit locally and exchanging this information between each other. It turns out this turbo decoding scheme is very effective and the resulting performance comes very close to that of the corresponding ML decoder. Because of its outstanding performance and moderate hardware complexity, the class of turbo codes has found its way into industrial standards like 3GPP [1, 2, 3], IEEE 802.16 [56], DVB-RCS/RCT [37, 38], etc.

The turbo decoding process was formulated by Hagenauer *et al.* [50] as a soft-in soft-out inference process that accepts soft inputs—including a priori and channel values—and generate soft outputs which consists of the a priori and channel values and the extrinsic values. The extrinsic value is then used as an a priori value for the ensuing decoding round. The turbo decoding procedure and the way the extrinsic information is passed is often referred to as the *turbo principle*. This principle has been applied to construct and decode serial concatenated convolutional codes [10], turbo TCMs [84], turbo BCH codes [109], turbo product codes or block turbo codes (BTCs) [81, 80], turbo Reed-Muller codes [109], and asymmetric turbo codes [91] (which has different component convolutional codes and offers better performance). Replacing the inner code of a serial concatenated coding system by a differential phase shift keying modulator or signal mapper, one obtains turbo DPSK [53] or iterative-decoded bit-interleaved coded modulation (ID-BICM) [72, 71]. Modelling a channel with memory as a linear FIR filter or equivalently a finite-state machine so that the combination of coding and channel effects becomes a serial concatenated system with the channel performing inner coding, one can detect the received signal by iteratively equalizing the channel effect and then decoding, resulting in turbo equalization [36], turbo space-time processing [6] and turbo (iterative) MIMO detection [7], [19]. By using more than two component codes and

multiple interleavers, Boutillon and Gnaeding [22] investigated the structure of multiple turbo codes. Replacing the one-to-one permutation function by a many-to-one mapping, Frey and MacKay [45] proposed the so-called irregular turbo codes. stream-oriented turbo code was proposed by Hall [51], a pipeline architecture was studied in [101].

1.2 Performance analysis and graph codes

The reasons for the extraordinary performance of the TC, although unclear initially, has been unraveled through many excellent and intensive research efforts. Benedetto and Montorsi [9] showed that the iterative (turbo) decoding algorithm is capable of achieving near-ML performance and the error rate performance improves as the interleaver length increases. They also found that the interleaver length can be traded with component code's complexity and that the number of nearest neighbors rather than the minimum distance dominates the performance, at least when SNR is small. Perez *et al.* [74] analyzed the TC ensemble (over all possible interleavers), examined the code's distance spectrum and came to a similar conclusion that the error floor occurs at moderate to high SNR is due to the relatively small free distance of the component code and its excellent performance at low SNR is resulted from *spectrum thinning*. They found that the low weight codewords, in particular those generated by weight-2 input sequences, dominate the error rate performance especially at the error floor region.

The convergence behavior of the turbo decoding algorithm was analyzed by Richardson [82, 83] from a geometric viewpoint. He also suggested a density evolution approach to compute the thresholds for low density parity check (LDPC) codes. The concept of density evolution was later extended to be applicable to turbo codes [82]. The analysis of El Gamal and Hammons [39] is based on the fact that the extrinsic values at the output of an APP decoder is well approximated by Gaussian random variables and channel values are also Gaussian distributed when the only noise source is AWGN—an observation first noticed by Wiberg [108]. As a Gaussian distribution is completely char-

acterized by its first two moments, the density evolution information can be replaced by SNR transfer. Both [39] and Divsalar *et al.* [34] use similar SNR measures to study the convergence of turbo decoders. Extrinsic information transfer (EXIT) chart [27, 28] proposed by ten Brink plots the increase of extrinsic information through the component decoders based on the measure of mutual information between extrinsic information and the associated information symbol or code symbol.

A special class of graphs called factor graphs [60, 42] can be used to describe the behavior and structure of a turbo-like algorithm. A factor graph decomposes the algorithmic structure into function nodes and variable nodes with edges connecting these function nodes. McEliece *et al.* [66] discovered the connection between the turbo decoding algorithm and the belief propagation (BP) algorithm in artificial intelligence. The graphic and BP interpretations of the turbo decoder have great impacts and opened new arenas on many fronts: new decoding algorithms, schedules and new (graph) codes were proposed, a unified view on iterative decoding algorithm, Kalman filter, the forward-backward, Baum-Welch, and Viterbi algorithms become possible, connection between turbo codes and LDPC codes was established, that amongst coding theory, statistical inference, physics was exploited to the benefits of all involved research communities .

1.3 Low latency/high performance interleavers

The role played by the interleaver in determining the decoding latency, weight distributions and performance of a TC is of critical importance. A TC usually employs a block-oriented interleaving so that the message-passing process associated with an iterative decoder is confined to proceed within a block. The performance of such a TC improves as the block size increases. This is in part due to the fact that the range (interleaving length) of the extrinsic information collected for decoding increases accordingly. But the interleaving size along with the number of iterations are the dominant factors that determine the decoding latency and complexity which, in turn, are often the main

concerns that precluding the adoptability of such codes in high rate communication or storage applications.

The semi-random interleaver [44] increases the codeword weights corresponding to the weight-2 input sequences but not those for weight-4 input sequences. The code matching interleaver of [40] considers the influence of component codes and extended the optimization criterion to include the weight-4 input sequences, yielding performance superior to that based on a semi-random interleaver. The methods proposed in [54, 86] are based on the analysis of the correlation of extrinsic information resulted from cycles in the code graph. They design interleaving rules to reduce the cycle effect and obtain slightly-improved performance.

A common shortcoming of these interleaver designs is their lack of an algebraic structure. A look-up table is therefore needed in implementation. Structural interleavers are now abundant: 3GPP Rel'99 and Rel'6 interleaver [1, 2], dithered relative prime interleaver [30, 31], dithered golden interleaver (DRP) [30], almost regular permutation (ARP) [12, 37, 38, 56], quadratic polynomial permutation (QPP) [90, 85, 92, 93, 3], to name the important ones. These interleavers are generated by few parameters and the corresponding storage requirements are moderate.

Besides a simple algebraic structure, we notice a recent trend indicating that high throughput (> 100 Mbps) turbo decoders [3, 56] are in great demand. The low latency/high throughput applications require that the interleaver be such that the corresponding turbo code is parallel decodable. Some design issues arise because of this requirement. Firstly, we notice that since a parallel decoder consists of many APP decoders, each responsible for decoding a (non-overlapping) part of the incoming block, these decoders would simultaneously and periodically access the memory banks that store the extrinsic information and channel values through hardwires. Thus the parallel decodable requirement implies that the interleaver structure must be memory contention-free and allow simple interconnecting network for hardwire routing. Next, practical

system design concerns call for flexible degrees of parallelism and arbitrary continuous interleaving length so that one has flexible choice on both the number of APP decoders and the frame (packet) size.

Random interleavers, although offer satisfactory error rate performance, incur serious memory contention. Implementing temporary memory buffer [49] to avoid memory contention is a viable solution but the storage grows linearly with the number of APP decoders. Resolving the contention by a sophisticated memory mapping function [95] requires a table for each interleaving length. The table requires memory storage for addressing and memory control which results in extra hardware complexity. A large number of interleaving lengths thus need many memory addressing tables and causes increased hardware complexity. New industrial standards such as 3GPP [1, 2, 3], DVB-RCS/RCT [37, 38], IEEE 802.16 [56] do not favor this approach for the demand of large number of interleavers and low complexity memory addressing. A variable length interleaver structure that resolves memory contention with on-fly generated memory mapping function is an efficient and welcome approach.

Some of the existing interleavers like the DRP, ARP and QPP do have simple algebraic structures and possess the memory contention-free property, the DRP even yields a minimum distance that is close to the known upper bound, resulting in outstanding performance especially for frame error rate below 10^{-6} . However, none of them takes into account the other requirements which are of concern mainly to the circuit design community.

A fully-connected network can be used but the complexity grows in proportion to the square of the parallelism degree. The average routing length also increases, bringing about longer routing latency and higher power consumption. The network configuration is close related to the interleaving/deinterleaving rule used. Irregular routing control should be avoid as it necessitates complex controlling signalling. For more detailed discussion on the routing network complexity and network configuration signalling please

refer to [78, 68, 67, 33]. Memory addressing also depends on interleaver design. [70, 95, 96] have investigated the memory addressing and permutation table storage problems. 3GPP Rel'99 and Rel'6 [1, 2] applies the prime interleaver as the intra-row permutation and this interleaver needs storage for the permutation table of each row. The intra-memory bank addressing induces storage complexity if the table can not be generated on-fly. Therefore 3GPP LTE QPP [3] avoids the storage and replaces the interleaver. Popular hardware-friendly designs that avoids the above-mentioned storage requirement ARP [12, 37, 38, 56] and QPP [90, 85, 92, 93, 3]. Some parallelization methods are suggested in [49, 20, 98, 97].

1.4 Statement of purpose: main contributions

The above discussion shows designing the interleaver for a TC for wireless applications has to consider both error rate performance and hardware/memory complexity. The main purpose of this thesis is to present a systematic and unified interleaver design flow and guideline that enable one to construct an interleaver of arbitrary practical lengths which not only meet all the above requirements but also guarantee little or no performance loss with respect to the that achievable by the best known code of the same or comparable interleaving lengths. To distinguish the TC without any constraints and the TC applying out interleaver, we refer the former class as the classic TC.

The structure we propose is called inter-block permutation (IBP) interleaver. The IBP technique can be regarded as a simple way to build a larger interleaver based on smaller interleavers. It performs an extra inter-block permutation on those blocks that have already been interleaved by intra-block permutation. As the interleaver along with the component code determines the structure of classic TC, the concept of IBP interleaver is also similar to Tanner's approach for constructing a large (long) code with small codes. The proposed interleaver structure is general enough to encompass all known important interleavers as special cases yet viable for generating new solutions.

We summarize the main contributions of our work documented in this thesis as follows.

1. We present a unified approach and design flow to build interleavers that not only have good distance properties but also meet all the major implementation requirements for high throughput turbo decoders. More specifically, the resulting interleaver structure (i) is maximum memory contention-free, (ii) allows low complexity routing network structure and simple signalling circuits, (iii) offers flexible choice in the degrees of parallelism, (iv) is easily tailored to serve the continuous interleaving length requirement, (v) supports the high-radix APP decoder structure and (vi) includes all existing good interleavers as its subclasses.
2. For a stream-oriented application, our technique overcomes the dilemma between increasing the range of message exchange and extrinsic information collection and reducing the interleaving size (and therefore the decoding delay). It outperforms classic TCs with the same decoding delay and offers new design choices and trade-offs that are unavailable for classic TC design.
3. We derive codeword weight bounds for weight-2 and weight-4 input sequences. More importantly, we use these bounds and computer simulations to prove that the advantages mentioned in 1-2 are achieved with little or no loss in error rate performance.
4. We present a dynamic cooperative pipelined decoder structure that incorporate an efficient memory manager and a class of highly reliable early-stopping rules. The proposed decoder structure gives improved performance with reduced latency and memory requirement.
5. For non-parallel decoding, the IBP interleaver is still capable of reducing the decoding latency while maintaining satisfactory performance.

6. A graphic tool called multistage factor graphs is developed to analyze the behavior of parallel and pipelined decoding schedules. It is applied to design a new pipelined decoding schedule with reduced memory requirement and can also be used to design better schedule for decoding LDPC codes.

1.5 Existing interleavers as instances of IBP interleaver

For a classic TC using an IBP interleaver, the encoder partitions the incoming data sequence into L -bit blocks upon which the IBP interleaver performs intra-block and then inter-block permutations. For example, the IBP interleaver may move contents of a block either to coordinates within the same block or to its $2S$ immediate neighboring blocks so that the IBP-interleaved contents of a block are spread over a range of $2S + 1$ blocks centered at the original block. Such an IBP interleaver is said to have the (left or right) IBP span S .

For any reasonable good interleaver of size N , partitioning each N -bit group into $L = \lfloor N/W \rfloor$ -bit blocks immediately transforms the interleaving rule into an IBP structure like that shown in Fig. 1.1. Such a structure can also be found in other codes such as product codes. Consequently, all classic TCs and product codes can be regarded as subclasses of inter-block permutation turbo codes (IBPTCs). There are, however, two major distinctions between classic TCs and most other subclasses of IBPTCs.

Firstly, for a classic TC with an interleaving length of W blocks, encoding within each disjoint group of W consecutive blocks is continuous across blocks while a product code encodes each row (column) separately (discontinuously). More specifically, the product code encoder divides information stream into multiple blocks and independently encodes each block. In general, the class of IBPTCs can encode each block either separately or continuously. Secondly, an interleaver used in a classic TC, after the above virtual regular partition, usually yields a non-regular local interleaving structure, i.e., the interleaving

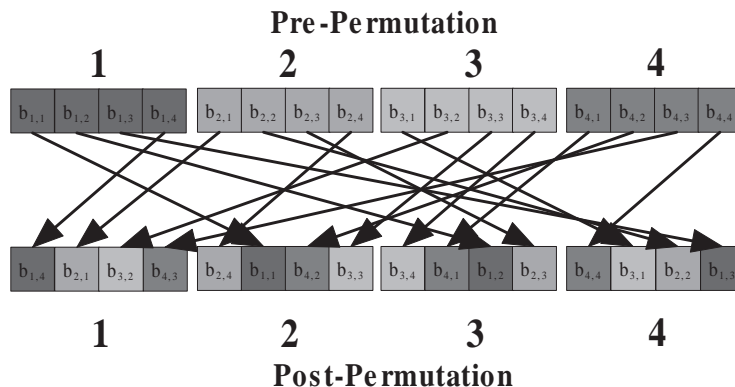


Figure 1.1: An inherent IBP structure can be found in most practical interleavers.

relation between a block and other blocks in the same group does not follow the same permutation rule. In contrast, product codes and the proposed IBPTCs have much more regular local interleaving structures. An appropriate regular local interleaving (and deinterleaving) structure makes implementation easier and, as mentioned before, provides properties that are useful for parallel decoding, e.g., (memory access) contention-free and simple routing requirement. Moreover, with or without parallel decoding, as the examples in Section 6.2 show, it also results in reduced decoding delay. Regular (identical) local interleaving structure supports large range of interleavers, makes the resulting IBP interleaver expandable and minimize the associated implementation cost, e.g. 3GPP LTE QPP [3], IEEE 802.16 [56].

1.6 Overview of chapters

In order that this thesis be self-contained we provide major background material related to our work in Chapter 2. Two fundamental guidelines are provided in Chapter 3 for constructing IBP interleavers with good distance and maximum contention free properties. The first rule demands that the IBP rule be block-invariant and identical intra-block permutation e used. The second rule implies that the permutation should be periodic within its span. Following these and other minor guidelines we are able to

construct interleavers that meet most of the hardware requirements while maintaining good distance properties. Searching for large range of interleaving lengths also become easier.

We divide the class of IBP interleavers into block-oriented IBP (B-IBP) interleaver and stream-oriented IBP (S-IBP) interleaver. The B-IBP interleavers are treated in Chapter 4 while Chapter 5 deals with the stream-oriented IBP (S-IBP) interleavers. The B-IBP interleavers include popular interleavers such as the ARP and QPP and usually have hardware constraints more stringent than those on stream ones. Encoding variable information lengths with the same hardware architecture. We suggest simple memory mapping functions that support flexible choices in the number of memory banks and APP decoders. An alternate decomposition of an IBP rule called reverse IBP manner offers additional flexibility. In order to support the high-radix APP decoder and the generalized maximum memory contention-free property, we impose two constraints on the intra-block permutation and obtain simple and easily generated memory mapping functions. Our network-oriented design allows low complexity butterfly network and simple routing control signalling. To deal with variable message lengths without throughput loss, a shortening and puncturing algorithm is proposed to maintain both performance and hardware implementation edge. We provide an interleaver design with the interleaving length ranging from 40 to 6144 bits. A VLSI implementation example based on this design with a specific interleaving length of 4096 bits is also given.

We prove that our S-IBP interleaver construction gives larger codeword weight upper-bounds for the weight-2 and weight-4 input sequences than those of classic TCs with the same interleaver latency. Our S-IBP interleaver is well suited to pipeline decoder architectures [101]. To improve both hardware/memory efficiency and error rate performance we propose a dynamic decoder architecture which includes a memory manager and an early-stopping mechanism. The decoder also admits new decoding schedules and offers trade-off between throughput and hardware/memory complexity.

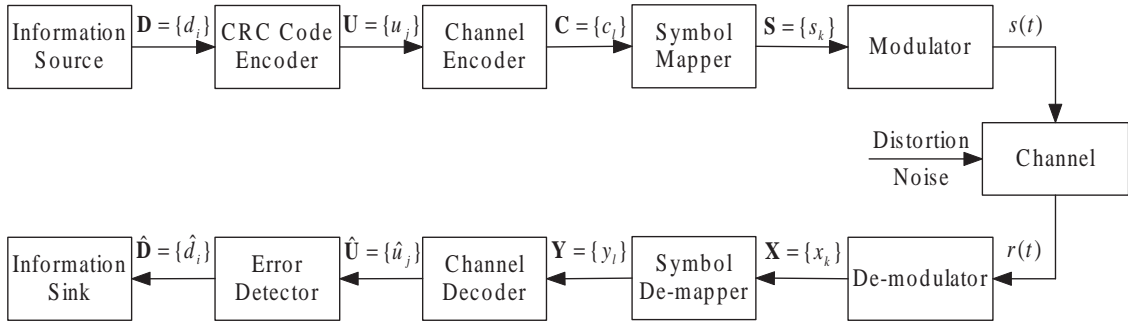
In Chapters 6 we discuss issues concerning the pipelined decoders. Early-stopping in iterative decoding is an critical and very practical issue. Regarding the iterative decoding as an instance of sequential decision processes, early-stopping reduces the number of iterations (and thus the computation complexity/power) at high SNR without performance loss at low SNR. CRC code, sign check, soft value (cross entropy) are some of the more popular stopping schemes [50, 88, 65, 4]. The CRC code offers more reliable stopping decision than other schemes do at the cost of increased overhead and reduced bandwidth efficiency. The proposed multiple-round stopping mechanism enhances the stopping reliability with a smaller overhead, leading to the improved latency and error rate performance.

Judicial design of decoding schedules is crucial for the parallel or pipeline turbo decoding. Conventional factor graphs are incapable of describing such decoding schedules for turbo codes and LDPC codes. In Chapter 7, we develop a new graphic tool called the multi-stage factor graphs to describe the the time-evolving message-passing and evaluate the performance of various decoding schedules for the parallel and pipeline decoders. A good decoding schedule is important in rendering satisfactory performance, e.g., the horizontal shuffled belief propagation algorithm [63] outperforms conventional belief propagation algorithm [60] in terms of the number of iterations required to achieve a desired error rate performance. Multi-stage factor graphs can be used to show the cycle effect and design new decoding schedule to avoid short cycles. We propose a novel decoding schedule for a stream-oriented IBPTC that requires much less memory storage and slightly increased computation but yields similar error rate performance. Finally, in Chapter 8 we summarize the main results of our work.

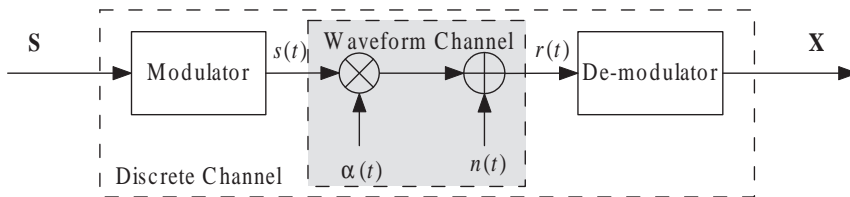
Chapter 2

Fundamentals

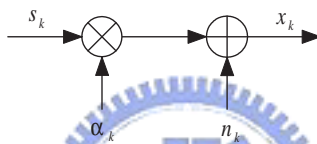
This chapter provides the backgrounds of this thesis. Channel coding embedded in a digital communication system [79] overcomes channel impairments, e.g. thermal noise, multi-path fading, etc. Turbo code [13] is an important candidate among these channel coding schemes. This code possesses better error rate performance comparing with the convolutional code with constraint length 41 [79] and can apply iterative decoding algorithm with less complexity to achieve the performance comparing to the Viterbi algorithm. The algorithm adopts maximum a posteriori (MAP) [8] algorithm to generate the extrinsic information for successive decoding as the a priori information and corrects errors after several iterations. In order to further reduce implementation complexity, many researches [50, 104] focus on the MAP algorithm simplification. Due to high performance with low computation complexity, many standards adopt turbo code, e.g. 3GPP Rel'99 and Rel'6 [1, 2], 3GPP LTE [3], DVB-RCS[37], DVB-RCT[38] etc. 3GPP LTE further requires the throughput exceeding 100Mbps and high throughput turbo decoder architecture becomes important topic; interleaver determines the implementation complexity and error rate performance. Theoretical performance analysis and codes characteristics are also of our interest. The factor graph [60, 42] expounds the structure of turbo code and some decoding algorithms are derived. Given the graph and decoding algorithm, the extrinsic information transfer (EXIT) chart [27, 28] and density evolution [34] explain the convergence behavior at various signal-to-noise ratios



(a)



(b)



(c)

Figure 2.1: (a) The block diagram of a generic digital communication system; (b) the block diagram of a simplified channel model; (c) the block diagram of a discrete memoryless channel model.

(SNRs) and help us choosing a good code. We can further modify the graph to acquire some distance bounds [75] which dominate the performance at high SNR. The following sections detail these implemental and theoretical backgrounds.

2.1 Digital communication system

Fig. 2.1 (a) shows a generic digital communication system block diagram which includes three parts: 1) channel; 2) modulation, demodulation, mapper and de-mapper; 3) error correction and detection. Channel imposes non-ideal effects and distorts the modulated continuous waveform. Demodulator and de-mapper convert the distorted waveform into samples. Error correction recovers these samples and renders decoded

sequences. At last error detection verifies the correctness of decoded sequences. The following subsections will elaborate these three parts.

2.1.1 Discrete memoryless channel model

A memoryless discrete channel model is characterized by a multiplicative distortion and a complex AWGN in Fig. 2.1 (c) and the received sample is

$$x_k = \alpha_k \cdot z_k + n_k, \quad (2.1)$$

where $\mathbf{A} = \{\alpha_k\}$ and $\mathbf{N} = \{n_k\}$ are identical independent Rayleigh (Rician) distributed random variables and complex Gaussian distributed random variables with $C(0, N_0)$ respectively.

This model can replace channel, modulator and de-modulator in Fig. 2.1 (a). We combine modulator and demodulator to construct a discrete channel model which is shown in Fig. 2.1 (b). The thermal noise introduced by component devices can be modelled by the white Gaussian random process. As for the non-selective fading [79], we model fading process $\alpha(t)$ as a Rayleigh or Rician distributed random process and the value is almost invariant during each symbol period. As we apply a perfect channel interleaver between de-mapper and channel decoder, the correlation between adjacent modulation symbols diminishes after channel de-interleaving. Since the fading attenuation is uncorrelated and the thermal noise is white, the channel model can be further simplified as shown in Fig. 2.1 (c).

Time domain dispersive multi-path fading effect introducing inter-symbol interference (ISI) also can be modelled by a memoryless discrete channel model as we apply orthogonal frequency division modulation (OFDM). OFDM applies a cyclic-prefix (CP) to maintain the longest path delay within the interval of the CP and we can sample a symbol period without interferences from other symbols and the ISI effect disappears. The frequency domain amplitude attenuation incurred from all paths can be modelled as a Rayleigh distributed random variable if there is no line of sight. If there is a line

of sight, the distribution is Rician distributed probability density function (pdf). The model in Fig. 2.1 (c) can be re-applied.

The shadowing effect is a long-term effect and generally lasts more than one coding block. The effect can be modelled into the noise strength. Therefore the simplified memoryless discrete channel model properly covers most scenarios and this thesis will apply this model as our simulation assumption.

2.1.2 Mapper and de-mapper

Mapper bridges channel encoder and modulator; de-mapper generates log-likelihood or log-likelihood ratio for code bits corresponding to a sample x_k . Mapper maps n code bits into a modulated symbol $S_m \in \mathbf{S}$ by a mapping rule

$$\Phi : \{b_{m,0}, b_{m,1}, \dots, b_{m,n-1}\} \rightarrow S_m, \quad (2.2)$$

where $b_{i,j} \in \{0, 1\}$ and $|\mathbf{S}| = 2^n$. Based on the mapping rule, de-mapper can apply maximum a posteriori (MAP) algorithm to generate a log-likelihood ratio of the t th code bit corresponding to the i th mapping bit as

$$L(c_t) = \log \frac{\sum_{b_{m,i}=0, b_{m,j}, j \neq i} p(S_m | x_k)}{\sum_{b_{m,i}=1, b_{m,j}, j \neq i} p(S_m | x_k)}. \quad (2.3)$$

$L(c_t)$ is generally applied as y_t to the following channel decoder.

We further consider binary phase shift keying (BPSK) and the mapping rule is

$$\Phi : \begin{cases} b_{0,0} = 0 \rightarrow S_0 = +1 \\ b_{1,0} = 1 \rightarrow S_1 = -1 \end{cases}. \quad (2.4)$$

If $P(c_t = 0) = P(c_t = 1) = 0.5$ and the pdf of x given S_m and channel attenuation α is

$$p_{X|S_m, \alpha}(x | S_m, \alpha) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x - \alpha S_m)^2}{2\sigma^2}} \quad (2.5)$$

with $\sigma^2 = N_0/2$, the log-likelihood ratio in eqn. (2.3) with $k = t$ becomes

$$L(c_t) = \log \frac{p(S_0 | x_t, \alpha_t)}{p(S_1 | x_t, \alpha_t)} = \log \frac{p_{X|S_0, \alpha}(x_t | S_0, \alpha_t)}{p_{X|S_1, \alpha}(x_t | S_1, \alpha_t)} + \log \frac{p(S_0)}{p(S_1)} = \frac{4\alpha_t x_t}{N_0}. \quad (2.6)$$

2.1.3 Error control system

Error correction and detection are two main error control functions. Error correction function applies channel encoder and decoder to overcome channel distortion. Channel encoder generates coded sequence and channel decoder recovers the distorted sequence after channel corruption. Convolutional code, turbo code, LDPC code and Reed-Solomon code are now popular error correction codings. Error detection function generally relies on a cyclic redundancy check (CRC) code [109]. The CRC code encoder adds the error check parities behind information sequence and error detection function verifies the consistency between the decoded information sequence and parities. If the error verification fails, error control system discards decoded sequence or requests re-transmission to render higher successful transmission probability. These two functions enhance and guarantee data transmission robustness via channel corruption.

2.2 Convolutional code

Convolutional code encoder features simple structure which can be realized by finite shift registers and adders and is shown in Fig. 2.2. The codeword length is not stringently constrained comparing to the block code, e.g. RS code. The simple structure results in that the Viterbi algorithm [103, 41], soft decoding algorithm, is applicable and the resultant performance is better than the performance of RS code. In order to expound the code structure, we provide some mathematical notations at first.

2.2.1 Mathematical notations

Denote by $\mathbb{F}_2[D]$ the ring of binary polynomials, where the power of D is nonnegative and the coefficients are in Galois field $GF(2)$, e.g. $p(D) = \sum_{i=0}^{\infty} p_i D^i \in \mathbb{F}_2[D]$ and $p_i \in GF(2)$. Multiplication of two polynomials is a polynomial multiplication with coefficient operation performed under $GF(2)$. Since the power of D is nonnegative, 1 is the only only element with its inverse and $\mathbb{F}_2[D]$ is a ring.

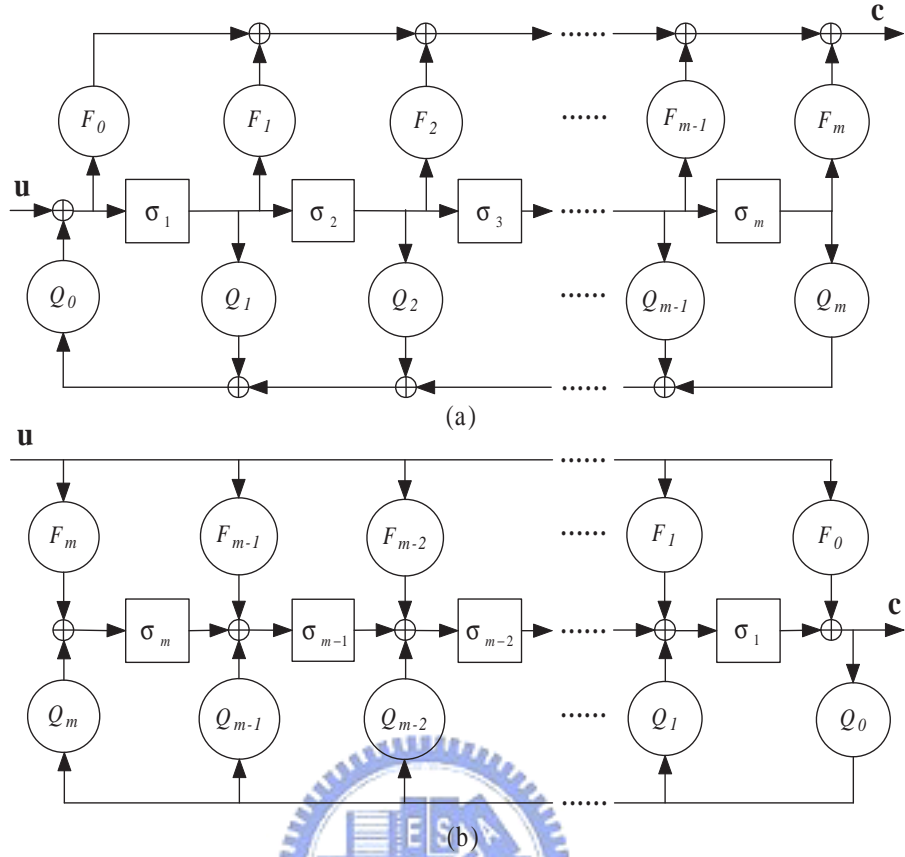


Figure 2.2: (a) The controller canonical form; (b) The observer canonical form.

$\mathbb{F}_2(D)$ denotes the field of binary rational functions. Each element in $\mathbb{F}_2(D)$ is expressed by $\frac{x(D)}{y(D)}$, where each pair $x(D), y(D) \in \mathbb{F}_2[D]$ with $y(D) \neq 0$. Apparently all elements $\frac{x(D)}{y(D)} \in \mathbb{F}_2(D)$ are invertible and they form a field.

We further denote by $\mathbb{F}_2^{a \times b}[D]$ and $\mathbb{F}_2^{a \times b}(D)$ the $a \times b$ matrix of ring of binary polynomials and the $a \times b$ matrix of the field of binary rational functions respectively. All entries in $X^{a \times b}(D) \in \mathbb{F}_2^{a \times b}[D]$ and $Y^{a \times b}(D) \in \mathbb{F}_2^{a \times b}(D)$ belong to $\mathbb{F}_2[D]$ and $\mathbb{F}_2(D)$ respectively. If $a = 1$, the notations are simplified as $\mathbb{F}_2^b[D]$ and $\mathbb{F}_2^b(D)$.

2.2.2 Encoder

A rate $R = a/b$ convolutional code encodes input sequence $\mathbf{u} = \mathbf{u}_0 \mathbf{u}_1 \mathbf{u}_2 \dots$ and generates code sequence $\mathbf{c} = \mathbf{c}_0 \mathbf{c}_1 \mathbf{c}_2 \dots$, where $\mathbf{u}_i = \{u_i^1, u_i^2, \dots, u_i^a\}$ and $\mathbf{c}_j = \{c_j^1, c_j^2, \dots, c_j^b\}$.

A sequence can be represented by the delay operator D (D-transform), and we have

$$\mathbf{u}(D) = \mathbf{u}_0 + \mathbf{u}_1 D + \mathbf{u}_2 D^2 + \dots, \quad (2.7)$$

$$\mathbf{c}(D) = \mathbf{c}_0 + \mathbf{c}_1 D + \mathbf{c}_2 D^2 + \dots, \quad (2.8)$$

where $\mathbf{u}(D) \in \mathbb{F}_2^a[D]$ and $\mathbf{c}(D) \in \mathbb{F}_2^b[D]$. The encoder input and output relation can be expressed as

$$\mathbf{c}(D) = \mathbf{u}(D)\mathbf{G}(D), \quad (2.9)$$

where $\mathbf{G}(D) \in \mathbb{F}_2^{a \times b}(D)$ is a transfer function matrix. The matrix has a form

$$\begin{aligned} \mathbf{G}(D) &= \mathbf{Q}^{-1}(D)\mathbf{F}(D) \\ &= (Q_0 + Q_1 D + \dots + Q_m D^m)^{-1}(F_0 + F_1 D + \dots + F_m D^m), \end{aligned} \quad (2.10)$$

where $\mathbf{Q}(D) \in \mathbb{F}_2^{a \times a}[D]$ and $\mathbf{F}(D) \in \mathbb{F}_2^{a \times b}[D]$. The polynomial matrix $\mathbf{Q}(D)$ is a diagonal matrix with $Q_0 = \mathbf{I}_a$, \mathbf{I}_a is an $a \times a$ identity matrix, i.e. the elements off the diagonal entries are zero. The matrix $\mathbf{G}(D)$ can be realized in the controller canonical form shown in Fig. 2.2 (a). Observer canonical form is another way in realizing transfer function matrix. We substitute eqn. (2.10) into eqn. (2.9) to render

$$\mathbf{c}(D) = \mathbf{u}(D)(F_0 + F_1 D + \dots + F_m D^m) + \mathbf{c}(D)(Q_1 D + \dots + Q_m D^m). \quad (2.11)$$

Fig. 2.2 (b) shows the corresponding linear circuit which is the observer canonical form for a convolutional code encoder. Both controller and observer forms realize the transfer function matrix.

From the above description, we give a formal definition of a convolutional code as follows.

Definition 1 A code rate $R = a/b$ (binary) convolutional code encoder over the field of rational functions $\mathbb{F}_2(D)$ is a linear mapping

$$\begin{aligned} \Phi_{cc} : \mathbb{F}_2^a(D) &\rightarrow \mathbb{F}_2^b(D) \\ \mathbf{u}(D) &\mapsto \mathbf{c}(D) \end{aligned}$$

which can be represented as

$$\mathbf{c}(D) = \mathbf{u}(D)G(D),$$

where $G(D) \in \mathbb{F}_2^{a \times b}(D)$ is an $a \times b$ transfer function matrix of rank a and $\mathbf{c}(D)$ is a code sequence generated from the information sequence $\mathbf{u}(D)$.

Systematic recursive convolutional code is of our further interest. The systematic code has information bits in a code sequence and the transfer function matrix $\mathbf{G}(D)$ can be

$$\mathbf{G}(D) = [\mathbf{I}_a \ \mathbf{R}(D)], \quad (2.12)$$

where $\mathbf{R}(D) \in \mathbb{F}_2^{a \times (b-a)}(D)$. The recursive code generates large number of nonzero code bits if $\mathbf{u}(D)\mathbf{Q}^{-1}(D) \notin \mathbb{F}_2^a[D]$. For both structures, we give definitions as below.

Definition 2 A rate $R = a/b$ convolutional code encoder whose information sequence appears unchanged among the code sequences is called a systematic encoder.

Definition 3 A rate $R = a/b$ convolutional code encoder whose transfer function matrix $\mathbf{G}(D)$ has $\mathbf{Q}(D) \neq \mathbf{I}_a$ is a recursive encoder.

2.2.3 State space, state diagram and trellis representation

The encoder shown in Fig. 2.2 (a) is finite state machine and the values in registers characterize the state space Σ , i.e. $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_m) \in \Sigma$. When information sequence is input, the state varies with time. In order to record the state change, we further define the state at time t as $\sigma^t = (\sigma_1^t, \sigma_2^t, \dots, \sigma_m^t) \in \Sigma$. The input and output relation at the time t is

$$\sigma^{t+1} = \sigma^t \mathbf{A} + \mathbf{u}_t \mathbf{B}, \quad (2.13)$$

$$\mathbf{c}_t = \sigma^t \mathbf{C} + \mathbf{u}_t \mathbf{D}, \quad (2.14)$$

where \mathbf{A} is the $(m \times m)$ state matrix, \mathbf{B} is the $(a \times m)$ control matrix, \mathbf{C} is the $(m \times b)$ observation matrix and \mathbf{D} is the $(a \times b)$ transition matrix [64]. These two equations

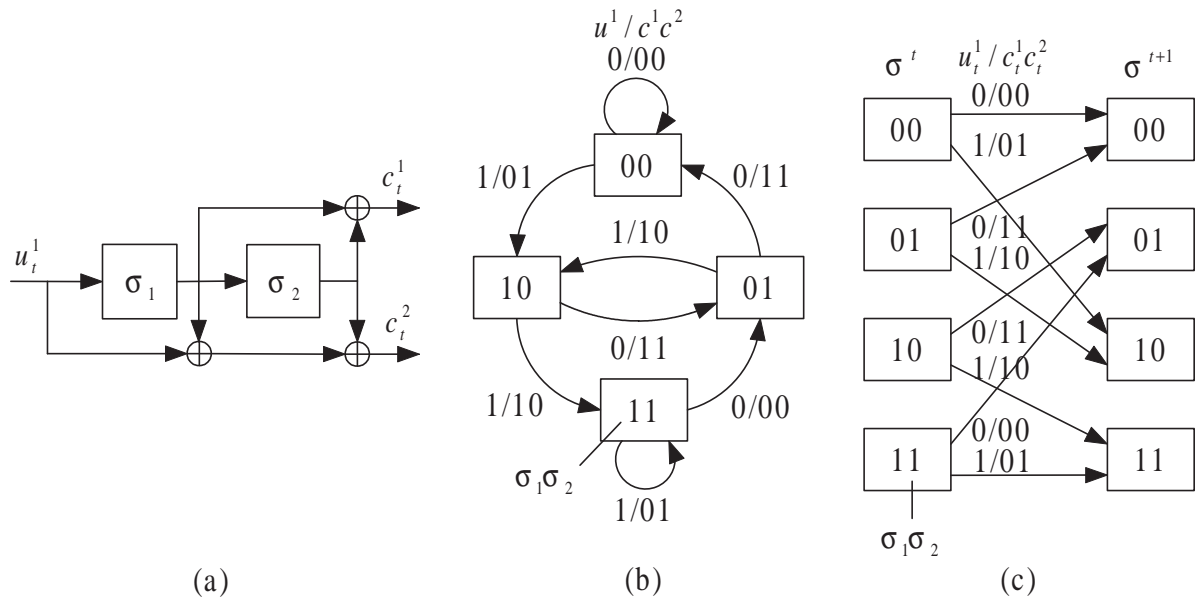


Figure 2.3: (a) The encoder associated with $G(D) = (D + D^2, 1 + D + D^2)$; (b) state diagram; (c) trellis segment.

are state space representation of a convolutional code encoder and show the state time evolution involving an input information sequence.

Two kinds of graphical representations, state diagram and trellis diagram, can characterize the encoding and facilitate decoding algorithm derivation. The state diagram draws the state transition of the encoder. If there is a transition from the state σ to state σ' , we draw a directed edge from state σ to state σ' and note “input information bits/output code bits” on the edge to represent input/output relations. However the state diagram does not represent the state transition involving with time. We draw a trellis segment associate with states σ^t and σ^{t+1} . If there is a transition from state σ^t to state σ^{t+1} , we plot a directed edge from state σ^t to state σ^{t+1} and note “input information bits/output code bits” on the edge. Then we connect trellis segments into a trellis diagram to represent input information sequence, code sequence and state transition sequence involving with time. Owing to the trellis representation, the Viterbi [103, 41] and BCJR [8] algorithms are easily visualized and demonstrated.

Example 1 Fig. 2.3 (a) shows a rate $R = 1/2$ convolutional code with the transfer function matrix $G(D) = (D + D^2, 1 + D + D^2)$, where the associated the state matrix, control matrix, observation matrix and transition matrix are

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \mathbf{B} = [1 \ 0], \mathbf{C} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \text{ and } \mathbf{D} = [0 \ 1]. \quad (2.15)$$

Figs. 2.3 (b) and (c) depict the graphical representations. According to eqns. (2.13) and (2.14), we draw the state diagram shown in Fig. 2.3 (b). A trellis segment associated with σ^t and σ^{t+1} is plotted in Fig. 2.3 (c). Given an initial state condition and a termination scheme, a complete trellis diagram can be obtained by connecting these trellis segments.

2.2.4 Termination

Tail-padding and tail-biting are two popular termination methods for a convolutional code encoding an information sequence of finite length L . The tail-padding terminates codeword at a specific state by padding bits and avoids an unknown end state on decoder side. The tail-biting keeps the initial state and end state the same and the decoder can guess the both states. However the first scheme decreases code rate and the second scheme induces extra decoder computational complexity.

The tail-padding assigns extra bits behind an information sequence to terminate the end state to all zero state. The bit assignment is different for the recursive and non-recursive convolutional codes. For the non-recursive code, we pad $a \cdot m$ zeros behind an information sequence and the end state becomes the all zero state. However padding $a \cdot m$ zeros behind an information sequence does not guarantee the end state being the all zero state for the recursive code. In order to find the padding bits for the recursive code, we can solve the following equation with $\sigma^{L+m} = 0$ to obtain the padding bits

$(\tilde{\mathbf{u}}_L, \tilde{\mathbf{u}}_{L+1}, \dots, \tilde{\mathbf{u}}_{L+m-1})$.

$$\begin{aligned}\sigma^{L+1} &= \sigma^L \mathbf{A} + \tilde{\mathbf{u}}_L \mathbf{B} \\ \sigma^{L+2} &= \sigma^{L+1} \mathbf{A} + \tilde{\mathbf{u}}_{L+1} \mathbf{B} \\ &\vdots \\ \sigma^{L+m} &= \sigma^{L+m-1} \mathbf{A} + \tilde{\mathbf{u}}_{L+m-1} \mathbf{B}\end{aligned}\tag{2.16}$$

Solving the eqn. (2.16) seems hard but the padding bits can be acquired easier. Take the encoder shown in Fig. 2.2 (a) as an example, the padding bits are $\tilde{\mathbf{u}}_{L+j} = \sum_{i=1}^m \sigma_j^{L+i} Q_i$, where $0 \leq j < m$.

The tail-biting for a non-recursive convolutional code acquires the end state by flushing the tail bits of an information sequence into an encoder. Because the encoder is non-recursive, the end state only depends on the last $a \cdot m$ bits of an information sequence. We can flush $(\mathbf{u}_{L-m}, \mathbf{u}_{L-m+1}, \dots, \mathbf{u}_{L-1})$ into an encoder and initial state is $\sigma_0 = \sigma_L = \sum_{s=0}^{m-1} \mathbf{u}_s \mathbf{B} \mathbf{A}^{(m-1)-s}$. Take the encoder shown in 2.2 (a) as an example, the initial state is $(\mathbf{u}_{L-m}, \mathbf{u}_{L-m+1}, \dots, \mathbf{u}_{L-1})$.

The tail-biting method for the recursive code requires double computation complexity to encode information sequence twice: the first encoding obtains the initial state and the second encoding generates a codeword. Denote by $\sigma^{t,[zi]}$ and $\sigma^{t,[zs]}$ the zero-input solution and the zero-state solution. Eqn. (2.13) implies

$$\sigma^t = \sigma^{t,[zi]} + \sigma^{t,[zs]} = \sigma^0 \mathbf{A}^t + \sum_{s=0}^{t-1} \mathbf{u}_s \mathbf{B} \mathbf{A}^{(t-1)-s} = \sigma^0 \mathbf{A}^t + \sigma^{t,[zs]}.\tag{2.17}$$

The zero-state solution is $\sigma^{t,[zs]} = \sum_{s=0}^{t-1} \mathbf{u}_s \mathbf{B} \mathbf{A}^{(t-1)-s}$. When $t = L$ and $\sigma^0 = \sigma^L$, we have

$$\sigma^0 = \sigma^L = \sigma^0 \mathbf{A}^L + \sigma^{L,[zs]}.\tag{2.18}$$

If the matrix $\mathbf{I} + \mathbf{A}^L$ is invertible, we have the initial state as

$$\sigma^0 = \sigma^{L,[zs]} (\mathbf{I} + \mathbf{A}^L)^{-1}.\tag{2.19}$$

The encoder acquires the zero-state solution $\sigma^{L,[zs]}$ by inputting information sequence and calculates the initial state σ^0 by eqn. (2.19). Then the encoder applies the initial state σ^0 to generate a codeword.

The length of an input information sequence determines if the tail-biting is applicable. An invertible matrix $\mathbf{I} + \mathbf{A}^L$ is the necessary condition to calculate the initial state σ^0 . If there exists an L such that $\mathbf{I} + \mathbf{A}^L = 0$, the initial state can not be found and encoding fails. Take 3GPP turbo code [1] as an example. The state matrix is $\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$ and $\mathbf{A}^7 = \mathbf{I}_3$. It can not encode information sequence when the length is the multiple of 7.

2.2.5 Soft output decoding algorithm for convolutional code

Soft output convolutional code decoding algorithms are applicable for iterative decoding and of our interest. Popular convolutional code decoding algorithms can be roughly classified into three classes: maximum a posteriori (MAP) algorithm [8], Viterbi algorithm [103, 41] and sequential decoding algorithm [58]. The MAP algorithm sums up the likelihoods of all codeword sequences corresponding to a symbol at time t and chooses the most likely symbol as a decision. The algorithm achieves the best symbol error rate performance but requires the highest complexity due to large amount of codewords. Thanks for the independent noise corruption assumption, Bahl et al. [8] provides a simplified decoding algorithm and the complexity is affordable. The Viterbi algorithm is a well-known maximum likelihood decoding algorithm. Due to independent noise corruption assumption, the algorithm can discard partial codewords with less likelihood in advance and minimize the complexity in searching a most likely codeword. However the complexity of both algorithms are linear to the number of state and the state grows exponentially with the length of the constraint length or the number of registers. In fact, only partial codeword sequences are necessary to record at high SNR and the complexity can be further minimized by sequential decoding algorithm. Sequential decoding algorithm is a tree search algorithm and always extends the partial codeword on the most likely path. If the channel condition is good, it acquires the maximum likelihood codeword with less complexity.

Among these three algorithms, MAP algorithm makes decision upon the summed up likelihoods and the sum can be used as the soft output for iterative decoding. The Viterbi algorithm and sequential algorithms make decisions upon a searched sequence and outputting likelihood for all information symbols necessitates larger complexity. Hagenauer et al. [50] proposed a novel soft output Viterbi algorithm (SOVA) to provide likelihoods. The SOVA inherits the same low complexity operation on codeword path searching but still requires the extra complexity for early outputting decisions, temporary storage proportional to the truncation length and necessitating the trace back [17] or register exchange [89] to output decisions. The decision making imposes extra complexity comparing to the MAP algorithm. Therefore the following focuses on the MAP algorithm and its variants.

Due to the independent noise corruption assumption, the MAP algorithm for convolutional code decoding can be decoupled into three steps: forward recursion, backward recursion and combination. Assume a code rate a/b convolutional code of length L , denote a received codeword samples and a sample vector at time t by $\mathbf{Y} = \{\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_{L-1}\}$ and $\mathbf{y}_t = \{y_t^0, y_t^1, \dots, y_t^{b-1}\}$. Let $\mathcal{C}(\sigma^t, \mathbf{c}_t, \mathbf{u}_t, \sigma^{t+1})$ be a set containing the codewords passing through branch $\mathcal{B}(\sigma^t, \mathbf{u}_t, \sigma^{t+1}) \in \mathcal{T}$, \mathcal{T} is a set of all branches in the trellis

segment. The likelihood of information symbol $\mathbf{u}_t = \{u_t^0, u_t^1, \dots, u_t^{a-1}\}$ given \mathbf{Y} is

$$\begin{aligned}
& p(\mathbf{u}_t | \mathbf{Y}) \\
&= \frac{1}{p(\mathbf{Y})} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \sum_{\mathbf{c} \in \mathcal{C}(\sigma^t, \mathbf{c}_t, \mathbf{u}_t, \sigma^{t+1})} p(\mathbf{Y}, \mathbf{c}) \\
&= \frac{1}{p(\mathbf{Y})} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \sum_{\mathbf{c} \in \mathcal{C}(\sigma^t, \mathbf{c}_t, \mathbf{u}_t, \sigma^{t+1})} p(\mathbf{Y}_{[0,t]}, \mathbf{c}_{[0,t]}, \sigma^t) p(\mathbf{y}_t, \mathbf{c}_t, \sigma^{t+1} | \sigma^t, \mathbf{Y}_{[0,t]}, \mathbf{c}_{[0,t]}) \\
&\quad p(\mathbf{Y}_{[t+1,L]}, \mathbf{c}_{[t+1,L]} | \sigma^{t+1}, \mathbf{Y}_{[0,t+1]}, \mathbf{c}_{[0,t+1]}) \\
&= \frac{1}{p(\mathbf{Y})} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \sum_{\mathbf{c} \in \mathcal{C}(\sigma^t, \mathbf{c}_t, \mathbf{u}_t, \sigma^{t+1})} p(\mathbf{Y}_{[0,t]}, \mathbf{c}_{[0,t]}, \sigma^t) p(\mathbf{y}_t, \mathbf{c}_t, \sigma^{t+1} | \sigma^t) p(\mathbf{Y}_{[t+1,L]}, \mathbf{c}_{[t+1,L]} | \sigma^{t+1}) \\
&= \frac{1}{p(\mathbf{Y})} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \sum_{\mathbf{c} \in \mathcal{C}(\sigma^t, \mathbf{c}_t, \mathbf{u}_t, \sigma^{t+1})} p(\mathbf{Y}_{[0,t]}, \mathbf{c}_{[0,t]}, \sigma^t) p(\mathbf{y}_t | \mathbf{c}_t, \sigma^{t+1}, \sigma^t) p(\mathbf{c}_t, \sigma^{t+1} | \sigma^t) \\
&\quad p(\mathbf{Y}_{[t+1,L]}, \mathbf{c}_{[t+1,L]} | \sigma^{t+1}) \\
&= \frac{1}{p(\mathbf{Y})} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} p(\mathbf{Y}_{[0,t]}, \sigma^t) p(\mathbf{y}_t | \mathbf{c}_t) p(\mathbf{u}_t, \sigma^{t+1} | \sigma^t) p(\mathbf{Y}_{[t+1,L]} | \sigma^{t+1}) \\
&= \frac{1}{p(\mathbf{Y})} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) \gamma(\mathbf{u}_t) \beta(\sigma^{t+1}), \tag{2.20}
\end{aligned}$$

where $\mathbf{Y}_{[e,f]} = \{\mathbf{y}_e, \mathbf{y}_{e+1}, \dots, \mathbf{y}_{f-1}\}$, $\mathbf{c}_{[e,f]} = \{\mathbf{c}_e, \mathbf{c}_{e+1}, \dots, \mathbf{c}_{f-1}\}$, $\alpha(\sigma^t) = p(\mathbf{Y}_{[0,t]}, \sigma^t)$, $\gamma(\mathbf{u}_t) = p(\mathbf{y}_t | \mathbf{c}_t) p(\mathbf{u}_t, \sigma^{t+1} | \sigma^t)$ and $\beta(\sigma^{t+1}) = p(\mathbf{Y}_{[t+1,L]} | \sigma^{t+1})$. The likelihood function $p(\mathbf{u}_t, \sigma^{t+1} | \sigma^t)$ can be further decomposed as

$$p(\mathbf{u}_t, \sigma^{t+1} | \sigma^t) = p(\mathbf{u}_t) \delta(\mathbf{u}_t, \sigma^{t+1} | \sigma^t), \tag{2.21}$$

where $p(\mathbf{u}_t)$ is the a priori likelihood and $\delta(\mathbf{u}_t, \sigma^{t+1} | \sigma^t)$ is an indicator function.

$$\delta(\mathbf{u}_t, \sigma^{t+1} | \sigma^t) = \begin{cases} 1 & , \mathcal{B}(\sigma^t, \mathbf{u}_t, \sigma^{t+1}) \in \mathcal{T} \\ 0 & , \text{otherwise} \end{cases}. \tag{2.22}$$

$\alpha(\sigma^t)$ and $\beta(\sigma^t)$ can be recursively calculated and eqns. (2.23) and (2.24) show the

forward and backward recursions.

$$\begin{aligned}\alpha(\sigma^t) &= \sum_{\sigma^{t-1}} p(\mathbf{Y}_{[0,t]}, \sigma^{t-1}, \sigma^t) = \sum_{\sigma^{t-1}} p(\mathbf{y}_{t-1}, \sigma^t | \mathbf{Y}_{[0,t-1]}, \sigma^{t-1}) p(\mathbf{Y}_{[0,t-1]}, \sigma^{t-1}) \\ &= \sum_{\sigma^{t-1}} p(\mathbf{y}_{t-1}, \mathbf{c}_{t-1}, \sigma^t | \sigma^{t-1}) p(\mathbf{Y}_{[0,t-1]}, \sigma^{t-1}) = \sum_{\sigma^{t-1}} \alpha(\sigma^{t-1}) \gamma(\mathbf{u}_{t-1})\end{aligned}\quad (2.23)$$

$$\begin{aligned}\beta(\sigma^t) &= \sum_{\sigma^{t+1}} p(\mathbf{Y}_{[t,L]}, \sigma^{t+1} | \sigma^t) = \sum_{\sigma^{t+1}} p(\mathbf{Y}_{[t+1,L]} | \mathbf{y}_t, \sigma^{t+1}, \sigma^t) p(\mathbf{y}_t, \sigma^{t+1} | \sigma^t) \\ &= \sum_{\sigma^{t+1}} p(\mathbf{Y}_{[t+1,L]} | \sigma^{t+1}) p(\mathbf{y}_t, \mathbf{c}_t, \sigma^{t+1} | \sigma^t) = \sum_{\sigma^{t+1}} \beta(\sigma^{t+1}) \gamma(\mathbf{u}_t)\end{aligned}\quad (2.24)$$

In summary, given initial condition $\alpha(\sigma^0)$ and $\beta(\sigma^L)$, the MAP algorithm calculates $\alpha(\sigma^t)$ and $\beta(\sigma^t)$. Then the MAP algorithm computes the likelihood in eqn. (2.20) and makes decisions.

The initial condition $\alpha(\sigma^0)$ and $\beta(\sigma^L)$ varies with the termination of convolutional code and decoding algorithms. The tail-padding terminates the end state to all zero state and the initial condition is set to $\alpha(\sigma^0) = \begin{cases} 1, & \sigma^0 = 0 \\ 0, & \text{otherwise} \end{cases}$ and $\beta(\sigma^L) = \begin{cases} 1, & \sigma^L = 0 \\ 0, & \text{otherwise} \end{cases}$.

The tail-biting keeps the initial state and end state the same and the initial condition is $\alpha(\sigma^0) = \beta(\sigma^L) = 1/|\Sigma|$. The optimal MAP algorithm becomes

$$p(\mathbf{u}_t | \mathbf{Y}) = \frac{1}{p(\mathbf{Y})} \sum_{\sigma^0 = \sigma^L \in \Sigma} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) \gamma(\mathbf{u}_t) \beta(\sigma^{t+1}) \quad (2.25)$$

and it requires approximate $|\Sigma|$ times computational complexity than the decoding for convolutional code applying the tail-padding termination. The complexity comes from unknown $\alpha(\sigma^0)$ and $\beta(\sigma^L)$. In order to reduce the complexity, we can extend the forward recursion and backward recursion from $\alpha(\sigma^{L-TL})$ and $\beta(\sigma^{TL})$ to estimate the initial condition $\alpha(\sigma^0)$ and $\beta(\sigma^L)$, where TL is the training length and $\alpha(\sigma^{L-TL}) = \beta(\sigma^{TL}) = 1/|\Sigma|$. It only requires $2TL/L$ more computational complexity. The concept could also be applied to reduce the MAP decoding storage [104] or for parallel MAP decoding.

The decision of an information bit is our next concern. The likelihood of u_t^i is $p(u_t^i | \mathbf{Y}) = \sum_{u_t^j, j \neq i} p(\mathbf{u}_t, u_t^i | \mathbf{Y})$. Therefore we can acquire the decision of the i th bit of an information symbol by $\hat{u}_t^i = \arg \max_{u_t^i} p(u_t^i | \mathbf{Y})$.

The Log-MAP features less implementation complexity and can be acquired by transforming the likelihood into the log-likelihood ratio. The relation is

$$L(u_t^i) = \log \frac{p(u_t^i = 0 | \mathbf{Y})}{p(u_t^i = 1 | \mathbf{Y})}. \quad (2.26)$$

The ratio can be computed on the log-domain and the corresponding computation can be further simplified as

$$\begin{aligned} L(u_t^i) &= \log \frac{\sum_{u_t^j, j \neq i} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) \gamma(\mathbf{u}_t | u_t^i = 0) \beta(\sigma^{t+1})}{\sum_{u_t^j, j \neq i} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) \gamma(\mathbf{u}_t | u_t^i = 1) \beta(\sigma^{t+1})} \\ &= \log \frac{\sum_{u_t^j, j \neq i} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \exp^{\hat{\alpha}(\sigma^t) + \hat{\gamma}(\mathbf{u}_t | u_t^i = 0) + \hat{\beta}(\sigma^{t+1})}}{\sum_{u_t^j, j \neq i} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \exp^{\hat{\alpha}(\sigma^t) + \hat{\gamma}(\mathbf{u}_t | u_t^i = 1) + \hat{\beta}(\sigma^{t+1})}}, \end{aligned} \quad (2.27)$$

where $\hat{\alpha}(\sigma^t) = \log \alpha(\sigma^t)$, $\hat{\beta}(\sigma^t) = \log \beta(\sigma^t)$, $\hat{\gamma}(\mathbf{u}_t | u_t^i) = \log \gamma(\mathbf{u}_t | u_t^i)$ and $\gamma(\mathbf{u}_t | u_t^i) = p(\mathbf{y}_t | \mathbf{c}_t) p(\mathbf{u}_t, u_t^i, \sigma^{t+1} | \sigma^t)$. The forward and backward recursions become

$$\hat{\alpha}(\sigma^t) = \log \sum_{\sigma^{t-1}} \exp^{\hat{\alpha}(\sigma^{t-1}) + \hat{\gamma}(\mathbf{u}_{t-1})} \quad (2.28)$$

$$\hat{\beta}(\sigma^t) = \log \sum_{\sigma^{t+1}} \exp^{\hat{\beta}(\sigma^{t+1}) + \hat{\gamma}(\mathbf{u}_t)}. \quad (2.29)$$

[104] proposed the following function

$$\max^*(A, B) = \log (\exp^A + \exp^B) = \max(A, B) + \log (1 + \exp^{-|A-B|}) \quad (2.30)$$

to deal with the log of the sums of two exponential terms. This function is composed of a maximization and a compensation term $\log (1 + \exp^{-|A-B|})$ which can be implemented by a look-up-table. Although log and exponential are high complexity operations, both operations can be substituted by less complexity operations. If there are three terms, eqn. can be recursively applied as

$$\max^*(A, B, C) = \max^*(A, \max^*(B, C)), \quad (2.31)$$

and the log-sum on multiple terms is capable. Therefore the Log-MAP algorithm transfers the multiplication and addition on the real-domain into addition, maximization and compensation on the log-domain.

We describe two kinds of simplification for the Log-MAP algorithm: MAX Log-MAP and linear Log-MAP [102]. The simplification is related to the compensation term in eqn. (2.2.5). The MAX Log-MAP algorithm substitutes the summation in eqns. (2.27)-(2.29) by the maximization and these equations are approximated as

$$\begin{aligned}
L(u_t^i) &\approx \log \frac{\max_{u_t^j, j \neq i} \max_{\sigma^t, \sigma^{t+1} \in \Sigma} \exp^{\hat{\alpha}(\sigma^t) + \hat{\gamma}(\mathbf{u}_t | u_t^i = 0) + \hat{\beta}(\sigma^{t+1})}}{\max_{u_t^j, j \neq i} \max_{\sigma^t, \sigma^{t+1} \in \Sigma} \exp^{\hat{\alpha}(\sigma^t) + \hat{\gamma}(\mathbf{u}_t | u_t^i = 1) + \hat{\beta}(\sigma^{t+1})}} \\
&= \max_{u_t^j, j \neq i} \max_{\sigma^t, \sigma^{t+1} \in \Sigma} \hat{\alpha}(\sigma^t) + \hat{\gamma}(\mathbf{u}_t | u_t^i = 0) + \hat{\beta}(\sigma^{t+1}) \\
&\quad - \max_{u_t^j, j \neq i} \max_{\sigma^t, \sigma^{t+1} \in \Sigma} \hat{\alpha}(\sigma^t) + \hat{\gamma}(\mathbf{u}_t | u_t^i = 1) + \hat{\beta}(\sigma^{t+1}) \tag{2.32}
\end{aligned}$$

$$\hat{\alpha}(\sigma^t) \approx \log \max_{\sigma^{t-1}} \exp^{\hat{\alpha}(\sigma^{t-1}) + \hat{\gamma}(\mathbf{u}_{t-1})} = \max_{\sigma^{t-1}} \hat{\alpha}(\sigma^{t-1}) + \hat{\gamma}(\mathbf{u}_{t-1}) \tag{2.33}$$

$$\hat{\beta}(\sigma^t) \approx \log \max_{\sigma^{t+1}} \exp^{\hat{\beta}(\sigma^{t+1}) + \hat{\gamma}(\mathbf{u}_t)} = \max_{\sigma^{t+1}} \hat{\beta}(\sigma^{t+1}) + \hat{\gamma}(\mathbf{u}_t). \tag{2.34}$$

It is equivalent to remove the compensation term $\log(1 + \exp^{-|A-B|})$ in eqn. (2.2.5). The linear Log-MAP applies a linear function $f(x) = \begin{cases} a(x-b) & , 0 \leq x < b \\ 0 & , x \geq b \end{cases}$ to approximate the compensation term and requires less complexity without the performance loss comparing to Log-MAP. Two parameters (a, b) are subject to the minimization of $\int_{x=0}^{\infty} (f(x) - \log(1 + \exp^{-x}))^2 dx$ and the optimized value of a and b are -0.24904 and 2.5068 .

2.3 Turbo code

Turbo code [13] provides outstanding performance by low complexity iterative decoding algorithm but features a simple parallel concatenated structure. The performance comes from the few low weight codewords comparing to the convolutional code and this characteristic enhances error rate performance at low SNR. However the performance loses to convolutional code with the large free distance at high SNR. The low weight codeword dominates the performance at high SNR and causes the error floor. The interleaver determines the error floor and an interleaver and the joint design between an interleaver and component code improves the error floor shape. The interleaver design

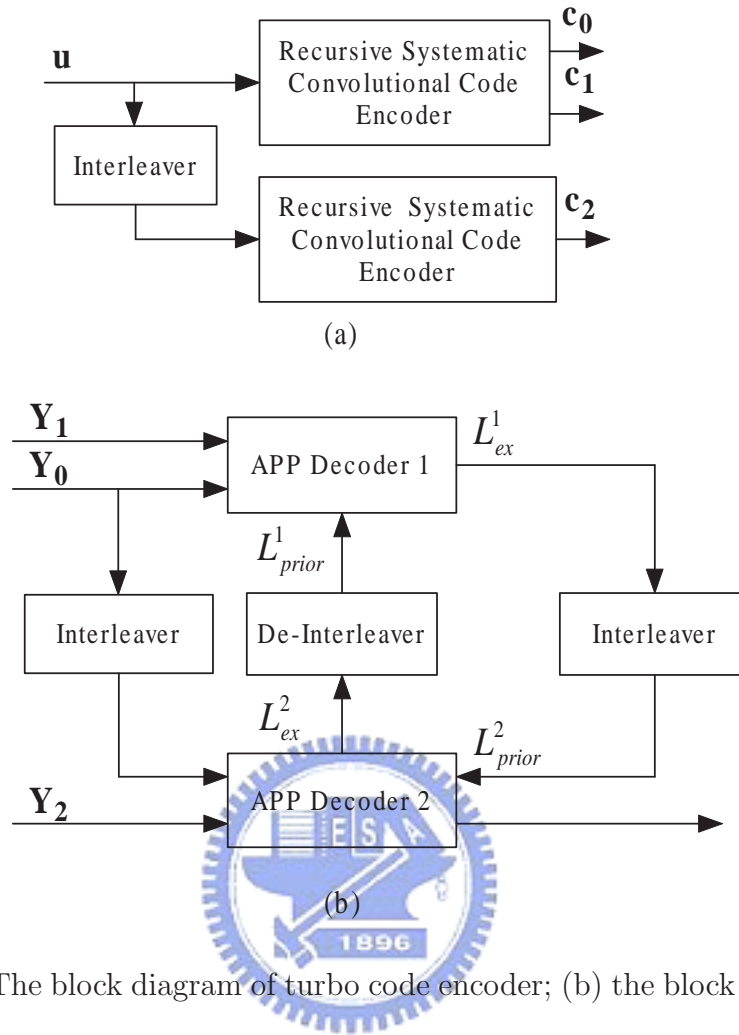


Figure 2.4: (a) The block diagram of turbo code encoder; (b) the block diagram of turbo code decoder.

rules will be provided in chapter 3, 4 and 5. The following subsections will describe the encoding and decoding methodologies.

2.3.1 Encoder

Turbo code encoder, shown in 2.4 (a), composes of two recursive systematic convolutional codes and an interleaver. The interleaver permutes information sequence to reduce the possibility of generating low weight codeword by both convolutional codes and enlarge the minimum weight of codewords. Two convolutional codes encode information sequence and the interleaved information sequence respectively. Due to the sophisticated systematic encoding, the systematic part \mathbf{c}_0 generated from the upper

convolutional code can be shared to the lower convolutional code. Therefore the turbo encoder applies the systematic part \mathbf{c}_0 and parity parts $\mathbf{c}_1, \mathbf{c}_2$ as a codeword.

2.3.2 Decoder

Turbo code decoder applies iterative decoding and Fig. 2.4 (b) shows the block diagram. The decoder possesses two a posteriori probability (APP) decoders which manipulate the received samples and the a priori information to generate the extrinsic information and estimate the likelihood. The extrinsic information is interleaved or de-interleaved as the a priori information for the successive APP decoder. The estimated likelihood is used for decision making as iterative decoding reaches the maximum decoding round or satisfies the stopping condition.

The decoder can operate information exchange on the bit level and symbol level. The bit level operation can process the information represented by log-likelihood ratio and the necessary storage is less than that represented by likelihood. The symbol level operation keeps the correlation of a symbol. If a symbol level interleaver is applied, the symbol level operation converges faster and suits for high rate systems, e.g. duobinary turbo code [37, 38]. In order to keep the correlation of these bits, the decoder requires more storage for the symbol level operation. Both algorithms are detailed in the following.

Binary iterative decoding algorithm

Due to the systematic encoding, we assume $c_t^i = u_t^i$ for $0 \leq i < a$. We decompose the eqn. (2.27) into three terms: the channel value, the a priori information and the

extrinsic information. The decomposition is shown as follows.

$$\begin{aligned}
& L(u_t^i) \\
&= \log \frac{\sum_{u_t^i, j \neq i} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) \gamma(\mathbf{u}_t | u_t^i = 0) \beta(\sigma^{t+1})}{\sum_{u_t^i, j \neq i} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) \gamma(\mathbf{u}_t | u_t^i = 1) \beta(\sigma^{t+1})} \\
&= \log \frac{\sum_{u_t^i=0, u_t^j, j \neq i} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) p(\mathbf{y}_t | \mathbf{c}_t) p(\mathbf{u}_t) \delta(\mathbf{u}_t, \sigma^{t+1} | \sigma^t) \beta(\sigma^{t+1})}{\sum_{u_t^i=1, u_t^j, j \neq i} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) p(\mathbf{y}_t | \mathbf{c}_t) p(\mathbf{u}_t) \delta(\mathbf{u}_t, \sigma^{t+1} | \sigma^t) \beta(\sigma^{t+1})} \\
&= \log \frac{p(y_t^i | c_t^i = 0)}{p(y_t^i | c_t^i = 1)} + \log \frac{p(u_t^i = 0)}{p(u_t^i = 1)} \\
&+ \log \frac{\sum_{u_t^i=0, u_t^j, j \neq i} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) \delta(\mathbf{u}_t, \sigma^{t+1} | \sigma^t) \beta(\sigma^{t+1}) \prod_{0 \leq j < b, j \neq i} p(y_t^j | c_t^j) \prod_{0 \leq j < a, j \neq i} p(u_t^j)}{\sum_{u_t^i=1, u_t^j, j \neq i} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) \delta(\mathbf{u}_t, \sigma^{t+1} | \sigma^t) \beta(\sigma^{t+1}) \prod_{0 \leq j < b, j \neq i} p(y_t^j | c_t^j) \prod_{0 \leq j < a, j \neq i} p(u_t^j)} \\
&= L_c(u_t^j) + L_{\text{prior}}(u_t^j) + L_{\text{ex}}(u_t^j), \tag{2.35}
\end{aligned}$$

where $L_c(u_t^j) = \log \frac{p(y_t^j | c_t^j = 0)}{p(y_t^j | c_t^j = 1)}$ and $L_{\text{prior}}(u_t^j) = \log \frac{p(u_t^j = 0)}{p(u_t^j = 1)}$. Then the APP decoder simply generates $L_{\text{ex}}(u_t^j)$ by

$$L_{\text{ex}}(u_t^j) = L(u_t^j) - L_c(u_t^j) - L_{\text{prior}}(u_t^j). \tag{2.36}$$

Non-binary iterative decoding algorithm

The decomposition is derived from the eqn. (2.20).

$$\begin{aligned}
& p(\mathbf{u}_t | \mathbf{Y}) \\
&= \frac{1}{p(\mathbf{Y})} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) \gamma(\mathbf{u}_t) \beta(\sigma^{t+1}) \\
&= \frac{p(\mathbf{u}_t) p(\mathbf{y}_t^{[0,a]} | \mathbf{c}_t^{[0,a]})}{p(\mathbf{Y})} \sum_{\sigma^t, \sigma^{t+1} \in \Sigma} \alpha(\sigma^t) p(\mathbf{y}_t^{[a,b]} | \mathbf{c}_t^{[a,b]}) \delta(\mathbf{u}_t, \sigma^{t+1} | \sigma^t) \beta(\sigma^{t+1}) \\
&= p(\mathbf{u}_t) p(\mathbf{y}_t^{[0,a]} | \mathbf{c}_t^{[0,a]}) p_{\text{ex}}(\mathbf{u}_t), \tag{2.37}
\end{aligned}$$

where $\mathbf{y}_t^{[i,j]} = \{y_t^i, y_t^{i+1}, \dots, y_t^{j-1}\}$, $\mathbf{c}_t^{[i,j]} = \{c_t^i, c_t^{i+1}, \dots, c_t^{j-1}\}$ and $\mathbf{c}_t^{[0,a]} = \mathbf{u}_t$ due to the systematic encoding. The extrinsic information $p_{\text{ex}}(\mathbf{u}_t)$ is acquired by

$$p_{\text{ex}}(\mathbf{u}_t) = \frac{p(\mathbf{u}_t | \mathbf{Y})}{p(\mathbf{u}_t) p(\mathbf{y}_t^{[0,a]} | \mathbf{c}_t^{[0,a]})}. \tag{2.38}$$

Decoding on the log-domain requires less complexity. We take logarithm on eqn. (2.38) and obtain

$$L_{ex,S}(\mathbf{u}_t) = L_S(\mathbf{u}_t) - L_{c,S}(\mathbf{u}_t) - L_{prior,S}(\mathbf{u}_t), \quad (2.39)$$

where $L_S(\mathbf{u}_t) = \log p(\mathbf{u}_t|\mathbf{Y})$, $L_{c,S}(\mathbf{u}_t) = \log p(\mathbf{y}_t^{[0,a]}|\mathbf{c}_t^{[0,a]})$ and $L_{prior,S}(\mathbf{u}_t) = \log p(\mathbf{u}_t)$. The log-likelihood $L_S(\mathbf{u}_t)$ is acquired by the log-domain calculation, and the forward and backward recursions refer to eqns. (2.28) and (2.29). Multiplication and addition are replaced by the addition, maximization and compensation, and log-domain computation requires less complexity.

We can further subtract $\log P_{ex}(\mathbf{u}_t)$ by $\log P_{ex}(\mathbf{u}_t = 0)$ and the eqn. (2.39) becomes

$$\begin{aligned} & \log \frac{P_{ex}(\mathbf{u}_t)}{P_{ex}(\mathbf{u}_t = 0)} \\ = & \log \frac{p(\mathbf{u}_t|\mathbf{Y})}{p(\mathbf{u}_t = 0|\mathbf{Y})} - \log \frac{p(\mathbf{y}_t^{[0,a]}|\mathbf{c}_t^{[0,a]})}{p(\mathbf{y}_t^{[0,a]}|\mathbf{c}_t^{[0,a]} = 0)} - \log \frac{p(\mathbf{u}_t)}{p(\mathbf{u}_t = 0)}, \end{aligned} \quad (2.40)$$

and the equation is simplified as

$$L_{ex,0}(\mathbf{u}_t) = L_0(\mathbf{u}_t) - L_{c,0}(\mathbf{u}_t) - L_{prior,0}(\mathbf{u}_t), \quad (2.41)$$

where $L_{ex,0}(\mathbf{u}_t) = \log \frac{P_{ex}(\mathbf{u}_t)}{P_{ex}(\mathbf{u}_t=0)}$, $L_0(\mathbf{u}_t) = \log \frac{p(\mathbf{u}_t|\mathbf{Y})}{p(\mathbf{u}_t=0|\mathbf{Y})}$, $L_{c,0}(\mathbf{u}_t) = \log \frac{p(\mathbf{y}_t^{[0,a]}|\mathbf{c}_t^{[0,a]})}{p(\mathbf{y}_t^{[0,a]}|\mathbf{c}_t^{[0,a]}=0)}$ and $L_{prior,0}(\mathbf{u}_t) = \log \frac{p(\mathbf{u}_t)}{p(\mathbf{u}_t=0)}$. Apparently, the subtraction avoids the storage on $L_0(\mathbf{u}_t = 0) = L_{c,0}(\mathbf{u}_t = 0) = L_{prior,0}(\mathbf{u}_t = 0) = L_{ex,0}(\mathbf{u}_t = 0) = 0$.

2.4 Factor graph

Factor graph [60, 42] is one of the many popular graphs to facilitate our understanding on code structure and help us to develop the decoding algorithms. It has been widely used to explain and study low density parity check (LDPC) code [46] and is also extended to explain turbo code, MMSE equalizer and channel estimation etc. The graph encompasses factor nodes and edges. The factor node describes the relation between

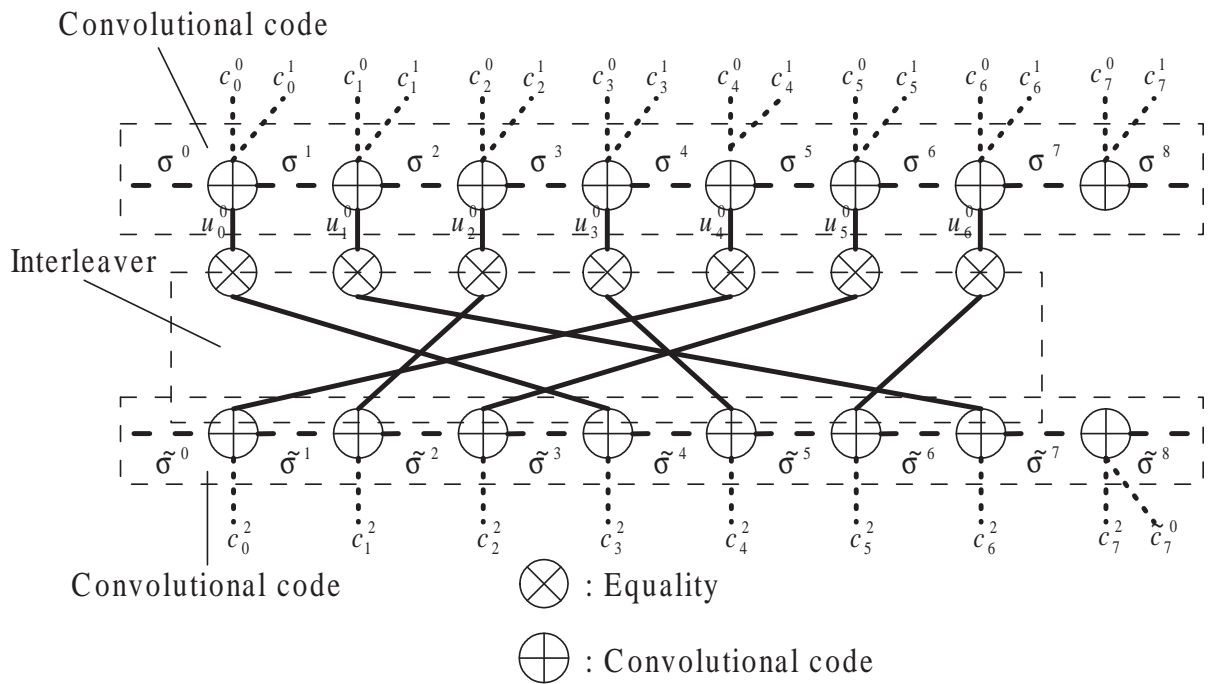


Figure 2.5: An example of a turbo code factor graph.

connected edges, e.g. equality. The edge indicates the variable, e.g. information bit, state. One can easily schedule operations of these factor nodes to decode.

Fig. 2.5 provides a factor graph example for turbo code. The code applies one bit to terminate both convolutional codes. 7 information bits are encoded and 25 code bits are generated; the code rate is 7/25. The graph is composed of two kinds of factor nodes: convolutional code and equality. The code node connects information bit, code bits and states of convolutional code. The equality node connects information bit and interleaved information bit. Both APP decoders generate the extrinsic information and then APP decoders pass the generated extrinsic information through the equality node to the interleaved or de-interleaved coordinates for the other APP decoder respectively. The graph also shows that decoder does not generate and exchange the extrinsic information of the terminating bits. The iterative decoding process is visualized and turbo code structure is understood.

2.5 Convergence analysis

One interesting question is how worse the environment an iterative decoding works. The APP decoder generates the extrinsic information for the successive APP decoder as the a priori information. We hope the extrinsic information or the a priori information increasing after several iterations. However it does not always hold. In order to analyze the convergence behavior, extrinsic information transfer (EXIT) chart and density evolution are introduced. The following subsections describe both methods.

2.5.1 EXIT chart

The EXIT [27, 28] chart, proposed by ten Brink, describes the input and output information change during the iterative process and the convergence behavior is predicted. The mutual information between the a priori information and information bit is the measure for the input information and the mutual information between the extrinsic information and information bit is the measure for the output information. The mutual information generally increases with the iteration but it does not always hold at low SNR. The mutual information of the generated extrinsic information may be the same as that of the a priori information and the performance does not improve anymore. Therefore we can search the minimum SNR among the generated EXIT curves corresponding to various SNRs such that the cross point disappears. It implies that the iterative decoding only works above the searched minimum SNR.

Gaussian distribution and independence are two assumption for both input and output information to facilitate the analysis. The Gaussian distributed a priori information implies a one-to-one mapping between the quantity of the mutual information and the variance of the a priori information. We can easily generate the a priori information source given mutual information for the decoder to generate the extrinsic information. Reversely, we can also estimate the variance of the extrinsic information to obtain the output mutual information. The independent input assures that we can generate the a

priori information source only based on the statistics and the independent output assures that the likelihoods of a priori information to the next APP decoder are independent. However the independence assumption does not always hold and the input information is correlated during the iterative process. Although we may acquire an estimated SNR, the actual minimum SNR is sometimes higher for the correlated source. We take turbo code as an example. The error rate curves are not steep when interleaver length is small due to highly correlated input information. Therefore the method fails as the interleaver is small. Fortunately both assumptions asymptotically holds for a large code and the method is still quite helpful in searching a good code.

We describe the relation between the variance and the mutual information. Assume the Gaussian distributed random variable and information bit random variable are G and U . Note the conditional probability density function of a priori information $G = g$ given the information bit $U = u$ by

$$p_G(g|u) = \frac{1}{\sqrt{2\pi\sigma_G^2}} \exp\left[-\frac{\left(g - \frac{u\sigma_G^2}{2}\right)^2}{2\sigma_G^2}\right], \quad (2.42)$$

and $p_G(g|u) = p_G(-g|u) \exp^{gu}$. The mutual information I_G between the a priori information and information bit is

$$\begin{aligned} I_G &= I(G;U) \\ &= \sum_u \int_{-\infty}^{\infty} p_{G,U}(g,u) \log_2 \frac{p_{G,U}(g,u)}{p_G(g)p_U(u)} dg \\ &= \frac{1}{2} \sum_u \int_{-\infty}^{\infty} p_{G|U}(g|u) \log_2 \frac{p_G(g|u)}{\frac{1}{2}p_G(g|U=0) + \frac{1}{2}p_G(g|U=1)} dg, \end{aligned} \quad (2.43)$$

where $0 \leq I_G \leq 1$. The relation between the variance σ_G and I_G is represented as

$$I_G(\sigma_G) = 1 - \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma_G^2}} \exp\left[-\frac{\left(g - \frac{\sigma_G^2}{2}\right)^2}{2\sigma_G^2}\right] \log_2(1 + \exp^g) dg. \quad (2.44)$$

We can apply eqn. (2.44) for the a priori information and extrinsic information to generate input random variable for decoder and calculate the corresponding output mutual information.

2.5.2 Density evolution

The density evolution [82, 83] describes the probability density function change during the iterative decoding process. The method stops when the density function converges. However the density function is hard to estimate and the histogram is used to record the distribution. For the simplicity Gaussian distribution is assumed to approximate the density function with only two parameters, mean and variance. We calculate the SNR by the mean and variance and the change of the SNR of the extrinsic and a priori information describes the change of the density function. The SNR change can be used to predict the convergence behavior for the EXIT chart.

2.6 High throughput turbo decoder architecture

Enabling multiple APP decoders processing at the same time increases the throughput of turbo decoding, and there are two classes of high throughput turbo decoder architectures: the parallel turbo decoder and the pipeline turbo decoder. The parallel turbo decoder shown in Fig. 2.6 processes multiple sequences corresponding to the same decoding round. A codeword sequence is partitioned into multiple sub-sequences, which are stored in the memory banks, and these APP decoders can apply the sliding-window APP (SWAPP) decoding algorithm [104, 21] to deal with these sub-sequences and generate the extrinsic information. The turbo decoder writes the extrinsic information back to these memory banks according to interleaving or de-interleaving rules.

The pipeline turbo decoder shown in Fig. 2.7 processes continuous APP decoder rounds at the same time to improve the throughput and therefore these APP decoders process codewords corresponding to different blocks with block length L or codewords with information length L . Fig. 2.7 (a) is a decoding module for one iteration. \mathbf{Y}_k^0 , \mathbf{Y}_k^1 , \mathbf{Y}_k^2 correspond to the k th block or codeword; $\mathbf{L}_{app}(\mathbf{u}_k)$ and $\mathbf{L}_{ex}(\mathbf{u}_k)$ correspond to the a priori information and the generated extrinsic information corresponding to the k th block or codeword. These modules can be serially concatenated to have the pipeline

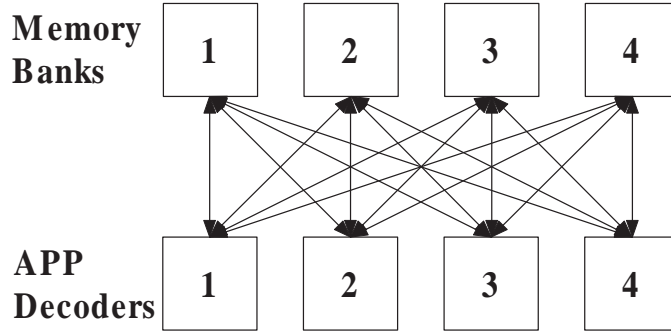


Figure 2.6: The parallel turbo decoder architecture.

turbo decoder shown in Fig. 2.7 (b).

2.7 Notations

2.7.1 Definitions

Definition 4 $\lfloor x \rfloor$ is the maximum integer small or equal to x .

Definition 5 $\lceil x \rceil$ is the minimum integer larger or equal to x .

Definition 6 $|i|_M = i \bmod M$, where i and M are positive integers.

Definition 7 $|i|^M = \begin{cases} i, & |i|_M = 0 \\ |i|_M, & |i|_M \neq 0 \end{cases}$, where i is a non-negative integer and M is a positive integer.

Definition 8 $\|i\|_M = \lfloor \frac{i}{M} \rfloor$, where i is a non-negative integer and M is a positive integer.

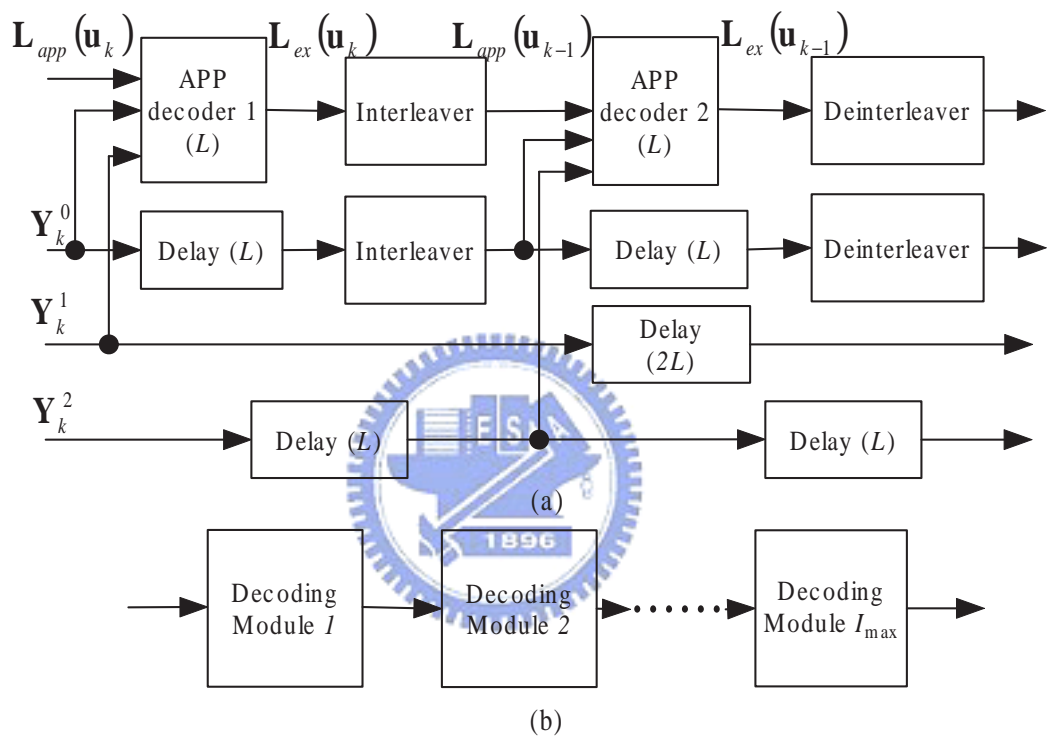


Figure 2.7: a) The block diagram of a decoding module for one iteration; (b) The block diagram of the pipeline decoder.

Chapter 3

Inter-block permutation interleaver

Inter-block permutation (IBP) interleaver features a simple interleaver structure and one can acquire the interleaver with good distance properties for turbo codes. A general IBP interleaver encompasses many existing interleavers as special subclasses. It is built upon smaller interleavers and a re-permutation is applied on these interleavers to construct a larger interleaver. By using a suitable IBP rule, an IBP turbo code (IBPTC) can possess good distance properties. It is therefore reasonable to conjecture that the distance spectrum of an IBPTC applying separate encoding would offer some desired properties.

Constructing an interleaver based on any existing block interleavers to render the better distance properties of an IBPTC is our first purpose. The periodic and invariant permutations are the construction rules. Both rules reduce long length interleaver searching effort and avoid low weight codeword events. If we have to design 5000 kinds of interleavers, e.g. 3GPP turbo code interleaver [1], the proposed rule is useful. However, both rules are not strong enough to guarantee the distance properties and we provide some loose constraints for the block interleavers to achieve the derived distance bounds.

Providing the joint design between an inter-block permutation and a block interleaver based on separate tail-biting encoding is the second purpose. We derive a general lower bound for codeword associated with the weight-2 input sequences for the IBP-interleaved turbo codes. By analyzing the effects of selected particular system parameters on this

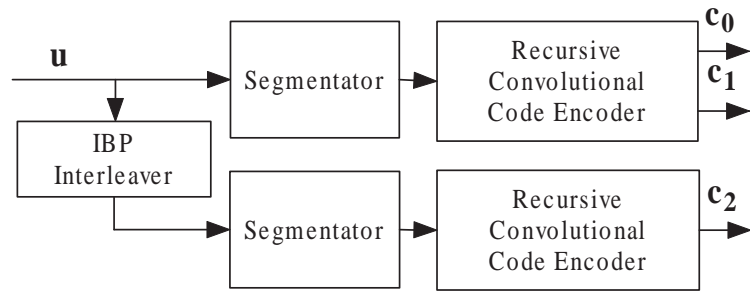


Figure 3.1: The block diagram of inter-block permutation turbo code encoder.

general bound we obtain some useful design guidelines. We apply a simplified partition rule presented in [25] and a regular permutation function to derive the bound. We also examine some special cases and evaluate the codeword lower bounds of the weight-2 input sequences.

3.1 Inter-block permutation turbo code

Fig. 3.1 shows the block diagram of an IBPTC encoder. The main difference is that the encoder includes extra two segmentators which partition information sequence and the interleaved information sequences into multiple short blocks in accordance with the block length of an IBP interleaver. The recursive systematic convolutional (RSC) code encodes these blocks and can apply these three termination methods, namely continuous [11], tail-padding and tail-biting [106]. In accordance with these options, we define a *continuous* IBPTC (C-IBPTC) as one that encodes each data block using the end state of the previous data block as the initial state and adds the tail-bits only for the last data block. On the other hand, a *discontinuous* IBPTC (D-IBPTC) encodes each data block individually, either by appending the tail-bits at the end of a block or by using the tail-biting encoding. We refer to the former class as the tail-padding IBPTC (TP-IBPTC) while the latter class as the tail-biting IBPTC (TB-IBPTC).

3.2 Inter-block permutation interleaver

Let $\mathbf{u} = \{\mathbf{u}_i\}_{i=0}^{N-1}$ and $\mathbf{u}_i = \{u_{i,k}\}_{k=0}^{L-1} = \{u_k\}_{k=iL}^{(i+1)L-1}$ be an input information sequence composed of N blocks and the i th block with L symbols respectively, i.e. the information length is NL symbols. Denote by Π a permutation function that maps \mathbf{u} into \mathbf{u}' such that $u_{i,k} = u'_{\pi(i,k)}$ or $u_k = u'_{\pi(k)}$.

Coordinates of an input information sequence are represented in one-dimension and two-dimension to simplify our discussion in distance property. Two-dimension representation indicates the ordinal of a block and the ordinal within a block. IBP interleaver is composed of an inter-block permutation and block interleaver. An inter-block permutation and block interleaver determine the first and second ordinals and the concept of the interleaver construction can be demonstrated easier. Moreover if we only discuss the behavior within a block, we can discuss the permutation function of the second ordinal. When we derive the distance properties of a complete codeword, all coordinates have to be considered and one-dimension representation simplifies the notations in this case. Therefore both representations are used in this literature. The transformation between both representations are

$$f_{21}(i, j) = iL + j \quad (3.1)$$

$$f_{12}(i) = (||i||_L, |i|_L), \quad (3.2)$$

where $i, j \in \mathbb{Z}^+$, $||i||_L = \lfloor \frac{i}{L} \rfloor$ and $|i|_L = i \bmod L$ defined in definitions 6 and 8.

Inter-block permutation interleaver $\Pi_{ibp} = \Pi_{inter} \circ \Pi_{intra}$ shown in Fig. 3.2 (a) is composed of two permutation functions Π_{inter} and Π_{intra} , where Π_{inter} and Π_{intra} are inter-block and intra-block permutations. Π_{inter} maps \mathbf{u} into \mathbf{u}' such that $u_{i,k} = u'_{\pi_{inter}(i,k)}$. Π_{intra} is composed of N block interleavers $\Pi_{intra,i}$, $0 \leq i < N$, where $\Pi_{intra,i}$ maps \mathbf{u}_i into \mathbf{u}'_i such that $u_{i,k} = u'_{i,\pi_{intra,i}(k)}$. Π_{intra}^{-1} and Π_{inter}^{-1} denote the inverse permutation functions of Π_{intra} and Π_{inter} respectively. Therefore Π_{ibp} maps \mathbf{u} into \mathbf{u}' such that $u_{i,k} = u'_{\pi_{inter}(i,\pi_{intra,i}(k))}$.

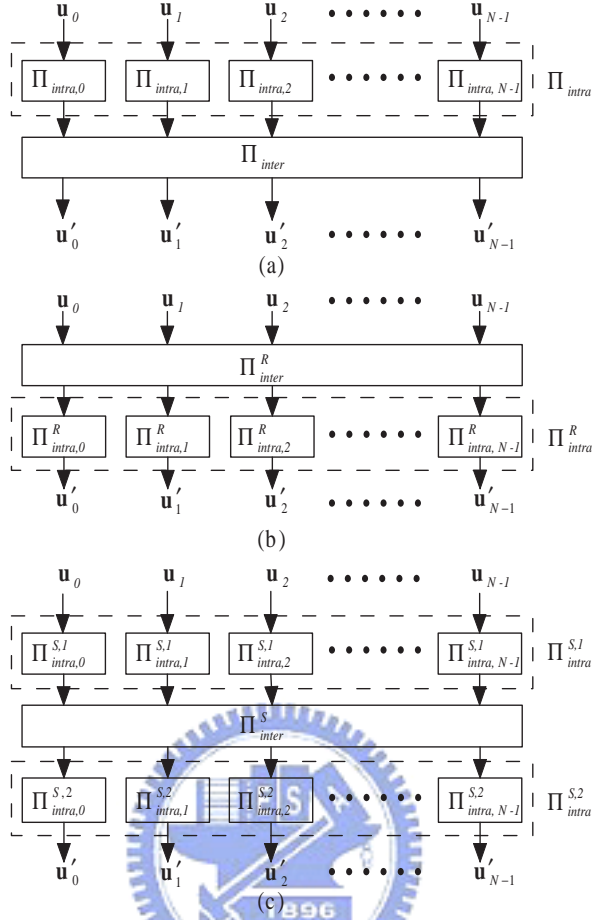


Figure 3.2: (a) Inter-block permutation interleaver; (b) Reversed inter-block permutation interleaver (c) Sandwich inter-block permutation interleaver.

Π_{inter} and Π_{inter}^{-1} are characterized by $f_n(i, j)$, $f_b(j)$ and $f_n^d(i, j)$, $f_b^d(j)$ respectively, where $f_b(j)$ and $f_b^d(j)$ are length- L permutation functions and $f_n(i, j)$ and $f_n^d(i, j)$ are length- N permutation functions corresponding to each j . The permutation rules of Π_{inter} and Π_{inter}^{-1} are

$$\pi_{inter}(i, j) = (f_n(i, j), f_b(j)) \quad (3.3)$$

$$\pi_{inter}^{-1}(i, j) = (f_n^d(i, j), f_b^d(j)), \quad (3.4)$$

or

$$\pi_{inter}(i) = f_n(|i|_L, |i|_L) \cdot L + f_b(|i|_L) \quad (3.5)$$

$$\pi_{inter}^{-1}(i) = f_n^d(|i|_L, |i|_L) \cdot L + f_b^d(|i|_L). \quad (3.6)$$

Our discussion in this thesis mainly focuses on these four functions. We will demonstrate that $f_b(j) = f_b^d(j) = j$ is a good permutation to retain the property of Π_{intra} . Periodic $f_n(i, j)$ and $f_n^d(i, j)$ eliminate the most important low weight codeword error event. These types of Π_{inter} are defined in the following.

Definition 9 *If $f_b(j) = f_b^d(j) = j, \forall j$, then the corresponding inter-block permutation is called a Type I (inter-block) permutation.*

Definition 10 *An inter-block permutation is a Type II (inter-block) permutation if $f_n(i, j) = f_n(i, j + nT_s), \forall i, n$ and $0 \leq j, j + nT_s < L$, where the integer-valued function $f_n(i, j)$ is injective within a period T_s .*

Definition 11 *An inter-block permutation is a Type III (inter-block) permutation if $f_n^d(i, j) = f_n^d(i, j + nT_s), \forall i, n$ and $0 \leq j, j + nT_s < L$, where the integer-valued function $f_n^d(i, j)$ is injective within a period T_s .*

Definition 12 *An inter-block permutation that possesses all the properties of the Types I, II, III (inter-block) permutations is a Type IV (inter-block) permutation.*

Definition 13 *An interleaver is a swap interleaver if $\forall i, \pi(i) = j \Rightarrow \pi(j) = i$, i.e. $\forall i, \pi(i) = \pi^{-1}(i)$.*

Reversed IBP interleaver Π_{ibp}^R and sandwich IBP interleaver Π_{ibp}^S are two variants of IBP interleaver, and there exists an IBP interleaver identical to reversed or sandwich IBP interleaver. Reversed IBP interleaver is constructed in the reversed manner $\Pi_{ibp}^R = \Pi_{intra}^R \circ \Pi_{inter}^R$ shown in Fig. 3.2 (b). When a turbo code applies this interleaver, there

exists a turbo code applying Π_{ibp} with $\Pi_{ibp}^{-1} = \Pi_{inter}^{-1} \circ \Pi_{intra}^{-1}$ where $\Pi_{inter}^{-1} = \Pi_{inter}^R$ and $\Pi_{intra}^{-1} = \Pi_{intra}^R$. Therefore both turbo codes have the same distance spectrum. Sandwich IBP interleaver is constructed by $\Pi_{ibp}^S = \Pi_{intra}^{S,2} \circ \Pi_{inter}^S \circ \Pi_{intra}^{S,1}$ shown in Fig. 3.2 (c). Because $\Pi_{intra}^{S,2}$ only permutes symbols in each block to the same block, there exists a $\Pi'_{inter} = \Pi_{intra}^{S,2} \circ \Pi_{inter}^S$ and $\Pi_{ibp}^S = \Pi'_{inter} \circ \Pi_{intra}^{S,1}$ which is identical to an IBP interleaver. The properties of Π_{ibp}^R and Π_{ibp}^S are equivalent to that of Π_{ibp} and the discussion of Π_{ibp}^R and Π_{ibp}^S are omitted.

3.3 IBP properties

This section discusses the properties of an inter-block permutation Π_{inter} while an intra-block permutation Π_{intra} is unknown, where $\Pi_{intra,i} = \Pi_{block} \forall i$. Denote by $\mathbf{C} = \{\mathbf{c}^0, \mathbf{c}^1, \mathbf{c}^2\}$ a turbo code codeword associated with an input information sequence \mathbf{u} , where \mathbf{c}^j is the output parity-bit sequence of the j th component code while $\mathbf{c}^0 = \mathbf{u}$ represents both the input sequence and the systematic (uncoded) output sequences. A sequence \mathbf{c}^j also can be partitioned into N blocks corresponding to an input sequence \mathbf{u} and the corresponding sequence is $\mathbf{c}^j = \{\mathbf{c}_0^j, \mathbf{c}_1^j, \dots, \mathbf{c}_{N-1}^j\}$. These sequences are not necessary with the same length and various lengths do not influence our results.

Define two equivalent relations “ \sim ” and “ \cong ” on the set of integers \mathbb{Z} by

$$|i - j|_{T_c} = 0 \iff i \sim j$$

$$\|i\|_L = \|j\|_L \iff i \cong j$$

where $i, j \in \mathbb{Z}$, T_c is the period (to be defined later) of an RSC code. Clearly, $i \approx j$ or $i \not\approx j$ means i is not equivalent to j in either sense. The first relation $i \sim j$ indicates (i, j) pair causing a finite weight codeword corresponding to the RSC code with the period T_c , and distance property discussion mainly focuses on this kind of finite weight codeword. The second relation $i \cong j$ indicates that (i, j) pair is in the same block and simplifies our discussion.

3.3.1 First property: invariant permutation

The following theorem specifies the conditions under which the free distance of a C-IBPTC will be greater than or at least equal to that of its corresponding TC applying Π_{block} as its interleaver.

Theorem 3.1 *For a classic TC \mathbf{C}_{TC} applying a Π_{block} as its interleaver, the corresponding C-IBPTC \mathbf{C}_{ibp} has a free distance greater than or equal to that of \mathbf{C}_{TC} if the Type I permutation is used and all sequences $\{\bar{\mathbf{c}}_i^j, j = 0, 1, 2, i = 0, 1, \dots, N - 1\}$ of a minimum weight codeword, $\mathbf{C}_{min} = \{\bar{\mathbf{c}}^0, \bar{\mathbf{c}}^1, \bar{\mathbf{c}}^2\}$, of \mathbf{C}_{ibp} are also valid codewords of the corresponding component codes, where $\bar{\mathbf{c}}_i^j = \{\bar{c}_{i,k}^j, 0 \leq k < L\}$.*

Proof: For a C-IBPTC \mathbf{C}_{ibp} , there exists at least a finite-weight data sequence $\mathbf{u}_{min} = \bar{\mathbf{c}}^0$ whose corresponding codeword has the minimum weight. Suppose the nonzero elements of \mathbf{u}_{min} are at positions $\{(i_1, k_1), (i_2, k_2), \dots, (i_n, k_n)\}$ and the corresponding codeword is $\bar{\mathbf{C}}_{min}$. Denote by $\tilde{\mathbf{c}}^0$ the IBP-interleaved version of $\bar{\mathbf{c}}^0$, where $\{\tilde{\mathbf{c}}_i^j, j = 0, 1, 2, i = 0, 1, \dots, N - 1\}$ and $\tilde{\mathbf{c}}_i^j = \{\tilde{c}_{i,k}^j, 0 \leq k < L\}$. The $\bar{\mathbf{c}}_j^0$ and $\tilde{\mathbf{c}}_j^0$ generate, for the two component codes, the encoded parity-bit sequences, $\bar{\mathbf{c}}_j^1$ and $\bar{\mathbf{c}}_j^2$ with Hamming weights $w_t(\bar{\mathbf{c}}_j^1)$ and $w_t(\bar{\mathbf{c}}_j^2)$, respectively. The systematic parts of both component codes are the same and the corresponding block are denoted by $\bar{\mathbf{c}}_j^0$. Let M_l be the permutation defined on the space of all blocks that moves a block to the l th block, i.e., $M_l : \mathbf{c}_k^i \rightarrow \mathbf{c}_l^i, \forall k$. If $f_b(j) = j$ and $\Pi_{intra,i} = \Pi_{block}$, shifting each position (n_i, k_i) to the l th block by M_l gives

$$M_l(\tilde{c}_{\pi_{ibp}(n_i, k_i)}) = M_l(\tilde{c}_{f_n(n_i, k_i), f_b(\pi_{block}(k_i))}) = \tilde{c}_{l, \pi_{block}(k_i)}. \quad (3.7)$$

As the 2-tuple $(\bar{\mathbf{c}}_j^0, \bar{\mathbf{c}}_j^1)$ is a valid codeword of the first component code of \mathbf{C}_{ibp} according to our assumption, $(\bigoplus_j M_l(\bar{\mathbf{c}}_j^0), \bigoplus_j M_l(\bar{\mathbf{c}}_j^1))$, \bigoplus denoting addition of binary vectors, is also a codeword of the same component code. (3.7) implies that $(\bigoplus_j M_l(\tilde{\mathbf{c}}_j^0), \bigoplus_j M_l(\bar{\mathbf{c}}_j^2))$ are valid codewords for the second component code of \mathbf{C}_{ibp} and \mathbf{C}_{TC} since the additional IBP does not change the relative positions of input bits within a block. The inequality

$$w_t(\mathbf{c}_j^i) + w_t(\mathbf{c}_l^i) = w_t(M_l(\mathbf{c}_j^i)) + w_t(M_l(\mathbf{c}_l^i)) \geq w_t(M_l(\mathbf{c}_j^i) \oplus M_l(\mathbf{c}_l^i)), \quad \forall j \neq l \quad (3.8)$$

then implies that the free distance of \mathbf{C}_{ibp} , $d_{free}(\mathbf{C}_{ibp})$ satisfies

$$d_{free}(\mathbf{C}_{ibp}) = \sum_i \sum_j w_t(\bar{\mathbf{c}}_j^i) \geq \sum_i w_t\left(\bigoplus_j M_l(\bar{\mathbf{c}}_j^i)\right) \geq d_{free}(\mathbf{C}_{block}) \quad (3.9)$$

■

For a D-IBPTC, the sub-codewords $\bar{\mathbf{c}}_i^j$ associated with each input block automatically satisfy the requirement on all blocks. Since both tail-padding and tail-biting convolutional codes are linear codes, we have

Corollary 3.1 *For a classic TC \mathbf{C}_{TC} , the corresponding D-IBPTC \mathbf{C}_{ibp} has a free distance greater than or equal to that of \mathbf{C}_{block} if Π_{inter} is a Type I permutation.*

3.3.2 Second property: periodic permutation

The encoder of an RSC code acts like a scrambler and can be realized by using a shift register with both feedback and feedforward branches shown in Fig. 2.2. It is obvious that such an encoder would have a periodic impulse response. The rate 1/2 RSC code is specified by the transfer matrix $[1, Q^{-1}(D)F(D)]$, where $Q(D), F(D) \in \mathbb{F}_2(D)$ and $Q(D)$ is usually a primitive binary polynomial of degree m . The period of the impulse response of the non-systematic part, $Q^{-1}(D)F(D)$, is given by T_c whose maximum value is $2^m - 1$. We denote by \mathbf{u}^{ij} a weight-2 input sequence whose only nonzero elements are at coordinates i and j ; the corresponding codeword is denoted by \mathbf{C}^{ij} . Therefore, T_c is also the smallest integer such that $\mathbf{u}^{ij}, i \sim j$, will generate a finite-weight output parity sequence. It is thus easy to show [24]

Lemma 3.1 *Let \mathbf{u}^{ij} be an input sequence to a scrambler with period T_c and $scrb(\mathbf{u}^{ij})$ be the corresponding output parity sequence. If $i \sim j$, then there exist $\alpha \in \mathbb{N}$ and $\beta \in \mathbb{Z}$ such that*

$$w_t(scrb(\mathbf{u}^{ij})) = \alpha|i - j|/T_c + \beta, \quad (3.10)$$

where \mathbb{N} is the set of positive integers and α, β depend on the encoder (scrambler) structure.

The puncturing may result in that the sequence generation of a scrambler becomes time-variant. In this case we can search α , β_U , β_L and the weight of sequence is bounded as

$$\alpha|i - j|/T_c + \beta_L \leq w_t(\text{scrb}(\mathbf{u}^{ij})) \leq \alpha|i - j|/T_c + \beta_U, \quad (3.11)$$

where $i \sim j$. In order to simplify our discussion the puncturing case will not be considered in the following. We only remind the effect of the puncturing.

Obviously, if $i \approx j$, \mathbf{u}^{ij} will generate an infinite weight parity sequence if there is no termination at the end of a block. *Lemma 3.1* implies that the codeword weight, $w_t(\mathbf{C}^{ij})$, of a turbo code satisfies

$$w_t(\mathbf{C}^{ij}) \geq 2 + \alpha \left(\frac{|i - j| + |\pi(i) - \pi(j)|}{T_c} \right) + 2\beta, \quad (3.12)$$

with equality holds iff

$$i \sim j \text{ and } \pi(i) \sim \pi(j). \quad (3.13)$$

Define $\tilde{w}_{2,min} \stackrel{\text{def}}{=} \min_{(i,j) \in s_m} w_t(\mathbf{C}^{ij})$, where $s_m \stackrel{\text{def}}{=} \{(i,j) | i \sim j, \pi_{ibp}(i) \sim \pi_{ibp}(j)\}$ and let

$$\delta_{min} = \min_{(i,j) \in s_m} [|i - j| + |\pi_{ibp}(i) - \pi_{ibp}(j)|]. \quad (3.14)$$

For the class of C-IBPTCs, $\tilde{w}_{2,min} = w_{2,min} \stackrel{\text{def}}{=} \min_{(i,j)} w_t(\mathbf{C}^{ij})$, therefore, maximizing the minimum weight of the codewords associated with the weight-2 input sequences is equivalent to maximizing δ_{min} . The next theorem provides an upper bound of $w_{2,min}$ any IBPTC can achieve, if choosing the intra-block permutation is not an option.

Theorem 3.2 *For an IBPTC using an inter-block permutation Π_{inter} , there exists a Π_{intra} such that $w_{2,min} \leq 2 + \alpha(T_s + 1) + 2\beta$, if $L > T_c \cdot T_s$, where T_s is the number of inter-block permuted blocks.*

Proof: Consider the partition $\{0, 1, \dots, L - 1\} = \bigcup_{j=0}^{T_c-1} \bigcup_{l=0}^{T_s-1} S_{jl}$, where $S_{jl} = \{g | g \in S_j, f_n^d(p, g) = l\}$ for some p , $S_j = \{h | h \sim j, 0 \leq h < L\}$, $0 \leq j < T_c$ and

$0 \leq l < T_s$. Note that the decomposition $S_j = \bigcup_{l=0}^{T_s-1} S_{jl}$ is induced by the function $f_n^d(p, g)$ or equivalently, by $f_n(r, g)$. Obviously, the codeword weight of the weight-2 information sequence $\mathbf{u}^{\pi_{ibp}^{-1}(g)\pi_{ibp}^{-1}(h)}$ is large, if \mathbf{u}^{gh} with $g \approx h$. As we are concerned with $w_{2,min}$, only those weight-2 sequences with nonzero coordinate pairs in the set, $\{(g, h) | g \sim h \sim j, \text{ for some } j \text{ and } g, h \in S_{jl} \text{ for some } l\}$ have to be considered.

Assume that $\forall j, l$ all pairs $\{(g, h) \in S_{jl}\}$ satisfy the inequality $|g - h| > T_c \cdot T_s$. For any pair $(g, h) \in S_{jl}$, $g < h$ and the associated interior set $V = \{g+1, g+2, \dots, h-1\}$, we have $|V| \geq T_c \cdot T_s$. If $S_{jl} \cap V \neq \emptyset$, there exists a pair $(g', h') \in S_{jl}$, where $|g' - h'| < |g - h|$. Otherwise, if $S_{jl} \cap V = \emptyset$, by the pigeonhole principle [29], there exists a set S_{uv} such that $|S_{uv} \cap V| \geq 2$, which implies that there is a pair $(g', h') \in S_{uv}$, where $|g' - h'| < |g - h|$.

As both cases lead to contradictions, we conclude that there exists a pair $(g, h) \in S_{jl}$ for some j, l , such that $|g - h| \leq T_c \cdot T_s$. Since it is always possible to find Π_{intra} such that $|\pi_{ibp}^{-1}(g) - \pi_{ibp}^{-1}(h)| = T_c$, eqns. (3.12) and (3.13) then imply that $w_{2,min} \leq 2 + \alpha((T_c + T_c \cdot T_s)/T_c) + 2\beta = 2 + \alpha(T_s + 1) + 2\beta$. ■

Theorem 3.2 indicates that lack of control on the intra-block permutation imposes an upperbound for $w_{2,min}$ which an IBPTC can achieve. The coordinates of nonzero elements of the interleaved sequence \mathbf{u}^{ij} with $i \cong j$ will either remain in the same block or be in the different blocks with probabilities close to $1/T_s$ and $(T_s - 1)/T_s$ when all coordinates in one block are evenly permuted to these blocks. The resulting codewords for the latter case are very likely to have large weights while those for the former case have smaller weights with the worst-case weight of $2 + 2\alpha + 2\beta$ only.

To avoid generating low weight codewords for \mathbf{u}^{ij} , we first notice that eqn. (3.12) implies $\tilde{w}_{2,min} \geq 2 + \alpha(\delta_{min}/T_c) + 2\beta$. The IBP along with the intra-block permutation determines the relation between $|i - j|$ and $|\pi_{ibp}(i) - \pi_{ibp}(j)|$, and their structures can be optimized to maximize δ_{min} . For a pair of coordinates $(i, j) \in s_m$, if the integer-valued function $f_n^d(p, k)$ is injective and satisfies the locally-periodic property for some p , $f_n^d(p, k) = f_n^d(p, k + nT_s)$ for $0 \leq k + nT_s < L$, then the requirements, $i \cong j$ and $\pi_{ibp}(i) \cong$

$\pi_{ibp}(j)$ imply $|\pi_{ibp}(i) - \pi_{ibp}(j)|_{T_s} = 0$ and therefore $|\pi_{ibp}(i) - \pi_{ibp}(j)| \geq lcm(T_c, T_s)$, where $lcm(a, b)$ represents the least common multiple of a and b . In other words,

Lemma 3.2 *An IBPTC that uses a Type III permutation satisfies*

$$\min_{i \cong j, \pi_{ibp}(i) \cong \pi_{ibp}(j), (i,j) \in s_m} w_t(\mathbf{C}^{ij}) \geq 2 + \alpha \cdot \{[T_c + lcm(T_c, T_s)] / T_c\} + 2\beta. \quad (3.15)$$

If T_c and T_s are relative prime, then

$$\min_{i \cong j, \pi_{ibp}(i) \cong \pi_{ibp}(j), (i,j) \in s_m} w_t(\mathbf{C}^{ij}) \geq 2 + \alpha \cdot (T_s + 1) + 2\beta. \quad (3.16)$$

3.4 Constraints on the intra-block permutations

The constraints for intra-block permutation Π_{intra} are also proposed under the derived rules, where $\Pi_{intra,i} = \Pi_{block} \forall i$. The proposed bounds in eqn. (3.16) can not hold for all intra-block permutations Π_{intra} due to the termination method for each block. The different methods provide exceptions on the boundary of each block. We derive some loose constraints for intra-block permutation Π_{intra} to retain the bounds under these IBP rules and various termination methods.

Theorem 3.2 reminds us of the importance of a judicious choice of an intra-block permutation. For the question of how to choose an intra-block permutation whose associated $w_{2,min}$ is guaranteed to surpass the worst-case upperbound of *Theorem 3.2*, *Lemma 3.2* gives only an unrefined answer. We need more elaborate constraints on the selection of the intra-block permutation to avoid producing a $w_{2,min}$ smaller than that bound. In general, any one of the four conditions, $i \not\cong j$, $\pi_{ibp}(i) \not\cong \pi_{ibp}(j)$, $i \not\approx j$, $\pi_{ibp}(i) \not\approx \pi_{ibp}(j)$, is very likely to result in large $w_t(\mathbf{C}^{ij})$. However, there is still a small possibility that low weight codewords will be generated. Before presenting the requirements for eliminating these low weight codewords by using a proper intra-block permutation, we need to define a few new functions to facilitate our discussion.

We denote by \mathbf{u}^k the weight-1 sequence whose only nonzero element is at coordinate k and by $\widetilde{scr}b(\cdot)$ the RSC encoder that encodes a length- L sequence and terminates at the all-zero state using proper tail-bits. Based on the above definitions, we further define, for $0 \leq i, j < L$

$$f_1(i, j) = \begin{cases} \alpha|i-j| + \beta, & \text{if } i \sim j \\ w_t(\widetilde{scr}b(\mathbf{u}^{ij})), & \text{otherwise} \end{cases} \quad (3.17)$$

$$f_2(i, j) = w_t(\widetilde{scr}b(\mathbf{u}^i)) + w_t(\widetilde{scr}b(\mathbf{u}^j)) \quad (3.18)$$

$$f_3(i, j) = \begin{cases} |i-j|, & \text{if } i \sim j \\ \infty, & \text{otherwise} \end{cases} \quad (3.19)$$

$$f_4(i, j) = \min(f_3(i, j), f_3(i, j+L), f_3(i, j-L)). \quad (3.20)$$

As the way a low weight codeword is generated depends on how the encoder terminates its state at the end of a block, we begin with TP-IBPTC.

3.4.1 TP-IBPTC

For the class of TP-IBPTC, a weight-2 input sequence $\mathbf{u}^{ij}, (i, j) \notin s_m$, can not generate an infinite-weight codeword because the encoder state is forced to be terminated at the all-zero state at the end of each block. On the other hand, low-weight codewords may be generated if

$$d_L(i) + d_L(j) + d_L(\pi_{ibp}(i)) + d_L(\pi_{ibp}(j)) < lcm(T_s, T_c) + T_c \quad (3.21)$$

where $d_L(n) = L - |n|_L$ and in addition, (i) both i, j and $\pi_{ibp}(i), \pi_{ibp}(j)$ are near the ends of different blocks, (ii) $i \cong j, \pi_{ibp}(i) \cong \pi_{ibp}(j)$ and both pairs lie close to the end of a block, or (iii) $i \not\cong j$ or $\pi_{ibp}(i) \not\cong \pi_{ibp}(j)$ but both pairs lie close to the end of a block. To avoid generating low weight codewords out of case (i), we require that

$$f_2(|i|_L, |j|_L) + f_2(f_b(\pi_{block}(|i|_L)), f_b(\pi_{block}(|j|_L))) \geq B(T_c, T_s), \quad (3.22)$$

where $B(T_c, T_s) = \alpha [T_c + lcm(T_c, T_s)/T_c] + 2\beta$. Similarly, for cases (ii)-(iii), Π_{block} must satisfy

$$f_1(|i|_L, |j|_L) + f_1(f_b(\pi_{block}(|i|_L)), f_b(\pi_{block}(|j|_L))) \geq B(T_c, T_s), \quad (3.23)$$

if $i \cong j$ and $\pi_{ibp}(i) \cong \pi_{ibp}(j)$, and

$$f_1(|i|_L, |j|_L) + f_2(f_b(\pi_{block}(|i|_L)), f_b(\pi_{block}(|j|_L))) \geq B(T_c, T_s) \quad (3.24)$$

if $i \cong j$ but $\pi_{ibp}(i) \not\cong \pi_{ibp}(j)$, and

$$f_2(|i|_L, |j|_L) + f_1(f_b(\pi_{block}(|i|_L)), f_b(\pi_{block}(|j|_L))) \geq B(T_c, T_s) \quad (3.25)$$

if $i \not\cong j$ but $\pi_{ibp}(i) \cong \pi_{ibp}(j)$.

Π_{intra} may not meet the above conditions in eqns. (3.22)-(3.25). If we impose more constraints on Π_{inter} and the conditions can be relaxed. It is straightforward to show

Lemma 3.3 *For a TP-IBPTC whose inter-block permutation Π_{inter} is of Type IV*

$$\min_{i,j} w_t(\mathbf{C}^{ij}) \geq B(T_c, T_s)$$

if each element in the set $\Gamma_{T_s} = \{(i, j) : 0 \leq i, j \leq L - 1, |\pi_{block}(i) - \pi_{block}(j)|_{T_s} = 0\}$ satisfies

$$\begin{aligned} f_2(i, j) + f_2(\pi_{block}(i), \pi_{block}(j)) &\geq B(T_c, T_s) \\ f_1(i, j) + f_1(\pi_{block}(i), \pi_{block}(j)) &\geq B(T_c, T_s) \end{aligned} \quad (3.26)$$

and $\forall (i, j) \notin \Gamma_{T_s}$ the following two inequalities are satisfied

$$\begin{aligned} f_1(i, j) + f_2(\pi_{block}(i), \pi_{block}(j)) &\geq B(T_c, T_s) \\ f_2(i, j) + f_1(\pi_{block}(i), \pi_{block}(j)) &\geq B(T_c, T_s). \end{aligned} \quad (3.27)$$

Starting with an arbitrary intra-block permutation, say an s -random interleaver [44], we can apply the above criterion iteratively to find the smallest L for a given component code such that $w_t(\mathbf{C}^{ij}) \geq 2 + B(T_c, T_s)$. When L is large enough, e.g., $L > 2(T_c + lcm(T_c, T_s))$, the constraints imposed by the above lemma are relatively easy to meet, i.e., an intra-block permutation that satisfies these constraints is easy to find. For example, it just has to permute the bits near both ends of a block to those coordinates far away from the ends.

3.4.2 TB-IBPTC

We discuss the constraints of intra-block permutation Π_{intra} for TB-IBPTC. The tail-biting encoding results in the codeword weight of weight-1 and weight-2 input information sequences determined by block length L . We give two definitions as follows.

Definition 14 $scr_{tb}^l(\mathbf{u})$ is the weight of a length- L tail-biting convolutional code output for an input sequence \mathbf{u} .

Definition 15

$$S_k = \bigcup_{l=k}^{\infty} [S_1(l) \cup S_2(l)], \quad (3.28)$$

where

$$\begin{aligned} S_1(l) &= \{M = \alpha + 2scr_{tb}^l(\mathbf{u}^i) \mid 0 \leq i < l\} \\ S_2(l) &= \{M = scr_{tb}^l(\mathbf{u}^{ij}) + scr_{tb}^l(\mathbf{u}^{\pi_{block}(i)\pi_{block}(j)}) \mid i \approx j, i \approx j \pm l, \pi_{block}(i) \approx \pi_{block}(j), \\ &\quad \pi_{block}(i) \approx \pi_{block}(j) \pm l, 0 \leq i, j < l\} \end{aligned} \quad (3.29)$$

Let m_k be the smallest integer of the set S_k . Obviously, $\{m_k\}$ is a nondecreasing series of k . Denote the least integer k such that $m_k \geq B(T_c, T_s)$ by k_{min} .

We observed that, for a TB-IBPTC whose block size $L \geq k_{min}$, a weight-2 sequence \mathbf{u}^{ij} generates a codeword whose weight is not larger than the bound $B(T_c, T_s)$ only if $(i, j) \in \Gamma_{T_s}$ and (i, j) satisfies the following conditions:

$$\min \{(|i - j|)_{T_c}, |(L - |i - j|)_{T_c}\} = 0 \quad (3.30)$$

$$\min \{(|\pi_{ibp}(i) - \pi_{ibp}(j)|)_{T_c}, |(L - |\pi_{ibp}(i) - \pi_{ibp}(j)|)_{T_c}\} = 0. \quad (3.31)$$

$$\min \{|i - j|, L - |i - j|\} + \min \{|\pi_{ibp}(i) - \pi_{ibp}(j)|, L - |\pi_{ibp}(i) - \pi_{ibp}(j)|\} < lcm(T_c, T_s) + T_c. \quad (3.32)$$

Such (i, j) pairs will not exist if Π_{inter} is of *Type III* and the corresponding Π_{block} satisfies

$$f_4(|i|_L, |j|_L) + f_4(f_b(\pi_{block}(|i|_L)), f_b(\pi_{block}(|j|_L))) \geq T_c + lcm(T_c, T_s), \quad 0 \leq i, j < L,$$

$\forall (i, j) \in \Gamma_{T_s}$. In manner similar to the TP-IBPTC case, the above constraint on Π_{block} can be further lessened when a *Type IV permutation* is used. In summary,

Lemma 3.4 *For a TB-IBPTC that uses a Type IV permutation with a block length $L \geq k_{min}$, $w_{2,min} \geq B(T_c, T_s)$ if the corresponding Π_{block} satisfies*

$$f_4(|i|_L, |j|_L) + f_4(\pi_{block}(|i|_L), \pi_{block}(|j|_L)) \geq T_c + lcm(T_c, T_s). \quad (3.33)$$

for all $(i, j) \in \Gamma_{T_s}$.

Note that in designing the interleaver for the classic TCs that use the identical tail-biting convolutional code as the component codes, one must also consider the constraint similar to *Lemma 3.4*.

3.4.3 C-IBPTC

For the class of C-IBPTC, we only have to consider $(i, j) \in s_m$. Low weight codewords are associated with those (i, j) pairs whose combined pre-interleaved and post-interleaved distance, $|i - j| + |\pi_{ibp}(i) - \pi_{ibp}(j)|$, is small. The upperbound promised by *Theorem 3.2* can be achieved if

$$f_4(|i|_L, |j|_L) + f_4(f_b(\pi_{block}(|i|_L)), f_b(\pi_{block}(|j|_L))) \geq T_c + lcm(T_c, T_s), \quad (3.34)$$

$\forall (i, j) \notin \Gamma_{T_s}$, if Π_{inter} is of *Type III*.

The constraint (3.34) is used to ensure that the pair $(\pi_{ibp}(i), \pi_{ibp}(j))$ though in different blocks (since $(i, j) \notin \Gamma_{T_s}$) are separated by a large distance.

In analogy to the case of TB-IBPTC, the constraint on Π_{block} can be relaxed if the corresponding Π_{inter} is more restricted. It is easy to show

Lemma 3.5 *For a C-IBPTC that uses a Type IV permutation, if the associated Π_{intra} is such that for all $(i, j) \notin \Gamma_{T_s}$,*

$$f_4(|i|_L, |j|_L) + f_4(\pi_{block}(|i|_L), \pi_{block}(|j|_L)) \geq T_c + lcm(T_c, T_s),$$

then $w_{2,min} \geq B(T_c, T_s)$.

The above discussion shows that the *Type IV permutation* possesses some desired properties and should be used in conjunction with a proper intra-block permutation. Hokfelt *et al.* [54] showed that, as the correlation function of the extrinsic output is exponentially decayed, the interleaver should separate neighboring bits as far as possible. The local periodicity requirement of *Type IV permutation* is consistent with this intuition and let bits or samples within the neighborhood of $T_s - 1$ blocks be moved to the different blocks.

3.5 TB-IBPTC bounds of codeword weights for weight-2 input sequences

Number of blocks for D-IBPTC is of our interest. Classic TC encodes information sequence and permuted information sequence continuously and this is equivalent to the D-IBPTC with only one block. On the other hand, the product code [106] arranges N information bits in a two dimensional array and encodes each row and column separately (discontinuously). The number of blocks for pre-permutation and post-permutation associated with the D-IBPTC are number of rows and columns respectively. Obviously both coding schemes are two special cases of D-IBPTC and the optimum segmentation rule becomes our concern.

We investigate the properties of TB-IBPTC. TB-IBPTC and TP-IBPTC, two D-IBPTC options, are distinguished by termination methods. The tail-padding assigns termination bits and some low weight codeword events are associated with these bits. However the extrinsic information of these bits are unexchangeable during iterative decoding and this causes low weight codeword for TP-IBPTC. The tail-biting avoids these error events induced by the termination bits and the associated codeword weight depends on the length of a block. Furthermore avoiding termination bits does not induce a loss in spectral efficiency. Therefore we derive the codeword bounds for TB-IBPTC.

0	9	18	27	36	45	54	63
1	10	19	28	37	46	55	64
2	11	20	29	38	47	56	65
3	12	21	30	39	48	57	
4	13	22	31	40	49	58	
5	14	23	32	41	50	59	
6	15	24	33	42	51	60	
7	16	25	34	43	52	61	
8	17	26	35	44	53	62	

Figure 3.3: Partition of equivalence classes; $L = 66$, $T_c = 9$.

3.5.1 The achievable weight-2 lower bound

We provide a simplified coordinate partition rule [25] according to the period of an RSC code. The RSC code used in a turbo code is equivalent to an IIR scrambler whose period has a great impact on the distance property of the associated turbo code. The codeword weight associated with a weight-2 input sequence is finite when the difference of these two nonzero coordinates divisible by the period. Breiling [25] applies this property to partition the coordinates of input sequences into some equivalence classes in which any two coordinates is associated with a finite weight codeword. Following the same concept, the simplified partition rule for the i th pre-permutation ($k = 0$) and post-permutation ($k = 1$) sets $F_i^{(k)}$, $k = 0, 1$ is given by

$$F_i^{(k)} = \left\{ \begin{array}{l} \left\{ i + T_c j : 0 \leq j < \left\lfloor \frac{L}{T_c} \right\rfloor \right\}, 0 \leq i < |L|_{T_c} \\ \left\{ i + T_c j : 0 \leq j < \left\lfloor \frac{L}{T_c} \right\rfloor \right\}, |L|_{T_c} \leq i < T_c \end{array} \right. . \quad (3.35)$$

An exemplary partition of eqn. (3.35) is shown in Fig. 3.3 where the integers represent the coordinates of either a pre-permutation or post-permutation sequence. Each row represents an index set $F_i^{(k)}$ and is of size 8 or 7.

The codeword weights of weight-1 and weight-2 input sequences influence the derivation of the lower bound. For example: two non-zero coordinates at i and j of a weight-2

input information sequence permuted to two different blocks and the RSC generates two codewords associated with the weight-1 input information sequence. In order to facilitate our discussion, we give two definitions as follows.

Definition 16 *Given a tail-biting recursive convolutional code, we have*

$$W_2(L) = \min_{i,j,|i-j|_{T_c} \neq 0, |L-i+j|_{T_c} \neq 0} \text{scrb}_{tb}^L(\mathbf{u}_k^{ij}), \quad (3.36)$$

where \mathbf{u}_k^{ij} is a weight-2 input sequence with nonzero elements at coordinates i and j in the k th block.

Definition 17 *Given a tail-biting recursive convolutional code, we have*

$$W_1(L) = \min_i \text{scrb}_{tb}^L(\mathbf{u}_k^i), \quad (3.37)$$

where \mathbf{u}_k^i is a weight-1 input sequence with nonzero element at coordinate i in the k th block.

The weight of a weight-2 input sequence is our main concern. The codeword weight $\text{scrb}_{tb}^L(\mathbf{u}_k^{ij})$ associated with two coordinates i, j satisfying $|i-j|_{T_c} \neq 0, |L-i+j|_{T_c} \neq 0$ is lower-bounded by $W_2(L)$. When two coordinates i, j satisfy $|i-j|_{T_c} = 0$ or $|L-i+j|_{T_c} = 0$, the associated codeword weights are lower-bounded by $\alpha \frac{|i-j|}{T_c} + \beta$ or $\alpha \frac{(L-|i-j|)}{T_c} + \beta$ respectively. We have

$$W(i, j, L) = \begin{cases} \alpha \frac{|i-j|}{T_c} + \beta & , |i-j|_{T_c} = 0 \\ \alpha \frac{L-|i-j|}{T_c} + \beta & , |L-|i-j||_{T_c} = 0 \\ W_2(L) & , \text{otherwise} \end{cases} \quad (3.38)$$

Furthermore, if the puncturing is not applied, $W_2(L)$ can be bounded by $\frac{\alpha(L-T_c)}{T_c} + \beta \leq W_2(L) \leq \frac{\alpha(L+T_c)}{T_c} + \beta$.

Since the RSC code output weights of the weight-2 error events are lower-bounded by the difference of a coordinate pair (i, j) , the weight of a tail-biting encoded classic TC is lower-bounded by

$$\min_{i,j} (2 + W(i, j, L) + W(\pi(i), \pi(j), L)) \quad (3.39)$$

where Π is a length L permutation function.

Before we establish our main result, we need the following two lemmas.

Lemma 3.6 *For each integer set $S_P = \{0, 1, 2, \dots, P - 1\}$, there exists a permutation function Π_P such that $\min_{i \neq j \in S_P} (|i - j|_P + |\pi_P(i) - \pi_P(j)|_P, |i - j|_P + P - |\pi_P(i) - \pi_P(j)|_P, P - |i - j|_P + |\pi_P(i) - \pi_P(j)|_P, 2P - |i - j|_P - |\pi_P(i) - \pi_P(j)|_P) \geq r + 1$, where $r = \lceil \sqrt{P} \rceil - 1$. A permutation function satisfying these constraints is*

$$\pi_P(i) = \left\lfloor ri + \frac{i - |i|_q}{q} \right\rfloor_P, \quad q = \frac{P}{\gcd(P, r)}. \quad (3.40)$$

Proof: See A. ■

Lemma 3.7 *Given N_1 distinct n -element sets and N_2 distinct $(n - 1)$ -element sets, where $n > 1$. If we arrange all elements in these $N_1 + N_2$ sets into a cycle, the minimum separation among elements in the same set is lower-bounded by $N_1 + N_2 - \lceil \frac{N_2}{n} \rceil$ for the n -element sets, and $N_1 + N_2 - \lfloor \frac{N_2}{n} \rfloor$ for the $(n - 1)$ -element sets. Moreover, there are at most $|N_2|_n$ element pairs with separation $N_1 + N_2 - \lceil \frac{N_2}{n} \rceil$ for these n -element sets.*

Proof: See B. ■

Fig. 3.4 shows an exemplary placement for $N_1 = 3$, $N_2 = 6$ and $n = 8$. The minimum separation in these N_1 8-element and N_2 7-element sets is at least 7 and 8, respectively. Moreover, there are only $|N_2|_8 = 6$ element pairs with separation 7 for these 8 element sets.

Based on the above results, we can prove

Theorem 3.3 *There exists a TB-IBPTC of block length L whose minimum codeword weight $w_{2,min}$ for weight-2 input sequences is lower-bounded by*

$$\begin{aligned} w_{2,min} &\geq 2 + 2\beta + \min(W_2(L) + \alpha D_{min} - \beta, \\ &\alpha D_{min} \min \left(\left\lceil \sqrt{N_{max}} \right\rceil, 2|N_2|_{N_{max}} \right) + \\ &\alpha D_{max} \max \left(\left\lceil \sqrt{N_{max}} \right\rceil - 2|N_2|_{N_{max}}, 0 \right) \right), \end{aligned} \quad (3.41)$$

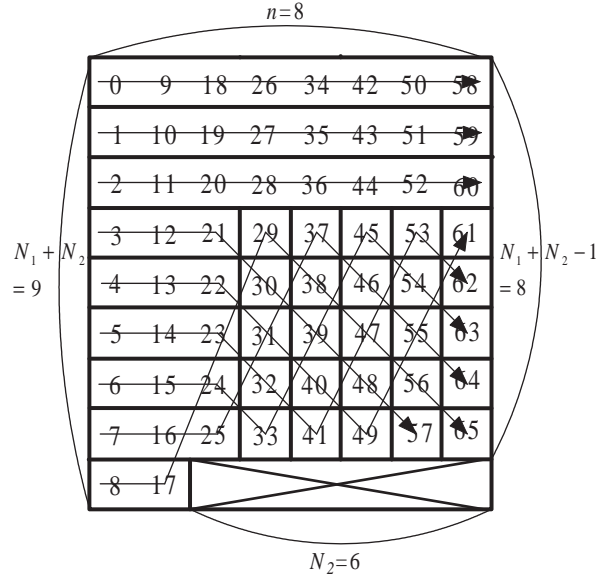
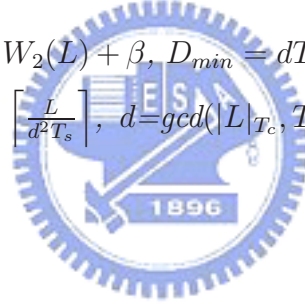


Figure 3.4: Set mapping; $N_1 = 3$, $N_2 = 6$ and $n = 8$.

where $2W_1(L) \geq 2 + \alpha D_{min} + W_2(L) + \beta$, $D_{min} = dT_s - \left\lceil \frac{N_2}{N_{max}} \right\rceil$, $D_{max} = dT_s - \left\lfloor \frac{N_2}{N_{max}} \right\rfloor$, $N_2 = dT_s - \left\lfloor \frac{L}{d} \right\rfloor^{dT_s}$, $N_{max} = \left\lceil \frac{L}{d^2 T_s} \right\rceil$, $d = \gcd(|L|_{T_c}, T_c)$ and T_s is the number of blocks involved in encoding.



Proof: See C. ■

If $L \geq (T_c + 2d)M$, we have

$$\begin{aligned}
& T_c D_{min} \min(2|N_2|_{N_{max}}, \sqrt{N_{max}}) + T_c D_{max} \max(\sqrt{N_{max}} - 2|N_2|_{N_{max}}, 0) \\
& \leq T_c d T_s \left\lceil \sqrt{\left\lceil \frac{L}{d^2 T_s} \right\rceil} \right\rceil < M d \left(\sqrt{\frac{L}{d^2 T_s} + 1} + 1 \right) \leq \sqrt{T_c^2 T_s L + d^2 M^2} + dM \\
& \leq \sqrt{(L - 2dT_c T_s)L + d^2 M^2} + dM \leq \sqrt{L^2 - 2dML + d^2 M^2} + dM = L, \quad (3.42)
\end{aligned}$$

where $M = T_c T_s$. Then

$$\begin{aligned}
& \alpha D_{min} \min\left(2|N_2|_{N_{max}}, \left\lceil \sqrt{N_{max}} \right\rceil\right) + \alpha D_{max} \max\left(\left\lceil \sqrt{N_{max}} \right\rceil - 2|N_2|_{N_{max}}, 0\right) + \beta \\
& \leq \frac{\alpha L}{T_c} + \beta \leq W_2(L) + \alpha, \quad (3.43)
\end{aligned}$$

if no puncturing is applied for the RSC code. Therefore we reach a Corollary as

Table 3.1: (α, β) for some RSC codes.

RSC codes	T_c	(α, β)
$\frac{1+D^2}{1+D+D^2}$	3	(2, 2)
$\frac{1+D+D^3}{1+D^2+D^3}$	7	(4, 2)
$\frac{1+D^2+D^3+D^4}{1+D+D^4}$	15	(8, 2)

Corollary 3.2 *If the block length L is greater than $(T_c + 2d)M$ and no puncturing is applied, then there exists a TB-IBPTC whose minimum codeword weight $w_{2,min}$ for weight-2 input sequences is lower-bounded by*

$$w_{2,min} \geq 2 + \alpha D_{min} \min \left(2|N_2|_{N_{max}}, \left\lceil \sqrt{N_{max}} \right\rceil \right) + \alpha D_{max} \max \left(\left\lceil \sqrt{N_{max}} \right\rceil - 2|N_2|_{N_{max}}, 0 \right) + 2\beta, \quad (3.44)$$

where $2W_1(L) > 2 + \alpha D_{min} + W_2(L) + \beta$, $M = T_c T_s$, $D_{min} = dT_s - \left\lfloor \frac{N_2}{N_{max}} \right\rfloor$, $D_{max} = dT_s - \left\lfloor \frac{N_2}{N_{max}} \right\rfloor$, $N_2 = dT_s - \left\lfloor \frac{L}{d} \right\rfloor^{dT_s}$, $N_{max} = \left\lfloor \frac{L}{d^2 T_s} \right\rfloor$, $d = \gcd(|L|_{T_c}, T_c)$ and T_s is the number of blocks involved in encoding.

3.5.2 Analytical results

We evaluate lower bounds for the RSC codes given in Table 3.1. Figs. 3.5-3.7 plot the lower bounds for various interleaver lengths $T_s L$. Larger component code period generally gives higher lower bound, as indicated by these curves.

Separate encoding improves the lower bounds for some interleaver lengths but also imposes constraints on interleaver lengths. These figures shows 10–50 weight improvements on the lower bound for long interleaver lengths but $W_2(L)$ is small for short interleaver lengths. Fig. 3.7 indicates that, the lower bound is a decreasing function of T_s for short block length. *Corollary 3.2* says that $W_2(L)$ is not a dominant factor of the lower bound if the block length constraint $L \geq (T_c + 2d)M$ is satisfied.

Fig. 3.5 compares the upper bound [25] and the lower bound we derived. The large “gap” between the upper and lower bounds is due to the fact that [25] does not consider the weight-2 error events resulted from adjacent partitions but our derivation does. The

gap would be much reduced if these events were taken into account.

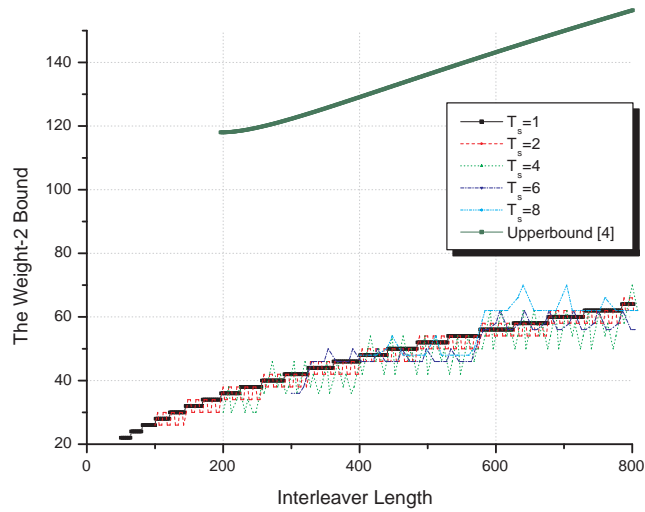


Figure 3.5: The weight 2 lower bound for the Scrambling function $\frac{1+D^2}{1+D+D^2}$.

We derive a general achievable codeword weight lower bound for the weight-2 error events when a TB-IBPTC uses two identical RSC code. The bound implies separate encoding stands a better chance to obtain a weight-2 lower bound larger than that of the conventional continuous encoding scheme if the block length is not too small and is properly chosen. The relationships between these two parameters and the lower bound provide useful design guideline for TB-IBPTC.

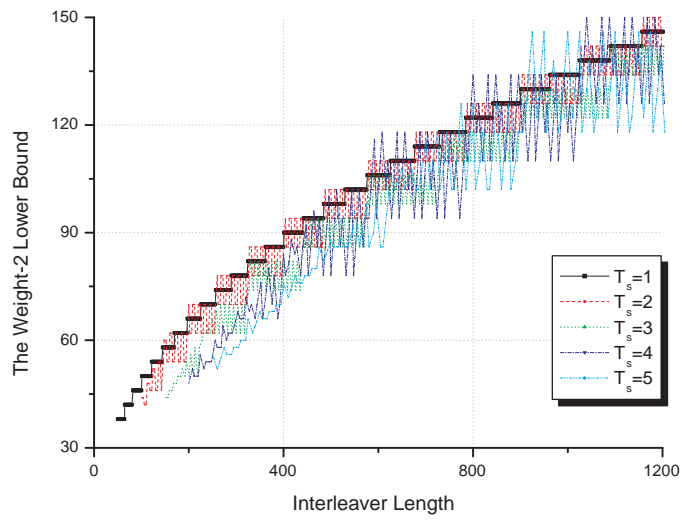


Figure 3.6: The weight 2 lower bound for the Scrambling function $\frac{1+D+D^3}{1+D^2+D^3}$.

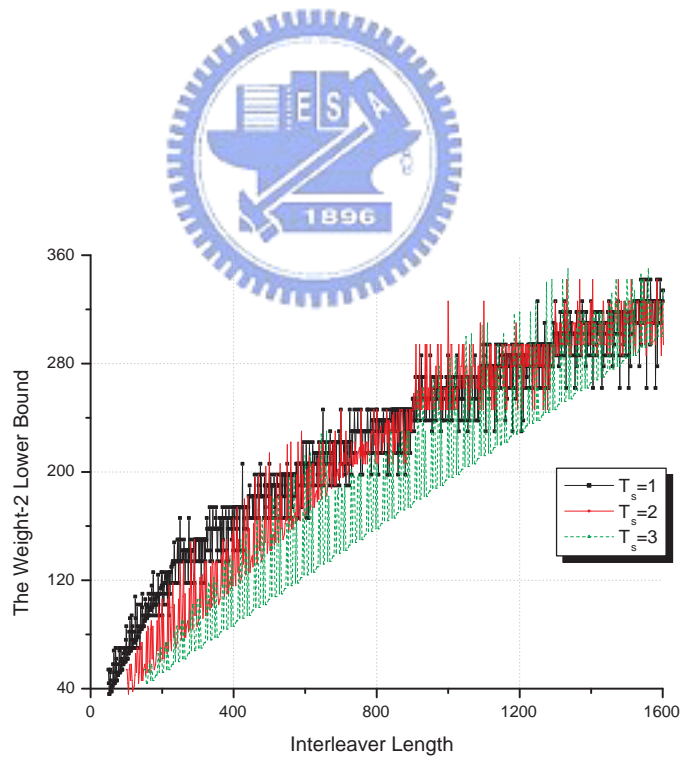


Figure 3.7: The weight 2 lower bound for the Scrambling function $\frac{1+D^2+D^3+D^4}{1+D+D^4}$.

Chapter 4

Block-oriented inter-block permutation interleaver

Block-oriented inter-block permutation (B-IBP) interleaver is a definite length turbo code interleaver design regarding to the parallel turbo decoder architecture. The architecture has five implementation issues: memory contention [49, 20, 98, 97], network routing and control signalling [78, 68, 67, 33], permutation table storage [70, 95, 96] and the support of high-radix APP decoding. B-IBP interleaver can well address these issues. The invariant IBP and identical intra-block permutation mentioned in *Theorem 3.1* resolve memory contention. The network-oriented B-IBP design reduces network routing complexity and simplifies network control signaling. Choosing an intra-block permutation whose permutation table can be generated on-fly avoids memory storage for permutation table. If the block interleaver is generalized maximal contention-free, the associated B-IBP interleaver is also generalized maximal contention-free. Furthermore if the block interleaver supports the high-radix APP decoder [16], the associated B-IBP interleaver also supports the high-radix APP decoder. One can find popular interleavers such as almost regular permutation (ARP) [12, 37, 38, 56], quadratic polynomial permutation (QPP) [90, 85, 92, 93, 3] and inter-window shuffle interleaver [69, 70] belonging to this class of B-IBP interleaver and enjoy these properties. However the interleaver restricts the interleaver length which is identical to the multiple of the block length. Therefore we propose shortening position assigning algorithm. This algorithm

not only supports various input information lengths but also reduces implementation complexity without obvious performance degradation. At last an example of the B-IBP interleavers ranging from 40 to 6144 bits is proposed and this interleaver supports the above mentioned hardware properties. The associated implementation applying the interleaver with 4096 bits is provided and requires less power consumption comparing to fashion designs.

4.1 The parallel turbo decoder architecture and memory contention

The parallel turbo decoder applies N APP decoders instead of a single APP decoder to increase decoding throughput by N . Since there are N APP decoders, the turbo decoder requires N memory banks to store received samples and the extrinsic messages while a network connects these APP decoders and memory banks. Fig. 4.1 (a) shows an example of the architecture with $N = 4$ and a fully-connected network [35] bridges both sides. These APP decoders apply the sliding-window APP (SWAPP) decoding algorithm [104, 21] which can manipulate partial coded sequence to generate the extrinsic information. Therefore we partition a coded sequence into N segments so that each segment is decoded by one APP decoder.

Memory contention occurs due to the serial memory access of these APP decoders and influences the turbo decoding throughput and turbo decoder complexity. Each APP decoder requires prior and successive extrinsic messages to decode each bit (symbol) and it must decode information bits sequentially using a reasonable window size. The APP decoders sequentially access certain memory banks to fetch or write the extrinsic messages according to an interleaving-deinterleaving rule. The interleaving-deinterleaving rule may induce memory contention and more than one APP decoder want to access the same memory bank simultaneously; see Fig. 4.1 (b). Memory contention decreases decoding throughput and increases decoder complexity because these APP decoders require

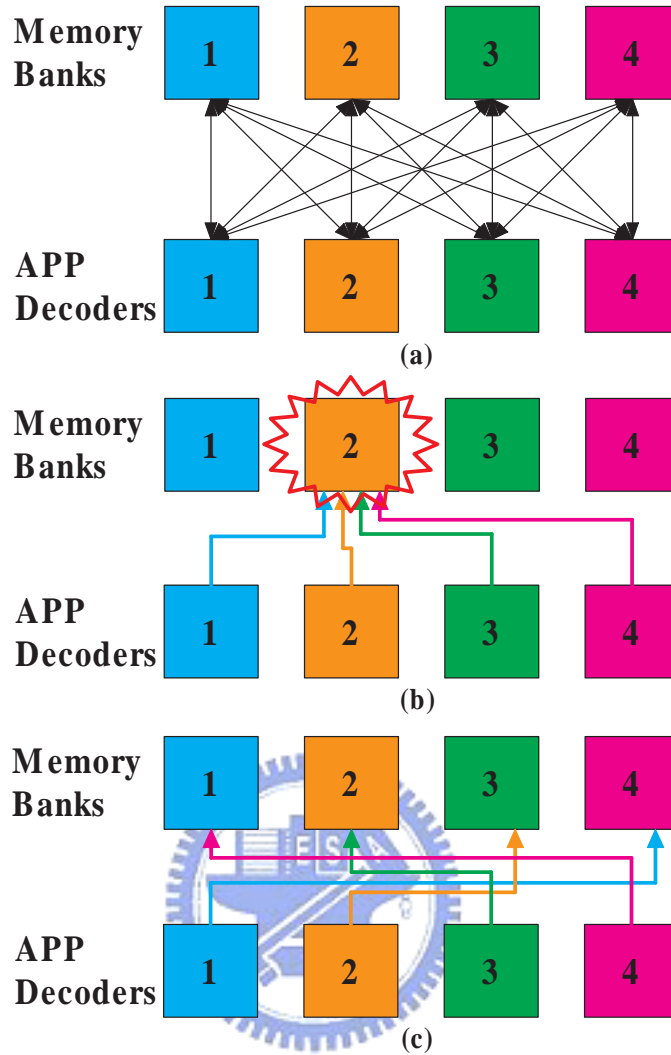


Figure 4.1: (a) The block diagram of the parallel turbo decoder architecture with parallelism degree 4; (b) memory contention; (c) memory contention-free.

contention avoidance circuit to stagger memory access. [49] proposed using a buffer to store temporary data to resolve memory contention but the complexity increases linearly with the number of APP decoders.

An interleaver that resolves memory contention without extra buffer and contention avoidance circuit will minimize the corresponding decoder complexity. To assure memory contention-free (see Fig. 4.1 (c)), the interleaving law has to be such that at each instance there is a one-to-one mapping between the memory banks and APP decoders.

It is also desired that the interleaving rule supports various numbers of APP decoders so that trade-off between complexity and throughput is available. We give a definition for memory contention-free property to facilitate our discussions.

Definition 18 *A length- K interleaver supports memory contention-free property for $N = \lceil \frac{K}{L} \rceil$ APP decoders, if there exist two K -to- N mapping functions \mathcal{M} and \mathcal{M}^d such that $\mathcal{M}(iL+k) \neq \mathcal{M}(jL+k)$, $\mathcal{M}^d(iL+k) \neq \mathcal{M}^d(jL+k)$, $\mathcal{M}^d(\pi(iL+k)) \neq \mathcal{M}^d(\pi(jL+k))$ and $\mathcal{M}(\pi^{-1}(iL+k)) \neq \mathcal{M}(\pi^{-1}(jL+k))$, $\forall 0 \leq i < j < N$, $0 \leq k < L$, $0 \leq iL+k < K$ and $0 \leq jL+k < K$.*

Assume there are N APP decoders, where the j th APP decoder processes information sequence ranging from the jL th symbol to the $(j+1)L-1$ th symbol. We allocate information symbols satisfying the mapping properties described in *Definition 18*. Then the j th APP decoder fetches message from memory bank $\mathcal{M}(jL+k)$ ($\mathcal{M}^d(jL+k)$) and write message to memory bank $\mathcal{M}^d(\pi(jL+k))$ ($\mathcal{M}(\pi^{-1}(jL+k))$) without memory contention if these N APP decoders process the k th bits in each length- L information symbol sequence concurrently. The definition is well-defined.

One may find a contention-free definition as

Definition 19 *A length- K interleaver Π is contention-free for $N = \lceil \frac{K}{L} \rceil$ APP decoders if both $\phi = \pi$ and $\phi = \pi^{-1}$ satisfy*

$$\left\lfloor \frac{\phi(k+iL)}{L} \right\rfloor \neq \left\lfloor \frac{\phi(k+jL)}{L} \right\rfloor. \quad (4.1)$$

This definition is only a special case corresponding to our *Definition 18* with the mapping functions $\mathcal{M}(k+iL) = i$ and $\mathcal{M}^d(k+iL) = i$.

4.2 Block-oriented IBPTC

Block-oriented IBPTC (B-IBPTC) as shown in Fig. 4.2 applies B-IBP interleaver. B-IBP interleaver describes most popular memory contention-free interleavers such as the

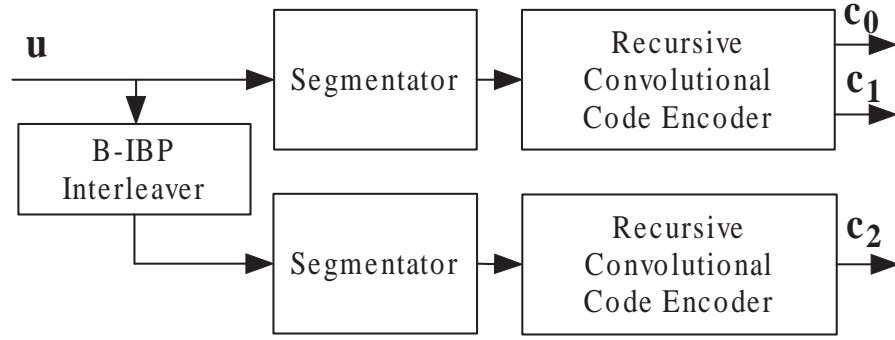


Figure 4.2: The block diagram of a block-oriented IBPTC encoder.

quadratic polynomial permutation (QPP) [90, 85, 92, 93, 3], almost regular permutation (ARP) [12, 37, 38, 56], inter-window shuffle [69, 70] interleavers. Multiple slice turbo code [47] also belongs to the B-IBPTC with separate encoding. Turbo code applying these interleavers can utilize the same decoder architecture to increase turbo decoder throughput. Regarding to maximal contention-free property, we also prove that the B-IBP interleaver is maximal contention-free if the block interleaver is also a maximal contention-free interleaver and the corresponding general memory mapping function is described. At last, we discuss the support of the high-radix APP decoder and the associated memory mapping functions.

4.2.1 B-IBP interleaver

B-IBP interleaver is a class of IBP interleaver. This interleaver is composed of definite N length- L blocks. B-IBP interleaver resolves memory contention by two design rules which are the invariant permutation property and identical intra-block permutation. Given these two rules, the B-IBP interleaver Π_{B-ibp} with N blocks is formulated as

$$\begin{aligned}
 \pi_{B-ibp}(iL + k) &= \pi_{B-inter}(\pi_{intra}(iL + k)) = \pi_{B-inter}(iL + \pi_{block}(k)) \\
 &= f_n(i, \pi_{block}(k))L + \pi_{block}(k), \quad \forall 0 \leq k < L, 0 \leq i < N, \quad (4.2)
 \end{aligned}$$

where $f_b(k) = k$ for $\Pi_{B-inter}$ and $|\pi_{intra}(k + iL)|_L = \pi_{block}(k)$. Fig. 4.3 shows an example of the B-IBP interleaver with interleaver length 16 bits and block length 4 bits. In this

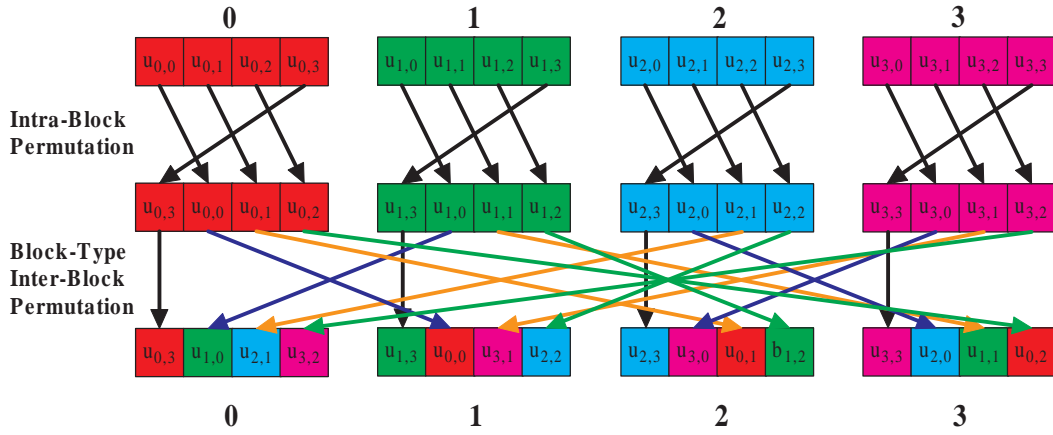


Figure 4.3: An example of block-oriented IBP interleaving.

example, $\pi_{block}(i) = |i + 1|_4$ and $f_n(i, j) = i \oplus j$, where \oplus is an exclusive-or operation, e.g. $1 \oplus 3 = 2$.

Creating good B-IBP interleavers requires less effort and the potential implementation complexity is avoidable. B-IBP interleaver generally has good distance property if the block interleaver possesses good distance property in each block. The B-IBP design guideline in *Theorem 3.2* further enhances the distance property of B-IBP interleaver. Therefore we can search a good short length interleaver and then apply the above B-IBP design guidelines to acquire good B-IBP interleavers. Since finding good short length interleavers requires less effort than finding good long length interleavers, constructing good B-IBP interleavers becomes simple. We can design B-IBP interleaver in two ways: given a B-IBP to find a good block interleaver and given a block interleaver to search a good B-IBP. The first way can impose hardware constraints to B-IBP and reduce the routing complexity and simplify the associated control signalling. Then the successive effort is to search a good short length interleaver given the constrained B-IBP. The second way is applied when the intra-block permutation is given. Then the next step is to find good B-IBP. Both ways leave degree of freedom in designing high performance B-IBP while the implementation complexity is to the least.

B-IBP is also a general description for popular classes of interleavers which support

parallel decoding without memory contention. There are two popular interleavers: QPP and ARP. The QPP has been defined in the 3GPP LTE [3] and the ARP has been applied to the DVB-RCS/RCT [37, 38] and WiMAX [56]. A length- K QPP is formulated as

$$\pi_{QPP}(i) = |f_2 i^2 + f_1 i|_K, \quad (4.3)$$

where f_1 and f_2 are two integers and references [90, 85] discuss the constraints for both f_1 and f_2 . A length- K ARP is formulated as

$$\pi_{ARP}(i) = |iP_0 + A + d(i)|_K, \quad (4.4)$$

where P_0 is the relative prime to K , A is a constant, $d(i)$ is a “dither” vector of length C which is the factor of K . For all block sizes, $d(i)$ has the form

$$d(i) = P_0 \alpha(|i|_C) + \beta(|i|_C), \quad (4.5)$$

where $\alpha(\cdot)$ and $\beta(\cdot)$ are both vectors of length C . The following two theorems prove that QPP and ARP are B-IBP interleavers.

Theorem 4.4 *QPP is a B-IBP interleaver.*

Proof: Suppose L is a factor of K . We decompose eqn. (4.3) as

$$\begin{aligned} & \pi_{QPP}(iL + k) \\ &= |f_2(iL + k)^2 + f_1(iL + k)|_K \\ &= |(f_2 i^2 L + 2f_2 ik + f_1 i)L + f_2 k^2 + f_1 k|_K \\ &= |(f_2 i^2 L + 2f_2 ik + f_1 i + |f_2 k^2 + f_1 k|_L)L + |f_2 k^2 + f_1 k|_L|_K \\ &= |(f_2 i^2 L + 2f_2 ik + f_1 i + |f_2 k^2 + f_1 k|_L)|_{\frac{K}{L}}L + |f_2 k^2 + f_1 k|_L. \end{aligned} \quad (4.6)$$

■

Theorem 4.5 *ARP is a B-IBP interleaver.*

Proof: We partition a coordinate set $\{0, 1, 2, \dots, K - 1\}$ into C subsets $S_k = \{iC + k | 0 \leq i < \frac{K}{C}\}$, $0 \leq k < C$. According to eqn. (4.4), we have

$$\begin{aligned}
\pi_{ARP}(iC + k) &= |(iC + k)P_0 + A + \alpha(|iC + k|_C)P_0 + \beta(|iC + k|_C)|_K \\
&= |iCP_0 + kP_0 + A + \alpha(k)P_0 + \beta(k)|_K \\
&= |iCP_0 + A'(k)|_K,
\end{aligned} \tag{4.7}$$

where $0 \leq i < \frac{K}{C}$. Since Π_{ARP} is a permutation function and P_0 is relative prime to K , there are no $0 \leq i', j' < \frac{K}{C}$ such that $|iCP_0|_K = |i'C|_K = |j'C|_K = |jCP_0|_K$ if $i \neq j$. Then Π_{ARP} maps all elements in S_k to $S_{A'(k)}$ and any two elements in different subsets are mapped to different subsets.

Suppose L is a factor of K and C is a factor of L , eqn. (4.7) is further expressed as

$$\begin{aligned}
&\pi_{ARP}(iL + jC + k) \\
&= |(iL + jC)P_0 + A'(k)|_K \\
&= |iP_0L + jCP_0 + A'(k)|_K \\
&= |(iP_0 + ||jCP_0 + A'(k)||_L)L + |jCP_0 + A'(k)|_L|_K \\
&= |iP_0 + ||jCP_0 + A'(k)||_L|\frac{K}{L}L + |jCP_0 + A'(k)|_L.
\end{aligned} \tag{4.8}$$

Since eqn. (4.8) has the same form to eqn. (4.2) and all elements in each subset S_k are permuted to the same subset $S_{A'(k)}$, ARP is a B-IBP interleaver. ■

B-IBP interleaver features memory contention-free with simple memory mapping function. The invariant permutation property and identical intra-block permutation avoid memory contention when the number of APP decoders is identical to the factor of N . The reversed B-IBP manner also resolves memory contention when the number of APP decoders is smaller than the factor of N . Both methods only require simple on-fly generated memory mapping functions and avoid the large storage of complex memory mapping functions [95, 96]. The parallelization methods are described in the following sections.

4.2.2 Parallelization method in the B-IBP manner

B-IBP interleaver composed of N blocks supports turbo decoder with parallelism degree N or the factor of N . We apply B-IBP to configure the network and intra-block permutation to determine memory addressing within a memory bank. Then we define two memory mapping functions \mathcal{M} and \mathcal{M}^d

$$\mathcal{M}(jL + k) = j, \quad (4.9)$$

$$\mathcal{M}^d(jL + k) = j. \quad (4.10)$$

Following eqns. (4.2) and (3.6), we have

$$\mathcal{M}(\pi_{B-ibp}^{-1}(jL + k)) = \mathcal{M}(f_n^d(j, k)L + \pi_{block}^{-1}(k)) = f_n^d(j, k), \quad (4.11)$$

$$\mathcal{M}^d(\pi_{B-ibp}(jL + k)) = \mathcal{M}^d(f_n(j, \pi_{block}(k))L + \pi_{block}(k)) = f_n(j, \pi_{block}(k)). \quad (4.12)$$

Because $f_n(j, k)$ and $f_n^d(j, k)$ are permutation functions corresponding to each k and block interleavers are identical for all blocks, according to *Definition 18*, B-IBP interleaver supports memory contention-free with parallelism degree N . If the parallelism degree decreases to any factor of N , B-IBP interleaver also supports memory contention-free. A theorem is given as follows.

Theorem 4.6 *If R is a factor of N , there exist memory mapping functions \mathcal{M}_R and \mathcal{M}_R^d of a B-IBP interleaver composed of N length- L blocks such that the parallelism degree R is achievable without memory contention.*

Proof: For an integer $k \in [0, L)$, we first assume $\tilde{k} = \pi_{block}(k)$, $\mathcal{M}_R(jL + k) = \lfloor j \rfloor_{\frac{N}{R}}$, $\mathcal{M}_R^d(jL + \tilde{k}) = \lfloor j \rfloor_{\frac{N}{R}}$, $S_i = \{i, i + \frac{N}{R}, i + \frac{2N}{R}, \dots, i + N - R\}$ and $S = \{0, 1, \dots, R - 1\}$. We define an index vector $\mathbf{I}(i) = N, \forall 0 \leq i < N$.

Start from $i = 0$. If there exist any two $j_1, j_2 \in S_0$ such that $\mathcal{M}_R^d(f_n(j_1, \tilde{k})L + \tilde{k}) = \mathcal{M}_R^d(f_n(j_2, \tilde{k})L + \tilde{k}) = m \in S$, there exists a value $m' \in S$ such that $\mathcal{M}_R^d(j'L + \tilde{k}) = m' \neq \mathcal{M}_R^d(f_n(j, \tilde{k})L + \tilde{k}), \forall j \in S_0$, where $\mathbf{I}\left(\frac{m'N}{R} + \lfloor j' \rfloor_{\frac{N}{R}}\right) = N$ and $\lfloor j' \rfloor_{\frac{N}{R}} =$

$|f_n(j_2, \tilde{k})|_{\frac{N}{R}}$. We set $\mathcal{M}_R^d(f_n(j_2, \tilde{k})L + \tilde{k}) = m'$ and $\mathcal{M}_R^d(j'L + \tilde{k}) = m$. We do this procedure until $\mathcal{M}_R^d(f_n(j_1, \tilde{k})L + \tilde{k}) \neq \mathcal{M}_R^d(f_n(j_2, \tilde{k})L + \tilde{k}), \forall j_1 \neq j_2 \in S_0$. Then we set $\mathbf{I}\left(\frac{mN}{R} + |f_n(j, \tilde{k})|_R\right) = j, \forall j \in S_0$, where $m = \mathcal{M}_R^d(f_n(j, \tilde{k})L + \tilde{k})$.

Increase i to $i + 1$. If there exist any two $j_1, j_2 \in S_i$ such that $\mathcal{M}_R^d(f_n(j_1, \tilde{k})L + \tilde{k}) = \mathcal{M}_R^d(f_n(j_2, \tilde{k})L + \tilde{k}) = m \in S$, we search a value $m' \in S$ such that $\mathcal{M}_R^d(j'L + \tilde{k}) = m' \neq \mathcal{M}_R^d(f_n(j, \tilde{k})L + \tilde{k}), \forall j \in S_i$, where $\mathbf{I}\left(\frac{m'N}{R} + |j'|_{\frac{N}{R}}\right) = N$ and $|j'|_{\frac{N}{R}} = |f_n(j_1, \tilde{k})|_{\frac{N}{R}}$. If j' exists, we set $\mathcal{M}_R^d(f_n(j_1, \tilde{k})L + \tilde{k}) = m'$ and $\mathcal{M}_R^d(j'L + \tilde{k}) = m$. If j' does not exist, we search a value $m' \in S$ such that $\mathcal{M}_R^d(j''L + \tilde{k}) = m' \neq \mathcal{M}_R^d(f_n(j, \tilde{k})L + \tilde{k}), \forall j \in S_i$, where $\mathbf{I}\left(\frac{m'N}{R} + |j''|_{\frac{N}{R}}\right) = N$ and $|j''|_{\frac{N}{R}} = |f_n(j_2, \tilde{k})|_{\frac{N}{R}}$. If j'' exists, we set $\mathcal{M}_R^d(f_n(j_2, \tilde{k})L + \tilde{k}) = m'$ and $\mathcal{M}_R^d(j''L + \tilde{k}) = m$. If j'' does not exist, we start the following procedure.

There exists j^0 such that $\mathcal{M}_R^d(j^0L + \tilde{k}) = m' \neq \mathcal{M}_R^d(f_n(j, \tilde{k})L + \tilde{k}), \forall j \in S_i$, where $|j^0|_{\frac{N}{R}} = |f_n(j_1, \tilde{k})|_{\frac{N}{R}}$. We set $\mathcal{M}_R^d\left(f_n\left(\mathbf{I}\left(\frac{m'N}{R} + |j^0|_{\frac{N}{R}}\right), \tilde{k}\right)L + \tilde{k}\right) = m, \mathcal{M}_R^d(f_n(j_1, \tilde{k})L + \tilde{k}) = m'$ and $\mathbf{I}\left(\frac{mN}{R} + |f_n(j_1, \tilde{k})|_{\frac{N}{R}}\right) = \mathbf{I}\left(\frac{m'N}{R} + |j^0|_{\frac{N}{R}}\right)$. There exists j^1 such that $\mathcal{M}_R^d(f_n(j^1, \tilde{k})L + \tilde{k}) = \mathcal{M}_R^d\left(f_n\left(\mathbf{I}\left(\frac{mN}{R} + |j^0|_{\frac{N}{R}}\right), \tilde{k}\right)L + \tilde{k}\right) = m \in S$, where $\|j^1\|_{\frac{N}{R}} = \left\|\mathbf{I}\left(\frac{mN}{R} + |j^0|_{\frac{N}{R}}\right)\right\|_{\frac{N}{R}}$. If there exists j^2 such that $\mathcal{M}_R^d(j^2L + \tilde{k}) = m'$ and $\mathbf{I}\left(\frac{m'N}{R} + |j^2|_{\frac{N}{R}}\right) = N$, where $|j^2|_{\frac{N}{R}} = |f_n(j^1, \tilde{k})|_{\frac{N}{R}}$, we set $\mathcal{M}_R^d(f_n(j^1, \tilde{k})L + \tilde{k}) = m', \mathcal{M}_R^d(j^2L + \tilde{k}) = m, \mathbf{I}\left(\frac{m'N}{R} + |j^2|_{\frac{N}{R}}\right) = j^1$ and $\mathbf{I}\left(\frac{m'N}{R} + |f_n(j^1, \tilde{k})|_{\frac{N}{R}}\right) = N$ and stop the this procedure. If $\mathbf{I}\left(\frac{m'N}{R} + |j^2|_{\frac{N}{R}}\right) \neq N$, we set $\mathcal{M}_R^d(f_n(j^1, \tilde{k})L + \tilde{k}) = m', \mathcal{M}_R^d(j^2L + \tilde{k}) = m, \mathbf{I}\left(\frac{m'N}{R} + |f_n(j^1, \tilde{k})|_{\frac{N}{R}}\right) = \mathbf{I}\left(\frac{mN}{R} + |j^2|_{\frac{N}{R}}\right)$ and $\mathbf{I}\left(\frac{mN}{R} + |j^2|_{\frac{N}{R}}\right) = f_n(j^1, \tilde{k})$. We search the following j^l and j^{l+1} , the same as the search of j^1 and j^2 , we update the corresponding memory mapping function and index factor, where l is odd. Since there are i occupied entries in memory bank m' and at least $i + 2$ occupied entries in memory bank m , there exist a j^{l+1} such that $\mathbf{I}\left(\frac{m'N}{R} + |j^{l+1}|_{\frac{N}{R}}\right) = N$ and we stop this procedure.

Then we go back to check if there exist any two $j_1, j_2 \in S_i$ such that $\mathcal{M}_R^d(f_n(j_1, \tilde{k})L + \tilde{k}) = \mathcal{M}_R^d(f_n(j_2, \tilde{k})L + \tilde{k}) = m \in S$ and update the corresponding mapping function and index vector. If there exist no $j_1, j_2 \in S_i$, we set $\mathbf{I}\left(\frac{mN}{R} + |f_n(j, \tilde{k})|_{\frac{N}{R}}\right) = j, \forall j \in S_i$, where $m = \mathcal{M}_R^d(f_n(j, \tilde{k})L + \tilde{k})$ and increase i to $i + 1$ and update the corresponding

mapping function and index vector.

We do the same procedure on \mathcal{M}_R to acquire the memory mapping function avoiding memory contention. Therefore for each k , there exist memory mapping functions such that memory contention-free holds. ■

Theorem 4.6 indicates that there exist memory mapping functions \mathcal{M}_R and \mathcal{M}_R^d such that the parallelism degree can decrease to R . Tarable [95, 96] also provided a similar proof and result to support memory contention-free but the storage of memory mapping functions is large. The same case occurs for the memory mapping functions if there is no constraint on the B-IBP. The following definition describes the constraint on the B-IBP such that the necessary storage for the memory mapping function is to the least.

Definition 20 *The memory mapping functions \mathcal{M}_R and \mathcal{M}_R^d for a B-IPB interleaver composed of N length- L blocks is regular if $\mathcal{M}_R(jL + k_1) = \mathcal{M}_R(jL + k_2)$ and $\mathcal{M}_R^d(jL + k_1) = \mathcal{M}_R^d(jL + k_2)$, $\forall 0 \leq i < N, 0 \leq k_1, k_2 < L$.*

If the memory mapping functions are regular, the necessary storage of memory mapping function decreases to $1/L$.

Fig. 4.4 demonstrates two examples of the parallel turbo decoding procedure for the memory mapping and merged memory mapping. The length-42 B-IBP interleaver is composed of $N = 6$ blocks and each block contains 7 symbols. At first, suppose there are $N = 6$ APP decoders, the j th APP decoder processes from $u_{j,0}$ to $u_{j,6}$ in Fig. 4.4 (a). At each instance, these bits processed by these APP decoders are permuted to different memory banks and memory contention is avoided. If the necessary parallelism degree is 3 which is the factor of $N = 6$, we merges even memory banks with the prior odd memory banks. Then these three APP decoders access information bits from $u_{0,0}, u_{2,0}, u_{4,0}$ as shown in Fig. 4.4. The lower turbo decoder complexity is achievable and the memory contention-free holds.

Unified memory bank is one implementation advantage of B-IBP interleaver. One can find that the block permutation is identical in each memory bank. This implies

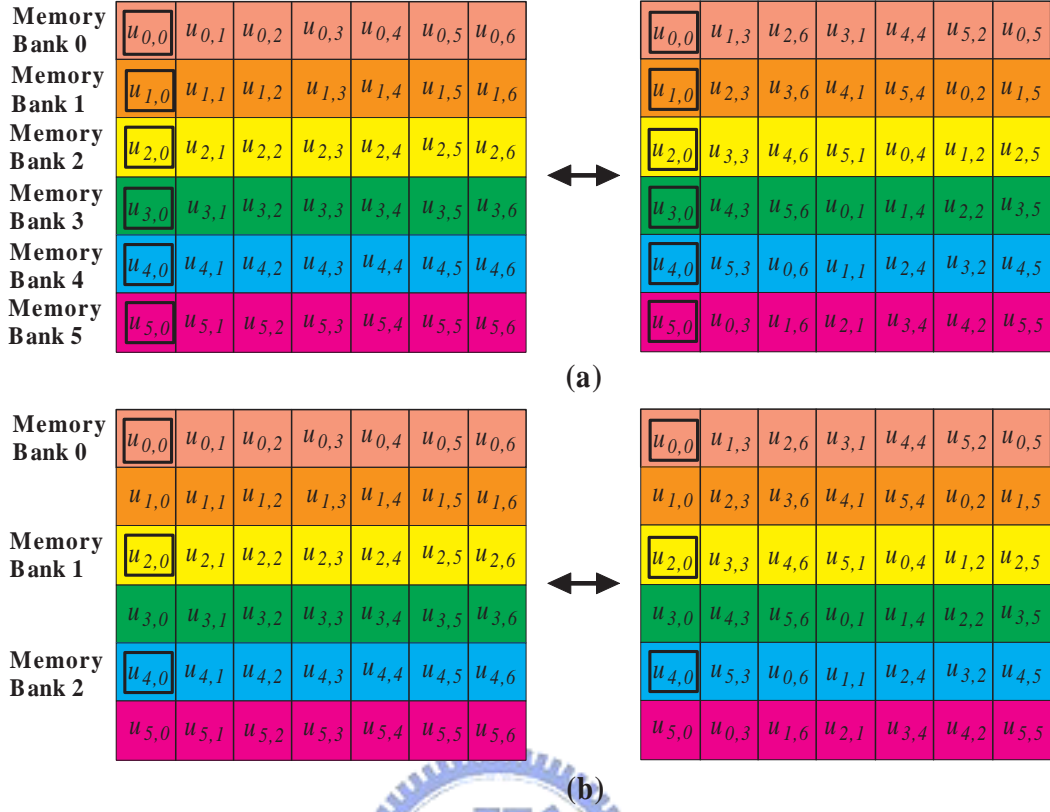


Figure 4.4: (a) The memory mapping for a B-IBP interleaver composed of 6 blocks to support parallelism degree 6; (b) the merged memory mapping for a B-IBP interleaver composed of 6 blocks to support parallelism degree 3.

that only one memory control element is necessary to coordinate these memory banks. In other words we can apply a memory bank with wider input/output port instead of multiple memory banks with narrower input/output port. This saves control elements and the resultant turbo decoder complexity is further eliminated.

This method does not resolve memory contention for the number of APP decoders is not equal to any factor of N . If these APP decoders start decoding from the beginning of a block, there is at least one block which can not be processed at the same time, i.e. the decoding latency increases. If these APP decoders do not start decoding from the beginning of a block, memory contention may occur given the memory mapping functions in eqns. (4.9), (4.10). A dilemma occurs. The following subsection provides another view for the B-IBP interleaver to overcome this situation.

4.2.3 Parallelization method in the reversed B-IBP manner

The reversed manner of B-IBP interleaver resolves memory contention for the case that the number of APP decoders P is less than N or not the factor of N . We exchange the roles of intra-block permutation and inter-block permutation; the intra-block permutation configures the network and the inter-block permutation determines memory addressing within a memory bank. Then we give memory mapping functions \mathcal{M} and \mathcal{M}^d as

$$\mathcal{M}(jL + k) = k, \quad (4.13)$$

$$\mathcal{M}^d(jL + k) = k. \quad (4.14)$$

We allocate the starting ordinals of the adjacent APP decoders are separated by $\lceil \frac{K}{P} \rceil$ or $\lfloor \frac{K}{P} \rfloor$ such that their decodings finish at close instance while the ordinals corresponding to these blocks are different. Following the eqn. (4.2), we acquire

$$\mathcal{M}(\pi_{B-ibp}^{-1}(jL + k)) = \mathcal{M}(f_n^d(j, k)L + \pi_{block}^{-1}(k)) = \pi_{block}^{-1}(k), \quad (4.15)$$

$$\mathcal{M}^d(\pi_{B-ibp}(jL + k)) = \mathcal{M}^d(f_n(j, \pi_{block}(k))L + \pi_{block}(k)) = \pi_{block}(k). \quad (4.16)$$

Since these decoders access from different ordinals in these blocks, memory contention is avoided. Even these APP decoders fetch or write on the successive S symbols, $\pi_{block}^{-1}(|k + S|_L)$ and $\pi_{block}^{-1}(|k + S|_L)$ are still different for these APP decoders and memory contention-free still holds.

Fig. 4.5 demonstrates an example of this reversed memory mapping. The length-42 B-IBP interleaver is composed of 7 blocks. We apply 6 memory banks for $P = 3$ APP decoders. The stating positions for these APP decoder are 0, 14 and 28 and the corresponding ordinals in these blocks are 0, 2 and 4 respectively. Therefore these APP decoders write messages back to memory banks without memory contention. The same as the interleaving procedure, the deinterleaving procedure also starts from 0, 14, 28 and the mapping avoids memory contention.

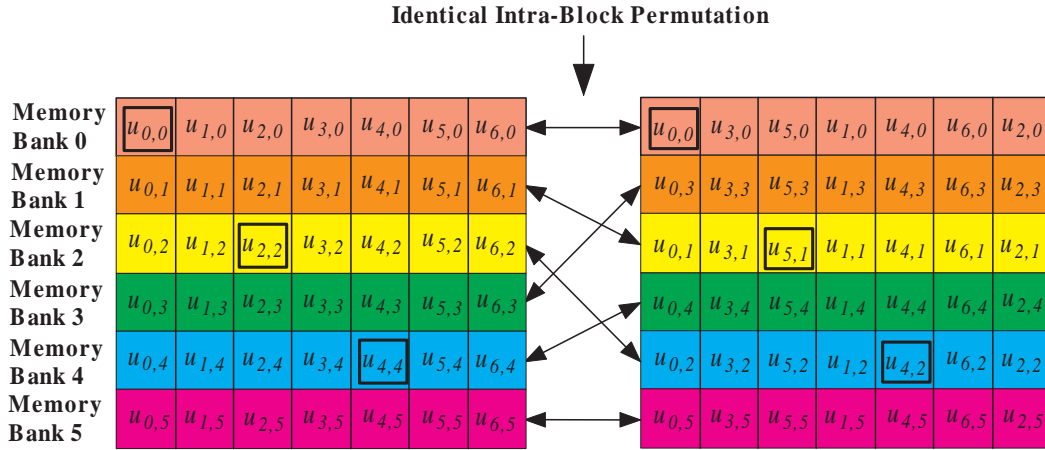


Figure 4.5: The reversed memory mapping for the B-IBP interleaver composed of 7 blocks to support parallelism degree 3.

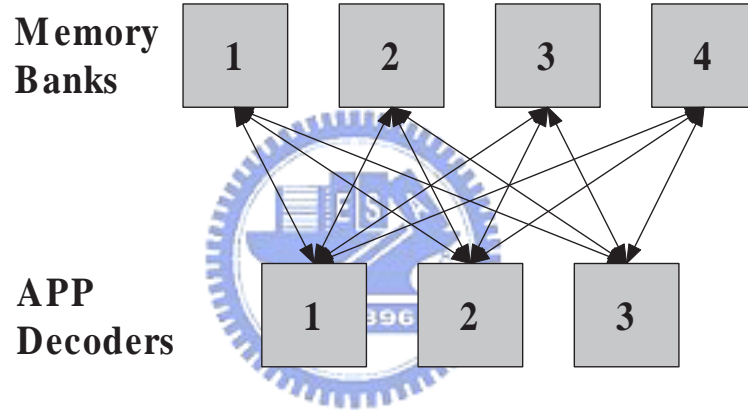


Figure 4.6: The block diagram of the asymmetric parallel turbo decoder architecture with 4 memory banks and 3 APP decoders.

Since the number of APP decoders is different to the number of blocks, an asymmetric decoder architecture appears. Fig. 4.6 shows an example of the architecture which possesses three APP decoders and four memory banks. One can find that one extra memory bank is necessary and at least one more memory control element is required. Furthermore, the starting positions of these APP decoders in memory banks are not identical and these memory banks apply different memory addressing for these APP decoders. Unified memory bank is generally impossible. These implementation disadvantages comes from the asymmetric architecture.

4.2.4 Generalized maximal contention-free and intra-block permutation

Maximal contention-free property is proposed in [92] and an interleaver possessing this property supports flexible parallelism degree. The definition is given below.

Definition 21 A length- K interleaver Π is maximal contention-free if both $\phi = \pi$ and $\phi = \pi^{-1}$ satisfy

$$\left\lfloor \frac{\phi(k+iL)}{L} \right\rfloor \neq \left\lfloor \frac{\phi(k+jL)}{L} \right\rfloor, \quad (4.17)$$

for all factors L of K , where $0 \leq k < L$ and $0 \leq i < j < \frac{K}{L}$.

When we apply the memory mapping function in eqns. (4.9) and (4.10), the interleaver supports memory contention-free with parallelism degree the factors of K .

Definition 21 is narrow sense and some good interleavers satisfying *Definition 18* may be skipped due to eqn. (4.17). When we apply memory mapping functions in eqns. (4.13) and (4.14), there exists an interleaver satisfying *Definition 21* avoids memory contention. *Definition 21* may not support memory contention-free as memory mapping functions in eqns. (4.13) and (4.14) are applied. *Definition 21* seems restrictive and necessitates modification. We give a new generalized definition as below.

Definition 22 A length- K interleaver is generalized maximal contention-free if there exist memory mapping functions for all factors of K satisfy *Definition 18*.

This definition provides a concrete picture in finding interleavers with both good error rate performance and more flexibility in memory contention-free property.

The B-IPB interleaver with the block interleaver satisfying generalized maximal contention-free property is also a generalized maximal contention-free interleaver. In most cases B-IBP interlaever is not a generalized maximal contention-free interleaver because intra-block permutation is not an option. Section 4.2.2 shows that a length- NL B-IBP interleaver supports parallelism degree to any factor of the total number blocks N

but does not promise that any factor of NL is supportable. However if the block interleaver is a generalized maximal contention-free interleaver, there exists memory mapping functions to support parallelism degree which is any factor of L . Therefore these exist composite memory mapping functions such that B-IBP interleaver is also a generalized memory contention-free interleaver. The theorem is provided as follows.

Theorem 4.7 *If the block interleaver is a generalized maximal contention-free interleaver, the B-IBP interleaver is also a generalized maximal contention-free interleaver.*

Proof: The block interleaver is a generalized maximal contention-free interleaver and there exist memory mapping functions $\mathcal{M}_{i,block}$ and $\mathcal{M}_{i,block}^d$ avoiding memory contention with parallelism degree i which is the factor of block length L . Suppose there are N blocks for the B-IBP interleaver and *Theorem 4.6* shows that there exist memory mapping functions $\mathcal{M}_{k,B-IBP}$ and $\mathcal{M}_{k,B-IBP}^d$ avoiding memory contention with the degree k which is the factor of N . Assume $Q = gcd(\frac{N}{k}, i)$. We construct a composite memory mapping functions $\mathcal{M}_{ki,C-B-IBP}$ and $\mathcal{M}_{ki,C-B-IBP}^d$ supporting parallelism degree ki as

$$\begin{aligned} \mathcal{M}_{ki,C-B-IBP} \left(m \frac{NL}{ki} + n \right) &= \mathcal{M}_{kQ,B-IBP} \left(\left\| m \frac{NL}{ki} + n \right\|_L L + \left\| m \frac{NL}{ki} + n \right\|_L \right) \frac{i}{Q} \\ &\quad + \mathcal{M}_{\frac{i}{Q},block} \left(\left\| m \frac{NL}{ki} + n \right\|_L \right), \end{aligned} \quad (4.18)$$

$$\begin{aligned} \mathcal{M}_{ki,C-B-IBP}^d \left(m \frac{NL}{ki} + n \right) &= \mathcal{M}_{kQ,B-IBP}^d \left(\left\| m \frac{NL}{ki} + n \right\|_L L + \left\| m \frac{NL}{ki} + n \right\|_L \right) \frac{i}{Q} \\ &\quad + \mathcal{M}_{\frac{i}{Q},block}^d \left(\left\| m \frac{NL}{ki} + n \right\|_L \right). \end{aligned} \quad (4.19)$$

kQ and $\frac{i}{Q}$ are factors of N and L respectively, and these memory mapping functions $\mathcal{M}_{kQ,B-IBP}$, $\mathcal{M}_{kQ,B-IBP}^d$, $\mathcal{M}_{\frac{i}{Q},block}$ and $\mathcal{M}_{\frac{i}{Q},block}^d$ exist. If $\left\| \frac{m_1 NL}{ki} \right\|_L = \left\| \frac{m_2 NL}{ki} \right\|_L$, where $0 \leq m_1 < m_2 < ki$, $\mathcal{M}_{kQ,B-IBP}$ and $\mathcal{M}_{kQ,B-IBP}^d$ avoid memory contention. If $\left\| \frac{m_1 NL}{ki} \right\|_L \neq \left\| \frac{m_2 NL}{ki} \right\|_L$, where $0 \leq m_1 < m_2 < ki$, $\mathcal{M}_{\frac{i}{Q},block}$ and $\mathcal{M}_{\frac{i}{Q},block}^d$ avoid memory contention due to $\left\| \frac{m_1 NL}{ki} \right\|_L - \left\| \frac{m_2 NL}{ki} \right\|_L \Big|_{LQ/i} = 0$.

We apply both memory mapping functions to Π_{B-ibp} and Π_{B-ibp}^{-1} and have

$$\begin{aligned}
& \mathcal{M}_{ki,C-B-IBP} \left(\pi_{B-ibp}^{-1} \left(m \frac{NL}{ki} + n \right) \right) \\
= & \mathcal{M}_{kQ,B-IBP} \left(\pi_{B-ibp}^{-1} \left(\left\| m \frac{NL}{ki} + n \right\|_L L + \left\| m \frac{NL}{ki} + n \right\|_L \right) \right) \frac{i}{Q} \\
& + \mathcal{M}_{\frac{i}{Q},block} \left(\pi_{block}^{-1} \left(\left\| m \frac{NL}{ki} + n \right\|_L \right) \right), \\
= & \mathcal{M}_{kQ,B-IBP} \left(f_n^d \left(\left\| m \frac{NL}{ki} + n \right\|_L, \pi_{block}^{-1} \left(\left\| m \frac{NL}{ki} + n \right\|_L \right) \right) L + \right. \\
& \left. \pi_{block}^{-1} \left(\left\| m \frac{NL}{ki} + n \right\|_L \right) \right) \frac{i}{Q} + \mathcal{M}_{\frac{i}{Q},block} \left(\pi_{block}^{-1} \left(\left\| m \frac{NL}{ki} + n \right\|_L \right) \right), \quad (4.20)
\end{aligned}$$

$$\begin{aligned}
& \mathcal{M}_{ki,C-B-IBP}^d \left(\pi_{B-ibp} \left(m \frac{NL}{ki} + n \right) \right) \\
= & \mathcal{M}_{kQ,B-IBP}^d \left(\pi_{B-ibp} \left(\left\| m \frac{NL}{ki} + n \right\|_L L + \left\| m \frac{NL}{ki} + n \right\|_L \right) \right) \frac{i}{Q} \\
& + \mathcal{M}_{\frac{i}{Q},block}^d \left(\pi_{block} \left(\left\| m \frac{NL}{ki} + n \right\|_L \right) \right), \\
= & \mathcal{M}_{kQ,B-IBP}^d \left(f_n \left(\left\| m \frac{NL}{ki} + n \right\|_L, \pi_{block} \left(\left\| m \frac{NL}{ki} + n \right\|_L \right) \right) L + \right. \\
& \left. \pi_{block} \left(\left\| m \frac{NL}{ki} + n \right\|_L \right) \right) \frac{i}{Q} + \mathcal{M}_{\frac{i}{Q},block}^d \left(\pi_{block} \left(\left\| m \frac{NL}{ki} + n \right\|_L \right) \right). \quad (4.21)
\end{aligned}$$

If $\left\| \frac{m_1 NL}{ki} \right\|_L = \left\| \frac{m_2 NL}{ki} \right\|_L$, where $0 \leq m_1 < m_2 < ki$, $\mathcal{M}_{kQ,B-IBP}$ and $\mathcal{M}_{kQ,B-IBP}^d$ avoid memory contention. If $\left\| \frac{m_1 NL}{ki} \right\|_L \neq \left\| \frac{m_2 NL}{ki} \right\|_L$, where $0 \leq m_1 < m_2 < ki$, $\mathcal{M}_{\frac{i}{Q},block}$ and $\mathcal{M}_{\frac{i}{Q},block}^d$ avoid memory contention due to $\left\| \frac{m_1 NL}{ki} \right\|_L - \left\| \frac{m_2 NL}{ki} \right\|_L \Big|_{LQ/i} = 0$. ■

Theorem 4.7 introduces memory mapping functions for the B-IBP interleaver supporting generalized maximal contention-free property and the necessary condition is that the block interleaver is a generalized maximal contention-free interleaver. Therefore we can search existing good short block interleaver such as QPP or ARP to construct the B-IBP interleaver; the resultant distance property is generally good and generalized maximal contention-free is satisfied.

4.2.5 High-radix APP decoder and intra-block permutation

The high-radix APP decoder [16] improves turbo decoding throughput, network complexity and storage by paying the trellis complexity. The APP decoder processes multiple trellis segments or multiple information bits at each unit time to increase decoding throughput. Since the decoding throughput increases, less APP decoders can achieve the same decoding throughput comparing to the baseline radix-2 APP decoder and the associated network complexity is less. Less APP decoders also require less storage for state metrics, received samples and extrinsic information in the turbo decoder. However more trellis segments or information bits processed at each unit time induces exponential growth trellis complexity in the APP decoder. For example, 6 bits processed at one unit time implies $2^6 = 64$ edges coming out from one state and in total $64 \cdot |\Sigma|$ edges appears but 1 bit processed at one unit time means 2 edges emit from one state and in total $2 \cdot 6 \cdot |\Sigma| = 12 \cdot |\Sigma|$ edges come out, where $|\Sigma|$ is the number of total states. Therefore the high-radix APP decoder enlarges trellis complexity or routing complexity.

Take the radix-4 APP decoder [16] as an example to compare the routing complexity. Fig. 4.7 draws two trellises to compare both the radix-2 and radix-4 APP decoders. Fig. 4.7 (a) is the trellis composed of two trellis segments referring to Fig. 2.3 (c). The radix-2 APP decoder processes this trellis by two unit times. Fig. 4.7 (b) plots the merged trellis and in one unit time two bits are processed. If the parallelism degree 32 is necessary, a 32×32 network can be replaced by two 16×16 networks when the radix-4 APP decoder substitutes the radix-2 APP decoder. The associated network complexity decreases and the number of APP decoders decrease by two respectively.

In order to support the radix- 2^B APP decoder, the APP decoder has to access and write consecutive B information bits without memory contention. Suppose the block interleaver Π_{block} has length L and B is the factor of L , where the condition that B is the factor of L implies that the trellis segment does not change at the last for the APP decoder. We give a definition as follows.

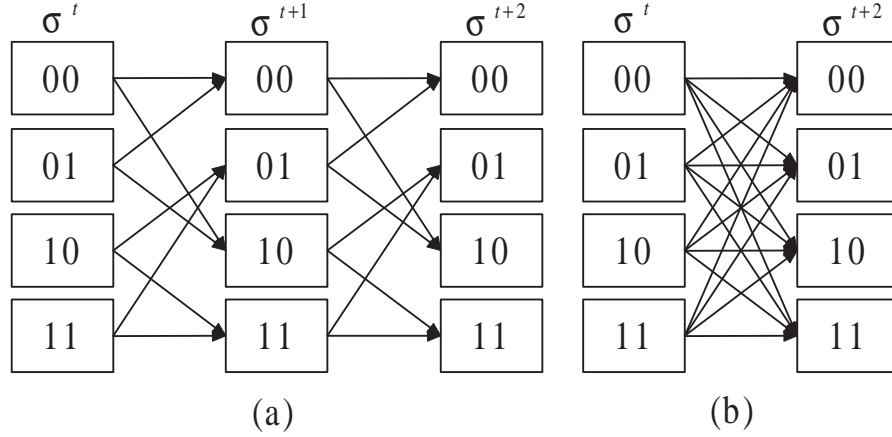


Figure 4.7: (a) Two connected trellis segments referring to Fig. 2.3 (c); (b) The merged trellis segment for the radix-4 APP decoder.

Definition 23 *If the block interleaver Π_{block} supports memory contention-free for the radix- 2^B APP decoder, there exist memory mapping functions $\mathcal{M}_{B,block}$ and $\mathcal{M}_{B,block}^d$ such that*

$$\mathcal{M}_{B,block}(iB + j) \neq \mathcal{M}_{B,block}(iB + k), \quad (4.22)$$

$$\mathcal{M}_{B,block}^d(iB + j) \neq \mathcal{M}_{B,block}^d(iB + k), \quad (4.23)$$

$$\mathcal{M}_{B,block}(\pi_{block}^{-1}(iB + j)) \neq \mathcal{M}_{B,block}(\pi_{block}^{-1}(iB + k)), \quad (4.24)$$

$$\mathcal{M}_{B,block}^d(\pi_{block}(iB + j)) \neq \mathcal{M}_{B,block}^d(\pi_{block}(iB + k)), \quad (4.25)$$

where $0 \leq i < \frac{L}{B}$, $0 \leq j < k < B$.

Recall the length- L ARP interleaver in eqn. (4.4) and the proof in *Theorem 4.5*, the permutation function moves elements in S_k to $S_{A'(k)}$, where $S_k = \{iC + k | 0 \leq i < \frac{L}{C}\}$, $0 \leq k < C$. Therefore, the ARP interleaver can support the radix- 2^C APP decoder by the memory mapping functions

$$\mathcal{M}_{C,block}(iC + j) = j, \quad (4.26)$$

$$\mathcal{M}_{C,block}^d(iC + j) = j. \quad (4.27)$$

The B-IBP interleaver supports the radix- 2^B APP decoder if the block interleaver

supports the radix- 2^B APP decoder. A theorem is given below.

Theorem 4.8 *If the block interleaver supports the radix- 2^B APP decoder, the associated B-IBP interleaver also supports the radix- 2^B APP decoder.*

Proof: The block interleaver supports the radix- 2^B APP decoder and there exists memory mapping functions $\mathcal{M}_{B,block}$ and $\mathcal{M}_{B,block}^d$ avoiding memory contention where B is the factor of L . Suppose there are N blocks for the B-IBP interleaver and *Theorem 4.6* shows that there exist memory mapping functions $\mathcal{M}_{k,B-IBP}$ and $\mathcal{M}_{k,B-IBP}^d$ avoiding memory contention where k is the factor of N . We construct a composite memory mapping functions $\mathcal{M}_{kB,C-B-IBP}$ and $\mathcal{M}_{kB,C-B-IBP}^d$ as

$$\mathcal{M}_{kB,C-B-IBP}(mL + n) = \mathcal{M}_{k,B-IBP}(mL + n)B + \mathcal{M}_{B,block}(n), \quad (4.28)$$

$$\mathcal{M}_{kB,C-B-IBP}^d(mL + n) = \mathcal{M}_{k,B-IBP}^d(mL + n)B + \mathcal{M}_{B,block}^d(n). \quad (4.29)$$

Then we have

$$\begin{aligned} & \mathcal{M}_{kB,C-B-IBP}(\pi_{B-ibp}^{-1}(mL + n)) \\ = & \mathcal{M}_{k,B-IBP}(\pi_{B-ibp}^{-1}(mL + n))B + \mathcal{M}_{B,block}(\pi_{block}^{-1}(n)), \end{aligned} \quad (4.30)$$

$$\begin{aligned} & \mathcal{M}_{kB,C-B-IBP}^d(\pi_{B-ibp}(mL + n)) \\ = & \mathcal{M}_{k,B-IBP}^d(\pi_{B-ibp}(mL + n))B + \mathcal{M}_{B,block}^d(\pi_{block}(n)). \end{aligned} \quad (4.31)$$

■

4.3 Network-oriented interleaver design

Network-oriented B-IBP design avoids the routing complexity and simplifies the control signalling for the parallel turbo decoder architecture. In general the network complexity is a critical issue for the parallel decoder architectures. Large network complexity decreases the density on the chip and increases the cost, e.g. LDPC code decoders [26, 62, 55]. Large network complexity also lengthens the routes on the chip. The longer

routs, the longer interconnection delay [61] as the process goes advance. The parallel turbo decoder also necessitates a network between APP decoders and memory banks, and the network complexity increases with decoding throughput. Therefore network complexity issue also occurs to the parallel turbo decoder. For the B-IBPTC decoder, the B-IBP generally configures the network routing and we can design the B-IBP based on a low complexity routing network to prevent the potential routing complexity.

4.3.1 Network-oriented B-IBP design

B-IBP generally determines the network configuration and intra-block permutation controls the memory addressing. When we consider a B-IBP interleaver with N blocks and turbo decoder composed of P APP decoders, there exists a $P \times N$ network between APP decoders and memory banks. We can choose any network with less complexity comparing to the fully-connected $P \times N$ network [35]; the resultant routing complexity decreases. Therefore the construction method is described in the following three steps.

1. Choose a $P \times N$ network bridge P APP decoders and N memory banks.
2. Select some routes on this network.
3. Arrange the selected routes as the B-IBP.

The following two subsections will provide two low complexity networks and we will discuss their features.

4.3.2 Butterfly network

An $N \times N$ Butterfly network [68, 67] is the simplest network with the edge complexity $2N \log N$ comparing to the fully-connected network with the edge complexity N^2 [35], where N is the power of 2. The number of network configurations is $2^{\frac{N}{2} \log_2 N}$. However this network results in the most routing congestions and therefore the network generally is not applicable for the parallel turbo decoder design as the interleaver is not an

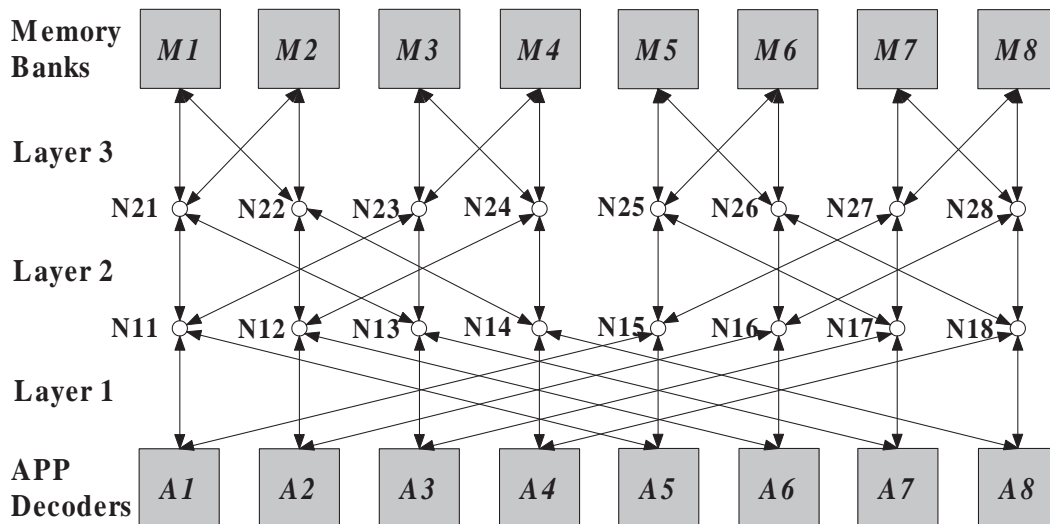


Figure 4.8: An example of butterfly network-oriented turbo code decoder architecture with parallelism degree 8.

option such as QPP interleaver. However the network-oriented design prevents routing congestions even if butterfly network is applied for the turbo decoder. Enjoying this low complexity network becomes possible.

Fig. 4.8 shows a turbo decoder architecture [112] composed of 8 APP decoders and 8 memory banks, and there is a three-stage butterfly network in between. We choose five network configurations:

- C_1 : $A_1 \rightarrow M_1$, $A_2 \rightarrow M_2$, $A_3 \rightarrow M_3$, $A_4 \rightarrow M_4$, $A_5 \rightarrow M_5$, $A_6 \rightarrow M_6$, $A_7 \rightarrow M_7$ and $A_8 \rightarrow M_8$.
- C_2 : $A_1 \rightarrow M_2$, $A_2 \rightarrow M_1$, $A_3 \rightarrow M_4$, $A_4 \rightarrow M_3$, $A_5 \rightarrow M_6$, $A_6 \rightarrow M_5$, $A_7 \rightarrow M_8$ and $A_8 \rightarrow M_7$.
- C_3 : $A_1 \rightarrow M_3$, $A_2 \rightarrow M_4$, $A_3 \rightarrow M_1$, $A_4 \rightarrow M_2$, $A_5 \rightarrow M_7$, $A_6 \rightarrow M_8$, $A_7 \rightarrow M_5$ and $A_8 \rightarrow M_6$.
- C_4 : $A_1 \rightarrow M_5$, $A_2 \rightarrow M_6$, $A_3 \rightarrow M_7$, $A_4 \rightarrow M_8$, $A_5 \rightarrow M_1$, $A_6 \rightarrow M_2$, $A_7 \rightarrow M_3$ and $A_8 \rightarrow M_4$.

- C_5 : $A1 \rightarrow M7$, $A2 \rightarrow M8$, $A3 \rightarrow M5$, $A4 \rightarrow M6$, $A5 \rightarrow M3$, $A6 \rightarrow M4$,
 $A7 \rightarrow M1$ and $A8 \rightarrow M2$.

We suppose the network configuration changing with the repeated order $\{C_1, C_2, C_3, C_4, C_5, C_1, C_2, \dots\}$ and this is the constructed B-IBP.

B controlling bits can configure a B -layer butterfly network instead of $B \cdot 2^{B-1}$ controlling bits. In Fig. 4.8 the butterfly network generally necessitates $3 \cdot 4 = 12$ controlling bits to configure the network with $3 \cdot 2^2$ crossbar switches. For the selected network configurations $\{C_1, C_2, C_3, C_4, C_5\}$, three bits can configure this 3-layer butterfly network due to each layer switched together; the configuration C_2 switches the layer 3, the configuration C_3 switches the layer 2, the configuration C_4 switches the layer 1, the configuration C_5 switches the layers 1 and 2. Therefore network oriented design can further take controlling signalling into account to simplify network configuration.

The B-IBP interleaver which supports butterfly network with the identical layer-switching is formulated as

$$\pi_{butterfly, B-ibp}(iL + j) = (i \oplus I(\pi_{block}(j)))L + \pi_{block}(j), \quad (4.32)$$

where $I(\cdot)$ corresponds to the control signalling sequence in which each symbol has bit width B , $0 \leq i < 2^B$ and \oplus is a width B exclusive-or operation.

If we want to decrease the parallelism degree to the factor R of N , we modify eqns. (4.9) and (4.10) to acquire

$$\mathcal{M}(jL + k) = \lfloor j \rfloor_{\frac{N}{R}}, \quad (4.33)$$

$$\mathcal{M}^d(jL + k) = \lfloor j \rfloor_{\frac{N}{R}}, \quad (4.34)$$

where R is the factor of N . Eqns. (4.11), and (4.12) become

$$\mathcal{M}(\pi_{B-ibp}^{-1}(jL + k)) = \lfloor f_n^d(j, k) \rfloor_{\frac{N}{R}}, \quad (4.35)$$

$$\mathcal{M}^d(\pi_{B-ibp}(jL + k)) = \lfloor f_n(j, \pi_{block}(k)) \rfloor_{\frac{N}{R}}. \quad (4.36)$$

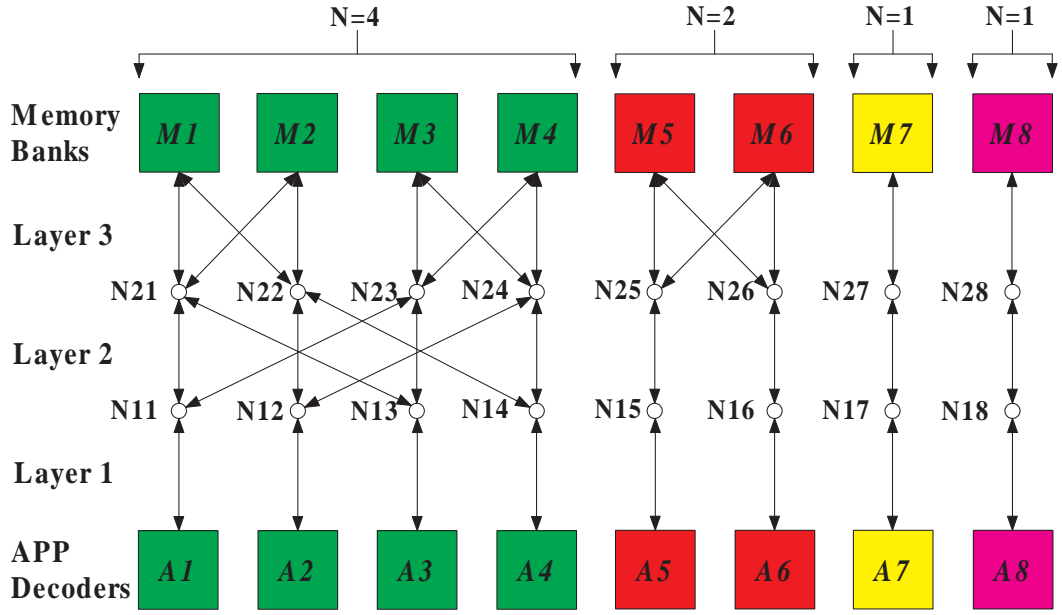


Figure 4.9: Multiple streams decoding.

One can find the memory mapping functions are not only regular but also simple.

Butterfly network-oriented turbo decoder supports multiple streams decoding which highly benefits uplink transmission. In downlink, multiple streams occur less for a mobile station. In uplink multiple users interact with a base station concurrently and multiple short length streams occur often. In order to achieve high throughput, the previous parallelization method with the sliding window APP algorithm [104] can be applied. However the window-based algorithm introduces a training window interval and outputting time interval which process no information and introduce extra latencies. Therefore the shorter the segment length, the lower the decoder throughput. Fig. 4.9 shows a sub-network configuration from Fig. 4.8; the turbo decoder with $N = 8$ APP decoders becomes four turbo decoders with $N = 4, 2, 1, 1$ APP decoders; the decoder can decode multiple short streams separately without extra storage for the extrinsic information and received samples, and the portion of wasted time intervals becomes less. Therefore butterfly network-oriented turbo decoder benefits uplink decoding throughput.

An example is given in analyzing the throughput under this parallel multiple streams

Table 4.1: Throughput and latency analysis for 8 APP decoders

$f_{c[MHz]} :$ 100MHz	8 streams and 8 APP decoders			
	Parallel decoding 8 streams with P=1		Serial decoding 1 stream with P=8	
	Throughput [Mbps]	Latency [μs]	Throughput [Mbps]	Latency [μs]
$K=512$	44.44	92.16	25	20.48
$K=256$	40	51.2	16.67	15.36
$K=128$	33.33	30.72	10	12.8

decoding and serial single stream decoding. We have parameters as follows.

- I_{\max} : number of maximum iterations
- K : length of processed data
- W : training window
- P : number of APP decoders processing one stream
- $f_{c[MHz]}$: operating frequency

The formula of latency and throughput are

$$Latency_{[\mu s]} = \frac{2I_{\max} \left(\lceil \frac{K}{P} \rceil + 2W \right)}{f_{c[MHz]}} \quad (4.37)$$

$$Throughput_{[Mbps]} = \frac{K f_{c[MHz]}}{2I_{\max} \left(\lceil \frac{K}{P} \rceil + 2W \right)} \quad (4.38)$$

Tables 4.1 and 4.2 provide throughput and latency analysis for multiple streams and single stream decoding. The training window W is 32 and the maximum iteration I_{\max} is 8. Operating frequency is 100 MHz. The throughput of multiple streams decoding is higher than that of single stream decoding and the difference becomes larger as the data length goes down. Obviously, the training window induces the decrease of throughput and shorter data length suffers more.

The butterfly network-oriented B-IBP interleaver imposes the restriction on the number of blocks which is the power of 2. If the necessary number of APP decoders is not the

Table 4.2: Throughput and latency analysis for 16 APP decoders

$f_c [MHz] :$ 100MHz	16 streams and 16 APP decoders			
	Parallel decoding 16 streams with P=1		Serial decoding 1 stream with P=16	
	Throughput [Mbps]	Latency [μs]	Throughput [Mbps]	Latency [μs]
$K=512$	88.89	92.16	33.33	15.36
$K=256$	80	51.2	20	12.8
$K=128$	66.67	30.72	11.11	11.52

power of 2, this restricts the turbo decoder design. For example: the desired throughput only necessitates 17 APP decoders, and either extra 15 APP decoders or 15 extra memory banks with complex memory addressing is necessary. One solution is to apply the graph using to describe Winograd transform [110]. Another solution is to apply the barrel shifter network to retain low complexity network configuration.

4.3.3 Barrel shifter network

Barrel shifter network [111] is also a low complexity routing network and fashion QC-LDPC [43] also designs the parity check matrix based on this network. The objective of this network is to cyclically shift bits and this network switches each layer by left or right $1, 2, \dots, 2^B$, where B is the total number of layers. The network requires $B = \lceil \log_2 N \rceil$ controlling bits to determine the network configuration due to each layer switching together. The number of N does not restrict to 2^B and this provides a flexibility as designing the B-IBP. If $N = 2^B$, the network has the same edge complexity comparing to the butterfly network. However the identical layer switching also imposes a stringent constraint in constructing the B-IBP. The maximum possible number of network configuration is restricted to 2^B which is much less than that of butterfly network $2^{\frac{N}{2} \log_2 N}$. This may incur performance degradation as creating the associated B-IBP interleaver. By the way this network, the same as butterfly network, is generally not applicable for the parallel turbo decoder if the interleaver is not an option.

Fig. 4.10 shows a turbo decoder architecture which applies barrel shifter network

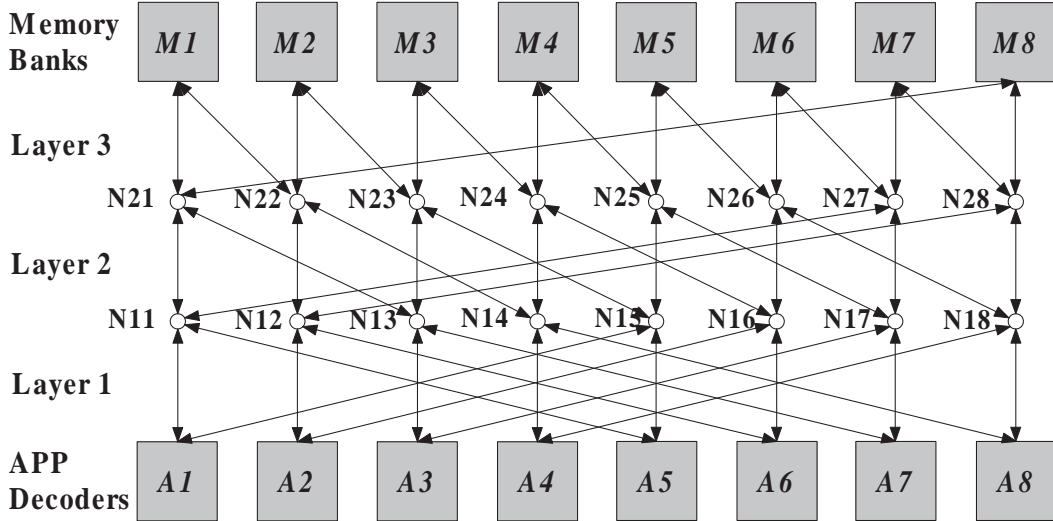


Figure 4.10: An example of barrier shifter network-oriented turbo code decoder architecture with parallelism degree 8.

to interconnect both APP decoders and memory banks. The network is composed of three layers. Layer 1 switches routes right by 0 or 4, Layer 2 switches routes right by 0 or 2 and Layer 3 switches routes right by 0 or 1. If we want to route A1, A2, A3, A4, A5, A6, A7, A8 to M6, M7, M8, M1, M2, M3, M4, M5, the network switches layers 1 and 3 by 4 and 1 and layer 2 by 0.

Since the operation of barrel shifter network is cyclic shift, the B-IBP interleaver is described as

$$\pi_{butterfly, B-ibp}(iL + j) = |i + I(\pi_{block}(j))|_N \cdot L + \pi_{block}(j), \quad (4.39)$$

where N is the number of blocks and $I(\cdot)$ corresponds to the control signalling sequence in which each symbol has bit width $\lceil \log N \rceil$.

4.4 B-IBP interleaver supports variable information length

B-IBP interleaver imposes a constraint on the input sequence length which is the multiple of block length. WiMAX [56] pads dummy bits to resolve this problem but

extra dummy bits decrease bandwidth efficiency. Information bit shortening [109] and interleaver pruning [32] are both applicable for turbo code to match various information sequence lengths. As for maintaining identical code rate, the shortening requires puncturing extra code bits but it may pay at most 2 to the decrease of the minimum distance. Interleaver pruning shortens interleaver by removing permutation positions but this may cause serious minimum distance decrease due to the change of permutation table. The following subsections will describe and compare both strategies.

4.4.1 Shortening and puncturing

The shortening and puncturing support various lengths of input data sequence without sacrificing code rate and the spaced shortening assigning algorithm reduces decoder control complexity without significant performance loss. The shortening is applicable to increase supportable length for input data sequence by removing the dummy bits to fit the desired interleaver length before transmission. The puncturing further removes the parity bits corresponding to the dummy bits and code rate maintains the same.

The shortening and puncturing strategy decreases the distance by at most 2 for each shortened position. The shortening strategy assigns a dummy bit before encoding. After encoding it removes the dummy bit and the associated parity bits. Because the parity bit is removed, the shortened codeword weight decreases by at most 2.

However, the shortening induces extra complexity in APP decoder design and the extra shortening cycle is necessary. In order to maintain multiple processors processing shortening cycle as possible at the same time, the shortening positions have to be spaced by an interval. The space also spreads the influence of shortening positions and keeps the performance without significant degradation induced by the puncturing.

Shortening position assigning algorithm is characterized by two stages. The first stage calculates the shortening width $W_{shorten}$. The second stage assigns the shortening positions. There are $N_{shorten} = K_{B-IBP} - K$ shortening positions to be shortened, where

K is input information length and K_{B-IBP} is the length of B-IBP interleaver. The maximum number of shortened positions for all blocks is $N_{L,shorten} = \lceil \frac{N_{shorten}}{N} \rceil$ and there are $N_{N,MAX} = \lfloor N_{shorten} \rfloor / N$ blocks shortening $N_{L,shorten}$ positions and the associated shortening width is $W_{shorten} = \lfloor \frac{L}{N_{L,shorten}} \rfloor$. The second stage determines the shortening positions. The first $N_{N,MAX}$ blocks shorten symbols at $\{0, W_{shorten}, 2W_{shorten}, \dots, (N_{L,shorten} - 1)W_{shorten}\}$. The following $N - N_{N,MAX}$ blocks shorten symbols at $\{0, W_{shorten}, 2W_{shorten}, \dots, (N_{L,shorten} - 2)W_{shorten}\}$. This assigning algorithm promises these APP decoders manipulate the shortening cycles at the same time as much as possible when the number of APP decoders are the factor of N while the shortened positions are spread to the farthest.

4.4.2 Pruning

Interleaver pruning [32] removes the permutation entries to shorten the interleaver. Assume there exists an interleaver with permutation function Π . We prune one position j and the resultant pruned interleaver Π_{prune} has the function

$$\pi_{prune}(i) = \begin{cases} \pi(i) & , i < j \text{ and } \pi(i) < \pi(j) \\ \pi(i) - 1 & , i < j \text{ and } \pi(i) > \pi(j) \\ \pi(i+1) & , i > j \text{ and } \pi(i+1) < \pi(j) \\ \pi(i+1) - 1 & , i > j \text{ and } \pi(i+1) > \pi(j) \end{cases} \quad (4.40)$$

We can apply this formula recursively to create an interleaver with any length shorter than the length of Π .

4.4.3 Comparison between shortening and pruning

Turbo code error events are highly correlated with the non-zero element coordinates of an input sequence. The component code of turbo code is a recursive convolutional code and low weight output sequences are usually corresponding to some special patterns. Fig. 4.11 shows two important error events, the weight-2 and weight-4 error events. A coordinate pair generates a finite weight codeword if coordinates difference is

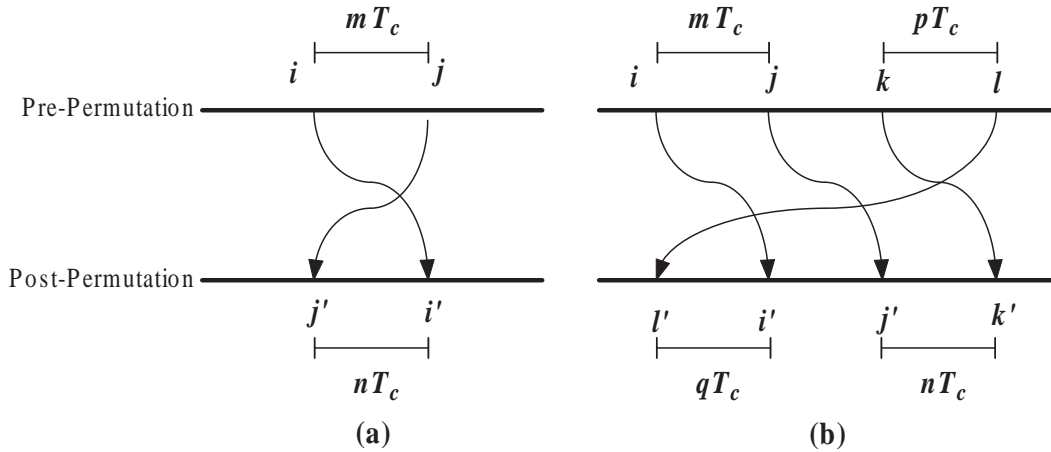


Figure 4.11: Weight-2 and 4 error events for a turbo code.

multiple of T_c for both pre-permutation and post-permutation where T_c is the period of the component code. A good interleaver always promises large m and n in Fig. 4.11 (a) and m, n, p and q in Fig. 4.11 (b) and the corresponding codeword weights are large. A good interleaver generally has large distance and shortening strategy does not influence performance significantly if the shortening positions are not close as our algorithm.

The pruning strategy may significantly decrease the minimum distance of the turbo code with the pruned interleaver comparing to a mother interleaver. The strategy assigns a dummy bit and skips the bit when encoding. Therefore a pruned interleaver is completely different to the mother interleaver. Fig. 4.12 (b) shows an example. The pruning shifts $j'' = j + 1$ to j due one position is pruned before j'' and after i . Because the difference between i and j'' is $T_c + 1$, it does not generate low weight codeword. However the difference between i and j' is T_c and a low weight codeword occurs for the turbo code applying the pruned interleaver because the difference between i' and j' is also T_c . Even if a mother interleaver has an outstanding distance property, the pruned interleaver has completely different nature to the mother interleaver. Furthermore the more bits pruned, the more kinds of interleavers generated.

Interleaver pruning [32] has been applied in 3GPP Rel'99 and Rel'6 [1, 2] turbo code

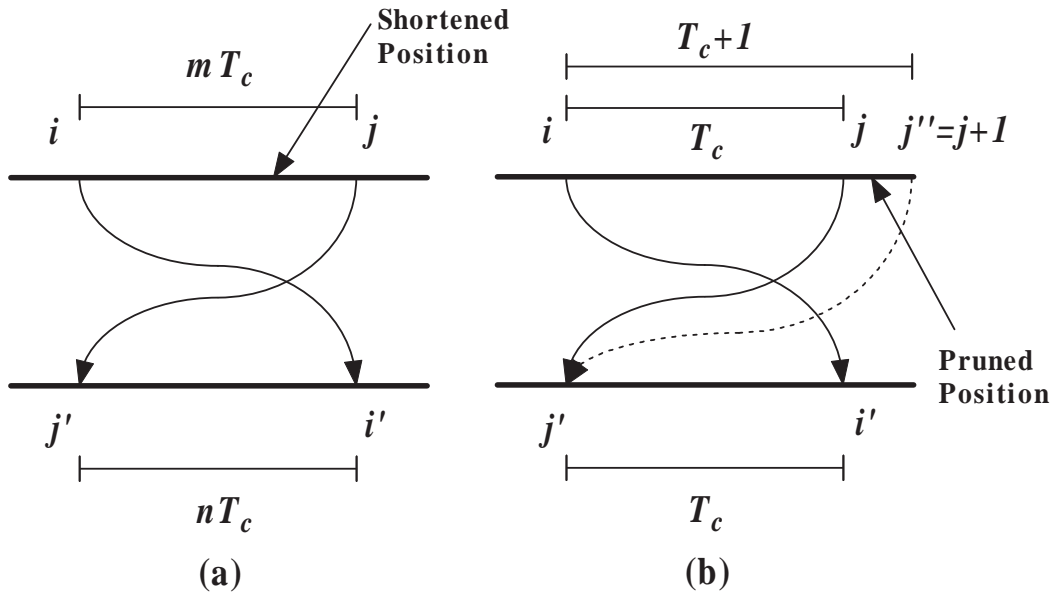


Figure 4.12: Influence of the weight-2 error events for both shortening and pruning strategy.

to create various interleaver lengths ranging from 40 to 5114 bits. This strategy provides various interleavers but it induces low weight codeword for turbo code with the pruned interleaver. The performance is not stable due to the pruning strategy. This explains the 3GPP Rel'99 and Rel'6 [1, 2] turbo coding possessing higher error floor for some interleaver lengths. Our simulation results evidence this.

The shortening and puncturing strategy requires less efforts in searching wide range of interleavers than the pruning strategy. The shortening only decreases the distance by at most 2 for each shortened bit. We can simply verify the case shortening most bits. If the performance does not degrade obviously, we accept the interleaver. However, the pruning strategy generates different nature interleavers. If at most 30 positions have to be pruned, 31 kinds of interleavers include the mother interleaver have to be verified. Therefore shortening and puncturing strategy simplifies our verification flow in searching interleavers within a wide range.

Table 4.3: Parallelism degree corresponding to various data lengths K and the supported number of interleavers.

K	N	Number of interleavers
$40 \leq K \leq 200$	1	69
$201 \leq K \leq 320$	2	26
$321 \leq K \leq 960$	8	34
$961 \leq K \leq 2560$	16	43
$2561 \leq K \leq 6144$	32	48

4.5 An interleaver design

An example is provided in this section. The interleaver length ranges from 40 to 6144 bits. The block interleaver supports generalized maximal contention-free and the permutation function can be generated on-fly for both interleaving and de-interleaving. Furthermore the radix-4 APP decoder [16] is applicable. The B-IBP supports butterfly-network and controlling signals can be generated by shift register. Our simulation results also show that the performance is not inferior to the QPP interleaver which has been defined in the 3GPP LTE [3]. The required storage for the parameters is also less than that for the QPP interleaver.

4.5.1 Interleaver description

The interleaver Π_{B-IPBI} is shown in eqn. (4.41), where $0 \leq i < K_{B-IBP}$, $I(\cdot)$ is the IBP sequence and the IBP period T is defined in Table 4.4. The block interleaver shown in eqn. (4.42) is double prime interleaver Π_{DP} composed of two prime interleavers permuting odd and even ordinals individually. This interleaver is characterized by two parameters p and s , the values associated various block lengths L are shown in Table 4.5. This formula matches eqn. (4.39) and simple butterfly network is applicable for the parallel turbo decoder.

$$\pi_{B-IPBI}(iL + j) = (i \oplus I(|\pi_{DP}(j)|_T)) \cdot L + \pi_{DP}(j). \quad (4.41)$$

Table 4.4: Butterfly B-IBP sequences and the corresponding generator polynomials

N	T	$I(\cdot)$	Polynomial
1	1	{0}	No
2	2	{0,1}	No
8	8	{0,1,2,4,3,6,7,5}	$\text{Poly}_{IBP}(8) = 11 : x^3 + x + 1$
16	16	{0,1,2,4,8,3,6,12,11, 5,10,7,14,15,13,9}	$\text{Poly}_{IBP}(16) = 19 : x^4 + x + 1$
32	31	{1,2,4,8,16,5,10,20,13,26,17,7, 14,28,29,31,27,19,3,6,12,24,21, 15,30,25,23,11,22,9,18}	$\text{Poly}_{IBP}(32) = 37 : x^5 + x^2 + 1$

$$\pi_{DP}(i) = \begin{cases} \left(\left\lfloor \frac{i}{2} \right\rfloor \times p \Big|_{\frac{L}{2}} \right) \times 2 & , i \text{ is even} \\ \left(\left\lfloor \frac{i}{2} \right\rfloor \times p + s \Big|_{\frac{L}{2}} \right) \times 2 + 1 & , i \text{ is odd} \end{cases} . \quad (4.42)$$

The inter-block permutation sequences shown in Table 4.3 can be generated by shift register which is friendly to implementation. Since these sequences can be generated by shift register and the corresponding storage can be reduced. The initial values for these sequences are set to $I(0) = 0$ and $I(1) = 1$ for $N = 1, 2, 8, 16$ and $I(0) = 1$ for $N = 31$. The rest numbers are generated by eqn. (4.43), where $\text{Poly}_{IBP}(N)$ is shown in Table 4.4.

$$I(i) = \begin{cases} I(i-1) \lll 1 & , I(i-1) < N/2 \\ (I(i-1) \lll 1) \oplus \text{Poly}_{IBP}(N) & , I(i-1) \geq N/2 \end{cases} . \quad (4.43)$$

4.5.2 Comparison to 3GPP LTE QPP

3GPP LTE [3] adopts QPP interleaver [90] for turbo code. The interleaver length ranges from 40 to 6144 bits and this is the same as our design. There are 188 kinds of interleavers and each interleaver adopts two parameters. Each parameter requires 10 bits to store. Including interleaver length, the interleaver table necessitates $188 \cdot 3 \cdot 10 = 5760$ bits. The turbo code applying this interleaver generally necessitates fully-connected network to bridge APP decoders and memory banks. The network configuration requires to resolve the equation (4.6) to acquire the network control signalling.

Table 4.5: Double prime interleaver parameters

L	p	s	L	p	s	L	p	s	L	p	s	L	p	s	L	p	s
40	7	3	44	5	13	46	17	8	48	17	12	50	11	5	52	3	12
54	5	13	58	16	11	60	23	11	62	26	9	64	5	18	66	5	13
68	3	17	72	25	18	74	10	8	76	29	14	78	22	30	80	23	25
82	12	18	86	10	17	88	41	22	90	31	19	92	31	23	94	6	14
96	43	21	100	11	25	102	32	17	104	31	24	106	44	22	108	17	26
110	16	27	114	16	27	116	47	7	118	43	28	120	11	29	122	18	34
124	45	29	128	17	24	130	11	29	132	47	27	134	6	32	136	21	35
138	40	45	142	29	36	144	11	29	146	47	34	148	51	35	150	11	16
152	31	35	156	55	28	158	28	34	160	47	39	162	44	62	164	13	31
166	34	40	170	14	35	172	51	39	174	53	42	176	35	44	178	28	42
180	17	44	184	19	46	186	38	45	188	25	39	190	37	43	192	17	40
194	8	45	198	31	48	200	41	50									

Our design provides 220 kinds of interleavers. The parameters for double prime interleaver table, IBP generator polynomials and interleaver ranges necessitate storage. Double prime interleaver desires two parameters and each parameter only requires 6 bits. The interleaver length requires 8 bits. IBP sequence requires 5 bits. Interleaver range requires 13 bits for each length. In total, this design requires $69 \cdot 2 \cdot 6 + 69 \cdot 8 + 3 \cdot 5 + 6 \cdot 13 = 1473$ bits. This design further supports butterfly network and the control signalling can be generated by shift register. Our design also supports generalized maximal contention-free as the QPP interleaver. Our simulation results shown in Section 4.7 indicate that our design outperforms the QPP interleaver in many cases. Therefore no matter in hardware complexity or performance, our design is better than the 3GPP LTE QPP interleaver.

4.6 Implementation

This implementation [112] follows the structure proposed in Section 4.5 and suits for the butterfly network architecture. The interleaver length is 4096 bits and the associated B-IBP interleaver is composed of $N = 32$ length-128 blocks. Each block applies the double prime interleaver with parameters $(p, s) = (15, 23)$. The IBP sequence is $\{0, 1, 2, 4, 8, 16, 5, 10, 20, 13, 26, 17, 7, 14, 28, 29, 31, 27, 19, 3, 6, 12, 24, 21, 15, 30, 25, 23, 11, 22,$

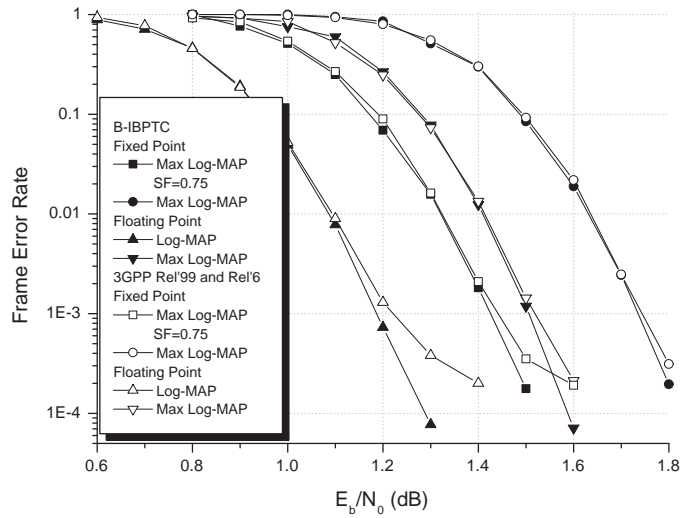


Figure 4.13: Frame error rate comparison for our implementation.

9, 18}. The component code is $G(D) = \left[1 \frac{1+D+D^3}{1+D^2+D^3} \right]$ which is identical to that defined in the 3GPP [1, 2, 3] but each block is separate tail-biting encoded. This implementation applies MAX Log-MAP with scaling factor 0.75 [73]. Fig. 4.13 shows the performance comparison between our B-IBPTC and 3GPP turbo code with 8 iterations. Our implementation after fixed point simulation is superior to the 3GPP code applying the maximum log-MAP algorithm with the scaling factor in fixed point by 0.1dB and inferior to the 3GPP code applying the log-MAP algorithm in floating point by 0.1dB at frame error rate (FER) 2×10^{-4} .

The decoder chip [112] is fabricated with a 0.13 μm 1P8M CMOS technology, and the die photo is shown in Fig. 4.14. The core area occupies 17.8 mm^2 with 2.67M gates count, including the 3.33 mm^2 memory block. The chip achieves maximum 160Mb/s with power consumption 275 mW at 1.32 V supply and operating frequency 80MHz and the associated energy efficiency is 0.22 nJ/b/iter at 8 iterations. Table 4.6 compares our result with other state-of-the-arts. Our design possesses the most superior energy efficiency [16, 20, 101].

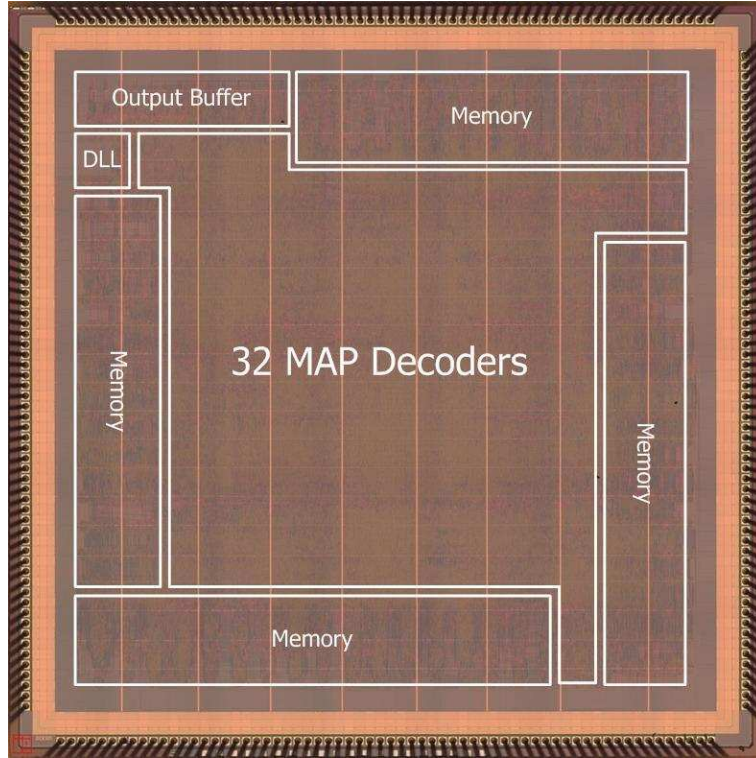


Figure 4.14: Chip photo [112].

4.7 Simulation results

We compare the performance based on the 3GPP defined turbo code with $G(D) = \left[1 \frac{1+D+D^3}{1+D^2+D^3}\right]$. The first part concerns the performance of our design proposed in Section 4.5. We compare our design with the 3GPP Rel'99 and Rel'6 [1, 2] and the 3GPP LTE QPP [3]. Then we evaluate the performance of our shortening and puncturing algorithm. At last the effect of separate encoding of various code rates is shown.

4.7.1 The interleaver design

This part simulates the turbo code with code rate=1/3 while the both convolutional code is tail-padded. All cases apply Max Log-MAP decoding algorithm with 8 iterations. AWGN channel is the simulation scenario. The compared interleaver lengths range from 40 to 6144 bits.

Figs. 4.15 and 4.16 plot the curves of the required E_b/N_0 for various error rate

Table 4.6: Comparison to different turbo decoder designs

	B-IBPTC [112]	[16]	[20]	[101]
Technology	0.13 μm	0.18 μm	0.18 μm	0.13 μm
Core Area	17.81 mm^2	14.5 mm^2	7.16 mm^2	10 mm^2
Block Size	4096	5114	384	2048
Iterations	8	6	4.43	5
Frequency	80MHz	145MHz	160MHz	352MHz
Throughput	160Mbps	24Mbps	71.7Mbps	352Mbps
Power	275mW	1450mW	N/A	2464mW
Energy Efficiency	0.22 nJ/b/iter	10.0 nJ/b/iter	2.19 nJ/b/iter	1.4 nJ/b/iter

performance to compare the 3GPP Rel'99 and Rel'6 turbo code [1, 2] with our design. Our design outperforms the 3GPP Rel'99 and Rel'6. The performance gain is significant especially for long interleaver length and FER=10⁻⁴. Our design provides stable performance for the entire range.

As our prediction in the comparison between interleaver pruning and shortening and puncturing. The pruning strategy generates different nature interleavers to the mother interleavers and may introduce low weight codeword for the turbo code applying the pruned interleaver. The performance results evidence that the pruning is not a good strategy to support variable input length and deteriorates the performance significantly.

Figs. 4.17 and 4.18 plot the curves of the required E_b/N_0 for various error rate performance to compare the 3GPP LTE QPP turbo code [3] with our design. Our design outperforms the 3GPP LTE for some cases. B-IBP interleaver has large degree of freedom and we can adjust the B-IBP or the block interleaver to reach the better distance spectrum and avoid the weakness for some interleaver length. However the QPP only requires two parameters to construct an interleaver which supports generalized maximal contention-free property, this also imposes limit to the distance property. Reference [93] has shown that the maximum achievable minimum distance for the interleaver length $2 \leq K \leq 4096$, and there are many cases the minimum weight are upper-bounded by 32. Our simulation results show that there are some spikes for the 3GPP LTE curves

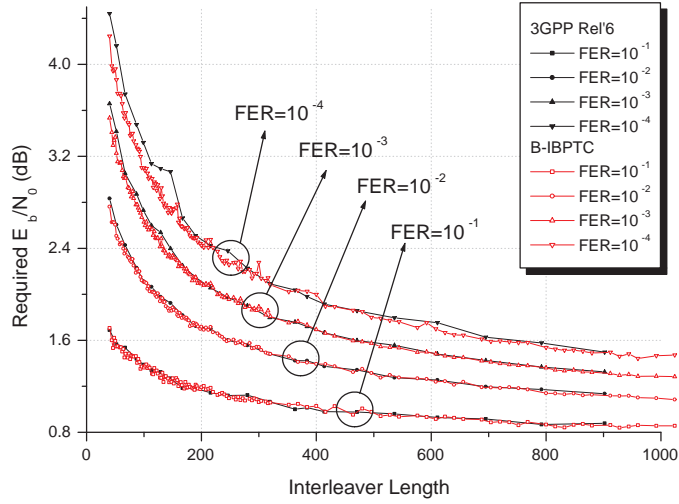


Figure 4.15: The B-IBP interleaver vs. 3GPP Rel'6 interleaver with the length ranging from 40 to 1000 bits.

and these results reflect this fact.

4.7.2 Shortening and puncturing

This part evaluates the performance of our proposed shortening and puncturing algorithm. We still simulate the turbo code with code rate=1/3 while the both convolutional codes are tail-padded. All cases apply Max Log-MAP decoding algorithm with 8 iterations. AWGN channel is the simulation scenario. Our design in Section 4.5 is applied and 219 cases are compared except for the length 40. The number of shortened bits is at most to 127 bits. Figs. 4.17 and 4.18 plot the curves of the required E_b/N_0 for various error rate performance and the performance is very similar for all cases. Although the shortening and puncturing may decrease the codeword weight by at most 2 for one shortened bit but the performance does not degrade significantly in our simulation even though 127 bits are shortened. Our spaced shortening position assigning algorithm not only favors the implementation but also retains the performance.

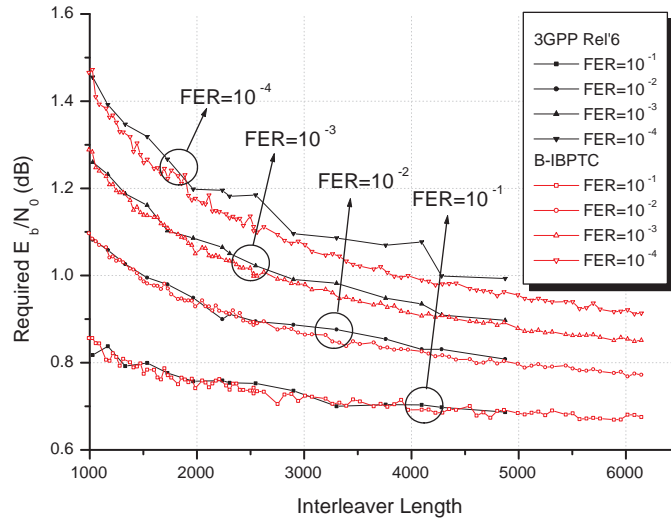


Figure 4.16: The B-IBP interleaver vs. 3GPP Rel'6 interleaver with the length ranging from 1000 to 6144 bits.

4.7.3 Separate and continuous encoding

The performance of the separate and continuous encoding is our last concern. We simulate the turbo code with code rate=1/2 and 3/4 while the tail-biting is applied. All cases apply linear Log-MAP decoding algorithm with 8 iterations. AWGN channel is the simulation environment. The compared interleaver lengths are 256, 512, 1024, 2046 and 4096 bits while the length-64 and length-128 block interleavers apply two parameters $(p, s) = (5, 32)$ and $(p, s) = (47, 7)$, respectively. When the code rate=1/2 shown in Fig. 4.21, the separate encoding outperforms the continuous encoding and it also implies the separate encoding enhances the distance property. When the code rate=3/4 shown in Fig. 4.22, the continuous encoding outperforms the separate encoding. The short block length and component code with code rate=6/7 induce low extrinsic information absolute value and the performance degrades. As for interleaver length=1024 bits and code rate=3/4, the separate encoding outperforms the continuous encoding when $E_b/N_0 > 3.5\text{dB}$. It means that the separate encoding has potential to provide the better distance

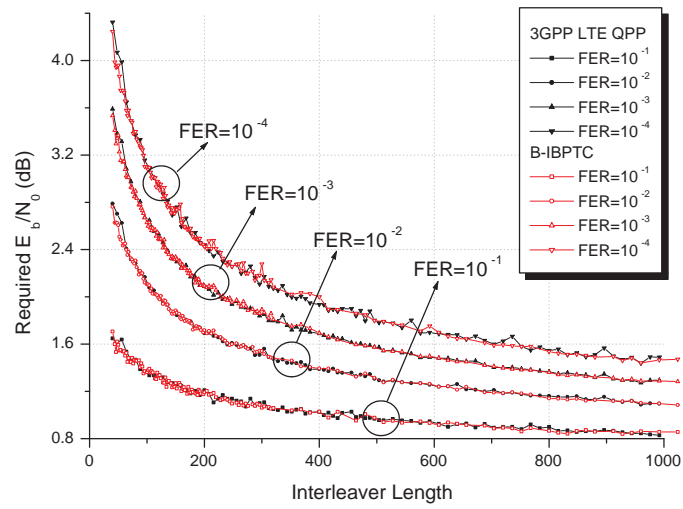


Figure 4.17: The B-IBP interleaver vs. 3GPP LTE QPP interleaver with the length ranging from 40 to 1000 bits.

property even for the high code rate.



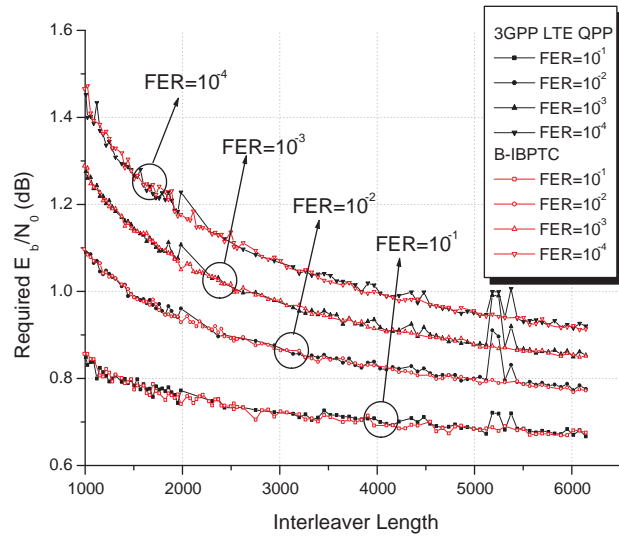


Figure 4.18: The B-IBP interleaver vs. 3GPP LTE QPP interleaver with the length ranging from 1000 to 6144 bits.

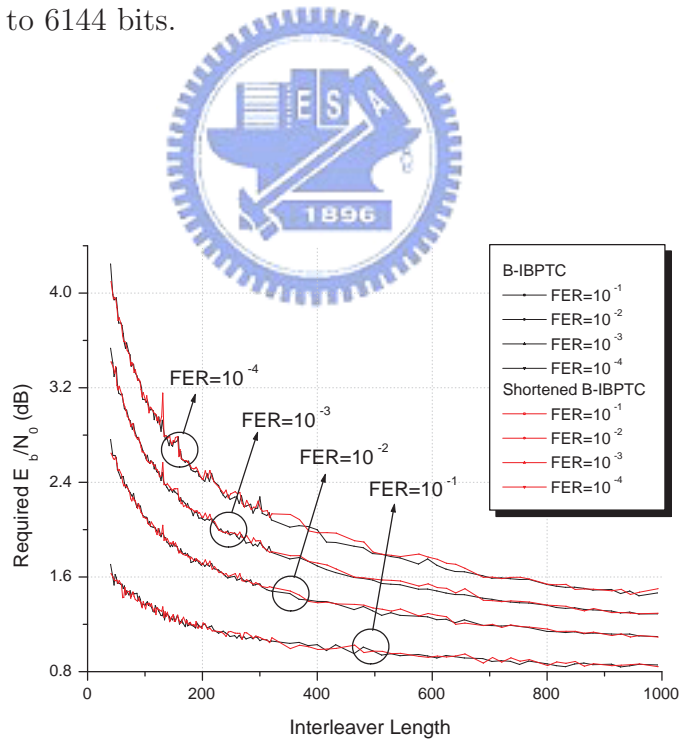


Figure 4.19: The shortening and puncturing effect for the B-IBP interleaver with the length ranging from 40 to 1000 bits.

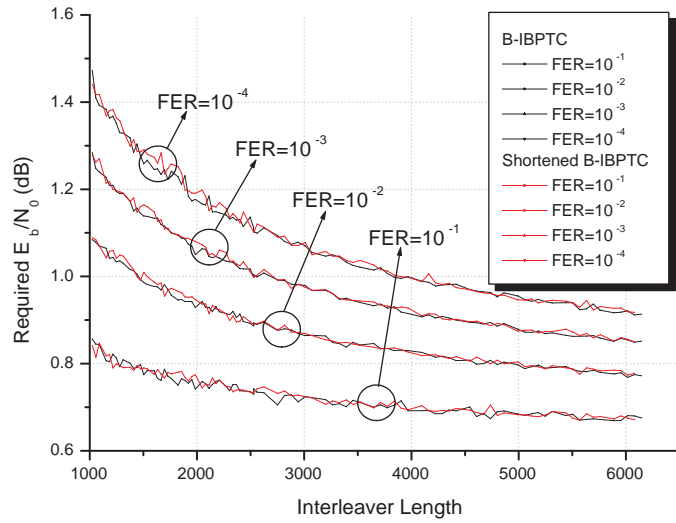


Figure 4.20: The shortening and puncturing effect for the B-IBP interleaver with the length ranging from 1000 to 6144 bits.

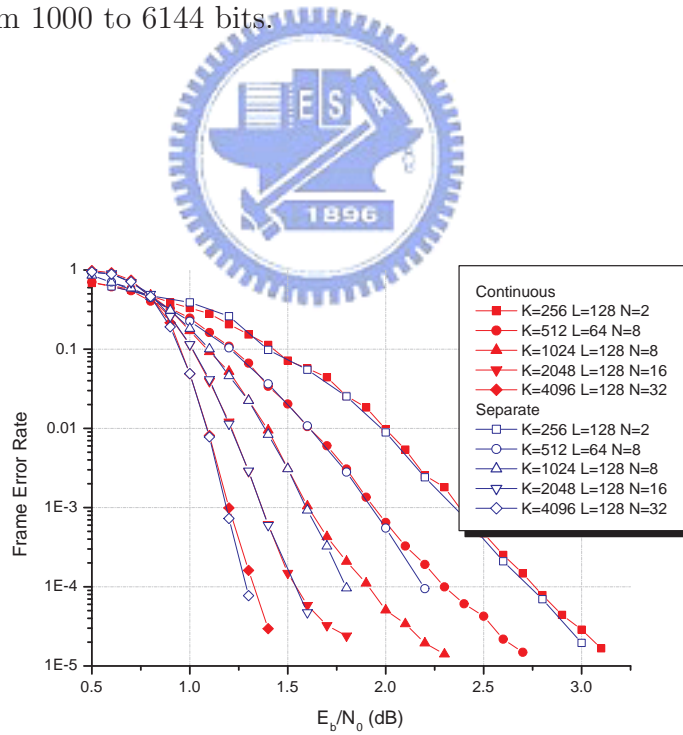


Figure 4.21: The comparison between the separate and continuous encoding for code rate 1/2.

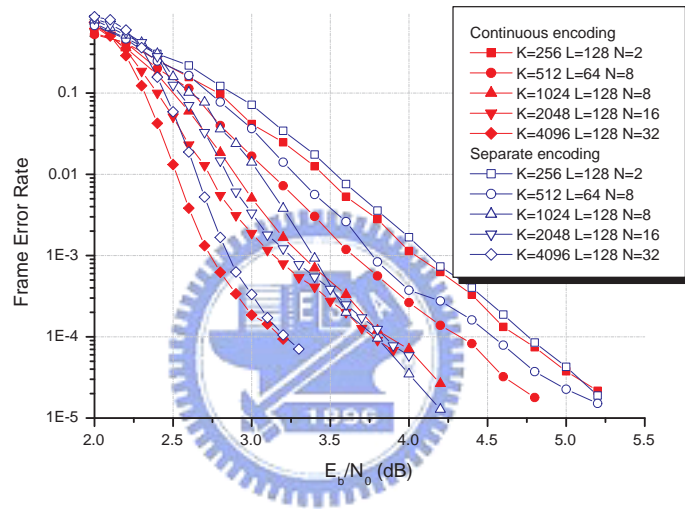


Figure 4.22: The comparison between the separate and continuous encoding for code rate 3/4.

Chapter 5

Stream-oriented inter-block permutation interleaver

stream-oriented IBP interleaver is designed for indefinite information length, and the associated stream-oriented IBPTC (S-IBPTC) suits for high throughput pipeline decoding architecture and outperforms classic TC under the same interleaver delay. Pipeline decoder is constructed by multiple APP decoders. These APP decoders serially decode codeword sequences and the interleaver delay determines the latency between two adjacent APP decoders. Since multiple APP decoders process at the same time, high throughput decoding is achievable. For example: [101] applies 10 APP decoders to achieve 352Mb/s throughput and interleaver delay is 2048 bits or $5.82\mu s$. However, these APP decoders decode these classic TCs independently and the performance is limited by the length of each codeword. In order to improve the performance, we construct a stream-oriented IBP (S-IBP) interleaver with the same interleaver delay. Due to the extra relation between neighboring blocks, information can be passed to the neighboring blocks in each decoding iteration. The more decoding iteration, the more information gathered, the better performance. By the way, we can apply a good block interleaver for the intra-block permutation and the resultant S-IBP interleaver would have good distance properties. Therefore, S-IBPTC has better performance and distance properties than classic TC under the same interleaver delay while we can enjoy the high throughput decoding coming from the pipeline architecture.

5.1 Stream-oriented IBP interleaver and the associated encoding storage

An S-IBP interleaver $\Pi_{S-ibp} = \Pi_{S-inter} \circ \Pi_{intra}$ is a sub-class of the IBP interleaver described in Section 3.2 and the interleaving and deinterleaving rules are formulated as

$$\pi_{S-ibp}(i) = f_n(|i|_L, \pi_{block}(|i|_L)) \cdot L + f_b(\pi_{block}(|i|_L)) \quad (5.1)$$

$$\pi_{S-ibp}^{-1}(i) = f_n^d(|i|_L, |i|_L) \cdot L + f_b^d(\pi_{block}^{-1}(|i|_L)). \quad (5.2)$$

The interleaver confines symbols in each block only permuted with that in at most priori S_b and successive S_f neighboring blocks; where S_b and S_f are backward and forward spans, i.e. $\|i\|_L - S_b \leq f_n(\|i\|_L, |i|_L) \leq \|i\|_L + S_f$ and $\|i\|_L - S_f \leq f_n^d(\|i\|_L, |i|_L) \leq \|i\|_L + S_b$. If $S = S_b = S_f$, Π_{S-ibp} is a symmetric S-IBP, which is of our main interest, and S is S-IBP interleaver span. These spans determine interleaver delay, encoding and decoding latencies.

We describe the relation between interleaver (deinterleaver) delay and these spans as follows. Denote the interleaver delay and deinterleaver delay by D_i and D_d , where

$$D_i = \max_k \{k - \pi(k)\}, \quad D_d = \max_k \{k - \pi^{-1}(k)\}. \quad (5.3)$$

From eqns. (5.1) and (5.2), the interleaver and deinterleaver delays of a Π_{S-ibp} are bounded by

$$\begin{aligned} D_{i,S-IBP} &= \max_k \{k - \pi_{S-ibp}(k)\} = \max_k \{k - f_n(\|k\|_L, |k|_L) \cdot L - f_b(|k|_L)\} \\ &= \max_k \{k - (|k|_L - S_b)L - f_b(k)\} \leq (S_b + 1)L \end{aligned} \quad (5.4)$$

and

$$\begin{aligned} D_{d,S-IBP} &= \max_k \{k - \pi_{S-ibp}^{-1}(k)\} = \max_k \{k - f_n^d(\|k\|_L, |k|_L) \cdot L - f_b^d(|k|_L)\} \\ &= \max_k \{k - (|k|_L - S_f)L - f_b(k)\} \leq (S_f + 1)L. \end{aligned} \quad (5.5)$$

The bounds $(S_b + 1)L$ and $(S_f + 1)L$ are set as the interleaver and deinterleaver delays for Π_{S-ibp} and Π_{S-ibp}^{-1} due to unknown intra-block permutation which is our one assumption described in Chapter 3. If $\Pi_{S-inter}$ is symmetric, both bounds are $(S + 1)L$.

5.2 Stream-oriented IBPTC encoding and the associated storage

An S-IBPTC encoder only requires temporary interleaving storage which is proportional to the interleaver delay. Figs. 2.4 (a) and 3.1 show the block diagrams of classic TC and IBPTC encoders. In general both interleavers have to store complete permuted sequence then encoding because the interleaving finishes almost when complete information sequence is input. Therefore, the storage is equal to the entire information length. In contrary to classic TC and IBPTC, the storage is not an entire information length for S-IBPTC because the interleaving of S-IBP interleaver can output permuted sequence in advance. The S-IBPTC encoder encodes the interleaved block \mathbf{u}'_i after the $(i + S_b)$ th block \mathbf{u}_{i+S_b} received and permuted while the $(i + S_b)$ th block \mathbf{u}_{i+S_b} is also permuted to the $(i + S_b + S_f)$ th block $\mathbf{u}'_{i+S_b+S_f}$. After encoding of the block \mathbf{u}'_i , the encoder discards the block \mathbf{u}'_i and the necessary temporary storage is equal to at most $(S_f + S_b + 1)L$ symbols instead of an entire information sequence. Even the input information length is infinite, the temporary storage for S-IBPTC does not exceed the total blocks within its S-IBP interleaver spans.

The swap interleaver further reduces temporary storage and implementation complexity for S-IBP interleaving and deinterleaving, where the swap interleaver has been defined as $\forall i \pi(i) = \pi^{-1}(i)$. In general S-IBP interleaver moves bits in a given block to positions within itself and those in the neighboring $S_f + S_b$ blocks, the associated interleaver normally stores at least $(S_f + S_b + 1)L$ symbols. If we can detain forward permutation procedure, it implies only $(S_b + 1)L$ symbols necessary for temporary interleaving storage. For the deinterleaving, the necessary storage is $(S_f + 1)L$ symbols. We take a symmetric swap S-IBP interleaver with $S = 1$ as an example. Fig. 1 (a) shows conventional storage for S-IBP interleaver. The symbols in the Block II have to be permuted to the Block I and Block III and the associated storage is $(2S + 1)L = 3L$ symbols. If the interleaver is a swap interleaver, we can detain the forward permutation

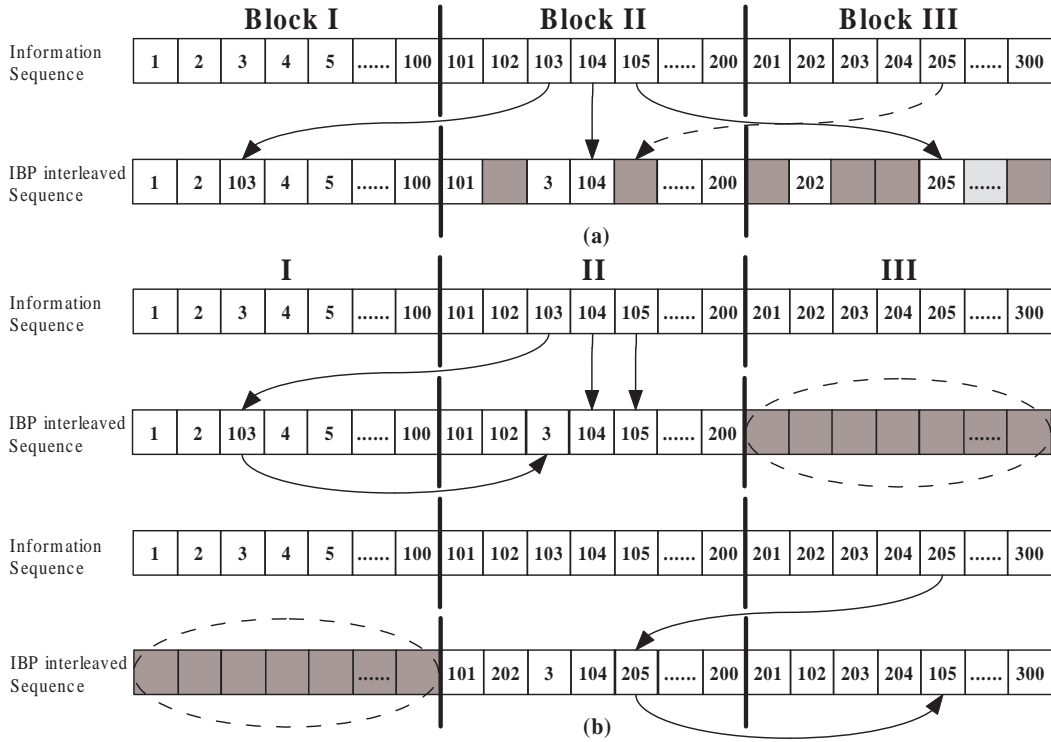


Figure 5.1: (a) Conventional inter-block permutation ($S = 1$); (b) Storage saving inter-block permutation ($S = 1$).

procedure, Fig. 5.1 (b) shows that the necessary storage is $(S + 1)L = 2L$ symbols. When one starts to interleave (or de-interleave) Block III, Block I has been completely interleaved (or de-interleaved), its content was dumped and the corresponding space is emptied and becomes available for storing new content again. The storage spaces enclosed by dotted ellipses are thus not needed. Therefore we do not have to move those forwardly-permuted symbols until after all earlier (backward) blocks have been filled by interleaved (or deinterleaved) symbols and after their contents have been dumped. For simplicity, the interleaver has only to perform memory content swapping between current block and the backward blocks. If $\Pi_{S-inter}$ is a *Type IV* inter-block permutation, the only IBP operation is simply $m \rightarrow m - nL$, where $n \in \{1, 2, \dots, S\}$. Moreover, a symmetric swap interleaver has the same interleaving and deinterleaving structure and can be implemented by single permutation table or algorithm. These advantages of symmetric

swap IBP interleavers will still be maintained when we consider the implementation of the combined intra- and inter-block permutations.

5.3 Pipeline decoder and the associated message-passing on the factor graph

Fig. 5.2 (a) shows an S-IBPTC decoding module for one iteration. The parenthesized numbers in each block indicate the corresponding latencies. A pipeline structure similar to that proposed by Hall [51] is shown in Fig. 5.2(b). The pipeline structure renders short decoding latency at the expense of increased complexity that is proportional to the number of the decoding modules. $\mathbf{L}_{app}(\mathbf{u}_k)$ and $\mathbf{L}_{ex}(\mathbf{u}_k)$ represent the a priori and the extrinsic information associated with the k th block \mathbf{u}_k , and $\mathbf{L}_{app}(\mathbf{u}'_{k-S_b-1})$ and $\mathbf{L}_{ex}(\mathbf{u}_{k-S_b-1})$ represent the a priori and the extrinsic information associated with the $(k - S_b - 1)$ th interleaved block \mathbf{u}'_{k-S_b-1} . $\mathbf{L}_{app}(\mathbf{u}_k)$ and $\mathbf{L}_{app}(\mathbf{u}'_{k-S_b-1})$ are acquired from the S-IBP deinterleaver and S-IBP interleaver, and the corresponding storage for both interleavers is $(S_f + S_b + 1)L$ symbols which is similar to the temporary storage in the encoding. If $\Pi_{S-inter}$ is a swap interleaver, the storage of interleaving and deinterleaving can be further reduced to $(S_b + 1)L$ and $(S_f + 1)L$ symbols respectively.

We apply the factor graph and decoding time diagram to demonstrate the edge of the S-IBPTC to classic TC under the same interleaver delay $2L$ when high throughput pipeline decoding is applied. Fig. 5.3 shows the pipeline decoding time diagram of the S-IBPTC and classic TC and Fig. 5.4 shows the factor graph of the S-IBPTC. An S-IBPTC composed of six length- L blocks with $S = 1$ is adopted in this example and the corresponding classic TC has interleaver length $2L$. We consider the third block \mathbf{u}_3 in Fig. 5.4. The first APP decoding of \mathbf{u}_3 starts at $2L$, and the generated extrinsic information is passed to \mathbf{u}'_2 and \mathbf{u}'_4 . The second APP decoding of \mathbf{u}'_2 and \mathbf{u}'_4 start at $3L$ and $5L$ respectively and the generated extrinsic information is passed to \mathbf{u}_1 and \mathbf{u}_5 as the a priori information for the third APP decoding. Therefore the information is passed to

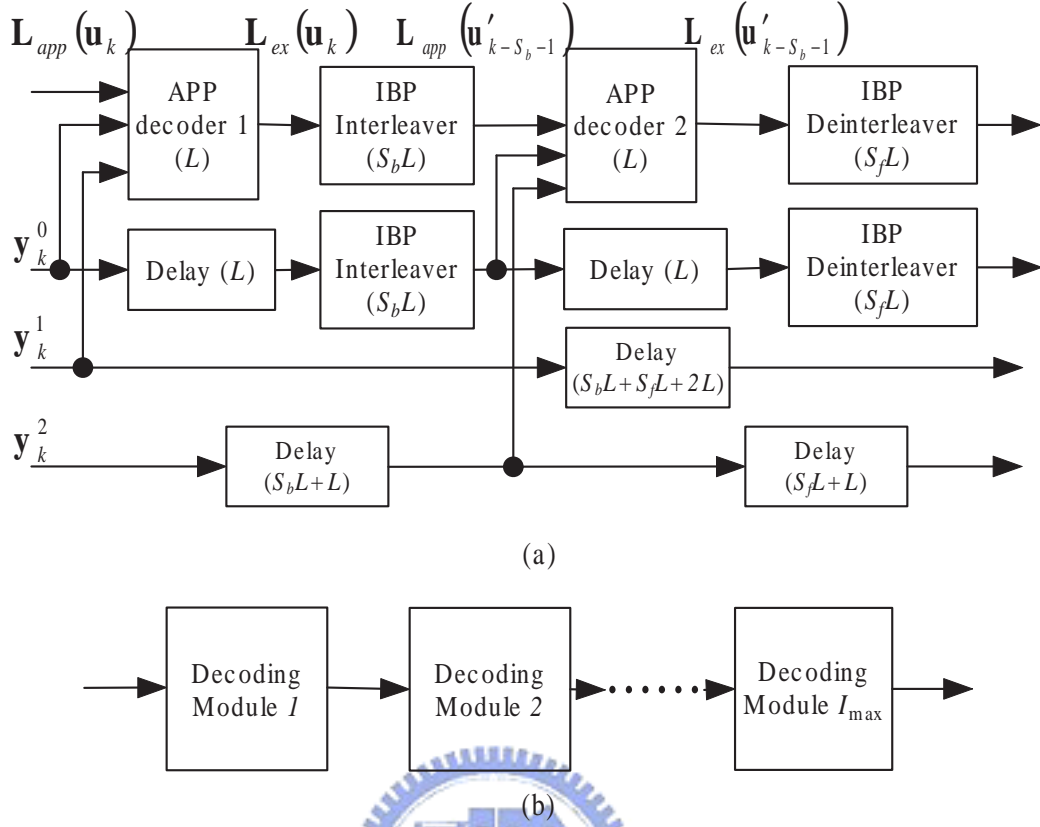


Figure 5.2: (a) The block diagram of an S-IBPTC decoding module for one iteration; (b) The block diagram of the S-IBPTC pipeline decoder.

five blocks in one iteration, i.e. the information spreads five blocks. Therefore the more blocks connected, the better performance. However the decoding of classic TC only acquires information within its block. When extreme throughput decoding throughput is necessary and the pipeline decoding is the only solution, S-IBPTC outperforms classic TC.

5.4 Bound and constraints modification for S-IBP interleaver

Theorem 3.2 provides an upper bound for IBPTC and this bound is also applicable for S-IBPTC. However all data sequences are of finite length in practice and there are either no or not enough blocks for the first $S_b - 1$ and the last $S_f - 1$ blocks to perform

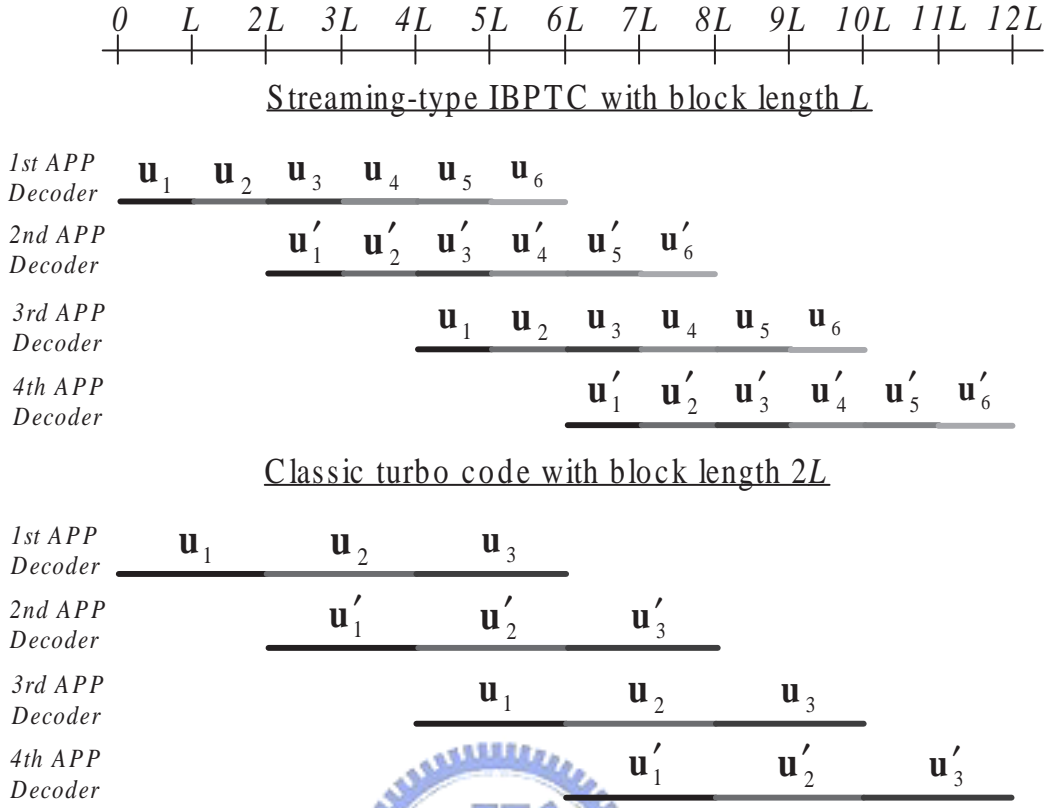


Figure 5.3: The time diagram of the pipeline decoder with 4 APP decoders or $I_{\max} = 2$.

either the complete backward or forward inter-block permutations. Therefore, we have to modify the S-IBP range for those blocks by reducing either the forward or the backward span. Assuming that there are $N \gg \max(S_f, S_b)$ blocks and denoting by $S_f(i)$ and $S_b(i)$ the forward and backward spans of the i th block, we require that for $0 \leq i < N$,

$$S_f(i) = \min(S_f, N - i + 1), \quad S_b(i) = \min(S_b, i). \quad (5.6)$$

Theorem 3.2 is modified accordingly.

Corollary 5.3 *For finite-length data sequences and a given inter-block permutation, Π_{inter} , whose spans are specified by eqn. (5.6), $\exists \Pi_{intra}$ such that the corresponding S-IBPTC satisfies $w_{2,min} \leq 2 + \alpha \cdot \min(S_f + 2, S_b + 2) + 2\beta$, if $L > T_c \cdot \min(S_f + 1, S_b + 1)$.*

To accommodate the S-IBP ranges defined by eqn. (5.6) for the finite-length inputs, the range of $f_n(|i|_L, |i|_L)$ and the period T_s of a *Type II* or *III* inter-block permutation

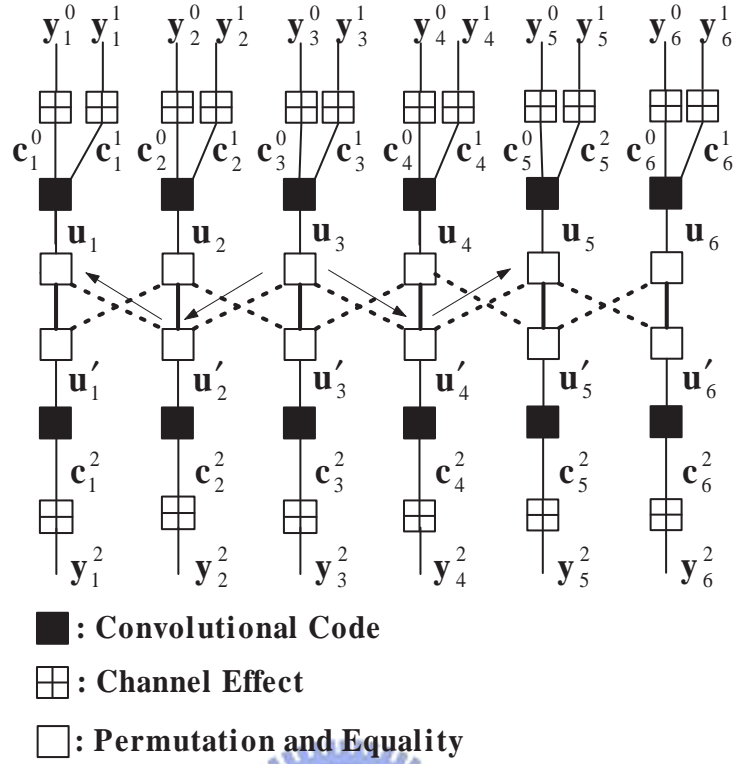


Figure 5.4: A factor graph representation for an S-IBPTC encoded system with the S-IBP span 1.



must be adjusted according to

$$\max(-S_b, -|i|_L) \leq f_n(|i|_L, |i|_L) \leq \min(S_f, N - 1 - |i|_L), \quad (5.7)$$

$$T_s(n) = \begin{cases} n + S_f + 1, & \text{if } 0 \leq n < S_b \\ N - n + S_b, & \text{if } N - S_f \leq n < N \\ S_f + S_b + 1, & \text{otherwise} \end{cases}, \quad (5.8)$$

where $0 \leq n < N$.

Even with the above modifications, low-weight codewords can still be generated for some weight-2 input sequences and block lengths for the first and last blocks must be adjusted. Periodic inter-block permutation is of our interest but there are too many backward permuted symbols due to same block length for all blocks and different inter-block permutation period T_s for the first block; the last block faces the same circumstance. A simple solution is to adjust the block lengths of the beginning S_b and the last S_f blocks

such that the block length of the i th block satisfies

$$nT_s(n) \leq L(n) < nT_s(n) + T_s(n), \quad (5.9)$$

for some n .

Lemma 5.8 *For an N -block S-IBPTC whose block lengths $L(n)$ are given by eqn. (5.9) and whose Π_{inter} is of a Type IV inter-block permutation with local interleaving periods defined by (5.8),*

$$\min_{i \cong j, \pi_{ibp}(i) \cong \pi_{ibp}(j)} w_t(\mathbf{C}^{ij}) \geq 2 + \alpha \left\lceil \frac{T_c + \text{lcm}(T_c, T_s(n))}{T_c} \right\rceil + 2\beta. \quad (5.10)$$

Finite-length versions of *Lemmas 3.3-3.4* can also be established if the block length and the corresponding S-IBP rule meet the requirements stated in the above lemma. For a stream-oriented C-IBPTC (S-C-IBPTC), however, Π_{block} needs to satisfy the additional requirement that for all $0 \leq i, j < L$ such that $\|\pi_{block}(i) - \pi_{block}(j)\|_{T_s} = 0$

$$f_1(i, j) + f_1(\pi_{block}(i), \pi_{block}(j)) \geq 2 + \alpha \left\lceil \frac{T_c + \text{lcm}(T_c, S_b + 1)}{T_c} \right\rceil + 2\beta. \quad (5.11)$$

When this requirement is also met then we have

$$\min_{i, j} w_t(\mathbf{C}^{ij}) \geq 2 + \alpha \left\lceil \frac{T_c + \text{lcm}(T_c, S_b + 1)}{T_c} \right\rceil + 2\beta. \quad (5.12)$$

5.5 Codeword weight upper-bounds of stream-oriented IBPTC

This section derives upper-bounds for the weights of S-IBPTC codewords associated with weight-2 and weight-4 input sequences. These upper-bounds are valid for all stream-oriented IBP interleavers.

Recall that *Lemma 3.1* implies that, the minimum codeword weight, $w_{2,min}$, for the weight-2 input sequences whose coordinates (i, j) of nonzero elements satisfy $i \sim j$ and $\pi(i) \sim \pi(j)$ is upper-bounded by

$$w_{2,min} \leq 2 + \alpha \cdot \left(\frac{|i - j| + |\pi(i) - \pi(j)|}{T_c} \right) + 2\beta, \quad (5.13)$$

where it is understood that the constants α and β might not have the same values as those of eqn. (3.12). A bound much tighter than eqn. (5.13) can be obtained by applying the approach suggested by Breiling [25] who partitions the coordinates set associated with both pre-interleaved and post-interleaved sequences into equivalence classes induced by the equivalent relation “ \sim ”. Each equivalence class is further divided into subsets $F_z = \{z + mT_c, m = 0, 1, \dots, |F_z| - 1\}$, where z is the smallest index in F_z .

An output (parity) sequence will be of finite weight if the coordinate pair (i, j) associated with the weight-2 input sequence \mathbf{u}^{ij} belongs to the same equivalence class. The parity sequence weight is small if the pair (i, j) , besides being in the same equivalence class, are in the proximity of each other, i.e., if $(i, j) \in F_z$ for some z and the width of $F_z = (|F_z| - 1)T_c$ is small.

To avoid generating low-weight codewords, therefore, an optimal interleaver should send any pair of coordinates in a given subset to different equivalent classes and, if that is not possible, to different subsets or at least to two far-apart coordinates within a subset. Let $F_z^{(m)}$ and Λ_m , ($0 \leq z < \Lambda_m$) be the subsets and the number of subsets associated with the coordinates of the m th component encoder input sequence. The cardinalities of the Λ_m subsets differ at most by 1, i.e., $|F_z^{(m)}| = \lfloor L/\Lambda_m \rfloor$ or $\lfloor L/\Lambda_m \rfloor + 1$. Invoking the aforementioned pigeonhole principle, Breiling showed that if the pair (Λ_1, Λ_2) is such that $\lceil L/\Lambda_1 \rceil > \Lambda_2$ then any interleaver would map a pair of coordinates (i, j) that lies in the same subset $F_z^{(1)}$ to $(\pi(i), \pi(j))$ which also belongs to an identical subset $F_{z'}^{(2)}$, resulting in

$$w_t(\mathbf{C}^{ij}) \leq 2 + \alpha \left(\left\lceil \frac{L}{\Lambda_1} \right\rceil + \left\lceil \frac{L}{\Lambda_2} \right\rceil - 2 \right) + 2\beta. \quad (5.14)$$

Minimizing the right hand side of the above inequality with respect to the the pair (Λ_1, Λ_2) , Breiling then obtained a very tight upper-bound.

5.5.1 The upper-bound for weight-2 input sequences

It is clear that, given the same set of parameters $\{L, \Lambda_i, T_c, |F_z^{(i)}|\}$, an S-IBP interleaver has subsets within its span to choose from for placing members of the set $\{\pi(i), i \in F_z^{(1)}\}$, for some $0 \leq z < L$. Thus, assuming a large enough block size L , the priority of an optimal S-IBP rule in permuting coordinates of the same equivalence class follows the order: (i) to different blocks, (ii) to different equivalent classes of the same block, (iii) to different subsets of the same equivalent class, and finally, (iv) to far-apart coordinates within the same subset. Obviously, the partition of an equivalence class into subsets plays a pivotal role in optimizing an S-IBP rule. With a minimum loss of generality, we assume $||\Lambda_1||_M = ||\Lambda_2||_M = 0$, $M = T_s T_c$, where $T_s = 2S + 1$. Given these parameter values, we consider the following (subset) partition.

$$F_i^{(k)} = \left\{ \begin{array}{l} \left\{ |i|_M + [i - |i|_M] \left\lceil \frac{L}{\Lambda_k} \right\rceil + Mj \mid 0 \leq j < \left\lceil \frac{L}{\Lambda_k} \right\rceil, 0 \leq i < |L|_{\Lambda_k} - |L|_M \right\}, \\ \left\{ |i|_M + [|L|_{\Lambda_k} - |L|_M] \left\lceil \frac{L}{\Lambda_k} \right\rceil + [i - |i|_M - |L|_{\Lambda_k} + |L|_M] \left\lceil \frac{L}{\Lambda_k} \right\rceil + Mj \mid \right. \\ \left. 0 \leq j < \left\lceil \frac{L}{\Lambda_k} \right\rceil, |L|_{\Lambda_k} - |L|_M \leq i < \Lambda_k - M \right\}, \\ \left\{ |i|_M + [|L|_{\Lambda_k} - |L|_M] \left\lceil \frac{L}{\Lambda_k} \right\rceil + [\Lambda_k - |L|_{\Lambda_k} - M + |L|_M] \left\lceil \frac{L}{\Lambda_k} \right\rceil + Mj \mid \right. \\ \left. 0 \leq j < \left\lceil \frac{L}{\Lambda_k} \right\rceil, \Lambda_k - M \leq i < \Lambda_k - (M - |L|_M) \right\}, \\ \left\{ |i|_M + [|L|_{\Lambda_k} - |L|_M] \left\lceil \frac{L}{\Lambda_k} \right\rceil + [\Lambda_k - |L|_{\Lambda_k} - M + |L|_M] \left\lceil \frac{L}{\Lambda_k} \right\rceil + Mj \mid \right. \\ \left. 0 \leq j < \left\lceil \frac{L}{\Lambda_k} \right\rceil, \Lambda_k - (M - |L|_M) \leq i < \Lambda_k \right\}. \end{array} \right. \quad (5.15)$$

An exemplary partition of eqn. (5.15) is shown in Fig. 5.5 where the integers represent the coordinates of either an input or output sequence and each row consists of three segments with a segment representing a subset of size 3 or 2. The S-IBP rule sends bits in rows labelled by different capital letters to different blocks while those in the same row are interleaved to the same block.

By using an argument similar to that leading to eqn. (5.14) and invoking the partition of eqn. (5.15) along with the permutation rule (i)-(iv) mentioned at the beginning paragraph of this subsection, we obtain

Theorem 5.9 *For the class of S-IBPTCs, the minimum codeword weight $w_{2,min}$ for*


$T_s=3$ {	A	$T_c=3$	0	9	18	27	36	45	54	63	
			1	10	19	28	37	46	55	64	
			2	11	20	29	38	47	56	65	
		B	$T_c=3$	3	12	21	30	39	48	57	66
		4		13	22	31	40	49	58	67	
		5		14	23	32	41	50	59	68	
		C	$T_c=3$	6	15	24	33	42	51	60	
		7		16	25	34	43	52	61		
		8		17	26	35	44	53	62		

Figure 5.5: Partition of equivalence classes into subsets; $L = 68$, $\Lambda = 27$, $T_c = T_s = 3$.

weight-2 input sequences is upper-bounded by

$$w_{2,min} \leq 2 + \alpha \left(\min_{(\Lambda_1, \Lambda_2)} \left\{ \left\lceil \frac{T_s L}{\Lambda_1} \right\rceil + \left\lceil \frac{T_s L}{\Lambda_2} \right\rceil \right\} - 2 \right) + 2\beta \quad (5.16)$$

where $(\Lambda_1, \Lambda_2) \in D \times D$, $D = \{1, 2, \dots, \lceil L/M \rceil - 1\}$, if $L > MT_c$ and

$$\left\lceil \frac{L}{\Lambda_1} \right\rceil > \frac{\Lambda_2}{T_s}.$$

When $\Lambda_1 = \Lambda_2$, we have

$$w_{2,min} \leq 2 + 2\alpha \frac{T_s L}{\sqrt{T_s L} - T_s T_c} + 2\beta, \quad (5.17)$$

Proof: Eqn. (5.17) follows directly from the partition in eqn. (5.15) and the optimal periodic S-IBP. The corresponding interleaver results in bound-achieving codewords \mathbf{C}^{ij} , $(i, j) \in s_m$, when $\lceil \frac{L}{\Lambda_1} \rceil > \frac{\Lambda_2}{T_s}$. Hence

$$\begin{aligned} \min_{(\Lambda_1, \Lambda_2)} \left\{ \left\lceil \frac{T_s L}{\Lambda_1} \right\rceil + \left\lceil \frac{T_s L}{\Lambda_2} \right\rceil \right\} &= \min_{\Lambda_1} \left\{ \left\lceil \frac{T_s L}{\Lambda_1} \right\rceil + \min_{\lceil \frac{L}{\Lambda_1} \rceil > \frac{\Lambda_2}{T_s}} \left\lceil \frac{T_s L}{\Lambda_2} \right\rceil \right\} \\ &\leq \min_{\Lambda_1} \left\{ \left\lceil \frac{T_s L}{\Lambda_1} \right\rceil + \left\lceil \frac{L}{T_c (\lceil \frac{L}{T_c \Lambda_1} \rceil - 1)} \right\rceil \right\} \end{aligned} \quad (5.18)$$

The upper-bound (5.16) can be rewritten as

$$w_{2,min} \leq 2 + \alpha \left(\min_{\Lambda_1} \left\{ \left\lceil \frac{T_s L}{\Lambda_1} \right\rceil + \left\lceil \frac{L}{T_c (\lceil \frac{L}{T_c \Lambda_1} \rceil - 1)} \right\rceil \right\} - 2 \right) + 2\beta \quad (5.19)$$

If we choose $(\Lambda_1, \Lambda_2) = (\Lambda_0, \Lambda_0)$ with $\Lambda_0 = M(\lceil \frac{\sqrt{T_s L}}{M} \rceil - 1)$, i.e., Λ_0 is a multiple of M and $\Lambda_0^2 < T_s L$, then (5.16) implies

$$\begin{aligned} w_{2,min} &\leq 2 + 2\alpha \left(\left\lceil \frac{T_s L}{M \cdot (\lceil \frac{\sqrt{T_s L}}{M} \rceil - 1)} \right\rceil - 1 \right) + 2\beta \\ &\leq 2 + 2\alpha \left(\left\lceil \frac{T_s L}{\sqrt{T_s L} - M} \right\rceil - 1 \right) + 2\beta \\ &< 2 + 2\alpha \left(\frac{T_s L}{\sqrt{T_s L} - T_s T_c} \right) + 2\beta \end{aligned} \quad (5.20)$$

■

Theorem 5.9 implies that $w_{2,min}$ grows linearly with $\sqrt{T_s L}$ when L is large.

5.5.2 The upper-bound for weight-4 input sequences

Let the coordinates of nonzero elements of a weight-4 input sequence be (i, j, k, l) , where $i < j < k < l$. If we divide these coordinates and their permuted positions respectively into two pairs each according to their natural order, i.e., $(i, j), (k, l)$ and say, $(\pi(i), \pi(k)), (\pi(j), \pi(l))$, then a low-weight codeword results if each pair belongs to the same subset. More specifically, the minimum codeword weight, $w_{4,min}$, for weight-4 input sequences whose nonzero coordinates (i, j, k, l) are such that $i \sim j, k \sim l, \pi(i) \sim \pi(k), \pi(j) \sim \pi(l)$ satisfy

$$w_{4,min} \leq 4 + \alpha \cdot \left(\frac{|i - j| + |k - l| + |\pi(i) - \pi(k)| + |\pi(j) - \pi(l)|}{T_c} \right) + 2\beta \quad (5.21)$$

or

$$w_{4,min} \leq 4 + \alpha \cdot \left(\frac{|i - j| + |k - l| + |\pi(i) - \pi(j)| + |\pi(k) - \pi(l)|}{T_c} \right) + 2\beta. \quad (5.22)$$

if (i, j, k, l) are such that $i \sim j, k \sim l, \pi(i) \sim \pi(j), \pi(k) \sim \pi(l)$.

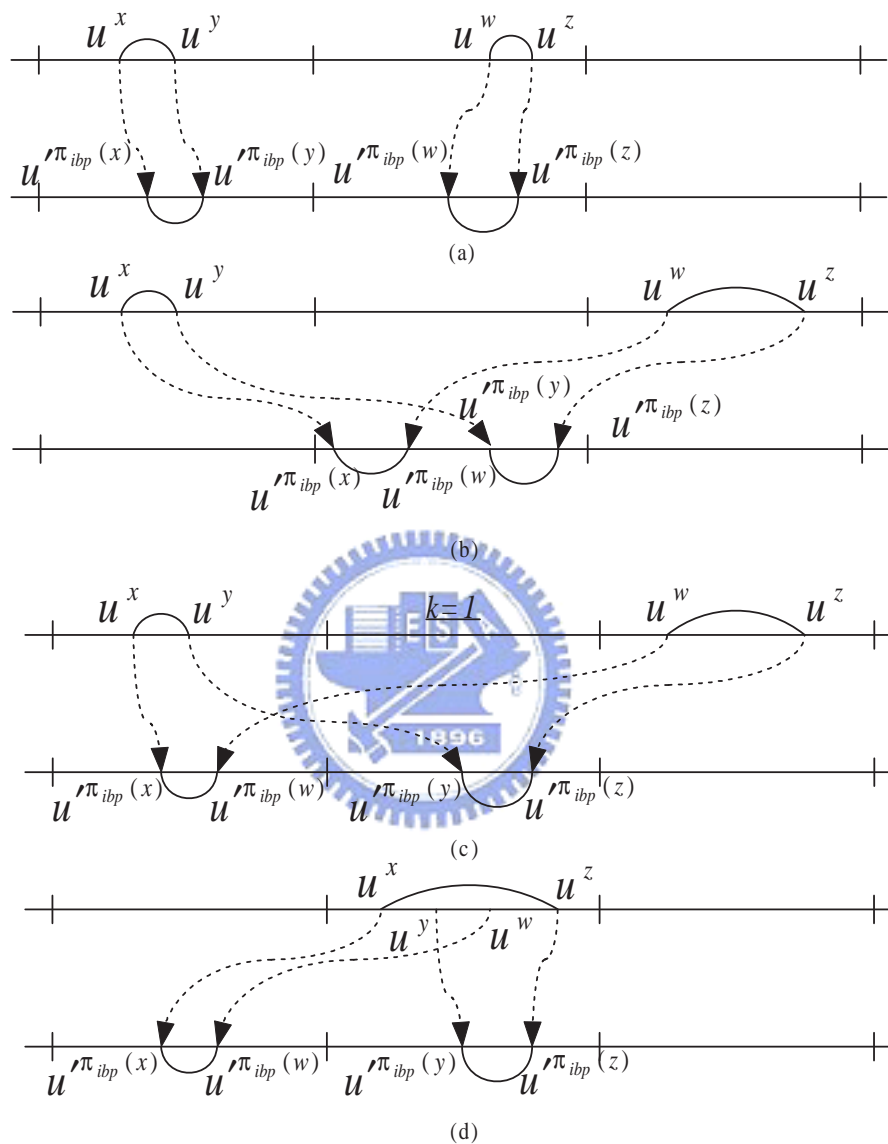


Figure 5.6: Pre- and post-interleaving nonzero coordinate distributions of weight-4 input sequences that result in low-weight S-IBPTC codewords.

These upper-bounds are obtained by considering the three pre- and post-interleaving distributions of the 4-tuple (i, j, k, l) shown in Fig. 5.6 (a)-(c). These three are the distributions that most likely lead to low-weight codewords. There are other candidate distributions (e.g., Fig. 5.6 (d)) but the corresponding upper-bounds are likely to be larger than those given by eqns. (5.21) and (5.22).

Following an approach similar to that of [25] and taking into account the extra degrees of freedom offered by an S-IBP interleaver, we obtain

Theorem 5.10 *The S-IBPTC minimum codeword weight for weight-4 input sequences is upper-bounded by*

$$w_{4,min} \leq 4 + 2\alpha \left(\min_{(\Lambda_1, \Lambda_2)} \left\{ \left\lceil \frac{T_s L}{\Lambda_1} \right\rceil + \left\lceil \frac{T_s L}{\Lambda_2} \right\rceil \right\} - 2 \right) + 4\beta \quad (5.23)$$

when $(\Lambda_1, \Lambda_2) \in D \times D$ satisfies (i) $\Lambda_1 \binom{\Omega}{2} > \left(\frac{\Lambda_2}{T_s}\right)^2$, (ii) $\frac{(T_s-k)}{T_s} \cdot \Lambda_1 \Omega^2 > \left(\frac{\Lambda_2}{T_s}\right)^2$, and (iii) $|\Lambda_i|_M = 0$, $i = 1, 2$, where $D = \{1, 2, \dots, \lfloor L/2 \rfloor\}$, $\Omega = \lfloor \frac{L}{\Lambda_1} \rfloor$ and $k = 1, 2, \dots, T_s - 1$. Moreover, for the special case, $\Lambda_1 = \Lambda_2$ and if $L > \frac{10}{3}T_s^3 + T_s^2 - \frac{T_s}{3}$ and $T_s > 1$ the upper-bound yields the compact expression

$$w_{4,min} \leq 4 + 4\alpha \frac{T_s L}{C - T_s T_c} + 4\beta, \quad (5.24)$$

where

$$C = \frac{T_s + 2T_s^2}{3} + \sqrt[3]{-\frac{q_1}{2} + \sqrt{\left(\frac{p_1}{3}\right)^3 + \left(\frac{q_1}{2}\right)^2}} + \sqrt[3]{-\frac{q_1}{2} - \sqrt{\left(\frac{p_1}{3}\right)^3 + \left(\frac{q_1}{2}\right)^2}} \quad (5.25)$$

$$p_1 = 3T_s^2 L - \frac{1}{3}(T_s + 2T_s^2)^2, \quad (5.26)$$

$$q_1 = -T_s^2 L^2 + (T_s^3 + 2T_s^4)L - \frac{2}{27}(T_s + 2T_s^2)^3. \quad (5.27)$$

Proof: See Appendix B. ■

Again, we observe that for large L , the upper-bound grows linearly with $(T_s L)^{\frac{1}{3}}$. The minimum codeword weights associated with weight-2 and weight-4 input sequences are upper-bounded by the increasing functions of $T_s L$.

5.5.3 Interleaving gain comparison

The minimum weight codeword upper-bound of weight-2 and weight-4 input sequences are derived above and S-IBPTC outperforms classic TC in distance properties when the interleaver delay is the same for both S-IBPTC and classic TC. We consider the interleaver delay is $(S + 1)L$ and the equivalent block size of classic TC is also $(S + 1)L$. Let $w_{2,min,block}$ and $w_{4,min,block}$ be the minimum codeword weights associated with weight-2 and weight-4 input sequences of classic TC with block size $(S + 1)L$, then we have [25]

$$w_{2,min,block} \leq 2 + 2\alpha \frac{(S + 1)L}{\sqrt{(S + 1)L - T_c}} + 2\beta \quad (5.28)$$

$$w_{4,min,block} \leq 4 + 4\alpha \frac{(S + 1)L}{((S + 1)L - 1)^{\frac{2}{3}} - ((S + 1)L - 1)^{\frac{1}{3}} + 1 - T_c} + 4\beta. \quad (5.29)$$

Comparing the above equations with eqns. (5.17) and (5.24) and noting that $T_s = 2S + 1$, we conclude that, as far as weight-2 and weight-4 input sequences are concerned, a ‘good’ S-IBPTC can bring about improvement factors of $(2 - \frac{1}{S+1})^{\frac{1}{2}}$ and $(2 - \frac{1}{S+1})^{\frac{1}{3}}$, respectively.



5.6 Stream-oriented IBP

Theorems 3.1 and 3.2 give us two guidelines for designing an IBP. In the previous paragraph, we show that the S-IBP with the swap structure has an implementation edge. Shown in Table 5.1 is a symmetric S-IBP with $S = S_f = S_b$ and interleaver delay is $(S + 1)L$. It can be easily seen that

Corollary 5.4 *The algorithm in Table 5.1 satisfies the requirements of both Type IV and Type V inter-block permutation.*

5.7 Modified semi-random interleaver

Semi-random interleavers [44] are designed to eliminate “short cycles” that send two close-by bits to the vicinity of each other after interleaving. These interleavers are,

Table 5.1: S-IBP Algorithm

<p><u>Variables</u></p> <p>L-block length</p> <p>N-total number of blocks</p> <p>K-block number index</p> <p>D(m,k)-data on the kth block mth position</p> <p><u>Recursion</u></p> <p>for K=0 to N-1</p> <p> for i=0 to i=S-1</p> <p> if (K-i > 0)</p> <p> if (K mod (2·(i+1)) < i+1)</p> <p> set m=2·i+1</p> <p> else</p> <p> set m=2·i+2</p> <p> end</p> <p> while (m < L)</p> <p> swap D(m,K) and D(m,K-i-1)</p> <p> set m=m+2S+1</p> <p> end</p> <p> end</p> <p> end</p> <p>end</p>

however, originally designed to work in the block interleaving setting, therefore they can not avoid two new classes of short cycles arising in S-TB-IBPTC and S-C-IBPTC. A tail-biting convolutional code begins and ends at the same state, hence if two close-by bits in a block are respectively intra-block permuted to the beginning and the ending parts of that block, and if the two bits remain in the same block after the S-IBP interleaving, a short cycle will result as the proposed S-IBP does not alter their relative positions within a block. For the class of S-C-IBPTC, we also want to prevent similar intra-block interleaving results because the S-IBP interleaver may send such a pair to the ending and beginning parts of two neighboring blocks. We therefore modify the constraint of [44] as

$$d_{min}(i, j) + d_{min}(\pi(i), \pi(j)) > S_2, \quad 0 \leq i, j < L \quad (5.30)$$

where $d_{min}(i, j) = \min(|i - j|, L - |i - j|)$. This new constraint excludes the possibility that two symbols at the beginning and the ending parts of a block would remain there after the interleaving.



5.8 Simulation Results

This section shows extrinsic information updating behavior and provides results of error rate performance. As mentioned in Section 5.3, inter-block permutation provides relation across adjacent blocks and the information are spread to other blocks by the assistance of message-passing but this is only an intuitive explanation for the edge of S-IBPTC. In order to visualize the effect of message-passing, we demonstrate the behavior of the evolution between the a priori information and extrinsic information at each iteration, and covariance, mutual information and SNR are used to measure this evolution. The covariance between the a priori and extrinsic information indicates how much new information generated after each APP decoding and less covariance implies more information generated. The mutual information and SNR come from the extrinsic

information transfer chart (EXIT chart) of [27, 28] and the extrinsic information SNR evolution chart of [34]. They are also used to study the convergence behavior of iterative decoding schemes and both methods have been described in Chapter 2. The quantization will show the edge of S-IBPTC to classic TC. Then we provide some curves of error rate performance for S-IBPTC and classic TC to evidence these behaviors.

In order to provide a fair comparison and for repeatable simulations, we use the component codes and interleavers defined in 3GPP and DVB-RCS/RCT [1, 37, 38] in this section.

5.8.1 Covariance and convergence behavior

At first we show the behavior of 3GPP [1] defined turbo code. The code rate is $1/3$, and the covariance behavior, mutual information and SNR evolutions are shown.

Fig. 5.7 shows the covariance behavior for both S-IBPTC and classic TC with the same interleaver delay, where the S-IBP interleaver has $S = 1$ and interleaver delay 800 and the interleaving depth for the classic TC is $L = 800$. It indicates that the covariance is small for the S-IBPTC even at $\text{SNR} = 0.5$ dB while much higher covariance is observed for the classic TC at much higher SNR. The S-IBP collects extrinsic information from farther and farther away as the number of iterations increases and we have expected that it results in smaller covariance.

Fig. 5.8 compares the EXIT evolutions of our proposal and the classic TC with the same interleaver delay. The S-IBPTC yields mutual information almost equal to one at $\text{SNR} = 0.5$ and 1.0 dB, but the classic TC needs $\text{SNR} = 2.0$ dB to reach similar convergence point. The SNR evolution chart shown in Fig. 5.9 exhibits similar behavior of the two codes, all indicating the proposed S-IBPTC gives superior convergence behavior. Both figures also reveal that our code has a much faster convergence speed. The much larger step of the S-IBPTC curves means the associated APP decoder generates more information or extrinsic information with larger signal to noise ratio for the next stage

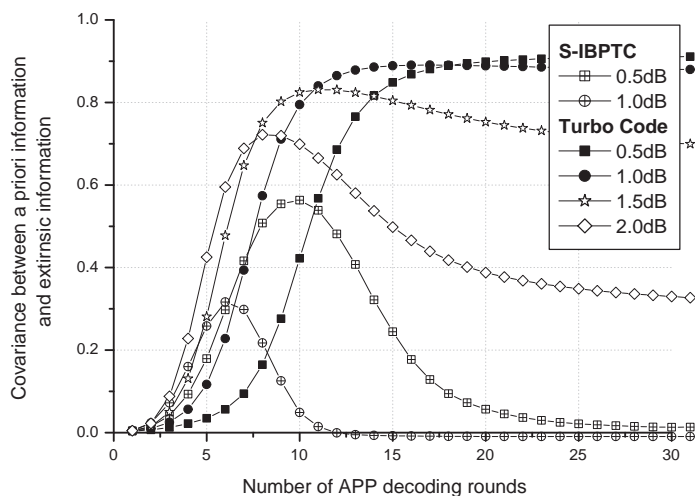


Figure 5.7: Covariance between a priori information input and extrinsic information output.

decoder. Such a trend has been expected when we examine the factor graph structure of the S-IBPTC in Fig. 5.4.

Bit-level and symbol-level IBP are compared for duo-binary turbo code (DTC) defined in DVB-RCS/RCT [37, 38]. This code applies duo-binary RSC and a symbol-based interleaver. This interleaver applies intra- and inter-symbol permutation and two bits in one symbol are grouped and permuted to very close destination. However [54] indicated that interleaver should lower down the covariance of the a priori information of two bits if two bits are close and a good interleaver permutes two close-by bits as far as possible. Intuitively this symbol-based interleaver results in high covariance of the extrinsic information for two bits belonging to one symbol and this may be harmful to error rate performance. Therefore two options of S-IBP are compared: bit-level and symbol-level S-IBPs. Fig. 5.10 shows a simulation examples of 0.5 and 1.0 dB for symbol-level and bit-level S-IBP. 53Bytes=424bits block interleaver are chosen as our intra-block permutation. The covariance of symbol-level S-IBP is always much higher than that of bit-level S-IBP. No matter on 0.5 dB and 1.0dB, symbol-level S-IBP provides near 0.7-

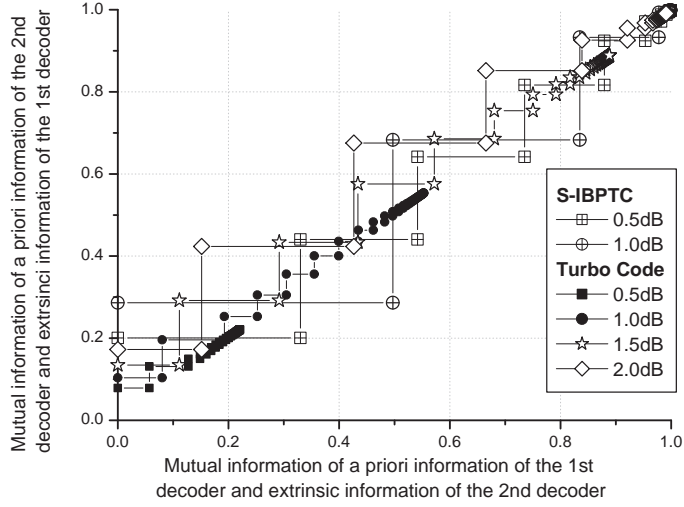


Figure 5.8: Exit chart performance of the S-IBPTC and the classic TC at different E_b/N_0 's.

0.8 covariance and this means at beginning some iterations, a priori information can not give enough information to both bits in one symbol. Bit-level S-IBP renders much lower correlation at beginning iterations, and it indicates that a priori information taking new information in assisting decoding. This result confirms our prediction. We have a property as follow.

Property 1 *For duo-binary turbo code, two bits in each symbol should be permuted to and from two bits in different blocks by an S-IBP.*

5.8.2 Error probability performance

Computer simulation results reported in this section firstly use the RSC code of the 3GPP standard, $G(D) = \left[1, \frac{1+D+D^3}{1+D^2+D^3}\right]$ [1], the interleaver of the same standard or the modified semi-random interleaver of eqn. (5.30) for the intra-block permutation while the S-IBP follows the algorithm of Table 5.1. Each simulation run consists of 1000 blocks for S-IBPTC. We use the Log-MAP or MAX-Log-MAP algorithms to decode classic TC

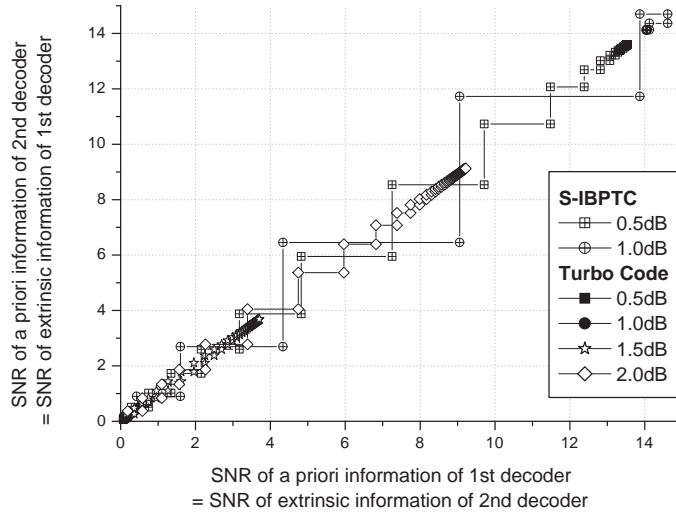


Figure 5.9: SNR evolution chart behavior of the S-IBPTC and the classic TC at different E_b/N_0 's.

and stream-oriented TP-IBPTC (S-TP-IBPTC) and the sliding-window Log-MAP or the sliding-window MAX-Log-MAP algorithms to decoding stream-oriented TB-IBPTC (S-TB-IBPTC) and S-C-IBPTC. In most cases, we compare the performance of classic TC and S-IBPTC under the assumption that either both codes have the same interleaver delay.

Figs. 5.11 and 5.12 show the BER performance of rate 1/3 turbo coded systems with 10 iterations and Log-MAP algorithm. The interleaver parameter values for the S-IBPTCs are $L = 402$, $S = 1$ or $L = 265$, $S = 2$. Compared with the performance of the classic TC with $L = 400$, the S-IBPTCs yield 0.7–0.9 dB performance gain at $\text{BER}=10^{-4}$ and 1.0–1.2 dB gain at $\text{BER}=10^{-6}$. When both codes have the same interleaver delay, the S-IBPTCs provides 0.4–0.6 dB performance gain at BER between 10^{-4} and 10^{-6} .

Figs 5.13 and 5.14 show the BER performance of rate 1/2 turbo coded systems. The MAX-Log-MAP algorithm is used in this example. We compare the performance of the classic TC with $L = 1320$ and the S-IBPTCs with $L = 660$, $S = 1$ and $L = 440$, $S = 2$. Using $L = 660$, $S = 1$ and the 3GPP interleaver as the intra-block permutation, the

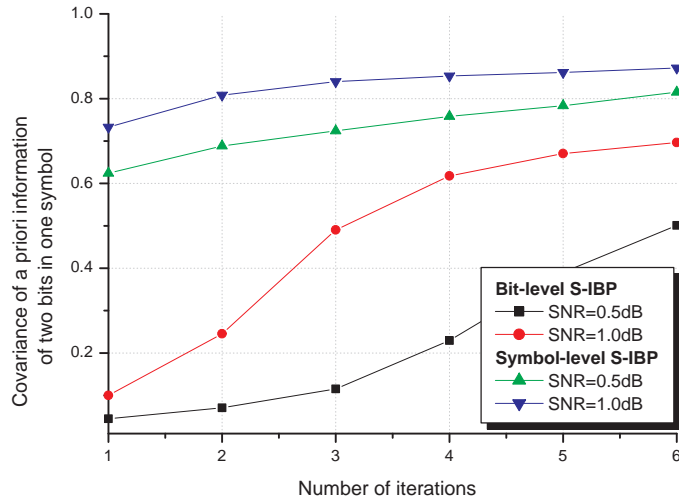


Figure 5.10: A comparison of covariance of bit-level and symbol-level IBP.

S-IBPTCs have 0.4–0.45 dB and 0.3 dB gain at $\text{BER}=10^{-5}$ and 10^{-6} , respectively. For other cases, the S-IBPTCs give 0.4–0.45 dB gain at $\text{BER}=10^{-5}$ and 0.4–0.6 dB gain (except for the case S-TP-IBPTC with $L = 440, S = 2$) at $\text{BER}=10^{-6}$. It is clear that the S-IBPTCs outperform the classic TCs with nearly the same interleaver delay. Furthermore, the proposed modified s-random interleaver outperforms the 3GPP defined interleaver, especially when the interleaver span is small $S = 1$.

These figures reveal that the proposed S-IBPTCs yield superior performance, sharper slope of the BER curve at the waterfall region and lower error floor when compared with the corresponding performance curves of the classic TCs for a variety of different code rates and decoding algorithms. The improvement is more impressive for smaller block interleaver with the same interleaver delay, i.e. a larger S-IBP interleaver span S leads to better performance.

Fig. 5.15 shows the BER performance of rate 1/3 S-IBPTCs that use the 3GPP interleaver as the intra-block interleaver. Either the Log-MAP algorithm or the Log-MAP algorithm is used and 15 decoding iterations is assumed. All these S-IBP parameter

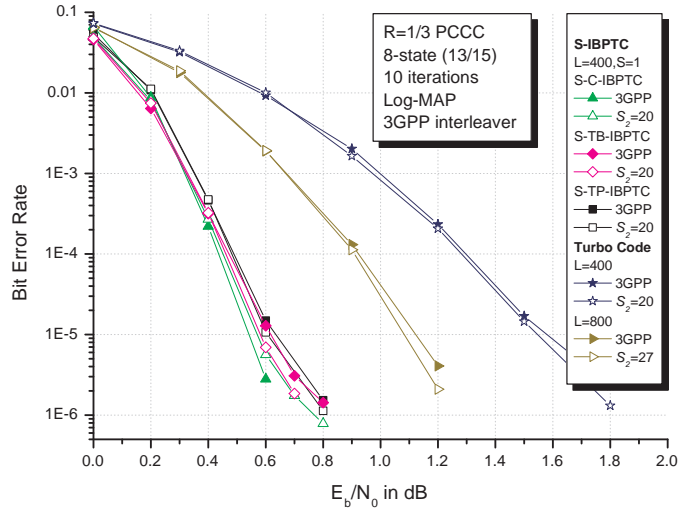


Figure 5.11: BER performance of the S-IBPTCs with interleaver delay ≈ 800 , block size $L = 402$ and interleaver span $S = 1$ and the classic TCs with block sizes $L = 400, 800$.

values, $(L, S) = (660, 1), (440, 2)$ or $(330, 3)$, give the same interleaver delay of 1320 bits. The performance is consistent with our prediction: the larger the interleaver span is, the better the system performance becomes. The performance deteriorates when the period of encoder, T_c , and the period of the S-IBP interleaver, T_s , are the same. For this case the lower-bound of eqn. (3.16) becomes $2(1 + \alpha + \beta)$ which is much smaller than the corresponding upper-bound given in *Theorem 3.2*. By contrast, the two bounds are much closer if $T_c \neq T_s$ and both bounds give identical value if T_c and T_s are relative prime.

Finally, we want to show that the S-IBPTC requires an interleaver latency much smaller than that of classic TCs with similar BER performance. Fig. 5.16 shows the BER performance of rate 1/3 turbo coded systems that employ 10 decoding iterations and the Log-MAP algorithm. All the interleavers are taken from the 3GPP interleaver. The average interleaver and deinterleaver latency of the S-IBPTCs is about 800. It is observed that the performance of the S-IBPTCs is bounded by those of turbo codes with block size $L = 2800$ and $L = 3600$. In other words, an S-IBPTCs achieves BER

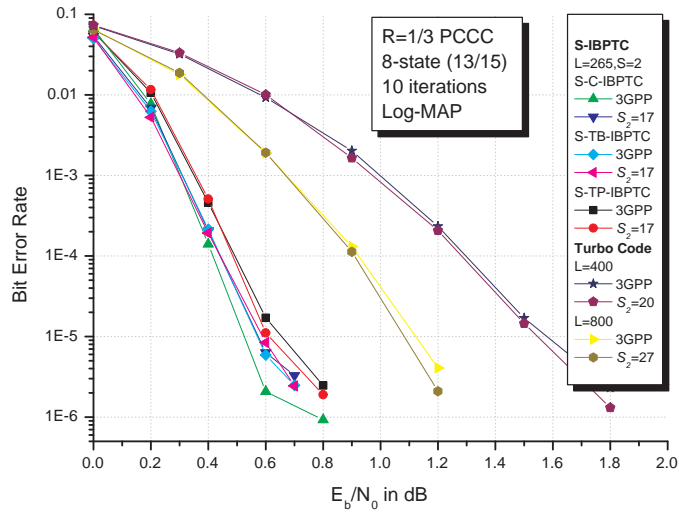


Figure 5.12: BER performance of the S-IBPTCs with interleaver delay ≈ 800 , block size $L = 265$ and interleaver span $S = 2$ and the the classic TCs with block sizes $L = 400, 800$ are also given.

performance similar to that of a classic TC which requires an interleaving latency 3.5 to 4.5 times longer.

All these figures show that the S-TB-IBPTC has the best performance, followed by the S-C-IBPTC and then the S-TP-IBPTC.

This part evaluates the performance of stream-oriented IBP duo-binary turbo code (S-IBPDTC) and compare with the duo-binary turbo code (DTC) defined in DVB-RCS/RCT [37, 38]. The RSC code, $G(D) = \begin{bmatrix} 1 & 0 & \frac{1+D+D^3}{1+D^2+D^3} & \frac{1+D^3}{1+D^2+D^3} \\ 0 & 1 & \frac{(1+D+D^3)(1+D+D^2)}{1+D^2+D^3} & \frac{(1+D^3)(1+D+D^2)}{1+D^2+D^3} \end{bmatrix}$, defined in the DVB-RCS/RCT [37, 38] is used as the component code. The stream-oriented IBP duo-binary turbo code (S-IBPDTC) applies the 53Bytes DVB-RCS/RCT interleaver as its intra-block permutation and the bit-level and symbol-level S-IBPs applies the algorithm shown in Table 5.1. The DTCs with 53Bytes and 106Bytes interleavers are simulated as reference curves. Each simulation run consists of 1000 blocks for the S-IBPDTC. Sliding window Log-MAP and sliding window MAX Log-MAP algorithms are also used in this simulation. Fig. 5.17 shows the simulation results. The

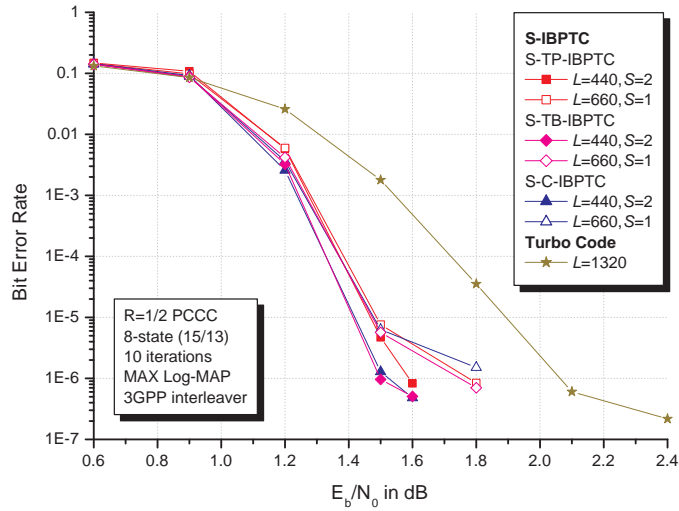


Figure 5.13: BER performance of S-IBPTCs and the classic TC with interleaver delay 1320 and the 3GPP interleaver.

bit-level and symbol-level S-IBPDTCs outperform the DTC with 53Bytes by 0.8 – 1.0dB and 0.6 – 0.8dB at block error rate (BLER)= 10^{-4} . If we consider the same interleaving delay, the bit-level and symbol-level S-IBPDTCs outperform the DTC with 53Bytes by 0.5 – 0.9dB and 0.2 – 0.6dB at BLER= 10^{-4} . Under the same block size or identical interleaving delay, the S-IBPDTC outperforms the DTCs.

The comparison between the bit-level and symbol-level IBPs is also of our interest. The symbol-level IBP outperforms the bit-level IBP at low SNR but loses at high SNR. This implies that the bit-level IBP provides better distance property than the symbol-level IBP. However the bit-level IBP necessitates the marginalization for conversion from a symbol to two bits on the decoder and this induces information loss. If the target BLER is on 10^{-2} , the symbol-level IBPDTC is a better choice. If the target BLER is on 10^{-4} or lower, the bit-level IBPDTC is our favorite.

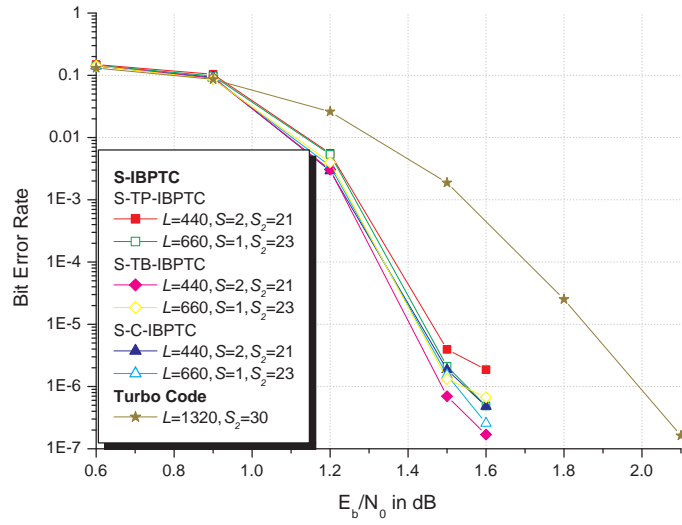


Figure 5.14: BER performance of S=IBPTCs and the classic TC with interleaver delay 1320 and the modified semi-random interleaver.

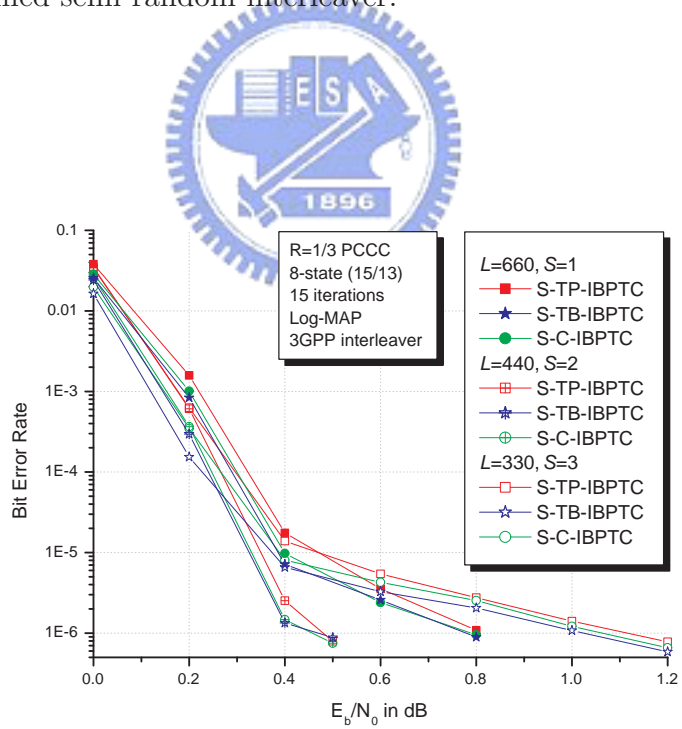


Figure 5.15: Influence of the interleaver span on the BER performance for various S-IBPTCs with interleaver delay 1320.

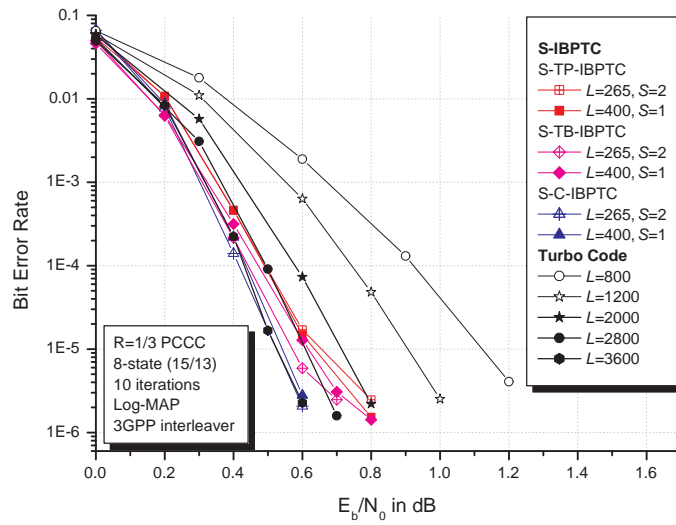


Figure 5.16: BER comparison of S-IBPTCs and the 3GPP defined turbo code of various block sizes.

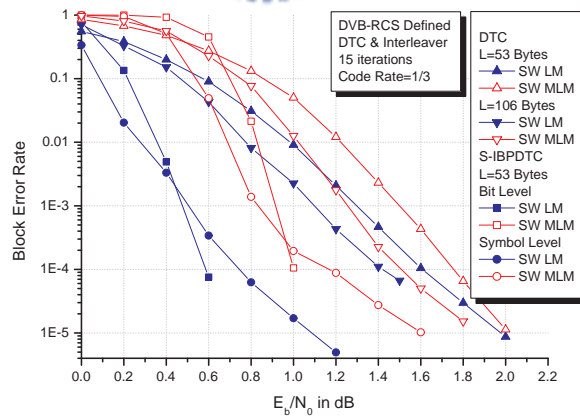


Figure 5.17: A comparison of S-IBPDTC applying bit-level and symbol-level IBP with DTC using both Log-MAP and MAX Log-MAP APP decoders.

Chapter 6

Dynamic IBPTC decoder and stopping criteria

This chapter presents a novel dynamic decoder architecture for IBPTC. A general multiple-round stopping test and a memory manager are proposed as an integrated part of the decoder. A scheduler is needed to coordinate the shared hardware resources, namely, the APP decoding units and memory storage. The scheduler can arrange the processing procedure of these APP decoders in accordance with the pipeline decoder and the dynamic decoder can reach the same performance; the decoder implementation trade-off between decoder complexity and decoding throughput is easily achievable. The dynamic decoder can implement stopping test inside to early stop decoding; the computation complexity or power consumption is further reduced. If the dynamic decoder decodes an S-IBPTC, the storage of received samples and extrinsic information can be further released in advance by the result of stopping test and the necessary storage of the dynamic decoder can be less comparing to that of the pipeline decoder. The highly reliable multiple-round stopping test provides extra information assisting in decoding neighboring unstopped blocks; this decreases both the number of average decoding rounds and error rates. Therefore the joint stopping mechanism dynamic decoder not only requires less hardware complexity but also achieves better error rate and average decoding rounds performance comparing to the pipeline decoder. The following sections will describe the dynamic decoder and the associated modification in reducing

complexity and power consumption for S-IBPTC. The multiple-round stopping test will be described in the later section.

6.1 IBP turbo coding system with stopping mechanism

This section presents a reliable stopping mechanism for an IBPTC codec system. The extrinsic information used in a conventional turbo decoder is usually generated in the course of decoding a component code and there are many well-developed soft output decoding algorithms. An error detection code that is often used in a packet switching network can also generate extrinsic information with no extra cost/complexity when the corresponding undetectable error probability is negligibly small. However, the detector output is generally used for making a stopping decision only. In order to utilize this stopping message, we can partition an information sequence into multiple blocks and these blocks are separately CRC-coded. If one block is stopped and some blocks are unstopped, the “pass” message can be passed to the unstopped blocks as the a priori information. IBPTC suits this nature and can have better performance by paying little incremental complexity and slightly extra CRC overhead. The detail will be expounded in the following subsections.

6.1.1 System model

Shown in Fig. 6.1 is a generic block diagram for a joint stopping test iterative encoding and decoding system using an IBPTC. The input data sequence \mathbf{D} is partitioned into blocks of the same length, $\{\mathbf{d}_1, \mathbf{d}_2, \dots\}$, where \mathbf{d}_i is a row vector of length $L - K_{CRC}$ representing the i th block. They are CRC-encoded into $\mathbf{u} = \{\mathbf{u}_1, \mathbf{u}_2, \dots\}$, where $\mathbf{u}_i = \{u_{i0}, u_{i1}, \dots, u_{i(L-1)}\}$ is a row vector with length L . \mathbf{u} is formed by padding at the end of each data block parity bits that are the coefficients of the remainder polynomial $r(x)$ obtained by dividing a data polynomial associated with a data block by a

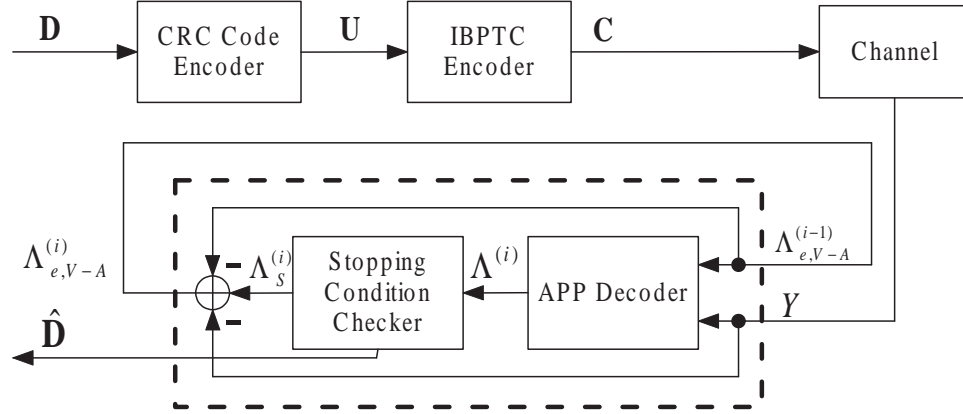


Figure 6.1: The block diagram of the proposed VTT-APP decoder applied IBP turbo coding system.

binary generator polynomial $g(x)$ of order K_{CRC} . Of course, the degree of $r(x)$ is less than K_{CRC} . The corresponding probability of undetectable error is roughly equal to $2^{-K_{CRC}}$. In other words, longer CRC codes possess better error detection capability.

The CRC encoder output \mathbf{u} and its permuted version $\mathbf{u}' = \{\mathbf{u}'_1, \mathbf{u}'_2, \dots\}$ are then encoded to form the coded sequence $\mathbf{c} = \{\mathbf{c}^0, \mathbf{c}^1, \mathbf{c}^2\}$, where $\mathbf{c}^i = \{\mathbf{c}^i_1, \mathbf{c}^i_2, \dots\}$ and the superscript i is used to denote the systematic part ($i = 0$), the first encoder's output (parity) sequence ($i = 1$) and the second encoder's output sequence ($i = 2$).

The decoder uses one or multiple APP decoding units (ADUs) like that shown in lower part of Fig. 6.1 to decode the corresponding received sequence $\mathbf{Y} = \{\mathbf{y}_1^0, \mathbf{y}_1^1, \mathbf{y}_1^2, \mathbf{y}_2^0, \mathbf{y}_2^1, \mathbf{y}_2^2, \dots\}$, where \mathbf{y}_j^i is the subsequence corresponding to \mathbf{c}_j^i ; other notations are defined in the Subsection 6.1.2. An ADU consists of an APP decoder and a stopping condition checker. It also performs the corresponding interleaving or de-interleaving and other related operations but for simplicity we do not show these operations in this figure.

The stopping condition checker applies CRC check and/or other forms of stopping

tests (STs) to verify if the APP decoder output satisfies the stopping criterion. An affirmative answer leads to the decision to stop (terminate) decoding the block in question and this is the only possible early-stopping opportunity for classic TCs. Besides such a regular early-stopping, however, there are two other early-stopping opportunities for IBPTCs since no matter whether the decoder output passes the ST, the corresponding soft output is interleaved or de-interleaved to the neighboring blocks. The ADU will then examine each related block to see if a block's content has been filled with stop-decoding decisions. If such a block is found the ADU will issue a termination decision accordingly. The ADU can also run STs on these blocks and make a termination decision. We refer to the latter two early-stopping possibilities as extended (or pre-decoding) early-stoppings.

Note that a decoding iteration consists of two decoding rounds (DRs) that are respectively responsible for decoding the pre-permuted (non-interleaved) \mathbf{c}_j^1 and post-permuted (interleaved) blocks \mathbf{c}_j^2 and CRC check is feasible for pre-permuted blocks only. Hence in the first DR one can perform both regular and extended early-stopping tests, but in the second DR, only extended early-stopping is viable unless the ST does not involve a CRC check. Examples are given in Section 6.2.4 to further elaborate this property of IBPTCs.

6.1.2 Iterative decoder with variable termination time

A conventional iterative decoder is composed of one or more APP decoders that will not stop decoding until a fixed number of decoding iterations have been performed. With an early-stopping mechanism in place, as shown in Fig. 6.1, the decoding procedure can stop (terminate) at the end of an iteration (two decoding rounds) or at the end of a DR. We refer to such a decoder as a variable termination time APP (VTT-APP) decoder or simply a VTT decoder. When an ST is included in the turbo decoding process, the test results in either a stop- or a continue-decoding decision. Given the decision, which is very useful side information, our computation of the extrinsic information and soft

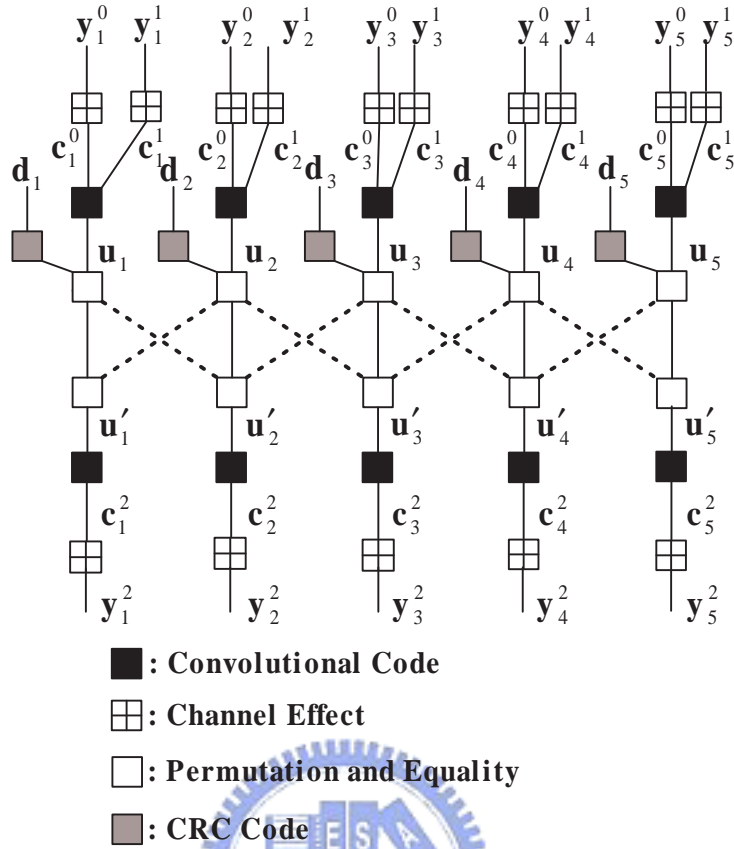


Figure 6.2: A graph representation for a CRC and S-IBPTC encoded system with interleaving span $S = 1$.

output should be modified accordingly.

Note that all STs, whether they are used in classic TCs or IBPTCs, incur additional computational complexity which is usually more than compensated for by the reduced average DRs brought about by the use of a ST.

Let $\Lambda(u) = \log \frac{p_U(u=0)}{p_U(u=1)}$ be the log-likelihood ratio of the random variable u where $p_U(\cdot)$ denotes the probability density function of u . If u_{jk} represents the k th bit of the j th block and $\Lambda^{(i)}(u_{jk})$, $\Lambda_e^{(i)}(u_{jk})$ denote the corresponding estimated log-likelihood ratio and the extrinsic information obtained at the end of the j th block's i th DR, we have [50]

$$\Lambda_e^{(i)}(u_{jk}) = \Lambda^{(i)}(u_{jk}) - \Lambda_e^{(i-1)}(u_{jk}) - L_c \cdot y_{jk}^0, \quad (6.1)$$

We assume that $\Lambda_e^{(-1)}(u_{jk}) = 0, \forall j, k$. $L_c = 4aE_s/N_0$ represents the channel reliability, where a is the signal amplitude which is usually normalized to 1 for additive white Gaussian noise (AWGN) channel, E_s being the signal energy per symbol while N_0 is the noise power spectral density.

For the i th DR of the j th data block, the VTT-APP decoder in charge uses received sequence $\mathbf{y}_j^0, \mathbf{y}_j^1$ or \mathbf{y}_j^2 and the a priori information $\{\Lambda_e^{(i-1)}(u_{jk})\}_{k=0}^{L-1}$ as its input and outputs $\{\Lambda_e^{(i)}(u_{jk})\}_{k=0}^{L-1}$ for use in the next DR as the a priori information until $i = D_{\max}$, where D_{\max} is the maximum allowed APP DRs; see Fig. 6.1.

A tentative decision \hat{u}_{jk}^i on the k th bit of the j th block at the end of the i th APP DR can be obtained by

$$\hat{u}_{jk}^i = \begin{cases} 0 & , \Lambda^{(i)}(u_{jk}) \geq 0, \\ 1 & , \Lambda^{(i)}(u_{jk}) < 0. \end{cases} \quad (6.2)$$

Let $Q(\hat{\mathbf{u}}_j^i)$ be the stopping indicator for the tentative decision vector of the j th block at the i th DR, $\hat{\mathbf{u}}_j^i = (\hat{u}_{j0}^i, \hat{u}_{j1}^i, \dots, \hat{u}_{j(L-1)}^i)$, $0 < i \leq D_{\max}$, where

$$Q(\hat{\mathbf{u}}_j^i) = \begin{cases} 1, & \text{if } \hat{\mathbf{u}}_j^i \text{ satisfies the ST} \\ 0, & \text{otherwise} \end{cases} \quad (6.3)$$

Given the ST result, the conditional soft value $\Lambda_S^{(i)}(u_{jk})$ and the extrinsic information $\Lambda_{e,S}^{(i)}(u_{jk})$ are given by

$$\Lambda_S^{(i)}(u_{jk}) = \begin{cases} \log \frac{P[u_{jk}=0|Q(\hat{\mathbf{u}}_j^i)=1]}{P[u_{jk}=1|Q(\hat{\mathbf{u}}_j^i)=1]}, & Q(\hat{\mathbf{u}}_j^i) = 1 \\ \Lambda^{(i)}(u_{jk}), & Q(\hat{\mathbf{u}}_j^i) = 0 \end{cases} \quad (6.4)$$

and

$$\Lambda_{e,S}^{(i)}(u_{jk}) = \Lambda_S^{(i)}(u_{jk}) - \Lambda^{(i)}(u_{jk}). \quad (6.5)$$

The extrinsic information $\Lambda_{e,VTT}^{(i)}(u_{jk})$ of an APP decoder then becomes

$$\Lambda_{e,VTT}^{(i)}(u_{jk}) = \Lambda_{e,S}^{(i)}(u_{jk}) + \Lambda_e^{(i)}(u_{jk}) = \Lambda_S^{(i)}(u_{jk}) - \Lambda_{e,VTT}^{(i-1)}(u_{jk}) - L_c y_{jk}^0. \quad (6.6)$$

The resulting VTT-APP decoder is shown in Fig. 6.1.

To ease the burden of computing the conditional log-likelihood function that appears in eqn. (6.4), we make the ideal assumption that the stopping test is perfect, i.e.,

$$P(\hat{u}_{jk}^i \text{ is correct} | Q(\hat{\mathbf{u}}_j^i) = 1) = 1, \forall k.$$

With this perfect stopping decision assumption, eqn. (6.4) becomes

$$\Lambda_S^{(i)}(u_{jk}) = \begin{cases} \Lambda^{(i)}(u_{jk}) \cdot \infty, & Q(\hat{\mathbf{u}}_j^i) = 1 \\ \Lambda^{(i)}(u_{jk}), & Q(\hat{\mathbf{u}}_j^i) = 0 \end{cases}, \quad (6.7)$$

and eqn. (6.6) is modified accordingly.

The perfect stopping assumption actually makes the computation of extrinsic information or soft output easier as when the tentative decision vector $\hat{\mathbf{u}}_j^i$ meets the stopping condition, then $\Lambda_{e,VTT}^{(i)}(u_{jk})$ has only two values $\pm\infty$. When the perfect stopping assumption is approximately true (say, the false stopping probability is less than 10^{-5}), a practical approximation is to assign a fixed large number to $\Lambda_{e,VTT}^{(i)}(u_{jk})$. However, it should be noted that, after interleaving or de-interleaving, the large metric value will be passed to neighboring blocks and then to the corresponding partial path metric computers, eliminating other branches which are not associated with these bits. Hence the passing of the extrinsic information of these perfect detected bits to neighboring blocks further reduce the complexity of the associated APP decoder. Moreover, as the APP decoder selects survivor branches based on the relative magnitudes of the partial path metrics only, the actual value assigned to $\Lambda_{e,VTT}^{(i)}(u_{jk})$ is immaterial. In fact, it can be as simple as a binary sign telling the APP decoder which branches should be eliminated.

All these nice features depend, besides the IBP design, on the availability of a highly reliable ST such that the perfect stopping assumption holds with a probability close to 1, which is the subject of the stopping mechanism design. Note that although there is no perfect ST and the probability that a ST gives a wrong block stopping decision is nonzero, the influence of these wrong indications result in no catastrophic failure as our numerical results will demonstrate in Section 6.4.

6.1.3 Graphical representation of an IBPTC and CRC codes

Fig. 6.2 is a graphical representation for the system of Fig. 6.1 with an S-IBPTC and CRC codes and is extended from Fig. 5.4 with additional CRC code functions. This graph plots the S-IBPTC exploiting a symmetric S-IBP interleaver of the span $S = 1$ and an input data sequence \mathbf{u} partitioned into five blocks. Each block is encoded by CRC code. The dark, gray, crossed and blank squares represent respectively the functions of convolution codec, CRC code, S-IBP interleaving and the channel effect. The CRC function is connected to the permutation and equality node. This graph further indicates the relation between IBPTC and CRC codes and shows the message-passing for the VTT-APP decoder.

Section 5.3 has elaborated the message-passing regarding to an S-IBPTC. An S-IBPTC decoder can exploit information collected from $4SI + 1$ adjacent blocks in I iterations as the number of block is large enough, e.g. the message of the block \mathbf{u}_3 in Fig. 6.2 can be passed to \mathbf{u}_1 and \mathbf{u}_5 in one iteration. The extra CRC nodes in Fig. 6.2 provide extra information after each APP DR. If the decoded block passes the CRC check condition, the CRC node generates extra extrinsic information for the block and the information will be passed to the other blocks as the a priori information to decode. Therefore the graph brings out the message-passing associated with the further CRC functions.

6.2 Dynamic decoder and the associated issues

This section describes a low complexity and flexible decoder architecture for IBPTC comparing to the pipeline decoder. The required number of APP decoders is flexible and the implementation cost can be eliminated to the least. The decoder further applies an ST to terminate decoding of blocks and reduce memory usage. The decoder can also apply the VTT-APP decoder to provide better performance. Therefore this dynamic decoder provides the most flexibility in implementing an IBPTC decoder with least

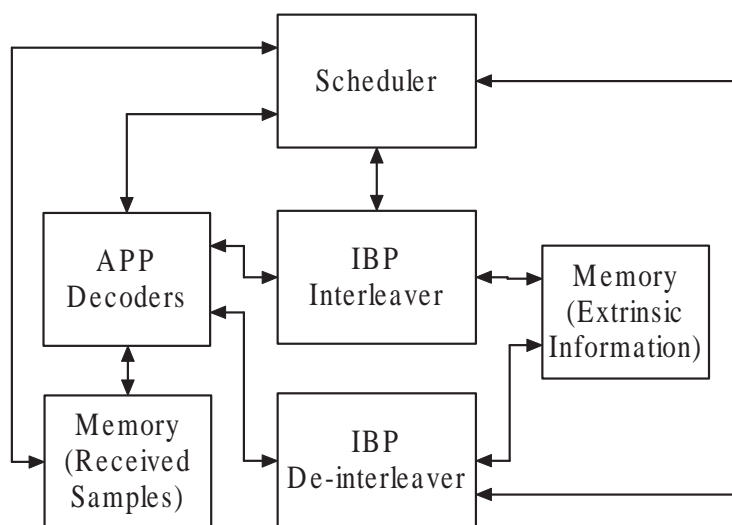


Figure 6.3: The block diagram of IBPTC dynamic decoder.

hardware complexity.

6.2.1 Dynamic decoder

Implementing a dynamic decoder shown in Fig. 6.3 has more flexibility between decoder complexity and decoding throughput comparing to implementing the pipeline decoder. The hardware complexity of the pipeline decoder is linear to the maximum number of decoding round D_{\max} because the decoder is composed of D_{\max} APP decoders. The throughput of the pipeline decoder is the same as the throughput of each APP decoder. If the designated decoding throughput is less than the decoding throughput of the pipeline decoder, the pipeline decoder becomes an over-design. In fact, we can decouple the decoding of an IBPTC into multiple sub-decodings associated with these blocks and schedule these sub-decodings. Therefore the dynamic decoder applies a scheduler to coordinate these APP decoders to decode an IBPTC. Since we can schedule these sub-decodings, the fixed number of APP decoders is not necessary and the complexity of dynamic decoder can vary with the decoding throughput. Even the desired throughput exceeds the throughput of the pipeline decoder, we also can imple-

ment more APP decoders to reach the desired throughput if the IBP interleaver is the contention-free interleaver which describes in Chapter 4. The dynamic decoder provides more implementation options than the pipeline decoder.

6.2.2 Decoding delay

Decoding delay is perhaps the most important issue in high speed decoder design. The decoding delay associated with an S-IBPTC for a parallel decoder is minimized by using a proper decoding schedule. Even if only one APP decoder is used, as we will see shortly, the decoding schedule still plays a pivotal role in minimizing the decoding delay of an S-IBPTC. The delay associated with B-IBPTC is similar to classic TC but B-IBPTC provides more decoding options due to its IBP nature. The decoding delay can also be reduced by a proper schedule as S-IBPTC and will be discussed later.

We first analyze the decoding delays when only one APP decoder is used. The single-round interleaving (or de-interleaving) delay is proportional to the interleaving delay. But the total decoding delay is a much more complicated issue. For a decoder that uses a single ADU, the decoding delay depends mainly on three variables: the single-round interleaving delay (SRID), the single-round APP decoding delay, and the number of decoding iterations. As the single-round APP decoding delay (speed) is usually much less than the SRID, we ignore the APP decoding delay in the subsequent discussion.

For the first decoding of each incoming block, there can be zero waiting time, but for later DRs the corresponding delays depend on, among other things, the decoding schedule used. With the same block size, the decoding delay of the first received block for the classic TC is definitely shorter than that for the S-IBPTC. But if one considers a period that consists of multiple blocks (otherwise one will not have enough blocks to perform inter-block permutation) and takes the decoding schedule into account, then the average decoding delay difference can be completely eliminated. This is because the APP decoder (including the interleaver and deinterleaver) will not stay idle until

Classic TC/S-IBPTC

APP Decoding Round \ Block Index	1	2	3	4	5	6	7
1	1 1	5 2	9 4	13 7	17 11	21 15	25 19
2	2 3	6 5	10 8	14 12	18 16	22 20	26 23
3	3 6	7 9	11 13	15 17	19 21	23 24	27 26
4	4 10	8 14	12 18	16 22	20 25	24 27	28 28

Figure 6.4: A comparison of exemplary decoding schedules for classic TC and S-IBPTC when decoding 7 blocks with 2 iterations (four decoding rounds). The numbers in the two rectangular grid-like tables represent the order the APP decoder performs decoding.

all blocks within the span of a given block are received. Instead, the APP decoder will perform decoding-interleaving or deinterleaving operations for other blocks according to a predetermined decoding schedule before it can do so for the given block (and the given DR).

If we define the total decoding delay as the time span between the instant a decoder receives the first input sample (from the input buffer) and the moment it outputs its last decision then it is possible that both the S-IBP and the classic approaches yield the same total decoding delay even if only one APP decoder is used. We use the following example and Fig. 6.4 to support our claim; its generalization is straightforward.

Suppose we receive a total of 7 blocks of samples (in a packet, say) and want to finish decoding in 2 iterations (4 DRs) and a schedule for both classic TC and S-IBPTC are shown in Fig. 6.4. The first block of the classic TC is decoded by the first 4 decoding rounds (the leftmost column) but that of the S-IBPTC is decoded by the first, third, sixth and tenth decoding rounds. One can easily see that a classic TC decoder would output the first decoded block in 4 DT cycles, where DT is the number of cycles needed to perform a single-block APP decoding plus SRID. The S-IBPTC decoder, on

the other hand, needs 10 DT cycles to output its first decoded block. However, if one further examines the decoding delays associated with the remaining blocks, then one finds they are 8, 12, 16, 20, 24, and 28 DT cycles for the classic TC decoder while those for the S-IBPTC decoder are 14, 18, 22, 25, 27 and 28 DT cycles, respectively. So in the end, both approaches reach the final decision at the same time.

It can be shown that, for a decoder with D_{\max} DRs and $S = 1$, both decoders result in a constant delay of $\frac{D_{\max}(D_{\max}-1)}{2}$ DT cycles between two adjacent output blocks, except for the first block and the last $D_{\max} - 1$ blocks. For an $S = 1$ S-IBPTC, the decoder requires a first-block decoding delay of $\frac{D_{\max}(D_{\max}+1)}{2}$ DT cycles while that for the classic TC is only D_{\max} DT cycles. The inter-block decoding delays, i.e., decoding latency between two consecutive output blocks, for the last $D_{\max} - 1$ output blocks of the S-IBPTC decoder using a decoding schedule similar to that shown in Fig. 6.4 (e.g., the one shown in Fig. 6.5) form a monotonic decreasing arithmetic sequence $\left\{ \frac{D_{\max}(D_{\max}-1)}{2} - 1, \frac{D_{\max}(D_{\max}-1)}{2} - 3, \dots, 0 \right\}$ (in DT cycles). The inter-block decoding delay of a classic TC decoder remains a constant D_{\max} DT cycles. On the average, both codes give the same inter-block decoding delay.

Although we have assumed a stream-oriented scenario so far, our arguments are valid for the conventional block-oriented consideration as well. It is thus of paramount importance that we recapture the IBP concept from the block-oriented viewpoint before returning to the main discourse.

Consider the example illustrated in Fig. 6.4. For a classic TC with an interleaving (block) size of $7L$ bits, the first-block decoding delay for a 2-iteration single-ADU decoder is 28 DT cycles. But if one divides this $7L$ -bit block into 7 subblocks and uses a special block-oriented interleaver which performs successive intra-subblock and inter-subblock permutations on these subblocks, the corresponding (2-iteration single-ADU) decoding delays in DT cycles for these subblocks are 14, 18, 22, 25, 27 and 28, respectively. Therefore, although both code structures result in identical total decoding delay the

IBPTC structure is able to supply partial decoded outputs much earlier. This feature, when combined with proper intra-(sub)block and inter-(sub)block interleaving rules, multiple ADUs, optimized decoding schedule and implementation resource management, become very beneficial for high speed applications. More importantly, it can be shown by computer simulations that a turbo code with such an interleaver does not yield performance inferior to that of a classic TC with a block-oriented interleaver (e.g., 3GPP interleavers) of the same size.

6.2.3 Memory contention and decoding schedule for multiple ADUs

The above assessment on the encoding/decoding delay is made under the assumptions that both codes use the same block size L , no early stopping mechanism is applied, and a single ADU is used. The delay will be shortened if the latter two assumptions are removed. In particular, the decoding delay can be reduced significantly by using multiple ADUs for parallel decoding. When iterative APP decoding is performed by multiple ADUs, these ADUs have to access memory via interleaver (or deinterleaver) for extrinsic information update and exchange. To have the maximum delay reduction, the interleaver should also have a parallel structure to avoid memory access collision. It can be shown that the structure of IBP interleavers allows flexible degrees of parallelism and highly parallel memory access. In fact, *Theorem 3.1* implies that a good IBP interleaver should possess the local-invariant property that preserves the relative position within a block during the IBP process. This property promises contention-free across the span (parallel-decodable blocks) of the IBP interleaver. Furthermore, like the contention-free interleaver design presented in [92], closed-form contention-free IBP rules are available, and more importantly they guarantee some good distance properties for the associated IBPTC.

Just the same as the single ADU case, the decoding schedule for multiple ADUs is a critical design concern. An example of decoding an S-IBPTC with multiple APP

Block Number \ APP Decoding Round	1	2	3	4	5	6	7	8	9	10	11	12
1	a ₁₁	b ₁₁	c ₁₁	d ₁₁	a ₂₁	b ₂₁	c ₂₁	d ₂₁	a ₃₁	b ₃₁	c ₃₁	d ₃₁
2	b ₁₂	e ₁₂	d ₁₂	a ₂₂	b ₂₂	c ₂₂	d ₂₂	a ₃₂	b ₃₂	c ₃₂	d ₃₂	a ₄₂
3	c ₁₃	d ₁₃	a ₂₃	b ₂₃	e ₂₃	d ₂₃	a ₃₃	b ₃₃	c ₃₃	d ₃₃	a ₄₃	b ₄₃
4	d ₁₄	a ₂₄	b ₂₄	c ₂₄	d ₂₄	a ₃₄	b ₃₄	e ₃₄	d ₃₄	a ₄₄	b ₄₄	c ₄₄
5	a ₂₅	b ₂₅	c ₂₅	d ₂₅	a ₃₅	b ₃₅	c ₃₅	d ₃₅	a ₄₅	b ₄₅	c ₄₅	d ₄₅
6	b ₂₆	c ₂₆	d ₂₆	a ₃₆	b ₃₆	c ₃₆	d ₃₆	a ₄₆	b ₄₆	c ₄₆	d ₄₆	a ₅₆

Figure 6.5: A multiple zigzag decoding schedule for an S-IBPTC with the span $S = 1$.

decoders is given in Fig. 6.5. Although both S-IBPTCs and classic TCs can use multiple decoders for parallel decoding and apply an early stopping mechanism to shorten the decoding latency, we will prove numerically in Section 6.4 that the former class does derive much more benefit in block error rate (BLER) performance.

We have demonstrated the importance of the decoding schedule in minimizing the decoding delay. Parallel decoding is a popular design option to shorten the latency. Fig. 6.5 shows a multiple expanding-window zigzag schedule table for decoding an S-IBPTC with the span $S = 1$ and four ADUs, denoted respectively by **a**, **b**, **c** and **d**. Data blocks processed in the odd rows are in the original (pre-permutation) order while those processed in the even rows are in the interleaved (post-permutation) order. Each dashed or dotted zigzag curve represents the schedule for an ADU. The symbol \mathbf{x}_{mn} denotes the n th DR of the m th phase in the ADU \mathbf{x} 's schedule, where a DR represents the APP decoding of a pre- or post-permuted block and the associated interleaving or de-interleaving and the m th phase refers to the m th parallel line associated with an ADU's decoding schedule. Obviously, the m th decoding phase of \mathbf{x} is followed by the $(m + 1)$ th decoding phase to its right.

Taking the decoding schedule of ADU \mathbf{c} as an example, its first DR of the first phase \mathbf{c}_{11} corresponds to the first DR of Block 3 while the first phase' second DR \mathbf{c}_{12} corresponds to the second DR of Block 2. \mathbf{c}_{12} can be performed, as the schedule table shows, after Blocks 1,2,3 have been decoded once and the corresponding extrinsic information output has been inter-block interleaved so that the post-permuted Block 2 has all a priori information needed for a new DR. \mathbf{c} finishes its first phase after \mathbf{c}_{13} is done. It then proceeds with the first DR of the next phase \mathbf{c}_{21} , i.e., the first DR of Block 7. An ADU can not start a new DR until the DR on its top is completed, e.g., \mathbf{a}_{2k} , $k > 2$ cannot start unless the DR corresponding to \mathbf{d}_{12} is finished.

An ADU can make regular stopping decisions in odd rows' DRs and extended stopping decisions (ESDs), unless a non-CRC-based ST is used. CRC-based ST makes regular stopping decision in odd row. ESD is generally made to stop decoding a block in even row by checking if all related blocks pass CRC condition or the shortage of memory which will be described in the following subsection. For example, in \mathbf{a}_{23} (\mathbf{c}_{25}) we check if Block 3 passes the ST and early stopping on this block becomes effective if affirmative. ADU \mathbf{a} (\mathbf{c}) then go on to examine whether \mathbf{a}_{24} (\mathbf{c}_{26}) is necessary by checking whether both \mathbf{c}_{13} and \mathbf{d}_{13} (\mathbf{a}_{25} and \mathbf{b}_{25}) pass the ST as well. When this condition is satisfied, decoding of Block 2 is terminated. On the other hand, in \mathbf{b}_{24} no ST is performed but after de-interleaving its output we run a ST on the content of \mathbf{b}_{25} before next APP decoding round, which contains de-interleaved outputs from \mathbf{d}_{14} and \mathbf{a}_{24} . We stop decoding Block 2 and \mathbf{b}_{25} is no longer needed (because of our schedule and the S-IBP structure, \mathbf{c}_{25} and \mathbf{d}_{25} can not yet be verified although extrinsic information from \mathbf{b}_{24} will be passed on to them) if the ST result is positive.

6.2.4 Memory management

From the above discussion, it is clear that decoding many blocks at the same time requires no small storage area for ASIC or DSP implementation. One should therefore

try to make the most of the memory space available. The decoder needs space to store (I) received samples undergoing decoding, (II) extrinsic information, (III) decoded bits to be forwarded to a higher layer for further processing, and (IV) received samples awaiting decoding. The management of the last category, assuming no buffer overflow, requires only an indicator signal to forward a new block of received samples to the part of the storage area designated for category (I) that was just released due to a stopping decision.

Category (III) is needed because of the stopping time variation across blocks. Its management is straightforward and, besides, it requires much less storage space. As mentioned in Subsection 6.1.2, assigning the extrinsic values for ST-approved bits a constant large value is equivalent to using a (special) binary-valued bit to indicate which partial paths should survive in the APP decoding process. Hence the decoded bits serve the dual purposes of representing the decoder decisions and bookkeeping the survivor paths. The management of categories (I) and (II), however, needs more efforts and careful considerations.

As long as the probability of termination-defying blocks exists, practical latency consideration will force us to set an upper limit D_{\max} on the number of DRs. It can be shown that an unterminated block prevents the decoder from discarding \mathbf{y}^2 associated with those terminated blocks within its span. When the number of blocks that terminate at or around the D_{\max} th DR is large so will be the memory required. Hardware constraint thus imposes another threshold M_{\max} , the maximum affordable (allowable) memory units (MU) where an MU refers to the space for storing categories (I) and (II) associated with a block of data in the decoder. As our sole purpose is to demonstrate the critical role a memory manager plays in the VTT-APP decoder, we assume, for simplicity, that the same number of bits is used to represent the extrinsic information of a bit and the corresponding received sample. An MU is thus assumed to contain KL bits, where a K -bit word is used to store either the extrinsic information or received baseband sample associated with a transmitted bit.

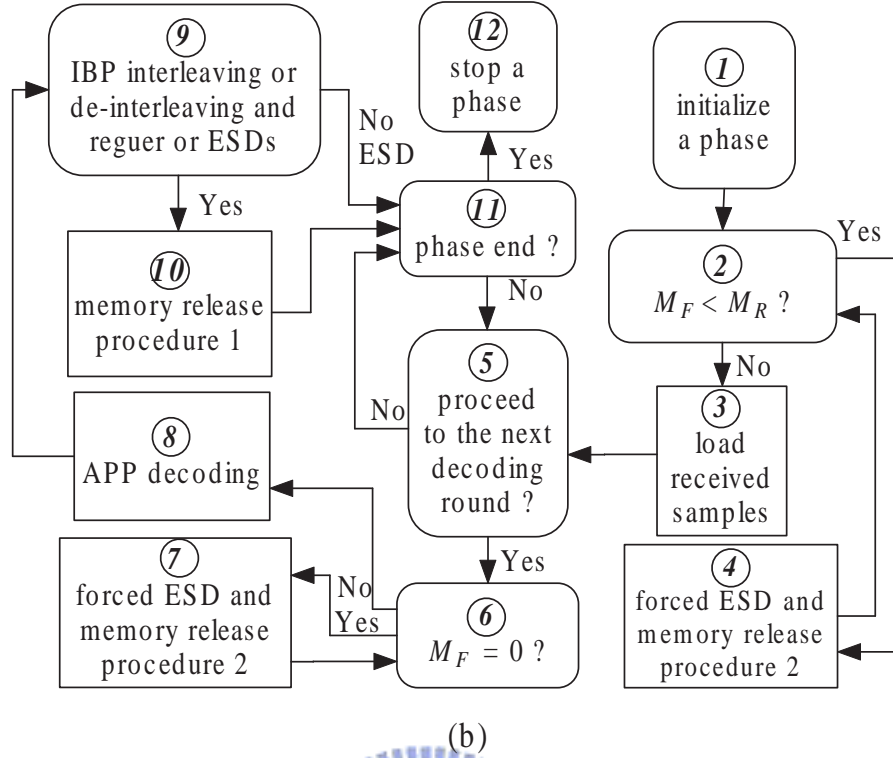


Figure 6.6: A joint memory management and IBPTC decoding procedure.

Because of the stopping time variation nature of our decoder, a memory manager has to take into account both thresholds, D_{\max} and M_{\max} so as to optimize the performance. When a block has failed to pass the ST for D_{\max} times, it will automatically be discarded and the MUs storing the corresponding categories (I) and (II) information are released accordingly. Chances are more than one block that reach the threshold D_{\max} simultaneously and it is even more likely that the decoder runs out of MUs before a block reaches the threshold D_{\max} . For both cases, one should then give up decoding one or some of the unterminated blocks. It is both reasonable and intuitively-appearing to terminate the most ancient block, i.e., the one which has failed the ST most often. We refer to these memory shortage induced stopping as forced early stopping.

Fig. 6.6 shows a finite-memory IBPTC decoding procedure for one phase of an ADU. The procedure involves APP decoding, interleaving, deinterleaving, regular and extended stopping decisions, and memory check and release. The last two operations are

collectively called the memory management scheme which is responsible for verifying if there is enough memory during the decoding process and make a proper memory-release decision if there is not enough storage space. As there is no computation involved at all, the complexity is moderate at most.

Denote by M_F , M_d and M_R , the numbers of free (unused) MUs, ADUs, and the required MUs for storing one received block. It follows that $M_R = x$ for a rate $R = 1/x$ turbo code. The decoder is initialized with $M_F = M_{\max}$. An ADU begins a phase by checking if $M_F > M_R$ (Box 2) where the additional MU is for storing extrinsic information. If M_F does not meet the condition, the memory manager determines which block is to be discarded, makes a forced ESDs, and releases the related storage space (Box 4). Otherwise, the decoder moves the received samples of the new block from where they were saved (in the buffer area) to the corresponding category (I) MUs (Box 3).

Deciding which block is to be given up is simple and clear since our decoding schedule allows only a single most ancient block in its left-most active column at any time. When a forced ESD is made the ADU makes hard decisions on the block to be discarded and releases the related categories (I) and (II) MUs. As the discarded block is always the most ancient block and our decoding schedule is such that all blocks to its left must have been terminated for one reason or another, we are left with the problem of dealing with the related unstopped blocks, the S adjacent blocks to its right for S-IBPTC, if they have not been terminated. At least two alternatives exist for solving this problem. The first solution, which leads to better performance at the cost of higher complexity, is to interleave or de-interleave the extrinsic values for use in decoding the related unstopped blocks, the S blocks to its right for S-IBPTC, without further updates. The second one is to make hard-decisions (stop any further decoding) on all related unstopped blocks, S blocks within its (right) span for S-IBPTC, releasing their category (I) MUs while keeping their category (II) MUs for use in decoding other related blocks.

At beginning of each DR, we ask the decoder whether the scheduled DR is needed (Box 5). Unless the decoder has been notified to by-pass the ensuing DR, we still have to ask if the space for storing the extrinsic information of the coming DR is available. When such a space is not available ($M_F = 0$) the decoder has to find room for the next DR by discarding the most ancient unterminated block and following the memory release procedure described above (Box 7). The operations in Box 9 include those described in the last paragraph of Subsection 6.1.2. When a regular or extended stopping decision is made, the memory manager releases the corresponding category (II) and parts of category (I) memory (Box 10), memory release procedure 1) and notifies the decoder that further decodings on these blocks are no longer necessary.

6.3 Multiple-round stopping tests

Early stopping mechanism offers the extra benefit of lower the computing power needed for achieving a given performance and further accelerates an iterative decoder's decoding speed. The issue of (decoders') stopping criteria has been widely discussed [50, 88, 65, 4]. These criteria can be classified into four categories: (i) cross entropy (CE) stopping criteria, (ii) sign check (SC) stopping criteria, (iii) soft value (SV) stopping criteria and (iv) cyclic redundancy check (CRC) stopping criteria. The last one guarantees the correctness of decoded bits with a high probability while the others only promise the convergence of the decoded bit sequence. The SC and the CRC stopping criteria use the bit operations only while the remaining two categories operate over the floating-point domain. Moreover, CE and SV stopping criteria have to optimize threshold for different channel conditions whence is less robust. On the other hand, CRC codes have been widely used in the data link or higher layer as part of the error-control mechanism and is an indispensable component of a packet-oriented data communication system. Using CRC codes as a part of the stopping criterion thus causes little or no extra complexity.

[65] summarized various STs for turbo decoders using sign check, soft values and CRC checks. The sign check stopping test (SCST) compares the tentative decoded bits from two successive rounds. A tentative decoded block passes the test if most or all of them are consistent. The soft value stopping test (SVST) compares the soft value(s) with a threshold; the soft values can be the reliability of tentative decoded soft bits, the average soft value of a block, the extrinsic value of the least reliable bit etc. The CRC stopping test (CRCST) uses the CRC result to decide if further decoding of a block is needed. SCST and CRCST operate over bit level but SVST operates over the real-domain. The performance of SVST is subject to the choice of the threshold which, in turn, is a function of the channel condition and code structure. Moreover, the convergence rate of soft bit values also depends on the above two factors [65, 4]. In short, the classes of CRCST, SCST or their variations have the complexity and robustness advantages over the class of SVST.

6.3.1 A general algorithm

All early stopping tests are sequential in nature. They either compare or manipulate some values corresponding to two consecutive DRs, or just check a single DR output to make a stop-or-continue decoding decision. In contrast, our proposed tests make a stop-decoding decision based on multiple observations and are thus referred to as multiple-round stopping tests (MRSTs)

It is well-known that a statistical decision based on a single observation is inferior to that based on multiple observations which, however, require a longer observation time (or equivalently, larger sample size). The MRST has the distinct capability of balancing performance (reliability of the test) and cost (time or sample size needed to make a termination decision). A dismissal on a decoder output is issued as soon as it fails a single test but a decision to stop decoding a block has to wait until the same block is verified by several rounds of test. Therefore, incorrect tentative decoder outputs are

quickly discarded while any final decision on a block is prudently made. While the first round of an MRST provides an initial tentative decision, the additional verification test rounds greatly reduce the probability of false stopping and give more robust and reliable decision; this avoids spreading incorrect information to neighboring blocks for IBPTC VTT-APP decoder. An m -round stopping test using a short CRC-8 code gives a false detection probability similar to that of using a long CRC- $8m$ code but with only $1/m$ overhead bits. Moreover, since a correct stopping on a certain block helps bringing earlier stoppings to its adjacent blocks for the IBPTC decoder, the average decoding delay is shortened as well.

A flow chart of the general MRST is shown in Fig. 6.7. In this figure, i is used to denote the i th DR, p represents the number of times a block has passed a ST and can be regarded as a quality indicator, m is the required quality condition and D_{\max} is the maximum number of DRs allowed. Either $p = m$ or $i = D_{\max}$ will force the decoding process to be terminated. As discussed in Subsection 6.1.2, an ST is performed at the end of an iteration (even DRs) or the beginning of an odd DR. For the latter case, an ST means checking if all pre-permuted blocks within its span have satisfied the stopping condition. A special case of MRST is the multiple-round SCST of [65]. It was found that the block error rate performance improves as the number of test rounds increases.

As mentioned before, we shall not consider the class of SVSTs. Multiple-round CRCST, SCST and a hybrid CRC-SC ST are briefly defined in the following.

6.3.2 T1.m: the m -round CRCST

This scheme is based on an m -round CRC test. A block is said to pass the m -round CRCST if all m consecutive tentative decision vectors $\hat{\mathbf{u}}_j^{i-m+1}, \hat{\mathbf{u}}_j^{i-m+2}, \dots, \hat{\mathbf{u}}_j^i$ succeed in passing the same CRC test, i.e., $I_{CRC}(\hat{\mathbf{u}}_j^l) = 1, l = i - m + 1, i - m + 2, \dots, i$ and $i \leq D_{\max}$, where

$$I_{CRC}(\hat{\mathbf{u}}) = \begin{cases} 1, & \hat{\mathbf{u}} \text{ passes CRC condition} \\ 0, & \text{otherwise} \end{cases} . \quad (6.8)$$

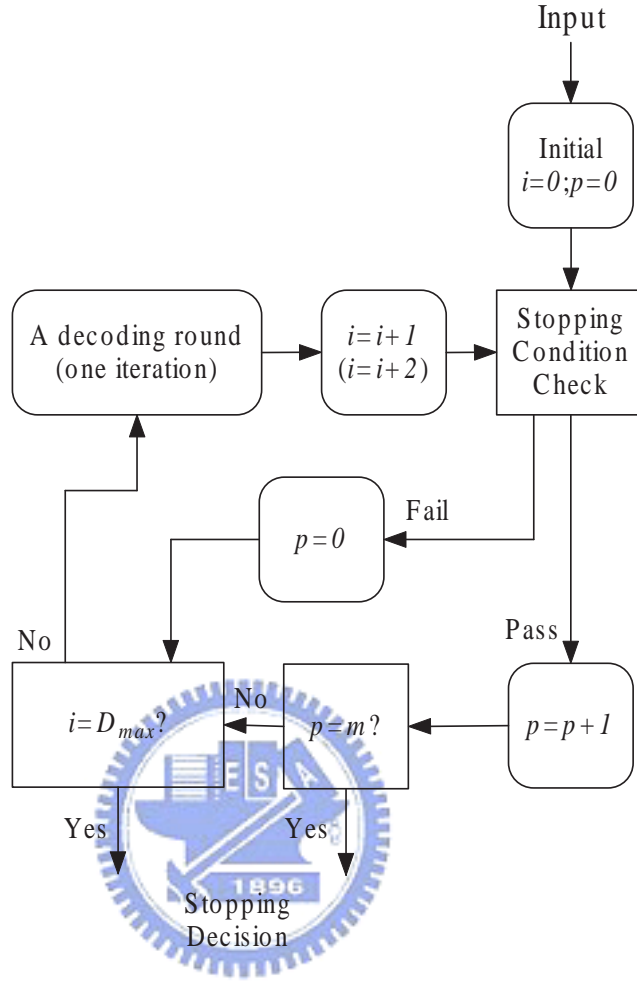


Figure 6.7: Flow chart of a general m -round stopping test.

As the error detection capability of a CRC code is an increasing function of the code length, one can trade the order m for the code length.

6.3.3 T2.m: the m -round SCST

This ST [65] compares tentative decoded bits in m ($m \geq 2$) consecutive DRs or iterations. The decoder stops when the n th tentative decision vector, $i \leq D_{\max}$, are the same with the previous $m - 1$ tentative decision vectors, i.e.,

$$\hat{u}_{jk}^{i-m+1} = \hat{u}_{jk}^{i-m+2} = \dots = \hat{u}_{jk}^i, \quad \forall k, \quad 0 \leq k < L. \quad (6.9)$$

Note that MR-SCST checks the convergence of tentative decisions, it does not guarantee the convergence to the correct decisions.

6.3.4 T3.m: the m -round hybrid stopping test (MR-HST)

Unlike classic TCs, errors in STs for IBPTC will propagate to different blocks and might lead to a catastrophic consequence. A highly reliable ST can be obtained by increasing m or it can be obtained by incorporating multiple criteria in a single round. A block that passes both CRC and SC tests is more reliable than one that passes only a single test.

Hence, we suggest the hybrid stopping criterion

$$I_{CRC}(\hat{u}_j^l) = 1, \forall l, i - m < l \leq i, i \leq D_{\max}, \quad (6.10)$$

and

$$\hat{u}_{jk}^{i-m+1} = \hat{u}_{jk}^{i-m+2} = \dots = \hat{u}_{jk}^l, \forall k, 0 \leq k < L, i \leq D_{\max}. \quad (6.11)$$

If the CRC-8 is used, the undetect error probability is approximately 2^{-8} only. The probability that the sign check does not match the CRC result is of the order 2^{-16} or 2×10^{-5} due to the CRC test is equivalently passed twice. Using a longer CRC code increases the reliability of a CRC stopping test but it also induces an increased overhead. Additional sign consistency check is the price we paid for using this stopping mechanism to cut down the CRC overhead but the cost is little.

6.3.5 Genie stopping test

Genie ST is a hypothetic ideal test that is capable of verifying the tentative decision vector without error. The performance of this ideal test is used as the ultimate bound for reference purpose.

At the first glance, we might expect the hybrid test or higher-order (larger m) tests to take more DRs since a received block is less likely to pass both SC and CRC or a

higher-order requirement. But the fact is that a correct block decision, through the IBP interleaving, will help other blocks to meet the stopping condition sooner while an incorrect one tends to have an adverse effect. Our numerical experiment indicates that the hybrid test not only gives better performance but also requires less average DRs. This is another advantage of IBPTCs that is not shared by classic TCs.

6.4 Simulation results

The simulation results reported in this section is based on the following assumptions and parameters. The component code of the rate=1/3 TC, $G(D) = \left[1, \frac{1+D^2+D^3}{1+D+D^3}\right]$, and the CRC-8(="110011011") code used are the same as those specified in the 3GPP standard [1] except that the component code is tail-biting [106] encoded. The APP decoder uses the Log-MAP algorithm and the S-IBP of Table 5.1 while the interleaving length and S-IBP span are left as variables; $M_R = 3$ MUs and $N = 1000$ per simulation run are assumed. Except for the Genie ST, our simulations do not assume a perfect stopping test for a block.

The effects of various STs on the S-IBPTC VTT-APP decoder performance for the system with $S = 1$, $L = 400$, $D_{\max} = 30$ and tail-biting encoding are shown in Figs. 6.8 and 6.9. Multiple-round CRCST, SCST and HST are considered. For comparison, we include performance curves of the decoder using the genie ST, that with fixed 20 and 30 DRs (10 and 15 iterations) and, for reference purpose, that of the classic TC with block length $L = 800$ using the genie ST with $D_{\max} = 30$.

Block error rate performance improves as the number of test rounds m increases no matter which ST is used. Fig. 6.8 shows that T1.3 outperforms T1.2 for E_b/N_0 greater than 0.3 dB. Tests using sign-check alone, T2.3 and T2.5, are inferior to other stopping tests since, as mentioned before, the class of sign-check tests check if decoded bits converge but does not guarantee the correctness of the tentative decoded vectors. Incorrect stopping decisions will spread false information to the neighboring blocks through in-

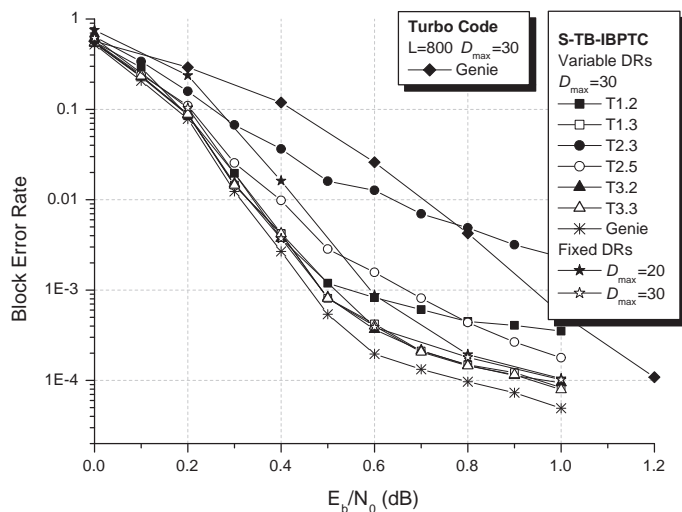


Figure 6.8: Block error rate performance of various stopping tests; no memory constraint; $D_{\max} = 30$ DRs.

terleaving and result in degraded performance. T1.3, T3.2, T3.3 and the one with fixed 30 DRs yield the best performance and they are almost as good as the genie ST. Using T3.2 for early stopping, the S-IBPTC has 0.4 ~ 0.6 dB gain against the classic TC for $\text{BLER}=10^{-3} \sim 10^{-4}$ although the average decoding delay per DR for both codes are about the same.

Fig. 6.9 shows the average DR performance of various STs. Except for the two sign-check tests, all STs require less than 20 or 10 APP DRs (10 or 5 iterations) when E_b/N_0 is greater than 0.2 or 0.6 dB. Considering both block error rate and average latency performance, we conclude that, among the STs we have examined, T3.2 is the best choice.

The numerical results presented so far assume no memory constraint. Figs. 6.10 and 6.11 reveal the impact of finite memory size for the system that employs a T3.2-aided VTT-APP decoder and the memory management algorithm of the previous section with block length $L = 400$, the span $S = 1$ and $M_d = 1$. Fig. 6.10 shows block error rate performance for different memory constraints. For convenience of comparison, we also

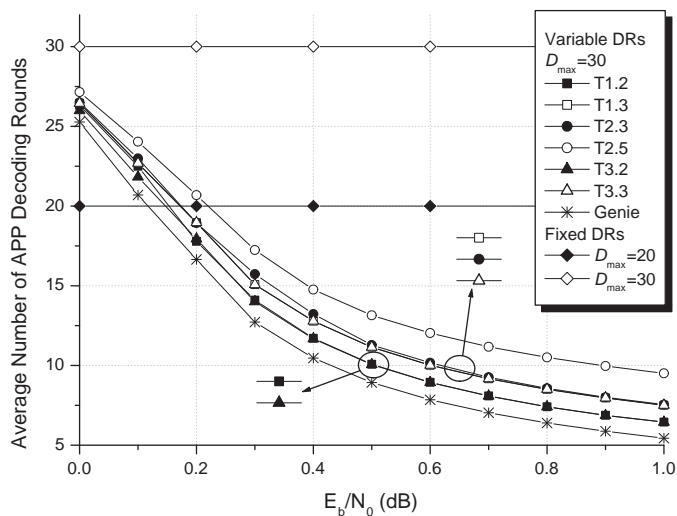


Figure 6.9: Average APP DR performance of various stopping tests; $D_{\max} = 30$ DRs, no memory constraint.

present three cases without memory constraint, one with $D_{\max} = 200$, the other two with fixed DRs. It is reasonable to find that larger memory sizes give better performance. At higher $E_b/N_0 (> 0.8$ dB), all performance curves converge to the same one since all VTT-APP decoders finish decoding after only a few DRs (see Fig. 6.11) and memory size is no longer a problem. The fact that the cases $D_{\max} = 100$ with 100 MUs, and $D_{\max} = 30$ with 100 MUs give almost identical performance indicates that increasing D_{\max} beyond a certain number (30 in this case) can not improve block error rate performance and the memory size becomes the dominant factor. Performance for the decoder with $D_{\max} = 200$ and no memory constraint (it can be shown that 804 MUs is sufficient for this case, which is at least eight time larger than that required by other decoders) is clearly better than the other decoders when $E_b/N_0 < 0.6$ dB but this edge is gradually diminished after 0.6 dB.

The average DR performance is given in Fig. 6.11. For $E_b/N_0 \geq 0.5$ dB, all VTT-APP decoders need less than or equal to 10 DRs (5 iterations). But when $E_b/N_0 < 0.3$ dB, the performance curves are distinctly different—if we do not impose a memory

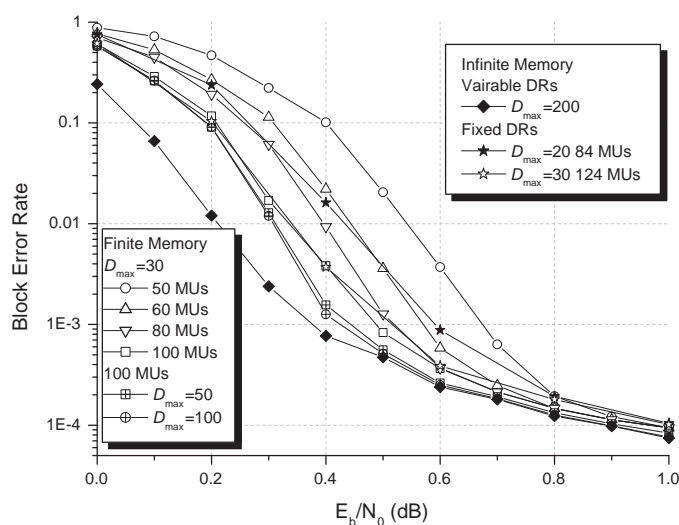


Figure 6.10: The effect of memory constraint and management on the block error rate performance. Curves labelled with infinite memory are obtained by assuming no memory constraint; “fixed DRs” implies that no early stopping test is involved.

constraint, the average DR will increase significantly as E_b/N_0 decreases. Most of the computation effort will be wasted, so is the memory. In other words, at the low E_b/N_0 region, ST can not offer early stopping decision. Imposing a memory constraint and invoking a proper memory management algorithm provide a solution that forces early stoppings, saving computing power and memory at the cost of a small performance loss. Finally, we find that, comparing with our proposed schemes, the two decoders with fixed DRs (20 and 30) usually need much more memory and DRs.

The effectiveness of various STs on the performance of a classic TC with $L = 800$ are shown in Fig. 6.12 and Fig. 6.13 where $D_{\max} = 30$ DRs and tail-biting encoding are assumed. The performance of T1.1 with CRC-24 is worse than those of T1.2 and T3.2 with CRC-8. Using CRC-8, T2.3 provides error rate performance similar to that of T3.2 but at the cost of one more DR. Both tests yield performance very close to that of the genie ST. In summary, these two figures show that (i) the proposed MRSTs can also be used in classic TC-coded systems and (ii) using a proper MRST has the

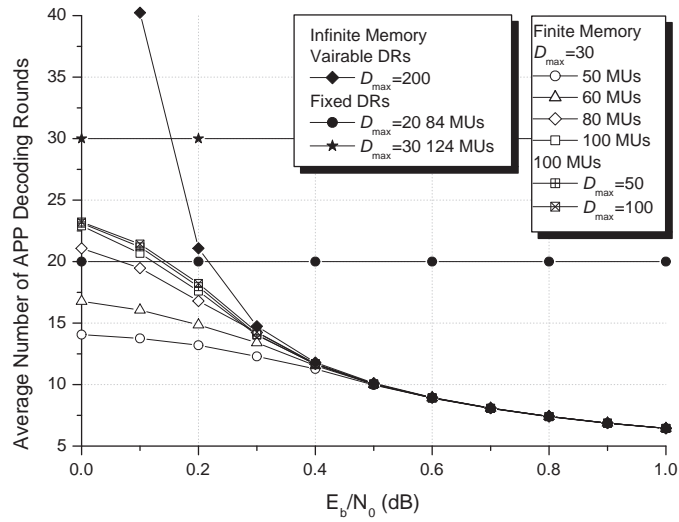


Figure 6.11: Average APP DR performance for various decoding schemes and conditions. Curves labelled with infinite memory are obtained by assuming no memory constraint; “fixed DRs” means no early-stopping condition is imposed.

benefits of reduced CRC overhead and DRs (decoding latency) without compromising the performance. The latter conclusion implies that a multiple-round stopping test with a short CRC code is better than a single-round stopping test with a much longer CRC code. Of course, the same advantages are shared by IBPTC-coded systems as well.

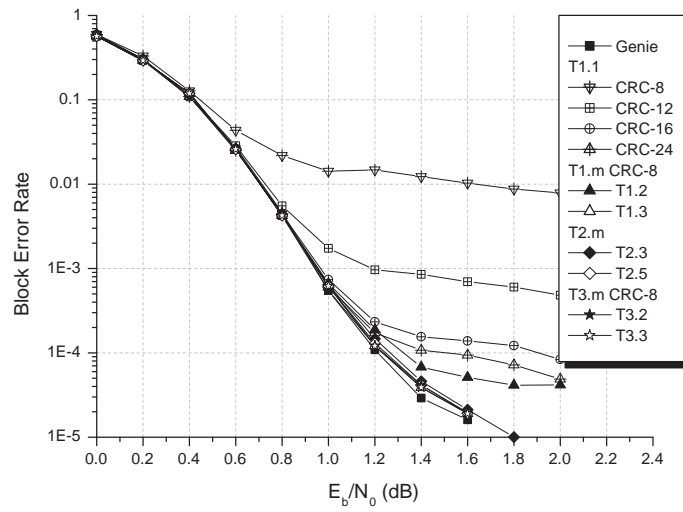


Figure 6.12: Block error rate performance of a classic TC using various STs; $L = 800$ bits and $D_{\max} = 30$ DRs.

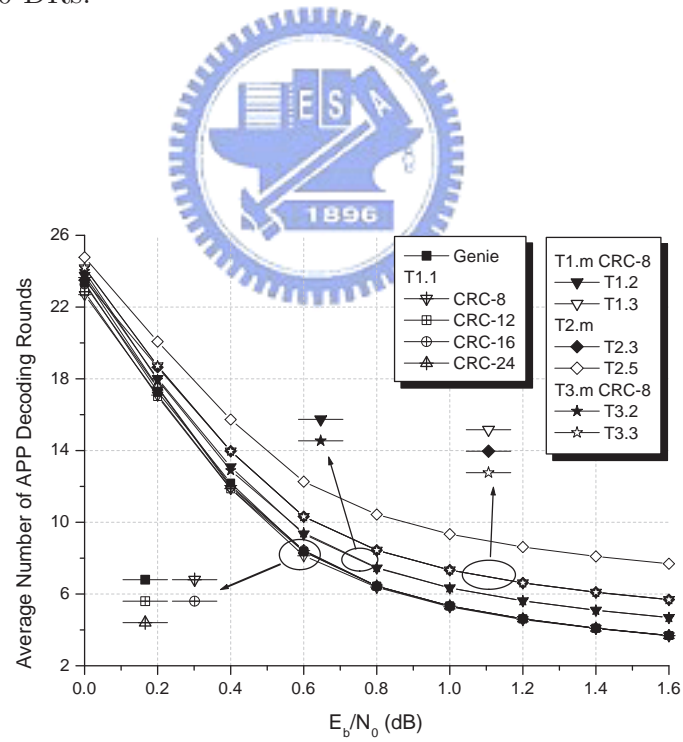


Figure 6.13: The effect of various STs on the average APP DR performance of a classic TC with $L = 800$ and $D_{\max} = 30$ DRs.

Chapter 7

Multi-stage factor graph

Multi-stage factor graph (MSFG) extended from factor graph [60, 42] elaborates message-passing for iterative decoding. Factor graph expounds a code structure and one can operate belief propagation (BP) algorithm on the graph. However, the graph only shows possible paths on the graph but does not indicate the schedule of real message-passing. The MSFG, a directed graph, describes the message-passing which reflects a decoding schedule. With the assistance of this graph the impact of the decoding schedule on computing complexity and storage requirements can be analyzed. Moreover, our representations avoid ambiguous description of cyclic or loopy message-passing events.

Multi-stage factor sub-graph (MSFSG) and causal multi-stage sub-graph (CMSSG) shorten representation of the lengthy MSFG without demonstration loss of message-passing and can be directed converted into hardware circuitry or used to design decoding schedule. The MSFG is a regular graph and looks like a duplication of a sub-graph when the decoding schedule is regular. MSFSG, a sub-graph extracted from MSFG, describes the operation procedure associated with decoding round or iteration and it is useful to represent block-oriented code such as B-IBPTC, classic TC, LDPC [46], etc. Causal multi-stage sub-graph (CMSSG), a sub-graph extracted from the MSFG, describes the operation procedure associated with each input bit or block of the stream-oriented code such as S-IBPTC, convolutional LDPC code [76, 94], etc. Therefore MSFSG and CMSSG reveal the decoding schedule which can be directly applied by the dynamic decoder shown

in Fig. 6.3 to coordinate multiple APP decoders. MSFSG and CMSSG also reflect the schedule of corresponding function nodes or hardware circuitry.

At last, we apply the CMSSG to acquire a new decoding schedule for the dynamic decoder. The new decoding schedule requires less storage space without compromising performance for S-IBPTC and the new schedule also offers performance improvement comparing to the pipeline decoder. The cost is more computing power especially at low SNR.

7.1 Multi-stage factor graph

The multi-stage factor graph (MSFG) describes message-passing for iterative decoding process. The edge between function nodes on factor graph are undirected but message-passing are different to the opposite connected function nodes. We modify the undirect edge into two opposite directed edges to reflect different message during iterative process. Then we duplicate this directed graph, redirect edges on the directed graph to connect these duplicated graph and label nodes by stage index. Then the new graph reflects real message-passing procedure during iterative process. In short, directed edges show the message-passing and stages mark the processing order on the MSFG.

The construction method is composed of following skills: grouping and labelling, duplication and stage stamping, edge replacement, edge redirecting, edge wiping and edge adding. Grouping and labelling provide a hierarchical graph representation to simplify the representation of MSFG. Duplication and stamping impose time concept on the graph. Edge replacement distinguishes messages by directions. Edge redirecting, wiping and adding connect multiple layers and reschedule message-passing. The first purpose of edge redirecting connects multiple grouped graphs. The second purpose of edge redirecting is to redirect the edge to prior stage or later stage to increase or detain message-passing. Edge wiping removes edges and the corresponding message-passing and node operations are deactivated. At last edge adding amends edges when a node

requires a message when edge redirecting removes some edges at the initial stages. These steps are described as follows.

- This step groups function nodes and edges into a grouped node and label these grouped nodes.
- This step duplicates the grouped graph into multiple layers and stamps stage on these labelled grouped nodes.
- This step replaces the undirected edges into two directed edges.
- This step redirects the directed edge to connect these grouped graphs, adds extra directed edges to enable the node processing and wipes some directed edges to detain the processing of function nodes.


We apply LDPC code [46] and S-IBPTC code as examples to draw the MSFGs corresponding to various schedules. For LDPC code, we compare conventional belief propagation (BP) algorithm [60] and horizontal-shuffled BP algorithm [63]. Furthermore we will provide another graph to demonstrate the new BP algorithm which reduces the cycle effect but requires more storage. We also plot two MSFGs for the S-IBPTC associated with Fig. 6.2. One graph is in accordance with the S-IBPTC pipeline decoding and the other is an aggressive schedule to increase message-passing speed. Both LDPC code and S-IBPTC are described in the following two subsections.

7.1.1 LDPC code

Fig. 7.1 (a) shows a $(7, 3)$ LDPC code factor graph with communication link and the associated parity check matrix is eqn. (7.1). The LDPC-encoded code bit c_i is corrupted by noise before becoming the received sample y_i . On this graph, the round node corresponds to the equality function and the cross square node corresponds to the parity check function. Conventional BP algorithm floods information based on this graph. The round node generates the extrinsic information based on the equality relation

for the cross square nodes, and the cross square node generates the extrinsic information based on the parity check relation for the round nodes. After several iterations, the estimated reliability converges and we apply the reliability on the round to acquire to the decoded codeword. However the decoded codeword may not be optimal because the cycle enhances the effect of some received samples. The following MSFGs will demonstrates the effect amongst various decoding schedules.

We construct the MSFG shown in Fig. 7.2 to demonstrate the message-passing of conventional BP algorithm. We group edges $\{y_i, c_i\}$ and function nodes into a node O_i and check node into a node \underline{Q}_i to render one grouped factor graph. Then we duplicate the graph into four graphs and remove check node part and the associated edges at the last graph. We label these nodes according to the stage from 1 to 7, where $\{O_i\}$ are labelled into $\{O_i^1, O_i^3, O_i^5, O_i^7\}$ and $\{\underline{Q}_i\}$ are labelled into $\{\underline{Q}_i^2, \underline{Q}_i^4, \underline{Q}_i^6\}$. Then we replace each edge by opposite directed edges and redirect the directed edges on even stage to nodes on the following odd stage. Then we have Fig. 7.2 showing the message-passing for conventional BP algorithm.



$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (7.1)$$

We plot another MSFG shown in Fig. 7.3 for the message-passing corresponding to horizontal-shuffled BP (HSBP) algorithm [63]. This algorithm groups check nodes and schedules the operation of these grouped check nodes. In Fig. 7.3, the nodes $\{\underline{Q}_1^i, \underline{Q}_2^i\}$ and the nodes $\{\underline{Q}_3^i, \underline{Q}_4^i\}$ are grouped separately, where the group $\{\underline{Q}_1^i, \underline{Q}_2^i\}$ and the group $\{\underline{Q}_3^i, \underline{Q}_4^i\}$ process in turn. Then the edges corresponding to $\{\underline{Q}_1^2, \underline{Q}_2^2, \underline{Q}_1^6, \underline{Q}_2^6, \underline{Q}_3^4, \underline{Q}_4^4, \underline{Q}_3^8, \underline{Q}_4^8\}$ are wiped out.

The fact that the HSBP algorithm outperforms conventional BP algorithm can be explained by both Figs. 7.2 and 7.3. In Fig. 7.2, start from the O_5^1 , there are two message-passing routes: $O_5^1 \rightarrow \underline{Q}_1^2 \rightarrow O_6^3 \rightarrow \underline{Q}_4^4 \rightarrow O_5^5$ and $O_5^1 \rightarrow \underline{Q}_4^2 \rightarrow O_6^3 \rightarrow \underline{Q}_1^4 \rightarrow O_5^5$.

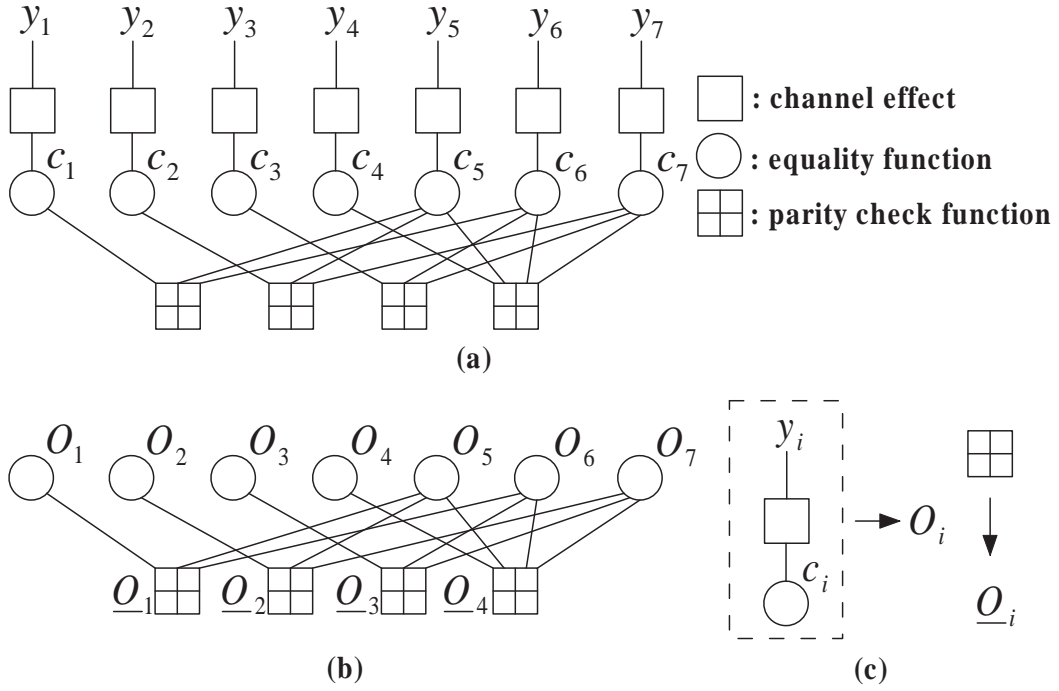


Figure 7.1: (a) Factor graph representation of an LDPC code; (b) the grouped factor graph; (c) node grouping.

Both routes merge at the fifth stage. In Fig. 7.3, the route $O_5^1 \rightarrow \underline{O}_4^2 \rightarrow O_6^3 \rightarrow \underline{O}_1^4 \rightarrow O_5^5$ is shifted to $O_5^3 \rightarrow \underline{O}_4^4 \rightarrow O_6^5 \rightarrow \underline{O}_1^6 \rightarrow O_5^7$. Therefore the message associated with the node O_5 come back to the node O_5 later and the influence of the node O_5 decreases more comparing to conventional BP algorithm. By the way the node \underline{O}_4^4 acquires information from the node O_7^3 which has been updated and therefore the node \underline{O}_4^4 can apply more information to generate the extrinsic information. This improves the convergent speed for LDPC code decoding.

Fig. 7.4 provides another schedule for LDPC code decoding to decrease the cycle effect but requires more buffer. The edge $O_2^i \rightarrow \underline{O}_2^{i+1}$ is redirected to $O_2^i \rightarrow \underline{O}_2^{i+3}$. The edge $O_4^i \rightarrow \underline{O}_4^{i+1}$ is redirected to $O_4^i \rightarrow \underline{O}_4^{i+3}$. The edges $O_6^i \rightarrow \underline{O}_1^{i+1}$, $O_6^i \rightarrow \underline{O}_3^{i+1}$ and $O_6^i \rightarrow \underline{O}_4^{i+1}$ are redirect to $O_6^i \rightarrow \underline{O}_1^{i+3}$, $O_6^i \rightarrow \underline{O}_3^{i+3}$ and $O_6^i \rightarrow \underline{O}_4^{i+3}$ respectively. We add five edges $O_2^1 \rightarrow \underline{O}_2^2$, $O_4^1 \rightarrow \underline{O}_4^2$, $O_6^1 \rightarrow \underline{O}_1^2$, $O_6^1 \rightarrow \underline{O}_3^2$ and $O_6^1 \rightarrow \underline{O}_4^2$ on this graph for the processing of the nodes $\{\underline{O}_1^2, \underline{O}_2^2, \underline{O}_3^2, \underline{O}_4^2\}$. Start from the node O_5^i , there are two

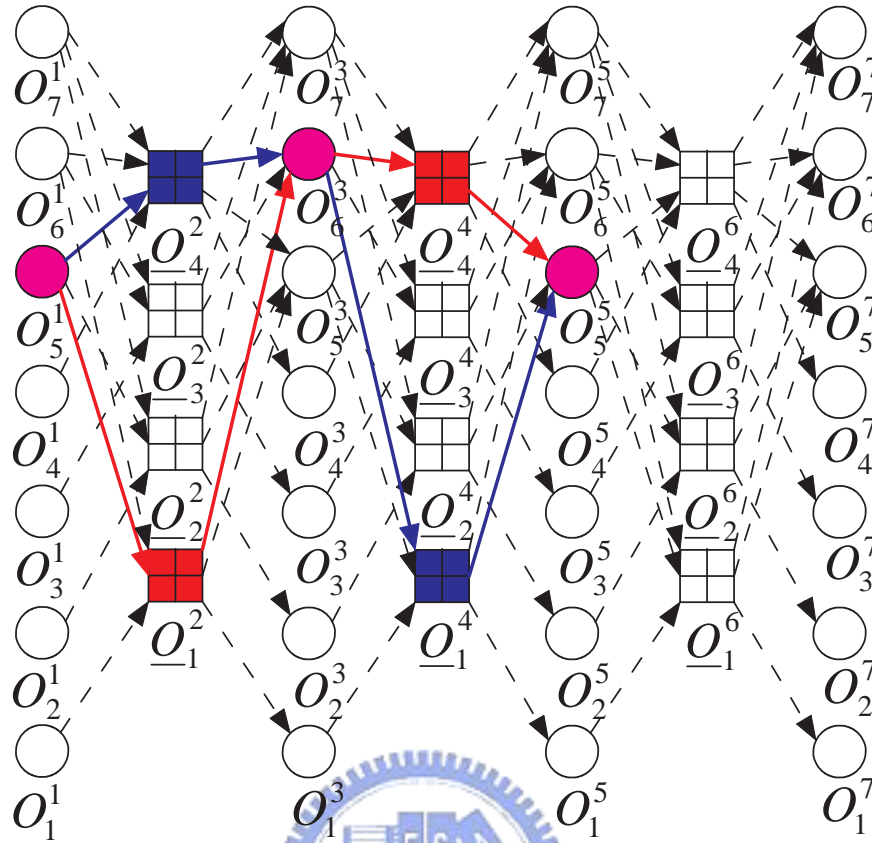


Figure 7.2: Multi-stage factor graph for conventional BP algorithm.

message-passing routes: $O_5^1 \rightarrow \underline{O}_1^2 \rightarrow O_6^3 \rightarrow \underline{O}_4^6 \rightarrow O_5^7$ and $O_5^1 \rightarrow \underline{O}_4^2 \rightarrow O_6^3 \rightarrow \underline{O}_1^6 \rightarrow O_5^7$.

These two routes merge at the seventh stage and this decreases the influence of the node O_5 . However the cost is extra buffer. Take the nodes \underline{O}_3^4 and \underline{O}_3^6 as an example, the node accesses information from the node O_6^1 and the information from the node O_6^3 has to be stored. Comparing to both conventional BP and the HSBP algorithms, the cycle effect decreases but the necessary storage increases.

Figs. 7.2-7.4 provide examples for the last MSFG construction step which influences the performance significantly. Edge wiping decreases the operation of function nodes to save computation power. Edge redirecting further detains or advances the message-passing. Edge adding amends edges on the redirected graph to enable the processing of the nodes whose edges redirected to another nodes. Next subsection depicts examples

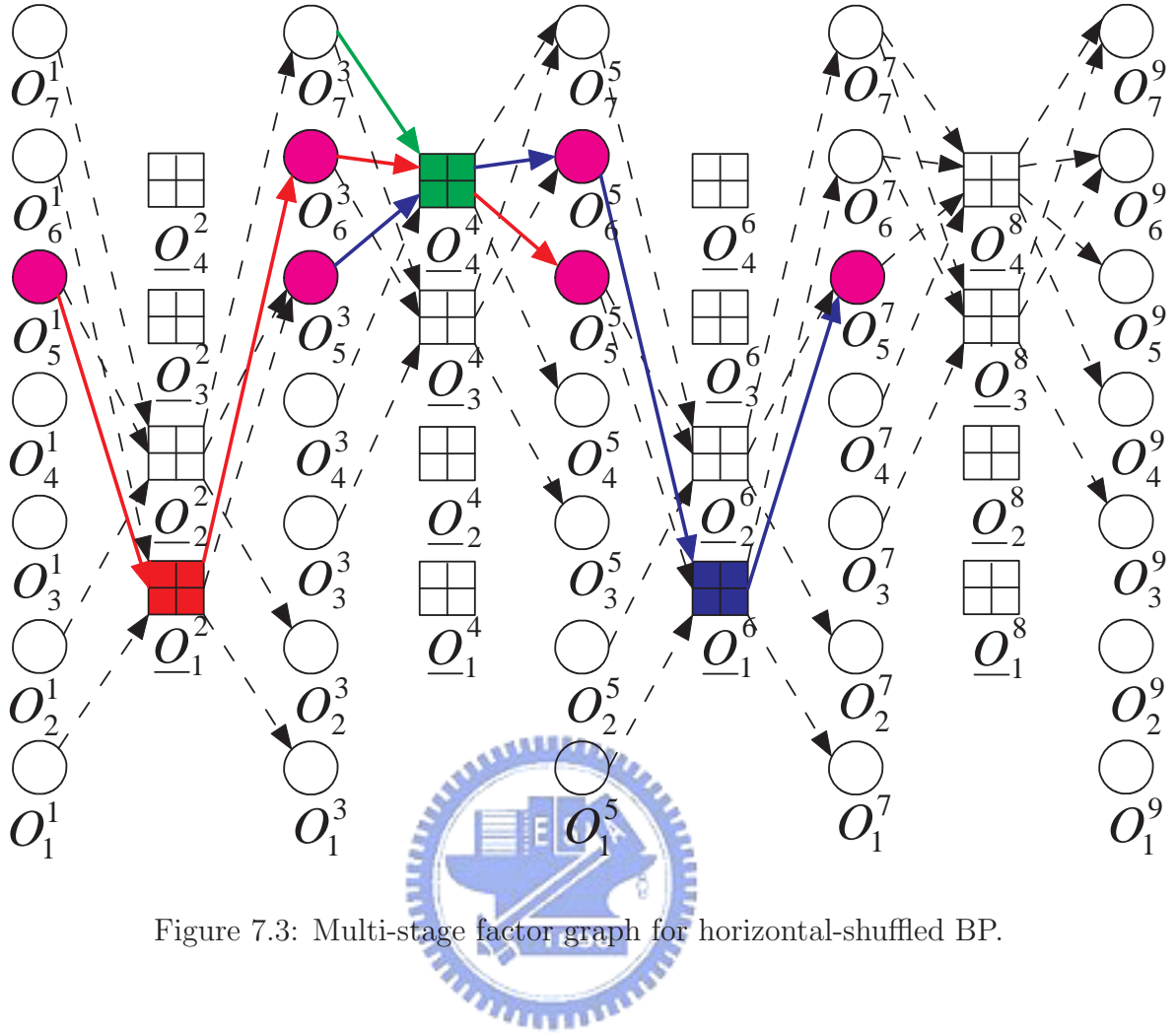


Figure 7.3: Multi-stage factor graph for horizontal-shuffled BP.

for S-IBPTC.

7.1.2 S-IBPTC

Fig. 7.5 represents a hierarchical factor graph of a coded communication link based on S-IBPTC and CRC codes. There are five data blocks and S-IBP interleaver span S is 1. \mathbf{d}_i denotes the i th input data block. \mathbf{u}_i and \mathbf{u}'_i denote the corresponding CRC encoded sequence and its permuted version. The convolutional-encoded codeword sequences $\mathbf{c}_i^0, \mathbf{c}_i^1, \mathbf{c}_i^2$ are corrupted by noise before becoming the received sequences $\mathbf{y}_i^0, \mathbf{y}_i^1, \mathbf{y}_i^2$. The decoder (node) performs APP decoding to generate the reliability for the data CRC encoded block and its permuted version and the CRC detector (node) checks the decoded block to make a stop-or-go decision for the ensuing APP decoding and outputs related

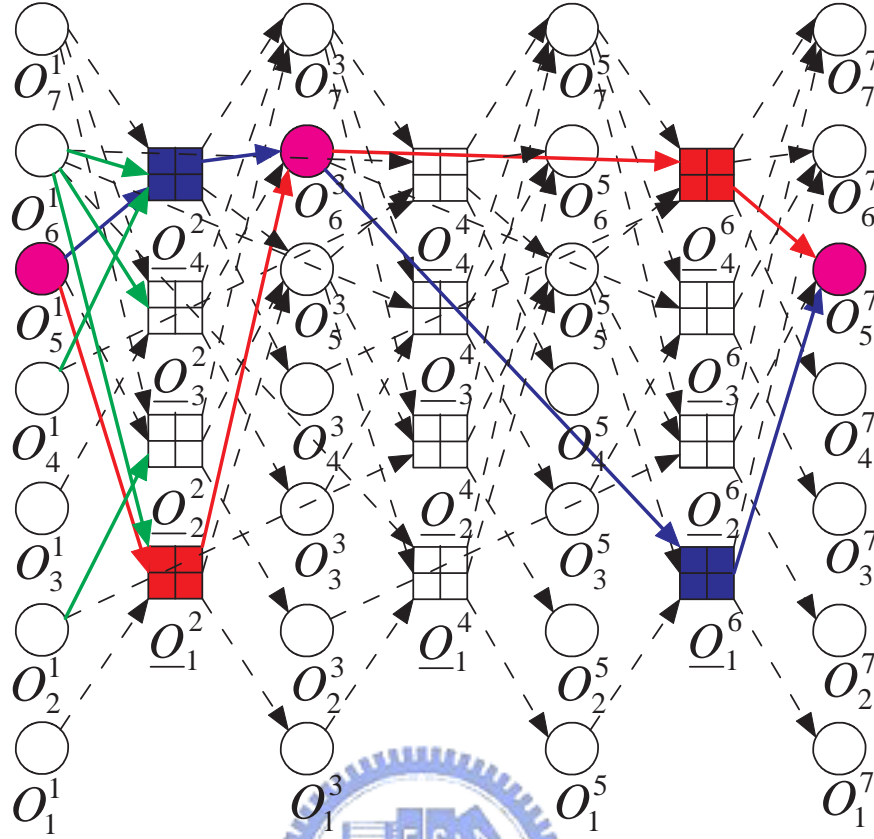


Figure 7.4: Multi-stage factor graph for the new scheduled BP which reduces cycle effect.

information. The IBP interleaver (node) sends the extrinsic information to due nodes. This graph indicates relations amongst nodes but still does not detail message-passing as previous LDPC code factor graph. We thus construct MSFG to describe the behavior of the decoding process.

The grouping and labelling step is used to simplify the following drawings and discussions. We use nodes O_i and \underline{O}_i to denote the upper branch that includes the set of nodes $\{\mathbf{d}_i, \mathbf{u}_i, \mathbf{c}_i^0, \mathbf{c}_i^1, \mathbf{y}_i^0, \mathbf{y}_i^1\}$ and the lower branch—nodes $\{\mathbf{u}'_i, \mathbf{c}_i^2, \mathbf{y}_i^2\}$ —respectively; see Fig. 7.6 (a). Fig. 7.6 (b) shows the grouped factor graph of the system shown in Fig. 7.5.

We extend undirected factor graph to a directed MSFG shown in Fig. 7.7. Nodes O_i and \underline{O}_i shown in Fig. 7.6 (b) are stamped by O_i^j and \underline{O}_i^j respectively, where j denotes

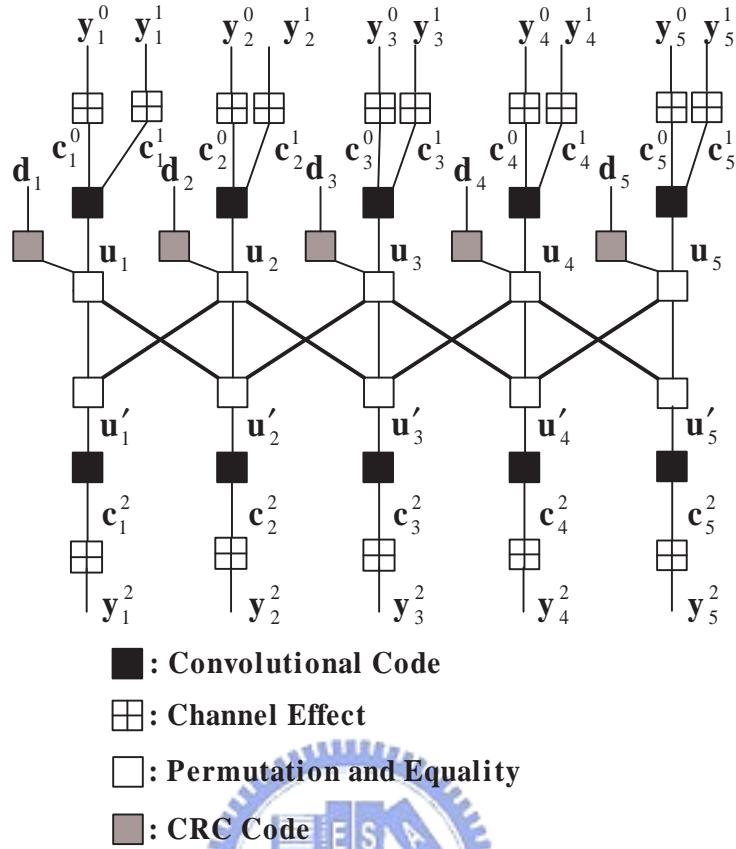


Figure 7.5: Factor graph representation of a CRC- and S-IBPTC-coded communication link.

the j th APP decoding round (ADR) corresponding to the i th block. This graph has six stages and the corresponding number of iterations is 3. The performance of Fig. 7.7 is equivalent to the pipeline decoder. We refer this schedule to pipeline schedule.

One can redirect directed edges to create a new MSFG. We redirect $\underline{O}_i^j \rightarrow O_{i+1}^{j+1}$ and $O_i^j \rightarrow \underline{O}_{i-1}^{j+1}$ to $\underline{O}_i^j \rightarrow O_{i+1}^{j-1}$ and $O_i^j \rightarrow \underline{O}_{i-1}^{j+3}$ and we a new MSFG shown in Fig. 7.8.

The redirecting enables early decoding of some selected blocks. Consider the two routes $O_5^1 \rightarrow \underline{O}_5^2 \rightarrow O_4^3 \rightarrow \underline{O}_3^4 \rightarrow O_2^5 \rightarrow \underline{O}_1^6$ and $O_3^1 \rightarrow \underline{O}_3^2 \rightarrow O_2^3 \rightarrow \underline{O}_1^6$ in Figs. 7.7 and 7.8 respectively. For the first route the node \underline{O}_1^6 is not activated until the 5th block is received while the same node can be activated when the 4th block is received. When the total encoded data length is much larger than 5 blocks, the decoding latency for node \underline{O}_1^6

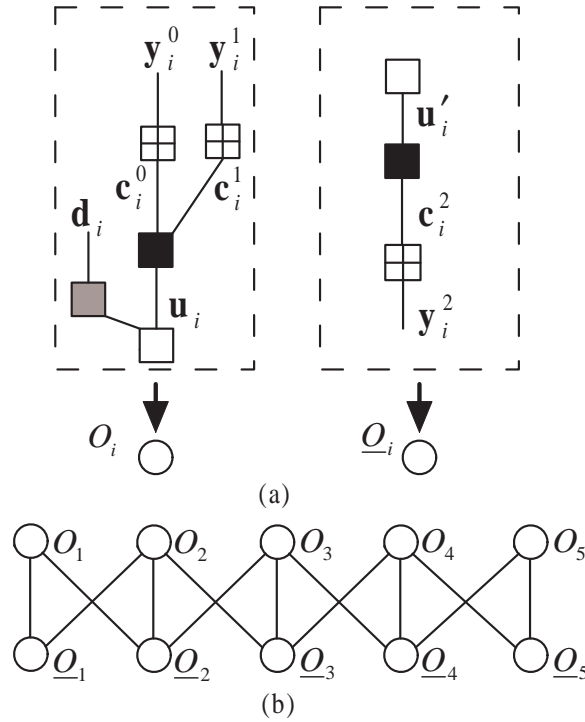


Figure 7.6: (a) Node grouping; (b) grouped factor graph.

of Fig. 7.7 becomes longer. However the message-passing range of a node in Fig. 7.8 is less than that in Fig. 7.7. The node \underline{O}_1^6 can be early activated and the necessary buffer is reduced. However this induces error rate performance loss due to less information acquired for processing as stopping mechanism and storage constraint are not applied and imposed.

7.2 Multi-stage factor sub-graph

Multi-stage factor sub-graph (MSFSG) is useful to describe message-passing related to one iteration or close several iterations, and this graph help us acquiring the corresponding hardware circuitry, decoding schedule or work balancing with the best trade-off between complexity and throughput. MSFSG is a sub-graph of MSFG without loss of message-passing description. The MSFG clearly shows message-passing but requires multiple stages. The number of stages increases with the number of iterations and this

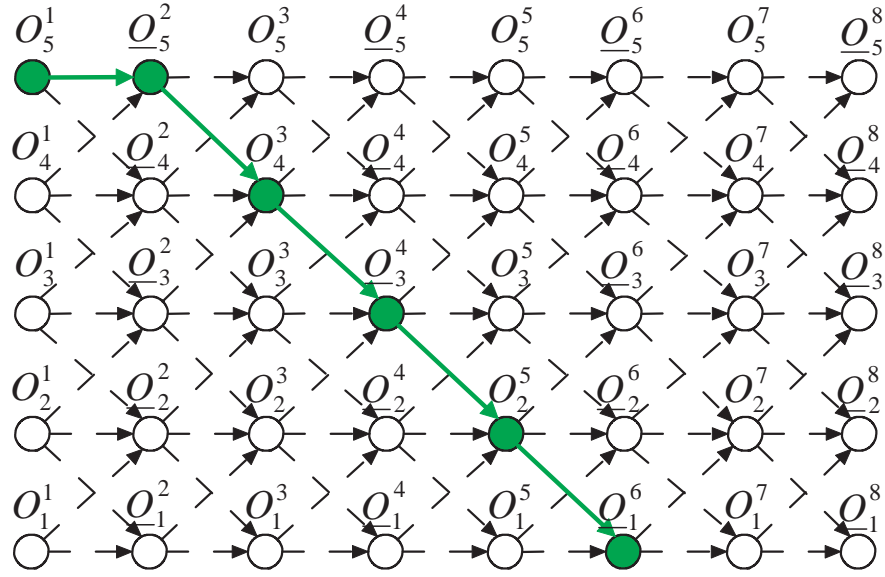


Figure 7.7: A multi-stage factor graph for the S-IBPTC pipeline decoding schedule.

enlarges the graph. Fortunately, the MSFG is regular and can be decomposed into multiple small and almost identical sub-graphs. We extract this sub-graph from the MSFG and name this sub-graph MSFSG. The function nodes equivalent to hardware circuit and this subgraph represents the operations corresponding to a time interval. One can group partial function nodes with similar computation complexity or hardware complexity and reschedule this groups to achieve the same performance. This indicates a hardware design flow for iterative decoding. The same as the previous section, we still apply the LDPC code and S-IBPTC code as examples. These exemplary graphs may be not the most compact graph but this help us to acquire drawing concepts.

7.2.1 LDPC code

Fig. 7.9 shows the MSFSGs which are sub-graphs of Figs. 7.9, 7.3 and 7.4. Conventional BP algorithm provides the most regular graph and three-stages sub-graph shown in Fig. 7.9 (a) are enough to represent Fig. 7.9.

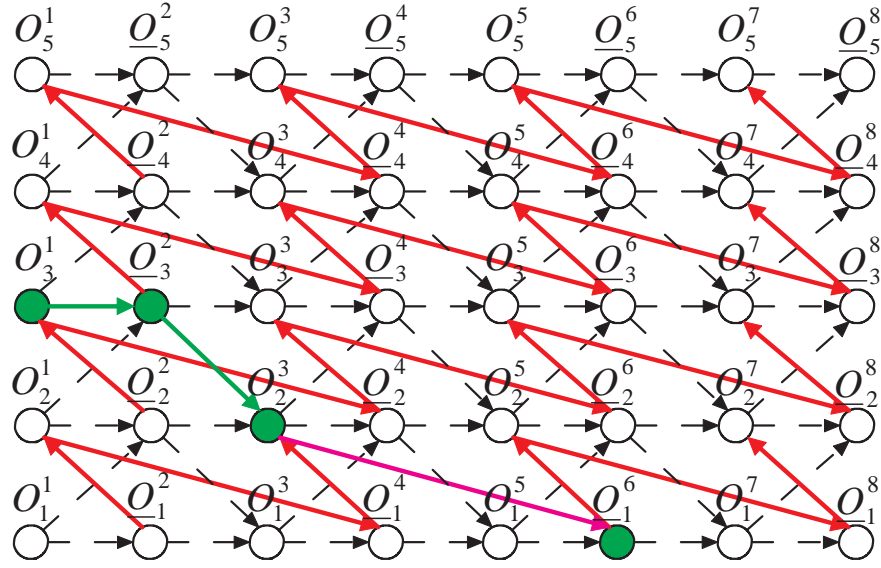


Figure 7.8: A multi-stage factor graph for the new S-IBPTC decoding schedule.

The HSBP algorithm decouples the check node processing into two groups and the necessary number of stages to represent this graph is five and the associated MSFSG is shown in Fig. 7.9 (b). Because only partial parity function nodes are processed at some stages, the nodes $\{O_3^{i+1}, O_4^{i+1}, O_1^{i+3}, O_2^{i+3}\}$ can be wiped out and the associated nodes $\{O_3^i, O_4^i, O_1^{i+4}, O_2^{i+4}\}$ can also be eliminated from this graph.

The new BP algorithm shown in Fig. 7.4 reduces the cycle effect but the graph is very irregular. Some edges go across two stages and five stages are necessary to demonstrate the graph of regular part. Fig. 7.9 (c) shows the associated MSFSG. However at initial stages the nodes $\{O_2^1, O_4^1, O_6^1\}$ pass messages to the second and fourth stages. This irregularity requires an extra initial sub-graph drawn in Fig. 7.13 (d) to complete the message-passing representation.

7.2.2 S-IBPTC

Fig. 7.11 plots the S-IBPTC MSFSG associated with Figs. 7.7 and 7.8. The pipeline schedule only passes information to the next stage and three stages are enough to represent the associated MSFG; Fig. 7.11 (a) plots the associated MSFSG. The new S-IBPTC

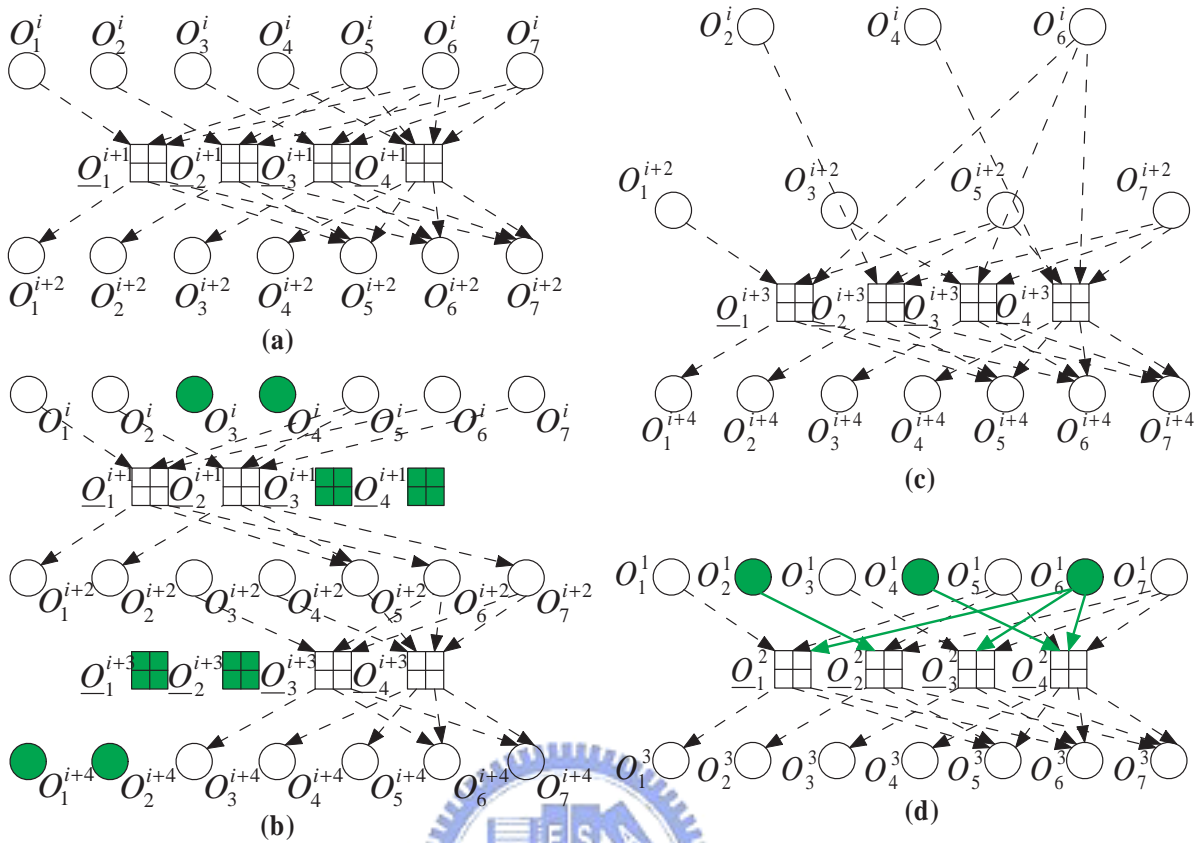


Figure 7.9: (a) The multi-stage factor sub-graph associated with Fig. 7.9; (b) the multi-stage factor sub-graph associated with Fig. 7.3; (c) the multi-stage factor sub-graph associated with Fig. 7.4; (d) the initial multi-stage factor sub-graph associated with Fig. 7.4.

schedule shown in Fig. 7.8 passes messages across three stages and therefore four stages are necessary to represent the complete graph. Fig. 7.11 (b) shows the associated MSFSG.

7.2.3 Discussion

The MSFG is generally regular. The extraction of the MSFSG depends on the number stages an edge striding across. The irregularity often occurs at the initial stage and an extra sub-graph is generally necessary.

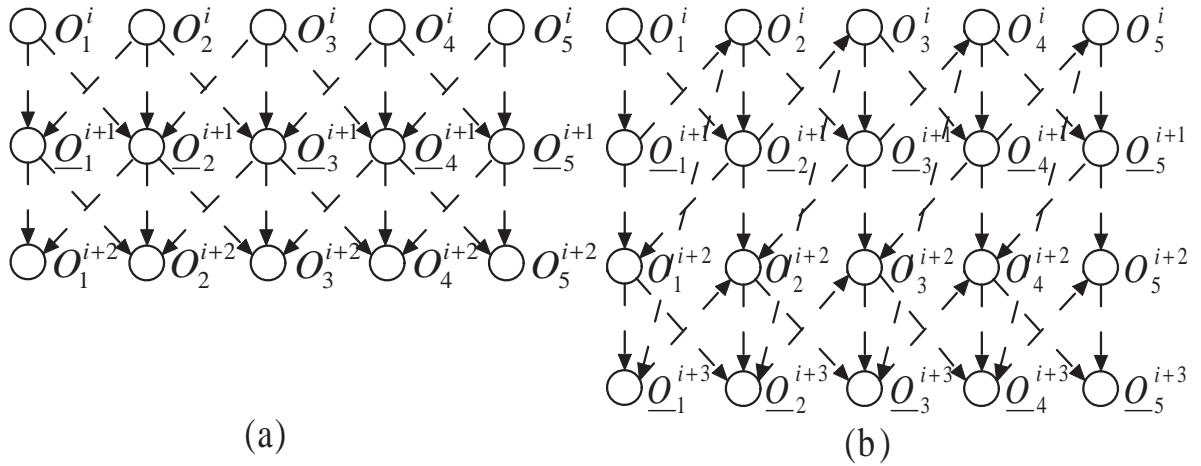


Figure 7.10: (a) The multi-stage factor sub-graph extracted from Fig. 7.7; (b) the multi-stage factor sub-graph extracted from Fig. 7.8.

7.3 Causal multi-stage sub-graph

A causal multi-stage sub-graph (CMSSG) is used to describe the message-passing for the indefinite length or stream-oriented codes such as our S-IBPTC and the LDPC convolutional code [76, 94], and a decoding schedule or the associated hardware circuit can be acquired via this graph. For an indefinite length code, a decoder generally decodes a code sequence by a sliding-window manner such as truncated Viterbi decoding algorithm and the pipeline decoding of S-IBPTC to reduce decoding complexity. Therefore we can give an order on the code sequence or information sequence by symbol or block. For one instant of a code symbol or coded block, we can divide the MSFG into three parts: the processed part, the processable part and unprocessable part corresponding to the code symbol or coded block. The processed part means function nodes which are processable if all code symbols and coded blocks prior the the code or coded block are input. The processable means function nodes which are processable when the processed part has operated and the code symbol or coded block is input. The unprocessable part corresponds to the rest of the MSFG. We can remove the processed part and unprocessable part from the MSFG and left the processable part. We modify the processable part as

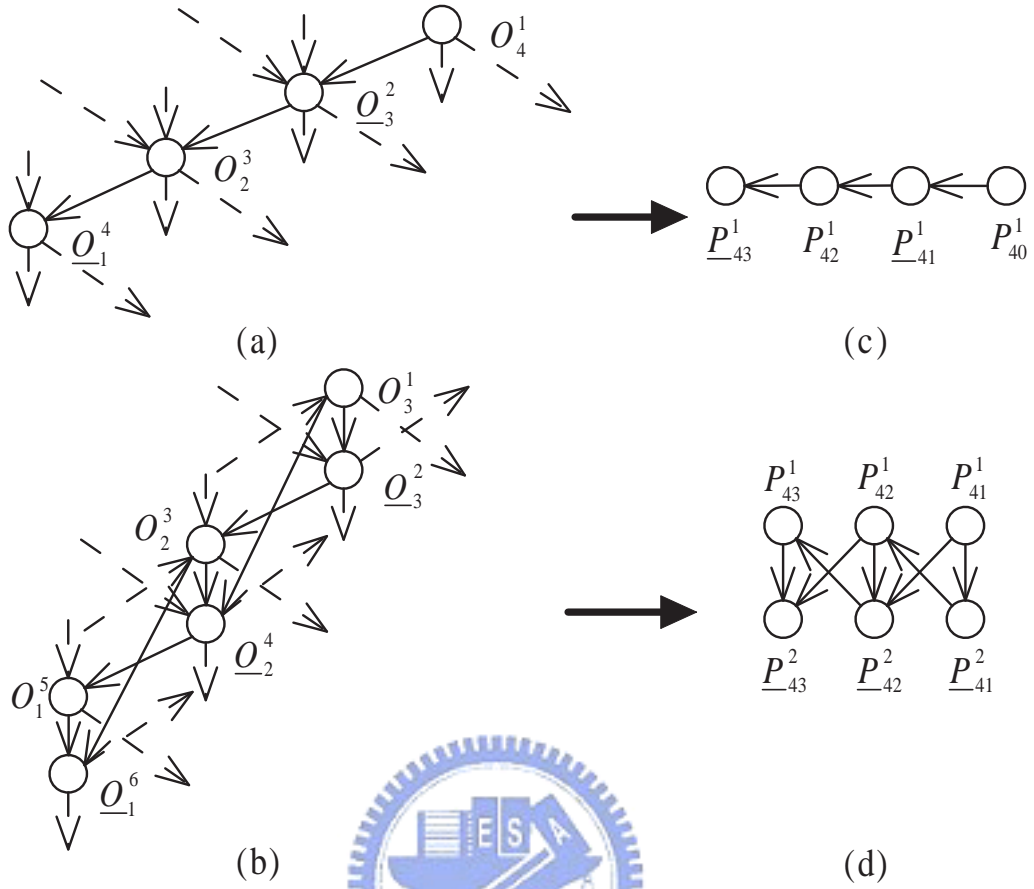


Figure 7.11: (a) The sub-graph of the MSFG shown in Fig. 7.7; (b) the sub-graph of the MSFG shown in Fig. 7.8; (c) the causal multi-stage sub-graph associated with the MSFG shown in Fig. 7.7; (d) the causal multi-stage sub-graph associated with the MSFG shown in Fig. 7.8.

the CMSSG and this graph shows the decoding schedule which can be used to design the decoding schedule, i.e. a schedule for the S-IBPTC dynamic decoder. We also can acquire the associated circuitry from the CMSSG.

Fig. 7.11 shows two examples. Assume the 4th block is input and we extract sub-graphs shown in Figs. 7.11 (a) and (b) from Figs. 7.7 and 7.8. In Fig. 7.7, the sub-graph is composed of $\{O_4^1, O_3^2, O_2^3, O_1^4\}$; real-line denotes message-passing for the instance of the 4th block input and dashed-line denotes message-passing with the other sub-graphs. We eliminate dashed-line and tilt sub-graph to render the corresponding CMSSG Fig.

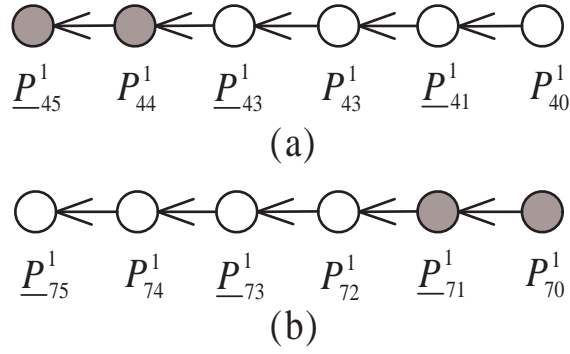


Figure 7.12: (a) Padded virtual nodes retain the regularity of the CMSSG for the beginning nodes on the MSFG; (b) padded virtual nodes retain the regularity of the CMSSG for the last nodes on the MSFG.

7.11 (c) composed of $\{P_{40}^1, \underline{P}_{41}^1, P_{42}^1, \underline{P}_{43}^1\}$. Similarly, we can have the CMSSG Fig. 7.11 (d) corresponding to Fig. 7.8.

Decoding schedule for each input instance is also clearly pointed out by these graphs. In Fig. 7.11 (c), decoder decodes align nodes $P_{40}^1 \rightarrow \underline{P}_{41}^1 \rightarrow P_{42}^1 \rightarrow \underline{P}_{43}^1$. In Fig. 7.11 (d), decoder decodes align nodes $P_{41}^1 \rightarrow \underline{P}_{41}^2 \rightarrow P_{42}^1 \rightarrow \underline{P}_{42}^2 \rightarrow P_{43}^1 \rightarrow \underline{P}_{43}^2$.

The ordinal number of the stage to the node P_{jl}^i or \underline{P}_{jl}^i for the original MSFG can be calculated as follows. Define the maximum stage of each column in the CMSSG by $N_S(l) = \arg_i \max(P_{jl}^i, \underline{P}_{jl}^i)$. The ordinal number of P_{jl}^i or \underline{P}_{jl}^i is $i + \sum_{m=1}^{l-1} N_S(m)$. P_{jl}^i and \underline{P}_{jl}^i in the CMSSG can be simply mapped to $O_{j-l}^{i+\sum_{m=1}^{l-1} N_S(m)}$ and $\underline{Q}_{j-l}^{i+\sum_{m=1}^{l-1} N_S(m)}$ in the MSFG respectively.

Virtual node concept simplifies the drawing of the CMSSG regarding to the beginning and last nodes which have no prior and successive nodes to process and we can apply one graph to represent corresponding to all code symbols or coded blocks. A virtual node is in a sense a deactivated node. Fig. 7.12 shows examples for decoding schedule shown in Fig. 7.7. Figs. 7.12 (a) and (b) plot virtual nodes $P_{44}^1, \underline{P}_{45}^1$ for the near beginning node and P_{70}^1 and \underline{P}_{71}^1 for the near ending node. Therefore P_{jl}^i is a virtual node if $(j-l) > N$ or $(j-l) < 1$, where N is total number of code symbols or coded blocks and 1 is the first ordinal. The inclusion of virtual nodes in a CMSSG enables us to describe the complete

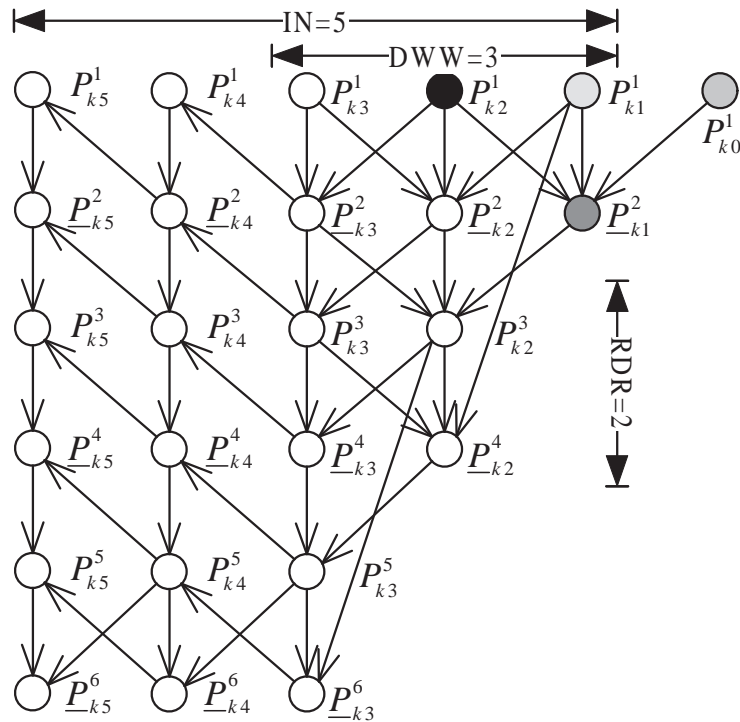
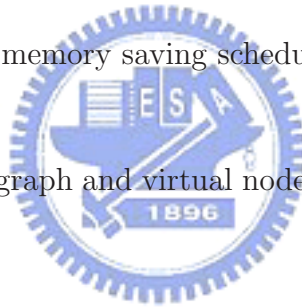


Figure 7.13: A memory saving schedule for the S-IBPTC.

decoding schedule by a single graph and virtual nodes retain the regularity of a CMSSG and simplify the drawing.



7.4 A memory-saving schedule for S-IBPTC

Fig. 6.3 has shown the dynamic decoder and the decoding schedule determines the error rate performance and memory usage. Increasing convergence rate implies that tentative information is to be kept in the memory pool shorter and the necessary memory storage decreases accordingly. Thus memory shortage is likely to occur at low SNRs as early-stopping implemented in the dynamic decoder, leading to more forced early-terminations and deteriorated performance. A candidate solution to avoid forced early-terminations is to start a new decoding round before all the information within its span becomes available. Inevitably, such an early-start decoding has to use some non-updated extrinsic information. Invoking the early-start concept, we propose the decoding

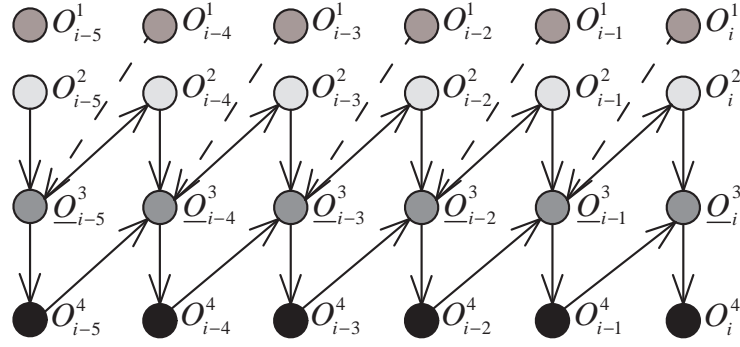


Figure 7.14: The beginning stages on the partial MSFG associated with Fig. 7.13.

schedule shown in Fig. 7.13. There are three schedule-related parameters: iteration number (IN), repeated decoding round (RDR) and decoding window width (DWW). When the maximum numbers l of P_{jl}^i of both schedule are set to 5, the pipeline schedule for each input block undergoes at most 6 ADRs but the proposed schedule undergoes at most $\frac{(IN+DWW)(IN+1)RDR}{4} + 1 = 25$ ADRs. Since the information associated with each block is used more times, the decoder converges with less information (but worse performance) and therefore requires smaller memory space.

The k th incoming block can be immediately decoded and interleaved by the pre-permutation node P_{k0}^1 whose post-permutation counterpart will not be initiated until all related blocks are received. Fig. 7.14 shows the corresponding partial MSFG with 4 stages and one can find the pre-permutation part successively processed by twice at beginning. This benefits the S-IBPTC performance but pays the computation power.

At low SNRs, the convergence speed is slow and the required memory storage is large. The proposed schedule allows more ADRs with less information for each block even if the available memory is limited. This feature provides an alternative to avoid memory shortage-induced performance loss at low SNRs.

7.5 Simulation results

Fig. 7.15 and Fig. 7.16 show the BER and average ADRs performance. The 3GPP defined interleaver and rate 1/3 turbo code [1] are used with a block length of 400 bits. Tail-biting encoding [107] and Log-MAP decoding with the T3.3 HST. One memory unit (MU) has space for 400 soft-bits. 3 MUs and 1 MU are needed for the received samples and extrinsic information per block. Denoted by Schedule A and Schedule B respectively the two schedules shown in Fig. 7.13 with and without the node P_{k0}^1 .

In Fig. 7.15, Schedule A with $IN=10$, $RDR=2$, $DWW=2$ and 45 MUs outperforms the pipeline schedule with 80 MUs, yielding a memory reduction greater than 43.75%. However, the former needs about ten and four more ADRs for $SNR = 0$ dB and $SNR > 0.4$ dB, respectively. The memory saving is obtained at the expense of higher computation complexity. It is comforting to see that the proposed schedule requires no more than ten average ADRs for $SNR > 0.8$ dB.

Increasing DWW improves the BER performance at the cost of increased computing complexity; see Fig. 7.16. For $IN=10$, $RDR=2$, Schedule A with $DWW=3$ and 45 MUs outperforms that with $DWW=2$ and 45 MUs. The same figure indicates that Schedule A with $DWW=2$ and 50 MUs outperforms that with $DWW=1$ and 50 MUs. The increase of memory also helps enhancing the BER performance. With $IN=10$, $RDR=2$ and $DWW=2$, the performance of Schedule A improves when the memory size increases from 40 MUs to 50 MUs. However, additional five more ADRs are needed at $SNR = 0.0$ dB. Finally, we note that Schedule A slightly outperforms Schedule B for 50 MUs, $IN=10$, $RDR=2$ and $DWW=2$. The required average ADRs is also slightly higher.

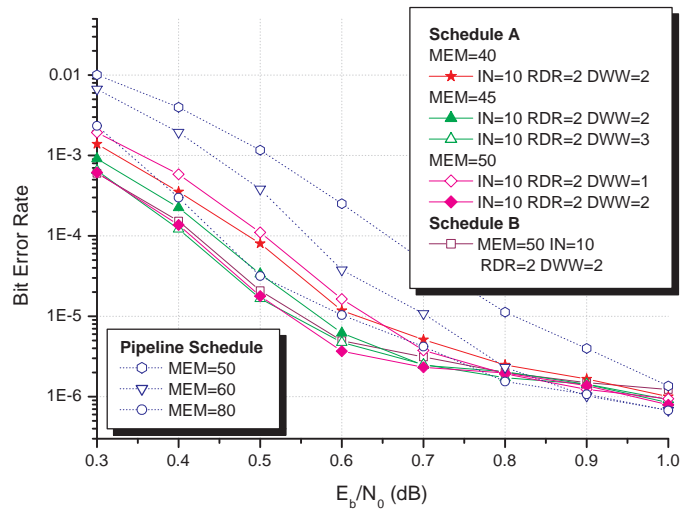


Figure 7.15: BER performance as a function of SNR for three decoding schedules.

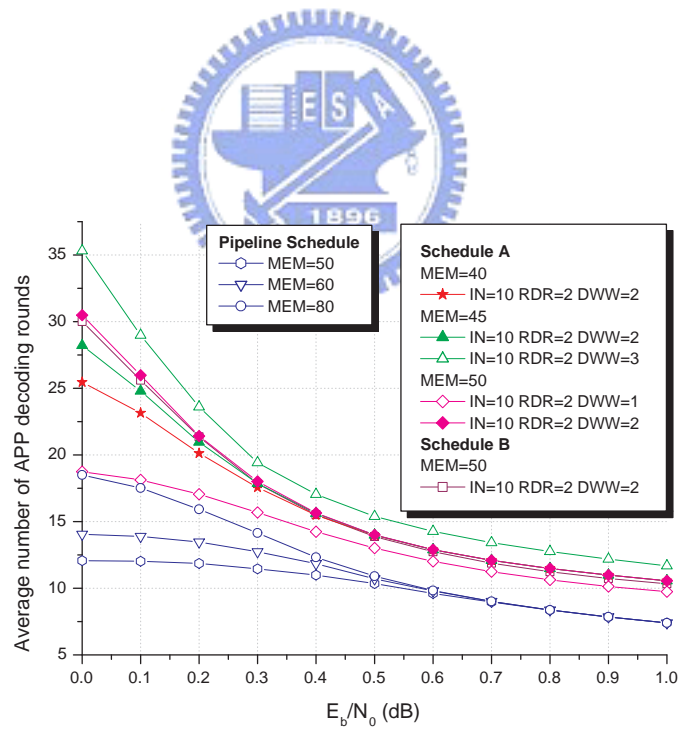


Figure 7.16: Average APP decoding round number performance as a function of SNR for three decoding schedules.

Chapter 8

Conclusions

This thesis consists of four major parts with a single focus on the interleaver design for low-latency high performance TCs. We propose a general class of interleavers and study its algebraic properties. We review hardware architecture and constraints for parallel decodable classic TCs and prove that the proposed IBP interleaver does satisfy all the hardware constraints. In particular, our design meets the memory contention-free requirement and is network-oriented, i.e., it yields low routing complexity and simple network configuration. The proposed interleaver includes most popular interleaver designs, e.g. QPP, ARP, inter-window shuffled interleaver as special cases.

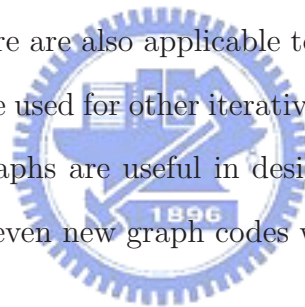
The IBP interleaver consists of two separable interleaving stages: intra-block and inter-block permutations. Our construction considers both unknown and known intra-block permutations. For unknown intra-block permutations we derive two permutation rules that guarantee desired distance properties for the resulting classic TC. The code distance upper-bounds and lower-bounds are derived for the case that one can freely choose both inter- and intra-block permutations. Continuous, tail-biting and tail-padding encodings are all considered. In order to support the high-radix APP decoder and generalized maximal contention-free requirements, we impose new constraints on the intra-block permutation and propose the associated memory mapping functions.

A decoder architecture for both stream-oriented and block-oriented IBPTCs is proposed. The decoder uses a parallel/pipelined decoding schedule which incorporates a

multiple-round early-stopping rule and a memory management scheme. The stopping rule requires short overhead but offers very reliable stopping decisions, giving improved latency and error rate performance. The memory manager makes efficient use of the memory space available and offers new trade-off between the complexity and performance.

To analyze and describe various decoding schedules for IBPTCs we generalize the static factor graph representation and develop the multi-stage factor graphs. This technique is capable of expounding the behaviors of IBPTCs and LDPC codes. Using MS-FGs, we develop a new decoding schedule for stream-oriented IBPTCs with the reduced memory storage and improved performance.

We have addressed almost all critical performance and implementation issues concerning the design of high throughput TCs and provided good, if not the best, solutions. Some algorithms proposed here are also applicable to other applications. For examples, the early-stopping rules can be used for other iterative decoding or equalization schemes, and the multi-stage factor graphs are useful in designing new LDPC or general graph code's decoding schedules or even new graph codes with enhanced performance.



Appendix A

Proof of Lemma 3.6

It is obvious that this lemma holds if $|i - j|_P \geq r$ and $P - |i - j|_P \geq r$. Hence we consider $|i - j|_P < r$ or $P - |i - j|_P < r$ only.

When $i > j$ and $0 < i - j < r$, $\gcd(P, r) \leq r$ and $r = \lceil \sqrt{P} \rceil - 1 < \sqrt{P}$ implies that $q = P/\gcd(P, r) \geq \frac{P}{r} > \sqrt{P} > r$ while $0 < i - j < r$ leads to

$$\frac{i - j + |j|_q - |i|_q}{q} = \begin{cases} \frac{i - j + |q + j - i|_q}{q} = \frac{i - j + q + (j - i)}{q} = 1, \\ \text{if } |j|_q - |i|_q > 0, \\ \frac{i - j - |i - j|_q}{q} = \frac{i - j - (i - j)}{q} = 0, \\ \text{if } |j|_q - |i|_q < 0. \end{cases} \quad (\text{A.1})$$

It follows that

$$\begin{aligned} |\pi_P(i) - \pi_P(j)|_P &= \left| \left| ri + \frac{i - |i|_q}{q} \right|_P - \left| rj + \frac{j - |j|_q}{q} \right|_P \right|_P \\ &= \left| r(i - j) + \frac{i - j + |j|_q - |i|_q}{q} \right|_P \geq r \end{aligned} \quad (\text{A.2})$$

and

$$\begin{aligned} P - |\pi_P(i) - \pi_P(j)|_P &= P - \left| \left| ri + \frac{i - |i|_q}{q} \right|_P - \left| rj + \frac{j - |j|_q}{q} \right|_P \right|_P \\ &\geq P - |r(r - 1) + 1|_P = P - r^2 + r - 1 \\ &\geq r^2 + 1 - r^2 + r - 1 = r. \end{aligned} \quad (\text{A.3})$$

Therefore, $\min_{i, j \in S_P} (i - j + |\pi_P(i) - \pi_P(j)|_P, i - j + P - |\pi_P(i) - \pi_P(j)|_P) \geq r + 1$.

This permutation function is q -invariant in that

$$\begin{aligned}
& |\pi_P(|i - q|_P) - \pi_P(|j - q|_P)|_P \\
&= \left| \left| r(i - q) + \frac{(i - q) - |i - q|_q}{q} \right|_P - \left| r(j - q) + \frac{(j - q) - |j - q|_q}{q} \right|_P \right|_P \\
&= \left| \left| ri + \frac{i - |i|_q}{q} \right|_P - \left| rj + \frac{j - |j|_q}{q} \right|_P \right|_P = |\pi_P(i) - \pi_P(j)|_P \tag{A.4}
\end{aligned}$$

We now show that both the remaining cases can be converted into the above case.

(A) For the case $i < j$ and $0 < j - i < r$, we have $|i - j|_P = |i + P - j|_P = |i + P - mq - (j - mq)|_P = |i' - j'|$ and $|\pi_P(|i + P - mq|_P) - \pi_P(|j - mq|_P)|_P = |\pi_P(i) - \pi_P(j)|_P$, $P > i' = |i + P - mq|_P > j' = |j - mq|_P \geq 0$ for some $m > 0$.

(B) If $i > j$, $P - |i - j|_P = |P + j - i|_P = |P + j - mq - (i - mq)|_P = |j' - i'|$ and $|\pi_P(|i - mq|_P) - \pi_P(|P + j - mq|_P)|_P = |\pi_P(i) - \pi_P(j)|_P$, $P > i' = |i + P - mq|_P > j' = |j - mq|_P \geq 0$ for some $m > 0$.



Appendix B

Proof of Lemma 3.7

We place the elements in the j th n -element set in a cycle by $j + i(N_1 + N_2 - \lfloor \frac{N_2}{n} \rfloor)$, where $0 \leq i < n$ and $0 \leq j < N_1$. The elements of the j th $(n-1)$ -element set are placed at positions indexed by

$$\begin{cases} \left\lfloor \frac{iN_2 + j - N_1}{M_1} \right\rfloor (N_1 + M_1) + N_1 + |iN_2 - N_1 + j|_{M_1}, & iN_2 + j \leq M_3, \\ N_1 \lfloor N_2 \rfloor_n + M_3 + \left\lfloor \frac{iN_2 + j - N_1 - M_3}{M_2} \right\rfloor (N_1 + M_2) + N_1 \\ \quad + |iN_2 + j - N_1 - M_3|_{M_2}, & \text{otherwise,} \end{cases} \quad (\text{B.1})$$

where $0 \leq i < n-1$, $N_1 \leq j < N_1 + N_2$, $M_1 = N_2 - \lfloor \frac{N_2}{n} \rfloor$, $M_2 = N_2 - \lceil \frac{N_2}{n} \rceil$ and $M_3 = M_1 \lfloor N_2 \rfloor_n$. It is easy to see that such an arrangement achieve the bounds and no larger minimum separation can be found.

Appendix C

Proof of Theorem 3.3

Tail-biting encoding results in low-weight codewords whose nonzero coordinates are confined to the tail and the head parts of two consecutive sets. This happens if one nonzero coordinate of a weight-2 input sequence belongs to $F_i^{(k)}$ and the other one belongs to $F_{|i+T_c-L|_{T_c}|_{T_c}}^{(k)}$. One can then place the set $F_{|i+T_c-L|_{T_c}|_{T_c}}^{(k)}$ right after the set $F_i^{(k)}$ so that they form a cycle. If $\gcd(|L|_{T_c}, T_c) = d$, we have d cycles with the m th cycle being $\tilde{F}_m^{(k)} = \left\{ F_m^{(k)}, F_{|m+T_c-L|_{T_c}|_{T_c}}^{(k)}, F_{|m+2(T_c-L)|_{T_c}|_{T_c}}^{(k)}, \dots, F_{|m+(\frac{T_c}{d}-1)(T_c-L)|_{T_c}|_{T_c}}^{(k)} \right\}$, where $0 \leq m < d$.

Mapping the coordinates in $\tilde{F}_m^{(k)}$ sequentially to the integers in the interval $[0, |\tilde{F}_m^{(k)}| - 1 = \frac{L}{d} - 1]$, we obtain the set $S_{|\tilde{F}_m^{(k)}|} = \{0, 1, 2, \dots, \frac{L}{d} - 1\}$. We further partition $S_{|\tilde{F}_m^{(k)}|}$ into dT_s sets $\{S_i\}$, where $|S_i| = N_{max} = \left\lceil \frac{L}{d^2 T_s} \right\rceil$ for $0 \leq i < N_1 = \left\lfloor \frac{L}{d} \right\rfloor^{dT_s}$ and $|S_i| = N_{min} = \left\lfloor \frac{L}{d^2 T_s} \right\rfloor$ for $dT_s - N_2 = \left\lfloor \frac{L}{d} \right\rfloor^{dT_s} \leq i < dT_s$. According to *Lemma 3.7*, we can maximize the minimum separation of S_i to $D_{min} = dT_s - \left\lceil \frac{N_2}{N_{max}} \right\rceil$ and $D_{max} = dT_s - \left\lfloor \frac{N_2}{N_{max}} \right\rfloor$ for $0 \leq i < N_1$ and $dT_s - N_2 \leq i < dT_s$ respectively.

We can construct an IBP rule such that $p \in S_i$ and $q \in S_j$ are permuted to the same block iff $|i - j|_{T_s} = 0$. Since all blocks can apply the same partition rule for permutation, such an IBP rule does exist.

Incorporating separate encoding results in that two indexes in two different blocks produce a codeword weight larger than the bound, either the pre-permuted or the post-permuted pair makes the codeword weight $2W_1(L)$. Therefore we consider the case two

indexes are permuted to the same block.

There are d sets S_i and d sets $S_{|\tilde{F}_m^{(2)}|}$. All $S_i \subset S_{|\tilde{F}_m^{(1)}|}$ can be permuted to different $S_{|\tilde{F}_m^{(2)}|}$. If two indexes are in two different S_i 's, either the pre-permuted or the post-permuted pair makes the codeword weight $\geq W_2(L)$, which is larger than the bound. Therefore we only have to consider the case when a coordinate pair belongs to the same S_i before and after permutation.

According to *Lemma 3.6*, the separation sum of pre-permutation and post-permutation for S_i with N_{max} and N_{min} elements can be $\lceil \sqrt{N_{max}} \rceil$ and $\lceil \sqrt{N_{min}} \rceil$ respectively. According to *Lemma 3.7*, the minimum separation of two adjacent indexes is D_{min} and there are at most $|N_2|_{N_{max}}$ pairs with such a separation. The minimum codeword weight is thus lower-bounded by $2 + \alpha D_{min} \min(2|N_2|_{N_{max}}, \lceil \sqrt{N_{max}} \rceil) + \alpha D_{max} \max(\lceil \sqrt{N_{max}} \rceil - 2|N_2|_{N_{max}}, 0) + 2\beta$.

Finally, we notice that small weight error event occurs when the two coordinate pair $(i, j) \in \tilde{F}_m$ is such that $|i - j|_{T_c} \neq 0$ and $|L - |i - j||_{T_c} \neq 0$ and the separation of the permuted pair $(\pi_{|\tilde{F}_m|}(i), \pi_{|\tilde{F}_m|}(j))$ is greater than $T_c D_{min}$. The corresponding codeword weight will be at least $2 + W_2(L) + \alpha D_{min} + \beta$. Therefore, we have

$$\begin{aligned}
 w_t(\mathbf{C}^{ij}) &\geq 2 + 2\beta + \min_{i,j} (W_2(L) + \alpha D_{min} - \beta, \\
 &\quad \alpha D_{min} \min(2|N_2|_{N_{max}}, \lceil \sqrt{N_{max}} \rceil) \\
 &\quad + \alpha D_{max} \max(\lceil \sqrt{N_{max}} \rceil - 2|N_2|_{N_{max}}, 0)). \tag{C.1}
 \end{aligned}$$

Appendix D

Proof of Theorem 5.10

We first notice that, besides those finite weight codewords resulting from termination, as illustrated in Fig. 5.6, there are three conditions under which a weight-4 input sequence of an S-IBPTC will generate a finite-weight codeword. In Case (a), the codeword consists of two finite-weight segments (in different blocks) generated respectively by two weight-2 input sequences and thus the corresponding codeword weight upper-bound is simply twice that given in *Theorem 5.9*. Case (b) considers the situation when two pairs of coordinates from $F_i^{(1)}$ and $F_j^{(1)}$ of either the same block or different blocks are permuted to the same block with one coordinate from each pair mapped to two subsets $F_k^{(2)}$ and $F_l^{(2)}$, where the pair $(k, l), k \neq l$ belongs to the same equivalence class while the remaining two coordinates mapped to another two subsets $F_m^{(2)}$ and $F_n^{(2)}$ with $m \neq n$ in another equivalence class. Case (c) is similar to Case (b) except that the two subsets that contain the two permuted pairs are in different blocks.

Note that if $k = l$ and $m = n$ then the both cases will result in a codeword weight upper-bound similar to that obtained in [25]. But this is impossible as coordinates from different blocks will not be mapped into coordinates in the same subset (defined by eqn. (5.15)) by an optimal interleaver. This is because the spatial symmetric structure of a classic TC implies that, for every input sequence \mathbf{u} of the code \mathbf{C} that uses the interleaver π , $\exists \mathbf{u}'$ such that the codewords generated by (\mathbf{u}, π) and (\mathbf{u}', π^{-1}) have identical weight. This observation and the fact that both component encoder outputs, \mathbf{c}_1 and \mathbf{c}_2 ,

contribute equally to the resulting codeword weight suggest that π and π^{-1} have the same effect on the weight distribution, and that optimizing the deinterleaver rule results in the same mapping as the optimal interleaver.

Since we have to consider the scenario $k \neq l$ and $m \neq n$ only, the worst case occurs when both $|k - l|$ and $|m - n|$ are less than $T_s T_c$. In other words, Cases (b) and (c) concern the situation in which the pairs $(\pi_{ibp}(x), \pi_{ibp}(w))$ and $(\pi_{ibp}(y), \pi_{ibp}(z))$ belong to distinct supersubsets where a supersubset $\tilde{F}_j^{(2)}$ consists of $M/T_s = T_c$ consecutive subsets of the same equivalence class. Each block therefore has $\frac{\Lambda}{T_s}$ supersubsets, and $F_k^{(2)}$ and $F_l^{(2)}$ are in the same supersubset $\tilde{F}_j^{(2)}$ if $|k|_M = |l|_M = j$ and $\|k - l\|_{T_s} = 0$, or equivalently, $\tilde{F}_p^{(i)} = \bigcup_{k=0}^{T_s-1} F_{\|j\|_{T_c} + kT_c + |j|_{T_c} M}^{(i)}$, $i = 1, 2$.

Let Λ_1 and Λ_2 be the number of coordinates subsets per block for the input and permuted sequences. The subset partition rule, eqn. (5.15), implies that $\Omega \leq |F_i^{(j)}| \leq \Omega + 1$, where $\Omega = \lfloor \frac{\Lambda}{\Lambda_i} \rfloor$. For Case (b), each subset has either $\binom{\Omega+1}{2}$ or $\binom{\Omega}{2}$ distinct coordinates pairs and each block has at least $\Lambda_i \binom{\Omega}{2}$ such pairs. Our S-IBP interleaver maps $\frac{\Lambda_1}{T_s}$ sets of coordinates to each block within its span, or equivalently, a block “receives” coordinates from T_s neighboring blocks. The optimal S-IBP rule would map a pair of coordinates in the same subset to different equivalence classes or blocks and, when this is not possible, to different supersubsets of the same block.

A pair of coordinates (i, j) in $F_i^{(1)}$ can be mapped to any one of the $\binom{\Lambda_2}{2}$ pairs of distinct supersubsets $\tilde{F}_j^{(2)}, \tilde{F}_k^{(2)}, j \neq k$ of a neighboring block. A periodic S-IBP requires that at least $T_s \frac{\Lambda_1}{T_s} \binom{\Omega}{2}$ distinct pairs of coordinates from T_s neighboring blocks be permuted to the same block. The pigeonholes principle implies that Case (b) will occur if

$$\Lambda_1 \binom{\Omega}{2} > \binom{\frac{\Lambda_2}{T_s}}{2}. \quad (\text{D.1})$$

For Case (c) the pairs $(\pi_{ibp}(x), \pi_{ibp}(w))$ and $(\pi_{ibp}(y), \pi_{ibp}(z))$ are in two distinct blocks. If the two distinct blocks are separated by k blocks ($k = 1$ means they are two successive blocks), then $(\pi_{ibp}(x), \pi_{ibp}(w)), (\pi_{ibp}(y), \pi_{ibp}(z))$ are mapped from $T_s - k$ neighboring

blocks in which each block contains $\frac{\Lambda_1}{T_s}$ supersubsets and each supersubset has at most $(\Omega + 1)^2$ and at least Ω^2 coordinates pairs to the two designated blocks. Therefore, finite weight codewords result if

$$(T_s - k) \frac{\Lambda_1}{T_s} \Omega^2 > \left(\frac{\Lambda_2}{T_s} \right)^2 \quad (\text{D.2})$$

and we obtain upper-bounded

$$w_{4,min,ibp} \leq 4 + 2\alpha \cdot \left(\min_{(\Lambda_1, \Lambda_2)} \left\{ \left\lceil \frac{T_s \cdot L}{\Lambda_1} \right\rceil + \left\lceil \frac{T_s \cdot L}{\Lambda_2} \right\rceil \right\} - 2 \right) + 4\beta \quad (\text{D.3})$$

where (Λ_1, Λ_2) are subject to the constraints, (C1): $\|\Lambda_1\|_M = \|\Lambda_2\|_M = 0$, (C2): $\Lambda_1 \binom{\Omega}{2} > \left(\frac{\Lambda_2}{2}\right)$, and (C3): $\frac{(T_s - k)}{T_s} \Lambda_1 \Omega^2 > \left(\frac{\Lambda_2}{T_s}\right)^2$. Since $\Omega = \lfloor \frac{L}{\Lambda_1} \rfloor > \frac{L}{\Lambda_1} - 1$, we rewrite (D.1) and (D.2) as

$$\Lambda_1 \left(\frac{L}{\Lambda_1} - 1 \right) \left(\frac{L}{\Lambda_1} - 2 \right) \geq \left(\frac{\Lambda_2}{T_s} - 1 \right) \frac{\Lambda_2}{T_s} \quad (\text{D.4})$$

$$\frac{T_s - k}{T_s} \Lambda_1 \left(\frac{L}{\Lambda_1} - 1 \right)^2 \geq \left(\frac{\Lambda_2}{T_s} \right)^2 \quad (\text{D.5})$$

We carry out the minimization with respect to (Λ_1, Λ_2) by first finding the two minimums with respect to the constraints (C1)/(C2) and (C1)/(C3), respectively, and then select the smaller one of these two. Using the simplified assumption [25] that the cardinalities of Λ_1 and Λ_2 are the same and to distinguish the two candidate minimums, we set $\Lambda_1 = \Lambda_2 = \Lambda_3$ in (D.4) and $\Lambda_1 = \Lambda_2 = \Lambda_4$ in (D.5) so that the above two inequalities become

$$\Lambda_3^3 - (T_s + 2T_s^2)\Lambda_3^2 + 3T_s^2\Lambda_3L - T_s^2L^2 \leq 0 \quad (\text{D.6})$$

$$\Lambda_4^3 - T_s(T_s - k)\Lambda_4^2 + 2T_s(T_s - k)\Lambda_4L - T_s(T_s - k)L^2 \leq 0 \quad (\text{D.7})$$

By defining $X_1 = \Lambda_3 - \frac{T_s - 2T_s^2}{3}$ and $X_2 = \Lambda_4 - \frac{T_s(T_s - k)}{3}$, we rewrite the above inequalities as

$$X_1^3 + \left(3T_s^2L - \frac{1}{3}(T_s + 2T_s^2)^2 \right) X_1 + \left(-T_s^2L^2 + (T_s^3 + 2T_s^4)L - \frac{2}{27}(T_s + 2T_s^2)^3 \right) \leq 0 \quad (\text{D.8})$$

$$\begin{aligned}
& X_2^3 + \left(2T_s(T_s - k)L - \frac{1}{3}T_s^2(T_s - k)^2 \right) X_2 \\
& + \left(-T_s(T_s - k)L^2 + \frac{2}{3}T_s^2(T_s - k)^2L - \frac{2}{27}T_s^3(T_s - k)^3 \right) \leq 0 \quad (\text{D.9})
\end{aligned}$$

Following the standard procedure for solving a cubic equation [48], we define

$$\begin{aligned}
p_1 &= 3T_s^2L - \frac{1}{3}(T_s + 2T_s^2)^2, \\
q_1 &= -T_s^2L^2 + (T_s^3 + 2T_s^4)L - \frac{2}{27}(T_s + 2T_s^2)^3, \\
p_2 &= 2T_s(T_s - k)L - \frac{1}{3}T_s^2(T_s - k)^2, \\
q_2 &= -T_s(T_s - k)L^2 + \frac{2}{3}T_s^2(T_s - k)^2L - \frac{2}{27}T_s^3(T_s - k)^3.
\end{aligned}$$

If $L > \frac{10}{3}T_s^3 + T_s^2 - \frac{T_s}{3}$, then

$$\begin{aligned}
p_1 &= 3T_s^2L - \frac{1}{3}(T_s + 2T_s^2)^2 = T_s^2 \left(3L - \frac{4}{3}T_s^2 - \frac{4}{3}T_s - \frac{1}{3} \right) \\
&> T_s^2 \left(10T_s^3 + 3T_s^2 - 1 - \frac{4}{3}T_s^2 - \frac{4}{3}T_s - \frac{1}{3} \right) > 0 \quad (\text{D.10})
\end{aligned}$$

$$\begin{aligned}
p_2 &= 2T_s(T_s - k)L - \frac{1}{3}T_s^2(T_s - k)^2 = 2T_s(T_s - k) \left(L - \frac{1}{6}T_s(T_s - k) \right) \\
&> 2T_s(T_s - k) \left(L - \frac{1}{6}T_s(T_s - 1) \right) > 0 \quad (\text{D.11})
\end{aligned}$$

$$\begin{aligned}
q_1 &= -T_s^2L^2 + (T_s^3 + 2T_s^4)L - \frac{2}{27}(T_s + 2T_s^2)^3 < -T_s^2L^2 + (T_s^3 + 2T_s^4)L \\
&= -T_s^2L(L - T_s - 2T_s^2) \\
&< -T_s^2L \left(\frac{10}{3}T_s^3 + T_s^2 - \frac{T_s}{3} - T_s - 2T_s^2 \right) < 0 \quad (\text{D.12})
\end{aligned}$$

$$\begin{aligned}
q_2 &= -T_s(T_s - k)L^2 + \frac{2}{3}T_s^2(T_s - k)^2L - \frac{2}{27}T_s^3(T_s - k)^3 \\
&< -T_s(T_s - k)L^2 + \frac{2}{3}T_s^2(T_s - k)^2L \\
&< -T_s(T_s - k)L \left(L - \frac{2}{3}T_s^2 \right) \\
&< -T_s(T_s - k)L \left(\frac{10}{3}T_s^3 + T_s^2 - \frac{T_s}{3} - \frac{2}{3}T_s^2 \right) < 0 \quad (\text{D.13})
\end{aligned}$$

$$\begin{aligned}
p_1 - p_2 &= 3T_s^2L - \frac{1}{3}(T_s + 2T_s^2)^2 - 2T_s(T_s - k)L + \frac{1}{3}T_s^2(T_s - k)^2 \\
&> 3T_s^2L - \frac{1}{3}(T_s + 2T_s^2)^2 - 2T_s^2L + \frac{1}{3}T_s^2 = T_s^2L - \frac{4}{3}T_s^2 - \frac{4}{3}T_s \\
&> T_s^2 \left(\frac{10}{3}T_s^3 + T_s^2 - \frac{T_s}{3} \right) - \frac{4}{3}T_s^2 - \frac{4}{3}T_s > 0 \quad (\text{D.14})
\end{aligned}$$

$$\begin{aligned}
& q_2 - q_1 \\
&= -T_s(T_s - k)L^2 + \frac{2}{3}T_s^2(T_s - k)^2L - \frac{2}{27}T_s^3(T_s - k)^3 \\
&\quad + T_s^2L^2 - (T_s^3 + 2T_s^4)L + \frac{2}{27}(T_s + 2T_s^2)^3 \\
&> kT_sL^2 + \frac{2}{3}T_s^2(T_s - k)^2L - (T_s^3 + 2T_s^4)L > LT_s \left(L + \frac{2}{3}T_s^2 - (T_s^2 + 2T_s^3) \right) \\
&> LT_s \left(\frac{10}{3}T_s^3 + T_s^2 - \frac{T_s}{3} - (T_s^2 + 2T_s^3) \right) > 0. \tag{D.15}
\end{aligned}$$

These results imply $(\frac{p_1}{3})^3 + (\frac{q_1}{2})^2 > 0$, $(\frac{p_2}{3})^3 + (\frac{q_2}{2})^2 > 0$ and

$$\Lambda_3 \leq \frac{T_s + 2T_s^2}{3} + \sqrt[3]{-\frac{q_1}{2} + \sqrt{(\frac{p_1}{3})^3 + (\frac{q_1}{2})^2}} + \sqrt[3]{-\frac{q_1}{2} - \sqrt{(\frac{p_1}{3})^3 + (\frac{q_1}{2})^2}} \stackrel{def}{=} C \tag{D.16}$$

$$\Lambda_4 \leq \frac{T_s(T_s - k)}{3} + \sqrt[3]{-\frac{q_2}{2} + \sqrt{(\frac{p_2}{3})^3 + (\frac{q_2}{2})^2}} + \sqrt[3]{-\frac{q_2}{2} - \sqrt{(\frac{p_2}{3})^3 + (\frac{q_2}{2})^2}} \stackrel{def}{=} D. \tag{D.17}$$

It can be shown that

$$G = \sqrt[3]{-\frac{q_1}{2} + \sqrt{(\frac{p_1}{3})^3 + (\frac{q_1}{2})^2}} + \sqrt[3]{-\frac{q_1}{2} - \sqrt{(\frac{p_1}{3})^3 + (\frac{q_1}{2})^2}} > 0 \tag{D.18}$$

$$H = \sqrt[3]{-\frac{q_2}{2} + \sqrt{(\frac{p_2}{3})^3 + (\frac{q_2}{2})^2}} + \sqrt[3]{-\frac{q_2}{2} - \sqrt{(\frac{p_2}{3})^3 + (\frac{q_2}{2})^2}} > 0 \tag{D.19}$$

are zeros of $f(x) = x^3 + p_1x + q_1$ and $h(x) = x^3 + p_2x + q_2$, respectively. As both $f(x)$ and $g(x)$ are monotonically increasing functions and $f'(x) = 3x^2 + p_1 > g'(x) = 3x^2 + p_2 > 0, \forall x, f(x) < g(x), \forall x < \hat{x}$, where \hat{x} is the single intersection point given by

$$\hat{x} = \frac{q_2 - q_1}{p_1 - p_2} > 0.$$

The fact that

$$\begin{aligned}
f(\hat{x}) &= \left(\frac{q_2 - q_1}{p_1 - p_2} \right)^3 + p_1 \frac{q_2 - q_1}{p_1 - p_2} + q_1 = \left(\frac{q_2 - q_1}{p_1 - p_2} \right)^3 + \frac{p_1q_2 - p_2q_1}{p_1 - p_2} \\
&> \left(\frac{q_2 - q_1}{p_1 - p_2} \right)^3 + \frac{p_2q_2 - p_2q_1}{p_1 - p_2} > 0. \tag{D.20}
\end{aligned}$$

implies that the only real zero of $f(x), G$, is larger than that of $g(x), H$, and thus $C > D$.

Substituting $\Lambda_i = C$ into (A.3), we obtain an upper-bound with a very complicated expression. To have an upper-bound with a simpler form, we notice that $\|\Lambda_i\|_{M=TC T_s} = 0$

gives

$$\max \Lambda_i = T_s T_c \left\lfloor \frac{C}{T_s \cdot T_c} \right\rfloor > C - T_s T_c \quad (\text{D.21})$$

Hence a less tight upper-bound is given by

$$\begin{aligned} w_{4,min,ibp} &\leq 4 + 4\alpha \left(\min_{\Lambda_i} \left\{ \left\lfloor \frac{T_s \cdot L}{\Lambda_i} \right\rfloor - 1 \right\} \right) + 4\beta \\ &= 4 + 4\alpha \left(\left\lfloor \frac{T_s L}{T_s T_c \left\lfloor \frac{C}{T_s T_c} \right\rfloor} \right\rfloor - 1 \right) + 4\beta \\ &\leq 4 + 4\alpha \left(\left\lfloor \frac{T_s L}{C - T_s T_c} \right\rfloor - 1 \right) + 4\beta < 4 + 4\alpha \cdot \frac{T_s L}{C - T_s T_c} + 4\beta \quad (\text{D.22}) \end{aligned}$$

The upper-bound of weight-4 input sequence in the Case (a) is twice the upper-bound of weight-2 input sequence shown in eqn. (5.17).

Note that

$$\begin{aligned} E \stackrel{def}{=} & C - \sqrt{T_s L} \frac{T_s + 2T_s^2}{3} + \sqrt[3]{-\frac{q_1}{2} + \sqrt{\left(\frac{p_1}{3}\right)^3 + \left(\frac{q_1}{2}\right)^2}} \\ & + \sqrt[3]{-\frac{q_1}{2} - \sqrt{\left(\frac{p_1}{3}\right)^3 + \left(\frac{q_1}{2}\right)^2}} - \sqrt{T_s L} \end{aligned} \quad (\text{D.23})$$

and

$$\left(E - \frac{T_s + 2T_s^2}{3} + \sqrt{T_s L} \right)^3 = -q_1 - p_1 \left(E - \frac{T_s + 2T_s^2}{3} + \sqrt{T_s L} \right). \quad (\text{D.24})$$

In other words, E is a zero of the polynomial

$$g(x) = \left(x - \frac{T_s + 2T_s^2}{3} + \sqrt{T_s L} \right)^3 + p_1 \left(x - \frac{T_s + 2T_s^2}{3} + \sqrt{T_s L} \right) + q_1$$

which, like $f(x)$ defined before, is a monotonically increasing function and has only one

real zero. For $T_s \geq 2$,

$$\begin{aligned}
g(0) &= \left(-\frac{T_s + 2T_s^2}{3} + \sqrt{T_s L} \right)^3 + p_1 \left(-\frac{T_s + 2T_s^2}{3} + \sqrt{T_s L} \right) + q_1 \\
&= -T_s^2 L^2 + \left(T_s^{\frac{3}{2}} + 3T_s^{\frac{5}{2}} \right) L^{\frac{3}{2}} - (T_s^2 + 2T_s^3) L \\
&< L^{\frac{3}{2}} \left(-T_s^2 L^{\frac{1}{2}} + \left(T_s^{\frac{3}{2}} + 3T_s^{\frac{5}{2}} \right) \right) \\
&< L^{\frac{3}{2}} \left(-T_s^2 \left(\frac{10}{3} T_s^3 + T_s^2 - \frac{T_s}{3} \right)^{\frac{1}{2}} + \left(T_s^{\frac{3}{2}} + 3T_s^{\frac{5}{2}} \right) \right) < 0 \quad (\text{D.25})
\end{aligned}$$

The last inequality holds because both $T_s^2 \left(\frac{10}{3} T_s^3 + T_s^2 - \frac{T_s}{3} \right)$ and $T_s^{\frac{3}{2}} + 3T_s^{\frac{5}{2}}$ are positive real numbers and

$$\begin{aligned}
T_s^4 \left(\frac{10}{3} T_s^3 + T_s^2 - \frac{T_s}{3} \right) - \left(T_s^{\frac{3}{2}} + 3T_s^{\frac{5}{2}} \right)^2 &= \frac{10}{3} T_s^7 + T_s^6 - \frac{T_s^5}{3} - 9T_s^5 - 6T_s^4 - T_s^3 \\
&> \frac{40}{3} T_s^5 + 2T_s^5 - \frac{T_s^5}{3} - 9T_s^5 - 6T_s^4 - T_s^3 \\
&> 12T_s^5 - 6T_s^4 - T_s^3 > 0.
\end{aligned}$$

Hence E is positive and so

$$\begin{aligned}
&4 + 4\alpha \cdot \frac{T_s L}{C - T_s T_c} + 4\beta \\
&< 4 + 4\alpha \frac{T_s L}{\sqrt{T_s L} - T_s T_c} + 4\beta \\
&= 2 \left(2 + 2\alpha \frac{T_s L}{\sqrt{T_s L} - T_s T_c} + 2\beta \right). \quad (\text{D.26})
\end{aligned}$$

Appendix E

Puncturing Patterns

Puncturing systematic bits enhances the error rate performance and references [59, 52] have shown evidences. This part proposes irregular puncturing patterns for code rates 3/4, 4/5 and 8/9. Tables E.1, E.2 and E.3 show the regular and irregular puncturing patterns for code rates 3/4, 4/5 and 8/9 respectively. C_0 corresponds to the systematic part, and C_1 and C_2 correspond to the parity part associated with information sequence and interleaved information sequence respectively. The puncturing period is 48 in these puncturing tables and '1' and '0' denote kept and punctured. These puncturing patterns are searched for the $G(D) = \left[1 \frac{1+D+D^3}{1+D^2+D^3} \right]$ defined in 3GPP turbo code [1, 2, 3].

These puncturing patterns are compared by 3GPP Rel'6 turbo code interleaver [1, 2] and the B-IBP interleaver described in Section 4.5. The parameters of the double prime interleaver are $(p, s) = (5, 18)$ for $L = 64$ and $(p, s) = (47, 7)$ for $L = 128$, and the IBP sequences are $\{0, 1\}$ for $N = 2$, $\{0, 1, 2, 4, 3, 6, 7, 5\}$ for $N = 8$, $\{0, 1, 2, 4, 8, 3, 6, 12, 11, 5, 10, 7, 14, 15, 13, 9\}$ for $N = 16$ and $\{0, 1, 2, 4, 8, 16, 5, 10, 20, 13, 26, 17, 7, 14, 28, 29, 31, 27, 19, 3, 6, 12, 24, 21, 15, 30, 25, 23, 11, 22, 9, 18\}$ for $N = 32$. Linear log-MAP algorithm with 8 iterations is applied and AWGN channel is considered.

Figs. E.4-E.9 show the simulation results. These figures reveals that irregular puncturing pattern outperforms regular puncturing pattern by 0.1-1.0 dB at FER= 10^{-4} except for code rate=8/9 with $K = 256$. Puncturing too many systematic bits at high rate for short block length does not benefit the error rate performance although the distance

Table E.1: Puncturing patterns for code rate=3/4.

Regular Puncturing Pattern						
C_0	11111111	11111111	11111111	11111111	11111111	11111111
C_1	10000010	00001000	00100000	10000010	00001000	00100000
C_2	10000010	00001000	00100000	10000010	00001000	00100000
Irregular Puncturing Pattern						
C_0	10111111	10111111	10111111	10111111	10111111	10111111
C_1	01000100	01000100	01000100	01000100	01000100	01000000
C_2	01000100	01000100	01000100	01000100	01000100	01000000

Table E.2: Puncturing patterns for code rate=4/5.

Regular Puncturing Pattern						
C_0	11111111	11111111	11111111	11111111	11111111	11111111
C_1	10000000	10000000	10000000	10000000	10000000	10000000
C_2	10000000	10000000	10000000	10000000	10000000	10000000
Irregular Puncturing Pattern						
C_0	10111111	10111111	10111111	10111111	10111111	10111111
C_1	01000100	01000000	01000100	01000000	01000100	01000000
C_2	01000100	01000000	01000100	01000000	01000100	01000000

property may be improved.

The performance gain for the B-IBPTC is more than that for 3GPP Rel'6 turbo code. When code rate=3/4 and 4/5, the performance curves for interleaver length 4096 bits are crossed but this does not occur for the B-IBPTC. Our irregular puncturing pattern benefit more a turbo code with better distance property.

Table E.3: Puncturing patterns for code rate=8/9.

Regular Puncturing Pattern						
C_0	11111111	11111111	11111111	11111111	11111111	11111111
C_1	10000000	00000000	10000000	00000000	10000000	00000000
C_2	10000000	00000000	10000000	00000000	10000000	00000000
Irregular Puncturing Pattern						
C_0	10111111	11111011	11111111	10111111	11111011	11111111
C_1	10100000	00000100	00000000	01000000	00000100	00000000
C_2	10100000	00000100	00000000	01000000	00000100	00000000

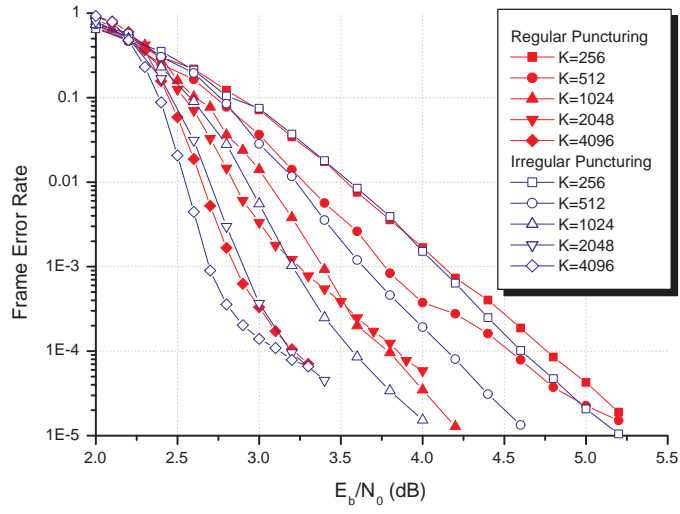


Figure E.1: Frame error rate comparison between regular and irregular puncturing patterns for code rate=3/4 3GPP Rel'6 turbo code.

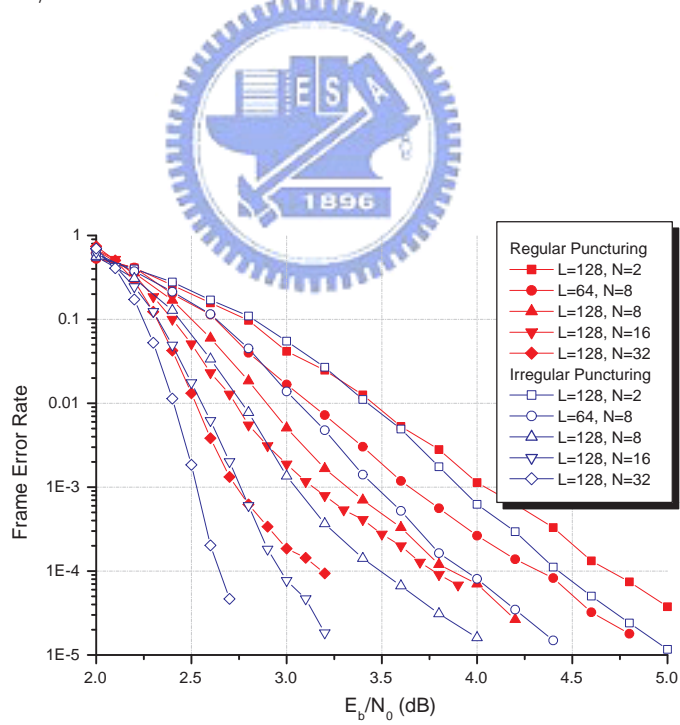


Figure E.2: Frame error rate comparison between regular and irregular puncturing patterns for code rate=3/4 B-IBPTC.

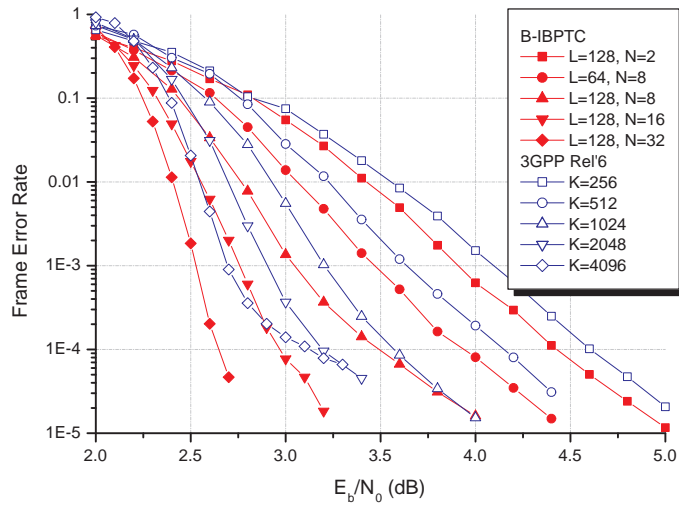


Figure E.3: Frame error rate comparison between 3GPP Rel'6 turbo code and B-IBPTC for code rate=3/4 irregular puncturing pattern.

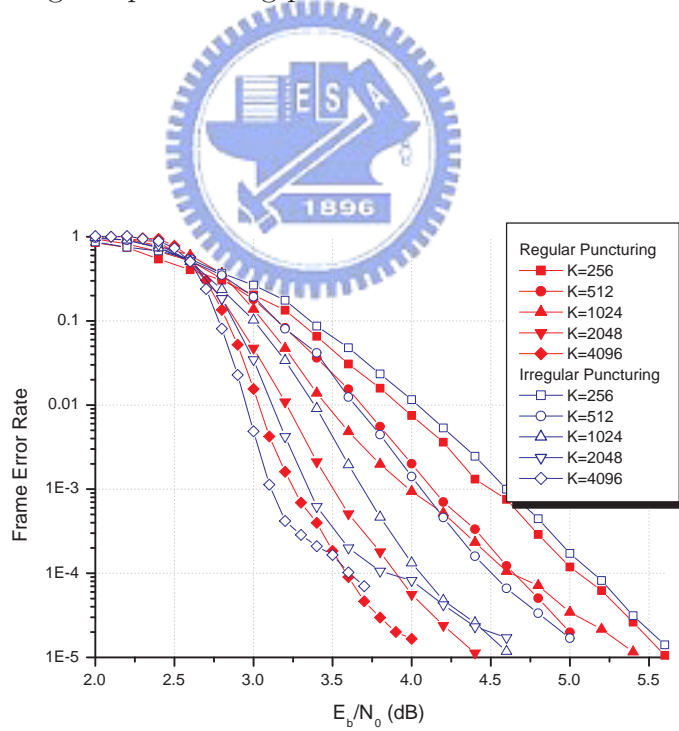


Figure E.4: Frame error rate comparison between regular and irregular puncturing patterns for code rate=4/5 3GPP Rel'6 turbo code.

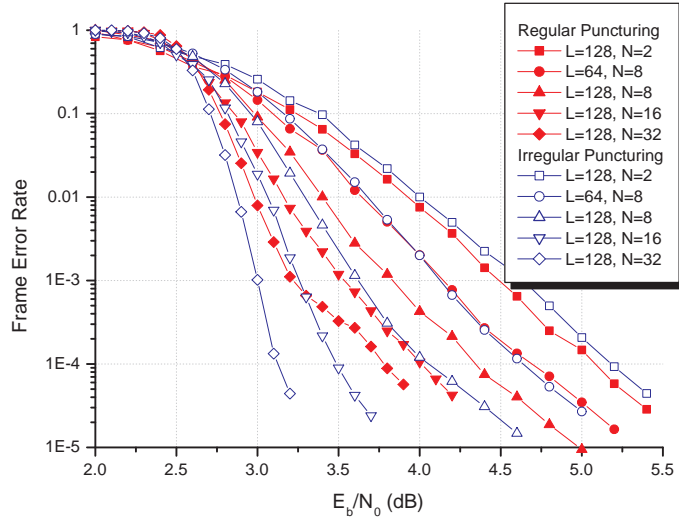


Figure E.5: Frame error rate comparison between regular and irregular puncturing patterns for code rate=4/5 B-IBPTC.

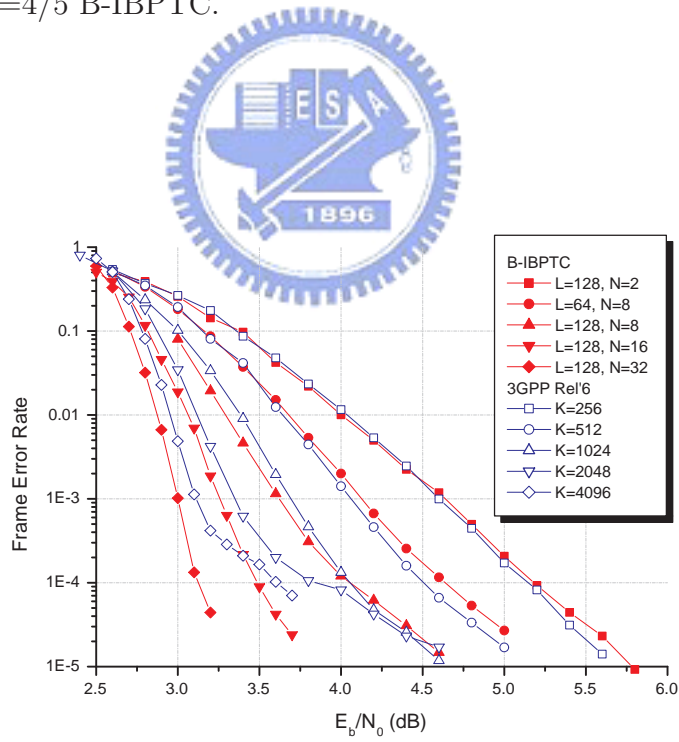


Figure E.6: Frame error rate comparison between 3GPP Rel'6 turbo code and B-IBPTC for code rate=4/5 irregular puncturing pattern.

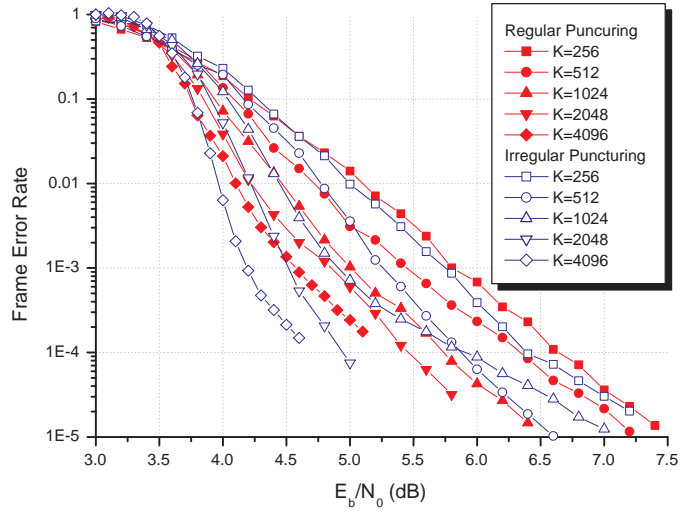


Figure E.7: Frame error rate comparison between regular and irregular puncturing patterns for code rate=8/9 3GPP Rel'6 turbo code.

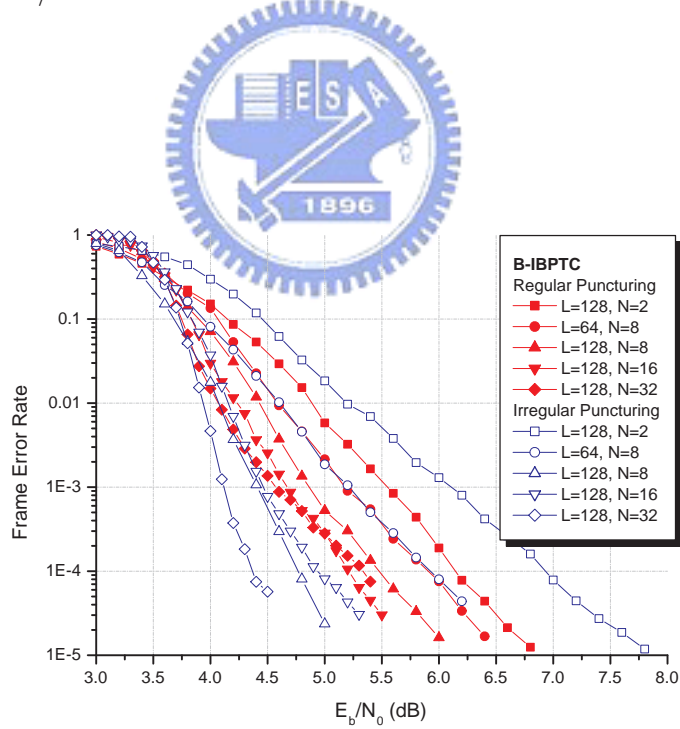


Figure E.8: Frame error rate comparison between regular and irregular puncturing patterns for code rate=8/9 B-IBPTC.

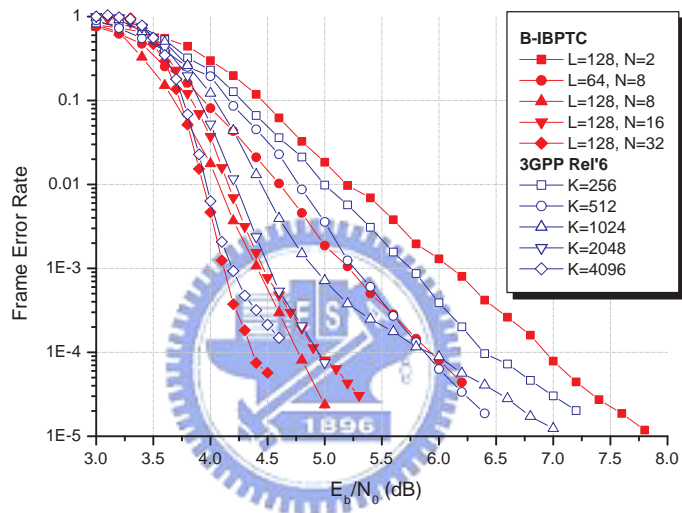


Figure E.9: Frame error rate comparison between 3GPP Rel'6 turbo code and B-IBPTC for code rate=8/9 irregular puncturing pattern.

Bibliography

- [1] *TS 25.222 v3.1.1 Multiplexing and channel coding (TDD)*, 3GPP TSG RAN WG1, Dec. 1999.
- [2] *TS 25.212 v6.4.0 Multiplexing and channel coding (FDD)*, 3GPP TSG RAN WG1, Mar. 2005.
- [3] *TS 36.212 V1.0.0 Multiplexing and channel coding*, 3GPP TSG RAN WG1, Mar. 2007.
- [4] D. Agrawal, A. Vardy, "The turbo decoding algorithm and its phase trajectories," in *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 699-722, Feb. 2001.
- [5] J. B. Anderson, S. M. Hladik, "Tailbiting MAP decoders," in *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 297-302, Feb. 1998.
- [6] S. L. Ariyavisitakul, "Turbo space-time processing to improve wireless channel capacity," *IEEE Trans. Commun.*, vol.48, no. 8, pp. 1347-1359, Aug. 2000.
- [7] S. Baero, J. Hagenauer, M. Witzke, "Iterative detection of MIMO transmission using a list-sequential (LISS) detector," in *Int'l Conf. Commun.*, Anchorage, USA, May 2003.
- [8] L. R. Bahl, J. Cocke, F. Jelinek, F. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on Inform. Theory*, vol. 20, no. 2, pp. 284-287, Mar. 1974.

- [9] S. Benedetto, G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 409-428, Mar. 1996.
- [10] S. Benedetto, D. Divsalar, G. Montorsi, F. Pollara, "A soft-input soft-output APP module for iterative decoding of concatenated codes," in *IEEE Commun. Letters*, vol. 1, no. 1, pp. 22-24, Jan. 1997.
- [11] S. Benedetto, G. Montorsi, "Performance of continuous and blockwise decoded turbo codes," *IEEE Commun. Lett.*, vol. 1, no. 3, pp. 77-79, May 1997.
- [12] C. Berrou, Y. Saouter, C. Douillard, S. Kérrouédan, M. Jézéquel, "Designing good permutations for turbo codes: towards a single model," in *Proc. IEEE Int'l Conf. Commun.*, Paris, France, vol. 1, pp. 341-345, Jun. 2004.
- [13] C. Berrou, A. Glavieux, and P. L. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," in *Proc. Proc. IEEE Int'l Conf. Commun.*, Geneva, Switzerland, pp. 1064-1070, May 1993.
- [14] C. Berrou, M. Jézéquel, "Non-binary convolutional codes for turbo coding," in *Electronics. Letters*, vol. 35, no. 1, pp. 39-40, Jan 1999.
- [15] C. Berrou, M. Jézéquel, C. Douillard, S. Kerouedan, "The advantages of non-binary turbo codes," in *Proc. ITW2001*, Sep. 2001.
- [16] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, C. Nicol, "A 24Mb/s radix-4 log MAP turbo decoder for 3GPP-HSDPA mobile wireless," in *ISSCC Dig. Tech. Papers*, pp. 151-484, 2003.
- [17] P.J. Black, T.H.-Y. Meng, "Hybrid survivor path architectures for Viterbi decoders," *ICASSP 93*, vol. 1, pp. 433-436, Apr. 1993.

- [18] W. J. Blackert, E. K. Hall, S. G. Wilson, "Turbo code termination and interleaver conditions," *Electron. Lett.*, vol. 31, pp. 2082-2084, Nov. 1995.
- [19] D. W. Bliss, A. M. Chan, N. B. Chang, "MIMO wireless communication channel phenomenology," in *IEEE Trans. Antennas and Propagation*, vol. 52, no. 8, pp. 2073-2082, Aug. 2004.
- [20] B. Bougard, A. Giulietti, V. Derudder, J. Willem, S. Dupond, L. Hollevoet, F. Catthoor, L. V. der Perre, H. D. Man, R. Lauwereins, "A scalable 8.7nj/bit 75.6Mb/s parallel concatenated convolutional (turbo-)codec," in *ISSCC Dig. Tech. Papers*, pp. 152-484, 2003.
- [21] E. Boutillon, W. J. Gross, P. G. Gulak, "VLSI architectures for the MAP algorithm," in *IEEE Trans. Commun.*, vol. 51, no. 2, pp. 175-185, Feb. 2003.
- [22] E. Boutillon, D. Gnaedig, "Maximum spread of D-dimensional multiple turbo codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1237-1242, Aug. 2005.
- [23] J. Boutros, G. Caire, E. Viterbo, H. Sawaya, S. Vialle, "Turbo code at 0.03 dB from capacity limit," in *Proc. Int'l Sympo. on Inform. Theory*, pp. 56, 30 Jun.-5 Jul. 2002.
- [24] M. Breiling, J. B. Huber, "Upper bound on the minimum distance of turbo codes," *IEEE Trans. Commun.*, pp. 808-815, May 2001.
- [25] M. Breiling, J. B. Huber, "Combinatorial analysis of the minimum distance of turbo codes," *IEEE Trans. Inform. Theory*, pp. 2737-2750, Nov. 2001.
- [26] A. J. Blanksby, C. J. Howland, "A 690mW 1Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," in *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 404-412, Mar. 2002.

- [27] S. T. Brink, "Convergence of iterative decoding," in *Electronics Letters*, vol. 35, no. 13, pp. 1117-1119, Jun. 1999.
- [28] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," in *IEEE Trans. Commun.*, vol. 49, no. 10, pp. 1727-1737, Oct. 2001.
- [29] R. Brualdi, *Introductory Combinatorics*, Amsterdam, The Netherlands: North-Holland, 1977.
- [30] S. Crozier, J. Lodge, P. Guinand, A. Hunt, "Performance of turbo codes with relative prime and golden interleaving strategies", in *Proc. of the 6th Int'l Mobile Satellite Conference (IMSC '99)*, Ottawa, Ontario, Canada, pp. 268-275, Jun. 16-18, 1999.
- [31] S. Crozier, P. Guinand, "Distance upper bounds and true minimum distance results for turbo codes with DRP interleavers," in *Proc. 3rd Int'l Sympo. on Turbo Codes & Related Topics*, Brest, France, pp. 169-172, Sep. 2003.
- [32] F. Daneshgaran, P. Mulassano, "Interleaver pruning for construction of variable-length turbo codes," in *Trans. Inform. Theory* vol. 50, no.3, pp. 455-466, Mar. 2004.
- [33] V. C. Gaudet, R. J. Gaudet, G. Gulak, "Programmable interleaver design for analog iterative decoders," in *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 49, no. 7, pp. 457-464, Jul. 2002.
- [34] D. Divsalar, S. Dolinar, F. Pollara, "Iterative turbo decoder analysis based on density evolution." in *IEEE J. Select. Areas Commun.*, vol. 19, no. 5, pp. 891-907, May 2001.

- [35] R. Dobkin, M. Peleg, R. Ginosar, "Parallel interleaver design and VLSI architecture for low-lantecy MAP turbo decoders," in *IEEE Trans. VLSI Systems*, vol. 13, no. 4, pp. 427-438 , Apr. 2005.
- [36] C. Douillard, M. Jézéquel, C. Berrou, "Iterative correction of intersymbol interference: turbo-equalization," in *European Trans. Telecommunications*, vol. 6, no. 5, pp. 507-511, Sep./Oct. 1995.
- [37] DVB, "Interaction channel for satellite distribution systems," ETSI EN 301 790, V1.2.2, pp. 21-24, Dec. 2000.
- [38] DVB, "Interaction channel for digital terrestrial television," ETSI EN 301 958, V1.1.1, pp. 28-30, Aug. 2001.
- [39] H. El Gamal, A. R. Hammons, Jr., "Analyzing the turbo decoder using the Gaussian approximation," in *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 671-686, Feb. 2001.
- [40] W. Feng, J. Yuan, B. S. Vucetic, "A code-matched interleaver design for turbo codes," in *IEEE Trans. Commun.*, vol. 50, no. 6, pp. 926-937, Jun. 2002.
- [41] G. D. Forney, "The Viterbi algorithm," *Proc. of the IEEE*, vol 61, no. 3, pp. 268-278, Mar. 1973.
- [42] G. D. Forney, Jr., "Codes on graphs: normal realizations," in *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 520-548, Feb. 2001.
- [43] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," in *IEEE Trans. Inform. Theory*, vol. 50, no. 8, pp. 1788-1793, Aug. 2004.
- [44] C. Fragouli, R. D. Wesel, "Semi-random interleaver design criteria," in *Proc. Globecom'99*, Rio de Janeiro, Brazil, pp. 2352-2356, 1999.

- [45] B. J. Frey, D. J. C. MacKay, "Irregular turbocodes," in *Proc. Int'l Symposium on Information Theory*, pp. 121, Jun. 25-30, 2000.
- [46] R. G. Gallager, *Low-density parity-check codes*, MIT Press, Cambridge, Mass., 1963.
- [47] O. Gazi, Ö. Yilmaz, "Fast decodable turbo codes," in *IEEE Commun. Letters*, vol. 11, no. 2, pp. 173-175, Feb. 2007.
- [48] W. Gellert, H. Justner, M. Hellwich, H. Kastner, *The Vnr Concise Encyclopedia of Mathematics*, Van Nostrand Reinhold, pp. 97-99, 1977.
- [49] A. Giulietti, L. Van der Perre, M. Strum, "Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements," in *Electronics Letter*, vol. 38, no. 5, pp. 232-234, Feb. 2002.
- [50] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes" *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 429-445, Mar. 1996.
- [51] E. K. Hall, S. G. Wilson, "Stream-oriented turbo codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 5, pp. 1813-1831, Jul. 2001.
- [52] K. Hasung, G. L. Stüber, "Rate compatible punctured turbo coding for W-CDMA," in *Proc. IEEE Int'l Conf. Personal Wireless Commun.*, pp. 143-147, Dec. 2000.
- [53] P. Hoeher, J. Lodge, "'Turbo DPSK': iterative differential PSK demodulation and channel decodign," in *IEEE Trans. Commun.*, vol. 47, no. 6, pp. 837-843, Jun. 1999.
- [54] J. Hokfelt, O. Effors, T. Maseng, "A turbo code interleaver design criterion based on the performance of iterative decoding," *IEEE Commun. Lett*, vol. 5, no. 2, pp. 52-54, Feb. 2001.

- [55] C. J. Howland, A. J. Blanksby, "A 220mW 1Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," in *IEEE Conf. on Custom Integrated Circuits*, pp. 293-296, May. 2001.
- [56] *IEEE Std 802.16e-2005*, 802.16 TGe, Feb. 2006.
- [57] H. Jin, A. Khandekar, R. J. McEliece, "Irregular repeat-accumulate codes," in *Proc. 3rd Int'l Sympo. on Turbo Codes & Related Topics*, Brest, France, pp. 1-8, Sep. 2000.
- [58] R. Johannesson, K. Sh. Zigangirov, *Fundamentals of Convolutional Codes*, The Institute of Electrical and Electronics Engineering, Inc, 1999.
- [59] M. A. Kousa, A. H. Mugaibel, "Puncturing effects on turbo codes," in *IEE Proc. Commun.*, vol. 149, pp. 132-138, Jun. 2002.
- [60] F. R. Kschischang, B. J. Frey, H.-A. Loeliger, "Factor graphs and the sum-product algorithm," in *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 498-519, Feb. 2001.
- [61] T.-C. Lee, J. Cong, "The new line in IC design," in *IEEE Spectrum*, pp. 52-58, Mar. 1997.
- [62] C.-C. Lin, K.-L. Lin, H.-C. Chang, C.-Y. Lee, "A 3.33Gb/s (1200,720) low-density parity check code decoder," in *Proc. ESSCIRC 2005*, pp.211-214, Sep. 2005.
- [63] M. M. Mansour and N. R. Shanbhag, "High throughput LDPC decoders," in *IEEE Trans. VLSI Systems*, vol. 11, pp. 976-996, Dec. 2003.
- [64] J. L. Massey, M. K. Sain, "Codes, automata, and continuous systems: explicit interconnections," *IEEE Trans. on Automatic Control*, AC-12:644-650, 1968.
- [65] A. Matache, S. Dolinar, F. Pollara, "Stopping rules for turbo decoders," in *TMO Progress Report 42-142*, 15 Aug. 2000.

- [66] R. J. McEliece, D. J. C. MacKay, J.-F. Cheng “Turbo decoding as an instance of Pearl’s “Belief propagation” algorithm,” in *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 260-264, Feb. 1998.
- [67] H. Moussa, O. Muller, A. Baghdadi, M. M. Jézéquel, “Butterfly and Benes-based on-chip communication networks for multiprocessor turbo decoding,” in *Proc. Design, Automation and Test in Europe*, pp. 654-659, Apr. 2007.
- [68] O. Muller, A. Baghdadi, M. Jézéquel, “ASIP-based multiprocessor SOC design for simple and double binary turbo decoding,” in *Proc. Design, Automation and Test in Europe*, pp. 6-10, Mar. 2006.
- [69] A. Nimbalkar, K. T. Blankenship, B. Classon, T. E. Fuja, D. J. Costello, Jr., “Inter-window shuffle interleavers for high throughput turbo decoding,” in *Proc. 3rd Int’l Sympo. on Turbo Codes & Related Topics*, Brest, France, pp. 355-358, Sep. 2003.
- [70] A. Nimbalkar, T. E. Fuja, D. J. Costello Jr., T. K. Blankenship, B. Classon, “Contention-free interleavers,” in *Proc. Int’l Symposium on Information Theory*, pp. 52, Jun. 27-Jul. 2, 2004.
- [71] S. Y. Le Goff, “Signal constellations for bit-interleaved coded modulation,” in *IEEE Trans. Inform. Theory*, vol. 49, no. 1, pp.307-313, Jan. 2003.
- [72] X. Li, J. A. Ritcey, “Trellis-coded modulation with bit interleaving and iterative decoding,” in *IEEE J. Select. Areas Commun.*, vol. 17, no. 4, pp. 715-724, Apr. 1999.
- [73] S. Paraharalabos, P. Sweeney, B. G. Evans, “Constant log-MAP decoding algorithm for duo-binary turbo codes,” in *Electronics Letters*, vol. 42, no. 12, pp. 709-710, Jun. 2006.

- [74] L. C. Perez, J. Seghers, D. J. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, Vol. 42, no. 6, pp. 1698-1709, Nov. 1996.
- [75] A. Perotti, S. Benedetto, "A new upper bound on the minimum distance of turbo codes," *IEEE Trans. Inform. Theory*, vol. 50, no. 12, pp. 2985-2997, Dec. 2004.
- [76] J. Pittermann, M. Lentmaier, K. S. Zigangirov, "On bandwidth-efficient convolutional LDPC code," in *Proc. Int'l Sympo. on Inform. Theory*, pp. 235, 29 Jun.-4 Jul. 2003.
- [77] V. Poor, S. Verdú, "Probability of error in MMSE multiuser detection," in *IEEE Trans. Inform. Theory*, vol. 43, no. 3, pp. 858-871, May 1997.
- [78] G. Prescher, T. Gemmeke, T. G. Noll, "A parametrizable low-power high-throughput turbo-decoder," in *IEEE ICASSP 2005*, vol. 5, pp. 25-28, Mar. 2005.
- [79] J. G. Proakis, *Digital communications 4th*, Mc Graw Hill, Inc., 2000.
- [80] R. M. Pyndiah, "Near-optimum decoding of product codes: block turbo codes," in *IEEE Trans. Commun.*, vol. 46, pp. 1003-1010, Aug. 1998.
- [81] R. M. Pyndiah, A. Glavieux, A. Picart, S. Jacq, "Near Optimum decoding of product codes," in *Proc. IEEE Globecom '94*, San Francisco, USA, vol. 1, pp. 339-343, Nov. 28-Dec. 2 1994.
- [82] T. J. Richardson, R. L. Urbanke, "Thresholds for turbo codes," in *Proc. IEEE Int'l Symp. Inform. Theory*, Sorrento, Italy, pp. 172, June 2000.
- [83] T. J. Richardson, R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," in *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 599-618, Feb. 2001.

- [84] P. Robertson, T. Wörz, “Bandwidth-efficient turbo trellis-coded modulation using punctured component codes,” in *IEEE J. Select. Areas Commun.*, pp. 206-218, Feb. 1998.
- [85] J. Ryu, O. Y. Takeshita, “On quadratic inverses for quadratic permutation polynomials over integer rings,” in *IEEE Trans. Inform. Theory*, vol. 52, no. 3, pp. 1254-1260, Mar. 2006.
- [86] H. R. Sadjadpour, N. J. A. Sloane, M. Salehi, G. Nebe, “Interleaver design for turbo codes,” *IEEE J. Select. Areas Commun.*, vol. 19, no. 5, pp. 831-836, May 2001.
- [87] I. Sason, S. Shamai, “Improved upper bounds on the ML decoding error probability of parallel and serial concatenated turbo codes via their ensemble distance spectrum,” in *IEEE Trans. Inform. Theory*, vol. 46, no. 1, pp. 24-47 Jan. 2000.
- [88] R. Y. Shao, S. Lin, and M. P. C. Fossorier, “Two simple stopping criteria for turbo decoding,” in *IEEE Trans. Commun.*, vol. 47, no. 8, Aug. 1999.
- [89] Jens Sparso, Henrik N. Jorgensen, Erik Paaske, Steen Pendersen, and Thomas Rubner-Petersen, “An area-efficient topology for VLSI implementation of Viterbi decoders and other shuffle-exchange type structures,” *IEEE J. Solid-State Circuit*, vol. 26, No.2, pp. 90-97, February 1991.
- [90] J. Sun, O. Y. Takeshita, “Interleavers for turbo codes using permutation polynomials over integer rings,” in *IEEE Trans. Inform. Theory*, vol. 51, no. 1, pp. 101-119, Jan. 2005.
- [91] O. Y. Takeshita, O. M. Collins, P. C. Massey, D. J. Costello, Jr., “A note on asymmetric turbo codes,” in *IEEE Commun. Letters*, vol. 3, pp. 69-71, Mar. 1999.

- [92] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," in *IEEE Trans. Inform. Theory*, vol. 52, no. 3, pp. 1249V1253, Mar. 2006.
- [93] O. Y. Takeshita, "Permutation polynomial interleavers: an algebraic-geometric perspective," in *IEEE Trans. Inform. Theory*, vol. 53, no. 6, pp. 2116-2132, Jun. 2007.
- [94] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, D. J. Costello "LDPC block and convolutional codes based circulant matrices," in *IEEE Trans. Inform. Theory*, vol. 50, no. 12, pp. 2966-2984, Dec. 2004.
- [95] A. Tarable, S. Benedetto, "Mapping interleaving laws to parallel turbo decoder architectures," in *IEEE Commun. Letters*, vol. 8, no. 3, pp. 162-164, Mar. 2004.
- [96] A. Tarable, S. Benedetto, G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures," in *IEEE Trans. Inform. Theory*, vol. 50, no. 9, pp. 2002-2009, Sep. 2004.
- [97] M. Thul, N. Wehn, L. Rao, "Enabling high-throughput turbo-decoding throughput concurrent interleaving," in *Proc. IEEE Int'l Sympo. on Circuits and Systems*, pp. 897-900, Phoenix, USA, May 2002.
- [98] M. J. Thul, F. Gilbert, N. Wehn, "Optimized concurrent interleaver for high-speed turbo-decoding," in *Proc. IEEE Int'l Conf. on Electronics, Circuits and Systems*, Dubrovnik, Croatia, pp. 1099-1102, Sep. 2002.
- [99] G. Ungerboeck, "Channel coding with multilevel/phase signals," in *IEEE Trans. Inform. Theory*, vol. 28, no. 1, pp. 55-67, Jan. 1982.
- [100] G. Ungerboeck, "Trellis-coded modulation with redundant signal sets part I: introduction," in *IEEE Commun. Magazine*, vol. 25, no. 2, pp. 5-11, Feb. 1987.

- [101] P. Urard, L. Paumier, M. Viollet, E. Lantreibeq, H. Michel, S. Muroor, B. Gupta, "A generic 350Mb/s turbo-codec based on a 16-states SISO decoder," in *ISSCC Dig. Tech. Papers*, pp. 424-536, 2004.
- [102] M. C. Valenti, J. Sun, "The UMTS turbo code and an efficient decoder implementation suitable for software-defined radios," in *Int'l Journal of Wireless Information Networks*, vol. 8, no. 4, pp. 203-215, Oct. 2001.
- [103] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. 13, no 2, pp. 260-269, Apr. 1967.
- [104] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 260-264, Feb. 1998.
- [105] K. Wu, H. Li, Y. Wang, "The influence of interleaver on the minimum distance of turbo code," in *Electron. Lett.*, vol. 35, no. 17, pp. 1456-1458, Aug. 1999.
- [106] C. Weiss, C. Bettstetter and S. Riedel, "Code construction and decoding of parallel concatenated tail-biting codes," *IEEE Trans. Inform. Theory*, Vol. 47, no. 1, pp. 366-386, Jan. 2001.
- [107] C. Weiss, C. Bettstetter, S. Riedel, D. J. Costello, "Turbo decoding with tail-biting trellis," in *ISSSE'98*, Pisa, Italy, pp. 343-348, 1998.
- [108] N. Wiberg, *Codes and decoding on general graphs*, Ph. D. thesis, Department of Electrical Engineering, U. Linkoping, Sweden, 1996.
- [109] S. B. Wicker, *Error Control Systems for Disgital Communication and Storage 2nd*, Prentice Hall Internationl, Inc., 1995.

- [110] S. Winograd, "On computing the discrete Fourier transform," in *Mathematics of Computation*, vol. 32, pp. 175-199, 1978.
- [111] W. Wolf *Modern VLSI Design-System on Chip Design 3rd*, Baker & Taylor Books, 2002.
- [112] C.-C. Wong, C.-H. Tang, M.-W. Lai, Y.-X. Zheng, C.-C. Lin, H.-C. Chang, C.-Y. Lee, Yu T. Su, "A 0.22nJ/iter 0.13 μ m turbo decoder chip using inter-block permutation interleaver," in *Proc. IEEE CICC 2007*, San Jose, California, USA, Sep. 16-19, 2007.



博士候選人資料

姓 名：鄭延修

性 別：男

出生年月日：民國 64 年 6 月 21 日

籍 貫：浙江省永嘉縣

學 歷：交通大學 電信工程學系 博士 88 年 9 月-
交通大學 電信工程學系 碩士 86 年 9 月-88 年 6 月
清華大學 電機工程學系 學士 82 年 9 月-86 年 6 月
國立師範大學附屬高級中學 79 年 9 月-82 年 6 月

經 歷：大華技術學院專任講師 89 年 9 月-91 年 6 月
大華技術學院專任講師 92 年 9 月-93 年 6 月
高維度有限公司負責人 94 年 1 月-

論文題目：用於高傳輸率渦輪碼之交錯器設計

尾 聲

與蘇育德教授這近十年的相處，教授對學生的幫助遠超過單純學術上的指導討論或論文繕改，還給予學生極大的空間去實踐想法，回溯 2002 年，教授鼓勵學生參加三明治計畫去德國接受國外研究環境的洗禮，瞭解國際學術圈的發展與研究態度，讓學生大大的提升了學術的視野，在 2005 年，教授牽起我與電子系的合作，讓整個研究內容變得更加務實與完整，在 2006 年，教授讓我能與工研院 4G 團隊在 3GPP 標準以我多年在學校所學在會議中盡情與國際大廠抗衡跟玩耍並給予學生支援，讓學生能夠開心的將 LDPC code 送出標準會場，也能讓 France Telecom 為學生的東西代言，在最後的 2007 年，教授將我從瘋狂帶回正常，並給我許多人生的建議，讓學生得以在未來數十年人生順遂圓滿。細數其間種種，難以用感謝兩字簡單帶過，學生希望未來能夠帶著老師當年回國的心情繼續為這塊土地努力並傳承。

在德國的那一年，因為三明治的關係能夠認識其他領域的朋友如斐敦、於琛、志偉、谷燕、秀慧、忠賢、昆平、書銘，並讓我能德國各地四處玩耍，讓我能對好東西能更多的品味，也讓我對研究有更加的判斷力，其間感謝亦書、伯薇讓我能德國的生活的輕鬆愜意。

在實驗同學中非常感謝彥志，你的見聞廣闊讓我在研究上的省去許多時間，你愛玩耍的個性讓我們可以去西班牙瘋狂的享受人生，聽著佛朗明哥，躺遍數個沙灘，因為你，讓我博士生活能夠多了許多樂趣不至讓我成為單純的工作狂。

另外我要感激實驗室學長陳聖志，讓我在博士班其間能夠有份工作不至在生活俗務上有諸多困擾，在博士班後期給予資金上的援助與周邊事務的幫忙，因為學長，讓我能夠在博士生涯有著不一樣的過程。

此外我也必須感謝銘賓，因為他的幫忙，在最後幾年的遊戲中得以揮灑的更為漂亮，在許多非學術上的討論中，可以得到許多的啟發。

當然也感謝在其他的學長、同學以及學弟妹們，立德、忠炫、錫嘉、昌明、人仰、淵斌，跟你們在一起相處討論，讓人生多了很多的樂趣，也讓我有了許多研究上的想法。

當然也感激我的父母這幾年來對我的等待，願意放他兒子當學生胡搞瞎搞不盡兒子的應盡責任。

最後我最想感謝的人，應該就是我花最多心思在帶的學妹汀華，在所有帶過的學弟妹中，她需要花最多的時間溝通與指導，也會在任何時候接到求助電話，但在所有帶過的學弟妹中，她也最為體貼讓人倍感溫馨，但在所有帶過的學弟妹中，她也最為惱人讓我起了無數的無名火。但也因為她，我能夠在畢業前找回人生的基調以更為謙卑的心態面對人生。少了她，我的人生不會有如此劇烈的改變，少了她，各位也應該就看不到這本論文，少了她，我這段博士生涯中許多經典瘋狂故事也都不會發生。因緣合和，難以言喻，高低起伏，驚險非常，人生如戲，莫過於此。

