

BLDSC no:- DX 95696

LOUGHBOROUGH
UNIVERSITY OF TECHNOLOGY
LIBRARY

AUTHOR/FILING TITLE

FRYDAS, N P

ACCESSION/COPY NO.

036000875

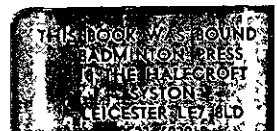
VOL. NO

CLASS MARK

Loan copy

27 JUN 1986

036000875 5



SOME NEW RESULTS

ON CONVOLUTIONAL CODES

by

Nikos P. Frydas

A Doctoral Thesis

Submitted in partial fulfilment of the requirements

for the award of

Doctor of Philosophy of the Loughborough University of Technology

1 January 1990

© by N.P. Frydas 1990

Loughborough University of Technology Library	
Date	Oct 91
Circ	
Acc No	036000875

63 y9908623

Dedicated

to my parents

Panos & Antigoni

for all they have done for me

2.10	A MATRIX EQUATION FOR THE hTH CHANNEL BLOCK	26
2.11	A NOVEL APPROACH TO THE GENERATOR MATRIX	28
2.12	GENERATOR POLYNOMIALS	31
2.13	GENERATOR-POLYNOMIAL MATRIX	33
2.14	NORMAL ENCODER	34
2.15	CATASTROPHIC CODES	35
2.16	SERIAL ENCODER	35
2.17	SYSTEMATIC CONVOLUTIONAL CODES	35
2.17.1.	Generator Matrix	36
2.17.2.	The Output of the Encoder	37
2.17.3.	Generator Sequences	38
2.17.4.	Generator Polynomials	38
2.17.5.	Generator-Polynomial Matrix	39
2.17.6.	Composite Generator-Polynomials	39
2.17.7.	Encoder	39
2.17.8.	Non-Catastrophic Codes	40
2.18	TYPE-II ENCODER	40
2.19	PARITY-CHECK POLYNOMIAL MATRIX	44
2.20	PARITY-CHECK MATRIX	45
2.21	SYNDROME	46
2.22	CONCLUSIONS	50
3	<u>DECODING OF CONVOLUTIONAL CODES</u>	53
3.1	INTRODUCTION	54
3.2	CONVOLUTIONAL ENCODER STATE-TRANSITION DIAGRAM	55
3.2.1.	Introduction	55
3.2.2.	Sequential Machines	56
3.2.3.	Structure of the Encoder State- Transition Diagram	57
3.3	TRELLIS DIAGRAM	64
3.4	VITERBI DECODING	68
3.5	SEQUENTIAL DECODING	70
3.6	SYNDROME DECODING	70
3.7	CONCLUSIONS	71
4	<u>ERROR-TRELLIS SYNDROME DECODING</u>	74
4.1	ANALYSIS OF ERROR-TRELLIS SYNDROME DECODING	75
4.1.1.	General Solution of the Syndrome Equation	75
4.1.2.	The Case of Systematic Codes	78
4.1.3.	Estimation of the Channel Error Digits	79
4.1.4.	Estimation of the Source Digits	80
4.1.5.	The Hyperchannel Error Polynomial	82
4.1.6.	Error-Trellis Syndrome Decoder	82
4.2	THE CONSTRAINED REGULATOR TRELLIS OF A SYSTEMATIC CODE	84
4.2.1.	Complexity of the Constrained Regulator Trellis	84
4.2.2.	Construction of the Trellis	90
4.3	DECODING ALGORITHM FOR SYSTEMATIC BINARY CODES	91
4.4	SIMPLIFICATION OF THE CONSTRAINED TRELLIS	92
4.4.1.	Partition of the Circuit Memory	93
4.4.2.	Structure of the Constrained Trellis	95
4.4.3.	The Simplified Trellis	98

4.5	CHARACTERISTICS OF THE GENERAL CONSTRAINED TRELLIS	102
4.5.1.	General Partition of the Memory of a Normal LSC	102
4.5.2.	Description of the Current State	104
4.5.3.	Transition from a Given State	105
4.5.4.	Summary of Results	107
4.6	CHARACTERISTICS OF THE GENERAL SIMPLIFIED TRELLIS	108
4.6.1.	Introduction	108
4.6.2.	Preparation	110
4.6.3.	Intermediate Results	113
4.6.4.	Conclusions	116
4.7	CONSTRAINED & SIMPLIFIED TRELLIS FOR SPECIAL CASES	121
4.7.1.	Normal LSC with $t=1$	122
4.7.2.	Normal LSC with $t=2$	126
4.7.3.	Equal-Length Shift Registers	128
4.7.4.	$(n,1,m)$ Normal LSCs	130
4.8	CONCLUSIONS	132
5	<u>THRESHOLD DECODING</u>	134
5.1	THE THRESHOLD-DECODING PROBLEM	134
5.2	THE DECODING ALGORITHMS	135
5.2.1.	The Non-Binary Case	136
5.2.2.	The Binary Case	136
5.3	GENERAL ASPECTS OF CODES FOR THRESHOLD DECODING	137
5.3.1.	Introduction - Terminology	138
5.3.2.	Decoding Modes	139
5.3.3.	Definite Decoding - Parity Squares	140
5.3.4.	Feedback Decoding - Parity Triangles	141
5.4	DISTANCE PROPERTIES OF CODES FOR THRESHOLD DECODING	143
5.5	CONVOLUTIONAL SELF-ORTHOGONAL CODES (CSOCs) ..	144
5.5.1.	Definite Decoding of CSOCs	144
5.5.2.	Structure of Parity Triangles for a CSOC	144
5.5.3.	Effective Constraint-Length of a CSOC	145
5.5.4.	Error Propagation in CSOCs	145
5.5.5.	Distance Properties of CSOCs	147
5.6	CONCLUSIONS	148
6	<u>OPTIMUM THRESHOLD FOR MAJORITY DECODING OF CSOCs</u> ..	150
6.1	PERFORMANCE OF MAJORITY-LOGIC DECODING	151
6.1.1.	Introduction to the Optimum Threshold	151
6.1.2.	Exact and Approximate Value of P_1	152
6.1.3.	Calculation of $P(\Sigma=p e_n=0)$	154
6.1.4.	General Expressions for P_d	156
6.2	OPTIMUM THRESHOLD FOR DEFINITE DECODING	159
6.2.1.	Study of $P(\Sigma=p e_n=0)$	160
6.2.2.	Optimum Threshold	161
6.2.3.	Probability of Decoding Error	164
6.3	OPTIMUM THRESHOLD FOR FEEDBACK DECODING	166
6.3.1.	The p th Generalized Mean	167
6.3.2.	Optimum Threshold	169

6.3.3.	Bounds & Approximation for the Optimum Threshold	171
6.3.4.	Probability of Decoding Error	175
6.4	CONCLUSIONS	177
7	<u>STRUCTURE OF 'CYCLIC' CSOCs</u>	179
7.1	AN ALTERNATIVE REPRESENTATION OF CONVOLUTIONAL CODES	180
7.1.1.	A Discussion on the Design Approach ..	180
7.1.2.	Introduction to Type-A Codes	182
7.1.3.	Introduction to Decoding of Type-A Codes	183
7.1.4.	Orthogonality Conditions for Type-A Codes	184
7.2	TYPE-B CODES - THE USE OF NUMBER THEORY	185
7.2.1.	Restriction on the Value of m for Type-B Codes	186
7.2.2.	Orthogonality Conditions for Type-B Codes	186
7.2.3.	A Discussion on the Design Approach ..	187
7.2.4.	Some Properties of Type-B Codes	191
7.3	TYPE-B ₁ CODES - A CLASS OF RATE-1/2 CSOCs ..	193
7.4	OTHER CLASSES OF TYPE-B SELF-ORTHOGONAL CODES	197
7.4.1.	Type-B Self-Orthogonal Codes with $m < p-1$ & $R \neq 1/2$	198
7.4.2.	Type-B Self-Orthogonal Codes with $p-2 < m < k-1$	204
7.5	$(n, k, k-1)$ TYPE-B SELF-ORTHOGONAL CODES	207
7.6	TYPE-C CODES: CYCLICALLY-DECODABLE TYPE-B CODES	209
7.6.1.	Cyclically-Decodable SO Type-B Codes ..	211
7.6.2.	Cyclically-Decodable Type-B _j Codes $/j=1,2,3,4$	213
7.6.3.	Cyclic Decoding of Type-B ₄ Codes	213
7.6.4.	Type-C ₅ Codes	216
7.7	PROPERTIES OF THE INITIAL ARRAY	221
7.8	EFFECTIVE CONSTRAINT-LENGTH	223
7.9	EXISTENCE THEOREMS FOR 'CYCLIC' CSOCs	226
7.10	CONCLUSIONS	227
8	<u>COMPUTER SIMULATION OF 'CYCLIC' CSOCs</u>	231
8.1	COMPUTER GENERATION OF 'CYCLIC' CSOCs	232
8.2	CHANNEL SIMULATION & S/W IMPLEMENTATION OF THE DECODER	232
8.3	PERFORMANCE DATA	235
8.3.1.	Estimation of G , P_e & P_d	235
8.3.2.	Confidence Intervals	236
8.3.3.	Presentation of Results	236
8.4	CODING GAIN AND OTHER PERFORMANCE CRITERIA UNDER FD	238
8.5	OPTIMUM THRESHOLD UNDER FEEDBACK DECODING	251
8.6	ERROR PROPAGATION	254
8.7	UNEQUAL ERROR-PROTECTION	263
8.8	CONCLUSIONS	266
	<u>CONCLUSIONS & FURTHER WORK</u>	270

Volume 2

APPENDICES	280
A1.1 THE FUNCTIONAL BLOCK-UNITS OF A DIGITAL COMMUNICATIONS SYSTEM	281
A1.2 BINARY PSK WITH COHERENT DEMODULATION	285
A1.2.1. PSK Modulation	285
A1.2.2. The Output of a PSK Coherent Demodulator	286
A1.2.3. Statistical Properties of n_c	287
A1.2.4. Hard-Decision Demodulation	290
A1.2.5. Probability of Error	290
A1.2.6. Bounds on, and Approximation to, P_e	291
A1.3 AVERAGE ERROR-RATE FOR A SIMPLE CHANNEL WITH MEMORY	294
A1.4 ASYMPTOTIC CODING GAIN FOR A BLOCK CODE	294
A2.1 INTRODUCTION TO ABSTRACT ALGEBRA	296
A2.2 INTRODUCTION TO LINEAR ALGEBRA	299
A2.3 PROOF OF THE THEOREMS IN APPENDIX 2.2	304
A2.4 THE POLYNOMIAL & MATRIX APPROACHES TO CONVOLUTIONAL-CODE THEORY	307
A2.5 DISTANCE MEASURES FOR CONVOLUTIONAL CODES	309
A2.6 PROOF OF RELATION (2.24)	311
A2.7 PROOF OF THEOREM 2.3	312
A2.8 PROOF OF RELATION (2.36)	313
A2.9 EXAMPLES OF NORMAL-ENCODER CONSTRUCTION	314
A2.10 CATASTROPHIC CODES	317
A2.11 COMPOSITE GENERATOR-POLYNOMIALS	320
A2.12 PROOF OF THEOREM 2.5	323
A2.13 COMPOSITE GENERATOR-POLYNOMIALS FOR SYSTEMATIC CONVOLUTIONAL CODES	323
A2.14 EXAMPLE OF A TYPE-II ENCODER	324
A2.15 PROOF OF THEOREM 2.10	325
A2.16 PROOF OF THEOREM 2.12	326
A2.17 PROOF OF THEOREM 2.15	328
A3.1 SEQUENTIAL MACHINES & STATE TRANSITIONS	329
A3.2 PROOF OF THEOREMS 3.1, 3.2 & 3.3	333
A3.2.1. Proof of Theorem 3.1	333
A3.2.2. Proof of Theorem 3.2	334
A3.2.3. Proof of Theorem 3.3	335
A3.3 EXAMPLE OF TRELLIS DIAGRAM	335
A3.4 EXAMPLE OF VITERBI DECODING	337
A3.5 SEQUENTIAL DECODING	339
A3.6 TABLE LOOK-UP DECODING	341
A4.1 PROOF OF THE THEORY IN SECTION 4.1	342
A4.1.1. Proof of Lemma 4.1	342
A4.1.2. Proof of Theorem 4.1	342
A4.1.3. Proof of Theorem 4.2	343
A4.1.4. Proof of Theorem 4.3	344
A4.1.5. Proof of Lemma 4.2	345

A4.1.6.	Proof of Theorem 4.4	345
A4.1.7.	Proof of Theorem 4.5	347
A4.1.8.	Proof of Theorem 4.6	348
A4.1.9.	Proof of Theorem 4.7	349
A4.2	SET THEORY AND PARTITIONS	350
A4.3	PROOF OF THEOREMS 4.9 & 4.10	353
A4.3.1.	Proof of Theorem 4.9	353
A4.3.2.	Proof of Theorem 4.10	354
A4.4	PROOF OF THEOREM 4.11	355
A4.5	EXAMPLE OF CONSTRAINED REGULATOR TRELLIS	356
A4.6	EXAMPLE OF ERROR-TRELLIS SYNDROME DECODING	361
A4.7	PROOF OF THE THEORY IN PARAGRAPH 4.4.1.	363
A4.7.1.	Proof of Theorem 4.12	363
A4.7.2.	Proof of Theorem 4.13	364
A4.7.3.	Proof of Theorem 4.14	365
A4.7.4.	Proof of Lemma 4.10	366
A4.8	PROOF OF THEOREM 4.15	366
A4.9	THE INTERMEDIATE RESULTS OF § 4.6.3.	368
A4.9.1.	Proof of Theorem 4.25	368
A4.9.2.	Proof of Lemma 4.13	368
A4.9.3.	Proof of Lemma 4.14	369
A4.10	CONSTRAINED & SIMPLIFIED STATE-TRANSITION DIAGRAMS FOR A $t=2$ NORMAL LSC	369
A4.11	PROOF OF THEOREMS 4.30 & 4.31	374
A4.11.1.	Proof of Theorem 4.30	374
A4.11.2.	Proof of Theorem 4.31	375
A5.1	PROOF OF THE THRESHOLD-DECODING THEOREMS	378
A5.1.1.	Proof of Theorem 5.1	378
A5.1.2.	Proof of Theorem 5.2	378
A5.1.3.	Proof of Theorem 5.3	379
A5.1.4.	Proof of Theorem 5.4	380
A5.2	DEFINITE DECODING - PARITY SQUARES	381
A5.3	FEEDBACK DECODING - PARITY TRIANGLES	389
A5.4	PROOF OF THE THEORY IN SECTION 5.3	397
A5.4.1.	Preliminary Results	397
A5.4.2.	Proof of Theorem 5.7	398
A5.4.3.	Proof of Theorem 5.8	400
A5.5	PROOF OF THEOREM 5.9	400
A5.6	PARITY-TRIANGLES & PARITY-SQUARES FOR CSOCs	402
A5.7	BLOCK EFFECTIVE CONSTRAINT-LENGTH FOR A CSOC	405
A5.8	DISTANCE PROPERTIES OF CSOCs	409
A5.8.1.	Proof of Theorem 5.11	409
A5.8.2.	Proof of Theorem 5.12	410
A5.8.3.	Proof of Theorem 5.13	411
A6.1	PROOF OF THE THEORY IN SECTION 6.1	412
A6.1.1.	Proof of Lemma 6.1	412
A6.1.2.	Proof of Theorem 6.1	412
A6.1.3.	Proof of Theorem 6.2	413
A6.1.4.	Approximation to $(1-2p)^c$	414
A6.1.5.	Examples of the Calculation of $P(\Sigma=u e_0=0)$	416
A6.2	CAPACITY OF THE BINARY SYMMETRIC CHANNEL	418
A6.3	STUDY OF $H(p,1)/H(p,c)$	421
A6.4	PROOF OF RELATION (6.38c)	426
A6.5	GENERALIZED MEANS	428
A6.5.1.	Proof of Theorem 6.9	428
A6.5.2.	Proof of Theorem 6.10	428

A6.6	OPTIMUM THRESHOLD FOR FEEDBACK DECODING	431
A7.1	INTRODUCTION TO ARITHMETICAL FUNCTIONS	435
A7.2	INTRODUCTION TO CONGRUENCES	440
A7.3	INTRODUCTION TO PRIMITIVE ROOTS	444
A7.4	PROOF OF THEOREM 7.2	447
A7.5	THE MINIMUM VALUE OF m FOR TYPE-B CODES	448
	A7.5.1. Proof of Theorem 7.3	448
	A7.5.2. Proof of Theorem 7.4	449
A7.6	ORTHOGONALITY CONDITIONS FOR TYPE-B CODES ...	450
	A7.6.1. Proof of Lemma 7.1	450
	A7.6.2. Proof of Theorem 7.5	450
A7.7	PROPERTIES OF TYPE-B CODES	451
	A7.7.1. Proof of Theorem 7.6	451
	A7.7.2. Proof of Theorem 7.7	452
	A7.7.3. Proof of Theorem 7.8	452
	A7.7.4. Proof of Theorem 7.9	453
A7.8	TYPE-B ₁ CODES	454
	A7.8.1. Examples	454
	A7.8.2. Table of Type-B ₁ Codes	456
A7.9	OTHER CLASSES OF TYPE-B SELF-ORTHOGONAL CODES	458
	A7.9.1. Proof of Theorem 7.13	458
	A7.9.2. Proof of Theorem 7.14	459
	A7.9.3. Proof of Theorem 7.16	460
	A7.9.4. Proof of Theorem 7.18	461
	A7.9.5. Proof of Theorem 7.19	463
	A7.9.6. Proof of Theorem 7.20	464
A7.10	$(n, k, k-1)$ TYPE-B SELF-ORTHOGONAL CODES	466
	A7.10.1. Proof of Theorem 7.21	466
	A7.10.2. Proof of Theorem 7.22	468
	A7.10.3. Proof of Theorem 7.23	476
A7.11	GENERAL PROPERTIES OF TYPE-C CODES	478
	A7.11.1. Proof of Theorem 7.24	478
	A7.11.2. Proof of Relations (7.27)	479
	A7.11.3. Proof of Theorem 7.25	479
	A7.11.4. Proof of Lemma 7.3	483
A7.12	CYCLICALLY-DECODABLE TYPE-B _j CODES	484
	A7.12.1. Proof of Theorem 7.27	484
	A7.12.2. Proof of Theorem 7.31	486
	A7.12.3. Proof of Theorem 7.32	489
	A7.12.4. Examples of Type-C ₅ Codes	491
A7.13	INTRODUCTION TO QUADRATIC RESIDUES	496
A7.14	PROPERTIES OF THE INITIAL ARRAY	497
	A7.14.1. Proof of Theorem 7.33	497
	A7.14.2. Proof of Theorem 7.34	498
	A7.14.3. Proof of Theorem 7.35	499
	A7.14.4. Proof of Theorem 7.36	500
A7.15	EFFECTIVE CONSTRAINT-LENGTH	504
	A7.15.1. Proof of Theorem 7.39	504
	A7.15.2. Proof of Theorem 7.40	505
A7.16	PROOF OF THEOREM 7.41	508
A8.1	COMPUTER GENERATION OF 'CYCLIC' CSOCs	510
	A8.1.1. Greatest Common Divisor	510
	A8.1.2. Sum Modulo m	510
	A8.1.3. Product Modulo m	511
	A8.1.4. Power Modulo m	512
	A8.1.5. Prime Decomposition	513
	A8.1.6. Primitive Root Modulo m	514

	A8.1.7. Order J Modulo any Divisor $d > 1$ of m	515
	A8.1.8. Encoding and Syndrome Arrays	516
	A8.1.9. Effective Constraint-Length	519
A8.2	FORTTRAN PROGRAMMES FOR APPENDIX 8.1	520
	A8.2.1. Greatest Common Divisor	520
	A8.2.2. Sum Modulo m	520
	A8.2.3. Product Modulo m	520
	A8.2.4. Power Modulo m	521
	A8.2.5. Prime Decomposition	522
	A8.2.6. Primitive Root Modulo m	524
	A8.2.7. Order J Modulo any Divisor $d > 1$ of m	525
	A8.2.8. Encoding and Syndrome Arrays	525
	A8.2.9. Effective Constraint-Length	533
A8.3	CHANNEL AND DECODER SIMULATION	533
A8.4	SIMULATION PROGRAMMES	541
	A8.4.1. Software Implementation of the Decoder	541
	A8.4.2. A Complete Simulation Programme for Long Codes	543
A8.5	SUBROUTINES USED BY THE MAIN PROGRAMME	556
A8.6	FORTTRAN PROGRAMMES FOR APPENDIX 8.5	559
	A8.6.1. Channel Capacity, Channel Error-Rate & Probability of Decoding Error under DD	559
	A8.6.2. Probability of First Decoding Error under FD	561
A8.7	SELECTION OF CODES WITH GIVEN PARAMETERS	564
	A8.7.1. Codes with Given Information Block-Length, k	564
	A8.7.2. Codes with Given Rate, $c/(c+1)$	564
	A8.7.3. Codes with Given Number of Orthogonal Checks, J	565
A8.8	FORTTRAN PROGRAMMES FOR APPENDIX 8.7	566
	A8.8.1. Codes with Given Information Block-Length, k	566
	A8.8.2. Codes with Given Rate, $c/(c+1)$	567
	A8.8.3. Codes with Given Number of Orthogonal Checks, J	568
A8.9	CONFIDENCE INTERVALS	569
A8.10	GRAPHS OF EER & NET CODING-GAIN vs Γ	571
A8.11	ERROR PROPAGATION	578
A8.12	UNEQUAL ERROR-PROTECTION	584
	REFERENCES	586



Acknowledgements

I should like to thank *Dr Mike E. Woodward*, my supervisor, for his valuable contribution to the thesis, for the time he spent discussing the various problems, for his patience and especially for his friendly approach.

My sincere thanks to *Professor G.W.R. Griffiths*, my Director of Research, for his generous assistance.

I should also like to express my gratitude to the late *Professor A.C. Clark* for his very valuable advice during a difficult period.



Abbreviations

APP = a posteriori probability (see p. 136)
AWGN = additive white Gaussian noise (see p. 287)
BSC = binary symmetric channel (see p. 6)
CC = convolutional code (see p. 18)
CEG = central group (see p. 58)
CSOC = convolutional self-orthogonal code (see p. 138)
c/w = codeword
DD = definite decoding (see p. 139)
DIG = discarded input group (see p. 87)
DMC = discrete memoryless channel (see p. 5)
EA = encoding array (see p. 220)
eqn = equation
FD = feedback decoding (see p. 139)
FEG = front-end group (see p. 58)
FF = feed-forward (see p. 317)
Fig. = Figure
gcd = greatest common divisor (see p. 435)
IA = initial array (see p. 183)
iff = if, and only if
ING = input group (see p. 87)
I/P = input
LHS = left-hand side
LSB = least significant bit
LSC = linear sequential circuit
MEG = memory group (see p. 58)
MIG = memory input group (see p. 87)
MLD = maximum likelihood decoding (see p. 10)
MSB = most significant bit
MUX = multiplexing (see p. 283)
O/P = output
PCM = pulse code modulation
PSK = phase-shift keying
REG = rear-end group (see p. 58)
reln = relation
RHS = right-hand side
SA = syndrome array (see p. 517)
SNR = signal-to-noise ratio
SO = self orthogonal (see p. 138)
SR = shift register
SYRE = syndrome register (see p. 210)
X-OR = exclusive-or
wrt = with respect to



Notation

\mathcal{E}	= = =	number of states (Chapter 4) (see p. 89)
\mathcal{E}	= = =	composite parity-check (Chapter 6) (see p. 135)
β	= = =	IA generating element (Chapters 7 & 8) (see p. 218)
$C(\tau)$	= =	autocovariance function (see p. 257)
$C(n, k)$	$\hat{=}$	$n!/[k!(n-k)!]$ = binomial coefficient
Γ	= = =	signal-to-noise ratio per information-bit
D	= = =	delay operator
d_{\min}	= =	minimum distance of a code
E	= = =	energy per received bit (see p. 4)
$E[?]$	= =	expected value of ?
e	= = =	channel-error sequence (see p. 46)
erfc	= =	complementary error function (see p. 291)
f	= = =	coherent demodulator O/P (Chapter 1) (see p. 286)
f	= = =	number of zero-length SRs (Chapters 3 & 4) (see p. 57)
$f(i)$	= =	memory-density function (Chapter 4) (see p. 102)
$F(i)$	= =	memory-distribution function (Chapter 4) (see p. 114)
Φ	= = =	Euler totient (Chapters 7 & 8) (see p. 436)
G	= = =	net coding-gain (see p. 13)
G	= = =	generator matrix (see p. 30)
$GF(q)$	=	Galois field q (see p. 297)
H	= = =	parity-check matrix (see p. 45)
I	= = =	identity matrix
J	= = =	number of orthogonal check-sums
M	= = =	total circuit memory (see p. 55)
m	= = =	memory order (see p. 19)
M_i	= = =	length of the i th SR of a normal LSC (see p. 33)
\tilde{n}	= = =	single-sided noise power spectral density
n_A	= = =	actual constraint-length (see p. 20)
n_E	= = =	effective constraint-length (see p. 145)
Q	= = =	number of input blocks (Chapter 4) (see p. 89)
$P_d, P_{d'}$	=	probability of bit decoding error
P_e	= = =	probability of channel error
r	= = =	received sequence (see p. 46)
$R(\tau)$	= =	autocorrelation function (see p. 257)
s	= = =	syndrome sequence (see p. 47)
T	= = =	syndrome threshold (see p. 151)
t	= = =	error-correcting capability (see p. 85)
T_o	= = =	optimum threshold (see p. 151)
u	= = =	message (or information) sequence (see p. 25)
v	= = =	channel sequence (see p. 25)

$w[?]$ = = Hamming weight of ?
 Ψ = = = (see p. 89)
 θ = = = theta function (see p. 218)
 $\lfloor x \rfloor$ = = greatest integer $\leq x$
 $\lceil x \rceil$ = = smallest integer $\geq x$
 \equiv = = = congruence symbol (see p. 441)
 $\hat{=}$ = = = equal by definition
 $\langle A, B \rangle$ = = partitioned by sets A & B (see p. 352)
 $A \subseteq B$ = = A is a subset of B
 $A \subset B$ = = A is a proper subset of B
 (a, b) = greatest common divisor of a & b (see p. 435)
 $x/y/z$ = $(x/y)/z = x/(yz)$



Synopsis

The thesis investigates various aspects of convolutional code (CC) theory, design and decoding.

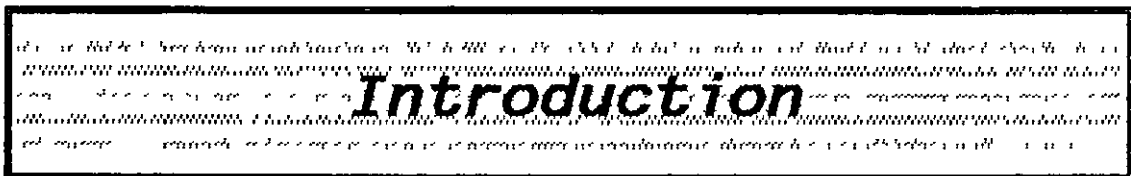
A relatively novel approach to *CC theory* has been developed. It is based on the concept of convolution and the properties of the encoder (a linear sequential circuit). Some new concepts & results were obtained.

The *complexity of the trellis diagram* of a 'normal' CC encoder is related to code parameters. The associated '*constrained trellis*' is obtained by deleting states & transitions, according to a certain criterion, while its complexity is linked to code parameters. Finally, the '*simplified trellis*' evolves from the constrained one by permitting transitions of length more than one time unit. The existence, number and state of origin of these long transitions is again related to code parameters, both for the general and for special cases; a decoding algorithm using the '*simplified trellis*' is finally developed.

The algorithm for the *majority-logic decoding* of systematic CCs is optimized via the introduction of a syndrome threshold which depends on the channel probability of error, as well as on code parameters. Analytical expressions for the *optimum threshold* have been developed.

A systematic search for the generation of *systematic self-orthogonal* CCs, with given properties, resulted in some *new classes* of such codes.

A family of codes, discovered by D McQuilton, is examined both theoretically & experimentally. Some new results concerning the *codes' structure*, have been obtained. A great number of these codes were tested on a *computer-simulated* binary symmetric channel. Graphs of the net coding-gain vs SNR per Information-bit have been produced. From the results, conclusions were obtained about the codes' *error-propagation* performance, the associated power-loss and the *correlation of the decoding errors*. The codes were also tested with the optimum threshold to determine the power gains obtained. Finally, some results on the codes' *unequal error-protection* properties were related to their parameters.



Chapter 2 ("Structure of Convolutional Codes") introduces the relevant terminology and builds the theory of the codes, starting from the concept of convolution between the input to, and the impulse response of, the encoder. Topics discussed include generator sequences, generator matrix, generator polynomials, composite generator polynomials, catastrophic codes, distance measures, systematic codes, parity-check matrix & syndrome. Both approaches ("matrix" & "poly-

* Although this limit was exceeded, by 10.8%.

nomial") to the theory of convolutional codes (CCs) are covered.

The original material in this chapter concentrates on the characteristics of the normal & the type-II encoders and on the code matrices. With respect to the latter, a matrix equation is developed which relates an arbitrary, but finite, portion of the encoder's output to the corresponding input, via a finite-dimensioned 'system' matrix. The latter is shown to be a generalization of the generator matrix. Similar results have been obtained for the parity-check matrix.

The aim of this chapter is to build the necessary CC infrastructure for the development of the rest of the thesis.

Chapter 3 ("Decoding of Convolutional Codes") classifies and explains the various decoding techniques, but concentrates on one aspect of one of them, namely the state-transition diagram of a normal convolutional encoder (a structure used in trellis decoding). This diagram is analysed and results are obtained which link its complexity to the encoder parameters. A very powerful tool, used to that end, is the partition of the encoder memory in groups, which play a specific role during a state transition.

The aim of Chapter 3 is both to complete the general introduction of CCs and to build an infrastructure for the next chapter.

Chapter 4 ("Error-Trellis Syndrome Decoding") retraces the decoding technique of the title (in the first two sections) and develops the algebra of the generalized trellis (in the remaining five sections). Reed & Truong [24] introduced the idea of a state diagram with a Hamming-weight constraint, t , on the input and contents of the encoder. The first section develops the general solution of the syndrome eqn $[s = f(e)]$, while the third section introduces the decoding algorithm; the material generalizes, adds to and corrects the work by Reed & Truong. The rest of the sections (especially Sections 4.4-7) contain original work which concentrates on three issues: 1. The constrained trellis is analysed and results linking its structure to the circuit's

parameters are obtained. 2. The simplified trellis, obtained from the constrained-one by removing all single-input transitions, contains transitions of length greater than one time-unit. Their length, number and existence conditions are related to the circuit's parameters. 3. The simplified trellis is used to improve the decoding algorithm.

The aim of Chapter 4 is the development of the algebra linking the parameters of a 'normal' linear sequential circuit to the complexity of its generalized trellis.

Chapter 5 ("Threshold Decoding") introduces the threshold decoding algorithms (developed by Massey [18]), defines the relevant concepts and discusses the properties of codes suitable for threshold decoding. Some original work is included in the discussion of the structure of the code system-matrix, both for definite and for feedback decoding. The chapter concludes with some results on convolutional self-orthogonal codes (CSOCs), including a new term, called block effective constraint-length.

This chapter forms the backbone of the rest of the thesis, which concentrates on the design and performance of CSOCs.

Chapter 6 ("Optimum Threshold for Majority Decoding of CSOCs") represents original work, which concentrates on the improvement of Massey's majority-decoding algorithm. In particular, the threshold used in majority decoding was set at $T = \lfloor J/2 \rfloor$ (where J is the minimum number of syndromes checking on any error bit). That setting guarantees correct decoding if no more than $\lfloor J/2 \rfloor$ errors have occurred among the error bits checked by the J syndromes. It is obvious, though, that as the channel error rate increases, correct decoding becomes less and less frequent.

An expression between the probability, P_d , of a bit decoding-error and the syndrome setting, T , is developed and the value of T which minimizes P_d is obtained. The case of constant-size syndromes (usually the case of definite decoding) results in a closed-form expression for the optimum threshold. The case of feedback decoding results in a recurrent equation and in a set of approximate closed-form ex-

pressions, as well as in upper & lower bounds.

The aim of Chapter 6 is the determination of that syndrome threshold which, for majority decoding, minimizes the probability of a decoding error.

Chapter 7 ("Structure of 'Cyclic' CSOCs") introduces & generalizes the above class of codes (discovered by McQuilton [42]), studies their properties and elaborates on the code-design technique. In particular, the problem of systematically constructing systematic CSOCs is tackled via the discovery of the necessary and sufficient conditions so that a general systematic CC is self-orthogonal (SO). A general solution is developed until either a class of SO codes is obtained, or a simplification becomes necessary. In this way, new classes of codes are discovered (which include McQuilton's codes), while the way for the discovery of other classes of codes has been left open. The basic difference between McQuilton's approach and that of the author is that the latter concentrates on the equivalent conditions for the existence of CSOCs with certain properties. Of necessity, the results obtained include McQuilton's work, as well as a definite opinion about the existence, or not, of other codes.

The last three sections contain some new results on the properties of McQuilton's codes.

Chapter 7 aims to analyse the structure, and investigate the relatives, of the class of 'cyclic' CSOCs, whose performance will be studied in the next chapter.

Chapter 8 ("Computer Simulation of 'Cyclic' CSOCs") presents & analyses the simulation results. The main computer-simulation programme used requires a minimal set of input data in order to generate a number of points of the "net coding-gain" vs "signal-to-noise power-ratio per information-bit" graph. As a result, the main programme requires the support of about twenty general-purpose subroutines (built and tested by the author) whose flow-charts, background theory and FORTRAN listings are given in appendices. Also, the very long actual constraint-length (≈ 1.7 Mbytes) of some of the codes required a special software implementa-

tion technique (for the decoder) that made use of bit-manipulation commands.

For the class of CSOCs, experimental (simulation) results were obtained on the net coding-gain of the codes, their performance using the optimum threshold, their 'behaviour' under error propagation and on their unequal error-protection properties.

The aim of Chapter 8 is to describe the computer-simulation techniques used and to explain the results of the simulation, using the previously developed theory.

A formal mathematical structure is used to develop this thesis, following the general-to-particular approach. Five main areas have been researched: 1. The algebra of the generalized trellis. 2. The optimum threshold for majority decoding. 3. Construction of systematic convolutional self-orthogonal codes. 4. Simulation of the class of 'cyclic' CSOCs (discovered by McQuilton [42]). 5. Simulation techniques, development of a library of number-theoretic subroutines and computer-selection/generation of 'cyclic' CSOCs.

All results are proved mathematically, except for a small number of general and well-known theorems (for which a reference is given). Examples are used to verify the 'predictions' of the theory, while the simulation results contribute to a deeper understanding of the 'behaviour' of the codes.

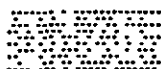
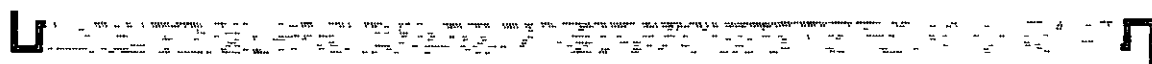
The aim of the thesis is to extend our knowledge about, and deepen our understanding of convolutional codes.

The thesis is organized into two volumes. Volume 1 contains the main body (Chapters 1-8 and Conclusions), while Volume 2 contains the Appendices and the References.

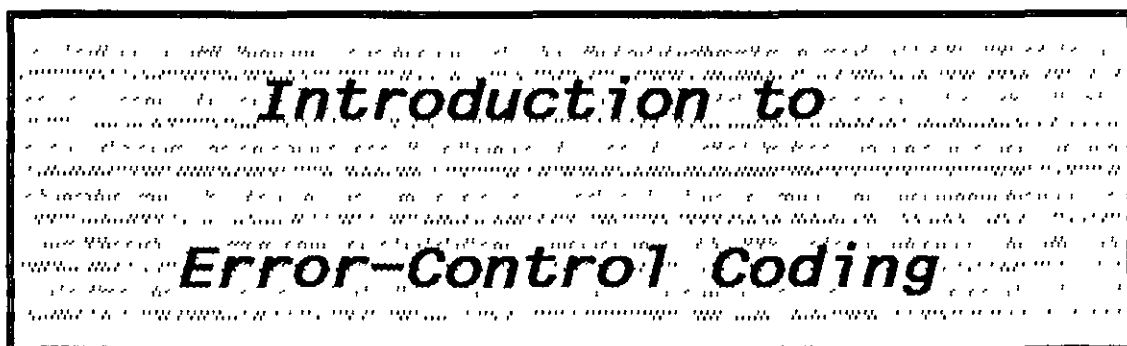
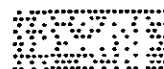
Nikos P. Frydas

Leicester, 1 January 1990

V O L U M E 1



CHAPTER 1



Chapter 1 will act as an introduction to the thesis. The functional block diagram of a typical digital communications system will serve as the starting point. This diagram will be reduced to one including only what the error-control encoder & decoder 'see' as their channel, source and destination. Based on this abstraction, channel models will be introduced, the general concept of error-control coding will be briefly discussed and some performance criteria will be considered.

In 1948, Claude Shannon proved that if the information signal is appropriately encoded, the rate of errors injected by a noisy channel can be reduced to any desired level without sacrificing the rate of information transmission or storage. What Shannon did not tell us though is how this objective will be achieved. The theory of error-control coding is concerned with this problem and has gone a long way, since 1948. Error-control coding forms today a 'stand-alone' branch of communications and is supported by a long list of error-control textbooks and journals (see, for example, the reference list, pp. 586-90).

1.1 BLOCK DIAGRAM OF ERROR-CONTROL SYSTEMS

In this section, a model of an error-control coding system will be gradually developed. The 'effort' starts with a 'complete' functional block diagram of a digital communica-

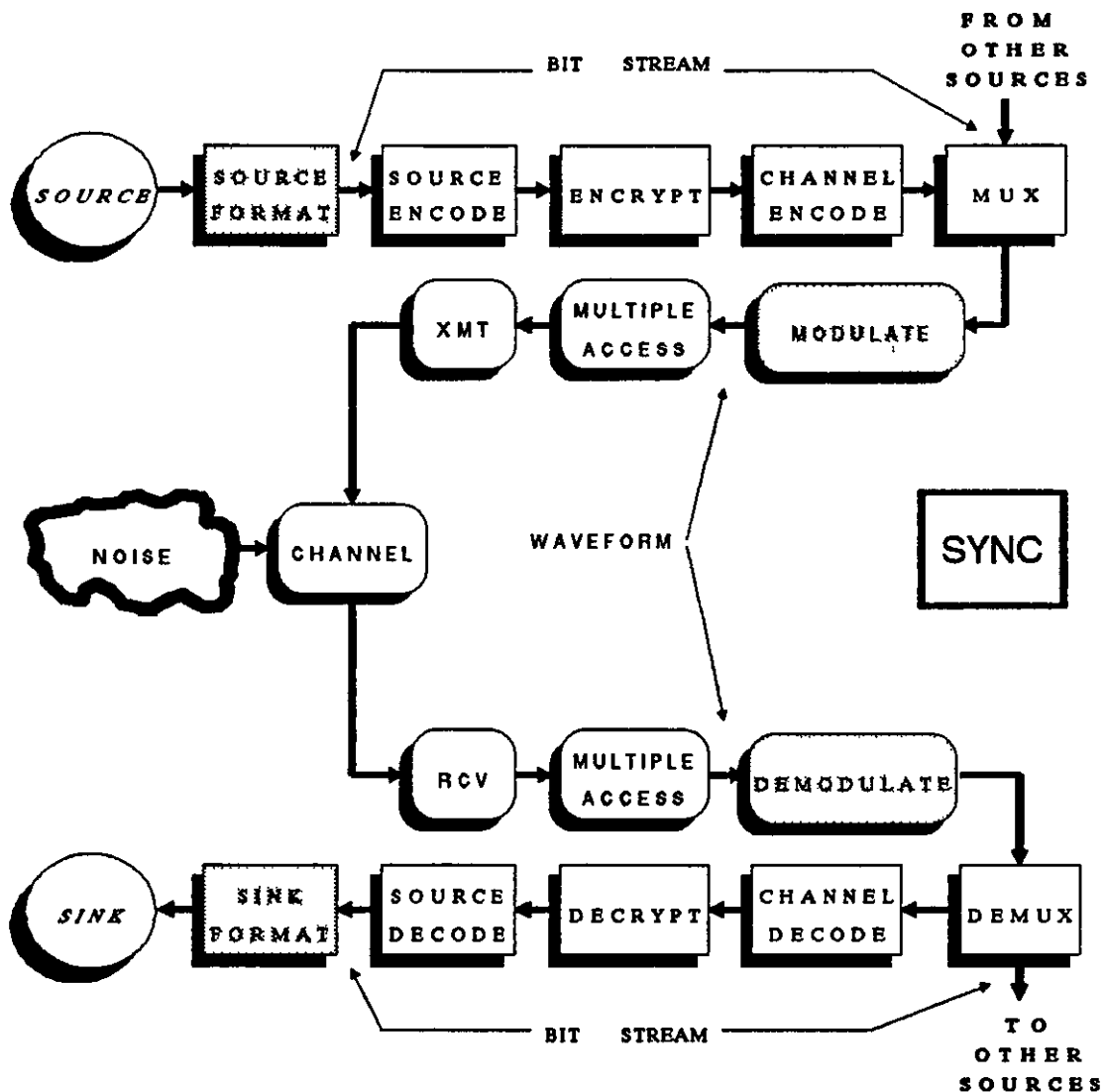


Figure 1.1: Functional block diagram of a typical digital communications system (after Sklar [1]).

tions system (see Fig. 1.1). This is reduced to a diagram with the minimum number of units necessary, to study error-control coding.

1.1.1. The Elements of a Digital Communications System

From a functional point of view, a digital communications system may be divided in a number of blocks, which have to be linked in a certain order (see Fig. 1.1).

Note that the transmission and the storage systems have essentially the same structure. From such a point of view they differ in that the one transmits information from here to there, while the other from now to then. Note also that all blocks appear in pairs (processor-deprocessor) except for synchronization (SYNC) and the channel.

The *formatting* units act as an interface between the communications system and the outside world. The *channel* links the transmitting with the receiving site. The *modulator* and the *demodulator* form the interface between the *bit-stream* and the *waveform* parts of the system. For a brief description of each block, see Appendix 1.1 (p. 281).

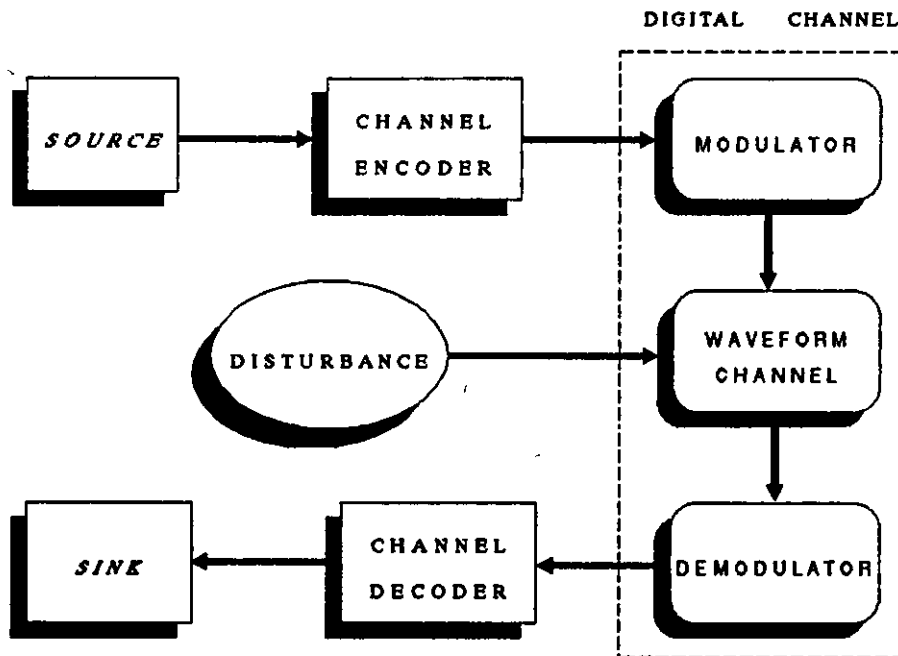


Figure 1.2: Simplified block diagram of an error-control coding system.

1.1.2. The Waveform Channel

The *waveform channel* consists of all the hardware and

physical media between the modulator output (O/P) and the demodulator input (I/P) (see Fig. 1.2). If one compares the waveform at the channel I/P, with the waveform out of the channel, one will find out that the latter is a scaled replica of the former to which some disturbance has been superimposed. This disturbance may be due to any combination of additive random noise, man-made interference, distortion, signal fading (time-varying attenuation), or even intentional jamming.

The most widely considered type of disturbance is the so-called *additive white Gaussian noise* (AWGN). If the modulator O/P is denoted by $s(t)$, the AWGN by $n(t)$ and the demodulator I/P by $r(t)$, then:

$$r(t) = s(t) + n(t) \quad (1.1)$$

In the presence of AWGN, a good modulation method is binary PSK (BPSK), with coherent demodulation [2]. In this case the modulator transmits one of the two waveforms:

$$s_0(t) = \sqrt{(2E/T)} \sin(2\pi f_0 t + \pi/2) \quad 0 \leq t \leq T \quad (1.2a)$$

$$s_1(t) = \sqrt{(2E/T)} \sin(2\pi f_0 t - \pi/2) \quad 0 \leq t \leq T \quad (1.2b)$$

The modulator transmits $s_0(t)$ & $s_1(t)$ in the place of binary 1 and 0, at a rate of $1/T$. f_0 is a multiple of $1/T$ and E is the energy of each signal element (see Appendix 1.2, § A1.2.1., p. 285). An optimum demodulator may include a correlation detector followed by a sampling switch [2] (see Fig. A1.2.1, p. 286). The O/P of the sampling unit is a real number (see § A1.2.2., p. 286):

$$f = \int_0^T r(t) \sqrt{(2E/T)} \sin(2\pi f_0 t + \pi/2) dt = \pm E + n_c \quad (1.3)$$

n_c is a zero-mean Gaussian random variable with variance $\sigma^2 = E\tilde{n}/2$, where $\tilde{n}/2$ is the double-sided noise power spectral density. $f = E + n_c$ if $s_0(t)$ is transmitted and $-E + n_c$ if $s_1(t)$ is transmitted.

If f is processed as an analogue number then the demodulator operates in analogue fashion. A more common approach is for f to be quantized by a q -bit quantizer to produce, thus, one of $2^q = Q$ different O/P symbols.

If hard-decision demodulation is used ($Q=2$), then the bit-error probability, P_e , is given by (see Appendix 1.2, § A1.2.5., p. 290):

$$P_e = \frac{1}{2} \text{erfc}[\sqrt{E/\bar{n}}] \quad (1.4)$$

where $\text{erfc}(x)$ is the *complementary error-function*. Bounds on, and an approximation to P_e , are obtained in Appendix 1.2 (§ A1.2.6., p. 291):

$$[1-1/(2\Gamma)]e^{-\Gamma}/[2\sqrt{(\pi\Gamma)}] < P_e < e^{-\Gamma}/[2\sqrt{(\pi\Gamma)}] \quad (1.5a)$$

$$\Longleftrightarrow -1/(2\Gamma) < 2e^{\Gamma}\sqrt{(\pi\Gamma)}P_e - 1 < 0 \quad (1.5b)$$

$$P_e \approx e^{-\Gamma}/[2\sqrt{(\pi\Gamma)}] \text{ as } \Gamma \rightarrow +\infty \quad (1.5c)$$

$$\text{where, } \Gamma \triangleq E/\bar{n} \quad (1.5d)$$

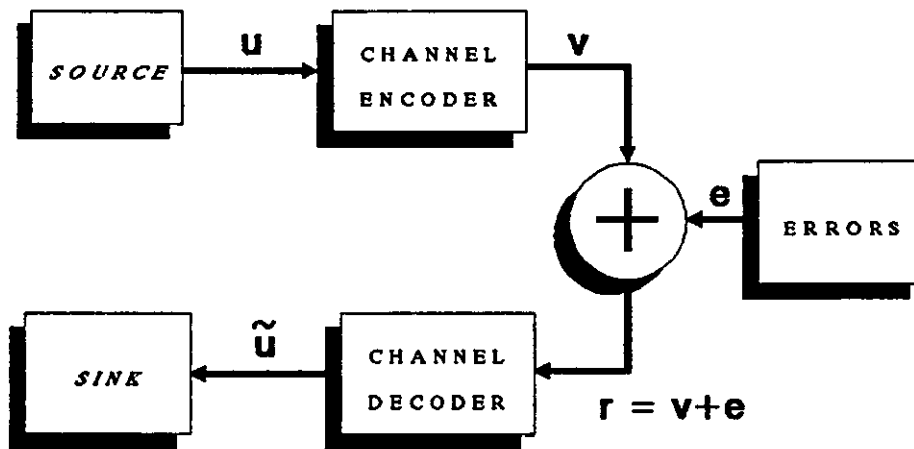
1.1.3. The Digital Channel

Consider the block diagram of Fig. 1.2. Note that from the point of view of the channel-encoder and channel-decoder, the channel between them, called the *digital channel*, is the most important. This channel is composed of the modulator, the waveform channel and the demodulator. If the modulator uses M different waveforms (i.e. if it replaces k given I/P bits by a specific waveform, where $2^k=M$) and the demodulator uses a q -bit quantizer to represent the correlated & sampled outputs f , then the channel is characterized by a set of M I/P symbols and a set of $Q(=2^q)$ O/P symbols; it is then called an M -ary in, Q -ary out, digital (or discrete) channel.

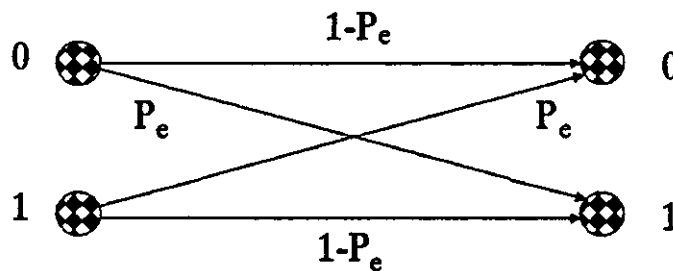
Furthermore, if each channel output depends only on the corresponding channel input symbol, and not on any previous transmissions, the channel is said to have no *memory* and is called a *discrete memoryless channel* (DMC). Such a channel is completely defined by the set of conditional probabilities $P(r_j|s_i)$ / $i=1,2,\dots,M$ & $j=1,2,\dots,Q$, where $P(r_j|s_i)$ denotes the probability that r_j will be received, given that s_i was transmitted.

1.1.4. Discrete Channel Models

The most common channel model used to test various coding schemes is a DMC with binary modulation ($M=2$), symmetric noise-amplitude distribution and two-level quantizer ($Q=2$ - *hard-decision demodulation*). This is called the *binary symmetric channel* (BSC) and it can be realized with a BPSK modulator, an AWGN waveform channel and a coherent hard-decision demodulator.



(a)



(b)

Figure 1.3: a) Channel coding over the BSC; b) transition diagram of the BSC.

Since the noise-amplitude distribution is symmetrical, $P(r_1|s_2) = P(r_2|s_1)$. Since also, $P(r_2|s_1) + P(r_2|s_2) = 1$ and $P(r_1|s_1) + P(r_1|s_2) = 1$, there is only one independent parameter in this channel model. If one lets $P_e = P(r_1|s_2)$, then the BSC

transition-diagram will be as in Fig. 1.3b. The bit error-rate (bit-error probability) is P_e [given by eqn (1.4)].

The BSC is a good test-channel for coding schemes operating over optical fibres*, satellite channels and space channels. Terrestrial links though, produce discrete channels with memory. A simple two-input, two-output, channel with memory would have two states; a 'good' state where the bit error-rate, p_1 , will be very small and a 'bad' state where the bit error-rate, p_2 , will be high ($p_2 \gg p_1$). The channel remains in the 'good' state for most of the time, but on occasion it shifts into the 'bad' state, where it remains for a brief period of time. If q_1 denotes the probability of a shift from the 'good' to the 'bad' state, and q_2 the probability of a shift from the 'bad' to the 'good' state, then $q_1 \ll q_2$. The state-transition diagram of such a channel is shown in Fig. 1.4. See Appendix 1.3 (p. 294), for the calculation of the average error-rate of such a channel.

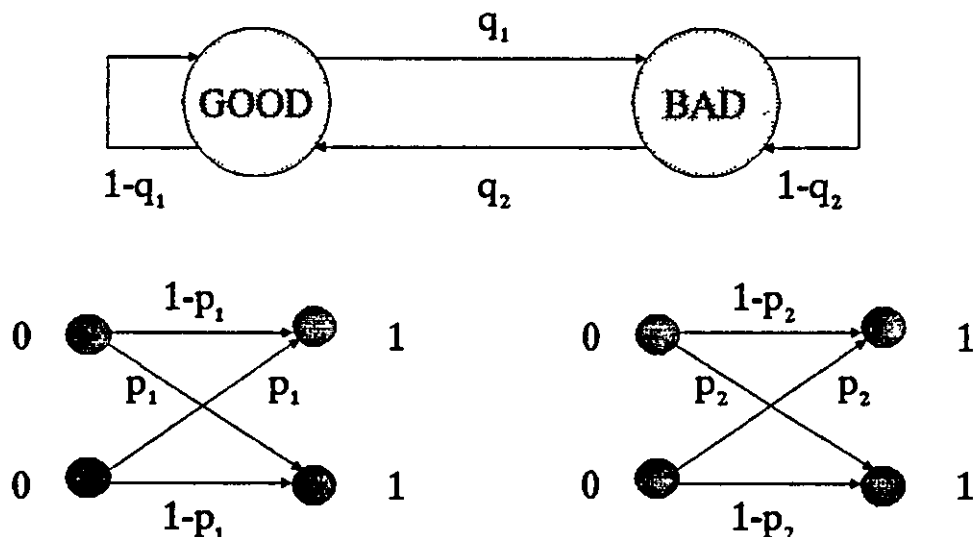


Figure 1.4: State transition-diagram of a binary channel with memory.

1.2 THE INGREDIENTS OF ERROR-CONTROL CODING

Error-control codes are based on diverse mathematical

* Although a direct-detection (non-coherent) optical system acts as a 'z-channel'.

disciplines. Nevertheless, they have two common ingredients: *Redundancy* and *noise averaging*.

1.2.1. Redundancy

Error-control coding, like most signal-processing techniques, operates on blocks rather than on individual bits. The channel encoder 'breaks' the source bit-stream, u , in blocks of k and replaces this with a block of n bits, to form the channel bit-stream, v .

Like in all coding schemes, the mapping should be one-to-one, so that the process is reversible; hence, $n \geq k$. Because of the action of the error sequence, e (see Fig. 1.3a), the transmitted block of n bits is altered so that any n -bit pattern is possible to be received by the decoder. The latter is asked to decide, if the received n -tuple is a legitimate message, or not. Clearly, the only way this can be accomplished is by not permitting all possible received messages to be valid ones. So, $2^n > 2^k \iff n > k$, hence redundancy is a necessary ingredient of error-control schemes.

So, for an (n, k) code, only k out of every n channel bits are message bits. The ratio $R \triangleq k/n$ is called the *code rate*. If the rate of reception of message bits is $1/T$ bps, then the transmission bandwidth, W , is $W = c/T$, where c is a constant that mainly depends on the unit-pulse used. Since the encoder has to transmit at $(n/k)(1/T) = 1/(RT)$ bps, the channel bandwidth increases* by a factor of $1/R$; this factor is called the *bandwidth expansion ratio*.

Consider now the redundancy ingredient in more detail. Assume that one wishes to correct all patterns of t or fewer errors in a block of n bits. This means that even with t errors, an n -tuple should still be identifiable with no other block apart from the original. Hence, all n -tuples should differ in at least $2t+1$ positions. In this way, an n -tuple with t errors will differ from any other n -tuple in at least $t+1$ positions, but it will differ from the original in only t positions, so correct decoding can be accomplished.

Hence, if all pairs of *codewords* (valid channel n -tuples) differ in at least $d_{\min} = 2t+1$ positions, the code can correct

* This is usually the case; an exception is *multilevel/phase modulation combined with a state-oriented trellis coding scheme*.

all patterns of t or fewer errors, in a block of n received bits. t is the *error-correcting capability* of the code. The number of positions any two sequences differ in, is called the *Hamming distance* between them. d_{\min} , mentioned above, is known as the *minimum distance of the code* and is defined to be the minimum Hamming distance between any two codewords.

At this stage, one is able to relate the amount of redundancy in a code to its error-correcting capability, t . Since there are $C(n,i) \triangleq n!/[i!(n-i)!]$ n -tuples with i errors, the decoder is asked to be able to recognize, in a unique way, and in any of the 2^k possible legitimate messages, any of the $1+C(n,1)+C(n,2)+\dots+C(n,t)$ error patterns, in that message; hence, for a t -error correcting code, there exist $[1+C(n,1)+C(n,2)+\dots+C(n,t)]2^k$ error conditions. The decoder is asked to relate, in a unique way, any of these error conditions to a specific n -tuple. Hence,

$$\left[1 + \binom{n}{1} + \binom{n}{2} + \dots + \binom{n}{t}\right]2^k \leq 2^n \quad \langle \text{---} \rangle$$

$$\langle \text{---} \rangle \quad 1 + \binom{n}{1} + \binom{n}{2} + \dots + \binom{n}{t} \leq 2^{n-k} \quad (1.6)$$

1.2.2. Noise Averaging

It has been mentioned already that the channel bit-stream is 'enriched' with $n-k$ redundant bits. These bits depend on a sequence of k message bits. The question that arises naturally is, if it is an advantage to use a large value of n .

Assume that an error-control coding scheme is able to correct all error patterns with βn or less errors, where $0 \leq \beta \leq 1$. If a BSC with probability of error p is used, then the probability that a block of n bits contains i errors is $p^i(1-p)^{n-i}C(n,i)$. The coding scheme will fail, if $\beta n + 1$ or more errors occur. So, the probability of decoder failure is

$$P(E) = \sum_{i=\beta n+1}^n p^i(1-p)^{n-i}C(n,i) \quad (1.7)$$

Consider, for example, an error-control scheme which can correct up to 5% of the block bits ($\beta=0.05$), when the BSC error-probability is $p=10^{-3}$. The probability of a decoder

error, for block-lengths of 20, 40 & 100, is [eqn (1.7)]:

For $\beta=0.05$ & $n=20$: $P(E) = 1.9 \times 10^{-4}$

For $\beta=0.05$ & $n=40$: $P(E) = 9.6 \times 10^{-8}$

For $\beta=0.05$ & $n=100$: $P(E) = 1.1 \times 10^{-9}$

From the example above it is seen that, if a coding scheme can correct up to 5% of the bits of a block of n , then, on average, if $n=20$ one block in 5,327 will be erroneously decoded, if $n=40$ one block in 104,000 will be erroneously decoded and if $n=100$ one block in 909,000,000 will be erroneously decoded. Eqn (1.7) & the above example show that the performance improvement obtained through noise-averaging increases with the block length. Hence, longer codes are expected to be more 'efficient' than shorter codes.

1.3 MAXIMUM LIKELIHOOD DECODING

Consider the block diagram of Fig. 1.3a. The decoder's task is to produce an estimate, \hat{u} , of the message sequence, u , based on the received sequence r . Since there is a one-to-one correspondence between the message sequence, u , and the channel sequence (or codeword), v , the decoder's task is, in effect, to produce an estimate, \hat{v} , of the transmitted codeword, v . A *decoding error* occurs iff v was transmitted and $\hat{v} \neq v$. Then, the probability of a decoding error, given that r was received, is $P(\hat{v} \neq v | r)$, while the average probability of a decoding error, $P(E)$, is,

$$P(E) = \sum_r P(\hat{v} \neq v | r) P(r) \quad (1.8)$$

A decoding rule (i.e. a decoding strategy) which minimizes $P(E)$ is called *maximum likelihood decoding* (MLD). Since r is fixed, minimizing $P(E)$ is equivalent to minimizing $P(\hat{v} \neq v | r)$, for all r , and this is equivalent to maximizing $P(\hat{v} = v | r)$. Hence, for a given r , $P(E)$ is minimized if \hat{v} is chosen to be that codeword, v , which maximizes $P(v | r)$.

By definition of the conditional probability,

$$P(A|B) = P(A,B)/P(B) \quad \longrightarrow$$

$$\longrightarrow \quad P(A,B) = P(A|B)P(B) = P(B|A)P(A) \quad (1.9)$$

Using eqn (1.9), $P(v|r) = P(r|v)P(v)/P(r)$. If all codewords are equally likely, then $P(v)$ is constant, hence the MLD rule is equivalent to choosing that codeword, v , which maximizes $P(r|v)$, for a given r . For a DMC, errors occur independently from each other, hence

$$P(r|v) = \prod_i P(r_i|v_i)$$

Since $\log x$ is a monotonously increasing function of x , a more useful expression is

$$\log P(r|v) = \sum_i \log P(r_i|v_i) \quad (1.10)$$

where i ranges over all bit positions in a block.

Finally, the MLD rule has become:

Theorem 1.1: Maximum likelihood decoding, for the DMC, if all codewords are equally-likely, is equivalent to choosing \hat{v} as this codeword, v , which maximizes the sum in (1.10), where r is the received sequence.

Consider now the application of the MLD rule to the case of a BSC. Let $p(r_i|v_i)=p$ if $r_i \neq v_i$ and $p(r_i|v_i)=1-p$ if $r_i=v_i$. Then, if r and v differ in $d=d(r,v)$ positions [where $d(r,v)$ is the Hamming distance between r & v] d of the $p(r_i|v_i)$ s of the sum in (1.10) will be p and the rest $n-d$ will be equal to $1-p$. Hence,

$$\log P(r|v) = d(r,v)\log p + [n-d(r,v)]\log(1-p) \quad \longrightarrow$$

$$\longrightarrow \log P(r|v) = d(r,v)\log[p/(1-p)] + n\log(1-p) \quad (1.11)$$

Since $n\log(1-p)$ is a constant and because, for $p < 0.5$, $p/(1-p)$ is less than 1, then $\log[p/(1-p)]$ is negative. Hence, the MLD rule for the BSC becomes:

Theorem 1.2: Maximum likelihood decoding, for the BSC, if all codewords are equally-likely, is equivalent to choosing \hat{v} as this codeword, v , which minimizes the Hamming distance $d(r,v)$, where r is the received sequence.

At this stage it is useful to refer to the *noisy channel coding theorem*, as stated in Lin & Costello [2], p. 10:

Theorem 1.3: Every channel has a capacity C and for any rate $R < C$, there exist codes of rate R which, with maximum likelihood decoding, have an arbitrarily low decoding error probability $P(E)$. In particular, for any $R < C$, there exist (n,k) block codes, of length n and with k information bits per block, such that

$$P(E) \leq 2^{-nE_b(R)} \quad (1.12)$$

and there exist (n,k,m) convolutional codes, of memory order m , such that

$$P(E) \leq 2^{-n(m+1)E_c(R)} \quad (1.13)$$

where $E_b(R)$ & $E_c(R)$ are positive functions of R for $R < C$ and are completely specified by the channel characteristics.

Note that the above bounds hold true for the average error probability of the ensemble of all codes. Hence, since some codes are bound to be better than the average code, Theorem 1.3 guarantees the existence of codes meeting these bounds. Furthermore, one can see from these bounds that the way to achieve very low probabilities of decoder failure is via the use of very long codes. But since a maximum likelihood decoder has to examine all possibilities before it makes a decision and since, for an (n,k,m) convolutional code, there are approximately $2^{k(m+1)}$ computations per block of k information bits, it becomes obvious that the way to achieve what Shannon predicted (i.e. arbitrarily low probabilities of decoder error) is by no means easy. If the serial channel bit-rate is C bps, then the decoder examines C/n blocks/sec or, otherwise, it has n/C sec to make $2^{k(m+1)}$ comparisons. Consequently, the decoder can spend no more than T

$= n2^{-k(m+1)}/C$ sec per comparison. So, in effect, the computation load increases exponentially with the product km . Using inequality (1.13) it becomes obvious then that the achievable performance of error-control coding schemes is limited by practical considerations:

$$P(E) \leq (CT/n)E_c(R)/R \quad (1.14)$$

So, if MLD is to be used, the obtainable performance strongly depends on data rate C and the processing speed of the available hardware (included in T).

One further problem arises from the fact that the noisy-channel theorem does not show the way to achieve the above mentioned performance, even if the practical considerations were not a problem.

Coding theory came to tackle both problems. There are techniques to design codes of given performance and there are decoding methods, other than MLD, that are sub-optimum but whose computational load does not increase exponentially with the code length.

1.4 NET CODING-GAIN

Consider the relation between the probability of a bit error and the signal-to-noise ratio (SNR) $\Gamma \triangleq E/\bar{n}$, where E is the energy per bit and $\bar{n}/2$ is the double-sided noise power spectral density. For uncoded BPSK transmission over the AWGN channel and coherent demodulation with hard decisions, this is relation (1.4). One would be interested to examine the E/\bar{n} ratio required (by the uncoded system) to achieve the same error-rate performance with a coded system of code-rate R . A 'fair' comparison should take into account the fact that the coded system uses more bits/sec to communicate the same information and hence it needs more power in order to maintain the same SNR in the channel. A 'fair' comparison should use the same SNR per message (or information) bit. This means that the coded system operates with a reduced (by a factor of R) energy/bit ratio.

Consequently, in order to assess the gains obtained from a particular channel-coding scheme, one would have to plot the probability of bit-error versus the SNR per information-bit, which is obviously the SNR/bit divided by R , ($R=1$ for the uncoded system), for each of the two cases (under, of course, the same conditions - modulation, demodulation, waveform channel). Then, for a given bit error-rate, the dB-difference between the two required SNRs, called the *net coding-gain*, would be a fair 'estimate' of the benefits of channel coding. A complete comparison, though, should take into account the delay imposed by the encoder & decoder, as well as their cost.

Coming back to the net coding-gain, G , let Γ denote, from now on, the SNR per information-bit. Then ΓR is the SNR per bit and the decoder 'sees' a channel error-rate [see (1.4)]

$$P_e = \frac{1}{2} \text{erfc}[\sqrt{\Gamma R}] \quad (1.15)$$

In response to this P_e , the decoder 'generates' P_d , the probability of decoding a bit in error. Let Γ' be the SNR required by the uncoded system so that the latter achieves the same error-rate performance. Then,

$$P_d = \frac{1}{2} \text{erfc}(\sqrt{\Gamma'}) \quad (1.16)$$

Let erfc^{-1} be the inverse of erfc , i.e.

$$\text{erfc}^{-1}[\text{erfc}(x)] = x \quad (1.17)$$

$$\text{Then,} \quad \Gamma' = [\text{erfc}^{-1}(2P_d)]^2 \quad (1.18)$$

$$\text{and} \quad \text{net coding-gain} = G \hat{=} 10 \log(\Gamma'/\Gamma) \text{ dB} \quad (1.19)$$

Asymptotic coding gain is often used as a figure of merit for a particular coding scheme. This is the limit value of G , as SNR tends to infinity. See Appendix 1.4 (p. 294) for a discussion on the asymptotic coding gain of a block code. In there, it is shown that $G_\infty \approx 10 \log[R(t+1)]$ dB. So, for a rate-3/4 single-error correcting code, $G_\infty = 1.8$ dB, for an $R=3/4$ double-error correcting code, $G_\infty = 3.5$ dB, etc.

1.5 CONCLUSIONS

In this introductory chapter, the model of a real digital communications system was considered as the starting point of the thesis. This model is based on the functional block diagram of the system (see Fig. 1.1). The remarkable feature of this diagram is its symmetry. Specifically, for every processing block-unit at the transmitting site, there is a de-processing one at the receiving site (with the exception of the channel and of SYNC). Of particular importance is the order according to which the various block units are linked, and the fact that this order is reversed at the other site.

The block diagram of Fig. 1.1 was easily (and quickly) reduced to that of Fig. 1.3a, made only of the channel encoder and decoder and their source, destination & channel. The last three units are logical, in that they do not exist physically but they simulate the corresponding portions of the system of Fig. 1.1. For example, the channel encoder 'sees' as its source the encryption unit (if there is one) and all that lies before it.

Of particular importance, for the study of error-control coding, is the *encoder-to-decoder channel*. This was assumed to be a 'black box' which reproduces at its O/P the bit-stream at its I/P. The two bit-streams are identical, except that some of the O/P bits have been inverted, at random. Two such binary channel-models were mentioned, the BSC and one with memory. Their only difference is the correlation among the errors they inject. The BSC corrupts bits independently from previous corruptions, while the other channel produces bursts of errors.

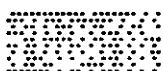
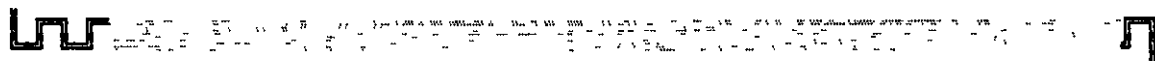
Having defined a channel model, over which an error-control scheme may be tested, the next two (obvious) problems are concerned with encoding and decoding.

In Section 1.2 it was reasoned that, if a code is to correct it needs to process bits in blocks (of k) and 'enrich' each block with $n-k$ parity-checks (redundancy). More parity-checks are expected to result in a greater error-correcting capability, t , but they will definitely increase the overhead (a penalty). Furthermore, it was found that 'long-

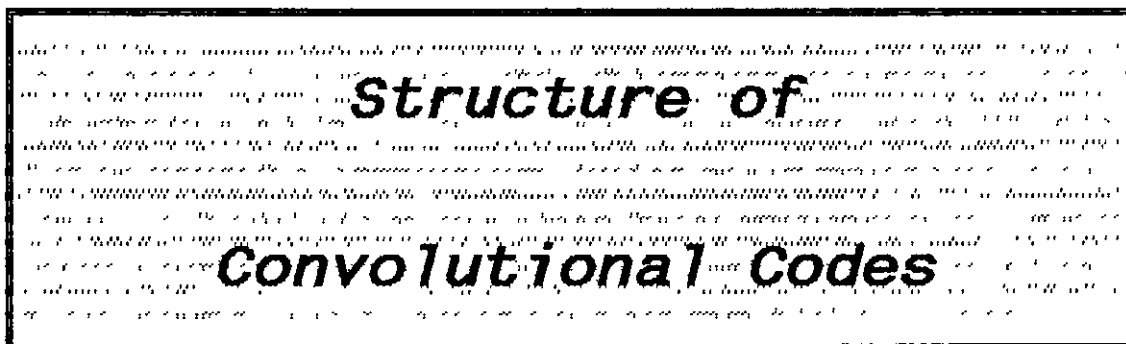
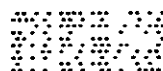
er' codes are expected to be better codes (*noise averaging*). This is, in effect, a collective security of the transmitted bits, where the greater their number the better their defense against errors.

As far as decoding is concerned, it was concluded that the optimum method, called *maximum likelihood decoding* (MLD - see Section 1.3), requires 2^k calculations per received block, in order to determine the nearest codeword. Shannon's second theorem guarantees the existence of codes which can make the probability of error arbitrarily low (under certain conditions). The problem is to find these codes and propose a feasible decoding method. It is not surprising that, usually, one has to start from the end because the decoder is very complex. Code length is limited by the technology of the day, and then by cost (MLD complexity increases as 2^k). A sub-optimum method would sacrifice some error-correcting power to allow the use of longer codes, or faster bit-rates. In designing a code, the obvious trade-off is between code rate, R , and error-correcting capability, t .

Finally, the performance (of an error-control system) is measured by the relative frequency of decoded-bit failures (P_d), for a given BSC bit error-rate P_e . Since, though, a coded system uses more bits/sec than the uncoded one, a 'fair' comparison should take this into account. The net coding-gain (see Section 1.4), G , measures the net gain in signal power per information-bit, achieved by the introduction of coding. This means though that the test-bed used should include the modulator & demodulator, as well as their (waveform) channel. The most widely used (and easy to simulate) waveform channel is the AWGN one (see § 1.1.2.). The best choice for the modulator-demodulator pair is then, BPSK with coherent demodulation. Note that the set of such a modulator, channel and demodulator is equivalent to the BSC, if the demodulator uses hard decisions (i.e. a one-bit quantizer).



CHAPTER 2



Chapter 2 has been designed to serve as a formal introduction to the general theory of convolutional codes. Choice of material and emphasis have been determined by the needs of the thesis and of mathematical formalism. As a consequence, otherwise important aspects of the theory have not been given much attention. On the other hand, a few topics that contain some original material have been treated rather extensively.

It is the author's opinion that proofs are an integral part of the process of learning but that they do not always offer 'good value for money'; so, the proofs that add relatively little to the understanding of the subject are given in appendices.

An effort has been made to document the results properly. To achieve this, some basic theorems are treated as axioms and the rest of the thesis is built on them, by use of mathematical logic.

Various authors present convolutional codes in various ways and use different notation. The author of this thesis finds the approach of Lin & Costello [2] more suitable.

Chapter 2 is made of twenty-one sections; it covers a comparison between the matrix and the polynomial approaches to convolutional-code theory, constraint-length, code rate, distance measures, generator sequences, generator matrix,

generator polynomials, encoder-output equations, generator-polynomial matrix, composite generator polynomials, catastrophic codes, systematic codes, parity-check matrix, syndromes, normal encoder and type-II encoder.

The decoding techniques for convolutional codes are discussed in the next chapter.

Throughout this thesis, and unless otherwise stated, only non-catastrophic codes (see Section 2.14) will be considered.

2.1 COMPARISON WITH BLOCK CODES

Consider an (n,k) block code. The n channel digits of any block are calculated on the basis of the k message digits of this block only.

An (n,k,m) convolutional code can be viewed as a generalization of block codes: The n channel digits of block h are calculated on the basis of the k message digits of blocks $h, h-1, \dots, h-m$ (or stated otherwise, the n channel digits of the current block depend on the current and the previous m blocks). Note that each channel block depends on $m+1$ other blocks and, from this point of view, linear block codes are convolutional codes with $m=0$.

Note also that the n channel digits do not form a codeword of their own but they constitute a *frame* within the codeword. In fact, the codeword is nL digits long, where kL is the length of the message sequence. To state it otherwise, the *codeword* lasts for as long as the encoder is in operation.

Hence, for a kL -digit message, a block encoder will transmit L codewords, while a convolutional encoder only one. Finally, while in a block code the codeword has the length of a single channel block (i.e. n), in a convolutional code the codeword is nL digits long.

2.2 ENCODER

A (linear)* (n,k,m) convolutional encoder can be implemented with a k -input, n -output, linear sequential circuit (LSC) with input memory m [2] (see Fig. 2.1).

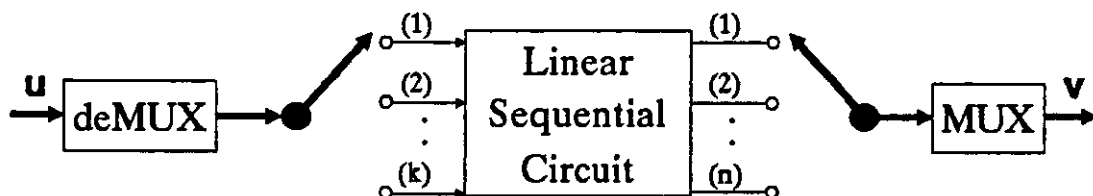


Figure 2.1: A general linear convolutional encoder with its serial-to-parallel and parallel-to-serial interfaces.

Such an encoder can be realized with k shift registers of various lengths and with up to n exclusive-or (X-OR) gates, as shown in Fig. 2.2. Note that the length of each shift register (SR) varies between 0 and m . In fact:

Definition 2.1: The length of the longest shift register of an encoder, for an (n,k,m) convolutional code, is m ; for this reason, m is called the *memory order*.

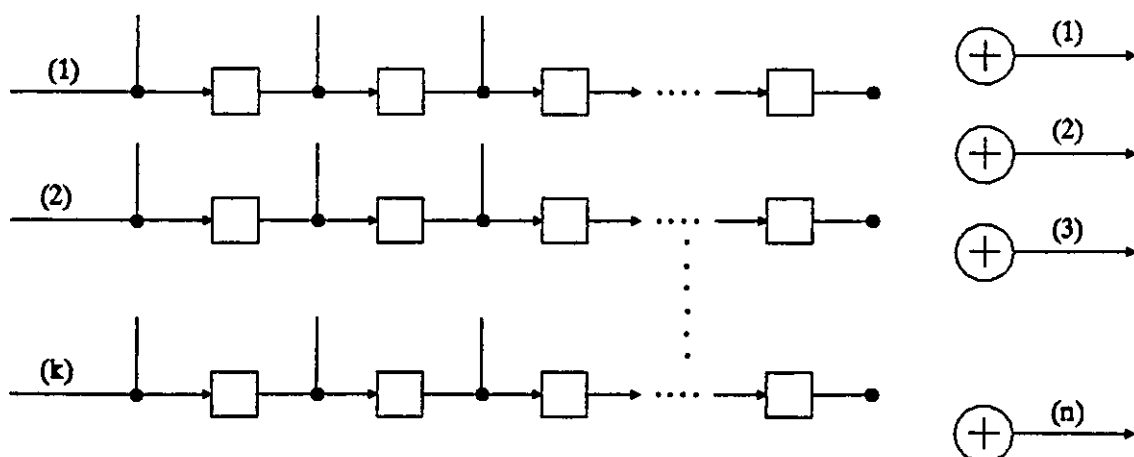


Figure 2.2: General diagram of a (parallel-in, parallel-out) 'normal' convolutional encoder.

* Strictly speaking, 'CCs' are linear by definition since they are a class of linear tree codes; unless otherwise stated, only linear codes will be considered.

An encoder for an (n,k,m) convolutional code is called *normal* iff it is realized with k SRs of various lengths and n X-OR gates (see Definition 2.9 for a complete description).

2.3 CONSTRAINT-LENGTH

It is obvious, from Figures 2.1 & 2.2, that each channel block depends on $m+1$ source blocks.

Definition 2.2: The quantity $N=m+1$ is called *block constraint-length*. ■

Inspection of the encoder of Fig. 2.2, reveals that a certain source digit may stay in the encoder for up to m time-units (in fact there is at least one shift register in which this occurs, according to Definition 2.1); during each time-unit, it affects up to n channel digits. Therefore, a certain message digit may affect up to $(m+1)n$ channel digits (this includes the current block).

Definition 2.3: The *actual constraint-length*, n_A , of a convolutional code is defined as

$$n_A \triangleq (m+1)n \quad (2.1)$$

Note that n_A participates explicitly in the "noisy-channel coding theorem", as may be seen in the next section. ■

2.4 THE CODING THEOREM

Theorem 2.1: The *noisy-channel coding theorem*, for convolutional codes:

For any $R < C$, there exists an (n,k,m) convolutional code such that, with maximum likelihood decoding,

$$\log_2 P(E) \leq -n_A E_c(R) \quad (2.2)$$

where: $R = k/n =$ code rate

$C =$ channel capacity

$P(E) =$ probability of decoding-error

$n_A =$ actual constraint-length

$E_c(R) =$ positive function of R (if $R < C$), completely determined by the channel characteristics

The bound (2.2) implies that for a fixed code-rate, R , arbitrarily small error probabilities can be achieved, by increasing the memory order m , or in general by increasing the actual constraint-length n_A , while keeping the ratio k/n constant.

2.5 COMPARISON BETWEEN THE POLYNOMIAL AND MATRIX APPROACH

Many quantities will be introduced in the remainder of this chapter; they include the I/P and the O/P of the encoder, of the channel and of other block units of the communications system. These quantities, and the relationships among them, can be expressed using either the *matrix approach* or the *polynomial approach* to convolutional codes.

The formal introduction of both approaches is necessary, because both are useful and both are used by various authors. On the other hand, complexity and confusion increases by a significant amount. Appendix 2.4 (p. 307) explains the basic difference between these two approaches and hence (partly) alleviates this problem.

2.6 CODE RATE

It is clear from Fig. 2.1 that a convolutional encoder generates n channel digits for every k message digits. Therefore:

$$\text{Code rate} = R \triangleq k/n \quad (2.3)$$

Nevertheless, the effective code-rate is somewhat smaller

than R and their difference becomes negligible, if the message length L becomes sufficiently large.

To understand this, consider the general convolutional encoder (see Fig. 2.2). At the time the 1st block of k message digits is applied at the input ports the encoder memory should be clear from any past digits; otherwise, the channel digits will depend not only on the source digits but also on a collection of unknown digits, hence the decoder would be unable to uniquely decode, even if there are no channel errors. These unknown digits, would reside in the encoder memory since the times of the previous message.

Consequently, each message should be followed by an encoder-resetting sequence of 'zeros'. Since the longest SR has length m (see Definition 2.1) this resetting sequence should be made of m blocks of zeros. *

Note 2.1: A message sequence must be terminated with mk 'zeros' in order to clear the encoder memory (for the next message sequence) [2].

Consider a message sequence of kL digits. According to the conclusion above, $kL+km$ digits must be shifted in the encoder and consequently $nL+nm$ channel digits must be transmitted. So:

Note 2.2: The encoding of a kL -digit message results in $n(L+m)$ channel digits and this means that

$$\text{Effective code-rate} = R_{\text{eff}} = kL/(nL+nm) \quad (2.4)$$

From eqns (2.3) & (2.4), one can obtain:

$$R_{\text{eff}} = R/(1+m/L) \quad (2.5)$$

Note [from eqn (2.5)] that the code rate R is reduced by a factor $1/(1+m/L) < 1$ and that

$$\text{As } L \rightarrow +\infty, \quad R_{\text{eff}} \rightarrow R \quad (2.6)$$

Definition 2.4: The relative decrease in code rate is called *fractional rate loss* [2] and equals:

* Alternatively, the tail of nm zeros terminates a codeword.

$$(R - R_{\text{eff}})/R = 1 - L/(L+m) = 1/(1+L/m) \quad (2.7)$$

For example, in order to keep the fractional rate loss below 1%, the message must be at least 99m blocks long.

2.7 GENERATOR SEQUENCES

For the rest of this chapter, and unless otherwise stated, only binary codes will be considered; so instead of "digit" the term "bit" will be used, if appropriate (note that BIT = BInary digiT).

Let the response of the encoder's j th output (see Fig. 2.1), to the binary impulse (1000...) applied at input i ($1 \leq i \leq k$), be represented by the row vector $g_j^{(i)}$ ($1 \leq j \leq n$).

Since the encoder has an m time-unit memory and no feedback loops, the impulse response can last for at most $m+1$ time-units.

Definition 2.5: The kn N -tuples $g_j^{(i)}$ ($1 \leq i \leq k$ & $1 \leq j \leq n$)

$$g_j^{(i)} \triangleq (g_{j,0}^{(i)} \ g_{j,1}^{(i)} \ \dots \ g_{j,m}^{(i)}) \quad (2.8)$$

are called *generator sequences* and they specify the code completely.

2.8 THE OUTPUT OF A BINARY CONVOLUTIONAL ENCODER

For the whole of Chapter 2, the terms addition and sum, as well as related symbols, will stand, indiscriminately, both for ordinary addition and for scalar or vector addition over GF(2). Explanations will be provided only when confusion might arise.

It is well known* that the convolution between the input and the impulse response of a linear time-invariant (LTI) system, equals the output of the system. The convolutional encoder is made of delay elements and X-OR gates (see Fig. 2.2), hence it is an LTI system.

* See for example Rabiner & Gold [11], pp. 12-4.

Let the semi-infinite vector,

$$\mathbf{u}^{(i)} \triangleq (u_0^{(i)} u_1^{(i)} u_2^{(i)} \dots) \quad /1 \leq i \leq k \quad (2.9)$$

represent the *input (or message, or information) sequence* at port i of the encoder. From eqn (2.8), the code generator sequence $\mathbf{g}_j^{(i)} = (g_{j,0}^{(i)} g_{j,1}^{(i)} \dots g_{j,m}^{(i)})$ is the impulse response corresponding to I/P i and O/P j . Hence:

Note 2.3: $\mathbf{u}^{(i)} * \mathbf{g}_j^{(i)}$ is the response of the encoder at its j th output port ($1 \leq j \leq n$), when the i th input port ($1 \leq i \leq k$) is excited by $\mathbf{u}^{(i)}$ and all other ports are held to zero (i.e. they receive the all-zero sequence). The symbol '*' denotes *convolution*.

Consider the case where all k I/P ports are excited by the general binary sequences $\mathbf{u}^{(i)} / i=1,2,\dots,k$. Since the system is linear, the j th encoder O/P will be the sum of all responses $\mathbf{u}^{(i)} * \mathbf{g}_j^{(i)}$.

Specifically, if the semi-infinite vector,

$$\mathbf{v}^{(j)} \triangleq (v_0^{(j)} v_1^{(j)} v_2^{(j)} \dots) \quad /1 \leq j \leq n \quad (2.10)$$

represents the *output (or channel) sequence* at port j , then

$$\mathbf{v}^{(j)} = \sum_{i=1}^k \mathbf{u}^{(i)} * \mathbf{g}_j^{(i)} \quad , \text{ for all } j=1,2,\dots,n \quad (2.11)$$

According to the definition of convolution*, $\mathbf{u}^{(i)} * \mathbf{g}_j^{(i)}$ is a semi-infinite row vector with elements

$$(\mathbf{u}^{(i)} * \mathbf{g}_j^{(i)})_h = \sum_z u_{h-z}^{(i)} g_{j,z}^{(i)} \quad /h=0,1,2,\dots \quad (2.12)$$

Since $g_{j,z}^{(i)} = 0$, for $z < 0$ or $z > m$ [see reln (2.8)], z should vary from 0 to m . Since also $u_z^{(i)} = 0$, for $z < 0$ [see relation (2.9)], h should be kept $h > z$. Then eqn (2.12) will give:

$$(\mathbf{u}^{(i)} * \mathbf{g}_j^{(i)})_h = \sum_{z=0}^{\theta} u_{h-z}^{(i)} g_{j,z}^{(i)} \quad /h=0,1,2,\dots \quad (2.13)$$

where $\theta \triangleq \text{MIN}\{h, m\}$.

* See for example Rabiner & Gold [11], pp. 12-4.

Combining equations (2.11) & (2.13) and interchanging the order of summation:

Theorem 2.2: The j th bit of the h th channel block is given by,

$$v_h^{(j)} = \sum_{z=0}^{\theta} \sum_{i=1}^k u_{h-z}^{(i)} g_{j,z}^{(i)} \quad / h \geq 0 \quad \& \quad j=1,2,\dots,n \quad (2.14)$$

where $\theta = \text{MIN}\{h, m\}$, $u_z^{(i)}$ is the i th bit of the z th source source block & $g_{j,z}^{(i)}$ is the z th bit of the i th code generator sequence.

The serial input and output sequences are represented by the semi-infinite row vectors u and v , respectively:

$$u \triangleq [u_0, u_1, \dots, u_h, \dots] \quad / u_h \triangleq (u_h^{(1)} u_h^{(2)} \dots u_h^{(k)}) \quad (2.15)$$

$$v \triangleq [v_0, v_1, \dots, v_h, \dots] \quad / v_h \triangleq (v_h^{(1)} v_h^{(2)} \dots v_h^{(n)}) \quad (2.16)$$

Careful inspection of eqn (2.14) reveals the following:

Note 2.4: Source blocks $u_h, u_{h-1}, \dots, u_{h-m}$ participate in the formation of $v_h^{(j)}$, unless $h \leq m$ in which case all the past source blocks participate.

Note 2.5: A particular source bit, say, $u_z^{(i)}$ takes part in the calculation of a particular channel bit, say, $v_h^{(j)}$ if, and only if, $g_{j,h-z}^{(i)} = 1$.

2.9 DISTANCE MEASURES FOR CONVOLUTIONAL CODES

The material on distance measures is given in Appendix 2.5 (p. 309) because, although it is an important part of convolutional-code theory, it is not directly useful to this thesis.

$$v_h = [u_{h-m}, u_{h-m+1}, \dots, u_h] \begin{bmatrix} g_{1,m} & g_{2,m} & \dots & g_{n,m} \\ g_{1,m-1} & g_{2,m-1} & \dots & g_{n,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{1,0} & g_{2,0} & \dots & g_{n,0} \end{bmatrix} \quad (2.20)$$

where $h=0,1,2,\dots$ and $u_x = 0$ if $x < 0$.

The matrix equation above can be written in a simpler form, if the following notation is adopted:

$$\text{For } z=0,1,\dots,m: \quad G_z \triangleq [g_{1,z}, g_{2,z}, \dots, g_{n,z}] \quad (2.21)$$

Combining eqns (2.20) & (2.21):

$$v_h = [u_{h-m}, u_{h-m+1}, \dots, u_h] \begin{bmatrix} G_m \\ G_{m-1} \\ \vdots \\ G_0 \end{bmatrix} \quad (2.22)$$

where $h=0,1,2,\dots$ and $u_x = 0$ if $x < 0$.

Note 2.6: Matrix equation (2.22) relates the h th channel block with the current and the past m message (source) blocks, through a $(kn) \times n$ system matrix (made of the coefficients of the encoder's impulse responses).

Examining eqn (2.21), one can easily deduce [using eqn (2.17b)] that

$$\text{For all } z=0,1,\dots,m: \quad G_z = \begin{bmatrix} g_{1,z}^{(1)} & g_{2,z}^{(1)} & \dots & g_{n,z}^{(1)} \\ g_{1,z}^{(2)} & g_{2,z}^{(2)} & \dots & g_{n,z}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ g_{1,z}^{(k)} & g_{2,z}^{(k)} & \dots & g_{n,z}^{(k)} \end{bmatrix} \quad (2.23)$$

Finally, using a theorem on the multiplication of parti-

tioned matrices (see Appendix 2.6, p. 311), matrix equation (2.22) can be written in a more compact form:

$$\text{For } h \geq 0 \text{ \& } \theta \triangleq \text{MIN}\{h, m\}: \quad v_h = \sum_{z=0}^{\theta} u_{h-z} G_z \quad (2.24)$$

It is the opinion of the author that matrix equations (2.22) & (2.24) are more useful for the representation of the encoder O/P, than any other expression found in any of the textbooks (Lin & Costello [2], Blahut [10], Lin [12], Clark & Cain [13], Wiggert [14], Peterson & Weldon [15], Lucky et al [16], Heller [17] & Massey [18],[19]) or papers (Forney [20],[21]) known to the author and discussing the matrix (or polynomial) theory of convolutional codes.

2.11 A NOVEL APPROACH TO THE GENERATOR MATRIX

The objective of this section is to develop a matrix equation for an arbitrary and finite portion $[v_h, v_{h+1}, \dots, v_{h+x}]$ of the channel sequence v , where h & x are nonnegative integers. This will be related to the corresponding message sequence through a system matrix, denoted by $[G]_x^h$. What is called *generator matrix* in the literature, will be readily obtained from $[G]_x^h$, if $h=0$ and $x \rightarrow +\infty$. For a proof of the following theorem see Appendix 2.7 (p. 312):

Theorem 2.3: Consider an (n, k, m) convolutional code. Let v_h denote the h th channel n -tuple and u_h the h th source k -tuple. For $h \geq 0$, $x \geq 0$ and $\theta \triangleq \text{MIN}\{h, m\}$:

$$\text{If} \quad [v]_x^h \triangleq [v_h, v_{h+1}, \dots, v_{h+x}] \quad (2.25a)$$

$$[u]_x^h \triangleq [u_{h-\theta}, u_{h-\theta+1}, \dots, u_{h+x}] \quad (2.25b)$$

$$\text{and} \quad [G]_x^h \triangleq \begin{bmatrix} G_{\theta} & G_{\theta+1} & \cdots & G_{\theta+x} \\ G_{\theta-1} & G_{\theta} & \cdots & G_{\theta+x-1} \\ \cdots & \cdots & \cdots & \cdots \\ G_{-x} & G_{-x+1} & \cdots & G_0 \end{bmatrix} \quad (2.25c)$$

then

$$[v]_x^h = [u]_x^h [G]_x^h \quad (2.26)$$

where G_i is specified by eqn (2.23) and it is understood that $G_i = 0$ for $i \notin [0, m]$.

In eqn (2.26), $[v]_x^h$ is a $1 \times (x+1)$ matrix of n -tuples, $[u]_x^h$ is a $1 \times (x+1+\theta)$ matrix of k -tuples and $[G]_x^h$ is an $(x+1+\theta) \times (x+1)$ matrix of $k \times n$ submatrices.

The following results are special cases of eqn (2.26):

Lemma 2.1: For $h \geq m$ and $x \geq 0$:

$$[v_h, v_{h+1}, \dots, v_{h+x}] = [u_{h-m}, u_{h-m+1}, \dots, u_{h+x}] \begin{bmatrix} G_m & 0 & \dots & 0 \\ G_{m-1} & G_m & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & G_0 \end{bmatrix} \quad (2.27)$$

Proof: It follows from eqns (2.25) & (2.26), noting that $\theta \hat{=} \text{MIN}\{m, h\} = m$ and $G_i = 0$ for $i \notin [0, m]$.

Lemma 2.2: For all $h \geq m$ & $h' \geq m$ and $x \geq 0$:

$$[G]_x^h = [G]_x^{h'} \quad (2.28)$$

Proof: It follows from inspection of eqn (2.27).

Lemma 2.3: If $G_i = 0$ for $i < 0$, for $x \geq 0$ & $h \geq 0$ / $x+h \leq m$:

$$[v]_x^h = [u_0, u_1, \dots, u_{h+x}] \begin{bmatrix} G_h & G_{h+1} & \dots & G_{h+x} \\ G_{h-1} & G_h & \dots & G_{h+x-1} \\ \dots & \dots & \dots & \dots \\ G_{-x} & G_{-x+1} & \dots & G_0 \end{bmatrix} \quad (2.29)$$

Proof: It follows from (2.25) & (2.26), noting that $x+h \leq m$ and $\theta \hat{=} \text{MIN}\{h, m\} = h$.

Lemma 2.4: For all $x \geq 0$:

$$[v_0, v_1, \dots, v_x] = [u_0, u_1, \dots, u_x] \begin{bmatrix} G_0 & G_1 & \dots & G_x \\ 0 & G_0 & \dots & G_{x-1} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & G_0 \end{bmatrix} \quad (2.30)$$

where $G_i = 0$ if $i < 0$ or $i > m$.

Proof: It follows from (2.25) & (2.26), if $h=0$ (in which case $\theta \triangleq \text{MIN}\{0, m\} = 0$).

Definition 2.6: The generator matrix, G , is defined by:

$$G \triangleq \lim_{x \rightarrow +\infty} [G]_x^0 \quad (2.31)$$

Using the definitions of G and $[G]_x^h$, one can obtain the following well-known form of the generator matrix (the elements outside the boxes are all zeros).

$$G = \begin{bmatrix} \begin{array}{|c|c|c|} \hline G_0 & G_1 & G_2 \\ \hline \end{array} & & \dots & \begin{array}{|c|c|} \hline G_{m-1} & G_m \\ \hline \end{array} & & \\ & \begin{array}{|c|c|} \hline G_0 & G_1 \\ \hline \end{array} & & \dots & \begin{array}{|c|c|c|} \hline G_{m-2} & G_{m-1} & G_m \\ \hline \end{array} & & \\ & & \begin{array}{|c|c|c|} \hline G_{m-3} & G_{m-2} & G_{m-1} \\ \hline \end{array} & & \begin{array}{|c|c|} \hline G_0 & G_1 \\ \hline \end{array} & & \begin{array}{|c|c|c|} \hline G_{m-1} & G_m & \\ \hline \end{array} & & \\ & & & \begin{array}{|c|c|c|} \hline G_0 & G_1 & G_2 \\ \hline \end{array} & & \begin{array}{|c|c|} \hline G_1 & G_2 \\ \hline \end{array} & & \begin{array}{|c|c|c|} \hline G_{m-1} & G_m & \\ \hline \end{array} & & \\ & & & & \begin{array}{|c|c|c|} \hline G_0 & G_1 & G_2 \\ \hline \end{array} & & \begin{array}{|c|c|} \hline G_1 & G_2 \\ \hline \end{array} & & \begin{array}{|c|c|c|} \hline G_{m-1} & G_m & \\ \hline \end{array} & & \\ & & & & & \begin{array}{|c|c|c|} \hline G_0 & G_1 & G_2 \\ \hline \end{array} & & \begin{array}{|c|c|} \hline G_1 & G_2 \\ \hline \end{array} & & \begin{array}{|c|c|c|} \hline G_{m-1} & G_m & \\ \hline \end{array} & & \\ & & & & & & \begin{array}{|c|c|c|} \hline G_{m-1} & G_m & \\ \hline \end{array} & & \dots & & \end{bmatrix} \quad (2.32)$$

Theorem 2.4:

$$v = uG \quad (2.33)$$

Proof: It follows from (2.30), if x is left to $\rightarrow +\infty$ and if the definitions of G (2.6) and v & u [eqns (2.15) & (2.16), respectively] are employed.

Matrix equation (2.30) appears also in Lin & Costello [2] (Sec. 10.3), in the discussion on minimum distance. But eqn (2.30) is a special case of Theorem 2.3; it represents original work and, to the best of the author's knowledge, it does not appear in any publication. On the other hand, Theorem 2.3 is not a new result; it is simply another way to look at the same thing. But the theorem certainly improves the presentation of convolutional codes because it is more general (it includes $v=uG$) and at the same time more specific (it offers controls that permit 'zooming' onto specific portions of the encoder's output sequence). Finally, note that this result (as well as preceding ones) follows analytically from the concept of the impulse response of a linear time-invariant circuit (see Section 2.8).

2.12 GENERATOR POLYNOMIALS

The objective of this section is to introduce the reader to the polynomial approach to convolutional codes.

Let the coefficients of polynomials in D represent the various binary sequences. In accordance with the notation previously introduced,

For $i=1,2,\dots,k$, the i th *message polynomial* is defined by

$$u^{(i)}(D) \triangleq u_0^{(i)} + u_1^{(i)}D + u_2^{(i)}D^2 + \dots \quad (2.34)$$

For $j=1,2,\dots,n$, the j th *channel polynomial* is defined by

$$v^{(j)}(D) \triangleq v_0^{(j)} + v_1^{(j)}D + v_2^{(j)}D^2 + \dots \quad (2.35)$$

The indeterminate D can be interpreted as a delay operator, the power of D denoting the number of time-units a bit is delayed with respect to the initial bit, in the sequence [2]. It can be shown (see Appendix 2.8, p. 313) that,

$$v^{(j)}(D) = \sum_{i=1}^k u^{(i)}(D) g_j^{(i)}(D) \quad / 1 \leq j \leq n \quad (2.36)$$

$$g_j^{(i)}(D) \triangleq g_{j,0}^{(i)} + g_{j,1}^{(i)}D + \dots + g_{j,m}^{(i)}D^m \quad / 1 \leq j \leq n \text{ \& } 1 \leq i \leq k \quad (2.37)$$

Definition 2.7: The kn polynomials $g_j^{(i)}(D)$, defined in (2.37), are called the *generator polynomials* of the corresponding code and they completely specify the code. ■

A comparison between the generator sequences [eqn (2.8)] and the generator polynomials [eqn (2.37)] reveals that:

Note 2.7: The coefficients of the generator polynomial $g_j^{(i)}(D)$, form the generator sequence $g_j^{(i)}$, for $i=1,2,\dots,k$ & $j=1,2,\dots,n$. ■

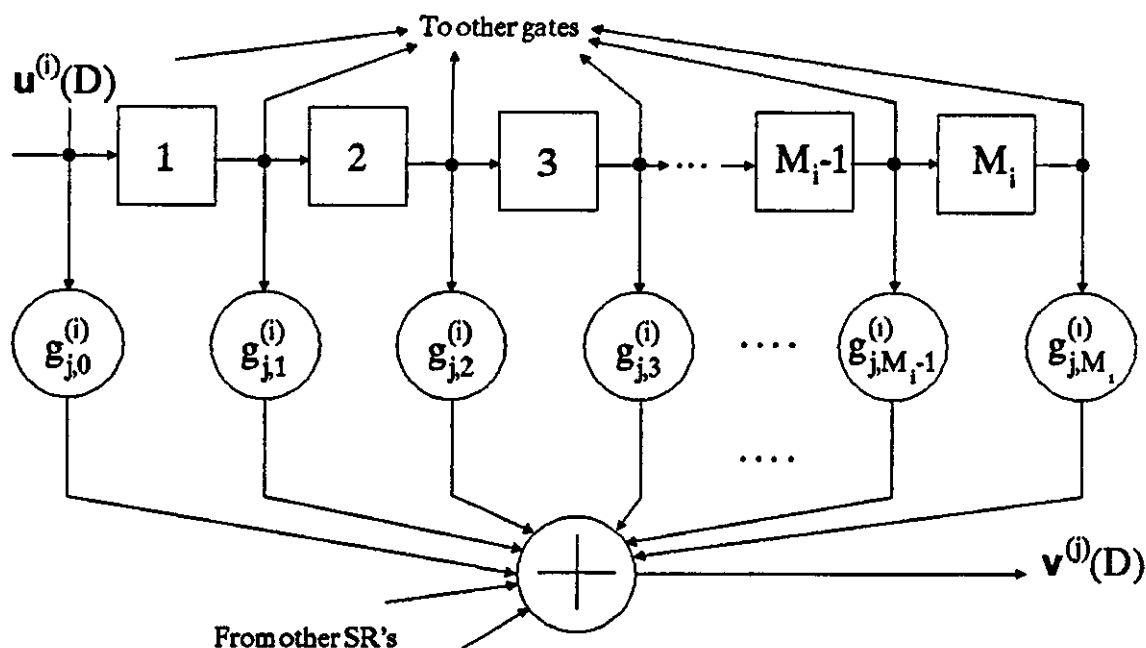


Figure 2.3: The i th SR with connections to the j th gate ($1 \leq i \leq k$ & $1 \leq j \leq n$).

The generator polynomials can be easily deduced from the inspection of the circuit diagram of the normal encoder*. According to Note 2.5, in order to determine $g_{j,z}^{(i)}$ / $z=0,1,\dots,m$, one should examine the connections from the i th SR to

* See Note 2.9.

the j th gate (see Fig. 2.3).

Note 2.8: For a normal encoder realization, $g_{j,z}^{(i)}$ / $0 \leq z \leq m$ is 1 iff there is a connection from the O/P of the z th stage of the i th SR ($1 \leq i \leq k$), to the j th X-OR gate ($1 \leq j \leq n$). It is understood that the O/P of the zeroth stage is the I/P of the SR.

It can be easily deduced that,

$$\text{Length of the } i\text{th SR} = M_i = \text{MAX}_{1 \leq j \leq n} \{ \deg g_j^{(i)}(D) \} \quad (2.38)$$

$$\text{and Memory order} = m = \text{MAX} \{ M_1, M_2, \dots, M_k \} \quad (2.39)$$

2.13 GENERATOR-POLYNOMIAL MATRIX

Expanding the system of eqns (2.36):

$$\begin{aligned} v^{(1)}(D) &= u^{(1)}(D)g_1^{(1)}(D) + u^{(2)}(D)g_1^{(2)}(D) + \dots + u^{(k)}(D)g_1^{(k)}(D) \\ v^{(2)}(D) &= u^{(1)}(D)g_2^{(1)}(D) + u^{(2)}(D)g_2^{(2)}(D) + \dots + u^{(k)}(D)g_2^{(k)}(D) \\ &\vdots \\ v^{(n)}(D) &= u^{(1)}(D)g_n^{(1)}(D) + u^{(2)}(D)g_n^{(2)}(D) + \dots + u^{(k)}(D)g_n^{(k)}(D) \end{aligned} \quad (2.40)$$

System (2.40) can be written in matrix form:

$$V(D) = U(D)G(D) \quad (2.41a)$$

$$\text{where: } U(D) \triangleq [u^{(1)}(D), u^{(2)}(D), \dots, u^{(k)}(D)] \quad (2.41b)$$

$$V(D) \triangleq [v^{(1)}(D), v^{(2)}(D), \dots, v^{(n)}(D)] \quad (2.41c)$$

$$G(D) \triangleq \begin{bmatrix} g_1^{(1)}(D) & g_2^{(1)}(D) & \dots & g_n^{(1)}(D) \\ g_1^{(2)}(D) & g_2^{(2)}(D) & \dots & g_n^{(2)}(D) \\ \vdots & \vdots & \ddots & \vdots \\ g_1^{(k)}(D) & g_2^{(k)}(D) & \dots & g_n^{(k)}(D) \end{bmatrix} \quad (2.41d)$$

Definition 2.8: The $k \times n$ matrix $G(D)$ of generator polynomials with i th-row, j th-column, entry $g_j^{(i)}(D)$, is called the *generator-polynomial matrix**.

2.14 NORMAL ENCODER

The generator-polynomial matrix $G(D)$ offers a very compact description of the code and also an easy way to deduce the normal encoder, which is defined below.

Definition 2.9: Let $G(D)$ be a $k \times n$ matrix of polynomials in D . Then, the *normal encoder* corresponding to $G(D)$ is made of k shift registers and n X-OR gates. The I/P of the i th SR is the i th ($1 \leq i \leq k$) I/P port, while the O/P of the j th gate is the j th ($1 \leq j \leq n$) O/P port. The length of the i th SR is the maximum exponent of D , along the i th row of $G(D)$ [as defined by eqn (2.38)].

The following supplementary results are easily obtained:

Note 2.9: With respect to the normal encoder, corresponding to the $k \times n$ generator-polynomial matrix $G(D)$:

The number of inputs to the j th gate ($1 \leq j \leq n$) equals the number of non-zero polynomial terms along the j th column.

The connections to the j th gate are determined by the j th column of $G(D)$ ($1 \leq j \leq n$). Specifically, $g_j^{(i)}(D)$ determines the contributions from the i th SR to the j th gate. In particular, a connection from the O/P of the h th stage ($0 \leq h \leq m$) of the i th SR ($1 \leq i \leq k$) to the j th gate ($1 \leq j \leq n$) exists iff the coefficient of D^h , in $g_j^{(i)}(D)$, is non-zero.

Note that the type of encoder which was called *normal*, above, has been given other names by various authors. For example, Lin & Costello (see [2], p. 305) use the name *straightforward*, while Forney (see [20] & [21]) uses the terms *obvious* and *controller canonical form* (the latter term

* Lin & Costello [2] call it, *transfer function matrix*.

is used in system theory). Note also that there exists another type of standardized encoder which was introduced by Massey [18], in 1963. This encoder will be introduced after the discussion about systematic codes (see Section 2.18).

Appendix 2.9 (p. 314) illustrates the construction of 3 normal encoders, given their generator-polynomial matrices.

2.15 CATASTROPHIC CODES

There are several convolutional codes for which a message sequence of infinite Hamming weight might produce a channel sequence of finite Hamming weight; if the latter is corrupted by a few channel errors it can be transformed into another codeword corresponding to a message sequence of finite Hamming weight. In such a case, the original and the decoded message sequences will differ in an infinite number of places. Such codes are called *catastrophic*, and are briefly discussed in Appendix 2.10 (p. 317).

2.16 SERIAL ENCODER

Expressions for serial bit-streams in to, and out of, the encoder are obviously useful and constitute a part of the theory of convolutional codes. Nevertheless, the parallel-in, parallel-out, approach is much more useful in describing their structure. In any case, the serial bit-stream expressions are not useful for the development of this thesis. For this reason, the relevant theory (of *composite generator-polynomials*) is given in Appendix 2.11 (p. 320).

2.17 SYSTEMATIC CONVOLUTIONAL CODES

Definition 2.10: An (n,k,m) convolutional code is called *systematic* iff the first k output polynomials $v^{(1)}(D)$, $v^{(2)}(D), \dots, v^{(k)}(D)$ equal the k input polynomials $u^{(1)}(D)$, $u^{(2)}(D), \dots, u^{(k)}(D)$.

$$\text{For } i=1,2,\dots,k: \quad v^{(i)}(D) = u^{(i)}(D) \quad (2.42)$$

The objective of this section is to restate the main results, so far, simplified for systematic codes.

2.17.1. Generator Matrix

Its basic building block is the $k \times n$ matrix G_z / $z=0,1,\dots,m$, whose ij th element ($1 \leq i \leq k$ & $1 \leq j \leq n$) is $g_{j,z}^{(i)}$ [see equations (2.21) & (2.23)].

It can be proved that:

Theorem 2.5: For systematic convolutional codes,

$$G_z = \left[\begin{array}{cc} [I, P_0] & /z=0 \\ [0, P_z] & /1 \leq z \leq m \end{array} \right] \rightarrow \quad (2.43)$$

where I is the $k \times k$ identity matrix ,

0 is the $k \times k$ all-zero matrix

& P_z / $z=0,1,\dots,m$, is the $k \times (n-k)$ submatrix of G_z , made of columns $k+1, k+2, \dots, n$.

Proof: See Appendix 2.12, p. 323.

Theorem 2.6: If I is the $k \times k$ identity matrix, 0 is the $k \times k$ all-zero matrix and P_z is the $k \times (n-k)$ system matrix ($z \geq 0$), given below,

$$\text{For all } z=0,1,\dots,m: \quad P_z = \begin{bmatrix} g_{k+1,z}^{(1)} & g_{k+2,z}^{(1)} & \cdots & g_{n,z}^{(1)} \\ g_{k+1,z}^{(2)} & g_{k+2,z}^{(2)} & \cdots & g_{n,z}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k+1,z}^{(k)} & g_{k+2,z}^{(k)} & \cdots & g_{n,z}^{(k)} \end{bmatrix} \quad (2.44)$$

then the generator matrix G has the form:

(2.45)

$$G = \begin{bmatrix} I P_0 & 0 P_1 & 0 P_2 & \dots & 0 P_{m-1} & 0 P_m \\ & I P_0 & 0 P_1 & \dots & 0 P_{m-2} & 0 P_{m-1} & 0 P_m \\ & & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & I P_0 & 0 P_1 & 0 P_2 \\ & & & & I P_0 & 0 P_1 \\ & & & & & I P_0 & \dots & 0 P_m \\ & & & & & & 0 P_{m-1} & \dots \\ & & & & & & \vdots & \vdots \\ & & & & & & 0 P_1 \\ & & & & & & I P_0 \end{bmatrix}$$

Proof: The form of G follows from Theorem 2.5 and eqn (2.32), while the form of P_z from eqn (2.23). ■

2.17.2. The Output of the Encoder

Lemma 2.5: For an (n,k,m) systematic convolutional code,

$$v_h^{(p)} \triangleq [v_h^{(k+1)} v_h^{(k+2)} \dots v_h^{(n)}] = \sum_{z=0}^{\theta} u_{h-z} P_z \quad /h \geq 0 \quad (2.46)$$

where $\theta \triangleq \text{MIN}\{h,m\}$.

Proof: It follows from eqn (C) of Appendix 2.12 (p. 323). ■

Lemma 2.6: For an (n,k,m) systematic convolutional code,

$$v_h^{(i)} = u_h^{(i)} \quad /i=1,2,\dots,k \quad (2.47a)$$

$$v_h^{(k+j)} = \sum_{z=0}^{\theta} \sum_{i=1}^k u_{h-z}^{(i)} g_{k+j,z}^{(i)} \quad /j=1,2,\dots,n-k \quad (2.47b)$$

where $h \geq 0$ and $\theta \triangleq \text{MIN}\{h,m\}$.

Proof: It follows from the definition of systematic codes and eqns (2.45) & (2.46). ■

2.17.3. Generator Sequences

The generator sequences $g_j^{(i)}$ / $i=1,2,\dots,k$ & $j=1,2,\dots,n$ [see eqn (2.8)] are made of the elements of the $k \times n$ matrices G_z / $z=0,1,\dots,m$ [see eqn (2.23)]. Indeed the ij th element of G_z [$g_{j,z}^{(i)}$] is the $(z+1)$ th element of $g_j^{(i)}$.

Theorem 2.5 states that, for systematic convolutional codes, all the elements in the first k columns of G_z / $z=0,1,\dots,m$ are zero, apart from the diagonal, $g_{1,0}^{(1)}, g_{2,0}^{(2)}, \dots, g_{k,0}^{(k)}$, of G_0 . It follows then that the generator sequences $g_j^{(i)}$ / $j=1,2,\dots,k$, are zero, except from $g_1^{(1)}, g_2^{(2)}, \dots, g_k^{(k)}$, which are impulsive. The following theorem has been proved:

Theorem 2.7: For an (n,k,m) systematic convolutional code, the generator sequences, $g_j^{(i)}$ / $i=1,2,\dots,k$, have the form below:

$$g_j^{(i)} = \begin{cases} 0 & \text{if } j \neq i \text{ \& } 1 \leq j \leq k \\ (1 \ 0 \ 0 \ \dots \ 0) & \text{if } i=j \text{ \& } 1 \leq j \leq k \\ (g_{j,0}^{(i)} \ g_{j,1}^{(i)} \ \dots \ g_{j,m}^{(i)}) & \text{if } k < j \leq n \end{cases} \quad (2.48)$$

Using the above result and Definition 2.5, one can easily prove the following:

Lemma 2.7: A systematic convolutional code is completely specified by its $k(n-k)$ generator sequences $g_{k+1}^{(1)}, g_{k+2}^{(1)}, \dots, g_n^{(1)}$ / $i=1,2,\dots,k$.

2.17.4. Generator Polynomials

Lemma 2.8: For an (n,k,m) systematic convolutional code, the generator polynomials $g_j^{(i)}(D)$ / $i=1,2,\dots,k$, have the following form [$\delta(z) = 0$ if $z \neq 0$ and $\delta(0) = 1$]:

$$g_j^{(i)}(D) = \begin{cases} \delta(i-j) & \text{if } 1 \leq j \leq k \\ \sum_{z=0}^m g_{j,z}^{(i)} D^z & \text{if } k < j \leq n \end{cases} \quad (2.49)$$

Proof: It follows from Note 2.7 & Theorem 2.7. ■

2.17.5. Generator-Polynomial Matrix

Using Definition 2.8, of the generator-polynomial matrix [see also reln (2.41d)] and Lemma 2.8, the following is easily proved:

Lemma 2.9: For an (n,k,m) systematic convolutional code, the generator-polynomial matrix has the form

$$G(D) = [I, P(D)] \quad (2.50)$$

where I is the $k \times k$ identity matrix and $P(D)$ is the $k \times (n-k)$ polynomial matrix

$$P(D) = \begin{bmatrix} g_{k+1}^{(1)}(D) & g_{k+2}^{(1)}(D) & \cdots & g_n^{(1)}(D) \\ g_{k+1}^{(2)}(D) & g_{k+2}^{(2)}(D) & \cdots & g_n^{(2)}(D) \\ \cdots & \cdots & \cdots & \cdots \\ g_{k+1}^{(k)}(D) & g_{k+2}^{(k)}(D) & \cdots & g_n^{(k)}(D) \end{bmatrix} \quad (2.51)$$
■

2.17.6. Composite Generator-Polynomials

See Appendix 2.13 (p. 323).

2.17.7. Encoder

Since the generator polynomial matrix, for the first k outputs, is the $k \times k$ identity matrix [see eqn (2.58)] the first k O/Ps are identical with the k I/Ps (see Fig. 2.4).

Note 2.10: A normal encoder for an (n,k,m) systematic convolutional code is implemented with a k -input, $(n-k)$ -output, LSC containing at most $n-k$ X-OR gates, and k shift registers of lengths varying between 0 and m stages. ■

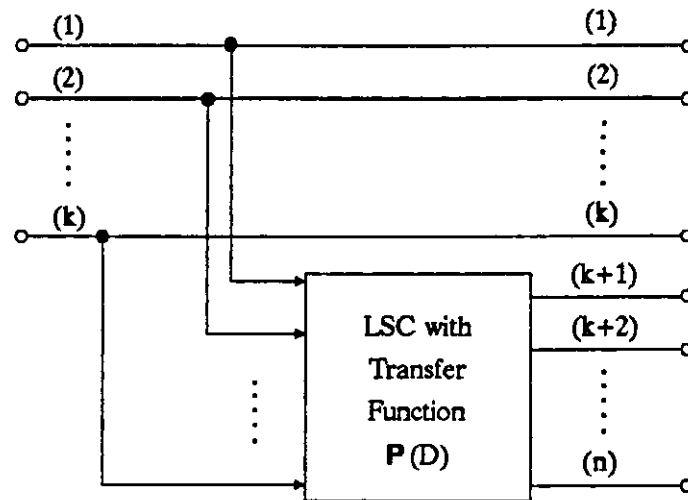


Figure 2.4: Systematic convolutional encoder.

2.17.8. Non-Catastrophic Codes

Theorem 2.8: Every systematic convolutional code is non-catastrophic.

Proof: From eqn (2.50), the first $k \times k$ submatrix of the generator-polynomial matrix, $G(D)$, is the identity matrix I , whose determinant is, of course, equal to 1. Then $\Phi_1(D)$, of Theorem A2.10.2, is 1 and so $\gcd[1, \Phi_i(D) / 2 \leq i \leq C(n, k)] = 1 = D^0$. Then, the systematic convolutional encoder has an FF inverse and hence the code is non-catastrophic (see Theorems A2.10.1 & A2.10.2, pp. 317-8).

QED

2.18 TYPE-II ENCODER

Consider the normal encoder of Fig. A2.9.3 (p. 316). It has 4 SR stages (hence its state-transition diagram has 16 states) but, it may be illustrated that the corresponding code can be realized via a 2-SR stage encoder. It will become clear, in the next chapter, that a reduction of the number of SR stages is highly desirable since the complexity of the Viterbi (and other decoding algorithms) strongly de-

pends on this number.

In most of the cases, the normal encoder uses the minimum number of SR stages. In some cases another type of encoder offers reduced complexity. For example, with systematic rate- $(n-1)/n$ codes, Massey's ([18], p. 23) type-II encoder uses one SR of length m , instead of $n-1$ SRs of lengths between 0 & m . In what follows, the above observations will be systematized. Note though that the results will concern the general (non-systematic) case. Although the encoder to be introduced below is not minimal (i.e. using the minimum number of SR stages) unless at least the code is systematic and $n-k < k$. Nevertheless, it was felt that a specialization for the systematic case only would introduce an unnecessary restriction.

Definition 2.11: Let $G(D)$ be a $k \times n$ matrix of polynomials in D . Then, the *type-II encoder* corresponding to $G(D)$ is made of n shift registers (SRs) and a number of X-OR gates interspersed among the stages of the SRs. The j th SR ends in an X-OR gate whose O/P is the j th O/P port ($1 \leq j \leq n$).

The length (number of stages), M_j , of the j th SR is the maximum exponent of D , along the j th column of $G(D)$:

$$\text{Length of the } j\text{th SR} = M_j \triangleq \max_{1 \leq i \leq k} \{ \deg_j^{(i)}(D) \} \quad (2.52)$$

A comparison between the normal and the type-II encoders reveals that while the former's SRs are defined by the rows of $G(D)$, the latter's SRs are defined by its columns. In both cases, the n O/P ports are the O/Ps of n X-OR gates. Finally, in the normal encoder the connections are from the various stages of the SRs to the n X-OR gates, while in the type-II encoder from the k I/P ports to the stages of the SRs, via the X-OR gates (see Fig. 2.5, below).

The following supplementary results are easily obtained:

Note 2.11: With respect to the type-II encoder, corresponding to the $k \times n$ polynomial-generator matrix $G(D)$:

The maximum number of gates along the j th SR is $M_j + 1$ (see Fig. 2.5). The j th column ($1 \leq j \leq n$) of $G(D)$ describes the

connections to the j th SR, while the i th row ($1 \leq i \leq k$) describes the connections from $u^{(i)}(D)$.

The number of inputs to the h th gate of the j th SR ($1 \leq j \leq n$ & $0 \leq h \leq M_j$) is equal to the number of non-zero coefficients of D^h along the j th column of $G(D)$ (note that the gates are counted from right to left - see Fig. 2.5). Note finally that, gates number $0, 1, \dots, M_j - 1$ have one more I/P, from the previous stage of the SR. Obviously, gates with one I/P do not exist.

A connection from the i th I/P port $u^{(i)}(D)$ ($1 \leq i \leq k$) to the h th gate of the j th SR ($1 \leq j \leq n$ / $0 \leq h \leq M_j$) exists iff the coefficient of D^h in $g_j^{(i)}(D)$ is non-zero.

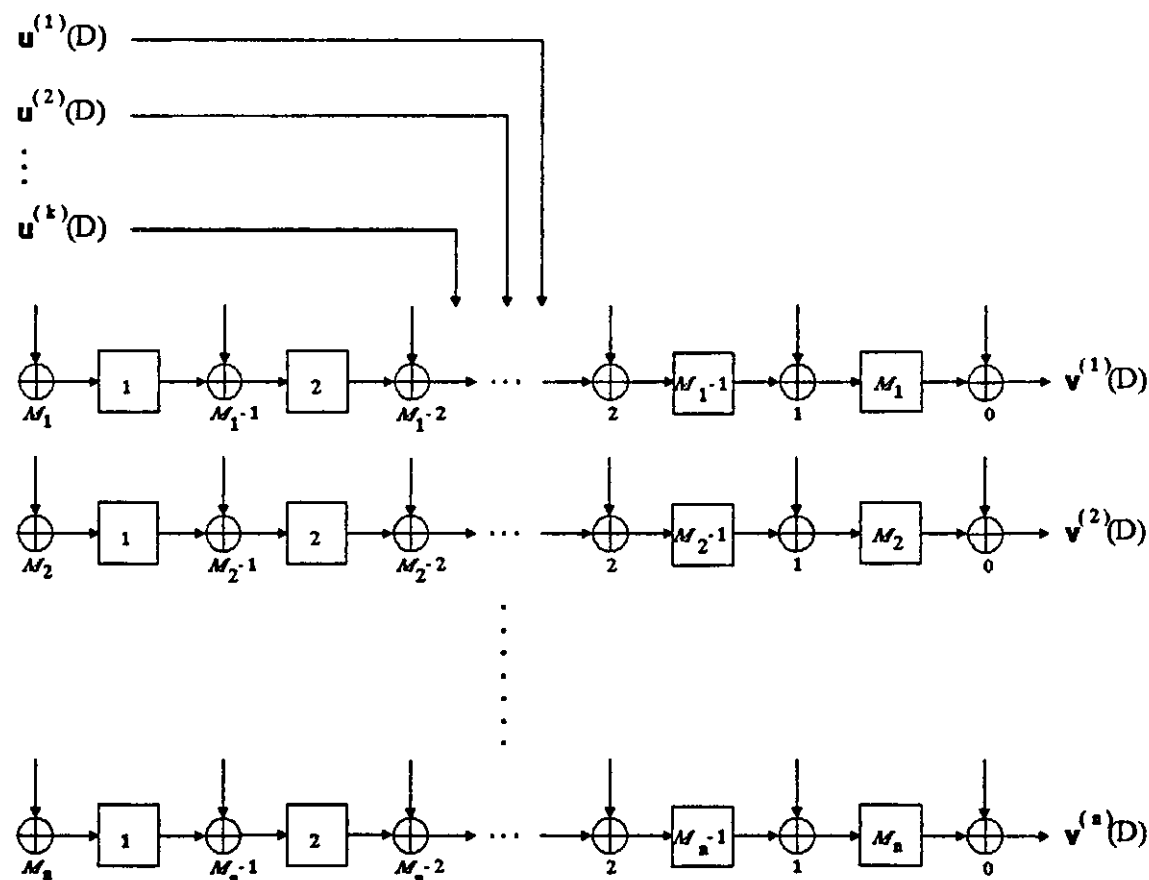


Figure 2.5: General diagram of a type-II convolutional encoder.

Note that the type-II encoder has $n-k$ more SRs than the normal one. The number of O/P gates is the same (i.e. n), while the type-II encoder has a great number of extra gates

dispersed among the stages of the SRs. Although a proof is required, in order to be certain, it is safe to state that for a non-systematic code it is unlikely that the type-II encoder is minimal (i.e. with the minimum number of SR stages). That is not the case, though, with systematic codes, and especially when $n-k < k$, or the same, $R > 1/2$.

Note 2.12: A type-II encoder, for an (n,k,m) systematic convolutional code, can be implemented with a k -input, $(n-k)$ -output, linear sequential circuit, containing $n-k$ SRs and a number of X-OR gates.

In Appendix 2.14 (p. 324) the type-II encoder, corresponding to the code of Example A2.9.3 (p. 316), is developed. Consider the savings obtained through the adoption of the type-II encoder. The normal encoder has $2^4 = 16$ states, while the type-II encoder has only $2^2 = 4$ states (for a formal treatment of the encoder state-diagram, see Section 3.2, pp. 55-64).

One question remains to be answered, before the conclusion of the current section. Given the code polynomial-generator matrix $G(D)$, which of the two types of encoder should be used?

Before attempting to answer the question above, one has to decide which (if any) is more important, the gates or the SR stages? It is rather obvious that the total number of SR stages is a 'fair' measure of the complexity of an encoder. In any case, the complexity of the trellis increases exponentially with the total number of encoder SR-stages.

The *complexity measure* is therefore $M_1 + M_2 + \dots + M_k$ for the normal encoder [see (2.38)], and $M_1 + M_2 + \dots + M_n$ for the type-II encoder [see (2.52)]. Hence, the following theorem:

Theorem 2.9: Let $G(D) = [g_j^{(i)}(D)]$ $/i=1,2,\dots,k$ & $j=1,2,\dots,n$ be a generator-polynomial matrix. Then corresponding to $G(D)$, the type-II encoder is less complex than the normal one if, and only if,

$$\sum_{j=1}^n \text{MAX}_{1 \leq i \leq k} \{ \deg g_j^{(i)}(D) \} < \sum_{i=1}^k \text{MAX}_{1 \leq j \leq n} \{ \deg g_j^{(i)}(D) \} \quad (2.53)$$

2.19 PARITY-CHECK POLYNOMIAL MATRIX

Definition 2.12: The *parity-check polynomial matrix* associated with the $k \times n$ generator-polynomial matrix $G(D)$, is any full-rank $(n-k) \times n$ matrix $H(D)$ of polynomials with elements in $GF(p^a)^*$, satisfying ([24]):

$$G(D)H^T(D) = 0 \quad (2.54)$$

Theorem 2.10: Let $G(D) = [I_k, P(D)]$ be the generator-polynomial matrix of an (n, k, m) systematic convolutional code. The parity-check polynomial matrix $H(D)$, associated with $G(D)$, has the general form:

$$H(D) = X(D) \begin{bmatrix} -P^T(D), I_{n-k} \end{bmatrix} \quad (2.55)$$

where $X(D)$ is any nonsingular $(n-k) \times (n-k)$ matrix of polynomials over $GF(p^a)$.

Proof: See Appendix 2.15 (p. 325).

Note 2.13: The parity-check polynomial matrix is not unique, as is obvious from the previous theorem.

Lemma 2.10: A parity-check polynomial matrix, associated with $G(D) = [I_k, P(D)]$, over $GF(2)$, has the form:

$$H(D) = \begin{bmatrix} P^T(D), I_{n-k} \end{bmatrix} \quad (2.56)$$

Proof: It follows from Theorem 2.10, if $X(D) = I_{n-k}$ (I is nonsingular) and noting that $-P(D) = P(D)$ over $GF(2)$.

* p^a = power of a prime.

2.20 PARITY-CHECK MATRIX

The definition given below is a modification of the definition proposed by Blahut [10], for the parity-check matrix.

Definition 2.13: Let G be the generator matrix for an (n,k,m) convolutional code. A *parity-check matrix* is any matrix H determined as following:

$$H \triangleq \lim_{z \rightarrow \infty} [H]_z \quad (2.57)$$

where $[H]_z$ is a $(z+1) \times (z+1)$ matrix of the submatrices $h_{i,j}$ / $0 \leq i \leq z$ & $0 \leq j \leq z$. $h_{i,j}$ are $(n-k) \times n$ matrices with elements in $GF(p^a)$ and $[H]_z$ satisfies the following condition:

$$[G]_z^0 [H]_z^T = 0 \quad / z=0,1,2,\dots \quad (2.58)$$

where $[G]_z^0$ is defined by eqn (2.25c).

Theorem 2.11: A parity-check matrix H , for an (n,k,m) systematic convolutional code, has the following form*:

$$H = \begin{bmatrix} P_0^T & -I & & & & & \\ P_1^T & 0 & P_0^T & -I & & & \\ P_2^T & 0 & P_1^T & 0 & P_0^T & -I & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ P_m^T & 0 & P_{m-1}^T & 0 & P_{m-2}^T & 0 & \dots & P_0^T & -I & \dots \\ & & P_m^T & 0 & P_{m-1}^T & 0 & \dots & P_1^T & 0 & \dots \\ & & & P_m^T & 0 & \dots & P_2^T & 0 & \dots \\ & & & & \vdots & \vdots & & & \\ & & & & \vdots & \vdots & & & \end{bmatrix} \quad (2.59)$$

where P_i / $i=0,1,\dots,m$ are $k \times (n-k)$ submatrices defined by eqn (2.44), I is the $(n-k) \times (n-k)$ identity matrix & 0 is the $(n-k) \times (n-k)$ all-zero matrix; the elements of all submatrices belong to $GF(p^a)$.

* Non-binary codes, are now considered.

Proof: It can be easily verified that $GH^T = 0$; G is given by eqn (2.45).

Theorem 2.12: Let G be the generator matrix for an (n,k,m) systematic convolutional code. A parity-check matrix associated with G has the following form:

$$H \triangleq \lim_{z \rightarrow 1} [H]_z$$

where

$$[H]_0 = [-P_0^T, I_{n-k}] \quad (2.60a)$$

$$[H]_z = \begin{bmatrix} [H]_{z-1} & 0 \\ R_z & [H]_0 \end{bmatrix} \quad z \geq 1 \quad (2.60b)$$

$$R_z = -[P_z^T, 0, P_{z-1}^T, 0, \dots, P_1^T, 0] \quad (2.60c)$$

and

$$\begin{array}{l} G_0 = [I_k, P_0] \\ G_i = [0, P_i] \quad / 1 \leq i \leq m \\ P_i = 0 \quad / i > m \end{array} \quad \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \rightarrow \quad (2.60d)$$

Proof: See Appendix 2.16 (p. 326).

2.21 SYNDROME

Let u and v denote the semi-infinite source and channel sequences, respectively [see eqns (2.15) & (2.16)]. Consider the following semi-infinite sequences (see also Fig. 1.3a):

$$\text{Received sequence: } r \triangleq [r_0, r_1, r_2, \dots] \quad (2.61a)$$

$$\text{where } r_i \triangleq (r_i^{(1)} \ r_i^{(2)} \ \dots \ r_i^{(n)}) \quad (2.61b)$$

$$\text{Channel-error sequence: } e \triangleq [e_0, e_1, e_2, \dots] \quad (2.62a)$$

$$\text{where } e_i \triangleq (e_i^{(1)} \ e_i^{(2)} \ \dots \ e_i^{(n)}) \quad (2.62b)$$

$$\text{Estimated information sequence: } \tilde{u} \triangleq [\tilde{u}_0, \tilde{u}_1, \tilde{u}_2, \dots] \quad (2.63)$$

Note that if the transmitted sequence v is corrupted by additive noise, only,

$$r = v + e \quad (2.64)$$

Definition 2.14: The syndrome sequence

$$s \triangleq [s_0, s_1, s_2, \dots] \quad /s_i \triangleq (s_i^{(1)} \ s_i^{(2)} \ \dots \ s_i^{(n-k)}) \quad (2.65a)$$

$$\text{is defined by:} \quad s \triangleq rH^T \quad (2.65b)$$

Theorem 2.13: If e is the error sequence, in a channel with additive noise, then:

$$s = eH^T \quad (2.66)$$

Proof: From Theorem 2.4, $v = uG \implies vH^T = (uG)H^T = u(GH^T)$

$$[\text{using Definition 2.13}] \implies vH^T = 0 \quad (2.67)$$

$$\begin{aligned} \text{If the channel suffers from additive noise: } s &= rH^T = \\ &= (v+e)H^T \quad [\text{by (2.64) \& (2.65)}] \implies s = vH^T + eH^T \end{aligned}$$

$$[\text{and using (2.67)}] \implies s = eH^T$$

QED

The above results can also be given in polynomial notation. Let:

Received polynomial:

$$R(D) \triangleq [r^{(1)}(D), r^{(2)}(D), \dots, r^{(n)}(D)] \quad (2.68a)$$

$$\text{where} \quad r^{(i)}(D) \triangleq r_0^{(i)} + r_1^{(i)}D + r_2^{(i)}D^2 + \dots \quad (2.68b)$$

Channel-error polynomial:

$$E(D) \triangleq [e^{(1)}(D), e^{(2)}(D), \dots, e^{(n)}(D)] \quad (2.69a)$$

$$\text{where} \quad e^{(i)}(D) \triangleq e_0^{(i)} + e_1^{(i)}D + e_2^{(i)}D^2 + \dots \quad (2.69b)$$

$V(D)$ & $U(D)$ are defined by (2.41).

If the noise in the channel is additive, then

$$R(D) = V(D) + E(D) \quad (2.70)$$

Following Definition 2.14, one may write:

$$S(D) = R(D)H^T(D) \quad (2.71)$$

Note that $R(D)$ is a $1 \times n$ vector and $H(D)$ is an $(n-k) \times n$ matrix, so $S(D)$ is a $1 \times (n-k)$ vector of polynomials:

Syndrome polynomial:

$$S(D) \triangleq [s^{(1)}(D), s^{(2)}(D), \dots, s^{(n)}(D)] \quad (2.72a)$$

$$\text{where} \quad s^{(i)}(D) \triangleq s_0^{(i)} + s_1^{(i)}D + s_2^{(i)}D^2 + \dots \quad (2.72b)$$

Theorem 2.14: In a channel with additive noise,

$$S(D) = E(D)H^T(D) \quad (2.73)$$

Proof: Similar to Theorem 2.13. ■

Lemma 2.11: Let $H(D) = [-P(D), I_{n-k}]$ be the parity-check polynomial matrix of an (n, k, m) systematic convolutional code. Then:

$$S(D) = R^p(D) - R^m(D)P(D) = E^p(D) - E^m(D)P(D) \quad (2.74a)$$

$$\text{where:} \quad R^m(D) \triangleq [r^{(1)}(D), r^{(2)}(D), \dots, r^{(k)}(D)] \quad (2.74b)$$

$$R^p(D) \triangleq [r^{(k+1)}(D), r^{(k+2)}(D), \dots, r^{(n)}(D)] \quad (2.74c)$$

$$E^m(D) \triangleq [e^{(1)}(D), e^{(2)}(D), \dots, e^{(k)}(D)] \quad (2.74d)$$

$$\text{and} \quad E^p(D) \triangleq [e^{(k+1)}(D), e^{(k+2)}(D), \dots, e^{(n)}(D)] \quad (2.74e)$$

Proof: The result, above, follows easily from eqns (2.70) & (2.71). ■

Consider the product, $[r^{(1)}(D), r^{(2)}(D), \dots, r^{(k)}(D)]P(D)$.

Using the partition of $P(D)$ into the polynomials $g_{k+j}^{(i)}(D)$ ($i=1, 2, \dots, k$ & $j=1, 2, \dots, n-k$) [see eqn (2.51)], one easily arrives at the following:

Lemma 2.12: For an (n, k, m) systematic convolutional code, the j th $/j=1, 2, \dots, n-k$ syndrome polynomial is given by:

$$s^{(j)}(D) = r^{(k+j)}(D) - \sum_{i=1}^k r^{(i)}(D) g_{k+j}^{(i)}(D) \quad (2.75a)$$

$$s^{(j)}(D) = e^{(k+j)}(D) - \sum_{i=1}^k e^{(i)}(D) g_{k+j}^{(i)}(D) \quad (2.75b)$$

It would be useful to obtain the result of Lemma 2.12 in non-polynomial form. This can be achieved by replacing the polynomials in D , in relation (2.75), with the corresponding sums of terms:

$$x(D) = \sum_{i=0}^{+\infty} x_i D^i$$

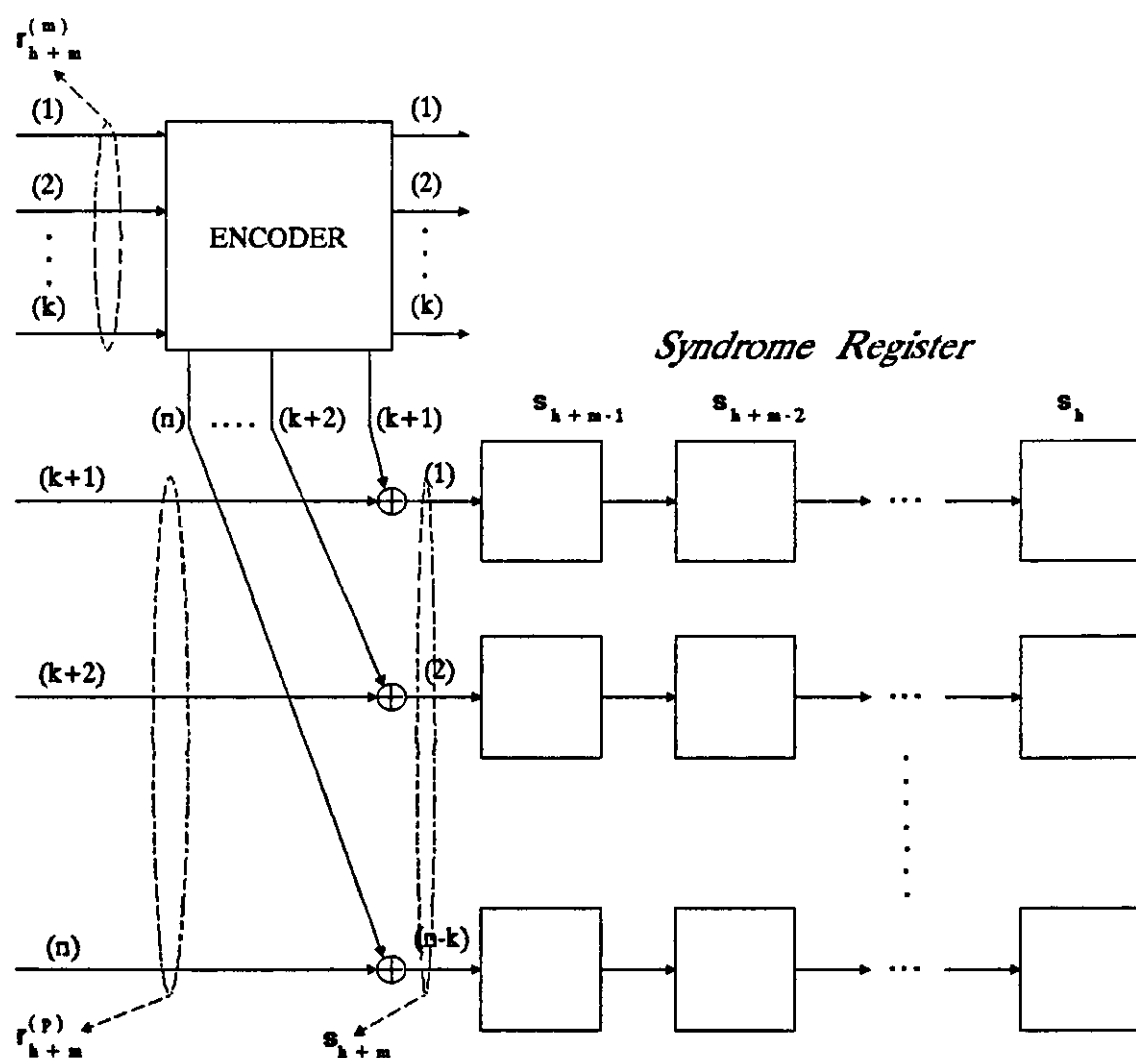


Figure 2.6: The syndrome portion of a decoder for an (n, k, m) binary systematic convolutional code.

Theorem 2.15: For an (n,k,m) systematic convolutional code, the j th syndrome digit of the h th received block is given by:

$$s_h^{(j)} = r_h^{(k+j)} - \sum_{z=0}^{\theta} \sum_{i=1}^k r_{h-z}^{(i)} g_{k+j,z}^{(i)} \quad / 1 \leq j \leq n-k \text{ \& } h \geq 0 \quad (2.76a)$$

$$s_h^{(j)} = e_h^{(k+j)} - \sum_{z=0}^{\theta} \sum_{i=1}^k e_{h-z}^{(i)} g_{k+j,z}^{(i)} \quad / 1 \leq j \leq n-k \text{ \& } h \geq 0 \quad (2.76b)$$

where $\theta \triangleq \text{MIN}\{h, m\}$.

Proof: See Appendix 2.17 (p. 328). ■

If one compares eqn (2.47) with eqn (2.76), one readily concludes that the syndrome bits can be calculated by re-encoding the received message bits and adding the last $n-k$ bits of the re-encoded received block, to the $n-k$ received parity-check bits, as shown in Fig. 2.6.

2.22 CONCLUSIONS

This chapter contains the *theory of convolutional codes* (CCs). In Section 2.1, an (n,k,m) CC was defined as an error-correcting code whose current channel block depends on the current and the past m message blocks. In other words, a CC is a *generalized block code*. Furthermore, it was argued that an encoder, for a CC, may be implemented with a k -in, n -out, linear sequential circuit (LSC), made of up to k shift registers and up to n X-OR gates (Section 2.2). Such an LSC is completely described by a set of kn $(m+1)$ -tuples, called the *generator sequences* (Sec. 2.7), which are the impulse responses of the circuit.

The next logical step was to relate the O/P with the I/P, via the set of generator sequences. To this end, the concept of *convolution* was used justifying, thus, the name of the codes. In particular, the O/P of the encoder's j th port

equals the sum of the I/P sequences, each *convoluted* with the corresponding generator sequence [see eqn (2.11)].

CC literature uses either the *matrix*, or the *polynomial*, (or both) approach(es) to the code-theory. Appendix 2.4 illustrates (in an original way) the difference between these two techniques so that the reader may grasp the reason of their existence and their advantages. In the matrix approach, the digits are grouped per block time-unit, while in the polynomial approach are grouped per port. Thus, the former technique uses semi-infinite dimensioned matrices of submatrices, while the latter uses finite matrices of polynomials in D^* of infinite degree. Note that this awkward 'infinite' arises because a CC encoder (unlike a block one) generates only one codeword (see Sec. 2.6) of infinite length, when it operates continuously.

Another piece of original work was developed in Sec. 2.11. Eqn (2.26) relates an arbitrary, but finite, portion of the encoder O/P (made of blocks h to $h+z$) to the corresponding part of the input sequence (made of blocks $h-\min\{h,m\}$ to $h+z$) and a finite-dimensioned 'generator matrix' $[G]_z^h$. A number of special cases are also examined and, finally, the *generator matrix* G is defined to be the limit of $[G]_z^h$, as $z \rightarrow +\infty$. The importance of $[G]_z^h$ lies in its flexibility. It is a generalization of previous work, but it also permits a simple treatment of special cases.

The corresponding polynomial expressions are easily translated from the matrix ones. Furthermore, the *generator-polynomial matrix*, $G(D)$ (also called *transfer-function matrix* - with good reason), is linked to what was called *normal encoder* (Sec. 2.14). Usually, the normal is also the minimal encoder ('exponentially' important for trellis decoding). Occasionally, though, the *type-II encoder* is the minimal one, especially if high powers of D concentrate along relatively few columns of $G(D)$ (Sec. 2.18).

The results of Sections 2.1-2.16 are simplified for the case of *systematic codes* (Sec. 2.17). The most important result is the form of the polynomial-generator matrix, $G(D) = [I_k, P(D)]$.

* D is the delay operator.

The *parity-check polynomial matrix* (Sec. 2.19) is defined by $G(D)H^T(D) = 0$. It is not unique and, for systematic codes, it has the form $H(D) = X(D)[-P^T(D), I_{n-k}]$, where $X(D)$ is any nonsingular matrix.

The *parity-check matrix* (Sec. 2.20), H , is defined as the limit of $[H]_z$, as $z \rightarrow +\infty$, where $[G]_z^0[H]_z^T = 0$ / $z \geq 0$. This constitutes an effort to avoid again infinite-dimensional matrices. Theorem 2.12 shows that $[H]_z$ may be defined by a recursive matrix equation. Both the idea of $[H]_z$ and the theorem, are original work.

Finally, Sec. 2.21 defines the *syndrome* as the product of the received sequence with the transpose of the parity-check matrix. It is proved that, if noise is additive, the syndrome digits are linear combinations of the channel-error digits only. This is central to the theory of CC for threshold decoding (see Chapters 5, 6 & 7). It is also shown that, for binary systematic codes, the syndrome is formed by adding to the received parity-bits, the parity-bits corresponding to the received message-bits (see Fig. 2.6).

CHAPTER 3

Decoding of

Convolutional Codes

Chapter 3 has been designed to serve as a brief presentation of decoding techniques for convolutional codes.

Choice of material and emphasis are again determined by the orientation of the thesis towards syndrome decoding and perhaps by the subjective opinion of the author.

Since the material in this chapter mainly helps to bridge the gap between convolutional code structure and syndrome decoding, the presentation has a tutorial flavour.

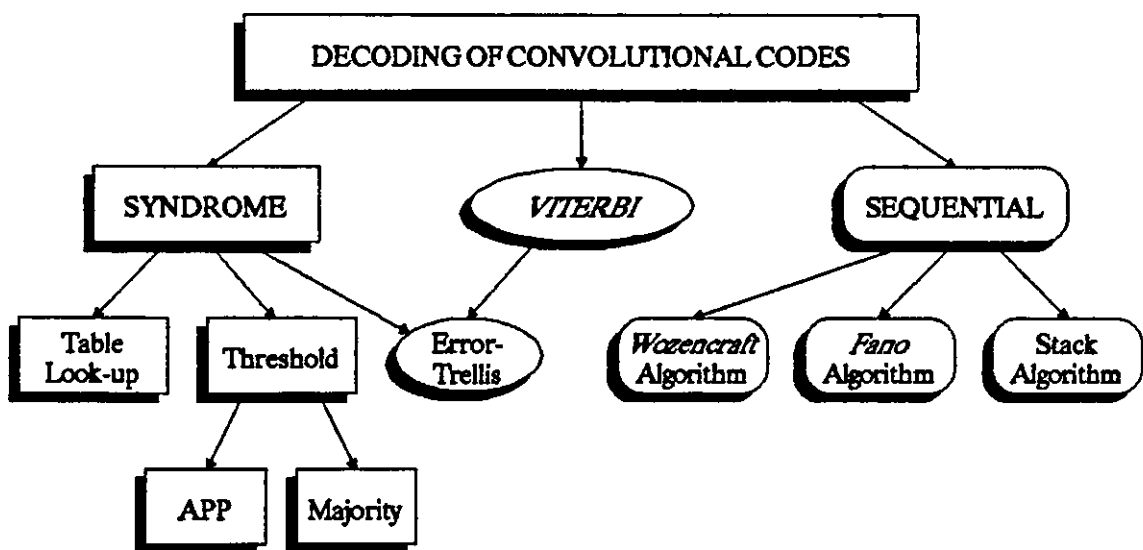


Figure 3.1: Classification of decoding techniques for convolutional codes.

Figure 3.1 illustrates the relation among the various decoding techniques for convolutional codes. Chapter 3 contains six sections: Introduction, convolutional encoder state-transition diagram, trellis diagram, Viterbi decoding, sequential decoding & syndrome decoding. Of these, greater attention is paid to the structure of the transition and trellis diagrams. In spite of the tutorial flavour of this chapter, some new (or at least originally presented) results were obtained during the consideration of the convolutional encoder transition and trellis diagrams.

3.1 INTRODUCTION

Convolutional decoding techniques have been studied for many years. About three years after the introduction of convolutional codes (CCs) (by P. Elias, in 1954), Wozencraft devised sequential decoding; this is, in effect, a trial and error search-decoding technique, of variable duration, which offers very good performance, but at that time it would have required a very costly implementation. To overcome this disadvantage of sequential decoding, researchers devised other techniques grouped under the name *syndrome decoding*. Generally, a set of syndrome equations is obtained which is then used to provide an estimate of each channel error-digit (each syndrome digit equals a specific linear combination of channel error-digits - see Theorem 2.15, p. 50). Syndrome decoding are suboptimum, easily implementable, techniques; they are also deterministic since each block is decoded within a computational cycle.

A Viterbi decoder is, in effect, a maximum likelihood decoder for convolutional codes that adopts a clever optimization strategy which minimizes the number of computations and/or memory cells required. It was introduced in 1967 by Andrew Viterbi and has since become one of the most widely-used decoding techniques.

The above mentioned techniques will be introduced briefly, in this chapter. Some of these will be studied further,

in subsequent chapters. Firstly, though, the encoder transition & trellis diagrams will be extensively discussed in Sections 3.1 & 3.2; some new results concerning the structure of these two diagrams are included in these sections.

The reader will notice that the material that follows discusses only the normal* encoder (and not the type-II one). This is so because for most codes the normal is also the minimal encoder and because most results are valid for both implementations (if not, all that is usually needed is a simple modification).

3.2 CONVOLUTIONAL ENCODER STATE-TRANSITION DIAGRAM

As Forney [25] wrote, convolutional codes may be easier to decode but they are harder to analyse, compared to block codes. The reason seems to be their complex structure. This forced many researchers to propose various approaches to the problem of convolutional code analysis. One of them is the *finite-state machine* approach.

3.2.1. Introduction

Consider a binary (n, k, m) convolutional code and the normal encoder* realization. Let M_i = length of the i th shift register ($1 \leq i \leq k$). Obviously, $0 \leq M_i \leq m$, for $i=1, 2, \dots, k$ and:

$$\text{Total Encoder Memory} \triangleq M = \sum_{i=1}^k M_i \quad (3.1)$$

The state of the encoder is determined by the contents of its memory, which in turn is a selection of its past m inputs (an input is a k -tuple). Using standard notation, the contents of the encoder memory (and hence the *state of the encoder*), at time-unit h , is

$$S(h) \triangleq \left[u_{h-M_1}^{(k)} \dots u_{h-2}^{(k)} u_{h-1}^{(k)} \dots u_{h-M_2}^{(2)} \dots u_{h-2}^{(2)} u_{h-1}^{(2)} \dots u_{h-M_1}^{(1)} \dots u_{h-2}^{(1)} u_{h-1}^{(1)} \right] \quad (3.2)$$

where, $u_{h-M_i}^{(i)}$ exists if, and only if, $M_i \geq 1 \quad / 1 \leq i \leq k$.

* See Definition 2.9 & Note 2.9, p. 34.

It is usually written, $S(h) = S_j$, where S_j is the symbol identifying the particular state. Normally, j is the decimal equivalent of the M -tuple in (3.2).

If q is the number of code symbols (usually, $q=2$):

$$\text{Total number of states} = q^M \quad (3.3)$$

Example 3.1: Consider the encoder of Example A2.9.1 (p. 314). It is a normal encoder for a $(3,2,1)$ code. It is made of 2 shift registers of length 1 ($M_1=M_2=1$). Hence its total memory is $M=2$ and it has $2^2=4$ states. If $S(h) = [BA]$, then: $S_0 = [00]$, $S_1 = [01]$, $S_2 = [10]$ & $S_3 = [11]$. ■

Example 3.2: Consider the encoder of Example A2.9.2 (p. 315). It is a normal encoder for a $(4,3,2)$ code. It is made of 3 shift registers with lengths $M_1=0$, $M_2=1$ & $M_3=2$. Hence, its total memory is $M=0+1+2=3$ and it has $2^3=8$ states. If $S(h) = [CBA]$, then: $S_0 = [000]$, $S_1 = [001]$, $S_2 = [010]$, $S_3 = [011]$, $S_4 = [100]$, $S_5 = [101]$, $S_6 = [110]$ & $S_7 = [111]$. ■

A state-change takes place iff at least one of the memory cells of the encoder changes content [as is evident from reln (3.2)]. Under normal operation this happens only when a new input is applied at the encoder; this causes the contents of each SR to be clocked one position towards the O/P, to make space for the new I/P. So, the current state of a convolutional encoder (and any general LSC) can be determined by its previous state and the current input.

Assume that the encoder is currently at state: $S(h)=S_c$, where $0 \leq c \leq q^M-1$. An interstate transition is caused when the current input u_h is clocked into the encoder. Consequently, given the current state, the next one depends entirely on the current input; it follows then, that to each transition there corresponds a unique group* of input blocks.

3.2.2. Sequential Machines

The discussion above indicates that a convolutional en-

* Usually a single-element group.

coder should be treated as a sequential finite-state machine; furthermore, the analysis of the encoder should be assisted by the results that have been obtained in the field of Sequential Machines and Automata Theory.

Appendix 3.1 (p. 329), contains some definitions and examples on state-transition diagrams.

3.2.3. Structure of the Encoder State-Transition Diagram

The following results relate the code parameters with a few state-transition diagram details, like number of states, number of transitions per state, multiplicity of transitions (multiple-edge transitions), etc. The analysis that follows considers only the *normal encoder*. The difference between the normal and the type-II encoders lies in the fact that the latter's SRs are interspersed with gates; in this way, the memory contents are not just shifted but also altered, with each block time-unit. Nevertheless, the analysis that follows needs a few minor (hopefully and most probably) modifications in order to be able to describe the behaviour of any general LSC.

Theorem 3.1: Consider an (n,k,m) normal* encoder with generator-polynomial matrix $G(D)$. Let $f[G(D)]$ denote the number of rows of $G(D)$ that contain only 1s and 0s. Then, the number of encoder SRs of length $M_1 > 0$ is $k - f[G(D)]$.

Proof: See Appendix 3.2 (§ A3.2.1., p. 333). ■

From above:

$$f[G(D)] = f = \text{Number of zero-length shift registers} \quad (3.4)$$

For instance, in Example A2.9.1 (p. 314), $f = 0$, while in Example A2.9.2 (p. 315), $f = 1$ [the 1st row of $G(D)$ contains only 'ones'].

The following definition will help to prove & discuss the basic results of the current and the next chapter. The concepts that will be introduced have a dual meaning; they can either be taken to mean parts of the physical or the logical

* See Definition 2.9 & Note 2.9.

memory of an encoder (and in general of any LSC), or sets of variables. The general term *group* will be used throughout and it will stand either for a set of specific and fully identifiable physical or logical SR-stages, or for a set of specific variables that represent digits from the message sequence u (which digits reside in the above-mentioned memory parts). The two concepts meet because an SR-stage can be identified by $u_{h-j}^{(i)}$ where, i is the SR number ($1 \leq i \leq k$), j is the stage-number within the i th SR ($1 \leq j \leq M_i$) and h is the reference time-unit.

Definition 3.1: Consider the memory of an encoder; let the term *memory group (MEG)* denote the set of its physical or logical SR-stages. The following subsets of MEG are introduced: The *front-end group (FEG)* contains the 1st stage of each SR; the *rear-end group (REG)* contains the last stage of each SR; the *central group (CEG)* contains the stages that do not belong to either the FEG or the REG (see Fig. 3.2). The above-mentioned terms are defined analytically below; note that this time they are given as functions of h , the reference time-unit:

$$\text{MEG}(h) \triangleq \{u_{h-j}^{(i)} / i=1,2,\dots,k: M_i \geq 1 \text{ \& } j=1,2,\dots,M_i\} \quad (3.5a)$$

$$\text{FEG}(h) \triangleq \{u_{h-1}^{(i)} / i=1,2,\dots,k: M_i \geq 1\} \quad (3.5b)$$

$$\text{REG}(h) \triangleq \{u_{h-M_i}^{(i)} / i=1,2,\dots,k: M_i \geq 1\} \quad (3.5c)$$

$$\text{CEG}(h) \triangleq \{u_{h-j}^{(i)} / i=1,2,\dots,k: M_i \geq 3 \text{ \& } j=2,3,\dots,M_i-1\} \quad (3.5d)$$

Notation: The 'groups', defined above, will be denoted without the reference time if they are taken to mean memory parts, while inclusion of the time variable will denote sets of variables.

The various groups, defined above, are sets of SR-stages. The terms to be introduced below are ordered sequences of a finite number, say, x , of components. These x -tuples represent the state of the various groups defined above, at a given time-unit. For example, the state of MEG is simply

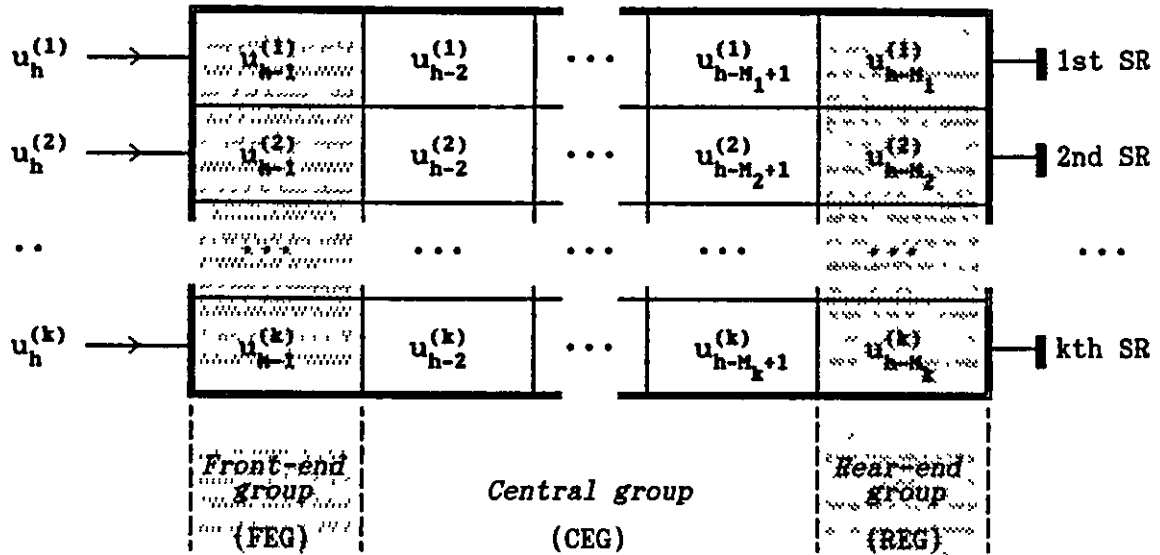
Input
block*Encoder contents*

Figure 3.2: The various groups of digits that influence the state of the encoder (note that the i th SR exists iff $M_i > 0$ / $1 \leq i \leq k$).

$S(h) = S_j$, where $j \in [0, q^M)$ [see eqn (3.2)]. Similarly, and using standard notation, the state of the FEG, CEG & REG at time-unit h , is denoted by $F(h)$, $C(h)$ & $R(h)$, respectively:

$$\text{If } S(h) = S_j \text{ then: } F(h) \triangleq [u_{h-1}^{(k)} \dots u_{h-1}^{(2)} u_{h-1}^{(1)}] = F_j \quad (3.6a)$$

where, $u_{h-1}^{(i)}$ is included iff $M_i \geq 1$ / $i=1, 2, \dots, k$.

$$\text{If } S(h) = S_j \text{ then: } R(h) \triangleq [u_{h-M_1}^{(k)} \dots u_{h-M_1}^{(2)} u_{h-M_1}^{(1)}] = R_j \quad (3.6b)$$

where, $u_{h-M_1}^{(i)}$ is included iff $M_i \geq 1$ / $i=1, 2, \dots, k$.

If $S(h) = S_j$, then $C(h) = C_j$, where:

(3.6c)

$$C(h) \triangleq [u_{h-M_1+1}^{(k)} \dots u_{h-M_1+1}^{(k)} u_{h-M_1+1}^{(k)} \dots u_{h-M_1+1}^{(2)} \dots u_{h-M_1+1}^{(2)} u_{h-M_1+1}^{(2)} \dots u_{h-M_1+1}^{(1)} \dots u_{h-M_1+1}^{(1)} u_{h-M_1+1}^{(1)}]$$

and, $u_{h-M_1+1}^{(i)}$ / $a_i \in (1, M_i)$ is included iff $M_i \geq 3$ / $i=1, 2, \dots, k$.

Note for instance that in the encoder of Fig. A2.9.1 (p. 314), $FEG = \{A, B\} = REG$ and $CEG = \emptyset$. For the encoder of Fig. A2.9.2 (p. 315), $FEG = \{A, B\}$, $REG = \{A, C\}$ and $CEG = \emptyset$. For the encoder of Fig. A2.9.3 (p. 316), $FEG = \{A, C\}$, $REG =$

$\{B,D\}$ and $CEG = \emptyset$.

Theorem 3.2: Consider a q -ary (n,k,m) normal encoder* with generator-polynomial matrix $G(D)$. With respect to its state-transition diagram, the number of transitions leaving a state equals the number of transitions entering that state and both are equal to q^{k-f} . Furthermore, each transition has q^f labels, i.e. it represents q^f input (and output) block(s).

Proof: See Appendix 3.2 (§ A3.2.2., p. 334). ■

Consider the encoders of Figs A2.9.1 (p. 314) & A2.9.2 (p. 315). Both are binary, hence $q=2$. The 1st has $f = 0$ & $k = 2$, hence according to Theorem 3.2, in its transition diagram $q^{k-f} = 2^{2-0} = 4$ transitions should leave each state. This can be verified from Fig. A3.1.1 (p. 331). The 2nd encoder has $f = 1$ & $k = 3$, hence $2^{3-1} = 4$ transitions leave each state (see Fig. A3.1.2, p. 332).

There is a disagreement in the literature about the representation of convolutional encoders. Some authors adopt the Mealey representation, some others the Moore one. Furthermore, there seem to be variations of the Mealey model. Specifically, some authors, although they adopt the Mealey model, they do not use the minimal-memory approach; they seem to reject the notion of SRs of length 0 [i.e. they set $M_i = \text{MAX}\{1, M_i\}$]. In such a case, the modified encoder will contain f SRs of length 1, whose O/P is not used (see for example, Fig. 4.2 in [26]), while total memory will be increased by f , hence total number of states will be increased q^f -fold. In such a case, each transition will represent a single I/P (& O/P) block (single-edge transitions).

Another group of authors employ the Moore model. Specifically, they add an extra stage at the beginning of each SR (i.e. they set $M_i = 1 + M_i$), so that the O/P gates are fed only from the encoder memory (see for example, Figures 6.1 & 6.6 in Clark & Cain [13] and Figures 5.3.1, 5.3.2, 5.3.3 & 5.3.10 in Proakis [27]). Nevertheless, when it comes to the transition diagram these authors use the minimum-memory en-

* A q -ary encoder is made of q -ary SR-stages and $GF(q)$ gates. See also, Definition 2.9.

coder realization; for instance, Proakis [27], uses a 4-state transition diagram for an encoder with total memory 4.

In this thesis, the Mealey-model minimum-memory approach has been adopted. It is the approach suggested by a great number of authors (see for example, Section II in Forney [20], Figure 1 in Forney [21], Figure 12.9 in Blahut [10], pp. 288-305 in Lin & Costello [2], etc). Its disadvantage is the introduction of the variable f (the number of SRs of length 0), while the definite advantage is the adoption of the minimum-memory encoder. Note that in both the Mealey models, mentioned above, the product "(number of transitions from any specific state) \times (number of I/P blocks that cause this transition)" is q^k .

Some more results, on the structure of the encoder state-transition diagram, will be presented:

Theorem 3.3: Consider the state-transition diagram of a q -ary (n,k,m) normal encoder*, with generator-polynomial matrix $G(D)$. Each of the q^{k-f} transitions that enter any specific state is generated by the same group of q^f I/P k -tuples.

Proof: See Appendix 3.2 (§ A3.2.3., p. 335). ■

Consider the transition diagram of Fig. A3.1.1 (p. 331). Only one I/P block causes a transition into a specific state (00 leads to S_0 , 10 leads into S_1 , etc); note that $f = 0$, for this code, and so $q^f = 1$.

In the transition diagram of Fig. A3.1.2 (p. 332), $f = 1$ & $k = 3$. The 4 transitions that enter any specific state are caused by the same group of $q^f = 2^1 = 2$, I/P 3-tuples. For example, the two I/P blocks that lead into state S_5 are 010 & 110; to state it otherwise, each of the 4 transitions that lead into S_5 has a double-edge label with I/P part $x10$, where $x=0,1$.

According to the above theorem, the number of ways one may arrive into any particular state is $(q^{k-f}) \times (q^f) = q^k$. So, one may arrive at S_5 (of Fig. A3.1.2, p. 332), from S_2

* A q -ary encoder is made of q -ary SR-stages and GF(q) gates. See also, Definition 2.9.

with I/Ps 010 & 110, from S_3 with I/Ps 010 & 110, from S_6 with I/Ps 010 & 110 and from S_7 with I/Ps 010 & 110.

Theorem 3.4: Consider the state-transition diagram of a q -ary (n,k,m) normal encoder*. A state S_g may undergo a one time-unit transition to itself (a self-loop in the transition diagram) if, and only if, the M digits that constitute state S_g satisfy the following restriction:

$$S(h) = S(h+1) \quad \langle \text{---} \rangle$$

$$u_{h-j}^{(i)} = c_i \quad / \text{all } j \in [0, M_i] \quad \& \quad \text{all } i \in [1, k] : M_i \geq 1 \quad (3.7)$$

where c_i , is a constant [an element of $GF(q)$].

Proof: Let z_s represent the encoder contents before the transition and w_s after the transition. Consider two neighbouring digits, $u_{h-a}^{(i)}$ & $u_{h-a-1}^{(i)}$ ($0 \leq a < a+1 \leq M_i$), in the i th SR. After any kind of transition, and for all $i \in [1, k] / M_i \geq 1$,

$$w_{h-a-1}^{(i)} = z_{h-a}^{(i)}, \quad \text{for all } a \in [0, M_i - 1] \quad (A)$$

i) Let $S(h) = S(h+1) = S_g$. Then,

$$w_{h-a-1}^{(i)} = z_{h-a-1}^{(i)} \quad \text{for all } a \in [0, M_i - 1] \quad (B)$$

From relations (A) & (B),

$$z_{h-a}^{(i)} = z_{h-a-1}^{(i)} \quad \text{for all } a \in [0, M_i - 1] \quad \text{---} \rangle$$

$$\text{---} \rangle \quad z_h^{(i)} = z_{h-1}^{(i)} = \dots = z_{h-M_i+1}^{(i)} = z_{h-M_i}^{(i)} = \text{constant} = c_i, \text{ say.}$$

ii) Let condition (3.7) hold true. Consider a state transition. From (3.7), provided that $0 \leq a < a+1 \leq M_i \text{ ---} \rangle$ $0 \leq a < M_i$:

$$z_{h-a}^{(i)} = z_{h-a-1}^{(i)} \quad \text{for all } a \in [0, M_i - 1]$$

and combining with (A):

$$w_{h-a-1}^{(i)} = z_{h-a-1}^{(i)} \quad \text{for all } a \in [0, M_i - 1]$$

$$\text{---} \rangle \quad w_{h-a}^{(i)} = z_{h-a}^{(i)} \quad \text{for all } a \in [1, M_i] \quad (\text{and all } i \in [1, k])$$

Hence, the two states are identical.

QED

In non-mathematical language, the above theorem states

* A q -ary encoder is made of q -ary SR-stages and $GF(q)$ gates. See also, Definition 2.9.

that a specific state S_g has a self-loop if, and only if, the digits that constitute this state [see eqn (3.2)], when organized in the encoder memory, are such that all stages of any specific SR contain the same digit. Furthermore, if S_g has a self-loop, then it will occur if, and only if, the digits of the current input block are identical with the contents of the SR they will reside in (excluding any digits that will not reside in an SR, because they correspond to a port whose SR does not exist).

An encoder with SRs of length 1 (like that of Fig. A2.9.1, p. 314), has a state diagram where all states have a self-loop.

For the encoder of Fig. A2.9.2 (p. 315), with one SR of length 1 (stage A) and one SR of length 2 (stages B & C), the states with self-loops have the format [xxy], where x & y are binary variables (note that the state is [CBA]). Hence there are four states with self-loop, corresponding to the four pairs x,y (000,001,110,111, i.e. S_0, S_1, S_6 & S_7), as can be verified by Fig. A3.1.2 (p. 332).

Theorem 3.5: Consider the state-transition diagram of a q-ary (n,k,m) normal encoder*, with generator-polynomial matrix $G(D)$. The I/P k-tuples that mark the self-loops of the transition diagram are identical with the front-end group (FEG) of the encoder in the corresponding k-f positions. Hence, the format of the I/P k-tuple $u_h = [u_h^{(1)}, u_h^{(2)}, \dots, u_h^{(k)}]$ that marks a self-loop is given by the relation,

$$u_h^{(i)} \rightarrow \begin{cases} \text{arbitrary} & \text{if } i \in [1,k] : M_i = 0 \\ u_{h-1}^{(i)} & \text{if } i \in [1,k] : M_i \geq 1 \end{cases} \quad (3.8)$$

Proof: Let us represent the encoder contents before a transition and w_s after a transition. Assume that state S_g goes through a self-loop. Then, for all $i \in [1,k] / M_i \geq 1$,

$$u_{h-1}^{(i)} = w_{h-1}^{(i)} \quad (A)$$

Also, since during a transition, I/P block \rightarrow FEG:

* A q-ary encoder is made of q-ary SR-stages and GF(q) gates. See also, Definition 2.9.

$$w_{h-1}^{(i)} = u_h^{(i)} \quad (B)$$

From eqns (A) & (B), it follows that

$$u_h^{(i)} = u_{h-1}^{(i)} \quad \text{for all } i \in [1, k] : M_i \geq 1.$$

QED

For example, for the encoder of Fig. A2.9.2 (p. 315) (state diagram in Fig. A3.1.2, p. 332), the I/P block that marks a self-loop must have the format $u_{h,1} = (xAB)$, where x is a binary variable and A & B are the current contents of the corresponding stages. So, for state S_1 the I/P block is $(x10)$.

Theorem 3.6: Consider the state-transition diagram of a q -ary (n, k, m) normal encoder*, with generator-polynomial matrix $G(D)$. The number of states with self-loop is q^{k-f} .

Proof: State S_0 is a state with self-loop. In how many ways can one change it and still preserve the self-loop property? From Theorem 3.4, one concludes that this property is preserved iff

$$u_{h-j}^{(i)} = c_i \quad \text{for all } j \in [0, M_i] \quad \& \quad \text{for all } i \in [1, k] : M_i \geq 1$$

Then, since the number of states satisfying the above condition is equal to the number of different combinations of the $k-f$ constants c_i , the theorem is proved.

QED

The trellis diagram is obtained from the transition diagram, if the latter is expanded in time so that there is a separate transition diagram for each time-unit. Obviously, all the theorems proved for the state-transition diagram can be used for the trellis, as well.

3.3 TRELLIS DIAGRAM

Consider a q -ary (n, k, m) normal convolutional encoder* and a message of L blocks. The encoder goes through a specific sequence of states, in response to the kL -digit input

* A q -ary encoder is made of q -ary SR-stages and GF(q) gates. See also, Definition 2.9.

message. The task of the decoder is to reconstruct this time sequence of state transitions, based on the information provided by the received sequence r . The decoder will, in fact, consider a number of possible time sequences of state transitions and choose the most 'promising' candidate. To depict these time sequences of state transitions one would need a time sequence of transition diagrams, i.e. one would need a *trellis diagram*.

It has already been mentioned (see Note 2.1) that a message sequence must always be terminated with m zeros, in order to clear the encoder memory. Hence, the first message block finds the encoder in state S_0 , while the 2nd message block finds the encoder in one of the states that can be reached from S_0 , in one time-unit, and so on. Since the message is terminated with m 0s, $L+m$ blocks are considered, while a time sequence of $L+m+1$ transition diagrams is required, or the same, a trellis of length $L+m+1$. Obviously, at time-unit $L+m+1$, the encoder returns to state S_0 .

According to the discussion above, the following construction is proposed:

Note 3.1: Consider a q -ary (n,k,m) normal convolutional encoder* and a message of L blocks. A trellis for this encoder is made of an array of points interconnected in the following way:

i) The points are arranged on a rectangular grid in a maximum of,

$L+m+1$ columns, labelled $0,1,\dots,L+m$ from left to right and
 q^M rows labelled $0,1,\dots,q^M-1$ from top to bottom
 (L =message length, in blocks).

ii) The points of column j represent the possible encoder states at time-unit j ($0 \leq j \leq L+m$), while the points of row i represent state S_i ($0 \leq i \leq q^M-1$) through time. During the discussion of the trellis diagram the terms *state* and *point* will be interchangeable.

iii) Column 0 contains only one state, S_0 ; column 1

* A q -ary encoder is made of q -ary SR-stages and GF(q) gates. See also, Definition 2.9.

contains those states that can be reached from S_0 , in one time-unit, and so on. Column $L+m$ contains only state S_0 .

iv) The trellis diagram contains exactly q^M states in columns $m, m+1, \dots, L$. This part is called the *central portion of the trellis*.

v) Only states belonging to neighbouring columns are interconnected. In particular, from any state S_x of any specific time-unit i ($0 \leq i < L+m$) originate interconnections to specific states of time-unit $i+1$; these interconnections correspond to all the transitions leaving S_x .

According to Note 3.1 the trellis diagram grows from left to right until it reaches the maximum number of rows (q^M), at time-unit m . This is so because at any time-unit $i < m$ part of the encoder memory is still reset to zero (in fact, the last $m-i$ stages of any SR). The central portion of the trellis will be reached when the whole of the encoder's memory has been 'upset' by the incoming source blocks. An I/P block needs m time-units to cover the length of the longest SR hence part (iv) of Note 3.1, above. Appendix 3.3 (p. 335), gives an example of a simple state-transition diagram with the associated trellis diagram.

The results developed for the normal encoder state-transition diagram will now be applied to the trellis diagram. Lemmas 3.1 & 3.2 below, follow easily from Theorems 3.2 & 3.3, respectively.

Lemma 3.1: Consider a q -ary (n, k, m) normal encoder*, with generator-polynomial matrix $G(D)$. Within the central portion of the encoder trellis-diagram, the number of transitions leaving a state equals the number of transitions entering that state and both are equal to q^{k-f} . Furthermore, each transition represents q^f input (and output) block(s).

Note that the result of Lemma 3.1 is valid only within the central portion of the trellis, and not at the limits

* A q -ary encoder is made of q -ary SR-stages and GF(q) gates. See also, Definition 2.9.

(i.e. at time-units m & L). This is so because at time-unit $m-1$ some states have not been reached, hence the group of transitions from time-unit $m-1$ to time-unit m is not from all possible states to all possible states; this means that, in general, the number of transitions entering a state at time-unit m , are less than the number of transitions leaving it. The situation at time-unit L is symmetrically similar.

Lemma 3.2: Consider a q -ary (n,k,m) normal encoder*, with generator-polynomial matrix $G(D)$. Within the central portion of the encoder trellis-diagram, all the q^{k-f} transitions that enter any specific state are caused by a specific group of q^f source blocks (k -tuples).

Any trellis diagram contains some 'horizontal' lines that extend along particular rows; this happens at least with states S_0 and S_{q^M-1} . These horizontal lines are transitions of a state to itself. Their occurrence is defined by the following lemma, which is based on Theorems 3.4, 3.5 & 3.6.

Lemma 3.3: Consider a q -ary (n,k,m) normal encoder*, with generator-polynomial matrix $G(D)$. In the central portion of its trellis-diagram there exist q^{k-f} horizontal lines (i.e. transitions of a state to itself). These lines correspond to states that satisfy the following condition:

$$u_{h-j}^{(i)} = c_i \quad / \text{all } j \in [0, M_i] \quad \& \quad \text{all } i \in [1, k] : M_i \geq 1 \quad (3.9)$$

where c_i is a constant and $u_{h-j}^{(i)}$ is the content of the j th stage of the i th SR [see Fig. (3.2)].

Also, the I/P k -tuples that mark the horizontal lines of the trellis diagram are identical with the front-end group (FEG) of the encoder in the corresponding $k-f$ positions. The format of the I/P k -tuple, $u_h = [u_h^{(1)}, u_h^{(2)}, \dots, u_h^{(k)}]$, that marks a horizontal line is given by the relation,

$$u_h^{(i)} \begin{cases} \text{arbitrary} & \text{if } i \in [1, k] : M_i = 0 \\ u_{h-1}^{(i)} & \text{if } i \in [1, k] : M_i \geq 1 \end{cases} \quad (3.10)$$

* A q -ary encoder is made of q -ary SR-stages and GF(q) gates. See also, Definition 2.9.

where $u_{h-j}^{(i)}$ is the digit in the j th stage of the i th SR.

3.4 VITERBI DECODING

In 1967, Viterbi [28] introduced a decoding algorithm which he called "A Probabilistic Nonsequential Decoding Algorithm" but which is since known by his name. Two years later, Omura suggested that "the Viterbi Algorithm can be thought of as a forward dynamic programming solution to a generalized regulator control problem" [29] or, as Lin & Costello put it, "...to the problem of finding the shortest path through a weighted graph" [2]. In 1974, Forney [25] wrote: "Convolutional codes are characterized by a trellis structure. Maximum likelihood decoding is characterized as the finding of the shortest path through the code trellis, an efficient solution for which, is the Viterbi Algorithm".

The encoder's sequence of states, during the encoding of a kL -digit long information sequence, is represented by a path in the trellis. The decoder attempts to retrace this path, using the received sequence r . This sequence is processed in blocks, but final decisions are delayed until the end of the sequence.

Definition 3.2: A path in the trellis is a time sequence of states that is represented by a sequence of message or channel digits. Note that in case $f > 0^*$ a sequence of two states corresponds to q^f paths, hence a path cannot be uniquely represented by the time sequence of states alone (see Theorem 3.2). A path that starts at time-unit j and ends at time-unit i , is said to have length $i-j$. A single path is any path of length 1.

Definition 3.3: Consider a path characterized by the sequence of channel digits v_j and let r_j be the corresponding sequence of received digits. Then, the log-likelihood function $\log P(r_j | v_j)$ [where $P(r_j | v_j)$ is the conditional prob-

* See Theorem 3.1, for a definition of f .

ability that r_j is received, given that v_j was transmitted], is called the *metric* of the path. The metric of a single path is called the *branch metric*.

Theorem 3.7: Consider the *Viterbi algorithm*:

Step 1: At time-unit $j=m$ the decoder examines all paths that lead to each of the 2^M states. Each path corresponds to a particular channel sequence v which is compared with the received sequence r ; a metric is computed for each path.

The metrics of all paths entering each state are compared and the path with the largest metric (the *survivor*) is stored, together with its metric. This is repeated for all states.

Step 2: Increase j by 1. For each state and for each single path entering this state, compare the received block r_j with the channel block v_j that corresponds to this particular path; deduce the branch metric $\log P(r_j|v_j)$.

Add this branch metric to the metric of the connecting survivor at the preceding time-unit. For each state, store the path with the largest metric (*survivor*), together with its metric.

Step 3: If $j < L+m$ go to step 2; otherwise stop.

If transmission is over the discrete memoryless channel, the final survivor has the largest metric, i.e. it is the *maximum likelihood path*.

Proof: See Lin & Costello [2], p. 318.

Note 3.2: If transmission is over the binary symmetric channel (2-input - 2-output DMC), then the branch metric is the Hamming distance $d(r_j, v_j)$ and the algorithm must find the path through the trellis with the smallest metric.

See Appendix 3.4 (p. 337), for two examples on Viterbi decoding.

3.5 SEQUENTIAL DECODING

For a brief description, see Appendix 3.5 (p. 339).

3.6 SYNDROME DECODING

The general outline of a syndrome decoder, for a systematic binary convolutional code is shown in Fig. 3.3.

Correction takes place in the right-hand side of the decoder by the mod-2 addition of the received sequence of message bits and the decoder's estimation of the corresponding channel error sequence.

The estimates of the error bits are obtained from the decision device (the 'brain' of the decoder) on information supplied by the syndrome register.

Syndrome generation for a systematic code requires a rep-

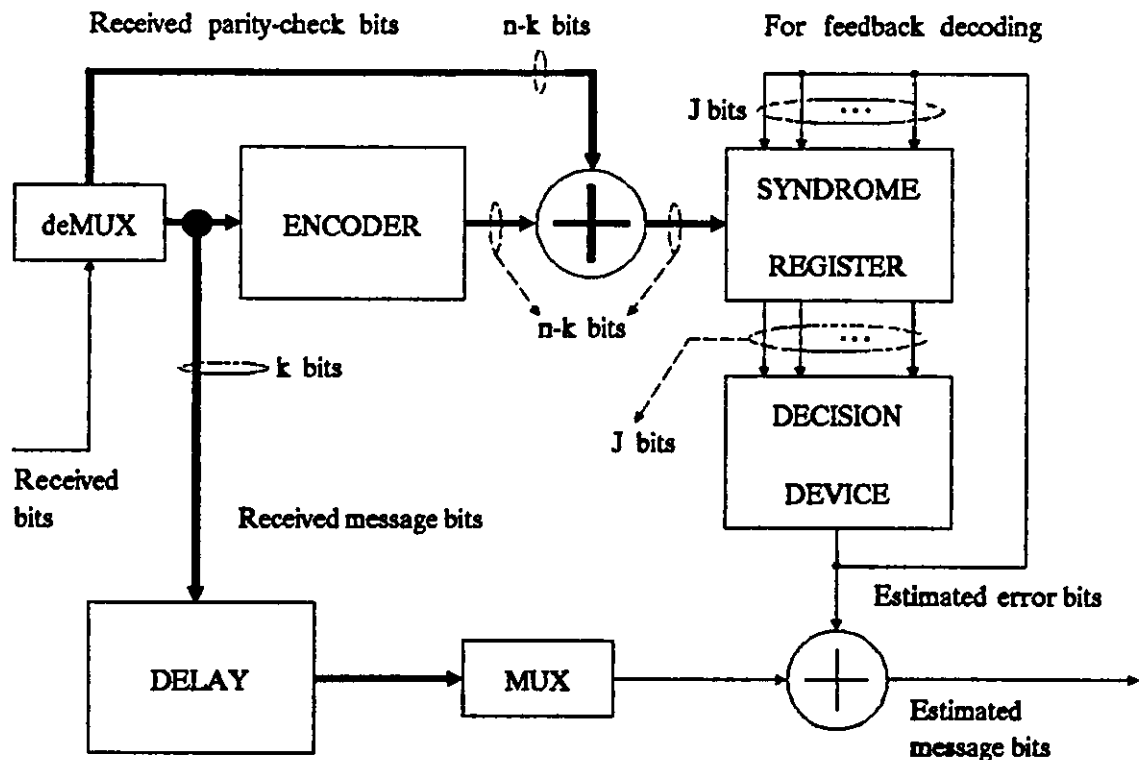


Figure 3.3: Block diagram of a syndrome decoder for binary systematic convolutional codes.

lica of the encoder and an X-OR gate. It has been shown in

Chapter 2 (see Theorem 2.14, p. 48) that the syndrome bits are linear combinations of the channel-error bits only (if channel noise is additive).

So far, the decoder is assumed to operate in the so-called *definite decoding (DD)* mode. It is possible though, to subtract every estimated error bit from the syndrome, via a feedback loop (see Fig. 3.3). In this mode, the so-called *feedback decoding (FD)* mode, the number of channel error bits that affect a syndrome bit is reduced.

The basic task of the decision device is:

Given the syndrome sequence s , find the channel error sequence e ; since (by Theorem 2.14) $s = f(e)$, the task of the decision device is specified to be:

Calculate $e = f^{-1}(s)$, where f^{-1} is such that the probability of a decoding error is minimized, under the given constraints of hardware complexity.

There are three basic syndrome decoding techniques for convolutional codes: 1. *Table look-up decoding* (see Appendix 3.6, p. 341), 2. *Error-trellis syndrome decoding* (see Chapter 4) & 3. *Threshold decoding* (see Chapter 5). They only differ in their respective decision devices.

3.7 CONCLUSIONS

The decoding techniques for convolutional codes were classified (see Fig. 3.1) and briefly described. They come under three main categories namely, Viterbi, sequential and syndrome decoding.

Syndrome decoding (Sec. 3.6), is distinguished in table look-up and threshold decoding, while the latter is further divided into majority and APP decoding. Because all these techniques are very simple to implement, they permit a cheap hardware realization and hence very high bit-rates. On the other hand, they are suboptimum because they make irreversible decisions for each block, based on one constraint-length of received digits.

Sequential decoding (Appendix 3.5) is a nearly optimum adaptive technique, which is fast when the channel is quiet but slow otherwise. Apart from its complexity, it suffers from randomly varying processing-time per block. On the other hand, it offers relatively high coding gains* (2-3 dB more than syndrome decoding), at bit rates of 1-20 Mbps (threshold decoders can offer their coding gain at higher bit rates, though - see Clark & Cain [13], pp. 342-4).

The *maximum likelihood decoding* technique, for convolutional codes, is the Viterbi algorithm (Sec. 3.4). This is a very clever method (widely used today), which returns high coding gains, at moderate to high bit rates (ibid.), depending on the channel and the modulation method employed. The algorithm is based on the encoder state-transition diagram.

A very important aspect of Viterbi decoding is the structure of the trellis (or the state-transition) diagram. This is so because the decoder has to store q^M paths, each L_k digit long, with L increasing by 1 with every new received block. Furthermore, at each block time-unit, and for each of the q^M paths, the decoder has to calculate q^k metrics, choose the best and store it. Hence, at each block-time-unit, $q^M q^k$ calculations have to be performed. Thus, the complexity of this technique increases exponentially with M & k (M is, usually, more important).

Section 3.2, contains a number of theorems on the complexity of the state-transition diagram (and hence on the complexity of trellis decoding). These results predict the structure of the state-transition diagram, given the code generator-polynomial matrix, $G(D)$ (assuming that a normal encoder is used - a reasonable assumption, since this is, usually, the minimal encoder). So, it is shown that the number of transitions into a state (or out of it) is q^{k-f} , while each transition has q^f labels. $f = f[G(D)]$ is defined in Theorem 3.1. Furthermore, it is shown that all transitions entering a state are caused by the same group of q^f input k -tuples. In Theorems 3.4 - 3.6, the self-loops are studied. A necessary & sufficient condition is developed, for the existence and generation of such a loop and it is shown that a

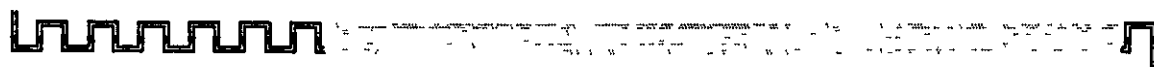
* See Section 1.4.

state-transition diagram has q^{k-f} such loops.

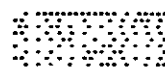
The originality of the work in that section lies in the generalization and systematization of some previously known observations about the state-transition diagram, as well as in some of the results themselves.

A final piece of original work is in Paragraph 3.2.3., where the (normal) encoder memory is divided into three sections, the one immediately affected by the current input-block (*FEG*), the one that loses its contents with each transition (*REG*) and the rest of the memory (*CEG*) which lies in between (see Fig. 3.2).

The work in Section 3.2 is also very useful to Chapter 4.



CHAPTER 4



Error-Trellis Syndrome Decoding

This chapter examines in detail the, so-called, constrained trellis, proposed by Reed & Truong [24] (their work was on error-trellis syndrome decoding). The material is divided into seven sections, of which the first and the third report on the background results, while the rest investigate the structure of the constrained trellis. An effort to relate a given circuit to the complexity of its constrained trellis has produced some original results and ideas. A *constrained trellis* is one in which the Hamming weight* of the current input block, together with the current state, is not allowed to exceed a certain threshold; this is, clearly, a 'generalized trellis' with a reduced number of states and transitions per state.

Section 4.1 aims to convince the reader about the existence of error-trellis syndrome decoding. Section 4.2, introduces the restriction which generates the constrained trellis and presents some fundamental relations about its complexity. Section 4.3, introduces the decoding algorithm. Section 4.4, develops the idea of the '*simplified*' trellis. The latter arises from the existence, in a constrained trellis, of states with only one path allowed in (or out of) them. In Section 4.5, a number of theorems are proved which relate the complexity of a constrained trellis with the associated circuit. Section 4.6, concentrates on the transitions of the simplified trellis and relates them to the associated circuit. Note that the simplified trellis contains transitions lasting for more than one time-unit. Finally,

* A binary code is assumed.

Section 4.7 discusses special cases.

4.1 ANALYSIS OF ERROR-TRELLIS SYNDROME DECODING

The first objective of this section is to obtain the general solution of the syndrome equation. This result will then be modified for the case of systematic codes. Equations for the best estimate of the source sequence will also be derived and finally the block diagram of an appropriate decoder will be proposed.

The work in this section is based on Reed & Truong [24], but because of its importance for the rest of this chapter (which is mainly original work), it is repeated here. The proofs, though, of the algebraic material are given in appendices.

4.1.1. General Solution of the Syndrome Equation

If the channel suffers from additive noise only, the task of the decoder is reduced to estimating the channel error-polynomial $E(D)$. This is so because $\tilde{V}(D) = R(D) - \tilde{E}(D)$ [see eqn (2.70), p. 47] and $\tilde{U}(D) = \tilde{V}(D)G'(D)$, where $G'(D)$ is the right-inverse of $G(D)$ (the code is assumed to be non-catastrophic - see Note A2.10.1, p. 319).

On the basis of the received polynomial $R(D)$, the decoder calculates the syndrome polynomial $S(D) = R(D)H^T(D)$ [see eqn (2.71)] and on the basis of $S(D)$ it provides its best estimate of $E(D)$. This latter calculation is based on the so-called syndrome eqn $S(D) = E(D)H^T(D)$ [see eqn (2.73)], which represents a linear system of $n-k$ equations in n unknowns [the polynomials $e^{(1)}(D), e^{(2)}(D), \dots, e^{(n)}(D)$]. It is obvious that this system of equations accepts many solutions, of which only one is valid [i.e. coincides with the channel-error polynomial $E(D)$]. The encoder has not enough information to make the correct choice each time, but it has a way to minimize its losses. This minimization procedure uses the general solution of the syndrome equation (the two of them constitute the 'heart' of this decoding technique).

Lemma 4.1: The polynomial-generator matrix of a non-catastrophic convolutional code can take the following form (usually called *Smith normal form*):

$$G(D) = A(D) \begin{bmatrix} I_k & 0 \end{bmatrix} B(D) \quad (4.1)$$

where 0 is the $k \times (n-k)$ zero matrix,
 $A(D)$ is a $k \times k$ nonsingular matrix,
 and $B(D)$ is an $n \times n$ nonsingular matrix.

Proof: See Appendix 4.1 (§ A4.1.1., p. 342). ■

Note 4.1: Let the following partition of $B(D)$ & $B^{-1}(D)$:

$$B(D) = \begin{bmatrix} X_1(D) \\ X_2(D) \end{bmatrix} \quad \& \quad B^{-1}(D) = \begin{bmatrix} Y_1(D) & Y_2(D) \end{bmatrix} \quad (4.2)$$

where $X_1(D)$, $X_2(D)$, $Y_1(D)$ & $Y_2(D)$ are $k \times n$, $(n-k) \times n$, $n \times k$ & $n \times (n-k)$ matrices, respectively. ■

The Smith normal form and the two partitions introduced above, are key steps towards the inversion of the syndrome equation. One important result based on them is the following theorem:

Theorem 4.1: Let $H(D)$ be the parity-check generator matrix of a non-catastrophic convolutional code. Then,

$$H(D) = Y_2^T(D) \quad (4.3)$$

where $Y_2(D)$ is defined by partition (4.2), above.

Proof: See Appendix 4.1 (§ A4.1.2., p. 342). ■

The above result is important in its own right, because it links $G(D)$ with $H(D)$; in this discussion it constitutes the basis for the following theorem, which is in fact the main result:

Theorem 4.2: For a non-catastrophic convolutional code, the general solution of the syndrome equation is

$$E(D) = T(D)X_1(D) + S(D)X_2(D) \quad (4.4)$$

where $T(D)$ is an $1 \times k$ matrix of polynomials which will be considered to be arbitrary and $X_1(D)$, $X_2(D)$ are defined by partitions (4.1) & (4.2).

Proof: See Appendix 4.1 (§ A4.1.3., p. 343). ■

Two more variations of the above result will be introduced. The first expresses $E(D)$ in terms of $G(D)$, $H(D)$ & $S(D)$, while the second avoids the use of $S(D)$.

Theorem 4.3: For a non-catastrophic convolutional code, the general solution of the syndrome equation can take the form

$$E(D) = Z(D)G(D) + S(D)H'^T(D) \quad (4.5)$$

where, $H'(D)$ is the right-inverse of $H(D)$ and $Z(D)$ is a $1 \times k$ matrix, which will be taken to be arbitrary.

Proof: See Appendix 4.1 (§ A4.1.4., p. 344). ■

In the paper by Reed & Truong [24], $X_2^T(D)$ is (mistakenly) referred to as the "left inverse of the parity-check matrix" [twice, in p. 78, between eqns (18) & (24)]. Since $H(D)$ is an $(n-k) \times n$ matrix of rank $n-k$, it can only have a right-inverse (see Theorem A2.2.11).

Lemma 4.2: For a non-catastrophic convolutional code, the general solution of the syndrome equation can take the form

$$E(D) = Z(D)G(D) + R(D)Q(D) \quad (4.6)$$

where, $Q(D) \triangleq Y_2(D)X_2(D)$, defined by partitions (4.1) & (4.2) and $Z(D)$ is a $1 \times k$ matrix (taken to be arbitrary).

Proof: See Appendix 4.1 (§ A4.1.5., p. 345). ■

4.1.2. The Case of Systematic Codes

Application of the above results, for the case of systematic codes is straightforward, but first it is necessary to elaborate on partition (4.2).

Theorem 4.4: For a systematic convolutional code, $B(D)$ & $B^{-1}(D)$ have the following general form

$$B(D) = \begin{bmatrix} A^{-1}(D) & A^{-1}(D)P(D) \\ C(D) & F^{-1}(D) + C(D)P(D) \end{bmatrix} \quad (4.7a)$$

$$B^{-1}(D) = \begin{bmatrix} A(D) + P(D)F(D)C(D)A(D) & -P(D)F(D) \\ -F(D)C(D)A(D) & F(D) \end{bmatrix} \quad (4.7b)$$

where $A(D)$ & $B(D)$ are part of the Smith normal form of the polynomial-generator matrix $G(D) = [I_k, P(D)]$, defined by eqn (4.1), $F(D)$ is an $(n-k) \times (n-k)$ nonsingular matrix and $C(D)$ is an $(n-k) \times k$ matrix.

Proof: See Appendix 4.1 (§ A4.1.6., p. 345). ■

Note that (4.7) is the most general form of $B(D)$ & $B^{-1}(D)$, more general than those used by Reed & Truong [24]. In these expressions, $A(D)$ & $F(D)$ may be any nonsingular $k \times k$ & $(n-k) \times (n-k)$ matrices, respectively and $C(D)$ can be any $(n-k) \times k$ matrix. If $A(D) = I_k$, $F(D) = I_{n-k}$ and $C(D) = 0$, the following lemma is proved:

Lemma 4.3: For a systematic convolutional code, $B(D)$ & $B^{-1}(D)$ may take the following form:

$$B(D) = \begin{bmatrix} I_k & P(D) \\ 0 & I_{n-k} \end{bmatrix} \quad \text{and} \quad B^{-1}(D) = \begin{bmatrix} I_k & -P(D) \\ 0 & I_{n-k} \end{bmatrix} \quad (4.8)$$

where $[I_k, P(D)] = G(D)$ is the polynomial-generator matrix of the code. ■

Theorem 4.5: For a systematic convolutional code the general solution of the syndrome equation is

$$E(D) = [Z(D), Z(D)P(D) + S(D)F^{-1}(D)] \quad (4.9a)$$

where $[I_k, P(D)] = G(D)$, $F(D)$ is an $(n-k) \times (n-k)$ nonsingular (arbitrary) matrix and $Z(D)$ is a $1 \times k$ (arbitrary) matrix. Since, usually, $F(D) = I_{n-k}$:

$$E(D) = [Z(D), Z(D)P(D) + S(D)] \quad (4.9b)$$

Proof: See Appendix 4.1 (§ A4.1.7., p. 347). ■

4.1.3. Estimation of the Channel Error Digits

So far, the main results concern the inversion of the syndrome eqn. In the solutions obtained, an arbitrary quantity is present; this quantity is chosen by the decoder so that the probability of a decoding error is minimized. For a binary code, this is equivalent to minimizing the Hamming weight of the channel-error sequence e , or the same, to minimizing the Hamming weight of the coefficients of $E(D)$. This is the application of maximum likelihood decoding for a BSC (see Theorem 1.2, p. 12).

Note 4.2: For a binary code, the arbitrary quantity in the general solution of the syndrome equation is chosen so that the Hamming weight of the channel-error sequence is minimized. The *Hamming weight* of, say, $X(D)$ is denoted by $w[X(D)]$ and is equal to the number of non-zero terms of the polynomial. The best estimate of, say, $X(D)$ is denoted by $\tilde{X}(D)$. It is understood that $X(D)$ denotes the objective value of whatever $X(D)$ represents, while $\tilde{X}(D)$ represents a 'subjective' evaluation of $X(D)$. ■

From Note 4.2 and the solution of the syndrome equations, the lemmas below follow easily:

Lemma 4.4: For a non-catastrophic convolutional code,

$$\tilde{E}(D) = \tilde{Z}(D)G(D) + S(D)H'^T(D) \quad (4.10)$$

where, for a binary code, $\tilde{Z}(D)$ is chosen so that $w[\tilde{Z}(D)G(D) + S(D)H'^T(D)]$ is minimized. ■

Lemma 4.5: For a systematic convolutional code,

$$\tilde{E}(D) = [\tilde{Z}(D), \tilde{Z}(D)P(D) + S(D)] \quad (4.11)$$

where, for a binary code, $\tilde{Z}(D)$ is chosen so that $w[\tilde{Z}(D), \tilde{Z}(D)P(D) + S(D)]$ is minimized. ■

4.1.4. Estimation of the Source Digits

The analysis of the examined decoding technique will be finalized with the derivation of an expression for $\tilde{U}(D)$, the decoder's best estimate of the source message $U(D)$.

Theorem 4.6: For a non-catastrophic convolutional code,

$$\tilde{U}(D) = R(D)G'(D) - \tilde{Z}(D) \quad (4.12)$$

where $G'(D)$ is the right-inverse of $G(D)$ and $Z(D)$ is an arbitrary $1 \times k$ matrix of polynomials.

Proof: See Appendix 4.1 (§ A4.1.8., p. 348). ■

Theorem 4.7: For a systematic convolutional code,

$$\tilde{U}(D) = R^{(m)}(D) - \tilde{Z}(D) \quad (4.13)$$

where $Z(D)$ is an arbitrary $1 \times k$ matrix.

Proof: See Appendix 4.1 (§ A4.1.9., p. 349). ■

Note that eqn (4.13) was derived using the general form of $B(D)$ & $B^{-1}(D)$ (see Theorem 4.4), instead of the simplified form used by Reed & Truong [24] (see Lemma 4.3). Nevertheless, the two results are identical.

It would be useful to mention two results obtained in Appendix 4.1, during the proof of the last two theorems.

Since $G(D)$ is a $k \times n$ matrix of rank k (by Lemma A2.10.1,

p. 319), it has a right-inverse (by Theorem A2.2.11, p. 303), denoted by $G'(D)$. From eqns (A4.1.8) & (A4.1.9):

For a general non-catastrophic convolutional code:

$$G'(D) = B^{-1}(D) \begin{bmatrix} I_k \\ 0 \end{bmatrix} A^{-1}(D) \quad (4.14)$$

For a systematic convolutional code:

$$G'(D) = \begin{bmatrix} I_k + P(D)F(D)C(D) \\ -F(D)C(D) \end{bmatrix} \quad (4.15)$$

where $A(D)$ & $B(D)$ are defined by Lemma 4.1, and $C(D)$ & $F(D)$ have been introduced in Theorem 4.4.

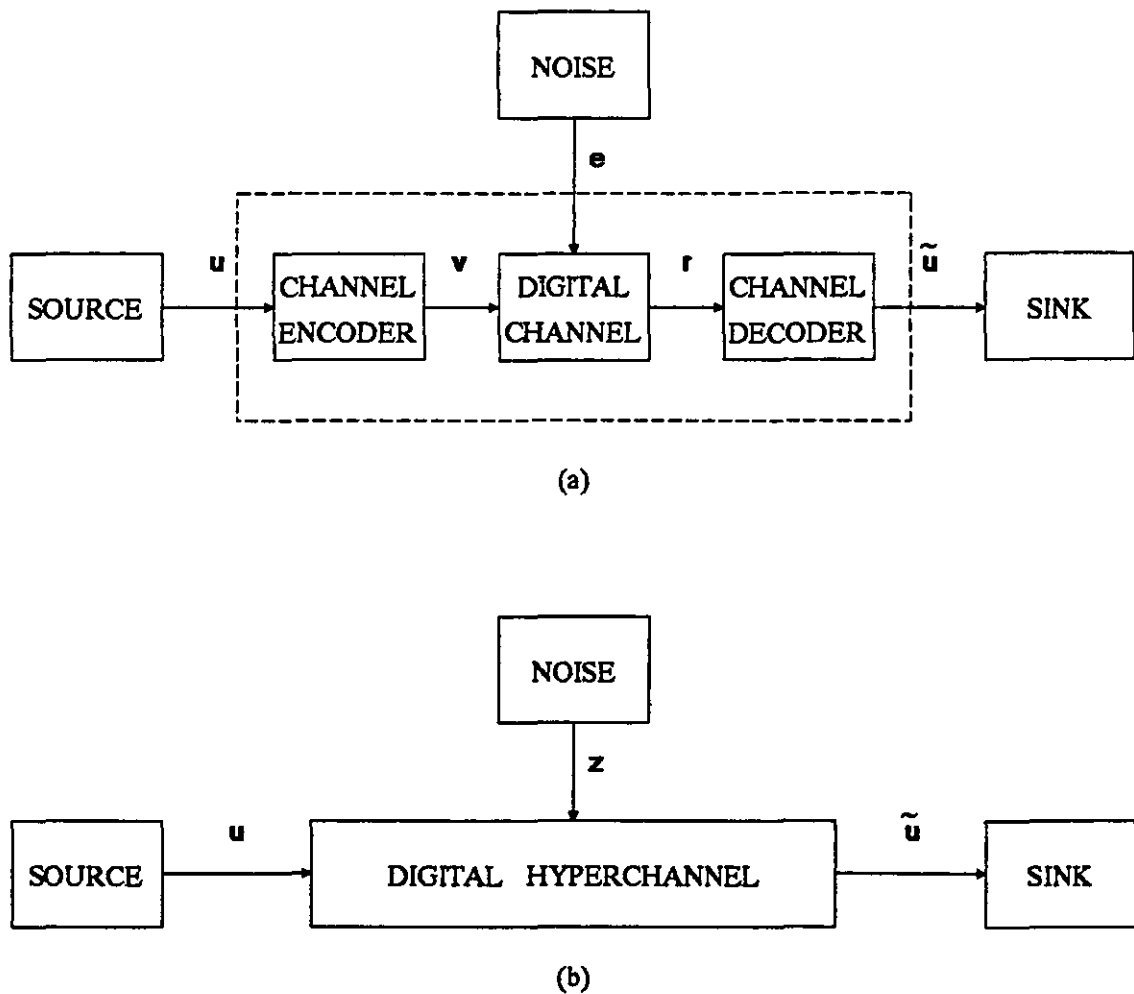


Figure 4.1: The communications system as seen by the channel codec (a) and by the user (b).

4.1.5. The Hyperchannel Error Polynomial

Note that for a noiseless channel, $\tilde{U}(D) = R(D)G'(D) = U(D)$, for general non-catastrophic codes, while for systematic codes, $\tilde{U}(D) = R^{(m)}(D) = U(D)$.

Comparing with the results of Theorems 4.6 & 4.7 above, one readily concludes that the arbitrary quantity $Z(D)$ plays the role of a 'correcting factor' (as Reed & Truong have put it). In effect, this arbitrary quantity represents noise referred to the information source or, to put it otherwise, it represents the noise digits of the (hyper)channel between the source O/P and the sink I/P (see Fig. 4.1). For this reason, the term *hyperchannel error polynomial* is proposed for $Z(D)$.

4.1.6. Error-Trellis Syndrome Decoder

From the analysis, so far, one concludes that for a general non-catastrophic convolutional code,

$$\begin{aligned}
 \tilde{U}(D) &= R(D)G'(D) - \tilde{Z}(D) \\
 \tilde{E}(D) &= \tilde{Z}(D)G(D) + S(D)H'^T(D)
 \end{aligned}
 \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \rightarrow (4.16)$$

where, for a binary code,

$$w[\tilde{E}(D)] = \underset{Z(D)}{\text{MIN}} \{ w[Z(D)G(D) + S(D)H'^T(D)] \}$$

The set of relations (4.16) can be translated into the block diagram of Fig. 4.2.

The analysis for a systematic convolutional code gave the following results:

$$\begin{aligned}
 \tilde{U}(D) &= R^{(m)}(D) - \tilde{Z}(D) \\
 \tilde{E}(D) &= [\tilde{Z}(D), \tilde{Z}(D)P(D) + S(D)]
 \end{aligned}
 \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \rightarrow (4.17)$$

where, for a binary code,

$$w[\tilde{E}(D)] = \underset{Z(D)}{\text{MIN}} \{ w[Z(D), Z(D)P(D) + S(D)] \}$$

The set of relns (4.17) suggest the decoder of Fig. 4.3.

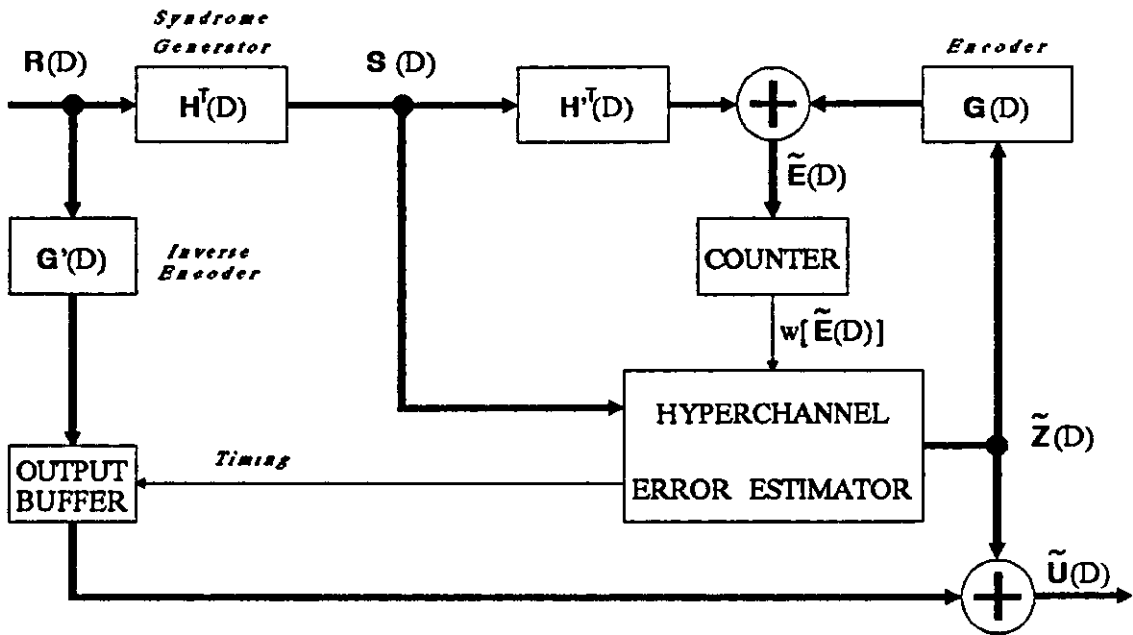


Figure 4.2: Block diagram of the error-trellis syndrome decoder for a binary non-catastrophic convolutional code.

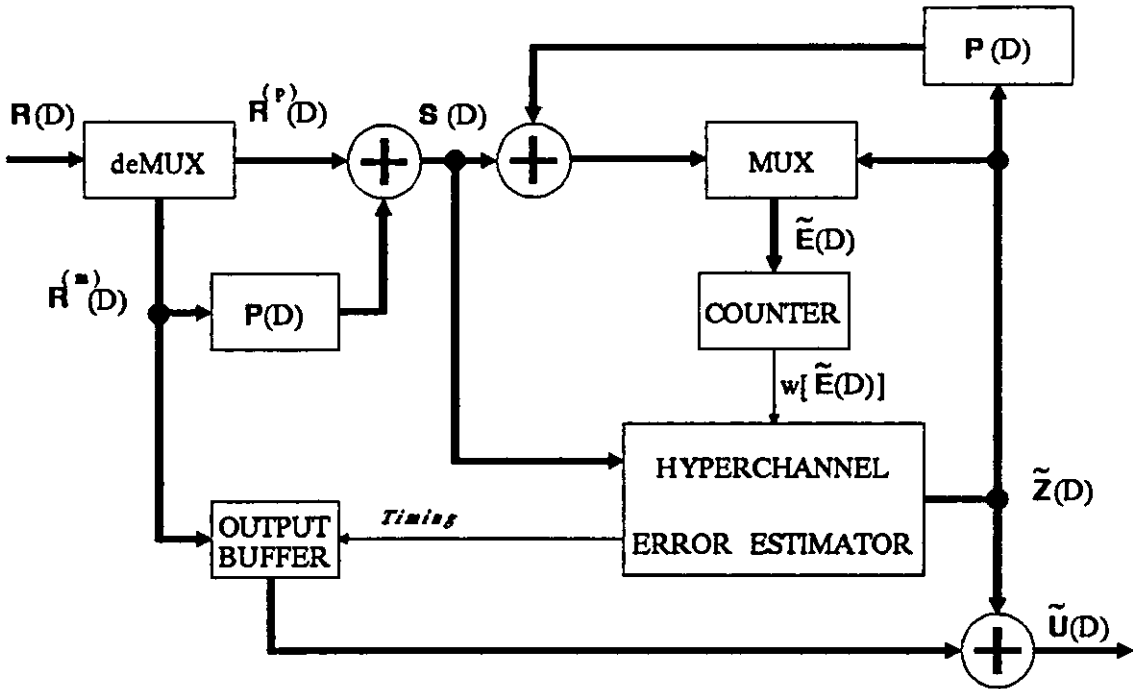


Figure 4.3: Block diagram of the error-trellis syndrome decoder for a binary systematic convolutional code.

The solid line in these block diagrams represents multi-bit buses. The counter counts the number of ones (i.e. it

evaluates the Hamming weight). The decoder's 'brain' is the *hyperchannel error estimator* (HEE); the name follows after eqns (4.12) & (4.13). The task of the HEE is best described in Section 4.3. The output buffer is used to delay the current input block until the HEE is satisfied that the coefficients of the hyperchannel error-polynomial are such that the estimate of the channel error-sequence has the minimum possible Hamming weight.

4.2 THE CONSTRAINED REGULATOR TRELLIS OF A SYSTEMATIC CODE

It has been shown that for a binary* systematic convolutional code, $\tilde{E}(D) = [\tilde{Z}(D), \tilde{Z}(D)P(D)] + [0, S(D)]$. The decoding rule is to choose $[\tilde{Z}(D), \tilde{Z}(D)P(D)]$, so that the Hamming weight of $\tilde{E}(D)$ is minimized. This rule can be stated otherwise: *Choose $\tilde{Z}(D)$, so that the Hamming distance between $[\tilde{Z}(D), \tilde{Z}(D)P(D)]$ & $[0, S(D)]$ is minimized.* The reader should recall that in Viterbi decoding, the rule is: *Choose $\tilde{U}(D)$ so that the Hamming distance between $[\tilde{U}(D), \tilde{U}(D)P(D)]$ & $[R^{(m)}(D), R^{(p)}(D)]$ is minimum.* It is obvious then that the technique under discussion leads naturally to some sort of trellis decoding.

Indeed, the part of the decoder that is a replica of the encoder and takes as input the arbitrary quantity $\tilde{Z}(D)$, to produce $[\tilde{Z}(D), \tilde{Q}(D)]$, where,

$$Q(D) \triangleq Z(D)P(D) \quad (4.18)$$

has been named the *regulator circuit*, by Reed & Truong [24] and has a trellis diagram. The advantage of this technique, over Viterbi decoding, is that the trellis diagram, here, is considerably simpler. In particular, the number of states is reduced and also the number of branches that leave a particular state is, in general, reduced. This is achieved by exploiting the existence of the *bounded error-correcting capability* of the code.

4.2.1. Complexity of the Constrained Regulator Trellis

In this paragraph, the constrained trellis of the regula-

* Unless otherwise stated, only binary codes will be considered.

tor circuit will be introduced. The analysis that follows is based on the restriction introduced by Note 4.3 (below).

Definition 4.1: The *error-correcting capability* of a convolutional code is denoted by t and is defined to be the maximum number of errors the code guarantees to correct, in any actual constraint-length, $n_A \hat{=} (m+1)n$.

If the need to relax the maximum likelihood search is accepted, then the reduced range of search should include (for best performance) the most critical elements. Since the code guarantees to correct t or less errors in $m+1$ blocks, the following restriction may be introduced.

Note 4.3: In error-trellis syndrome decoding, the error sequences examined must have a weight of t or less, over a length of $m+1$ blocks:

$$\sum_{i=0}^m w[e_{h+i}] \leq t \quad / \text{ for all } h \geq 0 \quad (4.19)$$

where e_h is the h th channel-error block [defined by eqn (2.62b)].

In Lemma 4.5, the general solution of the syndrome equation was given in polynomial form. Using eqn (4.18),

$$\tilde{E}(D) = [\tilde{Z}(D), \tilde{Q}(D)] + [0, S(D)] \quad (4.20)$$

Definition 4.2: Given the generator-polynomial matrix $G(D) = [I_k, P(D)]$ of an (n, k, m) systematic convolutional code, the *regulator circuit* of the error-trellis syndrome decoder is understood to be the k -input, $(n-k)$ -output, linear sequential circuit (LSC) with transfer-function matrix $P(D)$. Furthermore, any LSC, structured like the 'normal encoder' (see Definition 2.9, p. 34), will be called a *normal LSC*. The regulator circuit is then a *normal LSC*.

It is obvious from the definition above, that the regula-

tor circuit is a replica of the encoder; hence they have the same total memory M [see eqn (3.1)] and the same transition diagram if no constraints are imposed [see eqn (3.2)].

It is easy to show that the matrix version of (4.20) is,

$$\tilde{e}_h = [\tilde{z}_h, \tilde{q}_h] + [0, s_h] \quad (4.21)$$

where s_h is the current syndrome block,

z_h is the current O/P block of the HEE and

q_h is the current O/P block of the regulator circuit.

Note that z_h is the h th hyperchannel error block, i.e. it represents the block of k channel error bits that have corrupted the corresponding block of k message bits u_h .

From Note 4.3 and eqn (4.21), one concludes that the weight of \tilde{z} , over $m+1$ blocks, should not exceed t .

Theorem 4.8: In error-trellis syndrome decoding, the Hamming weight of $m+1$ consecutive hyperchannel error blocks, z_h , must not exceed the error-correcting capability, t , of the code:

$$\sum_{i=0}^m w[z_{h+i}] \leq t \quad \text{/for all } h \geq 0 \quad (4.22)$$

Proof: From eqn (4.21):

$$w[e_h] = w[z_h, q_h + s_h] = w[z_h] + w[q_h + s_h]$$

Substituting the above equation in (4.19):

$$\sum_{i=0}^m w[z_{h+i}] \leq t - \sum_{i=0}^m w[q_{h+i} + s_{h+i}] \leq t$$

QED

The following is based on Definition 3.1 which deals with various parts of the (logical or physical) memory of an LSC. Since the next state contains part of the current input, it is necessary to introduce terms that describe various parts of the I/P block; furthermore, these terms should be compatible with those introduced by eqns (3.5) (see p. 58).

Definition 4.3: Consider the I/P block u_h and assume that prior to its clocking in the memory it resides in a separate memory which will be called the *input group (ING)*; logically, ING should be visualized as a set of k shift registers (SRs) of length one, each. The following two subsets of ING are introduced: The *memory input group (MIG)* contains all the (logical or physical) SR-stages of ING that are connected with SRs of length one or more. The *discarded input group (DIG)* contains all the SR-stages of the ING that are not included in the MIG. Note that the ING contains k elements, the MIG contains $k-f$ elements, while the DIG contains f elements:

$$ING \triangleq \{u_h^{(i)} / i=1,2,\dots,k\} \quad (4.23a)$$

$$MIG \triangleq \{u_h^{(i)} / i=1,2,\dots,k: M_i \geq 1\} \quad (4.23b)$$

$$DIG \triangleq \{u_h^{(i)} / i=1,2,\dots,k: M_i = 0\} \quad (4.23c)$$

The various groups defined above are sets of logical or physical SR-stages. Notation $ING(h)$, $MIG(h)$ & $DIG(h)$ will be used to denote the set of bits that reside in ING, MIG & DIG, respectively, at time-unit h .

To complete the picture, the ordered sequences of the contents of the above sets of SR-stages will also be introduced. Clearly, the state of the ING, at time-unit h , is simply u_h . In a fashion similar to the one used by eqns (3.6), the states of the MIG and the DIG, at time-unit h , are denoted by $M(h)$ and $D(h)$, respectively.

The results below will help develop some very useful results. They can be considered as an interface between the algebra of the various groups (FEG,MEG,ING, etc) and the algebra of the weight of their state, at some time-unit.

Theorem 4.9: Let the partition $B = \langle A_1, A_2, \dots, A_a \rangle$, where $B \subseteq MEG^*$ and $A_i / i=1,2,\dots,a$ is a set of logical or physical SR-stages from the MEG; let also $B(h)$ & $A_i(h)$ denote the state of B & A_i , respectively, at time-unit h . Then:

$$w[B(h)] = \sum_{i=1}^a w[A_i(h)] \quad \text{/for all } h \geq 0 \quad (4.24)$$

* ACB denotes " A is a subset of B ".

Proof: See Appendix 4.3 (§ A4.3.1., p. 353).

Lemma 4.6: Let A & B be any two subsets of the MEG.
Then:

$$\text{If } A \subset B \implies w[A(h)] \leq w[B(h)] \quad (4.25)$$

Proof: Let A, B be any two subsets of the MEG. If $A \subset B$, i.e. A is a proper subset of B, then from Theorem A4.2.1, $B = \langle A, B-A \rangle$ and from Theorem 4.9, for all time-units h:

$$w[B(h)] = w[A(h)] + w[(B-A)(h)] \leq w[A(h)]$$

since the Hamming weight is a non-negative integer.

QED

Theorem 4.10: If the Hamming weight of the current state $S(h)$ of the regulator circuit is w ($0 \leq w \leq t$), the weight of the current input z_h must be $t-w$, or less:

$$w[z_h] + w[S(h)] \leq t \quad \text{for all } h \geq 0 \quad (4.26)$$

Proof: See Appendix 4.3 (§ A4.3.2., p. 354).

Lemma 4.7: The Hamming weight of any state of the regulator circuit does not exceed t :

$$w[S(h)] \leq t \quad \text{for all } h \geq 0 \quad (4.27)$$

Proof: From Theorem 4.10: $w[S(h)] \leq t - w[z_h] \leq t \quad / h \geq 0$.

QED

Lemma 4.8: The Hamming weight of any I/P block to the regulator circuit must not exceed t :

$$w[z_h] \leq t \quad \text{for all } h \geq 0 \quad (4.28)$$

Proof: From Theorem 4.10: $w[z_h] \leq t - w[S(h)] \leq t \quad / h \geq 0$.

QED

The results that follow will link the complexity of the constrained trellis to the associated circuit parameters. To facilitate discussion, suitable notation will be introduced.

Note 4.4: Consider the following notation with respect to the central portion of the constrained regulator trellis:

$E(i) \hat{=}$ Number of different allowable states of weight i .

$\varrho(i,w) \hat{=}$ Number of different allowable input blocks of weight i , when current state has weight w .

$\forall(i,w) \hat{=}$ Number of different allowable states that can be reached within one time-unit, from a given state of weight w , via an I/P block of weight i .

$\Sigma E \hat{=}$ Total number of different allowable states.

$\Sigma \varrho(w) \hat{=}$ Total number of different allowable input blocks, when current state has weight w .

$\Sigma \forall(w) \hat{=}$ Total number of different allowable states that can be reached within one time-unit, from a given state of weight w .

Expressions for E , ϱ & \forall can be easily developed:

Theorem 4.11: Consider an (n,k,m) regulator circuit with transfer-function matrix $P(D)$. Let t be the error-correcting capability of the associated code and M the total circuit memory. Then, with respect to the central portion of its constrained trellis:

$$E(i) = \binom{M}{i} \quad / 0 \leq i \leq t \quad (4.29a)$$

$$\varrho(i,w) = \binom{k}{i} \quad / 0 \leq i \leq t-w \quad \& \quad 0 \leq w \leq t \quad (4.29b)$$

$$\forall(i,w) = \binom{k-f}{i} \quad / 0 \leq i \leq t-w \quad \& \quad 0 \leq w \leq t \quad (4.29c)$$

where $f = f[P(D)]$ (see Theorem 3.1, p. 57) and $\binom{n}{k} \hat{=}$ $n!/[k!(n-k)!]$ = the binomial coefficient.

Proof: See Appendix 4.4 (p. 355).

Lemma 4.9: Consider an (n,k,m) regulator circuit with transfer-function matrix $P(D)$. Let t be the error-correcting capability of the associated code and M the total circuit memory. Then, with respect to the central portion of its constrained trellis:

$$\Sigma E = \sum_{i=0}^t \binom{M}{i} \quad (4.30a)$$

$$\Sigma \Omega(w) = \sum_{i=0}^{t-w} \binom{k}{i} \quad / 0 \leq w \leq t \quad (4.30b)$$

$$\Sigma \Psi(w) = \sum_{i=0}^{t-w} \binom{k-f}{i} \quad / 0 \leq w \leq t \quad (4.30c)$$

where $f = f[P(D)]$ (see Theorem 3.1, p. 57) and $\binom{n}{k} \triangleq n!/k!/(n-k)! \triangleq C(n,k)$ = the binomial coefficient. Note that $C(n,k) = 0$, for $n < k$.

Proof: The results are the cumulative quantities of Theorem 4.11. ■

According to the discussion above, the state-transition diagram of the regulator circuit is made of a total of ΣE states of weights ranging from 0 to t [there are exactly $E(i)$ states of weight i]. From each state of weight w originate $\Omega(i,w)$ transitions of weight i , where i is restricted between 0 and $t-w$. Some transitions may be multiple-edge ones (see Note A3.1.1, p. 330).

Each transition needs to be marked by the input block (z_h) that caused that transition. To facilitate decoding, each transition is marked by z_h/q_h , i.e. it includes the corresponding O/P block q_h . Note that, in reality, the circuit O/P is $[z_h, q_h]$, since the circuit's transfer-function, $P(D)$, is part of $G(D) = [I_k, P(D)]$; but since the regulator circuit for systematic-code cases is only examined, q_h is taken to be its O/P (see, also, Definition 4.2). The discussion above leads easily to the following note.

4.2.2. Construction of the Trellis

Note 4.5: The constrained regulator trellis is constructed exactly like the (unconstrained) encoder trellis (see Note 3.1, p. 65), except that only states of weight t or less are considered. The states are arranged in groups according to weight, with the all-zero state on top, below the weight-one group, etc. From a given state of weight w ,

only transitions that are caused by an I/P block of weight $t-w$, or less, are considered.

For an example see Appendix 4.5 (p. 356).

At this stage, one is able to assess the importance of the proposed decoding technique. Although it introduces some degradation, it compensates by allowing the use of longer, and hence more powerful, codes.

For example, according to Lin & Costello ([2], p. 337), the practical limit for the Viterbi algorithm is codes with a total encoder memory of $M=8$ (1983), i.e. with a total of $2^8 = 256$ trellis states. For error-trellis syndrome decoding, the number of states considered is $1 + M + \dots + C(M,t)$ (see Lemma 4.9). For a $t=2$ code, the constrained trellis has $1+M(M+1)/2$ states, while the unconstrained one has 2^M ; a Viterbi decoder for an $M=8$ code needs to consider a 256-state trellis, while an error-trellis decoder only about $(1/6)$ th of that (37-state trellis). From another point of view, if the maximum number of states is not to exceed 256, a $t=2$ code with an encoder of total memory $M=22$ can be used in combination with error-trellis syndrome decoding.

4.3 DECODING ALGORITHM FOR SYSTEMATIC BINARY CODES

Given the trellis diagram of the regulator circuit (defined above), the Viterbi algorithm (introduced in Section 3.4), the general solution of the syndrome equation (obtained in Section 4.1) and the decoding equation (Theorem 4.7) the procedure for error-trellis syndrome decoding follows easily:

Note 4.6: To decode:

Step 1: Let time-unit be $h=0$.

Step 2: Calculate the current syndrome block s_h .

Step 3: For each branch, with label z_h/q_h , form the sum $q_h + s_h$ and append the label with the number $W = w[z_h] + w[q_h + s_h]$.

Step 4: For each state, at time-unit $h+1$: From all paths entering that state, keep the one with the smallest metric M_{h+1} and store the path & the metric ($M_{h+1} = M_h + W$).

Step 5: If there are more blocks to be decoded, increase h by 1 and go to step 3; otherwise proceed.

Step 6: From the survivor path, deduce the hyperchannel error sequence $z = [z_0, z_1, z_2, \dots]$ and subtract it from $r^{(m)}$, to obtain \tilde{u} .

For an example, see Appendix 4.6 (p. 361).

4.4 SIMPLIFICATION OF THE CONSTRAINED TRELLIS

The purpose of this section is to investigate the gains obtained from a simplified trellis. Savings will be possible if one can exploit the redundancy of the diagram. In particular, the reader may have already noticed that some states have only one I/P (and one O/P) transition. The lack of choice, when the circuit is at such a state, suggests their elimination. If such a simplification is not accompanied by an increase in complexity (which may be a by-product of this modification), then net gains will have been obtained. These gains mean that for a given code, the memory requirement is reduced, or that for a given memory-constraint more powerful codes may be implemented.

The proposed simplifications may be very useful, because a reduction in the decoder memory-requirement makes the use of larger-distance codes possible. In order to assess the memory savings, quantitative results should be produced.

The idea of the constrained state-transition diagram belongs, of course, to Reed & Truong [24]. In the remainder of this chapter, the structure of the constrained trellis, and of its simplified version, will be studied. The direct application of such an effort is on error-trellis syndrome decoding, but the results that follow are more general. They apply to any normal LSC (see Definition 4.2), while the constraint on the Hamming weight of the error sequences examined, need not be the associated-code error-correcting capa-

bility, t . For this reason the terms, *constrained trellis* (instead of *constrained regulator trellis*), *normal LSC* (instead of *regulator circuit*) and *weight-constraint t* (instead of *error-correcting capability t*), will be used.

In fact, it will be interesting to attempt error-trellis syndrome decoding with a weight-constraint different than the error-correcting capability of the code.

4.4.1. Partition of the Circuit Memory

Theorem 4.12: Consider a normal LSC and let MEG represent the set of the contents of its memory.* Consider also the parts of the memory REG, CEG & FEG.* The following relations hold true:

$$\text{FEG} \cup \text{CEG} \cup \text{REG} = \text{MEG} \quad (4.31a)$$

$$\text{FEG} \cap \text{CEG} = \text{CEG} \cap \text{REG} = \emptyset \quad (4.31b)$$

$$\text{FEG} \cap \text{REG} = \left\{ u_{h-1}^{(i)} \quad / i=1,2,\dots,k: M_i = 1 \right\} \quad (4.31c)$$

Proof: See Appendix 4.7 (§ A4.7.1., p. 363).

Theorem 4.13: Consider a normal LSC and let MEG represent the set of the contents of its memory.* Consider also the parts of the memory REG, CEG & FEG.* MEG can be partitioned in many ways, including the following:

$$\text{Partition I:} \quad \text{MEG} = \langle \text{FEG}, \text{CEG}, \text{REG}' \rangle \quad (4.32)$$

where $\text{REG}' \triangleq \text{REG} - \text{FEG}$

$$\text{Partition II:} \quad \text{MEG} = \langle \text{FEG}', \text{CEG}, \text{REG} \rangle \quad (4.33)$$

where $\text{FEG}' \triangleq \text{FEG} - \text{REG}$

Proof: See Appendix 4.7 (§ A4.7.2., p. 364).

The following very useful results, partition the contents of the memory of a normal LSC, in terms of its contents one time-unit before. But first an example to illustrate the concepts introduced, so far.

* See Definition 3.1 (p. 58).

Example 4.1: Consider a normal LSC with SR_s of various lengths, say 0,1,2,3,4 & 5 (see Fig. 4.4). The various sets, defined above, will be as following:

TABLE 4.1

At time-unit h

$$FEG(h) = \{A, B, D, G, K\}$$

$$FEG'(h) = \{B, D, G, K\}$$

$$CEG(h) = \{E, H, I, L, M, N\}$$

$$REG(h) = \{A, C, F, J, O\}$$

$$REG'(h) = \{C, F, J, O\}$$

$$ING(h) = \{x_1, x_2, x_3, x_4, x_5, x_6\}$$

$$MIG(h) = \{x_2, x_3, x_4, x_5, x_6\}$$

$$DIG(h) = \{x_1\}$$

At time-unit h+1

$$FEG(h+1) = \{x_2, x_3, x_4, x_5, x_6\}$$

$$FEG'(h+1) = \{x_3, x_4, x_5, x_6\}$$

$$CEG(h+1) = \{D, G, H, K, L, M\}$$

$$REG(h+1) = \{x_2, B, E, I, N\}$$

$$REG'(h+1) = \{B, E, I, N\}$$

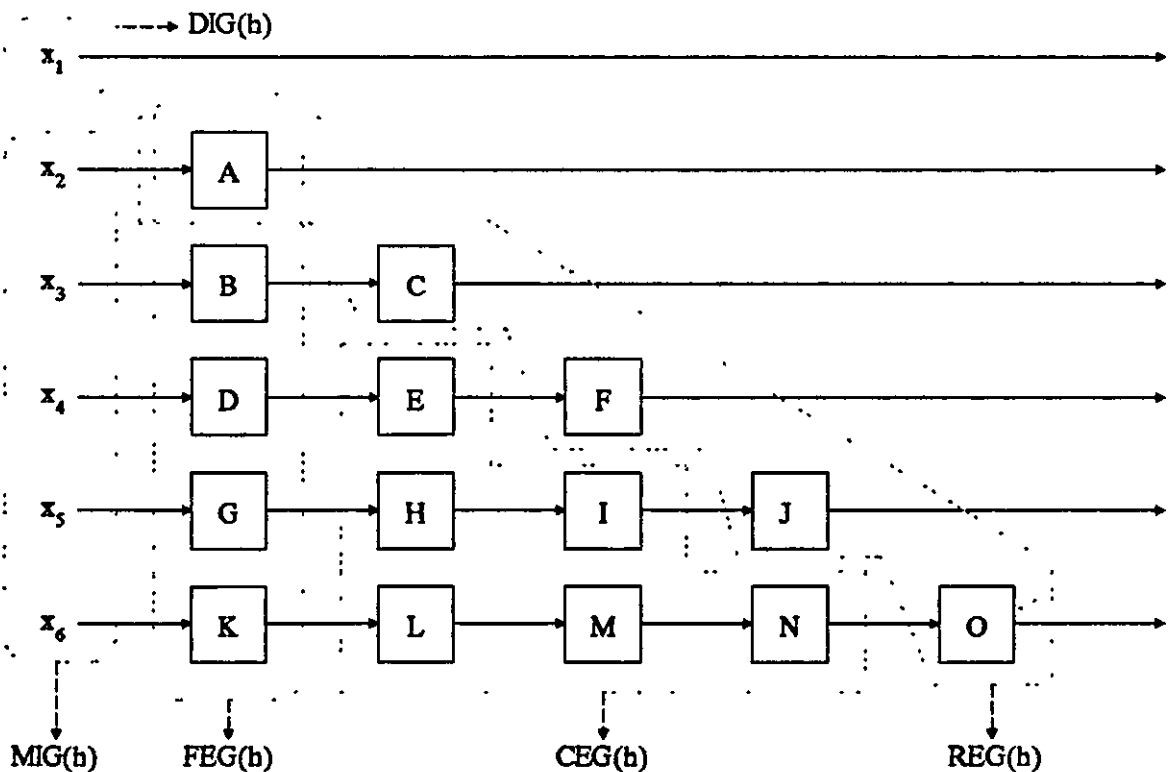


Figure 4.4: The memory partitions of an arbitrary normal LSC.

From the point of view of this discussion, this is the most complete circuit because all limit situations are considered (all SRs of length 5 or more behave alike).

Theorem 4.14: Consider a normal LSC and let $MEG(h+1)$ represent the set of the contents of its memory at time-unit $h+1$. Then, the following hold true:

$$MEG(h+1) = \langle MIG(h), FEG'(h), CEG(h) \rangle \quad (4.34)$$

$$FEG' = \{u_{h-1}^{(i)} \quad / i=1,2,\dots,k: M_i \geq 2\} \quad (4.35)$$

$$MIG(h) \cup FEG'(h) \cup CEG(h) \supseteq \{u_{h-j}^{(i)} \quad / i \in [1,k], j \in [0, M_i) \text{ \& } M_i \geq 1\} \quad (4.36)$$

where MIG is defined by eqn (4.23b), CEG by eqn (3.5d) and FEG' by eqn (4.33).

Proof: See Appendix 4.7 (§ A4.7.3., p. 365).

Lemma 4.10: Consider a normal LSC and let $MEG(h+1)$ represent the set of the contents of its memory at time-unit $h+1$. Then, the following partition holds true:

$$MEG(h+1) \cup DIG(h) = \langle ING(h), FEG'(h), CEG(h) \rangle \quad (4.37)$$

where DIG & ING are defined by eqns (4.23), CEG by eqn (3.5d) and FEG' by eqn (4.33).

Proof: See Appendix 4.7 (§ A4.7.4., p. 366).

4.4.2. Structure of the Constrained Trellis

It is evident from the above that during a transition, the I/P block feeds the FEG, the FEG feeds the CEG (and the REG in SRs of length 2), the CEG retains most of its elements with the rest feeding the REG, while all the elements of the REG are discarded. So, the only digits of $S(h)$ that do not participate in the formation of $S(h+1)$ are those of the $REG(h)$. To arrive at a specific state $S(h+1) = S_y$ one has to start from a state $S(h) = S_x^*$ with a specific and

* Assuming that transition $S_x \longrightarrow S_y$ is possible.

unique FEG'(h) & CEG(h) and to use a specific I/P block ING(h). Nevertheless, one is at liberty with respect to the elements of the REG, hence the various states from which S(h+1) can be reached are generated by letting REG(h) go through all permissible combinations of bits. The theorem that follows makes use of the discussion, above.

Theorem 4.15: Within the central portion of the constrained trellis, the number of transitions entering a state equals the number of transitions leaving that state.

Proof: Although this result is original, it is rather 'expected', since it holds true for the special case of the unconstrained trellis (see Lemma 3.1, p. 66). For this reason, and because its proof is a 'lengthy' one, it is given in Appendix 4.8 (p. 366). ■

A useful expression obtained during the course of the proof of the theorem, above, is:

$$w[R(h)] \leq t - w[S(h+1)] - w[D(h)] \leq t - w[S(h+1)] \quad (4.38)$$

Note that the different I/P blocks z_h that cause a transition into a specific (next) state $S(h+1) = S_n$, of weight \tilde{n} , can only differ in DIG(h); this is so because the elements of DIG(h) will not reside in the circuit memory, while all the elements of MIG(h) will do so. Hence MIG(h) has to be fixed because it will be part of S(h+1), which has a specific composition.

The following theorem gives the number of different z_h s that may lead to a specific state of weight \tilde{n} .

Theorem 4.16: Within the central portion of the constrained trellis, the number of input blocks z_h that cause a transition to a specific state $S(h+1) = S_n$, is $C(f,0) + C(f,1) + \dots + C(f,a)$, where $a \triangleq t - w[S(h)] - w[M(h)] = t - w[S(h+1)] - w[R(h)]$.

Proof: From the above discussion, the number of I/P blocks z_h differ only in their DIG part (i.e. the group of I/P bits

that will not reside in the circuit memory). The size of this group is f ($f \geq 0$). Hence there are $C(f, i)$ different $DIG(h)$ s of weight i , where i may vary between 0 and the maximum permissible weight of $D(h)$, the state of $DIG(h)$.

From reln (4.38),

$$w[D(h)] \leq t - w[S(h+1)] - w[R(h)] \quad (A)$$

From Theorem 4.14,

$$MEG(h+1) = \langle MIG(h), FEG'(h), CEG(h) \rangle \longrightarrow$$

$$MEG(h+1) = MIG(h) \cup FEG'(h) \cup CEG(h) \longrightarrow$$

$$MEG(h+1) \cup REG(h) = MIG(h) \cup FEG'(h) \cup CEG(h) \cup REG(h)$$

$$\longrightarrow MEG(h+1) \cup REG(h) = MIG(h) \cup MEG(h) \quad (4.39)$$

Since $REG(h)$ is mutually exclusive with $FEG'(h)$ [see (4.33)] and with $CEG(h)$ (see Definition 3.1) and with $MIG(h)$, it will also be mutually exclusive with $MEG(h+1)$ (which is partitioned by these three sets). Also, $MIG(h)$ & $MEG(h)$ are obviously mutually exclusive.

$$\langle MEG(h+1), REG(h) \rangle = \langle MIG(h), MEG(h) \rangle \quad (4.40)$$

Applying Theorem 4.9 to reln (4.40),

$$w[S(h+1)] + w[R(h)] = w[M(h)] + w[S(h)] \quad (4.41)$$

From relns (A) & (4.41),

$$w[D(h)] \leq t - w[S(h+1)] - w[R(h)] = t - w[S(h)] - w[M(h)] \triangleq \alpha$$

Note that if $\alpha > f$, then when i (which ranges between 0 & f) exceeds f , $C(f, i) = 0$.

QED

Lemma 4.11: Consider a normal LSC with weight-constraint t and total memory M . Within the central portion of its constrained trellis there are exactly $C(M, t)$ states that have a single I/P (and O/P) transition; these states have weight t , while the transition from any of these states is caused by $z_h = 0$.

Proof: According to Theorem 4.10 the sum of the Hamming weights of the current I/P, z_h , and the current state, $S(h)$, cannot exceed t . Hence, if $w[S(h)] = t \longrightarrow w[z_h] = 0$

$\implies z_h = 0$. So when the circuit memory contains t 'ones' only one I/P block is allowed, hence there is only one transition to and from a state of weight t and there are $C(M,t)$ such states.

QED

Note that this situation, where a state has only one I/P (and O/P) branch, is unique. It is the result of the restriction introduced by Note 4.3.

Usually, a node exists where there is choice. In the case discussed above there is only one path in and one path out of the node, hence there is no decision to be made. This type of nodes can be eliminated. Inevitably, such a 'simplified' trellis diagram will include interstate transitions of length greater than one. The reader should recall* that in an encoder trellis, all node-to-node transitions have length one.

4.4.3. The Simplified Trellis

The results in this paragraph are applications of the constrained-trellis results of the previous sections. They are largely based on the observation that some states have only one incoming and one outgoing path.

Definition 4.4: In the constrained-trellis, unlike in the ordinary one, the states are arranged according to weight. A *region of weight w* is the set of all states of weight w ($0 \leq w \leq t$).

Note that the regions are arranged starting from the $w=0$ one on top and ending with the $w=t$ one in the bottom. Inside a region the states may be arranged in any way, suitable to the application at hand; one possibility is the way used in the ordinary trellis.

In Lemma 4.11, above, it was mentioned that when the circuit is at a state of weight t , only the all-zero I/P block $z_h=0$ is permitted. Hence, once the normal LSC finds itself in the $w=t$ region, it will be driven by an all-zero bit-

* See Definition 3.2 (p. 68).

stream until it exits from this region. To put it otherwise, the circuit will be in its so-called *autonomous* state for as long as its memory contains t 'ones'.

The duration of the stay of the normal LSC in the autonomous state may vary between one time-unit and a maximum of m time-units. More precise information cannot be obtained for the general case because it depends on the way t 'ones' can be arranged in a set of k -f SRs of lengths varying between 1 & m . Nevertheless, some results for special cases will be developed.

The simplified trellis, discussed above, is easily constructed from the ordinary one by removing the weight- t states and modifying the diagram as necessary:

Note 4.7: Given the constrained-trellis of a normal LSC, with total memory M and weight-constraint t , the corresponding *simplified trellis* is obtained from the original constrained-one as following:

Step 1: All nodes corresponding to states of weight t [there are exactly $C(M,t)$ such states] are removed from the diagram.

Step 2: Step No 1, above, generates state-to-state transitions of length greater than one. These 'long' transitions are made by concatenating all single paths that come into contact after removing the weight- t nodes.

Step 3: The label of a 'long' transition is the concatenation of the corresponding single-path labels.

Step 4: Any transitions that start from, and end to, the same state are replaced by the corresponding multiple-edge transition.

Example 4.2: Consider the constrained-trellis diagram of Fig. A4.5.3 (p. 360). Note that all states, apart from S_0 , have one I/P and one O/P transition. If the above mentioned modifications are adopted, only S_0 will remain. From Fig. A4.5.3, one can deduce that the simplified trellis will have only three transitions, all from S_0 to S_0 . One will

have length 1 and label "0/0", the other will have length 2 and label "20/11" (which is the concatenation of "2/1" & "0/1"); finally, the third label will be a double-edge one, of length 3: labels "3/1", "0/1" & "0/1" collapse to "300/111" and labels "1/1", "0/0" & "0/1" collapse to "100/101". Fig. 4.5, shows the new diagram, the simplified trellis.

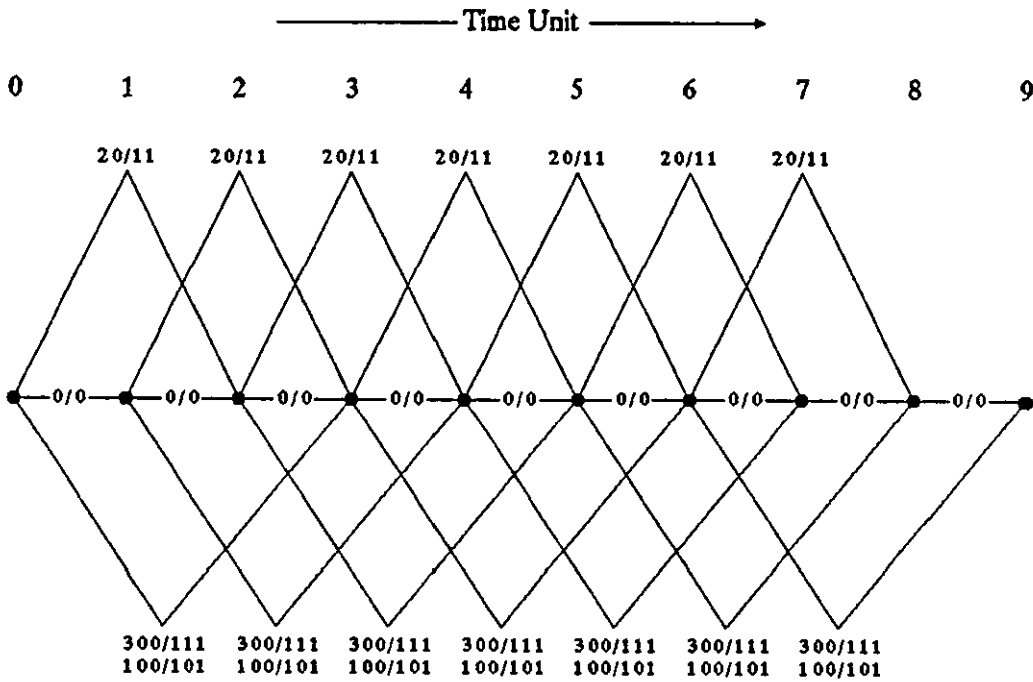


Figure 4.5: Simplified trellis diagram corresponding to the constrained trellis of Fig. A4.5.3 (p. 360).

Note that within the central portion of the simplified trellis, there are three transitions entering each state and three transitions leaving it.

The simplified and the constrained trellis diagrams differ in that the weight-t region of the former has been removed according to Note 4.7. This modification has generated transitions of length greater than one. Hence, any such transition is associated with states of weight t. In fact, the instructions for the construction of the simplified trellis, from the constrained trellis (see Note 4.7), lead easily to the following theorem:

Theorem 4.17: Consider a simplified trellis* and any particular transition of length $\beta > 1$. In the corresponding constrained-trellis* this long transition will correspond to a sequence of β transitions, of which the 1st enters the weight- t region, the last leaves the region and the rest are transitions between states of the region.

Although Theorem 4.17, above, is neither difficult to prove (since it is based on the construction notes for the simplified trellis), nor very useful by itself, it forms the backbone for a number of interesting results.

Theorem 4.18: The longest transition in the simplified trellis* of an (n,k,m) normal LSC cannot exceed $m+1$ time-units. This maximum can be achieved only if the circuit contains at least t SRs of length m (where t is the circuit's weight-constraint).

Proof: According to Theorem 4.17, any transition of length $\beta > 1$ is associated with a sequence of β single-path transitions in the corresponding constrained trellis. Furthermore, the 1st of the transitions brings the circuit in the weight- t region and the β th takes it out of the region. This means that the last 'ones' are injected in the circuit memory during the 1st single-path transition.

Since the circuit memory contains the maximum permissible number of 'ones' (i.e. t - see Theorem 4.10), during the remaining $\beta-1$ time-units no other 'ones' are injected, i.e. the circuit will be in its autonomous state. The 'long' transition terminates when the circuit memory leaves the region, i.e. when at least one 'one' leaves the memory. Hence, the length of the longest possible transition will correspond to the maximum possible number of time-units the circuit can keep t 'ones' while in its autonomous state.

According to Definition 4.2, the normal LSC and the encoder have the same transfer-function matrix $P(D)$ (which for the encoder is called generator-polynomial matrix). Hence, the encoder-synthesis instructions, given in Note 2.9, hold true for the normal LSC, as well. According to Note 2.9,

* The central portion of the trellis is, only, considered.

there is at least one SR of length m . Any particular 'one' can remain in this SR for at most m time-units, i.e. it needs $m+1$ units to clear off this register. The maximum number of time-units t 'ones' can reside in the circuit is determined by the 'one' which at time-unit 1 is closest to the end of its own SR. So, at time-unit 1, the t 'ones' should reside as far left as possible, in an as long as possible SR. In other words, right after the 1st single-path transition each of the t 'ones' should be at the first stage of an SR of length m .

Clearly, the circuit should contain at least t SRs of length m ; also these t 'ones' need $m+1$ time-units to clear of the circuit memory.

QED

4.5 CHARACTERISTICS OF THE GENERAL CONSTRAINED TRELLIS

In the last four sections numerous aspects of the constrained and simplified trellises were discussed and a number of examples were given. The accumulated experience can then assist in a deeper and more abstract elaboration of the structure of the constrained state-transition diagram.

4.5.1. General Partition of the Memory of a Normal LSC

The memory partition in FEG', CEG & REG (or FEG, CEG & REG'), firstly introduced by Definition 3.1, was very successful in explaining the mechanics of the state diagram. In this paragraph this idea is reformulated in an abstract way and is also extended with the introduction of a few more parameters, like the memory-density function.

Definition 4.5: For a normal LSC, the *memory-density function* $f(i)$ denotes the number of shift registers of length i / $i \geq 0$.

Theorem 4.19: Consider an (n, k, m) normal LSC, with total memory M . Then the following results hold true:

$$\text{i)} \quad f(0) = f \quad (4.42a)$$

$$\text{ii)} \quad f(i) = 0 \quad \text{for } i \notin [0, m] \quad (4.42b)$$

$$\text{iii)} \quad f(m) \geq 1 \quad (4.42c)$$

$$\text{iv)} \quad \sum_{i=0}^m f(i) = k \quad (4.42d)$$

$$\text{v)} \quad \sum_{i=1}^m if(i) = M \quad (4.42e)$$

Proof: The results follow easily from Definition 4.5. ■

Consider, at first, the partition of the circuit memory and of the current I/P block, shown in Fig. 4.6. Note that the circuit-memory Partition II (see Theorem 4.13), and the input-block partition into MIG & DIG [defined by eqns (4.23)], have been adopted.

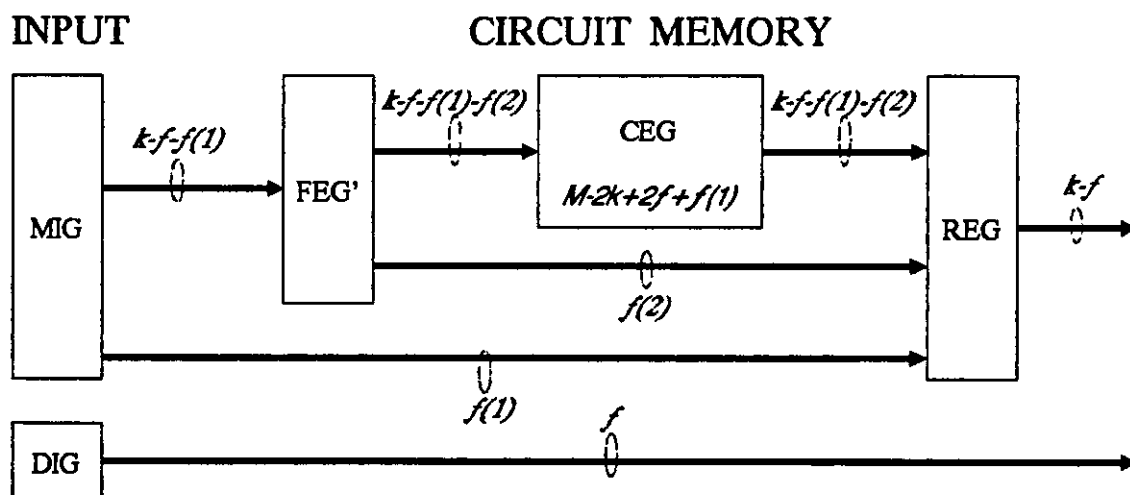


Figure 4.6: Partition of input block and circuit memory, of a normal LSC.

Referring to Fig. 4.6, the reader should note the following: Each box represents a parallel-in parallel-out memory block; all blocks impose a one time-unit delay, with the exception of CEG whose delay per bit varies between one and $m-2$ time-units; the size of each transition line is denoted by their label, while the number displayed in the CEG de-

notes its memory size.

According to the above notes, the memory size of the other blocks is: $k-f$ for the MIG, f for the DIG, $k-f-f(1)$ for the FEG' and $k-f$ for the REG. The transition lines represent a one time-unit parallel transition of the marked number of bits. Finally, the transitions in the RHS of the diagram represent bits that are discarded, i.e. bits that leave the memory.

The idea of the partition illustrated in Fig. 4.6, has a general application in life: For each process, the next state depends on the current action (I/P) and the current state. A part of the current action (DIG) does not participate in the shaping of the next state. Part of the current state (REG) will not influence the next state, while part of the next state (FEG) will be entirely due to the current action.

4.5.2. Description of the Current State

The current paragraph will discuss the current state, and specifically its partition and enumeration.

According to the adopted partitions:

$$\begin{aligned} \text{MEG}(h) &= \langle \text{FEG}'(h), \text{CEG}(h), \text{REG}(h) \rangle \\ \text{ING}(h) &= \langle \text{MIG}(h), \text{DIG}(h) \rangle \end{aligned} \quad \left. \vphantom{\begin{aligned} \text{MEG}(h) &= \langle \text{FEG}'(h), \text{CEG}(h), \text{REG}(h) \rangle \\ \text{ING}(h) &= \langle \text{MIG}(h), \text{DIG}(h) \rangle \end{aligned}} \right\} \rightarrow \quad (4.43)$$

Let φ be the weight of the current state. Then:

$$0 \leq w[S(h)] \hat{=} \varphi \leq t \quad (4.44)$$

Let τ denote the weight of $R(h)$. Obviously $0 \leq \tau \leq \varphi \leq t$, but since REG has $k-f$ positions, τ has to satisfy $0 \leq \tau \leq \text{MIN}\{k-f, \varphi\}$; since also the rest $\varphi - \tau$ 'ones' have to reside in the rest of the memory $\varphi - \tau \leq M - k + f \implies \tau \geq \varphi + k - M - f$.

$$\text{Hence: } \text{MAX}\{0, \varphi + k - M - f\} \leq w[R(h)] \hat{=} \tau \leq \text{MIN}\{k-f, \varphi\} \quad (4.45)$$

If τ satisfies (4.45) then there are $C(k-f, \tau)$ ways to place the τ 'ones' in the REG, and for each of these there are $C(M-k+f, \varphi - \tau)$ ways to place the rest of the 'ones' in the rest of the memory.

So the total number of states of weight φ , with τ 'ones' in the REG, is

$$C(k-f, \tau) C(M-k+f, \varphi-\tau) = \binom{k-f}{\tau} \binom{M-k+f}{\varphi-\tau} \quad (4.46)$$

Note that it is not necessary to restrict τ with (4.45). One can adopt a more relaxed range for τ , i.e. $0 \leq \tau \leq \varphi$ and use the fact that $C(a,b)=0$ if $a < b$.

4.5.3. Transition from a Given State

During the next transition the contents of the REG will be discarded, hence the next state's weight will be reduced by $w[R(h)]$, but on the other hand it will be increased by $w[M(h)]$, i.e. by the number of 'ones' in the part of the I/P block that will reside in the circuit memory.

$$\text{Hence, } w[S(h+1)] = w[S(h)] - w[R(h)] + w[M(h)] \quad (4.47)$$

Theorem 4.20: Consider the central portion of the constrained trellis of an (n,k,m) normal LSC, with weight-constraint t . If the current state, $S(h)$, is given, the weight of the next state, $S(h+1)$, satisfies the inequality

$$\varphi - \tau \leq w[S(h+1)] \leq \text{MIN}\{t-\tau, k-f+\varphi-\tau\} \quad (4.48)$$

$$\text{where: } \varphi \triangleq w[S(h)] \quad \& \quad \tau \triangleq w[R(h)]$$

Proof: For a given current state $S(h)$, φ & τ are fixed; hence, the weight, \tilde{n} , of the next state depends on $\mu \triangleq w[M(h)]$, according to eqn (4.47): $\text{MAX}\{\tilde{n}\} = \varphi - \tau + \text{MAX}\{\mu\}$ & $\text{MIN}\{\tilde{n}\} = \varphi - \tau + \text{MIN}\{\mu\}$.

Obviously $\text{MIN}\{\mu\} = 0$, hence: $\text{MIN}\{\tilde{n}\} = \varphi - \tau$.

There are two restrictions on μ . Obviously, it should not exceed the capacity of the MIG (i.e. $\mu \leq k - f$) and it should not violate the fundamental restriction of Theorem 4.10, i.e. $\varphi + \hat{i} + \mu \leq t \iff \mu \leq t - \varphi - \hat{i}$, where $\hat{i} \triangleq w[D(h)]$. Since the DIG does not participate in the formation of the next state, restrictions on its weight, \hat{i} , are rather relaxed, hence its value can be arbitrary, within limits. In the case examined \hat{i} will be given the value that allows a greater freedom for μ , which is equivalent with the ability to reach a maximum number of states from the current given

state:

$$\mu \leq \text{MAX}\{t - \varphi - \hat{i}\} = t - \varphi - \text{MIN}\{\hat{i}\} = t - \varphi$$

$$\text{Hence: } \text{MAX}\{\tilde{n}\} = \varphi - \tau + \text{MIN}\{t - \varphi, k - f\} = \text{MIN}\{t - \tau, k - f + \varphi - \tau\}$$

QED

Note that a restriction on μ was considered during the discussion of the proof of Theorem 4.20:

$$0 \leq \mu \leq \text{MIN}\{k - f, t - \varphi\} \quad (4.49)$$

Consider now the total number of transitions from any given state $S(h)$ to any state of given weight, say, \tilde{n} (\tilde{n} should satisfy, of course, Theorem 4.20).

Since $S(h)$ is given so is its weight φ and the distribution of the φ 'ones' in the memory of the associated circuit. Hence φ & τ are given as well; if \tilde{n} ($= w[S(h+1)]$) is given, then μ is also given, since eqn (4.47) determines that $\mu = \tilde{n} + \tau - \varphi$.

Note that the various transitions leaving a specific state are generated by 'playing' with the I/P block. In this case, the only restriction on z_h is on its weight. Specifically, $w[z_h]$ must satisfy Theorem 4.10 and eqn (4.47):

$$\hat{i} + \mu + \varphi \leq t \quad \& \quad \mu = \tilde{n} + \tau - \varphi \quad \longrightarrow \quad \hat{i} \leq t - \tilde{n} - \tau$$

Since the size of the DIG is f , \hat{i} may assume any value between 0 and f , provided it does not exceed $t - \tilde{n} - \tau$.

$$\text{Hence,} \quad 0 \leq \hat{i} \leq \text{MIN}\{f, t - \tilde{n} - \tau\} \quad \left. \begin{array}{l} \\ \mu = \tilde{n} + \tau - \varphi \end{array} \right\} \longrightarrow \quad (4.50)$$

Note that the contents of the DIG do not reside in the memory, hence they do not participate in the formation of the next state. The DIG provides only the multiplicity of the labels. On the other hand, the chosen combination of μ 'ones' will determine the next state, hence the MIG specifies $S(h+1)$, although $w[S(h+1)]$ is fixed.

Hence, the number of combinations of μ 'ones' in the MIG equals the number of transitions from $S(h)$ to a state of weight \tilde{n} ; for each such transition the number of single-edge labels (or *label multiplicity*) equals the total number of

combinations permitted in the DIG. If $a \triangleq \text{MIN}\{f, t - \tilde{n} - \tau\}$:

$$\text{Label multiplicity, per transition} = \sum_{i=0}^a C(f, i) \quad (4.51)$$

Obviously, the number of transitions equals the number of combinations of μ 'ones' in the MIG, which is the same with the number of ways one can place μ indistinguishable objects in $k - f$ places. Since $\mu = \tilde{n} + \tau - \varphi$, the number of transitions equals

$$\binom{k-f}{\tilde{n}+\tau-\varphi} \quad (4.52)$$

The results of the above discussion can be summarized in the following theorem:

4.5.4. Summary of Results

Theorem 4.21: Consider the central portion of the constrained trellis of an (n, k, m) normal LSC, with transfer-function matrix $P(D)$, total memory M and weight-constraint t . If $f \triangleq f[P(D)]^*$, the following hold true:

- i) $0 \leq w[S(h)] \triangleq \varphi \leq t$
and $\text{MAX}\{0, \varphi + k - M - f\} \leq w[R(h)] \triangleq \tau \leq \text{MIN}\{k - f, \varphi\}$
- ii) Number of states of weight φ , with an REG
of weight $\tau = \binom{k-f}{\tau} \binom{M-k+f}{\varphi-\tau}$
- iii) Total number of states of weight $\varphi = \binom{M}{\varphi}$
- iv) $\varphi - \tau \leq w[S(h+1)] \triangleq \tilde{n} \leq \text{MIN}\{t - \tau, k - f + \varphi - \tau\}$
- v) Number of transitions from a given state,
of weight φ , to some state of weight $\tilde{n} = \binom{k-f}{\tilde{n}+\tau-\varphi}$
- vi) Label multiplicity, per transition $= \sum_{i=0}^a C(f, i)$
where: $a \triangleq \text{MIN}\{f, t - \tilde{n} - \tau\}$.
- vii) $w[M(h)] \triangleq \mu = \tilde{n} + \tau - \varphi \quad \& \quad 0 \leq \mu \leq \text{MIN}\{k - f, t - \varphi\}$
- iix) $0 \leq w[D(h)] \triangleq \hat{i} \leq \text{MIN}\{f, t - \tilde{n} - \tau\}$

* See Theorem 3.1 (p. 57).

4.6 CHARACTERISTICS OF THE GENERAL SIMPLIFIED TRELLIS

The only difference between the constrained trellis and its simplified version is the removal of the weight- t region from the latter; this generates transitions of length more than one time-unit (the so-called *long transitions* - see Note 4.7).

The existence of the 'long' transitions is a feature of the simplified trellis. Since the rest of the details of the general constrained trellis (and hence of the simplified, as well) were discussed in the previous section, what remains to be examined is the number, length & labels of the 'long' transitions and, if possible, their destination.

4.6.1. Introduction

Consider any given state $S(h)$ of weight $w[S(h)] = w$, and an I/P block z_h that causes a one-time-unit transition to a state $S(h+1)$ of weight t . Since the circuit contains t 'ones' at time-unit $h+1$, it is assumed that no other 'one' will be loaded until it loses at least one of these bits (see Theorem 4.10). Hence, what is important here is the time-unit at which the circuit loses the first of its t 'ones'. This will be determined by the 'distance' (in SR stages) of the foremost 'one' in each of the SRs, from the SR's end. Obviously, the minimum of these 'distances' determines the duration of the stay of the circuit in the $w=t$ region, and consequently the length of the corresponding 'long' transition.

A 'long' transition of length β starts from a state of weight less than t and ends in a state of weight less than t ; hence at the 0th and the β th time-units the circuit contains less than t 'ones', while it contains exactly t 'ones' at time-units $1, 2, \dots, \beta-1$. Note that it makes no sense to expand the term 'long' transition to normal transitions, i.e. of length $\beta=1$. Hence, from now on the 'long' transition will have length $\beta>1$.

Theorem 4.22: Consider the central portion of the simplified trellis for a normal LSC with weight-constraint t . If a 'long' transition is to start at time-unit h , then

$$\begin{aligned} w[D(h)] &= w[R(h)] = 0 \\ w[M(h)] &= t - w[S(h)] \end{aligned} \quad \left. \begin{array}{l} \text{---} \\ \text{---} \end{array} \right\} \rightarrow \quad (4.53)$$

where, it is repeated that, $S(h)$ is the current state of the circuit and $R(h)$, $D(h)$ & $M(h)$ are the current states of the REG, DIG & MIG, respectively.

Proof: During the 1st time-unit (i.e. between time-units 0 & 1) the circuit loses τ 'ones', where $\tau \triangleq w[R(h)]$ and gains μ 'ones', where $\mu \triangleq w[M(h)]$.

There are two restrictions on τ , μ & $\varphi \triangleq [S(h)]$:

Firstly, Theorem 4.10 requires that the combined weight of the I/P block and of the state must not exceed t , at any time, which means that τ , φ and $\hat{i} \triangleq [D(h)]$, must satisfy:

$$\hat{i} + \mu + \varphi \leq t \quad (A)$$

Also, the next state's weight must be t , hence (by Theorem 4.21),

$$\varphi - \tau + \mu = t \quad (B)$$

From relns (A) & (B) it is easily concluded that

$$\hat{i} + \mu + \varphi \leq \varphi - \tau + \mu \quad \Longleftarrow \quad \hat{i} + \tau \leq 0 \quad \Longrightarrow \quad \hat{i} = \tau = 0.$$

Since $\tau = 0$, $\mu = t - \varphi$.

QED

So far, it has been concluded that the I/P block will contain no 'ones' in the DIG and $t-w$ 'ones' in the MIG, while the current state should be such that the REG contains no 'ones'. If the transition is to last exactly β time-units, then between time-units $h+\beta-1$ & $h+\beta$ the circuit must lose its first 'one' (or 'ones'). These 'foremost ones' should be in the last stage of their corresponding SRs at time-unit $h+\beta-1$. Since at time-unit $h+\beta-1$ these bits are at the M_j th stage of their corresponding SR, at time-unit $h+1$ they should have been at the $(M_j-\beta+2)$ th stage.

4.6.2. Preparation

The main conclusion of the previous paragraph is expressed by the theorem below:

Theorem 4.23: Consider the simplified trellis* for a normal LSC with weight-constraint t and SRs of length M_j / $j=1,2,\dots,k$. If a transition of length at least $\beta > 1$ is to start at time-unit h , then at time-unit $h+1$ the circuit must contain exactly t 'ones', restricted at stages $1,2,\dots,(M_j-\beta+2)$ of their corresponding SRs, which must satisfy $M_j-\beta+2 \geq 1$.

From the theorem above, the following three results/conditions, are extracted:

Theorem 4.24: Consider the central portion of the simplified trellis for a normal LSC with weight-constraint t and SRs of length M_j / $j=1,2,\dots,k$. If a transition of length $\beta > 1$, is to start at time-unit h from a state of weight w ($0 \leq w \leq t$), then it is necessary and sufficient that:

- i) At time-unit h , the circuit's memory should contain exactly w 'ones' restricted to stages $1,2,\dots,(M_j-\beta+1)$, of those of the k SRs that have length $M_j \geq \beta$.
- ii) The I/P block z_h should contain exactly $t-w$ 1s, at positions that correspond to SRs of length $\beta-1$ or more.
- iii) Either, a) at least one of the w 'ones' of $S(h)$ must be in the $(M_j-\beta+1)$ th stage of its SR, or b) at least one of the $t-w$ 'ones' of z_h must be in a position that corresponds to an SR of length $\beta-1$.

Proof: Let a transition of length β start at time-unit h . Then, according to Theorem 4.23, at time-unit $h+1$ t 'ones' are stored in the memory; furthermore, these 'ones' are restricted in the first $M_j-\beta+2$ stages of the 'long' SRs (i.e. of the SRs that have at least as many stages).

At time-unit h the circuit is at the given state, of weight w , hence it contains exactly w 'ones'. If at time-unit $h+1$, no 'ones' are to be found beyond the $(M_j-\beta+2)$ th

* The central portion of the trellis is, only, considered.

stage of their corresponding SR, then at the previous time unit the w 'ones' would have been restricted in the first $M_j - \beta + 1$ stages of their corresponding SRs. The SRs that can accommodate 'ones' at time-unit h are, obviously, those for which stages $1, 2, \dots, (M_j - \beta + 1)$ do exist, i.e. those SRs for which $M_j - \beta + 1 \geq 1 \iff M_j \geq \beta$.

According to Theorem 4.22, the I/P block z_h must contain $t-w$ 'ones' all in the MIG. The only remaining question now is, in which positions of the MIG are these $t-w$ 'ones' allowed to be placed? Firstly, there is no restriction because of the existence of the w 'ones' in the circuit memory. Even if some of the w 'ones' do reside in the FEG at time-unit h , they will evacuate it during the $S(h) \rightarrow S(h+1)$ transition, so that all the $k-f$ positions of the FEG can accept the I/P 'ones'. The restriction is imposed by Theorem 4.23. At time-unit $h+1$ the $t-w$ I/P 'ones' should not be beyond the $(M_j - \beta + 2)$ th stage of their SR. Since these bits are launched at the 1st stage of their SR, anyway, the restriction is reduced to the choice of SRs whose 1st stage does not exceed $(M_j - \beta + 2)$: $(M_j - \beta + 2) \geq 1 \iff M_j \geq \beta - 1$.

The first two restrictions consider the case of a transition of length β , or more. If this transition is to last exactly β time-units, then at least one 'one' should leave the memory at time-unit $h+\beta$ or, the same, at time-unit $h+1$ there must be at least one 'one' in the $(M_j - \beta + 2)$ th stage of its SR. The question now is, where this bit will originate from? The t 'ones' that exist in the circuit at time-unit $h+1$ come

- i) either from the I/P block z_h (exactly $t-w$ of them), or
- ii) from the 'early' stages of the circuit memory (exactly w of them).

So, either at least one 'one' of the I/P block should be in a position corresponding to an SR of length $\beta - 1$, or at least one 'one' of the current state should be in stage $(M_j - \beta + 1)$ of its SR.

QED

According to part (i) of Theorem 4.24, states of weight w from which transitions of length β or more may start, can be

generated by placing w 'ones' in stages $1, 2, \dots, (M_j - \beta + 1)$, of those SRs with length β or more. At this point it would be useful to define some new terms:

Definition 4.6: Consider an (n, k, m) normal LSC and its set of SR stages, MEG^* . The REG was defined as the part of the circuit-memory made of the last stage of each SR. An extension of this concept is the *REG of order i* , denoted by $REG(i,)$, $1 \leq i \leq m$, which is defined to be the set of the last i stages of each SR. To preserve compatibility, the contents of the $REG(i,)$, at time-unit h , are denoted by $REG(i, h)$. ■

Definition 4.7: Consider an (n, k, m) normal LSC and its input group, ING^* (i.e. the $k \times 1$ memory which holds the current input block, prior to its shifting into the LSC). The DIG was defined as the part of the I/P block that corresponds to SRs of length 0. An extension of this concept is the *DIG of order i* , denoted by $DIG(i,)$, $1 \leq i \leq m$, which is defined to be the set of the I/P block positions that correspond to SRs of length i or less. To preserve compatibility, the contents of the $DIG(i,)$, at time-unit h , are denoted by $DIG(i, h)$. ■

Note here that the REG is the REG of order one, while $REG(1, h) = REG(h)$. Similarly, the DIG of order 0 is the DIG, while $DIG(0, h) = DIG(h)$. A quantity of particular importance is the number of elements in the previously-defined sets.

$REG(i,)$ & $DIG(i,)$ are sets of memory SR-stages. $REG(i, h)$ & $DIG(i, h)$ are sets of bits residing in $REG(i,)$ & $DIG(i,)$, respectively, at time-unit h . These four sets can, also, be thought of as functions of i (and of h , for the first two).

Notation: Let A represent any part of the memory of a circuit. Then $|A|$ denotes the number of SR-stages in A .

Let $f(i)$ be any function of an integer variable i . Then the difference $f(i) - f(i-1)$ will be denoted by $\delta f(i)$. ■

* See also Definition 3.1 (p. 58) & Definition 4.3 (p. 87).

For example, since MEG represents the total of the circuit's memory, then $-MEG = \emptyset$, while $|MEG| = M$. Also, $\delta REG(i,) = REG(i,) - REG(i-1,)$, etc. Also, from Definition A4.2.2, $MEG - REG(i,) = -REG(i,)$, etc.

The following lemma defines the parts of the memory and the I/P block that are allowed to accomodate 'ones', if a 'long' transition is to be launched. This lemma is a combination between Theorem 4.24 and Definitions 4.6 & 4.7.

Lemma 4.12: Consider the central portion of the simplified trellis for a normal LSC with weight-constraint t and SRs of length M_j / $j=1,2,\dots,k$. If a transition of length β ($\beta > 1$) is to start at time-unit h , from a state of weight w , then:

- i) At time-unit h , the w 'ones' of the circuit's memory should be concentrated in the $-REG(\beta-1,)$ ($\neq \emptyset$).
- ii) The 'ones' of the I/P block z_h should be concentrated in the $-DIG(\beta-2,)$ ($\neq \emptyset$).
- iii) At least one 'one' should reside either in the $\delta REG(\beta,)$ ($\neq \emptyset$), or in the $\delta DIG(\beta-1,)$ ($\neq \emptyset$).

Proof: According to Definition 4.6, the $REG(\beta-1,)$ is made of stages $M_j, M_j-1, \dots, M_j-\beta+2$ of each SR, hence its complementary, $-REG(\beta-1,)$, will be made of SR stages $1, 2, \dots, M_j-\beta+1$. Similarly, according to Definition 4.7, the $DIG(\beta-2,)$ is made of the I/P block positions that correspond to SR lengths $0, 1, \dots, \beta-2$. Hence, the $-DIG(\beta-2,)$ will be made of the I/P block positions that correspond to SRs of length $\beta-1, \dots, m$.

Finally, the $\delta REG(\beta,)$ is made of the $(M_j-\beta+1)$ th stage of each SR. Similarly, the $\delta DIG(\beta-1,)$ contains those I/P block position(s) that correspond to SR(s) of length $\beta-1$.

QED

4.6.3. Intermediate Results

This paragraph will use the tools introduced by the previous paragraphs to produce some intermediate results, which in turn will be useful for the conclusion. Specific-

ly, the number of stages in the REG(i,) and in the DIG(i,) are parameters of particular importance.

In order to assist the presentation of the results the memory-distribution function, F , is defined below. Clearly, f & F have been modelled on the probability density function and the probability distribution function.

Definition 4.8: For a normal LSC, the *memory-distribution function* $F(i)$ denotes the number of shift registers of length i or less:

$$F(i) \triangleq \sum_{j=0}^i f(j) \quad / j \geq 0 \quad (4.54)$$

Theorem 4.25: Consider an (n, k, m) normal LSC, with total memory M . Then the following relationships hold true:

$$i) \quad f(i) = \begin{cases} F(i) & \text{if } i=0 \\ \delta F(i) & \text{if } i>0 \end{cases} \quad (4.55a)$$

$$ii) \quad F(m) = k \quad (4.55b)$$

$$iii) \quad \sum_{j=0}^m F(j) = (m+1)k - M \quad (4.55c)$$

Proof: See Appendix 4.9 (§ A4.9.1., p. 368).

The following two lemmas will provide some useful intermediate results:

Lemma 4.13: Consider an (n, k, m) normal LSC with total memory M . Then the following relationships hold true, for all $\beta \in [1, m]$:

$$\sum_{i=1}^{\beta} i f(i) = \beta F(\beta) - \sum_{i=0}^{\beta-1} F(i) \quad (4.56a)$$

$$\sum_{i=\beta}^m i f(i) = M - \beta F(\beta-1) + \sum_{i=0}^{\beta-1} F(i) \quad (4.56b)$$

Proof: See Appendix 4.9 (§ A4.9.2., p. 368).

Lemma 4.14: Consider any two parts A & B of a circuit's memory. Then

$$|A - B| = |A \cup B| - |B| \quad (4.57a)$$

and if, $B \subseteq A^*$, then: $|A - B| = |A| - |B| \quad (4.57b)$

where, if $X \cap Y = \emptyset$, then $|X| + |Y| = |X \cup Y| \quad (4.58)$

Proof: See Appendix 4.9 (§ A4.9.3., p. 369). ■

Theorem 4.26: With respect to an (n, k, m) normal LSC, with total memory M, for all $\beta \in [1, m]$:

$$i) \quad |REG(\beta,)| = \beta k - \sum_{i=0}^{\beta-1} F(i) \quad (4.59a)$$

$$ii) \quad |REG(1,)| = k - f \quad \& \quad |REG(m,)| = M \quad (4.59b)$$

$$iii) \quad |-REG(\beta,)| = M - |REG(\beta,)| = M - \beta k + \sum_{i=0}^{\beta-1} F(i) \quad (4.59c)$$

$$iv) \quad |\delta REG(\beta,)| = k - F(\beta-1) \quad (4.59d)$$

$$v) \quad |DIG(\beta,)| = F(\beta) \quad (4.59e)$$

$$vi) \quad |DIG(0,)| = f \quad \& \quad |DIG(m,)| = k \quad (4.59f)$$

$$vii) \quad |-DIG(\beta,)| = k - |DIG(\beta,)| = k - F(\beta) \quad (4.59g)$$

$$iix) \quad |\delta DIG(\beta,)| = f(\beta) \quad (4.59h)$$

Proof: By Definition 4.6, the $REG(\beta,)$ is made of the last β stages of each SR, i.e. of stages $M_i - \beta + 1, M_i - \beta + 2, \dots, M_i$ for all $i \in [1, k]$ for which $M_i - \beta + 1 \geq 1 \iff M_i \geq \beta$, where $\beta \in [1, m]$. Since $REG(\beta-1,)$ is made of stages $M_i - \beta + 2, M_i - \beta + 3, \dots, M_i$ then the relative complement, $\delta REG(\beta,)= REG(\beta,)-REG(\beta-1,)$, will be made of those stages of the $REG(\beta,)$ that do not belong to $REG(\beta-1,)$, i.e. of the $(M_i - \beta + 1)$ th stage of each SR that has length $M_i \geq \beta$. $|\delta REG(\beta,)|$ is then equal to the number of SRs that have length at least β , or since the number of SRs is k , $|\delta REG(\beta,)|$ equals k minus the number of SRs that have length at most $\beta-1$ [the latter quantity is $F(\beta-1)$, according to Definition 4.8]. Hence,

* $BCA = "B \text{ is a subset of } A".$

$$|\delta \text{REG}(\beta,)| = k - F(\beta-1) \quad (d)$$

Since $\text{REG}(\beta-1,) \subset \text{REG}(\beta,)$, then according to Lemma 4.14 and eqn (d):

$$|\delta \text{REG}(i,)| = |\text{REG}(i,)| - |\text{REG}(i-1,)| = k - F(i-1) \quad /i=2, \dots, m$$

$$\implies \sum_{i=2}^{\beta} |\text{REG}(i,)| - \sum_{i=1}^{\beta-1} |\text{REG}(i,)| = \sum_{i=1}^{\beta-1} [k - F(i)]$$

$$\implies |\text{REG}(\beta,)| - |\text{REG}(1,)| = (\beta-1)k - \sum_{i=1}^{\beta-1} F(i)$$

$\text{REG}(1,)$ is simply the REG, which contains $k-f$ stages. Since, $f = F(0)$ [see eqns (4.42a) & (4.55a)]:

$$\implies |\text{REG}(\beta,)| = \beta k - \sum_{i=0}^{\beta-1} F(i) \quad (a)$$

Eqns (b) are trivial cases, while eqn (c) is based on the fact that $\text{MEG} - \text{REG}(i,) = -\text{REG}(i,)$ and on eqn (a).

According to Definition 4.7, $|\text{DIG}(\beta,)|$ is the number of SRs of length β or less and this is simply $F(\beta)$, according to Definition 4.8:

$$|\text{DIG}(\beta,)| = F(\beta) \quad (e)$$

From eqn (e), $|\text{DIG}(0,)| = F(0) = f$ [by eqns (4.55a) & (4.42a)] and [by eqn (4.55b)]: $|\text{DIG}(m,)| = F(m) = k$ (f)

Since $\text{DIG}(i-1,) \subset \text{DIG}(i,)$, then according to Lemma 4.14 and eqns (e) & (4.55a) (the latter eqn holds true since $\beta \in [1, m]$):

$$|\delta \text{DIG}(\beta,)| = |\text{DIG}(\beta,)| - |\text{DIG}(\beta-1,)| = F(\beta) - F(\beta-1)$$

$$\text{Hence:} \quad |\delta \text{DIG}(\beta,)| = f(\beta) \quad (h)$$

Finally, since $-\text{DIG}(\beta,) = \text{ING} - \text{DIG}(\beta,)$ (see Definition 4.3): $|\text{ING} - \text{DIG}(\beta,)| = |\text{ING}| - |\text{DIG}(\beta,)| = k - F(\beta)$ (g)

QED

4.6.4. Conclusions

The results of the previous paragraphs will now be combined to produce the main theorem of this section.

Consider an (n, k, m) normal LSC, with total memory M and

weight-constraint t . Let a transition of length $\beta > 1$ start at time-unit h , from a state of weight w .

According to Lemma 4.12, the w 'ones' of the memory should reside in the $-\text{REG}(\beta-1,)$.

Hence, there are $\left(|-\text{REG}_w(\beta-1,)| \right)$ such states.

Consider now any specific state from which a transition of length β may start. There are two possibilities with respect to such a state. Since $\delta\text{REG}(\beta,)$ is a subset of $-\text{REG}(\beta-1,)$,

- i) either there is no 'one' in the $\delta\text{REG}(\beta,)$, or
- ii) there is at least one 'one' in the $\delta\text{REG}(\beta,)$.

This distinction is important because if the transition is to have length β , exactly, then the state has to be such that (ii) above holds true (see Lemma 4.12). If though (i) holds true, the I/P block should be such that at least one 'one' resides in the $\delta\text{DIG}(\beta-1,)$.

i) There are no 'ones' in the $\delta\text{REG}(\beta,)$. Then, this means that the w 'ones' of the memory are concentrated in the rest of the permitted region, i.e. in

$$-\text{REG}(\beta-1,) - \delta\text{REG}(\beta,) = -\text{REG}(\beta-1,) - [\text{REG}(\beta,) - \text{REG}(\beta-1,)]$$

$$\implies -\text{REG}(\beta-1,) - \delta\text{REG}(\beta,) = -\text{REG}(\beta,) \quad (\text{A})$$

The result, above, is based on Theorem A4.2.2 [eqn (c)] and on the fact that $\text{REG}(\beta-1,) \subseteq \text{REG}(\beta,)$.

So, there are $\left(|-\text{REG}_w(\beta,)| \right)$ such states.

ii) There is at least one 'one' in the $\delta\text{REG}(\beta,)$. The number of different ways one can place w 'ones' in a certain region R so that at least one 'one' is in a specific part A of that region, equals the total number of ways the w 'ones' can be placed in R , minus the number of ways the w 'ones' can be placed in R so that no 'one' is placed in A ; the latter equals the number of ways the w 'ones' can be placed in $R-A$ (which is the number of 'unacceptable' cases). In the case under examination, R is $-\text{REG}(\beta-1,)$ and A is $\delta\text{REG}(\beta,)$. Then the 'unacceptable' region [according to eqn (A), above] is $-\text{REG}(\beta-1,) - \delta\text{REG}(\beta,) = -\text{REG}(\beta,)$ and therefore:

There are $\left(\left| -\text{REG}_w(\beta-1,) \right| \right) - \left(\left| -\text{REG}_w(\beta,) \right| \right)$ such states.

From each of these states there is a number of transitions of length β . This depends on the number of permitted I/P blocks. This, in turn, is defined by Lemma 4.12, part (ii). The t-w 'ones' of the I/P block should be concentrated in the $-\text{DIG}(\beta-2,)$.

Hence, there are $\left(\left| -\text{DIG}_{t-w}(\beta-2,) \right| \right)$ such transitions.

i) Returning to the 1st case, for a transition of length β to start, at least one 'one' should reside in the $\delta\text{DIG}(\beta-1,)$ [see Lemma 4.12, part (iii)]. Then, using the same argument as above, one can count the number of 'acceptable' I/P blocks by subtracting the number of the 'unacceptable' ones, from the total. The 'unacceptable' region is $-\text{DIG}(\beta-2,) - \delta\text{DIG}(\beta-1,) = -\text{DIG}(\beta-2,) - [\text{DIG}(\beta-1,) - \text{DIG}(\beta-2,)] = -\text{DIG}(\beta-1,)$, where the above result is based on Theorem A4.2.2 and on the fact that $\text{DIG}(\beta-2,) \subseteq \text{DIG}(\beta-1,)$. So:

There are $\left(\left| -\text{DIG}_{t-w}(\beta-2,) \right| \right) - \left(\left| -\text{DIG}_{t-w}(\beta-1,) \right| \right)$

transitions of length β out of such a state.

If the results of Theorem 4.26 are also used, the following has been proved:

Theorem 4.27: Consider the central portion of the simplified state-transition diagram of a normal LSC, with total memory M and weight-constraint t. Then,

i) The number of states of weight w, from which a transition of length $\beta > 1$ may start is

$$\left(\left| -\text{REG}_w(\beta-1,) \right| \right) \quad (4.60a)$$

$$\text{where: } \left| -\text{REG}_w(\beta-1,) \right| = M - (\beta-1)k + \sum_{i=0}^{\beta-2} F(i) \quad (4.60b)$$

$$\text{ii) There are } \left(\left| -\text{REG}_w(\beta-1,) \right| \right) - \left(\left| -\text{REG}_w(\beta,) \right| \right) \quad (4.60c)$$

states, of weight w, from which

$$\left(\left| -\text{DIG}_{t-w}(\beta-2,) \right| \right) = \left(k - F_{t-w}(\beta-2) \right) \quad (4.60d)$$

transitions of length $\beta > 1$ start, where:

$$|-REG(\beta,)| = |-REG(\beta-1,)| + F(\beta-1) - k \quad (4.60e)$$

$$\text{iii) There are } \left(|-REG(\beta,)| \right) \quad (4.60f)$$

states, of weight w , from which

$$\left(|-DIG(\beta-2,)| \right) - \left(|-DIG(\beta-1,)| \right) \quad \left(k-F(\beta-2) \right) - \left(k-F(\beta-1) \right) \quad (4.60g)$$

transitions of length $\beta > 1$ start.

Example 4.3: Consider the arbitrary LSC of Fig. 4.7. Then, $m = 4$, $k = 6$ & $M = 12$. The f & F functions have the following values:

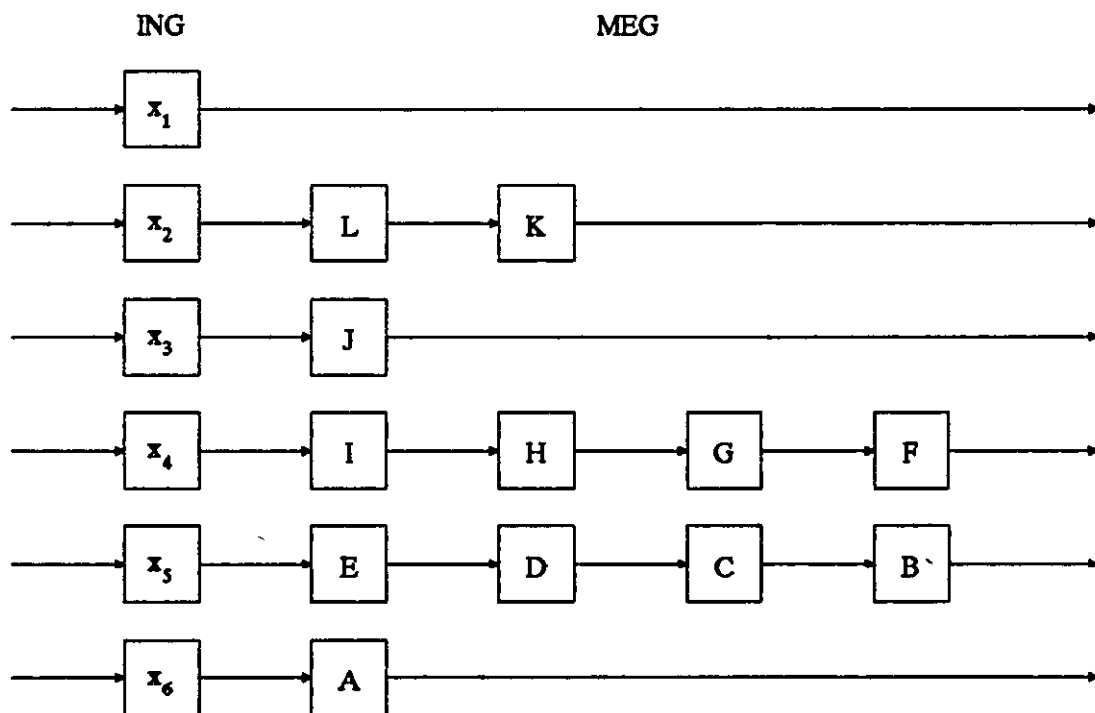


Figure 4.7: The input group (ING) and the memory group (MEG) of an arbitrary normal LSC.

TABLE 4.2

$f(0) = 1$	$f(1) = 2$	$f(2) = 1$	$f(3) = 0$	$f(4) = 2$
$F(0) = 1$	$F(1) = 3$	$F(2) = 4$	$F(3) = 4$	$F(4) = 6$

Note that the f & F functions do indeed satisfy previous results, like Theorems 4.19 & 4.25 and Lemma 4.13.

Consider now the REGs & DIGs of various orders:

TABLE 4.3

$\text{REG}(1,) = \{A, B, F, J, K\}$	$ \text{REG}(1,) = 5$
$\text{REG}(2,) = \{A, B, C, F, G, J, K, L\}$	$ \text{REG}(2,) = 8$
$\text{REG}(3,) = \{A, B, C, D, F, G, H, J, K, L\}$	$ \text{REG}(3,) = 10$
$\text{REG}(4,) = \text{MEG}$	$ \text{REG}(4,) = M = 12$
$\delta\text{REG}(2,) = \{C, G, L\}$	$ \delta\text{REG}(2,) = 3$
$\delta\text{REG}(3,) = \{D, H\}$	$ \delta\text{REG}(3,) = 2$
$\delta\text{REG}(4,) = \{E, I\}$	$ \delta\text{REG}(4,) = 2$
$-\text{REG}(1,) = \{C, D, E, G, H, I, L\}$	$ \text{-REG}(1,) = 7$
$-\text{REG}(2,) = \{D, E, H, I\}$	$ \text{-REG}(2,) = 4$
$-\text{REG}(3,) = \{E, I\}$	$ \text{-REG}(3,) = 2$
$\text{DIG}(0,) = \{x_1\}$	$ \text{DIG}(0,) = 1$
$\text{DIG}(1,) = \{x_1, x_3, x_6\}$	$ \text{DIG}(1,) = 3$
$\text{DIG}(2,) = \{x_1, x_2, x_3, x_6\}$	$ \text{DIG}(2,) = 4$
$\text{DIG}(3,) = \{x_1, x_2, x_3, x_6\}$	$ \text{DIG}(3,) = 4$
$\text{DIG}(4,) = \text{ING}$	$ \text{DIG}(4,) = k = 6$
$\delta\text{DIG}(1,) = \{x_3, x_6\}$	$ \delta\text{DIG}(1,) = 2$
$\delta\text{DIG}(2,) = \{x_2\}$	$ \delta\text{DIG}(2,) = 1$
$\delta\text{DIG}(3,) = \emptyset$	$ \delta\text{DIG}(3,) = 0$
$\delta\text{DIG}(4,) = \{x_4, x_5\}$	$ \delta\text{DIG}(4,) = 2$
$-\text{DIG}(0,) = \{x_2, x_3, x_4, x_5, x_6\}$	$ \text{-DIG}(0,) = 5$
$-\text{DIG}(1,) = \{x_2, x_4, x_5\}$	$ \text{-DIG}(1,) = 3$
$-\text{DIG}(2,) = \{x_4, x_5\}$	$ \text{-DIG}(2,) = 2$
$-\text{DIG}(3,) = \{x_4, x_5\}$	$ \text{-DIG}(3,) = 2$
$-\text{DIG}(4,) = \emptyset$	$ \text{-DIG}(4,) = 0$

Consider transitions of length $\beta = 3$, from a state of weight $w = 2$; let $t = 4$. According to Theorem 4.27,

there are $\left(|\text{-REG}(2,)|_w \right) = \binom{4}{2} = 6$ such states.

From these 6 states of weight 2,

$\left(\left| -\text{REG}_w(2, \cdot) \right| \right) - \left(\left| -\text{REG}_w(3, \cdot) \right| \right) = \binom{4}{2} - \binom{2}{2} = 5$ states have
 $\left(\left| -\text{DIG}_{t-w}(1, \cdot) \right| \right) = \binom{3}{2} = 3$ transitions (of length 3)
 each,

while one state has

$\left(\left| -\text{DIG}_{t-w}(1, \cdot) \right| \right) - \left(\left| -\text{DIG}_{t-w}(2, \cdot) \right| \right) = \binom{3}{2} - \binom{2}{2} = 2$ transitions.

For a transition of length 3 to start from a state of weight 2, if $t=4$ the I/P block must have weight 2. The question is where to place the 2 'ones' of the MEG and where the 2 'ones' of the ING. According to Lemma 4.12, at time-unit h the 2 'ones' of the MEG should be concentrated in the $-\text{REG}(2, \cdot) = \{D, E, H, I\}$. Note that there are indeed 6 ways to place these 2 'ones' in the $-\text{REG}(2, \cdot)$: $(H, D), (H, E), (H, I), (D, E), (D, I), (E, I)$. Note, also, that the first 5 combinations will result in a transition of length 3, provided of course that the 'ones' of the I/P block are concentrated in the $-\text{DIG}(1, \cdot) = \{x_2, x_4, x_5\}$. Hence, for each of these first 5 states there are 3 I/P blocks that cause transitions of length 3: $(x_2, x_4), (x_2, x_5), (x_4, x_5)$. For the 6th state though [with 'ones' in (E, I)], at least one of the 2 I/P-block 'ones' should be in the $\delta \text{DIG}(2, \cdot) = \{x_2\}$. Hence there are two transitions from such a state: (x_2, x_4) & (x_2, x_5) .

Appendix 4.10 (Example A4.10.2, p. 372), contains a second example, on Theorem 4.27.

4.7 CONSTRAINED & SIMPLIFIED TRELLIS FOR SPECIAL CASES

In this last section, simplified results for various special cases will be developed. The general parameter of interest is t and the simplest case is the one of $t=1$. Some other special cases correspond to the value of k ; the simplest case here is the one of an $(n, 1, m)$ circuit. Finally, the case of a circuit with equal-length SRs is of particular interest.

4.7.1. Normal LSC with $t=1$

The following lemma is a special case of Lemma 4.9.

Lemma 4.15: Consider an (n,k,m) normal LSC with total memory M and weight-constraint 1. Then:

$$\Sigma E = M + 1 \quad (4.61a)$$

$$\Sigma O(w) = \begin{cases} k + 1 & \text{if } w=0 \\ 1 & \text{if } w=1 \end{cases} \quad (4.61b)$$

$$\Sigma Y(w) = \begin{cases} k - f + 1 & \text{if } w=0 \\ 1 & \text{if } w=1 \end{cases} \quad (4.61c)$$

So, according to Lemma 4.15 above, the constrained trellis has exactly $M+1$ states; in its central portion, from a state of weight $w=1$ only one other state can be reached, while from S_0 , $k - f + 1$ states can be reached.

Theorem 4.28: Consider an (n,k,m) normal LSC with SR lengths M_1, M_2, \dots, M_k and weight-constraint 1. The central portion of its simplified trellis has the following characteristics:

- i) The only state is S_0 .
- ii) The total number of different labels is $k+1$.
- iii) Its transitions have lengths $1, 1+M_1, 1+M_2, \dots, 1+M_k$, (and hence, they vary between 1 and $m+1$).

Proof: Consider the corresponding constrained trellis and let M denote the total memory of the LSC; according to Lemma 4.15 the total number of states is $M+1$ (one state of weight 0 and M states of weight 1), while according to Lemma 4.11 there are M states with a single O/P (and I/P) transition, which are removed from the diagram in order to generate the simplified trellis (see Note 4.7). That leaves only S_0 .

According to Lemma 4.8 only I/P blocks of weight 0 or 1 are allowed. Obviously there are $k+1$ different blocks and hence $k+1$ labels (note though that the labels might be of various lengths).

The simplified trellis has only one state, S_0 . Hence

transitions are from S_0 to S_0 . Obviously transitions of the same length will merge into a multiple-edge one. According to Theorem 4.10, the I/P block z_h may contain up to one 'one'; this block is injected into the circuit only if the latter is at state S_0 , i.e. with its memory 'clear'. If $z_h = 0$ then the transition triggered, has length 1 because the circuit remains in state S_0 . If $w[z_h] = 1$ then there are k places for the single 'one'; these are the 1st stage of each of the k SRs of the circuit. The transition will end when the circuit returns to state S_0 again and this happens when the single 'one' leaves its SR. If its SR has length one then the transition will have length two because during the 1st time-unit the 'one' is stored in the SR and during the 2nd time-unit it is discarded. In general, if M_i is the length of the i th SR ($1 \leq i \leq k$) then the lengths of the $k+1$ transitions are $1, 1+M_1, 1+M_2, \dots, 1+M_k$. Since $0 \leq M_i \leq m$ (for all $i=1,2,\dots,k$ - see eqn (3.1)) then the minimum transition-length is one and the maximum one is $m+1$.

QED

Finally, the decoding algorithm (which is introduced by Note 4.6) will be modified for the simplified trellis of $t=1$ codes. Note that this special case is very important because at each time-unit there is only one path, hence the decoder memory requirement is very much relaxed. This is the result of the simplification of the constrained trellis.

Nevertheless, it is not easy to deduce the exact memory requirement. This is so because even if ties are not considered, there may exist 'dead-end paths' of various lengths, that need to be stored until it is made certain that they are not needed. This matter will be discussed later on, but it is clear that it needs further investigation.

Note 4.8: To decode:

- i) Let time-unit be $h=0$.
- ii) Calculate the current syndrome block s_h .
- iii) If $\beta+1$ is the length of the branch, for each one, with label $[z_{h-\beta} \dots z_{h-1} z_h] / [q_{h-\beta} \dots q_{h-1} q_h]$, calculate W , where $W \triangleq w([z_{h-\beta} \dots z_{h-1} z_h]) + w([q_{h-\beta} \dots q_{h-1} q_h] + [s_{h-\beta} \dots s_{h-1} s_h])$.

iv) For time-unit $h+1$ and for all paths entering S_0 , keep the one with the smallest metric M_{h+1} and store the path and the metric ($M_{h+1} = M_h + W$).

v) Store any dead branches with their own metric and the time-unit at which they leave the main path. Discard all information about dead branches ending $m+1$ time-units before. A *dead branch* is a part of the main path that is abandoned for another route forward.

vi) If there are more blocks to be decoded, increase h by 1 and go to (iii), otherwise proceed.

vii) Subtract the survivor path (which is the hyperchannel error sequence, \tilde{z}) from $r^{(m)}$, to obtain \tilde{u} .

Example 4.4: Consider the simplified trellis diagram of Example 4.2 (see Fig. 4.5). For comparison, the message and

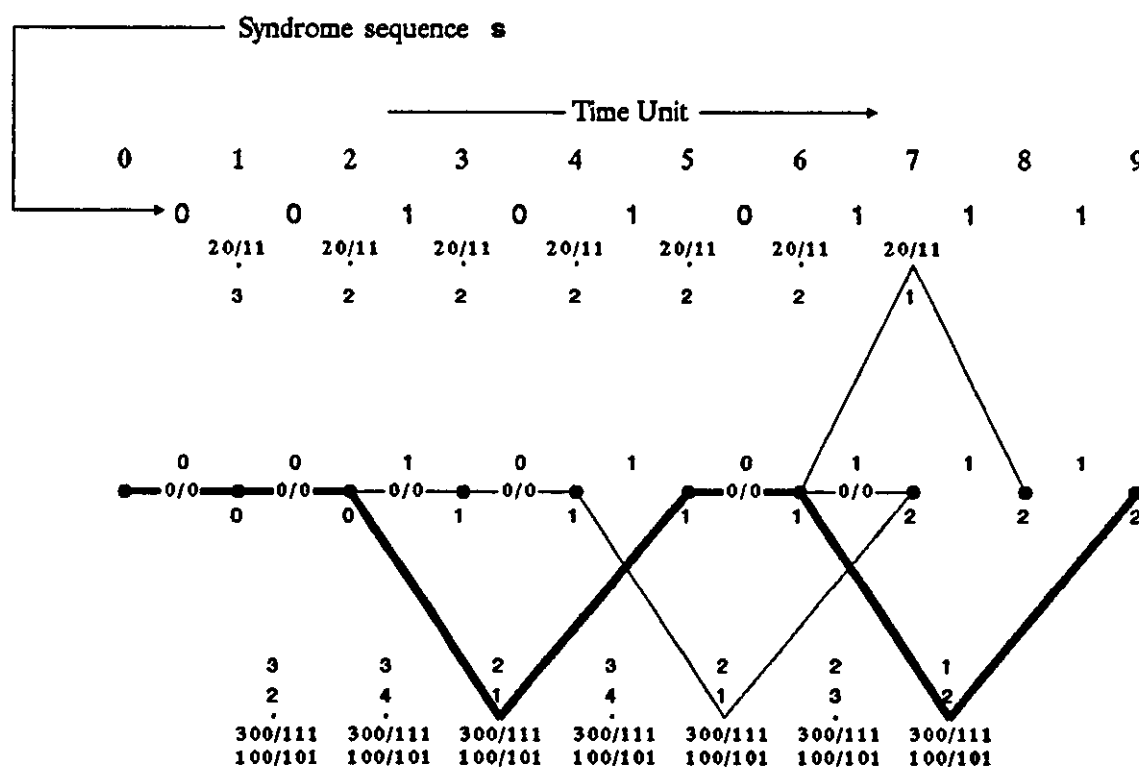


Figure 4.8: Decoding using the simplified trellis diagram of Fig. 4.5.

error sequences of Example A4.6.1 are used, i.e. $u=(000\ 111\ 111\ 000\ 000\ 111\ 111)$ and $e=(0000\ 0000\ 0010\ 0000\ 0000\ 0000\ 0000)$

1000 0000 0000). *v*, *r* and *s* have been calculated in Example A4.6.1. As can be seen in Fig. 4.8, the same *z* is obtained.

Further to the discussion earlier on, the path entering *S*₀ for each time-unit, is considered (see Fig. 4.9 - 3 more time-units have been added, corresponding to no channel errors). Obviously, the final survivor is the path marked ABDEHI, while the dead paths are BC, EF & EG.

The memory-size problem arises because the decoder, at any time-unit, must store the main path so far, as well as some of the dead ones. The dead paths should be stored because some of these may be activated later to form part of the main path; hence they are dead, given the information available at a certain time-unit.

TABLE 4.4

<u>Time-unit</u>	<u>Main Path</u>	<u>Dead Paths</u>			
2	00/0				
3	000/1				
4	0000/1				
5	00100/1	2/00/1			
6	001000/1	2/00/1			
7	0010000/2 0000100/2	Tie			
8	00100020/2	2/00100/2	6/0/1		
9	001000300/2	2/00100/2	6/0/1	6/20/1	
10	0010003000/2	2/00100/2	6/0/1	6/20/1	
11	00100030000/2	6/20/1			
12	001000300000/2				

NOTE: The 2nd column contains: (main path)/(metric). The 3rd column contains: (originating time-unit)/(dead path)/(metric). For the main path & dead branches: 0 $\hat{=}$ [000], 1 $\hat{=}$ [001], 2 $\hat{=}$ [010] & 3 $\hat{=}$ [100].

One observation from the 3rd column of TABLE 4.4 is the randomly varying amount of storage space required (when channel is noisy). Note, though, that if a dead branch terminates *m*+1 time-units before, it cannot be reactivated because the longest transition is *m*+1 time-units (see Theorem

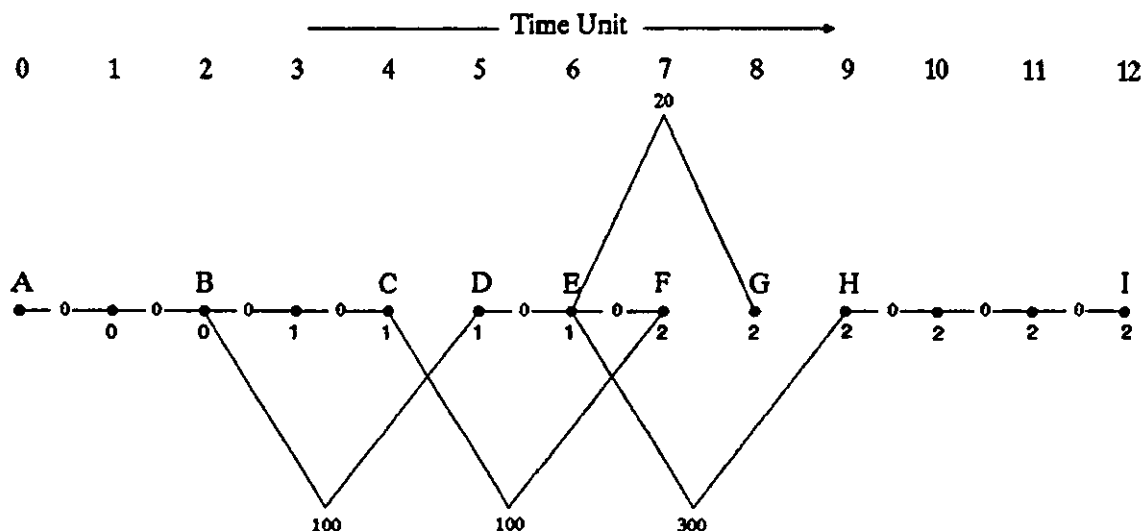


Figure 4.9: The survivor path at each time-unit, for the decoding example of Fig. 4.8.

4.28). Note, also, that at node F there is a tie, which may be resolved (if needed), by tossing a fair coin.

Hence the buffer for the dead paths need not be large. But the question that must be answered is: "Is there an upper bound in the number and length of the dead paths, and if yes what is it?"

4.7.2. Normal LSC with $t=2$

Consider an (n, k, m) normal LSC with total memory M and weight-constraint 2. According to Theorem 4.21, its constrained trellis will have $C(M, \varphi)$ states of weight φ , where $0 \leq \varphi \leq 2$. In particular, for each $\tau \in [\text{MAX}\{0, \varphi + k - M - f\}, \text{MIN}\{k - f, \varphi\}]$ there will be $C(k - f, \tau)C(M - k + f, \varphi - \tau)$ states of weight φ .

From each particular state of weight φ and for each τ , as defined above, $C(k - f, \tilde{n} + \tau - \varphi)$ other states of weight \tilde{n} can be reached, where $\tilde{n} \in [\varphi - \tau, \text{MIN}\{2 - \tau, k - f + \varphi - \tau\}]$. The $1 + M(M + 1)/2$ states are grouped as following:

There is one state of weight $\varphi = 0$, S_0 . Since $\tau = 0$, the weight, \tilde{n} , of the next state satisfies the inequality: $0 \leq \tilde{n} \leq \text{MIN}\{2, k - f\}$. There is one transition from S_0 to S_0 . There are $k - f$ transitions from S_0 to the weight-1 region. If $k - f \geq 2$

there are $C(k-f, 2)$ transitions to the weight-2 region.

There are $C(M, 1) = M$ states of weight $\varphi = 1$. τ , the weight of $R(h)$, satisfies $\text{MAX}\{0, 1+k-M-f\} \leq \tau \leq \text{MIN}\{k-f, 1\}$. Since, $k-f \geq 1$ & $M \geq k-f$, $1+k-M-f \leq 1$, then $0 \leq \tau \leq 1$; if, though, $M = k-f$, then $\tau = 0$.

There are $M-k+f$ states of weight $\varphi = 1$, with $\tau = 0$. The weight, \tilde{n} , of the next state satisfies $1 \leq \tilde{n} \leq \text{MIN}\{2, k-f+1\} = 2$, since $k-f \geq 1$. There is one transition to the weight-1 region and $k-f$ transitions to the weight-2 region.

There are $k-f$ states of weight $\varphi = 1$, with $\tau = 1$. The weight, \tilde{n} , of the next state satisfies $0 \leq \tilde{n} \leq \text{MIN}\{1, k-f\} = 1$ since $k-f \geq 1$. There is one transition to S_0 and $k-f$ transitions to the weight-1 region.

There are $C(M, 2)$ states of weight $\varphi = 2$. The restriction on τ becomes: $\text{MAX}\{0, 2+k-M-f\} \leq \tau \leq \text{MIN}\{k-f, 2\}$.

There are $C(M-k+f, 2)$ states of weight $\varphi = 2$ with $\tau = 0$, provided that $M-k+f \geq 2$. The weight, \tilde{n} , of the next state is restricted to $\tilde{n} = 2$. There is one transition to the weight-2 region.

There are $(k-f)(M-k+f)$ states of weight $\varphi = 2$ with $\tau = 1$, provided that $M-k+f \geq 1$. The weight, \tilde{n} , of the next state is restricted to $\tilde{n} = 1$. There is one transition to the weight-2 region.

There are $C(k-f, 2)$ states of weight $\varphi = 2$ with $\tau = 2$, provided that $k-f \geq 2$. The next state is S_0 .

Finally, each of the examined transitions has label multiplicity σ , where, if \tilde{n} is the weight of the next state and τ the weight of $R(h)$:

$$\sigma \triangleq 1 + C(f, 1) + C(f, 2) + \dots + C(f, a) \quad / a \triangleq \text{MIN}\{f, 2 - \tilde{n} - \tau\} \quad (4.62)$$

The following theorem has been proved (for an example see Appendix 4.10, p. 369).

Theorem 4.29: Consider an (n, k, m) normal LSC with total memory M and weight-constraint 2. If σ denotes the label multiplicity per transition, then in the central portion of its constrained trellis:

There is one state of weight 0, S_0 , with one transition to itself [$\sigma = 1 + C(f, 1) + C(f, 2)$], $k-f$ transitions to the

weight-1 region $[\sigma=1+\min\{1,f\}]$ and, if $k-f \geq 2$, $(k-f)(k-f-1)/2$ transitions to the weight-2 region $[\sigma=1]$.

There are M states of weight 1. $M-k+f$ of them have one transition, each, to a weight-1 state $[\sigma=1+\min\{1,f\}]$ and $k-f$ transitions, each, to the weight-2 region $[\sigma=1]$. The rest, $k-f$, states have one transition, each, to S_0 $[\sigma=1+\min\{1,f\}]$ and $k-f$ transitions, each, to the weight-1 region $[\sigma=1]$.

There are $M(M-1)/2$ states of weight 2, with one transition each $[\sigma=1]$. $(M-k+f)(M-k+f-1)/2$ states transit to another weight-2 state, provided that $M-2 \geq k-f$. $(k-f)(M-k+f)$ states transit to a weight-1 state, provided that $M-1 \geq k-f$. The rest, $(k-f)(k-f-1)/2$, states transit to S_0 , provided that $k-f \geq 2$.

Following the directions of Note 4.7, about the construction of the simplified from the constrained trellis, one readily concludes that the simplified trellis will have $M+1$ states only, which is an improvement of $[(M-1)/M] \times 100\%$ for large M , say $M > 5$. Note that the % improvement approaches 100% as M increases (for $M=10$, it is 90%). The reduced number of states means, of course, a reduced number of paths.

4.7.3. Equal-Length Shift Registers

The following theorem deals with the above-mentioned special case. It is based on Theorems 4.21 & 4.27.

Theorem 4.30: Consider an (n,k,m) normal LSC with weight-constraint t and shift registers of equal length.

With respect to the central portion of its constrained trellis:

i) There are $\binom{mk}{\varphi}$ states of weight φ , where:

$$0 \leq w[S(h)] \triangleq \varphi \leq t$$

ii) For each $\tau \in [\max\{0, \varphi - (m-1)k\}, \min\{k, \varphi\}]$

there are $\binom{k}{\tau} \binom{(m-1)k}{\varphi - \tau}$ states of weight φ ,

each of which has $\binom{k}{\tilde{n} + \tau - \varphi}$ single-edge

transitions, to the weight- \bar{n} region, where:

$$\varphi - \tau \leq w[S(h+1)] \hat{=} \bar{n} \leq \text{MIN}\{t-\tau, k+\varphi-\tau\}$$

With respect to the central portion of the simplified trellis:

- iii) There are $\binom{k}{t}$ maximum-length transitions ($\beta = m+1$), all starting from state S_0 .
- iv) There are $\binom{k(m-\beta+1)}{\varphi} - \binom{k(m-\beta)}{\varphi}$ states of weight φ , with $\binom{k}{t-\varphi}$ transitions of length β each, where $\beta \in [2, m]$.
- v) There are $\binom{k}{\varphi}$ states of weight φ , where $\varphi > 0$, with $\binom{k}{t-\varphi}$ transitions of length m , each.

Proof: See Appendix 4.11 (§ A4.11.1., p. 374). ■

In order for one to verify (part of) the above results, qualitatively, one has to remember that a transition of length β , starting at time-unit h , implies that the circuit memory contains t 'ones' at time-units $h+1$, $h+2$, $h+\beta$ (see § 4.4.3.); otherwise, the memory will arrive at a state (of the constrained transition diagram) of weight $< t$, hence with more than one transitions out of it.

From part (v), above, there are no transitions of length m , from S_0 . This is expected, since if the current state is S_0 , any 'ones' injected into it will remain in the circuit memory for m time-units, since all SRs have length m (causing thus a transition of length $m+1$). Furthermore, for a transition of length m to occur, at time-unit h there must be φ 'ones' in the 1st stage of as many SRs, hence there are $C(k, \varphi)$ starting states of weight φ . For each one of them, the I/P block must contain $t-\varphi$ 'ones', hence there are $C(k, t-\varphi)$ such blocks and because all transitions are single-edge ($f=0$), there are as many transitions out of each state.

According to part (iii), above, transitions of length $m+1$ may start, only from S_0 . This is so because for such a long

transition to occur, and since all SRs have length m , at time-unit h there must be no 'ones' in the memory (hence one must start from S_0), while at time-unit $h+1$, there must be t 'ones' in the 1st stage of t different SRs. Furthermore, there are $C(k,t)$ ways to place these t 'ones' in the 1st stage of k SRs.

4.7.4. $(n,1,m)$ Normal LSCs

The following theorem deals with the above-mentioned special case. It is based on Theorem 4.30.

Theorem 4.31: Consider an $(n,1,m)$ normal LSC with weight-constraint t .

With respect to the central portion of its constrained trellis:

- i) There are $\binom{m}{\varphi}$ states of weight φ , where:
 $0 \leq w[S(h)] \triangleq \varphi \leq t$. Transitions are single-edge.
- ii) From S_0 , there is one transition to S_0 and one to S_1 . There are $\binom{m-1}{\varphi}$ states of weight $\varphi \in [1,t)$, each with two transitions, to states of weight φ & $\varphi+1$ and $\binom{m-1}{t}$ states of weight t , each with one transition to a state of weight t . Also, there are $\binom{m-1}{\varphi-1}$ states of weight $\varphi \in [1,t)$, each with two transitions to states of weight $\varphi-1$ & φ and $\binom{m-1}{t-1}$ states of weight t , each with one transition to a state of weight $t-1$. If $t \geq m$, there is one state of weight m with one transition to a state of weight $m-1$ if $t=m$, and two transitions to states of weight $m-1$ & m if $t > m$.

With respect to the central portion of the simplified trellis:

- iii) There is one maximum-length transition ($\beta = m+1$), if $t=1$, starting from state S_0 , and none if $t > 1$.

- iv) There are $\binom{m-\beta+1}{\zeta} - \binom{m-\beta}{\zeta}$ states of weight $\zeta \in [t-1, t]$, with one transition of length β each, where $\beta \in [2, m+1-\zeta]$. There are no such transitions from states of weight less than $t-1$.
- v) The longest transition has length $m+2-t$. There is only one such transition, and starts from state S_a , where $a=2^{t-1}-1$, with an input of 1.

Proof: See Appendix 4.11 (§ A4.11.2., p. 375). ■

An $(n, 1, m)$ normal LSC is made of one SR of length m . Since the I/P block is made of one bit, there are, at most, two (single-edge) transitions from each state.

If the current state is S_0 , the next one may be $[00 \dots 0] = S_0$, or $[00 \dots 01] = S_1$. Remember that the bits of a state are arranged starting from the end [see (3.2)].

If the current state has weight $\zeta \in [1, t)$ and the last bit of the SR is 0, there are $C(m-1, \zeta)$ ways to arrange the ζ 1s in the rest of the stages, and hence as many states of weight ζ [note that $C(m-1, \zeta) = 0$, if $\zeta \geq m$]. Then, during the transition, the SR will not loose any 1s, hence the next state will have the same weight (if the I/P is 0), or weight $\zeta+1$ (if the I/P is 1). If though the state has weight t , the I/P must be 0.

If the current state has weight $\zeta \in [1, t)$ and the last bit of the SR is 1, there are $C(m-1, \zeta-1)$ ways to arrange the rest of the 1s in the rest of the stages, and hence as many states of weight ζ . During the transition, the SR will loose one 1, hence the next state will have the same weight (if the I/P is 1), or weight $\zeta-1$ (if the I/P is 0). If though the state has weight t , the I/P must be 0.

If $t > 1$, there is no maximum-length transition (see the proof of Theorem 4.18). For a transition of length m to occur, one must start with one 1 in the first stage, hence from state S_1 , and inject a 0 if $t=1$, or a 1 if $t=2$, but if $t > 2$, the transition will have length $< m$.

A one-SR memory can be brought into a weight- t state*, only if, at most, one 1 is missing.

* And, hence, start a long transition.

4.8 CONCLUSIONS

In this chapter, the theory of the *constrained state-transition diagram*, for binary normal LSCs*, was developed. Such a diagram may be obtained from the ordinary one by imposing a limit on the sum of the Hamming weights of the current state and the current input.

The idea is due to Reed & Truong [24], who used it to develop *error-trellis syndrome decoding* (its analysis may be found in Sec. 4.1). The advantage of using a constrained trellis lies with its reduced complexity (which may be a fraction of the ordinary trellis complexity) and as a consequence with the opportunity to use 'longer' codes. If a constrained trellis was to be used at all, for decoding, it seems rather obvious that it could not be the encoder-trellis, because all channel sequences, v , are equally probable (normally). On the other hand, the error sequence e tends to have small Hamming-weight (in fact its probability decreases exponentially with $w[e]$), hence a trellis operating with e is required. e is, of course, unknown but the syndrome sequence, s , is a function of e ($s = eH^T$).

For binary systematic codes it was proved that $u = r^{(n)} + z$, where $e = [z, zP + s]$. Clearly, z is the message-bit error sequence, and it is the quantity to be estimated. The estimation criterion is obviously $w[e]$, which must be minimized. Because high-weight e s are less likely, they are not considered. The weight-constraint is t , over an actual constraint-length. The decoder's task is to find z , so that the distance between $[z, zP]$ & $[0, s]$ is minimized. The trellis arises because an LSC is assumed to exist (the 'regulator circuit') driven by z and responding with $[z, zP]$. This is clearly a replica of the encoder. The decoding algorithm (Sec. 4.3) is very similar to the Viterbi one. The work, in Sec. 4.1, is a more complete and formal repetition of that by Reed & Truong [24]. It resulted in a deeper understanding of the theory of convolutional codes and syndrome decoding.

The rest of the work is original and is concerned with the complexity of the constrained trellis. A binary normal

* See Definition 4.2.

LSC is assumed with a total memory M and a general weight-constraint, t , on the sum of the weights of the current state & I/P block. The work in Sec. 4.5 evolves to Theorem 4.21 which, given the weight of the current state, provides expressions about the number of transitions to states of a given weight, the number of labels of a particular transition, etc. The theorem also provides the conditions for the existence of transitions, given certain information about the current state, etc.

The concept of the *simplified trellis* was introduced in Sec. 4.4; this is obtained from the constrained one, by removing all states with one transition, introducing, thus, *long transitions*. The aim of the rest of the work is to obtain formulae about the number and length of the long transitions. To this end, the following concepts were introduced: $f(i)$ = No of SRs of length i ; $F(i)$ = No of SRs of length $\leq i$; $\text{REG}(i,)$ = set of the last i stages of each SR; $\text{DIG}(i,)$ = set of I/P block positions corresponding to SRs of length $\leq i$.^{*} The next step was to obtain relations between the number of elements of the various sets and the memory functions f & F (Theorem 4.26). Finally, Theorem 4.27 provided expressions (in terms of F , k , M , t , β & w) about the number of transitions of length β from states of weight w , the number of such states, the maximum-length transitions and associated existence conditions, etc.

The last section (4.7) examined special cases ($t=1$, $t=2$, $M_1=m$ & $k=1$). For $t=1$ the simplified trellis has only one state. The decoding algorithm was modified and it was shown (via some examples) that the simplified trellis offers gains in decoding complexity. Some complications were also discussed, like the need to store 'dead paths' for a brief period of time (see Example 4.4). The rest of the cases were restricted to a restatement of Theorems 4.21 & 4.27. The formal mathematical language of these theorems was followed closely for the special cases and for some examples. The 'predictions' obtained were contrasted with qualitative explanations and corresponding state-transition diagrams. The main result here was the verification of the 'prediction' power of these theorems.

^{*} See Definitions 4.5, 4.8, 4.6 & 4.7, respectively.

CHAPTER 5

Threshold Decoding

Chapter 5 is an introduction to the basic terminology and theorems of threshold decoding. This technique, discovered by Massey [18], offers net coding-gains of 1 to 3 dBs, with relatively simple implementations and hence at very high data rates [13].

Unless otherwise stated, the communications channel between the encoder and the decoder is assumed to suffer from additive noise, which is statistically independent from digit to digit.

5.1 THE THRESHOLD-DECODING PROBLEM

In this section the basic decoding problem will be introduced. The various digits involved in the discussion will be, initially, denoted using a single subscript, like for example e_j , for the various error digits. The proper notation for the error digits is of course $e_h^{(i)}$ / $i=1,2,\dots,n$ & h = time-unit, but this notation would unnecessarily complicate the discussion, at this stage. This section is mainly based on the work by Massey [18].

Definition 5.1: Let the error digits e_j / $j=1,2,\dots,N$ take values from $GF(q)$. Consider J linear combinations of the e_j s:

$$E_i \hat{=} \sum_{j=1}^N a_{ij} e_j \quad / i=1,2,\dots,J \quad \& \quad a_{ij} \in GF(q) \quad (5.1)$$

E_i is called the *ith composite parity-check*. A composite parity check E_i is said to *check an error digit* e_m iff $a_{im} \neq 0$, i.e. iff e_m participates in the formation of E_i . The set of all E_i s is denoted by $\{E_i\}$.

Definition 5.2: A set of composite parity-checks $\{E_i\}$ is said to be *orthogonal* on an error digit e_m iff e_m is checked once by each member of the set, but no other error digit is checked by more than one member of the set:

$$\left. \begin{array}{l} a_{im}=1 \text{ for all } i \in [1, J] \\ a_{ij}=0 \text{ for all } i \in [1, J] \\ \text{except at most one } i \end{array} \right\} \rightarrow \text{For } j=1, 2, \dots, n / j \neq m \quad (5.2)$$

Note that, although $a_{ij} \in GF(q)$, most of the a_{ij} s are either 0 or 1.

It follows from Definition 5.2 that if $\{E_i\}$ is orthogonal on e_m then this digit affects all composite parity-checks, while any other error digit affects at most one. Then, the following formulation can be proposed:

Decoding Problem: Given a set of J composite parity-checks, orthogonal on e_m , determine e_m so that a certain 'criterion of goodness' is satisfied.

Massey [18] proposed two algorithms [for the non-binary case of $GF(q)$], that solve the above problem; the majority-decoding algorithm and the a posteriori probability (APP) decoding one.

5.2 THE DECODING ALGORITHMS

This section is made of four theorems; they cover majority & APP decoding for the nonbinary and the binary case. Again, the concern is on how to decode an abstract set of error digits, so the latter remain unstructured. The results

follow closely the work by Massey [18].

5.2.1. The Non-Binary Case

Theorem 5.1: Consider a set of J composite parity-checks E_1, E_2, \dots, E_J , orthogonal on e_m , and the set $\{e_j\}$ of error digits that are checked by $\{E_i\}$. Assume that no more than $\lfloor J/2 \rfloor$ of the e_j s are non-zero. Then, according to the *majority-decoding* algorithm, e_m is given correctly as that value of $GF(q)$ which is assumed by the majority of the composite parity-checks. If there is a tie that involves 0, let $e_m = 0$. $\lfloor x \rfloor$ denotes the greatest integer $\leq x$.

Proof: See Appendix 5.1 (§ A5.1.1., p. 378). ■

Definition 5.3: The *a posteriori probability* (APP) decoding algorithm assigns to e_m that value of $V \in GF(q)$ for which the conditional probability $P(e_m = V | \{E_i\})$ is maximum, where $\{E_i\}$ is orthogonal on e_m . ■

The APP decoding algorithm "...makes the best possible use of the information contained in a set of J parity-checks orthogonal on e_m in arriving at a decision on the value of e_m " [18].

Theorem 5.2: Consider a set of J composite parity-checks orthogonal on e_m . For APP decoding, let $e_m = V$, where $V \in GF(q)$ is such that (5.3), below, is maximized.

$$\log P(e_m = V) + \sum_{i=1}^J \log P(E_i | e_m = V) \quad (5.3)$$

Proof: See Appendix 5.1 (§ A5.1.2., p. 378). ■

5.2.2. The Binary Case

The two theorems that follow are special cases of Theorems 5.1 & 5.2.

Theorem 5.3: Consider a set of J composite parity-checks E_1, E_2, \dots, E_J , orthogonal on e_m , and the set $\{e_j\}$ of error bits that are checked by $\{E_i\}$. Assume that no more than $\lfloor J/2 \rfloor$ of the e_j s are one. Then, according to the *majority-decoding* algorithm for the binary symmetric channel, $e_m = 1$ iff

$$\Sigma \triangleq \sum_{i=1}^J E_i > \lceil J/2 \rceil \quad (5.4)$$

where the summation, above, denotes real-number addition and $\lceil x \rceil$ denotes the smallest integer $\geq x$.

Proof: See Appendix 5.1 (§ A5.1.3., p. 379). ■

Theorem 5.4: The APP decoding rule for the binary symmetric channel is: Choose $e_m = 1$ iff

$$\sum_{i=1}^J E_i \left[2 \log(q_i/p_i) \right] > \sum_{i=0}^J \log(q_i/p_i) \quad (5.5)$$

where, $p_0 = 1 - q_0 \triangleq P(e_m=1)$ and $p_i = 1 - q_i$ is the probability of an odd number of 'ones' in the error bits, exclusive of e_m , that are checked by the i th parity-check E_i .

Proof: See Appendix 5.1 (§ A5.1.4., p. 380). ■

5.3 GENERAL ASPECTS OF CODES FOR THRESHOLD DECODING

From Theorem 2.13 (p. 47), if e is the error sequence of an additive-noise channel, then the syndrome sequence is given by:

$$s = eH^T \quad (5.6)$$

It is obvious from the above that the syndrome bits* are linear combinations of the channel-error bits. If H is such that orthogonal check sums can be formed, for each of the error bits, then the code can be majority-logic decoded.

* Unless otherwise stated, only binary codes will be considered.

5.3.1. Introduction - Terminology

Consider the result of Theorem 2.15 (p. 50):

$$s_h^{(j)} = e_h^{(k+j)} + \sum_{z=0}^{\theta} \sum_{i=1}^k e_{h-z}^{(i)} g_{k+j,z}^{(i)} \quad /1 \leq j \leq n-k \text{ \& } h \geq 0 \quad (5.7)$$

where $\theta \triangleq \text{MIN}\{h, m\}$.

Consider all those syndrome bits that may check on e_0 :

$$s_0^{(j)} = e_0^{(k+j)} + \sum_{i=1}^k e_0^{(i)} g_{k+j,0}^{(i)} \quad /1 \leq j \leq n-k \quad (5.8a)$$

$$s_1^{(j)} = e_1^{(k+j)} + \sum_{z=0}^1 \sum_{i=1}^k e_{1-z}^{(i)} g_{k+j,z}^{(i)} \quad /1 \leq j \leq n-k \quad (5.8b)$$

.....
.....

$$s_m^{(j)} = e_m^{(k+j)} + \sum_{z=0}^m \sum_{i=1}^k e_{m-z}^{(i)} g_{k+j,z}^{(i)} \quad /1 \leq j \leq n-k \quad (5.8c)$$

$$s_{m+1}^{(j)} = e_{m+1}^{(k+j)} + \sum_{z=0}^m \sum_{i=1}^k e_{m+1-z}^{(i)} g_{k+j,z}^{(i)} \quad /1 \leq j \leq n-k \quad (5.8d)$$

Note from eqn (5.8a) that the 0th syndrome block, s_0 , checks on the 0th error block, e_0 ; note also, from eqn (5.8b), that the 1st syndrome block, s_1 , checks on the first two error blocks, e_0 & e_1 . Note finally [from eqns (5.8c) & (5.8d)] that the m th syndrome block, s_m , checks on e_0, e_1, \dots, e_m , while the $(m+1)$ th syndrome block does not check on e_0 . So the $(n-k)(m+1)$ syndrome bits of the 1st constraint-length, $n_A \triangleq (m+1)n$, check on the 0th error block, e_0 .

Definition 5.4: An (n, k, m) convolutional code is called *self-orthogonal* iff the set of J_1 syndrome bits which check $e_0^{(i)}$ are orthogonal on $e_0^{(i)}$ for $i=1, 2, \dots, k$. This code is capable of correcting any error sequence with $\lfloor J/2 \rfloor$ or fewer errors in a span of n_A consecutive positions, where $J \triangleq \text{MIN}\{J_1, J_2, \dots, J_k\}$.

Definition 5.5: Consider a set $\{E_i\}$ of J composite parity-checks, orthogonal on e_n . The number of error digits, exclusive of e_n , that are checked by E_i is called the size of E_i and is denoted by c_i . The total number of distinct error digits checked by $\{E_i\}$ is called the effective constraint-length for the decoding of e_n and is denoted by $n_e(e_n)$. It follows from the definition that,

$$n_e = 1 + \sum_{i=1}^J c_i \quad (5.9)$$

Definition 5.6: An (n,k,m) convolutional code is called J -orthogonalizable if it is possible to form J or more parity-check sums, orthogonal on $e_0^{(i)}$ for $i=1,2,\dots,k$ by taking linear combinations of the first $(m+1)(n-k)$ syndrome digits. If d_{\min} is the minimum distance of this code and $J=d_{\min}-1$, then this code is called *completely orthogonalizable*. [12]

5.3.2. Decoding Modes

Consider again eqns (5.8). Clearly, by the time s_1 is formed the decoder has already an estimate of the 0th error block (\hat{e}_0). If this estimate is fed back into the syndrome register, in an appropriate way, the effect of these past error bits will be removed. Obviously, this feedback operation can become permanent so that every error block that is estimated is fed back into the syndrome register to cancel its effect on the syndrome.

Definition 5.7: Under the *feedback decoding* (FD) mode a convolutional decoder will use the estimate of the currently decoded error block, $e_h = (e_h^{(1)} e_h^{(2)} \dots e_h^{(n)})$, to remove its effects from all those syndrome bits that check on these error bits. The altered syndrome bits are then used normally, for the estimation of the subsequent error blocks. The mode of operation of a convolutional decoder where no feedback takes place, is called *definite decoding* (DD).

5.3.3. Definite Decoding - Parity Squares

The following theorem has been discussed and proved in Appendix 5.2 (p. 381):

Theorem 5.5: Consider an (n,k,m) systematic convolutional code with generator sequences $g_{k+j}^{(i)}$ / $i=1,2,\dots,k$ & $j=1,2,\dots,n-k$. Then, under definite decoding, for $a \geq 0$:

$$[S]_a^{a+m} = [E^m]_{a-m}^{a+m} [H(\Gamma)]^T + [EP]_a^{a+m} \quad (5.10)$$

where the syndrome & parity error vectors extend from block a to block $a+m$, while the message error vector from block $a-m$ to $a+m$ (for their precise definition, see Appendix 5.2). If $a < m$, the message error vector is suitably truncated. If $-m \leq \beta \leq m$, $1 \leq \mu \leq k$, $0 \leq \tau \leq m$ & $1 \leq \sigma \leq n-k$, then:

$$s_{a+\tau}^{(\sigma)} \text{ checks on } e_{a-\beta}^{(\mu)} \text{ iff } g_{k+\sigma, \tau+\beta}^{(\mu)} = 1 \quad (5.11)$$

The syndrome bits checking on $e_{a-\beta}^{(\mu)}$ correspond to 1s along the $[(\mu-1)(2m+1)+m+1-\beta]$ th column of $H(\Gamma)$. The message error bits, checked by syndrome bit $s_{a+\tau}^{(\sigma)}$, correspond to 1s along the $[(\sigma-1)(m+1)+\tau+1]$ th row, of $H(\Gamma)$.

Furthermore, if J_i / $1 \leq i \leq k$ denotes the number of syndromes checking on error bit $e_h^{(i)}$ / $h \geq m$ and c_j / $1 \leq j \leq n-k$ denotes the size of syndrome bit $s_h^{(j)}$ / $h \geq m$, then:

$$J_i = \sum_{j=1}^{n-k} w[g_{k+j}^{(i)}] \quad / 1 \leq i \leq k \quad (5.12)$$

$$c_j = 1 + \sum_{i=1}^k w[g_{k+j}^{(i)}] \quad / 1 \leq j \leq n-k \quad (5.13)$$

In Example A5.2.1 (p. 387), the above theorem is tested against a $(2,1,6)$ systematic code.

Note, from (5.13), that under DD the various syndrome bits do not have the same size. The analysis is valid for any block h , beyond the first constraint-length ($h \geq m$). The results for the first constraint-length are similar to the FD case (to be examined below).

$$[S]_0^m = [E^m]_0^m [H(\lambda)]^T + [E^p]_0^m \quad (5.15)$$

where the syndrome & the two error vectors extend from block 0 to block m and $H(\lambda)$ is the *parity-triangle matrix* (for their precise definition, see Appendix 5.3).

If $0 \leq a \leq m$, $1 \leq \mu \leq k$, $0 \leq \tau \leq m$ & $1 \leq \sigma \leq n-k$, then:

$$s_\tau^{(\sigma)} \text{ checks on } e_a^{(\mu)} \text{ iff } g_{k+\sigma, \tau-a}^{(\mu)} = 1 \quad (5.16)$$

The syndrome bits checking on $e_a^{(\mu)}$ correspond to 1s along the $[(\mu-1)(m+1)+a+1]$ th column of $H(\lambda)$. The message error bits checked by syndrome bit $s_\tau^{(\sigma)}$ correspond to 1s along the $[(\sigma-1)(m+1)+\tau+1]$ th row, of $H(\lambda)$.

Furthermore, if $J_i / 1 \leq i \leq k$ denotes the number of syndromes checking on error bit $e_0^{(i)}$ and $c_{j,h} / 1 \leq j \leq n-k$ denotes the size of syndrome bit $s_h^{(j)} / h \geq 0$, then:

$$J_i = \sum_{j=1}^{n-k} w[g_{k+j}^{(i)}] \quad / 1 \leq i \leq k \quad (5.17)$$

$$c_{j,h} = 1 + \sum_{i=1}^k \sum_{z=0}^h g_{k+j,z}^{(i)} \quad / 1 \leq j \leq n-k \quad (5.18)$$

Proof: See Appendix 5.3 (p. 389). ■

In Examples A5.3.1-2 (pp. 394-7), Theorem 5.6 is tested against a (2,1,6) and a (3,2,13) systematic code.

Parity triangles are very useful tools for the analysis & synthesis of convolutional self-orthogonal codes (CSOCs). Nevertheless, they will not be useful for the further development of this thesis. The results obtained, in Appendices 5.2 & 5.3, are original, at least in their generality. This includes the matrix eqns involving the syndrome & error bits of one constraint-length.

Two interesting questions arise now that the structure of the parity-check matrix has been linked to the formation of orthogonal parity checks. First, what conditions should be imposed on the generator sequences so that the syndromes that check on an error bit are orthogonal on that bit? Second, how can one construct codes that are self-orthogonal?

5.4 DISTANCE PROPERTIES OF CODES FOR THRESHOLD DECODING

Consider a binary (n,k,m) convolutional code. According to Definition A2.5.3 (p. 310),

$$d_{\min} \triangleq \text{MIN}\{d([v']_m, [v'']_m) : [u']_0 \neq [u'']_0\} \quad (5.19)$$

Note that in computing d_{\min} only the 1st constraint-length is considered and only codewords that differ in their first source block are compared [2]. Hence, d_{\min} is a useful estimate of the code's power when decoding is based on only one constraint-length; this is the case with threshold decoding, but not with Viterbi, error-trellis or sequential decoding.

The following theorems relate d_{\min} with J .

Theorem 5.7: For an (n,k,m) convolutional code, with minimum distance d_{\min} , the 1st source block $[u]_0$ can be correctly decoded provided $\lfloor (d_{\min}-1)/2 \rfloor$ or fewer errors have occurred in the first constraint-length $[r]_m$ of the received sequence. Conversely, there are some received sequences, containing $\lfloor (d_{\min}-1)/2 \rfloor + 1$ errors in the first constraint-length, which will result in $[u]_0$ being incorrectly decoded.

Proof: See Appendix 5.4 (§ A5.4.2., p. 398). ■

Definition 5.8: The *maximum error-correcting capability* of a code, when the decoding of the 1st source block $[u]_0$ is based on the 1st constraint-length, is denoted by t and is defined as:

$$t \triangleq \lfloor (d_{\min}-1)/2 \rfloor \quad (5.20)$$

A feedback decoder which achieves the maximum error correcting capability t is called an *optimum feedback decoder*. ■

Theorem 5.8: Consider an (n,k,m) convolutional code. If at least J orthogonal parity checks can be formed for each of $e_0^{(i)} / i=1,2,\dots,k$, then $J \leq d_{\min} - 1$.

Proof: See Appendix 5.4 (§ A5.4.3., p. 400). ■

5.5 CONVOLUTIONAL SELF-ORTHOGONAL CODES (CSOCs)

In this section the very important class of CSOCs will be briefly examined. The previously derived results will be simplified for the case of CSOCs.

5.5.1. Definite Decoding of CSOCs

Theorem 5.9: The syndrome bits checking on $e_h^{(1)}$ are orthogonal on $e_h^{(1)}$ if, and only if, the code is a CSOC.[19]

Proof: See Appendix 5.5 (p. 400).

The implication of the above theorem is that a CSOC is capable of correcting up to $\lfloor J/2 \rfloor$ errors in both the DD & the FD mode. The difference between the two modes arises when one considers the total number of error bits which are 'allowed' to contain $\lfloor J/2 \rfloor$ or fewer errors. From Definition 5.5, it is obvious that a CSOC can correct up to $\lfloor J/2 \rfloor$ or fewer errors in a set of n_e error bits; nevertheless, n_e for a CSOC in the FD mode is less than n_e for a CSOC in the DD mode. Hence, the error correcting capability of the CSOC is weakened, by DD; on the other hand, a DD does not suffer from error propagation. Nevertheless, "...an analysis ... comparing the effect of error propagation with feedback to the reduced error correcting capability without feedback ... concludes that feedback decoders will usually outperform definite decoders." [2].

5.5.2. Structure of Parity Triangles for a CSOC

As mentioned earlier, parity triangles are useful, in general, for the study of orthogonal codes. In particular, for CSOCs they can provide an easily digested, illustration of the orthogonality conditions and how these can be fulfilled. A brief discussion and some examples, are included in Appendix 5.6 (p. 402).

5.5.3. Effective Constraint-Length of a CSOC

Note that the effective constraint-length depends on the specific error bit considered. Hence, what is required is a definition for the code, i.e. one that takes into account all error bits of a block. The adopted definition considers the code parameter to be the maximum among the bit parameters. It is felt, though, that this term is not very effective in describing how many bits are (in effect) involved in the decoding. Where the actual constraint-length includes all bits that are involved in the decoding of a block (and also of a bit, of that block) the effective constraint-length considers the exact maximum number of bits that are involved in the decoding of any one bit of a block. Maybe a more 'fair' parameter is a cumulative one, that considers the exact number of error bits that participate in the decoding of a block.

Definition 5.9: Consider an (n,k,m) systematic CSOC. Let J_1 be the number of syndromes checking on $e_0^{(1)}$ $/i=1,2,\dots,k$. The maximum of J_1, J_2, \dots, J_k is called the *effective constraint-length of the code* and is denoted by n_E . Furthermore, the total number of distinct error bits checked by the $J_1+J_2+\dots+J_k$ syndrome bits will be called the *block effective constraint-length of the code* and will be denoted by N_E . ■

For instructions on how to calculate N_E and an example, see Appendix 5.7 (p. 405).

5.5.4. Error Propagation in CSOCs

As mentioned earlier, feedback decoding (FD) suffers from error propagation. This occurs when an erroneous decision is made by the threshold decoder; because of the feedback mechanism, a number of syndrome bits (specifically those that check on the error bit that was erroneously decoded) are fed with the wrong information.

Consider an erroneous decoding decision on error bit $e_a^{(\mu)}$ where $\mu \in [1,k]$ & $a \geq 0$. The question that will be considered is if, in the absence of any channel errors, the decoder will

cease eventually to make decoding mistakes. The part of the decoder that needs examination is the syndrome register. Because the decoding decisions are based on one actual constraint-length, the syndrome register is an $(n-k)$ -input memory, organised in as many shift registers (SRs) of lengths up to m stages [for an (n,k,m) systematic CSOC - see Fig. 5.1]. In the absence of any channel errors, starting at time-unit $a+1$, all subsequent syndrome blocks will be 'zero', hence the I/P to the syndrome register may be disconnected and the circuit is driven by the outputs of the k

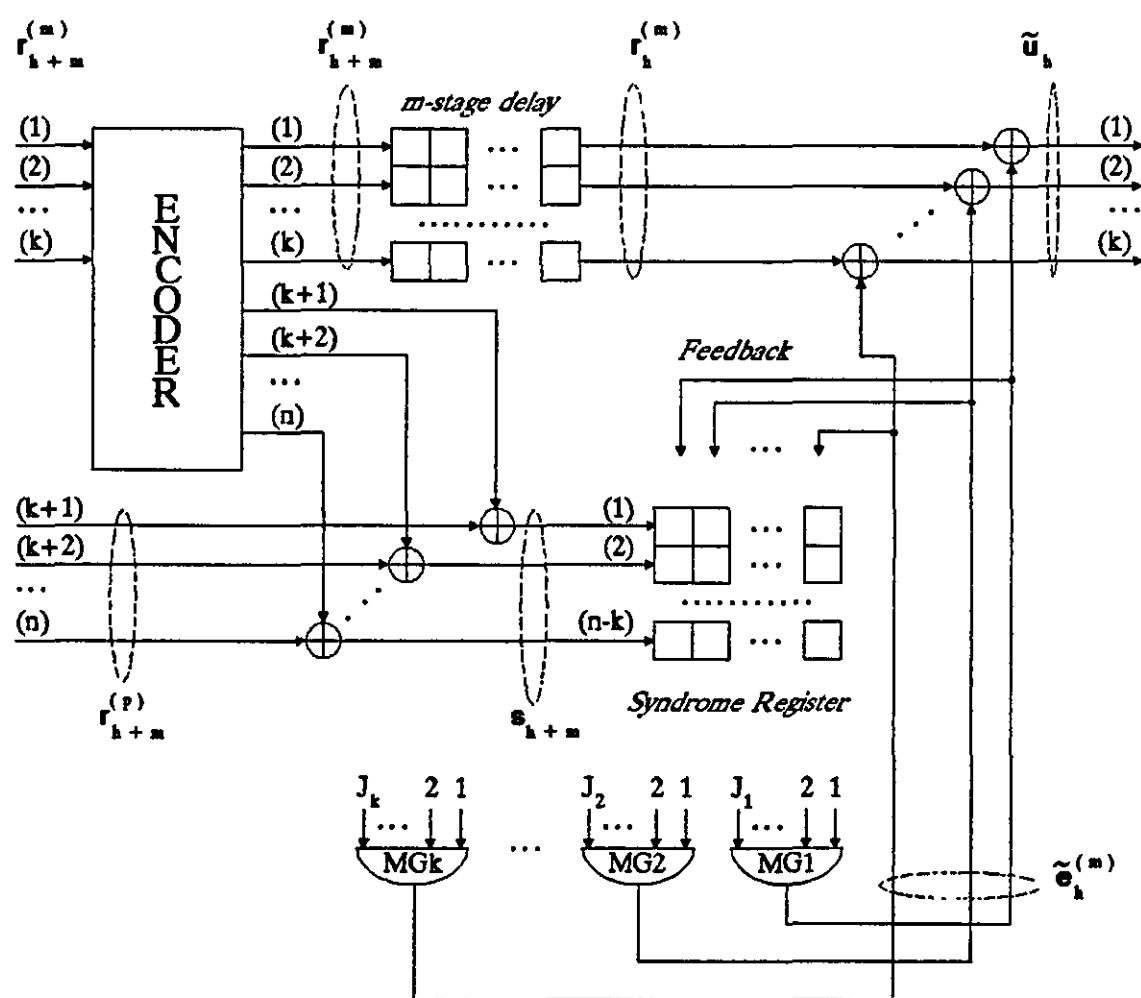


Figure 5.1: Complete majority-logic decoder for an (n,k,m) systematic convolutional code.

majority gates. Because of these gates, the circuit is a nonlinear feedback shift register (NFSR) and the task is to study its *autonomous* behaviour. For an interesting study,

the reader is referred to Massey & Liu [37].

Consider the syndrome register (SYRE) in its autonomous behaviour. The decoder will be able to 'safely' tackle channel errors again when the SYRE is completely reset (and in general earlier than that, but that depends on the specific circuit). Note that if no 'ones' are fed back, the SYRE will reset itself after at most m time-units; this situation corresponds to $\tilde{e}_a^{(\mu)} = 0$.

If, on the other hand, $\tilde{e}_a^{(\mu)} = 1$, then more than $T = J/2$ of the syndromes that check on the corresponding error bit are 'one', say $T+d$ of them, where $1 \leq d \leq T$. Because of the feedback action, these J syndromes are inverted. So, $J-T-d$ of these J syndromes are now 'one', i.e. the number of 'ones' in the syndrome register was reduced by $T+d-J+T+d = 2T-J+2d = 2d \geq 2$. Hence, in the autonomous state, every 'one' that is fed back reduces the weight of the syndrome register by at least 2. Hence, eventually all syndrome bits will be 'zero' and the decoder will be able to tackle channel errors again. The following theorem has been proved:

Theorem 5.10: Consider a feedback decoder for a CSOC. If no channel errors follow an erroneous decoding decision, the decoder will always recover from error propagation. ■

Robinson & Bernstein [38] & Robinson [39] have developed upper bounds for the extension of the error propagation effect in CSOCs. In particular they showed that one or two constraint-lengths are more than adequate for the recovery from a decoding error.

5.5.5. Distance Properties of CSOCs

In this paragraph, d_{\min} and J will be linked. Before that, though, it is necessary to explain how d_{\min} is calculated.

Theorem 5.11: Consider an (n,k,m) systematic convolutional code with parity-check matrix H . Let $[H]_m$ be the submatrix of H made of the first $(n-k)(m+1)$ rows and $n(m+1)$ columns of H .* Then d_{\min} equals the minimum number of col-

* See Definition 2.13 (p. 45).

umns of $[H]_m$, including at least one of the first k , that sum up to zero.

Proof: See Appendix 5.8 (§ A5.8.1., p. 409). ■

Theorem 5.12: Consider an (n, k, m) systematic CSOC with parity-check matrix H . Then, J_μ , the number of syndromes orthogonal on $e_0^{(\mu)}$ $/ 1 \leq \mu \leq k$ equals the weight of the μ th column of H (or the same of $[H]_m$).

Proof: See Appendix 5.8 (§ A5.8.2., p. 410). ■

Theorem 5.13: Consider a systematic CSOC which has at least J syndrome bits orthogonal on each of $e_0^{(1)}$. Then $d_{\min} = J+1$, i.e. the code is completely orthogonalizable.

Proof: See Appendix 5.8 (§ A5.8.3., p. 411). ■

5.6 CONCLUSIONS

This chapter discussed the method of threshold decoding and the properties required from codes so that they are threshold decodable.

All work starts with Massey's results, which are briefly presented in Sec. 5.2. The interest, in the rest of this thesis, remains with *majority-logic decoding*, for binary systematic codes. This algorithm requires, from the code, the ability to form J orthogonal parity checks, for each bit to be decoded. Over the BSC, the decoder estimates the error to be 1 iff the sum of these parity checks exceeds $\lceil J/2 \rceil$.

For an additive noise channel, it is known that each syndrome bit is a linear combination of error bits from the current and the last m blocks. These bits can then be used for majority decoding. Under *feedback decoding* (FD), the already estimated error bits are fed back to the syndrome register to cancel themselves out of those syndromes that check on them. This mode of operation improves the perform-

ance of the decoder because each syndrome depends on a smaller number of error bits, hence the likelihood of an erroneous estimation decreases; whenever the latter happens, though, *error propagation* occurs, but overall FD outperforms definite decoding (DD).

Both modes were examined, and matrix eqns were obtained relating one constraint-length of syndrome bits to the corresponding error bits, via a system matrix which was shown to have a structure made of *triangles* (for FD), or *squares* (for DD), of g -coefficients. This was then used to extract the necessary & sufficient conditions for a given syndrome bit to check on a given error bit. The last result was, in turn, used to obtain formulae for the number of syndromes checking an error bit and for the size of these checks. These results (original, in their generality) are given in Theorems 5.5 & 5.6, while examples, (which also verify them), are given in appendices. It is also shown that, under FD, the decoding circuit for the 1st block is identical to that for any subsequent block.

In Sec. 5.4 it is proved that the number of syndromes orthogonal on any error bit, cannot exceed $d_{\min} - 1$.

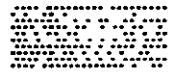
If all J_1 syndromes checking on $e_0^{(1)}$ are also orthogonal on this bit, the code is called *self-orthogonal*. The maximum of J_1 's $/1 \leq i \leq k$, is called the *effective constraint-length* of the code. A similar measure, called the *block effective constraint-length*, was introduced and defined to be the total number of distinct error bits involved in the decoding of e_0 . An example and instructions on how to calculate this, are also given. It is felt that this latter measure may be related more closely to the code's performance.

A brief discussion on error propagation concludes that it is limited, if no more errors occur.

Finally, it is shown that if a systematic CSOC has at least J syndromes checking on each error bit, then $d_{\min} = J+1$.



CHAPTER 6



Optimum Threshold for Majority Decoding of CSOCs

In Chapter 6, the threshold for majority decoding of CSOCs is re-examined; this threshold was set at $\lceil J/2 \rceil$, by Theorem 5.3 (p. 137). In this chapter, it is assumed that the threshold is not known. Its value, T , will be determined so that a certain performance measure is optimized. This measure is the probability P_d of decoding a bit in error.

An expression for P_d is developed, as a function of T , and then $P_d(T)$ is differentiated wrt T and set equal to 0. The solution of the resulting equation will give the optimum value of T for the given code and channel.

Complications arise because under feedback decoding (FD) the syndrome bits participating in the estimation of any error bit have different sizes (see Theorem 5.6) and hence different probabilities of error. This is true for definite decoding (DD) as well (see Theorem 5.5), but only in general. In most practical cases, the syndrome bits have the same size and hence the same probability of error; as a consequence, a simple solution is possible for DD.

In Sec. 6.1, some general results are obtained, including expressions for $P_d(T)$. In Sec. 6.2 the case of equal-size syndromes is investigated, while the last section deals with the optimum threshold under FD.

6.1 PERFORMANCE OF MAJORITY-LOGIC DECODING

Consider a set of J bits orthogonal (see Definition 5.2) on error bit e_n . Let Σ represent their (arithmetic) sum (i.e. $0 \leq \Sigma \leq J$) and T the threshold. e_n is erroneously decoded, either if e_n is received in error ($e_n=1$) and the decoder fails to recognize this (because $\Sigma \leq T$), or if e_n is received correctly ($e_n=0$) and the decoder 'thinks' otherwise (because $\Sigma > T$) (see Theorem 5.3, for the majority-decoding rule):

$$P_d = P(e_n=0)P(\Sigma > T | e_n=0) + P(e_n=1)P(\Sigma \leq T | e_n=1) \quad (6.1)$$

6.1.1. Introduction to the Optimum Threshold

Let p denote the probability of a channel bit error. Then, $P(e_n=1)=p$ and $P(e_n=0)=1-p$.

The probability that $\Sigma \leq T$, equals the probability that $\Sigma=0$, plus the probability that $\Sigma=1$, plus ... plus the probability that $\Sigma=T$. Also, the probability that $\Sigma > J$ is 0 and the probability that $\Sigma \leq J$ is 1. Then, from eqn (6.1)*:

$$P_d(T) = (1-p) \sum_{\mu=T+1}^J P(\Sigma=\mu | e_n=0) + p \sum_{\mu=0}^T P(\Sigma=\mu | e_n=1) \quad / T < J \quad (6.2a)$$

$$P_d(J) = p \quad (6.2b)$$

Note from eqns (6.2) that P_d is a function of T . Then it is sensible to attempt to optimize the decoding algorithm by using that value T_o , of T , which minimizes P_d . This can be achieved, of course, only if P_d decreases continuously as T increases from 0 and then at $T=T_o$ it starts increasing again; T_o is the *optimum threshold* for the particular code and channel. T is an integer variable with range $[0, J]$.

$$\text{Let} \quad \delta P_d(T) \triangleq P_d(T) - P_d(T-1) \quad / 0 < T \leq J \quad (6.3)$$

Then, from eqns (6.2) & (6.3):

$$\delta P_d(T) = pP(\Sigma=T | e_n=1) - (1-p)P(\Sigma=T | e_n=0) \quad / 0 < T < J \quad (6.4)$$

$$\text{From eqn (6.4):} \quad \delta P_d(T) = 0 \quad \longleftrightarrow$$

$$\longleftrightarrow \quad P(\Sigma=T | e_n=0) / P(\Sigma=T | e_n=1) = p / (1-p) \quad / 0 < T < J \quad (6.5)$$

* Transmission over the BSC is assumed (see Theorem 5.3, p. 137).

Note that in the above equations, the probabilities are conditioned on both $e_{\mathbf{m}}=0$ & $e_{\mathbf{m}}=1$. The following lemma will simplify calculations:

Lemma 6.1: Consider a set of J bits orthogonal on error bit $e_{\mathbf{m}}$ and let Σ be their (arithmetic) sum. Then:

$$P(\Sigma=\mu|e_{\mathbf{m}}=0) = P(\Sigma=J-\mu|e_{\mathbf{m}}=1) \quad / \mu=0,1,\dots,J \quad (6.6)$$

Proof: See Appendix 6.1 (§ A6.1.1., p. 412). ■

From the discussion on the derivation of result (6.5) and Lemma 6.1, the theorem below follows easily:

Theorem 6.1: Consider a set of J bits orthogonal on error bit $e_{\mathbf{m}}$ and let Σ represent their (arithmetic) sum. If $\delta P_d(T) < 0$ for $T < X$ and $\delta P_d(T) > 0$ for $T > X$ and, in case $X = \text{integer}$, $\delta P_d(X) = 0$, then, and only then, $T_o = [X]$ is the optimum threshold for the decoding of $e_{\mathbf{m}}$. Furthermore, if p is the channel bit-error probability, T_o may be determined by one of the following equations:

$$P(\Sigma=T_o|e_{\mathbf{m}}=0)/P(\Sigma=T_o|e_{\mathbf{m}}=1) = p/(1-p) \quad (6.7a)$$

$$P(\Sigma=T_o|e_{\mathbf{m}}=0)/P(\Sigma=J-T_o|e_{\mathbf{m}}=0) = p/(1-p) \quad (6.7b)$$

Proof: See Appendix 6.1 (§ A6.1.2., p. 412). ■

To solve the above eqn(s), one needs to express the conditional probabilities $P(\Sigma=\mu|e_{\mathbf{m}}=0)$ in terms of p and the code parameters. The probability that μ of the J syndrome bits are 1, depends on the probability that a syndrome bit is 1. Let:

$$P_i \triangleq P(s_i=1|e_{\mathbf{m}}=0) \quad / 1 \leq i \leq J \quad (6.8)$$

6.1.2. Exact and Approximate Value of P_i

The following theorem relates the probability that a syndrome is 1, with its size and the channel bit-error probability.

Theorem 6.2: If p is the probability that a bit is 1, then the probability, P , that the mod-2 sum of c statistically independent bits is 1, is

$$P = \frac{1}{2}[1-(1-2p)^c] \quad (6.9)$$

Proof: See Appendix 6.1 (§ A6.1.3., p. 413).

Note that p is a very small positive number (typically less than 10^{-3}). It may be possible therefore, to obtain an approximate expression for $(1-2p)^c$.

The following two approximations of $(1-2p)^c$ are derived in Appendix 6.1 (§ A6.1.4., p. 414):

$$\lim_{p \rightarrow 0} (1-2p)^c = e^{-2pc} \quad (6.10)$$

$$(1-2p)^c \approx 1-2pc \quad /pc \ll 1 \quad (6.11)$$

As mentioned earlier on, typical values of p range between, say, 10^{-6} and 10^{-3} . On the other hand c , the size of a syndrome, is usually small, say less than 100. From the restriction ($p \rightarrow 0$ & $pc < 1$) it is obvious that the approximation becomes better as $p \rightarrow 0$. On the other hand, $c=100$ also corresponds to a worst-case condition. TABLE 6.1, below, illustrates the accuracy of the approximations.

TABLE 6.1

		$(1-2p)^c$	e^{-2pc}	$1-2pc$
$p=10^{-3}$	$c=100$	0.818567	0.818731	0.800000
$p=10^{-3}$	$c=50$	0.904747	0.904837	0.900000
$p=10^{-3}$	$c=10$	0.980179	0.980199	0.980000
$p=10^{-4}$	$c=100$	0.980197	0.980199	0.980000
$p=10^{-4}$	$c=50$	0.990049	0.990050	0.990000
$p=10^{-4}$	$c=10$	0.998002	0.998002	0.998000

Note that for worst-case conditions ($p=10^{-3}$ & $c=100$) the 1st approximation is perfect to within 3 significant digits, while for a typical case ($p=10^{-4}$ & $c=50$), it is perfect to within 5 significant digits.

From eqns (6.9), (6.10) & (6.11):

$$P \approx (1 - e^{-2pc})/2 \quad /p \rightarrow 0 \quad (6.12)$$

$$P \approx pc \quad /pc \ll 1 \quad (6.13)$$

Note also, that:

$$e^{-2pc} \approx 1 - 2pc \quad /pc \ll 1 \quad (6.14)$$

TABLE 6.2

		$\frac{1}{2}[1 - (1 - 2p)^c]$	$(1 - e^{-2pc})/2$	pc
$p=10^{-3}$	$c=100$	0.090717	0.090635	0.100000
$p=10^{-3}$	$c=50$	0.047627	0.047581	0.050000
$p=10^{-3}$	$c=10$	0.009910	0.009901	0.010000
$p=10^{-4}$	$c=100$	0.009902	0.009901	0.010000
$p=10^{-4}$	$c=50$	0.004976	0.004975	0.005000
$p=10^{-4}$	$c=10$	0.000999	0.000999	0.001000

Consider the results of TABLE 6.2, above. Under worst-case conditions ($p=10^{-3}$ & $c=100$), the approximation error is 0.09% & 10%, while for a typical case ($p=10^{-4}$ & $c=50$), the error is 0.02% & 0.5%.

6.1.3. Calculation of $P(\Sigma=u|e_{\mathbf{m}}=0)$

P_i was defined by eqn (6.8). Consider the following definitions, as well:

$$Q_i \triangleq 1 - P_i \quad /i=1, 2, \dots, J \quad (6.15)$$

$$K_i \triangleq P_i / Q_i \quad /i=1, 2, \dots, J \quad (6.16)$$

$P(\Sigma=u|e_{\mathbf{m}}=0)$ is the probability that exactly μ of the J syndromes are 1. Then:

$$P(\Sigma=u|e_{\mathbf{m}}=0) = \sum_{\substack{x(1), y(J) \\ 1 \leq x(i) < x(i+1) \leq J \\ 1 \leq y(j) < y(j+1) \leq J \\ x(1) \neq y(j) \\ 1 \leq i \leq \mu, 1 \leq j \leq J-\mu}} P_{x(1)} P_{x(2)} \cdots P_{x(\mu)} Q_{y(1)} Q_{y(2)} \cdots Q_{y(J-\mu)} \quad (6.17)$$

Eqn (6.17) needs some explanation. The probability that μ of the J syndromes are 1, is the sum of the probabilities that any specific combination of μ syndromes, say, $s_{x(1)}, s_{x(2)}, \dots, s_{x(\mu)}$, are 1 and all the rest, $s_{y(1)}, s_{y(2)}, \dots, s_{y(J-\mu)}$,

are 0, over all $C(J, \mu)$ distinct combinations. Obviously, $1 \leq x(i) < x(i+1) \leq J$ & $1 \leq y(j) < y(j+1) \leq J$ and $x(i) \neq y(j)$ for all $i=1, 2, \dots, \mu$ & $j=1, 2, \dots, J-\mu$. The probability of the above mentioned combination is

$$P_{x(1)} P_{x(2)} \cdots P_{x(\mu)} Q_{y(1)} Q_{y(2)} \cdots Q_{y(J-\mu)}$$

because the channel noise is random, hence the error bits are statistically independent.

Using (6.15) & (6.16), eqn (6.17) can be re-written:

$$P(\Sigma=\mu | e_{\Sigma}=0) = (Q_1 Q_2 \cdots Q_J) \sum_{\substack{x(1) \\ 1 \leq x(1) < x(1+1) \leq J \\ 1 \leq i \leq \mu}} K_{x(1)} K_{x(2)} \cdots K_{x(\mu)} \quad (6.18)$$

$$P(\Sigma=\mu | e_{\Sigma}=0) = (P_1 P_2 \cdots P_J) \sum_{\substack{y(j) \\ 1 \leq y(j) < y(j+1) \leq J \\ 1 \leq j \leq J-\mu}} [K_{y(1)} K_{y(2)} \cdots K_{y(J-\mu)}]^{-1} \quad (6.19)$$

Appendix 6.1 (§ A6.1.5., p. 416) contains some examples on the calculation of $P(\Sigma=\mu | e_{\Sigma}=0)$. In Example A6.1.1, two expressions for $P(\Sigma=3 | e_{\Sigma}=0)$ were developed (corresponding to the two eqns, above). The first one required eight multiplications more than the other. This illustrates the need to minimize the number of calculations required for the computation of P_d .

For the arbitrary case of $J=4$ & $p=10^{-4}$, the probabilities were found to be as follows:

$$\begin{aligned} P(\Sigma=0 | e_{\Sigma}=0) &= 9.978 \times 10^{-1} & P(\Sigma=1 | e_{\Sigma}=0) &= 2.195 \times 10^{-3} \\ P(\Sigma=2 | e_{\Sigma}=0) &= 1.467 \times 10^{-6} & P(\Sigma=3 | e_{\Sigma}=0) &= 3.415 \times 10^{-10} \\ P(\Sigma=4 | e_{\Sigma}=0) &= 2.157 \times 10^{-14} \end{aligned}$$

Note that $P(\Sigma=\mu | e_{\Sigma}=0)$ decreases steadily, as μ increases from 0 to 4.

One important question with respect to the probability distribution $P(\Sigma=\mu | e_{\Sigma}=0)$ / $\mu=0, 1, \dots, J$ arises: "Is $P(\Sigma=\mu | e_{\Sigma}=0)$ a decreasing function of μ ?" If the answer is yes, then the existence of an optimum threshold for all codes is guaranteed; furthermore, this threshold will depend on the code parameters (specifically, c_1, c_2, \dots, c_J) and on the channel probability of error, p .

6.1.4. General Expressions for P_d

From eqn (6.2) & Lemma 6.1:

$$P_d(T) = (1-p) \sum_{\mu=T+1}^J P(\Sigma=\mu | e_n=0) + p \sum_{\mu=J-T}^J P(\Sigma=\mu | e_n=0) \quad / T < J \quad (6.20)$$

$$P_d(J) = p \quad (6.21)$$

Note from eqn (6.20) that the 1st summation uses the $J-T$ last probabilities of the distribution $P(\Sigma=\mu | e_n=0)$, while the 2nd summation uses the last $T+1$ probabilities. Since there is a common set of probabilities in each summation, from $\text{MAX}\{T+1, J-T\}$ to J , this can be exploited to reduce the number of calculations:

$$P_d(T) = \sum_{\mu=\Gamma}^J P(\Sigma=\mu | e_n=0) + \beta \sum_{\mu=J+1-\Gamma}^{\Gamma-1} P(\Sigma=\mu | e_n=0) \quad (6.22)$$

where: $\Gamma \triangleq \text{MAX}\{T+1, J-T\}$, $T < J$ and

$$\beta = \begin{cases} 1-p & / T+1 < J-T \\ 0 & / T+1 = J-T \\ p & / T+1 > J-T \end{cases} \quad \begin{matrix} \Longleftarrow \\ \Longleftarrow \\ \Longleftarrow \end{matrix} \quad \begin{matrix} T < (J-1)/2 \\ T = (J-1)/2 \\ T > (J-1)/2 \end{matrix}$$

Note that the expression for P_d , in (6.22), is economical with calculations, compared with (6.20). Let (6.22) be separated into its three cases:

For $J > T > (J-1)/2$:

$$P_d(T) = \sum_{\mu=T+1}^J P(\Sigma=\mu | e_n=0) + p \sum_{\mu=J-T}^T P(\Sigma=\mu | e_n=0) \quad (6.23a)$$

For $T < (J-1)/2$:

$$P_d(T) = \sum_{\mu=J-T}^J P(\Sigma=\mu | e_n=0) + (1-p) \sum_{\mu=T+1}^{J-T-1} P(\Sigma=\mu | e_n=0) \quad (6.23b)$$

$$P_d(T) = \sum_{\mu=T+1}^J P(\Sigma=\mu | e_n=0) \quad / T = (J-1)/2 \quad (6.23c)$$

The above relations are used in Example A6.1.4 (see Appendix 6.1, § A6.1.5., p. 417), to calculate the probability of decoding error for the case with parameters $J=4$, $p=10^{-4}$ and $c_1=1$, $c_2=3$, $c_3=6$ & $c_4=12$. The optimum threshold for this

case is $T_0=2$.

In calculating $P_d(T)$ no attention was paid to the probability of erroneous decoding at an earlier stage. In other words, it was assumed that there was no error propagation (see Paragraph 5.5.4., p. 145). It can be seen from eqn (A5.2.1) (p. 381) that this is not the case. Nevertheless, consideration of the past decoding errors would make calculations virtually impossible. Hence, it is assumed that either $P_d(T)$ expresses the probability of the *first decoding error* P_{fe} , or that "a magic genie always feeds back the correct channel error symbol" [13], thus eliminating error propagation; however, the decoding decision is not affected in any other way by the genie action. Then, P_d is simply P_{gd} . Clearly, $P_{fe}=P_{gd}$. The results derived in this paragraph can be put into the following theorem:

Theorem 6.3: Consider a set of J syndromes orthogonal on e_m and let Σ denote their (arithmetic) sum. If T is the threshold used, then the probability P_{fe} of first decoding error, or (the same) the probability P_{gd} of 'genie decoding', for error bit e_m , is given by:

$$P_{fe}(J) = P_{gd}(J) = p \quad (6.24a)$$

$$P_{fe}(T) = P_{gd}(T) = \sum_{\mu=\Gamma}^J P(\Sigma=\mu | e_m=0) + \beta \sum_{\mu=J+1-\Gamma}^{\Gamma-1} P(\Sigma=\mu | e_m=0) \quad (6.24b)$$

where: $\Gamma \triangleq \text{MAX}\{T+1, J-T\}$, $T < J$ and

$$\beta = \begin{cases} 1-p & /T+1 < J-T \\ 0 & /T+1 = J-T \\ p & /T+1 > J-T \end{cases} \begin{matrix} <=> \\ <=> \\ <=> \end{matrix} \begin{matrix} T < (J-1)/2 \\ T = (J-1)/2 \\ T > (J-1)/2 \end{matrix}$$

The concept of the optimum threshold will be discussed in the example below, for two cases and a number of channel error probabilities, p .

Example 6.1: Consider the case of Example A6.1.2 ("Case 1") and also "Case 2": $J=8$, $c_1=4, c_2=10, c_3=16, c_4=20, c_5=22, c_6=28, c_7=38, c_8=50$.

TABLE 6.3

Threshold	Probability P_{fe} for Case 1			
	$p=10^{-2}$	$p=10^{-3}$	$p=10^{-4}$	$p=10^{-5}$
0	1.896×10^{-1}	2.167×10^{-2}	2.200×10^{-3}	2.197×10^{-4}
1	1.223×10^{-2}	1.463×10^{-4}	1.467×10^{-6}	1.470×10^{-8}
2	4.058×10^{-4}	4.799×10^{-7}	4.882×10^{-10}	4.889×10^{-13}
3	1.917×10^{-3}	2.169×10^{-5}	2.200×10^{-7}	2.200×10^{-9}
4	1.000×10^{-2}	1.000×10^{-3}	1.000×10^{-4}	1.000×10^{-5}

TABLE 6.4

Threshold	Probability P_{fe} for Case 2			
	$p=10^{-2}$	$p=10^{-3}$	$p=10^{-4}$	$p=10^{-5}$
0	7.904×10^{-1}	1.689×10^{-1}	1.860×10^{-2}	1.878×10^{-3}
1	4.258×10^{-1}	1.273×10^{-4}	1.447×10^{-4}	1.466×10^{-6}
2	1.479×10^{-1}	5.288×10^{-4}	6.103×10^{-7}	6.192×10^{-10}
3	3.277×10^{-2}	1.304×10^{-5}	1.518×10^{-9}	1.541×10^{-13}
4	4.854×10^{-3}	2.064×10^{-7}	2.413×10^{-12}	2.451×10^{-17}
5	1.864×10^{-3}	5.310×10^{-7}	6.103×10^{-11}	6.192×10^{-15}
6	4.317×10^{-3}	1.274×10^{-5}	1.447×10^{-8}	1.466×10^{-11}
7	7.984×10^{-3}	1.691×10^{-4}	1.860×10^{-6}	1.878×10^{-8}
8	1.000×10^{-2}	1.000×10^{-3}	1.000×10^{-4}	1.000×10^{-5}

From TABLE 6.3 it is obvious that the optimum threshold for Case 1 is $T_0 = 2$, i.e. $\lceil J/2 \rceil$ (as set by Theorem 5.3).

From TABLE 6.4, it can be seen that for Case 2 the optimum threshold is $T_0 = \lceil J/2 \rceil = 4$ for $p < 10^{-2}$, but for $p = 10^{-2}$, it is $T_0 = 5$.

A first conclusion may be risked at this stage: For adequately small channel error probabilities p , the optimum threshold is indeed $T_0 = \lceil J/2 \rceil$, but as p increases so does T_0 ; this effect becomes more dramatic for 'longer' codes (i.e. for codes with large c_i s, or the same for codes with large effective constraint-lengths n_x - see Definition 5.9).

Consider two more cases, both with $J = 11$ syndromes, but with different effective constraint-lengths.

Example 6.2: Let the following two $J = 11$ cases:

Case 1: $c_1=4, c_2=10, c_3=15, c_4=20, c_5=30, c_6=40, c_7=60, c_8=80,$
 $c_9=100, c_{10}=120, c_{11}=150$

Case 2: $c_1=40, c_2=45, c_3=50, c_4=60, c_5=70, c_6=80, c_7=90, c_8=100,$
 $c_9=110, c_{10}=120, c_{11}=150$

The optimum threshold for each of the two cases is calculated, for various channel error probabilities, p . Note that although both cases use the same number of syndromes, the effect of increased p is more dramatic for Case 2 (which has $n_e = 916$, as opposed to $n_e = 630$ for Case 1).

TABLE 6.5 ($\lceil J/2 \rceil = 6$).

p	T_o /Case 1	T_o /Case 2
0.010	7	10
0.007	7	8
0.005	7	8
0.002	6	7
0.001	6	6

Instead of the term 'codes', the term 'cases' has been used. This is so because the choice of the c_i s is arbitrary and it is not known whether or not codes with such parameters exist. Furthermore, the channel error probability was used without any regard to the channel capacity. See Appendix 6.2 (p. 418) for an introduction to channel capacity and tables of R_{\max} versus p and p_{\max} versus R .

6.2 OPTIMUM THRESHOLD FOR DEFINITE DECODING

Definite decoding (DD) is the mode with no feedback from the decoding decision device to the syndrome register (see Definition 5.7). Obviously, there is no error propagation effect in the DD mode, hence the probability of first decoding error P_{fe} is also the probability of decoding error P_d . According to Theorem 5.9, CSOCs have the property that the set of syndromes orthogonal on $e_h^{(1)}$ is the same under both the FD & DD modes.

Under DD, according to Theorem 5.5 (p. 140), the size, c_j , of syndrome bit $s_h^{(j)}$ $/1 \leq j \leq n-k$ & $h \geq m$, is $c_j = 1 + w[g_{k+j}^{(1)}] + w[g_{k+j}^{(2)}] + \dots + w[g_{k+j}^{(k)}]$. This implies that, unless $n-k = 1$, the syndromes used for the decoding of an error bit will have different sizes, in general.

Note, though, that it is very likely for the syndrome bits to have the same size, under DD. In contrast, under FD, the size of $s_h^{(j)}$ depends not only on j , but also on h (see Theorem 5.6, p. 141). Hence, under FD, it is virtually impossible to have syndrome bits of the same size.

6.2.1. Study of $P(\Sigma=\mu|e_\mu=0)$

Consider now the case where all syndromes checking a particular error bit have the same size. Let this size be c : $c_1 = c_2 = \dots = c_j = c$. Then, from Theorem 6.2:

$$P \triangleq P_1 = P_2 = \dots = P_j = [1 - (1-2p)^c] / 2 \quad (6.25)$$

$$\text{Also,} \quad Q \triangleq 1-P \quad (6.26)$$

$$\text{and:} \quad K \triangleq P/Q \quad (6.27)$$

From (6.13), for $pc \ll 1$: $K = P/(1-P) \approx pc/(1-pc) \approx pc$

$$K \approx pc \quad /pc \ll 1 \quad (6.28)$$

From eqn (6.18), since there are $C(J, \mu)$ distinct products of the J K_i s, taken μ at a time:

$$P(\Sigma=\mu|e_\mu=0) = Q^J K^\mu \binom{J}{\mu} \quad / \mu=0, 1, \dots, J \quad (6.29)$$

Consider now the variation of $P(\Sigma=\mu|e_\mu=0)$ with μ :

$$\delta(\mu) \triangleq P(\Sigma=\mu|e_\mu=0) - P(\Sigma=\mu-1|e_\mu=0) \quad / \mu \geq 1 \quad (6.30)$$

Then: $\delta(\mu) \leq 0 \quad \longleftrightarrow$

$$\longleftrightarrow Q^J K^\mu J! / \mu! / (J-\mu)! \leq Q^J K^{\mu-1} J! / (\mu-1)! / (J-\mu+1)! \quad *$$

$$\longleftrightarrow K/\mu \leq 1/(J-\mu+1) \quad \longleftrightarrow K(J+1) - K\mu \leq \mu$$

$$\longleftrightarrow \mu \geq (J+1)K/(K+1) = (J+1)(P/Q)/(P/Q+1) = (J+1)P$$

If $\delta(\mu) \leq 0$, for all μ , since $\mu \geq 1$, it is enough for $(J+1)P$ to be < 1 . Since P depends on p & c , what is required is the set of conditions on p , c & J , that make $(J+1)P < 1$.

* $x/y/z$ denotes $(x/y)/z$.

$$P = [1 - (1-2p)^c]/2 < 1/(J+1) \quad \Longleftrightarrow \quad 1-2/(J+1) < (1-2p)^c$$

$$\Longleftrightarrow \quad (J-1)/(J+1) < (1-2p)^c \quad (A)$$

$$(A) \quad \Longleftrightarrow \quad [(J-1)/(J+1)]^{1/c} < 1-2p \quad \Longleftrightarrow$$

$$\Longleftrightarrow \quad p < \{1 - [(J-1)/(J+1)]^{1/c}\}/2$$

$$\text{Also: } (A) \quad \Longleftrightarrow \quad \ln[(J-1)/(J+1)] < c \ln(1-2p)$$

$$\Longleftrightarrow \quad c < \ln[(J-1)/(J+1)]/\ln(1-2p)$$

Hence:

Theorem 6.4: Consider an (n, k, m) binary CSOC, with J syndromes, each of size c , checking on $e_h^{(i)}/h \geq m$. Then, $P(\Sigma=\mu | e_h^{(i)}=0)$ is a continuously decreasing function of μ , for all $\mu \geq (J+1)P$ / $P=[1-(1-2p)^c]/2$. Also, $(J+1)P < 1$ if, and only if,

$$\text{either } p < \{1 - [(J-1)/(J+1)]^{1/c}\}/2 \quad (6.31)$$

$$\text{or } c < \ln[(J-1)/(J+1)]/\ln(1-2p) \quad (6.32)$$

From the above, $P(\Sigma=\mu | e_h^{(i)}=0)$ decreases continuously, iff either of (6.31) or (6.32) holds true. These two inequalities are not difficult to be satisfied, but do not hold true in 'extreme' cases. For example, from the 1st one and for $J=10$ & $c=50$, $p < 2 \times 10^{-3}$.

As a conclusion, $P(\Sigma=\mu | e_h^{(i)}=0)$, 'normally', is expected to decrease as μ increases, but if J , c & p are such that $(J+1)P > 1$, then $P(\Sigma=\mu | e_h^{(i)}=0)$ will increase with μ up to $\approx (J+1)P$ and then it will start decreasing.

6.2.2. Optimum Threshold

From eqn (6.4) and Lemma 6.1:

$$\delta P_d(T) = pP(\Sigma=J-T | e_h^{(i)}=0) - (1-p)P(\Sigma=T | e_h^{(i)}=0) \quad (A)$$

Using eqn (6.29) in eqn (A):

$$\delta P_d(T) = pQ^J K^{J-T} \binom{J}{J-T} - (1-p)Q^J K^T \binom{J}{T} \quad (B)$$

Since, $\binom{J}{J-T} = J!/(J-T)!/T! = \binom{J}{T}$, eqn (B) gives: *

* $x/y/z$ denotes $(x/y)/z$.

$$\delta P_d(T) = Q^J \left(\frac{J}{T} \right) \left[pK^{J-T} - (1-p)K^T \right] \quad (6.33)$$

$$\text{From (6.33): } \delta P_d(T) > 0 \quad \longleftrightarrow \quad pK^{J-T} > (1-p)K^T$$

$$\longleftrightarrow K^{J-2T} > (1-p)/p \quad \longleftrightarrow \quad (J-2T)\ln K > \ln[(1-p)/p]$$

$$\longleftrightarrow J-2T < \ln[(1-p)/p]/\ln K^*$$

$$\longleftrightarrow T > J/2 - \ln[(1-p)/p]/2\ln K = J/2 + \ln[p/(1-p)]/2\ln K$$

$$\text{Hence, } \delta P_d(T) > 0 \quad \longleftrightarrow \quad T > J/2 + \ln[p/(1-p)]/2\ln K$$

$$\text{Similarly, } \delta P_d(T) < 0 \quad \longleftrightarrow \quad T < J/2 + \ln[p/(1-p)]/2\ln K$$

and, if $\delta P_d(T)=0$ has a solution,

$$\delta P_d(T) = 0 \quad \longleftrightarrow \quad T = J/2 + \ln[p/(1-p)]/2\ln K$$

From the above results and Theorem 6.1:

Theorem 6.5: Consider an (n,k,m) binary CSOC with J syndromes, each of size c , checking on error bit $e_h^{(1)}$ / $h \geq m$. Then, the optimum threshold for $e_h^{(1)}$, is:

$$T_o = \lfloor J/2 + \frac{1}{2} \ln[p/(1-p)]/H(p,c) \rfloor \quad (6.34a)$$

$$\text{where: } H(p,c) \triangleq \ln \left\{ \frac{[1-(1-2p)^c]}{[1+(1-2p)^c]} \right\} \quad (6.34b)$$

Note that $H(p,1) = \ln\{[1-(1-2p)]/[1+(1-2p)]\} = \ln[p/(1-p)]$.

Consider now the behaviour of $H(p,1)/H(p,c)$. The results of Theorem 6.6, are proved in Appendix 6.3 (p. 421):

Theorem 6.6: Let $F(p,c) \triangleq H(p,1)/H(p,c)$, where $H(p,c)$ is defined by (6.34b). Then, the following hold true:

$$F(p,c) \geq 1 \quad (6.35a)$$

$$dF(p,c)/dp > 0 \quad \text{for } 0 < p < 0.5 \quad (6.35b)$$

$$dF(p,c)/dc > 0 \quad \text{for } c \geq 1 \quad (6.35c)$$

$$\lim_{p \rightarrow 0} F(p,c) = 1/(1+\ln c/\ln p) \quad /pc \ll 1 \quad (6.35d)$$

$$\lim_{p \rightarrow 0.5} F(p,c) = \lim_{c \rightarrow +\infty} F(p,c) = +\infty \quad (6.35e)$$

* From (6.9), if $p < \frac{1}{2}$, then $F < \frac{1}{2}$, hence $K < 1$, hence $\ln K < 0$.

The following conclusions can be drawn about T_o , from the results about F :

$$\begin{aligned} \text{From (6.35a): } F \geq 1 & \quad \longleftrightarrow \quad J/2 + F/2 \geq J/2 + 1/2 = (J+1)/2 \\ \longrightarrow [J/2 + F/2] & \geq [(J+1)/2] \longrightarrow \quad (\text{using Theorem 6.5}) \\ \longrightarrow T_o & \geq [(J+1)/2] = \lceil J/2 \rceil \quad (6.36) \end{aligned}$$

Consider the conditions under which $T_o = (T_o)_{\min}$. From Theorem 6.5 & reln (6.36):

$$T_o = [J/2 + F/2] = (T_o)_{\min} = [(J+1)/2]$$

$$\text{If } J=\text{even}, \quad T_o = J/2 + [F/2] = (T_o)_{\min} = J/2 \longrightarrow F < 2.$$

$$\begin{aligned} \text{If } J=\text{odd}, \quad T_o &= [(J+1)/2 + (F-1)/2] = (J+1)/2 + [(F-1)/2] = \\ &= (T_o)_{\min} = (J+1)/2 \longrightarrow F < 3. \end{aligned}$$

So, if $F < 2$, $T_o = (T_o)_{\min}$, and from (6.35d):

$$1/(1+\ln c/\ln p) < 2 \longrightarrow \ln c/\ln p > -1/2 \longrightarrow p < 1/c^2$$

The above condition on p is consistent with $p \ll 1/c$, hence the optimum threshold equals $\lceil J/2 \rceil$, the nominal threshold, for $p < 1/c^2$. This result was obtained in a different way, in Lemma A6.3.2 (p. 426).

Note from Theorems 6.5 & 6.6 that T_o is an increasing function of c or p . Note also that since F increases without bound as $p \rightarrow 0.5$, or $c \rightarrow +\infty$, so does T_o ; but T_o should not exceed J .

The following theorem is based on the above results:

Theorem 6.7: Consider an (n, k, m) binary CSOC with J syndromes, each of size c , checking on error bit $e_h^{(i)}/h \geq m$. Then the optimum threshold, for $e_h^{(i)}$, is an increasing function of c , or p , which satisfies the following:

$$T_o \geq [(J+1)/2] = \lceil J/2 \rceil \quad (6.37a)$$

$$\lim_{p \rightarrow 0} \{T_o\} = \lfloor \frac{1}{2}(J+1) \rfloor = \lceil \frac{1}{2}J \rceil \quad \text{OR} \quad T_o = \lceil \frac{1}{2}J \rceil \text{ if } pc^2 < 1 \quad (6.37b)$$

$$\lim_{p \rightarrow 0.5} \{T_o\} = \lim_{c \rightarrow +\infty} \{T_o\} = J \quad (6.37c)$$

In general, as p increases, T_0 increases in steps of one, from $\lceil J/2 \rceil$ to J . It is difficult though to obtain analytical expressions for the threshold values of p , i.e. for these values of p that cause an increase of T_0 by one. This can be done numerically, or via graphs. Theorem A6.3.4 (p. 425) provides some information towards this end.

6.2.3. Probability of Decoding Error

Consider now the probability of decoding error, with the optimum threshold T_0 . From (6.37a),

$$T_0 \geq \lceil J/2 \rceil \geq J/2 \quad \longrightarrow \quad J - T_0 \leq J/2 \leq T_0 < T_0 + 1 \quad \longrightarrow$$

$$\Gamma \triangleq \text{MAX}\{T_0 + 1, J - T_0\} = T_0 + 1$$

$$\lfloor (J+1)/2 \rfloor \leq (J+1)/2 < \lfloor (J+1)/2 \rfloor + 1 \quad \longrightarrow$$

$$\lfloor (J+1)/2 \rfloor > (J-1)/2 \quad (\text{and since, by Theorem 6.7})$$

$$T_0 \geq \lfloor (J+1)/2 \rfloor \quad \longrightarrow \quad T_0 > (J-1)/2$$

The first two eqns of the following theorem are based on the above conclusions, the expression for $P_d(T)$ (see Theorem 6.3) and the expression for $P(\Sigma=\mu | e_h^{(i)}=0)$ [see eqn (6.29)]. The last result is proved in Appendix 6.4 (p. 426).

Theorem 6.8: Consider an (n,k,m) binary CSOC with J syndromes, each of size c , checking on error bit $e_h^{(i)}$ / $h \geq m$. If the optimum threshold is used, the probability of decoding $e_h^{(i)}$ in error, under DD, is given by:

$$P_d(J) = p \quad (6.38a)$$

$$P_d(T_0) = Q^J \left[\sum_{\mu=T_0+1}^J K^\mu \binom{J}{\mu} + p \sum_{\mu=J-T_0}^{T_0} K^\mu \binom{J}{\mu} \right] \quad (6.38b)$$

$$\text{Also:} \quad P_d(T_0) < p \quad / T_0 < J \quad (6.38c)$$

Hence, the use of the optimum threshold guarantees that $P_d < p$ however high the BSC's error rate, p , is.

Consider now a few examples that illustrate the gains

that may be achieved by the use of the optimum, instead of the nominal, threshold.

Example 6.3: Let the following arbitrary cases:

Case:	1	2	3	4	5	6	7	8
J:	6	6	12	12	30	30	30	50
c:	20	100	20	100	20	100	400	500

For each of these cases, and for various error probabilities, p , the following will be calculated: The optimum threshold, T_o , the error extension ratio $EER \triangleq P_d(\lceil J/2 \rceil)/p$ (i.e. the ratio of the probability of decoding error with the nominal threshold, over p), and the error gain $G_e \triangleq P_d(\lceil J/2 \rceil)/P_d(T_o)$ (i.e. the ratio of the probability of decoding error with the nominal threshold, over that with the optimum threshold). p will range between $1/c^2$ and 0.01. Note that $1/c^2$ is the approximate value of the break point, after which the optimum threshold 'departs' from the nominal one.

TABLE 6.6 $\{c = 20; 1/c^2 = 2.5 \times 10^{-3}\}$

p	J = 6			J = 12			J = 30		
	T_o	G_e	EER	T_o	G_e	EER	T_o	G_e	EER
2.5×10^{-3}	4	1.0	3×10^{-2}	7	1.0	2×10^{-4}	16	1.0	2×10^{-11}
4×10^{-3}	4	1.5	1×10^{-1}	7	1.5	2×10^{-3}	16	1.5	1×10^{-8}
7×10^{-3}	4	2.3	4×10^{-1}	7	2.2	3×10^{-2}	16	2.2	1×10^{-5}
1×10^{-2}	4	2.8	9×10^{-1}	7	2.6	1×10^{-1}	16	2.5	5×10^{-4}

TABLE 6.7 $\{c = 100; 1/c^2 = 1 \times 10^{-4}\}$

p	J = 6			J = 12			J = 30		
	T_o	G_e	EER	T_o	G_e	EER	T_o	G_e	EER
1×10^{-4}	4	1.0	1×10^{-3}	7	1.0	7×10^{-8}	16	1.0	2×10^{-20}
2×10^{-4}	4	2.0	1×10^{-2}	7	2.0	4×10^{-6}	16	2.0	3×10^{-16}
4×10^{-4}	4	3.7	8×10^{-2}	7	3.6	2×10^{-4}	16	3.5	5×10^{-12}
7×10^{-4}	4	5.6	4×10^{-1}	7	5.3	4×10^{-3}	16	5.0	1×10^{-8}
1×10^{-3}	4	6.8	9×10^{-1}	7	6.0	3×10^{-2}	16	5.5	9×10^{-7}
2×10^{-3}	4	7.4	4×10^0	7	5.7	6×10^{-1}	16	4.7	2×10^{-3}
4×10^{-3}	5	13.9	1×10^1	8	9.6	6×10^0	17	6.7	6×10^{-1}
7×10^{-3}	6	21.7	2×10^1	10	17.6	2×10^1	19	10.9	1×10^1

TABLE 6.8

P	J = 30, c = 400			J = 50, c = 500		
	T_o	G_o	EER	T_o	G_o	EER
4×10^{-6}	15	1.0	6×10^{-32}	26		
7×10^{-6}	16	1.1	3×10^{-28}	26		
1×10^{-5}	16	1.6	6×10^{-26}	26	6.0	3×10^{-20}
2×10^{-5}	16	3.1	2×10^{-21}	26	7.4	4×10^{-34}
4×10^{-5}	16	5.9	4×10^{-17}	26	8.5	8×10^{-27}
7×10^{-5}	16	8.9	1×10^{-13}	26	11.3	4×10^{-21}
1×10^{-4}	16	10.4	2×10^{-11}	26	11.8	2×10^{-17}
2×10^{-4}	16	10.3	2×10^{-07}	26	9.2	5×10^{-11}
4×10^{-4}	17	9.7	8×10^{-04}	27	13.1	2×10^{-05}
7×10^{-4}	17	13.5	2×10^{-01}	28	15.2	6×10^{-02}
1×10^{-3}	18	17.8	$3 \times 10^{+00}$	29	19.5	$2 \times 10^{+00}$
2×10^{-3}	22	49.0	$5 \times 10^{+01}$	36	67.8	$7 \times 10^{+01}$
4×10^{-3}	30	85.9	$9 \times 10^{+01}$	50	98.6	$1 \times 10^{+02}$

The gains obtained (in error rate) may be substantial (up to an order of magnitude) for long codes, as can be seen from the tables above.

6.3 OPTIMUM THRESHOLD FOR FEEDBACK DECODING

The calculation of the optimum threshold for DD was not a very easy task. The problem becomes even more difficult for the case of FD. The reason is that while under DD there is (usually) only one K , in the case of FD there are J different (in general) K_s , for each of the k error bits $e_h^{(1)}$.

As a consequence, $P(\Sigma = \mu | e_h^{(1)} = 0)$ is proportional to the sum of all possible distinct products of μ K_s [see eqn (6.18)]. There are $C(J, \mu)$ such products and their sum may be replaced by the arithmetic mean of all the products, multiplied by $C(J, \mu)$. This arithmetic mean, a product of μ 'average' K_s may be replaced by the geometric mean of these K_s , raised to power μ . As a consequence, the concept of the μ th generalized mean takes shape.

6.3.1. The μ th Generalized Mean

Consider J positive real numbers K_1, K_2, \dots, K_J . To form their μ th generalized mean A_μ , where $\mu=1, 2, \dots, J$, the arithmetic mean of all distinct products of μ K_i s must firstly be formed. This may be expressed by:

$$L_{y(1)} L_{y(2)} \cdots L_{y(\mu)} \hat{=} \left[\sum_{x(1)=1}^{J-\mu+1} K_{x(1)} \sum_{x(2)=x(1)+1}^{J-\mu+2} K_{x(2)} \cdots \sum_{x(\mu)=x(\mu-1)+1}^J K_{x(\mu)} \right] / \binom{J}{\mu}$$

The geometric mean of the μ $L_{y(i)}$ s is the μ th root of their product. Hence:

Definition 6.1: The μ th generalized mean, of the J positive real numbers K_1, K_2, \dots, K_J , is denoted by A_μ and defined by:

$$A_\mu \hat{=} \left\{ \left[\sum_{x(1)=1}^{J-\mu+1} K_{x(1)} \sum_{x(2)=x(1)+1}^{J-\mu+2} K_{x(2)} \cdots \sum_{x(\mu)=x(\mu-1)+1}^J K_{x(\mu)} \right] / \binom{J}{\mu} \right\}^{1/\mu} \quad / 1 \leq \mu \leq J \quad (6.39)$$

In the rest of this paragraph, some properties of the generalized means will be discussed.

Theorem 6.9: Consider J positive real numbers K_1, K_2, \dots, K_J . Their arithmetic and their geometric mean is their 1st and J th generalized mean, respectively.

$$A_1 = (K_1 + K_2 + \cdots + K_J) / J \quad (6.40a)$$

$$A_J = (K_1 K_2 \cdots K_J)^{1/J} \quad (6.40b)$$

$$\text{Furthermore:} \quad A_\mu > A_J \quad \text{for all } \mu < J \quad (6.40c)$$

Proof: See Appendix 6.5 (§ A6.5.1., p. 428).

Theorem 6.10: Consider J positive numbers K_1, K_2, \dots, K_J . Then, if not all K_i are equal:

$$\text{If } K_i \leq 1: \quad (A_{\mu-1})^{\mu-1} > (A_\mu)^\mu \quad \text{for } \mu=2, 3, \dots, J \quad (6.41a)$$

$$\text{If } K_i \geq 1: \quad (A_{\mu-1})^{\mu-1} < (A_\mu)^\mu \quad \text{for } \mu=2, 3, \dots, J \quad (6.41b)$$

Proof: See Appendix 6.5 (§ A6.5.2., p. 428). ■

Unfortunately, the most important relation among the generalized means was impossible to prove. It is a very tight inequality and all known inequalities that could help in this matter were not tight enough.

Conjecture: Consider J positive numbers K_1, K_2, \dots, K_J , not all equal. Then:

$$A_{\mu-1} > A_{\mu} \quad \text{for } 1 < \mu \leq J \quad (6.42)$$

Discussion: Note first that if $K_i = K$ $/i=1,2,\dots,J$, then $A_{\mu} = K$ $/\mu=1,2,\dots,J$. If at least one of the K_i s is different then (6.42) holds true. The difference $A_{\mu-1} - A_{\mu}$ may be very small, if the K_i s are very close to each other. As an example consider the case $K_1 = K_2 = \dots = K_{15} = 4$ & $K_{16} = 4.1$; the results are arranged in TABLE 6.9, below.

TABLE 6.9

μ	A_{μ}	μ	A_{μ}	μ	A_{μ}
1	4.006250	7	4.006221	12	4.006197
2	4.006245	8	4.006216	13	4.006192
3	4.006240	9	4.006211	14	4.006187
4	4.006235	10	4.006206	15	4.006183
5	4.006231	11	4.006202	16	4.006178
6	4.006226				

Notice from TABLE 6.9 that A_{μ} decreases continuously as μ increases from 1 to $J=16$; furthermore, $A_{\mu-1}$ is consistently greater than A_{μ} , and what is more remarkable, by 5×10^{-6} (except for one or two cases, perhaps due to rounding errors).

It has been observed that the difference between $A_{\mu-1}$ & A_{μ} increases as the K_i s become more different from each other. Consider, for example the following three cases:

Case 1: $K_1=1$ $K_2=2$ $K_3=3$ $K_4=4$ $K_5=5$ $K_6=6$ ($\sigma = 1.71$)

Case 2: $K_1=1$ $K_2=3$ $K_3=5$ $K_4=7$ $K_5=9$ $K_6=11$ ($\sigma = 3.42$)

Case 3: $K_1=1$ $K_2=4$ $K_3=7$ $K_4=10$ $K_5=13$ $K_6=16$ ($\sigma = 5.12$)

TABLE 6.10

	Case 1	Case 2	Case 3
μ	A_μ	A_μ	A_μ
1	3.50	6.00	8.50
2	3.42	5.80	8.19
3	3.32	5.58	7.83
4	3.23	5.33	7.42
5	3.12	5.04	6.92
6	2.99	4.67	6.23

Consider also the effect of K_1 s less than 1:

Case 1: $K_1=0.01$ $K_2=0.1$ $K_3=4$ $K_4=22$

Case 2: $K_1=0.0001$ $K_2=0.001$ $K_3=0.01$ $K_4=0.1$

TABLE 6.11

	Case 1	Case 2
μ	A_μ	A_μ
1	6.53	0.0278
2	3.89	0.0137
3	1.34	0.0065
4	0.54	0.0032

Many more tests were carried, via a computer, trying to disprove the above conjecture. The results verified the author's belief that the conjecture is correct.

A computer was used to examine why all known inequalities could not assist in the proof of the conjecture. The conclusion was that they were not tight enough. In the process of trying to prove the above conjecture, many partial results were obtained, which nevertheless will not be discussed here; although they may contribute towards an eventual proof, their value is limited to just this.

6.3.2. Optimum Threshold

Consider firstly an expression for the conditional probability $P(\Sigma=\mu | e_n^{(1)}=0)$, in terms of the generalized means of

the quantities $K_i = P_i/(1-P_i)$, where $P_i \triangleq [1-(1-2p)^{c_i}]$ ($i=1,2,\dots,J$). Let A_μ denote the μ th generalized mean of the K_i s. Then the sum of all the distinct products of μ K_i s is simply $(A_\mu)^\mu C(J,\mu)$ (see Definition 6.1). From eqn (6.18), if

$$Q(J) \triangleq Q_1 Q_2 \cdots Q_J \quad (6.43)$$

$$P(\Sigma=\mu) \triangleq P(\Sigma=\mu | e_h^{(1)}=0) = Q(J) (A_\mu)^\mu \binom{J}{\mu} \quad / 0 \leq \mu \leq J \quad (6.44)$$

Note that A_μ is not defined for values of μ outside the interval $[1,J]$, but in $P(\Sigma=\mu | e_h^{(1)}=0)$ μ is defined in $[0,J]$. From eqn (6.18), $P(\Sigma=0 | e_h^{(1)}=0) = Q(J)$. So:

$$A_0 \triangleq 1 \quad (6.45)$$

Following eqn (6.44), an expression for $P_d(T) - P_d(T-1)$ will be developed. This will in turn be used to derive the optimum threshold, as the one that minimizes the probability of error, P_d .

Appendix 6.6 contains some intermediate results that lead to the proof of the theorem below.

Theorem 6.11: Let J syndrome bits, with sizes c_i $/i=1,2,\dots,J$, check on error bit $e_h^{(a)}$ and K_i be defined by eqn (6.16). If A_μ $/\mu=1,2,\dots,J$ denotes the μ th generalized mean of the K_i s, with $A_0=1$ and p the BSC's error probability, then the optimum threshold, T_0 , for FD of $e_h^{(a)}$ is at least equal to $\lceil J/2 \rceil$ and satisfies the following relation:

$$T_0 = \lfloor \{ \ln[p/(1-p)] + J \ln A_{J-T_0} \} / \ln(A_{J-T_0} A_{T_0}) \rfloor \quad (6.46a)$$

$$\text{with} \quad T_0 = \text{MIN}\{J, T_0\} \quad (6.46b)$$

Alternatively, T_0 is that integer $T \in [\lceil J/2 \rceil, J]$ which minimizes:

$$\left| T - \{ \ln[p/(1-p)] + J \ln A_{J-T} \} / \ln(A_{J-T} A_T) \right| \quad (6.46c)$$

Proof: See Appendix 6.6 (p. 431). ■

The result of Theorem 6.11 is very important, but unable to provide a closed-form expression for the optimum threshold under FD. Nevertheless, the RHS of (6.46a) is a weak function of T_0 , because the two generalized means, that are

functions of T_0 , can be approximated by other generalized means, independent of T_0 ; this will not cause a serious approximation error because the generalized means are extremely close to each other.

6.3.3. Bounds & Approximation for the Optimum Threshold

In this paragraph, an approximate solution of eqns (6.46) will be attempted. Note that there are two functions of T_0 in the RHS of eqn (6.46), A_{T_0} and A_{J-T_0} . From Theorem 6.11, T_0 is at least $\lceil J/2 \rceil$. Then:

$$1 \leq J - T_0 \leq J/2 \leq \lceil J/2 \rceil \triangleq T_n \leq T_0 \leq J \quad (6.47a)$$

From reln (6.47) and the Conjecture of Paragraph 6.3.1.:

$$A_1 \geq A_{J-T_0} \geq A_{T_n} \geq A_{T_0} \geq A_J \quad (6.47b)$$

Since the logarithm is an increasing function of its argument:

$$\ln A_1 \geq \ln A_{J-T_0} \geq \ln A_{T_n} \geq \ln A_{T_0} \geq \ln A_J \quad (6.47c)$$

Consider again eqn (6.46a), in relation to the inequalities of (6.47c):

$$T_0 = \lfloor \{ \ln[p/(1-p)] + J \ln A_{J-T_0} \} / \ln(A_{J-T_0} A_{T_0}) \rfloor \quad (6.46a)$$

To create an upper bound on T_0 replace the 1st $\ln A_{J-T_0}$ with something larger ($\ln A_1$) and the 2nd $\ln A_{J-T_0}$ & $\ln A_{T_0}$ with something smaller ($\ln A_{T_n}$ & $\ln A_J$). Hence:

$$T_0 \leq \lfloor \{ \ln[p/(1-p)] + J \ln A_1 \} / \ln(A_{T_n} A_J) \rfloor \quad (6.48a)$$

Similarly, for a lower bound:

$$T_0 \geq \lfloor \{ \ln[p/(1-p)] + J \ln A_{T_n} \} / \ln(A_1 A_{T_n}) \rfloor \quad (6.48b)$$

It is understood that if the bounds exceed J , they are reduced to J .

To obtain an approximate solution for T_0 , consider first the product $A_{J-T_0} A_{T_0}$. Note, from TABLES 6.9, 6.10 & 6.11, that $A_\mu - A_{\mu+1} \approx \text{constant} \triangleq \delta$. Hence $A_\mu - A_{\mu+d} \approx d\delta$, and then:

$$A_{\mu-d} A_{\mu+d} = (A_\mu + d\delta)(A_\mu - d\delta) = (A_\mu)^2 - (d\delta)^2 < (A_\mu)^2 \quad (6.49a)$$

$$\longrightarrow A_{\mu-d} A_{\mu+d} \approx (A_\mu)^2 \quad (6.49b)$$

The above approximation is usually good, because $(d\delta)^2 \ll (A_p)^2$. Note though that the product in the LHS of (6.49b) is overestimated.

Since $K_1 < 1$, their generalized means are also < 1 , hence their logarithm is negative. To draw the right conclusions one may multiply both numerator & denominator of eqn (6.46a) by -1 and let the logs 'absorb' the minus sign:

$$T_o = \lfloor \{ \ln[(1-p)/p] + J \ln(1/A_{J-T_o}) \} / \ln[1/(A_{J-T_o} A_{T_o})] \rfloor \quad (6.50)$$

From approximation (6.49a):

$$\ln[1/(A_{J-T_o} A_{T_o})] \gtrsim 2 \ln(1/A_{T_n}) \quad (6.51)$$

Hence, if (6.51) is used in eqn (6.50), T_o will be overestimated. To compensate, $\ln(1/A_{J-T_o})$, which appears in the numerator, should be underestimated. From (6.47c):

$$\ln(1/A_{J-T_o}) \gtrsim \ln(1/A_1) \quad (6.52)$$

From the last three expressions:

$$T_o \approx \lfloor \{ \ln[p/(1-p)] + J \ln A_1 \} / (2 \ln A_{T_n}) \rfloor \quad (6.53a)$$

The approximation, above, may be rewritten:

$$T_o \approx \lfloor (J/2)(\ln A_1 / \ln A_{T_n}) + \ln[p/(1-p)] / (2 \ln A_{T_n}) \rfloor \quad (6.53b)$$

Note the similarity between the above expression and eqn (6.34a) (for the optimum threshold under DD). This suggests that a 2nd approximation may be attempted.

Consider again inequalities (6.47c):

$$\ln A_{J-T_o} \gtrsim \ln A_{T_n} \gtrsim \ln A_{T_o}$$

Then, from (6.46a):

$$T_o \approx \lfloor (J/2) + \ln[p/(1-p)] / (2 \ln A_{T_n}) \rfloor \quad (6.53c)$$

Comparing (6.53b) with (6.53c), it is obvious that the latter is, in general, an overestimation of T_o , since, from (6.47c), $\ln A_1 \geq \ln A_{T_n} \iff \ln A_1 / \ln A_{T_n} \leq 1$ ($\ln A < 0$).

What remains to be done is to test the validity of the above results. Four cases will be considered, two $J=11$ and two $J=10$ cases. The results are displayed below:

Example 6.4: Consider Case 1: $c_1=4$, $c_2=10$, $c_3=15$, $c_4=20$, $c_5=30$, $c_6=40$, $c_7=60$, $c_8=80$, $c_9=100$, $c_{10}=120$, $c_{11}=150$.

TABLE 6.12: Case 1 * $\{n_x = 630\}$

P	LB	T_0	T_1	T_2	UB
0.200	11	11	11	11	11
0.100	11	11	11	11	11
0.070	11	11	11	11	11
0.050	10	10	11	11	11
0.020	7	8	8	9	9
0.010	6	7	7	7	8
0.007	6	7	6	7	8
0.005	6	7	6	7	7
0.002	6	6	6	6	7
0.001	6	6	6	6	6

Example 6.5: Consider Case 2: $c_1=40$, $c_2=45$, $c_3=50$, $c_4=60$, $c_5=70$, $c_6=80$, $c_7=90$, $c_8=100$, $c_9=110$, $c_{10}=120$, $c_{11}=150$.

TABLE 6.13: Case 2 * $\{n_x = 916\}$

P	LB	T_0	T_1	T_2	UB
0.200	11	11	11	11	11
0.100	11	11	11	11	11
0.070	11	11	11	11	11
0.050	11	11	11	11	11
0.020	11	11	11	11	11
0.010	10	10	10	10	10
0.007	8	8	8	9	9
0.005	7	8	8	8	8
0.002	6	7	7	7	7
0.001	6	6	6	6	6

Example 6.6: Consider Case 3: $c_1=4$, $c_2=10$, $c_3=20$, $c_4=30$, $c_5=40$, $c_6=60$, $c_7=80$, $c_8=100$, $c_9=120$, $c_{10}=150$.

See overleaf for the table.

* LB=Lower Bound, T_0 =Optimum Threshold, T_1 =ith approximation of T_0 , UB=Upper Bound.

TABLE 6.14: Case 3 * $\{n_2 = 615\}$

P	LB	T_0	T_1	T_2	UB
0.200	10	10	10	10	10
0.100	10	10	10	10	10
0.070	10	10	10	10	10
0.050	10	10	10	10	10
0.020	7	8	8	9	9
0.010	6	7	7	7	8
0.007	5	6	6	7	7
0.005	5	6	6	6	7
0.002	5	6	6	6	6
0.001	5	6	5	6	6

Example 6.7: Consider Case 4: $c_1=40$, $c_2=50$, $c_3=60$, $c_4=70$, $c_5=80$, $c_6=90$, $c_7=100$, $c_8=110$, $c_9=120$, $c_{10}=150$.

TABLE 6.15: Case 4 * $\{n_2 = 871\}$

P	LB	T_0	T_1	T_2	UB
0.200	10	10	10	10	10
0.100	10	10	10	10	10
0.070	10	10	10	10	10
0.050	10	10	10	10	10
0.020	10	10	10	10	10
0.010	10	10	10	10	10
0.007	8	8	8	8	8
0.005	7	7	7	7	7
0.002	6	6	6	6	6
0.001	6	6	6	6	6

Consider now the results displayed in the four tables, above. The two approximate expressions for T_0 will be compared with the actual value of the optimum threshold. Also, the width $UB-LB$ will be examined, along with the agreement between the two approximate expressions for T_0 .

1st Approximation: Out of a total of forty bit-error probabilities examined there is a disagreement of 1, in four

* LB=Lower Bound, T_0 =Optimum Threshold, T_i =ith approximation of T_0 , UB=Upper Bound.

instances; for $p = 0.050, 0.007$ & 0.005 , in Case 1 and for $p=0.001$ in Case 3.

Width of UB-LB: Consider the difference between the UB & the LB for each of the four cases. The differences are arranged in order of descending bit-error rate p .

Case 1: 0 0 0 1 2 2 2 1 1 0 Case 2: 0 0 0 0 0 0 1 1 1 0

Case 3: 0 0 0 0 2 2 2 2 1 1 Case 4: 0 0 0 0 0 0 0 0 0 0

Difference between the two approximations: Consider the difference between the 2nd & the 1st approximations; the results are arranged in order of descending p .

Case 1: 0 0 0 0 1 0 1 1 0 0 Case 2: 0 0 0 0 0 0 1 0 0 0

Case 3: 0 0 0 0 1 0 1 0 0 1 Case 4: 0 0 0 0 0 0 0 0 0 0

Consider now some general conclusions about the above results.

For Case 4 there was absolute agreement between the two approximations and the true value of T_0 ; also, the two bounds coincided in all cases examined. From the other three cases, Case 2 behaved best; the 1st approximation was accurate while the 2nd failed only once, by one. This distinct behaviour is due to the smaller spread of the c_i s (and consequently of the K_i s), for Cases 2 & 4, compared to 1 & 3.

The difference between the UB and the LB never exceeded two and was zero in 25 of the 40 instances considered. Furthermore, all values of T_0 and their two approximations were inside the range [LB,UB].

The 2nd approximation of T_0 overestimated T_0 , but only in 7 out of the 40 instances considered and only by one. In the rest of the instances the two approximations of T_0 coincide.

6.3.4. Probability of Decoding Error

In this paragraph, expressions for, and approximations to, the probability of decoding error will be developed. What is remarkable, is the similarity with the corresponding results for DD (see Paragraph 6.2.3.).

Consider firstly an exact expression for $P_d(T_0)$. From Theorem 6.11, $T_0 \geq \lceil J/2 \rceil = \lfloor (J+1)/2 \rfloor > (J+1)/2 - 1 = (J-1)/2$. Hence, $T_0 > (J-1)/2$ and from eqn (6.23a) and eqn (6.44):

$$P_d(T_o) = Q(J) \left[\sum_{\mu=T_o+1}^J (A_\mu)^\mu \binom{J}{\mu} + p \sum_{\mu=J-T_o}^{T_o} (A_\mu)^\mu \binom{J}{\mu} \right] \quad (6.54)$$

Consider next the limit value of T_o , as $p \rightarrow 0$.

The μ th generalized mean, A_μ , is the μ th root of the sum of all the μ -products of the K_i s, divided by $C(J, \mu)$. If approximation $K_i \approx pc_i$ [see (6.28)] is used, each of the distinct $C(J, \mu)$ products of μ K_i s becomes a product of the corresponding μ c_i s multiplied by p^μ . Let C_μ denote the μ th generalized mean of the J c_i s. Then:

$$\text{For } \mu=1, 2, \dots, J: \quad A_\mu \approx p C_\mu / pc_1 \ll 1 \quad (6.55)$$

$$\text{Obviously, as } p \rightarrow 0, \ln[p/(1-p)] \rightarrow \ln p \quad (6.56)$$

From the last two approximations and the 2nd approximation to T_o [(6.53c)]:

$$\lim_{p \rightarrow 0} \{T_o\} = \lfloor (J/2) + \ln p / [2 \ln(p C_{T_n})] \rfloor \quad (6.57)$$

The limit value of T_o under DD is $\lfloor (J/2) + \ln p / [2 \ln(p c)] \rfloor$ [from (6.34a), with approximations (6.55) & (6.56)]. Hence, the role played by c under DD, is played by C_{T_n} under FD. Following the same reasoning as in the discussion leading to Theorem 6.6, one concludes that as $p \rightarrow 0$, T_o assumes eventually its minimum value, $\lceil J/2 \rceil$; in particular, that happens when p becomes (approximately) smaller than $1/(C_{T_n})^2$. Hence:

$$T_o = \lceil J/2 \rceil \triangleq T_n \quad \text{for } p < 1/(C_{T_n})^2 \quad (6.58)$$

Finally, two approximations will be attempted, on eqn (6.54), for small values of p . If p satisfies the condition in (6.58), then from (6.55) & (6.54), for J =even:

$$P_d(T_o) \approx Q(J) \left[\sum_{\mu=J/2+1}^J p^\mu (C_\mu)^\mu \binom{J}{\mu} + p^{J/2+1} (C_{J/2})^{J/2} \binom{J}{J/2} \right] \quad (6.59)$$

In the above approximate expression for P_d , powers of p multiply each term in the summation, in the RHS. Since p is very small, one may assume that only the terms corresponding to the minimum power of p , $(J/2+1)$, are significant. Hence:

$$P_d(T_o) \approx Q(J) p^{J/2+1} \left[\binom{J}{J/2+1} (C_{J/2+1})^{J/2+1} + \binom{J}{J/2} (C_{J/2})^{J/2} \right] \quad (6.60)$$

The above expression is an approximation and hence one can assume that $C_{J/2+1} \approx C_{J/2}$. Furthermore, from eqn (6.13), if $pc_1 \ll 1$, then $P_1 \approx pc_1 \ll 1 \implies Q_1 \approx 1$. Hence, $Q(J) \approx 1$. From the results above:

$$P_d(T_0) \approx p^{J/2+1} \binom{J}{J/2} (C_{J/2})^{J/2} [C_{J/2}/(1+2/J)+1] \quad (6.61)$$

6.4 CONCLUSIONS

In this chapter, one aspect of *majority-logic decoding* for CSOCs, was examined. Specifically, while Massey set the *threshold* at $\lceil J/2 \rceil$, it was proved, by the author, that this is the optimum* setting only if the bit error rate, p , is smaller than a certain value.

An expression for $P_d(T)$, the probability of decoding a bit in error, was derived, as a function of the threshold, T [eqn (6.1)]. Given this eqn, it seemed natural to ask, what is the value of T that minimizes $P_d(T)$? The sign of the difference $P_d(T) - P_d(T-1)$ was examined, as T ranges from 0 to J . If the difference is negative for $T < T_0$, and positive for $T > T_0$, then T_0 is the *optimum threshold* (see Theorem 6.1).

$P_d(T)$ is a function of the probabilities, P_1 , that the various syndromes, checking on the bit to be decoded, are 1. In § 6.1.2., it was proved that $P_1 = \frac{1}{2}[1-(1-2p)^{c_1}] \approx pc_1$, where c_1 is the syndrome size. In § 6.1.3. & § 6.1.4. general expressions were obtained for $P(\Sigma=\mu|e_\mu)$, the probability that the sum of the syndromes checking on e_μ is μ and for $P_d(T)$. Example cases were considered which showed that there is, indeed, an optimum threshold, which is $\lceil J/2 \rceil$ for small values of p , but which increases as p increases.

The case of *constant-size syndromes* was considered, at first (this is, usually, the case of DD). The probability distribution $P(\Sigma=\mu|e_\mu)$ was studied and it was proved that it peaks for $\mu \approx P(J+1)$. If $P(J+1) < 1$ (which is usually the case) $P(\Sigma=\mu|e_\mu)$ decreases as μ increases (see Theorem 6.4).

Eqn $P_d(T) - P_d(T-1) = 0$ was solved for T , and the optimum threshold was found to be $\lceil J/2 + \frac{1}{2} \ln[p/(1-p)] / \ln K \rceil$, where $K \hat{=} P/(1-P)$. The relation between T_0 and p & c was studied and

* 'Optimum' means minimum probability of decoding error.

it was proved that T_0 increases with p , or c , while it tends to its nominal value, $T_n \hat{=} \lceil J/2 \rceil$, as $p \rightarrow 0$. It was also proved that T_0 'departs' from T_n from about $p=1/c^2$. Example cases showed that the error rate is improved by about one order of magnitude, for long codes.

Finally the case of FD was considered. This is very complicated because $P(\Sigma=p|e_n)$ is proportional to the sum of all combinations of the P_i s of the syndromes that check on the error bit to be decoded. It was thought, though, that there must be an 'average' syndrome size, which could play the role played by c , under DD. The concept of the μ th *generalized mean*, A_μ , of J positive numbers ($1 \leq \mu \leq J$) was introduced (an original idea), to assist in these calculations (see § 6.3.1.). This was a successful generalization, since it was proved that A_1 is the arithmetic mean and A_J the geometric mean, and also that $A_\mu > A_J / \mu < J$. It was further observed (but it was impossible to prove), that $A_{\mu-1} > A_\mu$ for all $\mu=2,3,\dots,J$. This conjecture was tested and was found to be true, in all cases. It was also observed that the generalized means are very close to each other. All expressions involving T_0 & $P_d(T)$, use some generalized means*.

A non-closed-form expression for the optimum threshold, under FD, was derived in Theorem 6.11. The inability to obtain a 'useful' expression for T_0 is due to the existence of syndromes of various sizes. Tight upper & lower bounds on T_0 were obtained [see relns (6.48)] as well as two good approximations, [see relns (6.3)]. The 2nd approximation is very similar to the corresponding approximation for DD. Their only difference is that FD uses C_{Tn} , instead of c , where C_μ is the μ th generalized mean of the sizes of the syndromes checking the error bit. It was also argued that T_0 'departs' from T_n from about $p=1/(C_{Tn})^2$. Four example cases were considered which proved the validity and the tightness of the bounds and the approximations. This result then suggests that the way to obtain the optimum threshold, under FD, is to use the bounds to restrict the range of T_0 (to usually one or two values), and then to employ the exact expression [(6.46c)] to determine which value minimizes T_0 .

* Either of the quantities $K_i \hat{=} P_i/(1-P_i)$, or of the syndrome sizes.

CHAPTER 7

Structure of 'Cyclic' CSOCs

Chapter 7 will introduce the class of 'cyclic' convolutional self-orthogonal codes. This class of systematic codes was proposed by McQuilton [42], in 1979; the term 'cyclic' refers to the fact that these codes can be decoded cyclically, using $k/(n-k)$ threshold gates (instead of k gates). Another important characteristic of this class of codes is that it has an infinite number of members including codes that are multiple error-correcting at extremely high rates.

'Cyclic' CSOCs are based on number theory and (as a consequence) they have a 'rich' mathematical structure. The latter may be exploited to achieve a number of results, including expressions for the code performance and ways to systematically alter the initial design in order to achieve specific goals. As an example, a method to shorten the codes was proposed by McQuilton [43], in 1980.

The author would like to admit that he was 'intrigued' by number theory. In particular, this theory seems to have a very rich structure which remains unexploited, and on many occasions unknown.

Although the class of 'cyclic' CSOCs was discovered by McQuilton, the work in this chapter is largely original. Its primary aim is to show that the above class of codes is the general solution of the problem of, systematically, constructing systematic CSOCs under a small set of specific constraints; the latter are introduced in an effort to facilitate the solution. A second aim is to generalize McQuilton's results, in the hope that other classes of similar

codes may be discovered, by other researchers.

To that end, an alternative representation of systematic convolutional codes is firstly considered. Specifically, an array of integers is used to generate the code. The problem which is addressed is the determination of the necessary and sufficient conditions so that this array generates self-orthogonal convolutional codes. The first 'arbitrary' restriction is imposed and the previous results are accordingly refined, up to the point where the effort exceeds the limits of the thesis. This process is repeated until McQuilton's work is duplicated.

7.1 AN ALTERNATIVE REPRESENTATION OF CONVOLUTIONAL CODES

Convolutional codes for threshold decoding were discussed in Chapter 5. It was explained there that in a CSOC the set of syndrome bits that check on error bit $e_0^{(i)}$ $/1 \leq i \leq k$ should be orthogonal on that error bit. Hence, a systematic way of constructing CSOCs is equivalent to a systematic way of constructing the syndrome equations, so that the orthogonality principle is satisfied.

7.1.1. A Discussion on the Design Approach

Consider eqn (5.7) (p. 138). Note that error bits $e_0^{(i)}$ $/i=1,2,\dots,k$ are checked by syndrome bits $s_h^{(j)}$ $/j=1,2,\dots,n-k$ & $h=0,1,\dots,m$. Hence, the syndrome bits that check on error bits $e_0^{(i)}$ $/i=1,2,\dots,k$ are:

$$s_h^{(j)} = e_h^{(k+j)} + \sum_{z=0}^h \sum_{i=1}^k e_{h-z}^{(i)} g_{k+j,z}^{(i)} \quad /1 \leq j \leq n-k \text{ \& } 0 \leq h \leq m \quad (7.1)$$

According to Theorem 5.6, $s_h^{(j)}$ checks on $e_0^{(i)}$ iff $g_{k+j,h}^{(i)} = 1$. Ideally, one would like to determine the necessary & sufficient conditions on the 'g-coefficients' so that the corresponding systematic convolutional code is self-orthogonal. Given that such a result is at least very difficult, and possibly impossible, to obtain, one would try to simplify the problem. This may be done either by following a differ-

ent approach, or by attacking a partial case.

In this section the work will start by following a different approach and narrowing the problem a little bit. Specifically, an array of integers will be used to completely specify the parity-check generation for systematic convolutional codes. The array will have $n-k$ rows, one for each parity-check. The integers along each row will specify the message bits from each block that participate in the formation of the corresponding parity-check. Since there are $m+1$ blocks that participate in the formation of the current one, the array will have $m+1$ columns. Each entry (cell) of the array (located by the column and row numbers) contains an unspecified number of integers. The above idea gives rise to a one-to-one correspondence between a systematic convolutional code and its associated array of integers, to be called the *initial array*.

So, the initial array (IA) for an (n,k,m) systematic convolutional code will be made of $(n-k) \times (m+1)$ cells. Row j ($1 \leq j \leq n-k$) of cells defines the way the j th parity-check is formed. Column i ($1 \leq i \leq m+1$) of cells provides the contributions from the $(h-i+1)$ th message block, where h th is the block currently encoded. Cell (j,i) may contain up to k entries (all positive integers) or it may be empty. Entry, say, x indicates that the x th bit ($1 \leq x \leq k$) of the $(h-i+1)$ th message-block participates in the formation of the j th parity-check of the h th channel-block, i.e. bit $v_h^{(k+j)}$ depends on bit $u_{h-i+1}^{(x)}$.

Because of the similarity between the parity-check and the syndrome equations [compare eqn (2.47b) with eqns (2.76)], the IA may be used to study the syndrome equations, and in particular the set of syndromes which check on each of the k error bits $e_h^{(1)}, e_h^{(2)}, \dots, e_h^{(k)}$. This may be done as following: Assume that one wishes to collect all the syndromes that check on error bit $e_h^{(x)}$ / $1 \leq x \leq k$. The IA must be scanned to determine all entries equal to x . Assume that such an entry is found in cell (j,i) / $1 \leq j \leq n-k$ & $1 \leq i \leq m+1$; according to the above discussion, syndrome bit $s_f^{(j)}$ checks on error bit $e_{f-i+1}^{(x)}$, hence syndrome bit $s_{h+1-i}^{(j)}$ checks on error bit $e_h^{(x)}$.

Since a three-dimensional array is difficult to work with, a partial case will be considered. A possible family of CSOCs could specify up to one bit from the current and each of the past m blocks; in such a case, 0s along the row would indicate no contribution from the corresponding block. Another construction could specify exactly two (or maybe up to two) bits from each block, etc. These two families may be described by two-dimensional arrays.

7.1.2. Introduction to Type-A Codes

The class of 'cyclic' CSOCs has been given a distinct property: From the current and each of the past m message error blocks, only one bit participates in the formation of each of the current syndrome bits. Accordingly, a family of systematic convolutional codes, described by a two-dimensional IA, is defined below:

Definition 7.1: An (n,k,m) type-A code is a systematic convolutional code satisfying

$$w[g_{k+j,z}^{(1)} g_{k+j,z}^{(2)} \cdots g_{k+j,z}^{(k)}] = 1 \quad (7.2)$$

for all $j=1,2,\dots,n-k$ & $z=0,1,\dots,m$. Equivalently, from each of the message error blocks $(e^m)_h, (e^m)_{h-1}, \dots, (e^m)_{h-m}$, exactly one bit participates in the formation of each of the current syndrome bits, $s_h^{(1)}, s_h^{(2)}, \dots, s_h^{(n-k)}$.

From (7.2), it is obvious that for each $g_{k+j,z}^{(i)}$ $/i=1,2,\dots,k$, exactly one of them is 1; hence, for each value of j ($1 \leq j \leq n-k$) and each value of z ($0 \leq z \leq m$) there exists a value of i , a function of j & z , for which $g_{k+j,z}^{(i)} = 1$.

$$\text{For } 1 \leq j \leq n-k \text{ \& } 0 \leq z \leq m: \quad g_{k+j,z}^{(i)} = \begin{cases} 1 & \text{if } i = a_{j,z+1} \\ 0 & \text{if } i \neq a_{j,z+1} \end{cases} \quad (7.3)$$

Notation: The a -coefficients of the initial array will be denoted either by $a[j,z]$, or $a_{j,z}$.

From the discussion above, it is obvious that each cell

of the IA has exactly one entry, hence the IA is a two-dimensional array of positive integers; furthermore, $a_{j,z}$ indicates the bit, from message error block $(e^m)_{h-z+1}$, which participates in the formation of $s_h^{(j)}$ / $j=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$. It is also obvious that $1 \leq a_{j,z} \leq k$ for all j & z . Using result (7.3) in eqn (7.1):

$$s_h^{(j)} = \sum_{z=0}^h e_{h-z}^{(a_{j,z+1})} + e_h^{(k+j)} \quad / 0 \leq h \leq m \text{ \& } 1 \leq j \leq n-k \quad (7.4)$$

The initial array (IA) for an (n,k,m) type-A code has the following format:

Determines contributions from block: (h is the current block)

Defines:

	h	h-1	...	h-z+1	...	h-m
$s_h^{(1)}$ —>	$a_{1,1}$	$a_{1,2}$...	$a_{1,z}$...	$a_{1,m+1}$
$s_h^{(2)}$ —>	$a_{2,1}$	$a_{2,2}$...	$a_{2,z}$...	$a_{2,m+1}$

$s_h^{(x)}$ —>	$a_{x,1}$	$a_{x,2}$...	$a_{x,z}$...	$a_{x,m+1}$

$s_h^{(n-k)}$ —>	$a_{n-k,1}$	$a_{n-k,2}$...	$a_{n-k,z}$...	$a_{n-k,m+1}$

Note 7.1: Consider an (n,k,m) type-A code. The initial array (IA), corresponding to this code, is the array of elements $a_{x,z}$, where $x=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$. The general element $a_{x,z}$ indicates the error bit, from message error block $(e^m)_{h-z+1}$, which participates in the formation of syndrome bit $s_h^{(x)}$.

7.1.3. Introduction to Decoding of Type-A Codes

To decode error bit $e_0^{(1)}$ / $1 \leq i \leq k$ one should collect all syndrome bits that check on this error bit. These syndrome bits may be determined from the IA and eqn (7.4), which is repeated below in a slightly different form:

$$s_h^{(j)} = \sum_{z=1}^{h+1} e_{h+1-z}^{(a_{j,z})} + e_h^{(k+j)} \quad / 0 \leq h \leq m \text{ \& } 1 \leq j \leq n-k \quad (7.5)$$

One should scan the IA to locate entries equal to i . If $a_{x,w}=i$, i.e. if an i exists in row x , column w , from (7.5), $j=x$ & $z=w$; from the latter and $h+1-z=0$, it follows that $h=w-1$. So, syndrome bit $s_{w-1}^{(x)}$ checks on $e_0^{(i)}$.

Assume that syndrome bit $s_{w-1}^{(x)}$ checks on error bit $e_0^{(i)}$. Then from eqn (7.5),

$$s_{w-1}^{(x)} = \sum_{z=1}^w e_{w-z}^{(a_{x,z})} + e_{w-1}^{(k+x)} \quad /1 \leq w \leq m+1 \quad (7.6)$$

Because $s_{w-1}^{(x)}$ checks on $e_0^{(i)}$, the $z=w$ term in the above summation will be the bit from message error block $(e^m)_0$; since exactly one bit from each error block participates in the formation of each syndrome bit (see Definition 7.1), this bit will be the i th one of that block, hence $a_{x,w}=i$.

The following theorem has been proved:

Theorem 7.1: Let $a_{x,z}$ $/x=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$ be the elements of the IA of an (n,k,m) type-A code. Syndrome bit $s_{w-1}^{(x)}$ $/1 \leq x \leq n-k$ & $1 \leq w \leq m+1$ checks on error bit $e_0^{(i)}$ $/1 \leq i \leq k$ if, and only if, $a_{x,w}=i$. ■

The question that arises, at this point, is "how to construct the IA so that the corresponding systematic convolutional code is self-orthogonal?".

7.1.4. Orthogonality Conditions for Type-A Codes

Let us consider now the equivalent condition under which the type-A code generated by the IA is not self-orthogonal.

Theorem 7.2: Let $a_{x,z}$ $/x=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$ be the elements of the initial array of an (n,k,m) type-A code. A necessary and sufficient condition for this code not to be self-orthogonal is that the same pair of integers, separated by the same number of places, appears along two of the rows of the IA. Stated otherwise, this condition requires that, for any two elements $a_{x,u}$ & $a_{v,w}$ of the IA which are equal, there exists at least one positive integer c , less than u &

w, such that $a_{r,u-c} = a_{v,w-c}$:

$$\text{If } a_{r,u} = a_{v,w}, \text{ there exists } c: a_{r,u-c} = a_{v,w-c} \quad (7.7a)$$

$$\text{where } 1 \leq r, v \leq n-k \quad \& \quad 1 \leq u, w \leq m+1 \quad \& \quad 0 < c < \min\{u, w\} \quad (7.7b)$$

Proof: See Appendix 7.4 (p. 447). ■

Note that in the above theorem it is not necessary to have $r \neq v$, i.e. to consider two distinct rows. Hence, even if the same number appears in the same row, more than once, the code will not be self-orthogonal if (7.7) do hold true.

7.2 TYPE-B CODES - THE USE OF NUMBER THEORY

Although Theorem 7.2 sets the conditions for orthogonality, it does not do so in a way that is practically useful. It may be possible to persist and be rewarded, but it is tempting to simplify the problem by introducing a new restriction. This time it concerns the way the elements of the IA are generated.

Definition 7.2: Let $a_{x,z}$ $/x=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$ be the elements of the initial array of an (n,k,m) type-A code. Then the IA corresponding to a type-B code is generated as following:

$$a_{x,z} \equiv za_{x,1} \pmod{k+1} \quad /x=1,2,\dots,n-k \quad \& \quad z=1,2,\dots,m+1 \quad (7.8) \quad \blacksquare$$

With the above definition, the construction and the properties of the IA come under the 'jurisdiction' of the theory of congruences*, and in general of number theory. Note that a special symbol (\equiv) was introduced; it was chosen by Gauss to suggest analogy with the equals sign ($=$).

Furthermore, the problem of the construction of the IA has been reduced to that of selecting its 1st column, so that the corresponding type-B code is self-orthogonal.

* See Appendix 7.2 for a brief introduction to congruences.

7.2.1. Restriction on the Value of m for Type-B Codes

Theorem 7.3: Let $a_{x,1}$ / $x=1,2,\dots,n-k$ be the elements of the 1st column of the initial array of an (n,k,m) type-B code (obviously, $1 \leq a_{x,1} \leq k$, / $x=1,2,\dots,n-k$). Then, if row x is not to contain a zero, its length should not exceed $(k+1)/(k+1, a_{x,1}) - 1$,* and consequently m is restricted by:

$$m \leq \min_{x=1}^{n-k} \left\{ (k+1)/(k+1, a_{x,1}) \right\} - 2 \quad (7.9)$$

Proof: See Appendix 7.5 (§ A7.5.1., p. 448)

Note that, according to (7.9), the maximum value of m is $k-1$. This is achieved if all elements of the first column are relatively prime to $k+1$.

Consider also the other extreme, namely the minimum of the maximum value of m , given by the following theorem.

Theorem 7.4: For the initial array of an (n,k,m) type-B code to exist, it is necessary that the maximum value of m , m_{\max} , satisfies

$$m_{\max} \geq p-2 \quad (7.10)$$

where p is the smallest prime factor of $k+1$. For a $(2k,k,m)$ type-B code to be self-orthogonal, it is necessary that $m_{\max} = p-2$.

Proof: See Appendix 7.5 (§ A7.5.2., p. 449).

7.2.2. Orthogonality Conditions for Type-B Codes

Lemma 7.1: Consider an (n,k,m) initial array with elements $a_{x,z}$, where $x=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$. Then, for any $a_{x,z}$ and for any c , nonnegative:

$$a_{x,z+c} - a_{x,z} \equiv ca_{x,1} \pmod{k+1} \quad (7.11)$$

Proof: See Appendix 7.6 (§ A7.6.1., p. 450).

Consider again Theorem 7.2; in effect, it gives a neces-

* (a,b) denotes the greatest common divisor of a & b .

sary & sufficient condition for an (n,k,m) type-B code not to be self-orthogonal: The code is not self-orthogonal if, and only if, there is at least one pair of elements $a_{r,u} = a_{v,w}$ and at least one integer c , such that $a_{r,u-c} = a_{v,w-c}$, where $0 < c < \min\{u,w\}$. The next theorem re-defines the orthogonality conditions, set by Theorem 7.2, exploiting the IA generation rule introduced by Definition 7.2.

Theorem 7.5: Consider an (n,k,m) type-B code and its initial array with elements $a_{x,z}$, where $x=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$ and m satisfying condition (7.9). The code is not self-orthogonal if, and only if, there are two equal elements in two rows, say elements $a_{r,u} = a_{v,w}$, where $1 < u,w \leq m+1$, and at least one positive integer c , less than u & w , such that:

$$c(a_{r,1} - a_{v,1}) \equiv 0 \pmod{k+1} \quad (7.12)$$

Proof: See Appendix 7.6 (§ A7.6.2., p. 450). ■

The corollary to the above theorem will be its logical negation:

Corollary: Consider an (n,k,m) type-B code and its initial array with elements $a_{x,z}$, where $x=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$ and m satisfying condition (7.9). The code is self-orthogonal if, and only if, for any two rows, say r & v , and for all elements $a_{r,u} = a_{v,w}$ of these two rows that are equal, where $1 < u,w \leq m+1$, and for all positive integers c , less than u or w :

$$c(a_{r,1} - a_{v,1}) \not\equiv 0 \pmod{k+1} \quad (7.13a)$$

$$\longleftrightarrow a_{r,1} \not\equiv a_{v,1} \pmod{(k+1)/(k+1,c)} \quad (7.13b)$$

The 2nd condition follows from the 1st & Theorem A7.2.4. ■

7.2.3. A Discussion on the Design Approach

The above corollary, although useful and important, lacks in clarity and presentation due to the number of conditions which accompany relation (7.13). An attempt to simplify this

will follow the examples below:

Example 7.1: Consider an (n,k,m) type-B code and its IA. Let $k+1=8$ and consider two rows with elements $a_{1,1}=1$ and $a_{2,1}=3$. Then, from (7.9), both rows may have length $m+1 = k = 7$:

1	2	3	4	5	6	7
3	6	1	4	7	2	5

The columns with common elements (apart from the 1st one) are: 2 & 6 (element 2, or 6), 4 & 4 (element 4) and 5 & 7 (element 5, or 7). Then: For element 2 (or 6), $u=2$ & $w=5$, hence $c=1$. For element 4, $u=w=4$, hence $c=1,2,3$. For element 5 (or 7), $u=5$ & $w=7$, hence $c = 1,2,3,4$. According to Lemma 7.2, the first-column elements (1 & 3) must be incongruent $(\text{mod } (k+1)/(k+1,c))$ for $c=1,2,3,4$, i.e. incongruent modulo $8/(8,1)=8$, $8/(8,2)=4$, $8/(8,3)=8$ and $8/(8,4)=2$, i.e. incongruent modulo 8, 4 and 2. They are incongruent modulo 4 and 8, but not modulo 2.*

Corresponding to the above IA the following syndrome equations may be obtained [see eqn (7.4)]:

$$s_h^{(1)} = e_h^{(1)} + e_{h-1}^{(2)} + e_{h-2}^{(3)} + e_{h-3}^{(4)} + e_{h-4}^{(5)} + e_{h-5}^{(6)} + e_{h-6}^{(7)} + e_h^{(8)} \quad (A)$$

$$s_h^{(2)} = e_h^{(3)} + e_{h-1}^{(6)} + e_{h-2}^{(1)} + e_{h-3}^{(4)} + e_{h-4}^{(7)} + e_{h-5}^{(2)} + e_{h-6}^{(5)} + e_h^{(9)} \quad (B)$$

The fact that the above IA does not generate a self-orthogonal code may be seen from the two syndromes checking on bit $e_0^{(5)}$ [or $e_0^{(7)}$]; they both check on $e_4^{(1)}$, as well:

$$s_4^{(1)} = e_0^{(5)} + e_1^{(4)} + e_2^{(3)} + e_3^{(2)} + e_4^{(1)} + e_4^{(8)}$$

$$s_6^{(2)} = e_0^{(5)} + e_1^{(2)} + e_2^{(7)} + e_3^{(4)} + e_4^{(1)} + e_5^{(6)} + e_6^{(3)} + e_6^{(9)}$$

What really matters in the two equations above is that, apart from the first column, the sequences of the numbers in brackets are not distinct. The first column contains the bit that is checked (5). The two syndromes above are not orthogonal because they have one more common element (1 in the 5th column). Hence, orthogonality can be checked by looking at the *leftwise sequences*:

* (a,b) denotes the greatest common divisor of a & b .

5 4 3 2 1
5 2 7 4 1 6 3

It is clear from eqns (A) & (B), above, that c ranges from 1 to a maximum value $\beta-1$, where β is the first element of the 'rightmost' pair of common elements in the two eqns. In the above example this 'pair' is element 5 in columns 5 & 7 (or, the same, element 7 in columns 7 & 5). Hence, $\beta=5$ in the above example and c ranges from 1 to 4. As a consequence $(k+1)/(k+1,c)$ takes on the values 8, 4, 8 & 2. The last value is the one that causes the 'damage', because one can only have two numbers that are incongruent modulo 2.

Definition 7.3: Consider an (n,k,m) type-B code and its initial array, with elements $a_{x,z}$ $/x=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$. The leftwise sequences of error bit $e_h^{(1)}$ $/1 \leq i \leq k$ & $h \geq 0$ are rows of elements of the IA, of lengths varying between 1 and $m+1$. In particular, for every entry of the IA which is equal to i there is a leftwise sequence of elements, which is the sequence of all IA elements to the left of i . In other words, for every $a_{v,u}=i$ $/1 \leq v \leq n-k$ & $1 \leq u \leq m+1$, there is a leftwise sequence $i = a_{v,u} a_{v,u-1} a_{v,u-2} \dots a_{v,2} a_{v,1}$.

Example 7.2: Consider an (n,k,m) type-B code and its initial array. Let $k+1=8$, $a_{1,1}=1$ and $a_{2,1}=7$:

1 2 3 4 5 6 7
7 6 5 4 3 2 1

The columns with common elements (apart from the 1st one) are: 2 & 6 (element 2 or 6), 3 & 5 (element 3 or 5) and 4 & 4 (element 4). Then, the 'rightmost' pair is 4 & 4 and the first of the pair is 4, hence $\beta=4$ and c ranges from 1 to 3, requiring thus that the first-column elements are incongruent modulo 8, 4 & 8. Since $7 \equiv 3 \pmod{4}$, the above IA should generate a self-orthogonal code. This may be checked by examining all leftwise sequences:

1	2 1	3 2 1	4 3 2 1
1 2 3 4 5 6 7	2 3 4 5 6 7	3 4 5 6 7	4 5 6 7

5 4 3 2 1	6 5 4 3 2 1	7 6 5 4 3 2 1
5 6 7	6 7	7

Since they are all distinct, the syndromes that check on every bit are orthogonal on that bit, hence the code is self-orthogonal. Incidentally, $n-k=2$, $k+1=8$ & $m+1=7$, hence the resulting systematic convolutional code is a (9,7,6) one. Furthermore, there are $J=2$ syndromes checking on each message bit, so the code may correct up to one error within one constraint-length [$n_A \hat{=} n(m+1) = 9 \times 7 = 63$].

The corollary of Theorem 7.5 places restrictions on the elements of the 1st column of the IA. Consider any two rows r & v and assume that they contain at least one common element in columns other than the first one. Let β denote the 1st column of the rightmost pair of columns which contain the common elements. Then the first elements of the two rows, $a_{r,1}$ & $a_{v,1}$, must be incongruent modulo $(k+1)/(k+1,c)$ for $c=1,2,\dots,\beta-1$. If the two rows contain no common element, then it is enough for $a_{r,1}$ & $a_{v,1}$ to be incongruent (mod $k+1$); in any case if they are not, the two rows will be identical.

The process followed so far leads to the proof of the existence of the 'cyclic' CSOCs. Nevertheless, the approach is such that the results are general; stated otherwise, they are meant to include McQuilton's work, hence the introduction of two broad categories of codes, namely type-A & type-B (see Definitions 7.1 & 7.2). So far, two 'arbitrary' interventions were made in the process of determining the necessary and sufficient conditions so that a systematic convolutional code is self-orthogonal. The 1st restriction was to use exactly one bit from the current and each of the past m message error blocks in the summation that determines the current block (see Definition 7.1). The 2nd restriction is the way the IA elements are generated from the elements of the 1st column (see Definition 7.2). It is the author's opinion that the main reason for the introduction of these two restrictions is the simplification of the calculations. They definitely limit the range of the search, but they also

bring the solution within the sphere of the feasible:

7.2.4. Some Properties of Type-B Codes

The following three theorems present some useful intermediate results on type-B codes. Specifically, some progress is made towards simplifying the conditions for the existence of type-B self-orthogonal codes.

Theorem 7.6: Let $a_{x,z}$ $/x=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$ be the elements of the initial array of an (n,k,m) type-B code, with m satisfying reln (7.9) . Then, the elements of any row of the IA are distinct, while the elements, say, $a_{r,u}$ and $a_{v,u}$ of column u ($1 \leq u \leq m+1$) are equal if, and only if, the corresponding elements of the first column, $a_{r,1}$ & $a_{v,1}$, are congruent modulo $(k+1)/(k+1,u)$:

$$a_{r,u} = a_{v,u} \iff a_{r,1} \equiv a_{v,1} \pmod{(k+1)/(k+1,u)} \quad (7.14)$$

Proof: See Appendix 7.7 (§ A7.7.1., p. 451). ■

Theorem 7.7: Let $a_{x,z}$ $/x=1,2,\dots,n-k$ & $z=1,2,\dots,m+1$ be the elements of the initial array of an (n,k,m) type-B code. If p is the smallest prime factor of $k+1$ and $m \leq p-2$ then a necessary and sufficient condition for the code to be self-orthogonal is that the elements of the first column are distinct.

Proof: See Appendix 7.7 (§ A7.7.2., p. 452). ■

Theorem 7.8: Consider an (n,k,m) type-B code with $m \leq p-2$, where p is the smallest prime factor of $k+1$. The following are necessary conditions for the code to be self-orthogonal:

- i) The code rate, R , is not less than 0.5: $R \geq 1/2$.
- ii) k is even.
- iii) No column of the IA contains two equal elements.

Proof: See Appendix 7.7 (§ A7.7.3., p. 452). ■

Example 7.3: Consider an (n,k,m) type-B code with $k=34$. The smallest prime factor of $k+1=35$ is $p=5$. If $m \leq p-2$, then $m=1, 2$ or 3 . Let $m=3$ and $R \triangleq k/n = 1/2$, hence $n=2k=68$. Then, the IA is a $k \times (m+1) = 34 \times 4$ array:

1	2	3	4					18	1	19	2
2	4	6	8					19	3	22	6
3	6	9	12					20	5	25	10
4	8	12	16					21	7	28	14
5	10	15	20					22	9	31	18
6	12	18	24					23	11	34	22
7	14	21	28					24	13	2	26
8	16	24	32					25	15	5	30
9	18	27	1					26	17	8	34
10	20	30	5					27	19	11	3
11	22	33	9					28	21	14	7
12	24	1	13					29	23	17	11
13	26	4	17					30	25	20	15
14	28	7	21					31	27	23	19
15	30	10	25					32	29	26	23
16	32	13	29					33	31	29	27
17	34	16	33					34	33	32	31

Note that each of the four columns contains the integers $1, 2, \dots, 34$ exactly once, hence the IA contains each integer exactly 4 times. Then, there are exactly 4 syndromes checking on each error bit, i.e. $J=4$. According to Theorem 5.3, this $(68, 34, 3)$ systematic convolutional code can correct up to 2 errors in a span of $n_A \triangleq n(m+1) = 68 \times 4 = 272$ bits. ■

Note, from the above example, that a higher code rate may be achieved if $n-k$ is reduced. This requires the deletion of at least one of the rows of the initial array and will result in the reduction of J , from 4 to 3. This will reduce the error-correcting capability, t , to $\lfloor J/2 \rfloor = 1$, hence one might, as well, reduce J to 2. This calls for the removal of as many rows as possible so that the remaining rows contain the integers $1, 2, \dots, 34$ at least twice. There may exist an analytical solution to this problem, but it seems possible that only computer-aided calculations may produce results.

The last result of this paragraph is concerned with the relation between the effective constraint-length, n_e , of type-B codes and the corresponding leftwise sequences.

Theorem 7.9: Let $a_{x,z}$ $/x=1, 2, \dots, n-k$ & $z=1, 2, \dots, m+1$ be the elements of the initial array of an (n, k, m) type-B code. Then, the effective constraint-length, n_e , corresponding to

any particular error bit $e_h^{(i)}$ $/1 \leq i \leq k$ & $h \geq 0$, equals one plus the number of elements in the leftwise sequences of that error bit. Furthermore, if the initial array elements which are equal to i , are $a[x_1, w_1], a[x_2, w_2], \dots, a[x_j, w_j]$, then the effective constraint-length, for that error bit, equals one plus the sum of all the column-numbers of the columns that have an entry equal to i :

$$n_i = 1 + \sum_{j=1}^J w_j \quad (7.15)$$

Proof: See Appendix 7.7 (§ A7.7.4., p. 453). ■

7.3 TYPE-B1 CODES - A CLASS OF RATE-1/2 CSOCs

The following results introduce a class of systematic CSOCs that have rate 1/2.

Theorem 7.10: For every even positive integer k and every integer J in the closed range $[2, p-1]$, where p is the smallest prime factor of $k+1$, there exists a $(2k, k, J-1)$ type-B self-orthogonal code, with exactly J syndromes checking on each error bit. Such a code will also be called the (k, J) type-B1 code.

Proof: Consider a $(2k, k, m)$ type-B code and its IA. Let k & p be as defined above and $1 \leq m \leq p-2$. The IA is a $k \times (m+1)$ array, and since its elements are restricted in the range $[1, k]$, the first column may be formed (without loss of generality) by letting $a_{x,1} = x$ $/x=1, 2, \dots, k$. Hence, by Theorem 7.7, the code is self-orthogonal. By Theorem 7.8, each column contains the integers $1, 2, \dots, k$ (in some specific order) exactly once. Hence, the $k \times (m+1)$ array contains each of $1, 2, \dots, k$ exactly $m+1$ times, hence there are exactly $J=m+1$ syndromes checking on each error bit.

QED

In Appendix 7.8 (§ A7.8.1., p. 454) there are some examples of type-B1 codes. Example A7.8.1 refers to a $k+1 = p_1 p_2$

case, while Example A7.8.2 refers to a $k+1=p^a$ case. The latter may be generalized by the following lemma:

Lemma 7.2: For every odd prime p , every positive integer b and every integer J in the range $[2, p-1]$, there exists a $(2p^b-2, p^b-1, J-1)$ type-B self-orthogonal code with exactly J syndromes checking on each error bit.

Proof: This is clearly a partial case of Theorem 7.10, for $k+1 = p^b/b \geq 1$. Since $p = \text{odd prime} \implies p \geq 3 \implies p^b \geq 3 \implies k = p^b-1 = \text{even positive integer}$. Also, since p is the smallest prime factor of $k+1 = p^b$, each error bit is checked by exactly J syndromes, according to Theorem 7.10.

QED

A measure of the 'power' of the code is the ratio $\lfloor J/2 \rfloor / n_A$, i.e. the maximum number of errors the code guarantees to correct within a certain span of bits, over that span of bits. For simplicity, the ratio $(J/2)/n_A$ may be considered. Since $n_A \hat{=} n(m+1)$, for the (k, J) type-B1 code, this ratio is $(J/2)/[n(m+1)] = J/[2(2k)J] = 1/(4k)$, hence it decreases as k increases, and is independent of J .

Nevertheless, $\lfloor J/2 \rfloor$ should be measured against the effective constraint-length n_e , which is equal to the maximum number of distinct error bits that affect the decoding of any message error bit $e_0^{(i)}/1 \leq i \leq k$ (see Definition 5.9 & Example A7.8.3).

Theorem 7.11: Consider the (k, J) type-B1 code. Its effective constraint-length, n_e , is given by:

$$n_e = 1 + J(J+1)/2 \quad (7.16)$$

Proof: According to Theorem 7.10, the (k, J) type-B1 code is a $(2k, k, J-1)$ self-orthogonal type-B code with $m = J-1 \leq p-2$, where p is the smallest prime factor of $k+1$. According to Theorem 7.8, it is necessary that all IA columns are made of a distinct set of integers. Since the IA elements are restricted in the range $[1, k]$ and since each column contains exactly k elements, then each column contains exactly one element equal to i , where $1 \leq i \leq k$.

From the above discussion, there are exactly J entries in the IA which are equal to i , one in each column. Hence, the column numbers of the elements which are equal to i are $1, 2, \dots, J$. According to Theorem 7.9, the effective constraint-length corresponding to error bit $e_h^{(i)}$ $/1 \leq i \leq k$ equals $1 + (1+2+\dots+J) = 1 + J(J+1)/2$.

QED

According to Massey [18] (p. 35) "it is impossible to find a rate-1/2 convolutional code with J parity-checks orthogonal on $e_0^{(1)}$, for which n_k is smaller than this number" [given by (7.16), above]. Hence, the type-B1 codes have the minimum possible effective constraint-length, for a given J .

For a given J , apart from n_k , it is also important for the actual constraint-length, n_A , to be as small as possible. According to Massey [18], a large ratio n_A/n_k is undesirable. It has been calculated already that $(J/2)/n_A = 1/(4k)$ (see discussion following Lemma 7.2). Hence, for a given J , $J \geq 2$, the best code is the one corresponding to minimum k . By Theorem 7.10, $J \leq p-1$, where $p | (k+1) \implies p \leq k+1$. Hence, $J \leq p-1 \leq k \implies k \geq J$ and the minimum value of k is J , BUT this is permitted only if $J+1$ is an odd prime. The following theorem summarizes these findings:

Theorem 7.12: Let J be any integer greater than one. The 'best' (k, J) type-B1 code has $k=p-1$, where p is the smallest prime greater than J . 'Best' here means a code with minimum actual constraint-length. For the above case,

$$n_A = 2J(p-1) \geq 2J^2 \quad (7.17)$$

Proof: Let $J > 1$ and consider the determination of those values of k , for which the (k, J) type-B1 code exists.

According to Theorem 7.10, $2 \leq J \leq p-1$, where p is the smallest prime factor of $k+1$, i.e.

$$k+1 = p_1^{a_1} p_2^{a_2} \dots p_r^{a_r} \quad /r \geq 1, p_1 < p_2 < \dots < p_r, a_i \geq 1 \quad (A)$$

Since $n_A = n(m+1) = (2k)J$ (see Theorem 7.10), the minimum- n_A code (i.e. the 'best' code) corresponds to the mini-

imum possible value of k , for a given J . From (A) it is obvious that this k is $p-1$ and since $J \leq p-1$, the 'best' code is the one with the minimum k , such that $k = p-1 \geq J$. Hence, the first prime p , greater than J is chosen for $k+1$.

The actual constraint-length is $n_A = 2kJ = 2J(p-1)$. The minimum possible value of p is achieved when the first integer, after J , is a prime, i.e. if $J+1=p$. Then $n_A = 2J^2$.

QED

The table in Appendix 7.8 (§ A7.8.2., p. 456) gives the 'best' type-B1 code, for various selected values of J , together with the corresponding values of k , n_E , n_A & n_A/n_E . The n_A of the 'best' type-B1 codes is compared with that of rate-1/2 CSOCs constructed by Massey [18], or Wu [45].

From the table, one may conclude that type-B1 codes, when compared with codes discovered by Massey [18], or Wu [45], are equally good as far as the effective constraint-length is concerned [all codes satisfy (7.16)], but longer, as far as the actual constraint-length is concerned.

Type-B1 codes become relatively shorter, as J increases (they are only 2.5% longer for $J=100$) and the trend is for this difference to disappear.

If only the codes meeting the lower bound [see relation (7.17)] on n_A ($n_A = 2J^2$) are considered (they are marked with an *), then type-B1 codes are only slightly longer in n_A compared with the Massey or Wu ones (the % extra n_A is shown below for various J s):

%:	100.0,	71.4,	23.5,	21.0,	8.0,	7.4,	3.3,	3.4,	2.8,	2.5
J:	4	6	10	12	18	28	42	60	72	82

It is obvious that type-B1 codes are less than 8% longer, for $J \geq 18$, than the corresponding Wu codes. This figure goes down to 2.8% for $J \geq 72$.

The class of type-B1 codes is obviously infinite, as far as the permitted values of J is concerned, although one cannot expect to think of applications of CSOCs with J s in excess of, say, 100. For example, the last entry of TABLE A7.8.1, is a rate-1/2 CSOC which guarantees to correct any error pattern, of weight up to 4,500, within a span of 162

million bits (= 19.3 Mbytes).

Finally, it is obvious from the table that, the ratio n_A/n_E takes on values around 4 and as J increases it stabilizes to 4. This may be verified theoretically, from eqns (7.16) & (7.17):

$$n_A/n_E = 2(p-1)J/[J(J+1)/2+1] \longrightarrow$$

$$n_A/n_E = 4J(p-1)/(J^2+J+2) = 4/[J/(p-1)+1/(p-1)+2/J/(p-1)]$$

According to Theorem 7.12, if $J+1 = \text{prime}$, $J = p-1$:

$$n_A/n_E = 4/(1+1/J+2/J^2) < 4 \quad \text{and} \quad n_A/n_E \longrightarrow 4 \quad /J \longrightarrow +\infty$$

For $J+1 < p$, and J greater than about 10, $1/(p-1) + 2/[J(p-1)] \approx 1/(p-1)$. For the same reason $J/(p-1)+1/(p-1) \approx J/(p-1)$. Hence, $n_A/n_E \approx 4(p-1)/J > 4$, but as J increases, $(p-1)/J$ tends to become 1 and $n_A/n_E \longrightarrow 4$.

7.4 OTHER CLASSES OF TYPE-B SELF-ORTHOGONAL CODES

Theorem 7.10 introduced a class of $(2k, k, m)$ type-B self-orthogonal codes, where $k=\text{even}$, $1 \leq m \leq p-2$ and p is the smallest prime factor of $k+1$, (type-B1 codes).

The important restriction with this class of codes is not their rate $(1/2)$, or that $k=\text{even}$, but that $m < p-1$. It has already been discussed (following Example 7.3), that the IA of a type-B1 code may be modified to produce a self-orthogonal code of higher rate (and lower J), by deleting some of its rows. That type-B1 codes are used as a starting point owes to the fact that each column of their initial array contains each of the integers $1, 2, \dots, k$ exactly once (see Theorem 7.8). Another group of codes that offer this 'facility' is the $(n, k, k-1)$ type-B group of codes. In this case, each row contains each of $1, 2, \dots, k$ exactly once.

It must be obvious, by now, that the relation between m & p is crucial for the structure of the initial array. So far, the case $m < p-1$ and $R \leq 1/2$ (in practice $R=1/2$ - see Theorem 7.8) has been considered. In the next paragraph, the case $m < p-1$ and $R > 1/2$ will be considered (this completes the general case $m < p-1$). In paragraph 7.4.2., the case $p-1 \leq m < k-1$

will be examined. Finally, in the next section, the case $m \geq k-1$ will be looked-over (in practice $m=k-1$, because $k-1$ is the maximum value of m - see immediately after Theorem 7.3).

7.4.1. Type-B Self-Orthogonal Codes with $m < p-1$ & $R \neq 1/2$

Example 7.4: Consider the initial array for the (24,4) type-B1 code introduced in Example A7.8.2. In order to increase the rate, $n-k$ (= number of rows) will have to decrease. A consequence will be the reduction of J from 4 to 3, and since even values of J are preferable the problem may be stated as following: "Delete as many rows as possible, so that there are at least two entries for each of the integers $1, 2, \dots, 24$ ". Obviously, the optimum solution is to delete exactly half of the rows. Then there will be 12 rows, of 4 elements each, a total of 48 elements, exactly enough for each of $1, 2, \dots, 24$ to appear twice. Then, $n-k = 12$ and since $k = 24$, $n = 24+12=36$ and $R \hat{=} k/n = 24/36 = 2/3$. So, the best solution will be a rate-2/3 code. Hence, the following strategy for the reduction of the IA:

"Choose a number of rows, so that no number appears twice. If necessary, add a minimum number of rows so that each of $1, 2, \dots, 24$, appears at least once. Repeat the above, until each of $1, 2, \dots, 24$ appears at least J times".

During the first 'effort' rows 1,5,6,11,16 & 21 are chosen. A quick check verifies that all 24 integers are included exactly once. During the second 'effort', rows 2,7,10,12,17 & 22 are chosen. Again it can be verified that no more rows are needed:

Original				IA		Modified				IA		
1	2	3	4	→	13	1	14	2	1	2	3	4
2	4	6	8		14	3	17	6	2	4	6	8
3	6	9	12		15	5	20	10	5	10	15	20
4	8	12	16		16	7	23	14	6	12	18	24
5	10	15	20		17	9	1	18	7	14	21	3
6	12	18	24		18	11	4	22	10	20	5	15
7	14	21	3		19	13	7	1	11	22	8	19
8	16	24	7		20	15	10	5	12	24	11	23
9	18	2	11		21	17	13	9	16	7	23	14
10	20	5	15		22	19	16	13	17	9	1	18
11	22	8	19		23	21	19	17	21	17	13	9
12	24	11	23		24	23	22	21	22	19	16	13

The resulting code is a rate-2/3, $J=2$ CSOC, with $n_A \hat{=} n(m+1) = (24+12)(3+1) = 144$. The effective constraint-length

for bit $e_h^{(1)}$ $/1 \leq i \leq 24$ is equal to one plus the sum of the IA columns that contain i (see Theorem 7.9). n_x for the code is the maximum for all $i = 1, 2, \dots, 24$. This is obviously $n_x = 1+3+4 = 8$ (for, say, $i=3$, or $i=13$, etc).

Theorem 7.13: Consider the initial array for the (k, J) type-B1 code. Then, for any positive integer y , less than k , there exists a $(2k-y, k, m)$ type-B self-orthogonal code. The minimum number of syndromes, J , that check on any error bit is upper-bounded by $(m+1)(1-y/k)$.

Proof: See Appendix 7.9 (§ A7.9.1., p. 458).

For any useful results to emerge, Theorem 7.13 alone is not sufficient. The problem area is the value of J . If rows are deleted at random (from the initial IA of the type-B1 code), it is possible that $J < 2$, which will make the code useless. Even if $J \geq 2$ the code may be inefficient, in case rows are deleted in such a way that some integers (from the set $\{1, 2, \dots, k\}$) are under-represented. The optimum solution would correspond to all integers being equally represented, which corresponds to the upper bound mentioned in the above theorem. For the case of Example 7.4, $k=24$, $m+1=4$ and $y=12$. Then the new code should have a value of J upper-bounded by $(m+1)(1-y/k) = 4(1-12/24) = 2$. In fact $J=2$, for the above example, hence the resulting type-B code is the best.

What is required thus, is a method for the best possible way to delete rows from the IA of a type-B1 code. Specifically and for a given value of J , a method is required to delete rows so that each integer in the set $\{1, 2, \dots, k\}$ appears exactly J times in the remaining initial array. If this is impossible (and under what conditions is this so?), the next best solution should be obtained.

Note from Example 7.4 that the first-column elements that were left after the deletion are 1, 2, 5, 6, 7, 10, 11, 12, 16, 17, 21 & 22. Ignoring for the time being 5 & 10, the rest form a pattern: 1, 2, 6, 7, 11, 12, 16, 17, 21, 22. Note also that multiples of 5 (i.e. integers 5, 10, 15 & 20) appear only along the rows

with first-column elements 5,10,15 & 20, while all other elements only along the rest of the rows. Hence there is a partition of the elements in two distinct groups of rows: Multiples of 5 (remember that 5 is the smallest prime divisor of $k+1$) appear along rows with 1st elements $5i$ $/i=1,2,3,4$, while all the rest along rows with 1st elements j $/j=1,2,\dots,24$ & $j \neq 5i$. The above may be generalized with the help of the following theorem:

Theorem 7.14: Consider the initial array of the (k,J) type-B1 code, with elements $a_{x,z}$ $/x=1,2,\dots,k$ & $z=1,2,\dots,J$. Let p be the smallest prime factor of $k+1$ and $a_{x,1} = x$ $/x=1,2,\dots,k$. Then the following hold true:

$$i) \quad a_{x,z} + a_{k+1-x,z} = k+1 \quad (7.18)$$

$$ii) \quad \text{For all } x \quad /1 \leq x \leq k: \quad p|x \iff p \mid a_{x,z} \quad (7.19)$$

$$iii) \quad \text{For any } b, i \neq j, z \text{ \& } w, \text{ such that } 1 \leq b < p, \\ 1 \leq i \neq j < (k+1)/p \text{ \& } 1 \leq z, w \leq J: \quad a_{b+ip,z} \neq a_{b+jp,w} \quad (7.20)$$

Proof: See Appendix 7.9 (§ A7.9.2., p. 459). ■

Hence, the IA of the (k,J) type-B1 code has the following properties: 1. The elements along rows with 1st elements x and $(k+1)-x$ add-up to $k+1$. 2. The elements along rows with 1st element ip $[i=1,2,\dots,(k+1)/p-1]$ are all multiples of p , while no multiple of p appears elsewhere. 3. The elements along any two rows, with row-numbers that differ by a multiple of p and which are not multiples of p , are distinct.

The above results may be used to delete rows from the IA of the $(k,p-1)^*$ type-B1 code. According to Theorem 7.10, the integers in the set $\{1,2,\dots,k\}$ appear exactly $p-1$ times each. By Theorem 7.14 the IA is partitioned into two sets of rows, according to the 1st element, x , of each row:

$$R1 \hat{=} \{x \quad /x=1,2,\dots,k \text{ \& } p \nmid x\} \quad (7.21a)$$

$$R2 \hat{=} \{ip, i=1,2,\dots,(k+1)/p-1\} \quad (7.21b)$$

Because $R1$ & $R2$ are disjoint and their union is $\{1,2,\dots,k\}$, $|R1| + |R2| = k$ (see Biggs [36], p. 44). Then, $|R1| =$

* The $(k,p-1)$ type-B1 code has the widest possible IA.

$(p-1)(k+1)/p$ and $|R_2| = (k+1)/p-1$. $|A|$ denotes the number of elements in set A .

Set R_1 contains the 1st-column elements that are not multiples of p , while set R_2 contains the multiples of p , only. Also, by Theorem 7.14, rows with 1st elements $1, p+1, 2p+1, \dots, k+2-p$ contain a distinct set of integers; the same applies to rows with first elements $2, p+2, 2p+2, \dots, k+3-p$, etc, $p-1, 2p-1, 3p-1, \dots, k$. Each subset of rows contains $(k+1)/p$ rows, while each row contains $p-1$ elements, hence each subset of rows contains $(p-1)(k+1)/p$ distinct elements. It would be ideal if this number equals the number of non-multiples of p in the range $[1, k]$, i.e. the elements in set R_1 . In such a case, if J of these subsets were to be chosen, then each of the elements of R_1 is chosen exactly J times and the code construction problem is reduced to the selection of rows whose 1st element is a multiple of p , so that each of the multiples of p appear also exactly J times (or, failing that, as close to, and not less than, J times). If this succeeds, then a self-orthogonal code has been constructed with J syndromes checking on each error bit.

According to the above discussion, and for the general (k, J') type-B1 code ($1 < J' < p$), the subset of R_1 containing rows with 1st elements $b, p+b, 2p+b, \dots, k+1+b-p$, where $1 \leq b < p$, contains $J'(k+1)/p$ distinct elements that are not multiples of p , while there are $|R_1| = (p-1)(k+1)/p$ distinct elements in the IA that are not multiples of p . Since, by Theorem 7.10, $2 \leq J' < p$, if $J' = p-1$, then each of the above-mentioned subsets of rows contains each of the numbers $1, 2, \dots, k$, that are not multiples of p , exactly once.

Then, to construct a self-orthogonal code with J checks on each error bit, one would start with the initial array of the $(k, p-1)$ type-B1 code and keep only rows with 1st-column elements $b, p+b, 2p+b, \dots, k+1+b-p$, for J values of b , between 1 and $p-1$. This will result in exactly J copies of each element in the new IA, but no elements that are multiples of p are included, as yet. Hence, the following theorem:

Theorem 7.15: Consider k & J integers, such that $1 < J < p$ and k is even and positive, where p is the smallest prime

factor of $k+1$. Then, there exists an $(n, k, p-2)$ type-B self-orthogonal code, with at least J syndromes checking on error bits $e_0^{(p^i)}$ $/i=1, 2, \dots, (k+1)/p-1$ and exactly J syndrome bits checking on the rest of the error bits.

The elements of the first column of the initial array are selected as following: J values of b between 1 and $p-1$ are chosen, at random. For each one of them, $b, b+p, b+2p, \dots, k+1+b-p$ are selected as elements of the first column, a total of $J(k+1)/p$ elements. Furthermore, from the set of the multiples of p , [a total of $(k+1)/p-1$ elements], a minimum number is selected so that the corresponding rows contain at least J copies of each of the multiples of p . n equals the number of rows of the resulting initial array plus k .

The theorem below gives bounds on n , for the $(n, k, p-2)$ type-B self-orthogonal codes, introduced above.

Theorem 7.16: Let k be a positive even integer and J an integer in the closed range $[2, p-1]$, where p is the smallest prime factor of $k+1$. Then the $(n, k, p-2)$ type-B self-orthogonal code has n bounded by:

$$Jk/(p-1)+k \leq n \leq (J+1)(k+1)/p-1+k \quad (7.22)$$

Proof: See Appendix 7.9 (§ A7.9.3., p. 460).

Let us recap. Consider the initial array for the $(k, p-1)$ type-B1 code, where k is even and p is the smallest prime factor of $k+1$. This IA is used as a starting point. A number of rows are to be deleted so that a higher-rate code is obtained (remember that the number of rows equals $n-k$), with at least J syndromes checking on each error bit. The new code will be an $(n, k, p-2)$ self-orthogonal one, but for a given value of J , n may be unnecessarily high. This will happen if some error bits are checked by more than J syndromes; the latter is equivalent with some of the integers in the set $\{1, 2, \dots, k\}$ appearing in the IA more than J times. Hence, the ideal solution is to include enough rows so that each integer to appear J times. Such an ideal solution may

be non-existent, for k & J given, and in any case it may not be easy to obtain. Note also that the code, to be constructed, has $m=p-2$, clearly a breach of generalization (which would require $2 \leq m+1 \leq p-1$). This is done because if $m < p-2$, it would not be possible to construct an IA containing each of the elements of R_1 , exactly J times (only at least J times).

The best that can be done, at this stage, is to consider a few special cases and propose a computer-aided search for the general case. The special cases will be related to the way the rows with the multiples of p are chosen.

Certainty and optimality can be achieved if $|R_2|$ equals $p-1$, i.e. the number of IA columns. Under such a condition, each row will contain all the multiples of p , exactly once. From (7.21b), $(k+1)/p-1=p-1 \implies k+1 = p^2$. Then, the new IA should include J of these rows. On top of these, there are $J(k+1)/p$ rows containing the non-multiples of p (see Theorem 7.15). Hence, the IA will have $J+J(k+1)/p = J(k+1+p)/p$ rows, so $n = k+J(k+1+p)/p = p^2-1+J(p+1) = (p+1)(p-1)+J(p+1) = (p+1)(J+p-1)$. Also, the actual constraint-length is $n_A \hat{=} n(m+1) = (p+1)(J+p-1)(p-1) = (p^2-1)(J+p-1)$ and the code rate is $R \hat{=} k/n = (p-1)(p+1)/[(p+1)(J+p-1)] = (p-1)/(J+p-1)$. Hence, the following theorem:

Theorem 7.17: For every odd prime p , and every integer J , such that $2 \leq J \leq p-1$, there exists an $(n, p^2-1, p-2)$ type-B self-orthogonal code, with exactly J syndromes checking on each error bit and block length $n = (p+1)(J+p-1)$. The code rate is $R = (p-1)/(J+p-1)$ and the actual constraint-length is $n_A = (p^2-1)(J+p-1)$. Such a code will also be called the (p, J) type-B2 code.

The elements of the first column of the IA are selected as following: J values of b between 1 and $p-1$ are chosen, at random. For each one of them, $b, b+p, b+2p, \dots, k+1+b-p$ are selected as elements of the first column, a total of $J(k+1)/p$ elements. Furthermore, J multiples of p are selected, also at random, to serve as the remaining 1st-column elements.

Another useful case is the one where the number of multiples of p is greater than the width of the array but not greater than twice that width. In such a case, two rows are enough if they contain a distinct set of integers.

Theorem 7.18: For every odd prime p , and every prime q , such that $p < q < 2p$ (if such a q exists) and every integer J , such that $2 \leq J \leq (q-1)/2$, there exists an $(n, pq-1, p-2)$ type-B self-orthogonal code, with at least J syndromes checking on each error bit and block length $n = (q+2)J + pq - 1$. Such a code will be called the (p, q, J) type-B3 code.

The elements of the first column of the initial array are selected as following: J values of b between 1 and $p-1$ are chosen, at random. For each one of them, $b, b+p, b+2p, \dots, pq+b-p$ are selected as elements of the first column, a total of Jq elements. Furthermore, elements ip & $pq-ip$ $/i=1, 2, \dots, J$ are selected to generate the multiples of p .

Proof: See Appendix 7.9 (§ A7.9.4., p. 461).

Example 7.5: Let $p=7$. Then q (if it exists) should be a prime such that $7 < q < 14$. There are two choices, $q=11$ or $q=13$. Let $q=11$. Then, $k=pq-1=76$, $m=p-2=5$ and $2 \leq J \leq (11-1)/2=5$. Let $J=4$. Then, $n=(q+2)J+pq-1=13 \times 4 + 76 = 128$, $R \hat{=} k/n = 76/128 = 19/32$ and $n_A \hat{=} n(m+1) = 128 \times 6 = 768$.

7.4.2. Type-B Self-Orthogonal Codes with $p-2 \leq m \leq k-1$

As mentioned earlier, the relation between m and the divisors of $k+1$ is an important factor influencing the construction of the initial array of a type-B self-orthogonal code. Let d_1 & d_2 be divisors of $k+1$ such that $p \leq d_1 \leq m+1 < d_2 < k+1$, where p is the smallest prime factor of $k+1$. For example, if $k+1 = 5^2 \times 7 = 175$, then $p=5$ and the divisors of $k+1$ (which are $< k+1$ & $\geq p$) are 5, 7, 25 & 35. If $d_1=7$, then $d_2=25$ and $m+1$ may be between 7 and 24.

It has been shown that the problem of constructing a self-orthogonal code has been reduced to the one of select-

ing a subset of $\{1, 2, \dots, k\}$ to serve as the 1st column of the IA. This subset must be such that: 1. m is $\leq p-2$ (or, the same, Theorem 7.3 is satisfied). 2. The code is self-orthogonal (or, the same, the corollary of Theorem 7.5 is satisfied). 3. J is ≥ 2 (or, the same, the IA contains at least 2 copies from each integer in the set $\{1, 2, \dots, k\}$).

The theorem below deals with the first two specifications:

Theorem 7.19: Consider an (n, k, m) type-B code* and its initial array with first-column elements x , where $1 \leq x \leq k$. Let d_1 & d_2 be two consecutive divisors of $k+1$, such that $p \leq d_1 \leq m+1 < d_2 < k+1$, where p is the smallest prime factor of $k+1$. Then if a row is not to contain a zero, its first element, x , must not be a multiple of any divisor of $k+1$ greater than $(k+1)/d_2$. Furthermore, J_1 , the number of syndromes checking on error bit $e_0^{(i)}$ $1 \leq i \leq k$, equals the number of pairs (x, z) which satisfy the congruence $xz \equiv i \pmod{k+1}$. z is restricted to be in the range $[1, m+1]$ while x must not be a multiple of any divisor of $k+1$ greater than $(k+1)/d_2$ and must be in the range $[1, k]$.

Proof: See Appendix 7.9 (§ A7.9.5., p. 463). ■

Consider now the congruence $xz \equiv i \pmod{k+1}$. This will be solved for x , for each value of $z=1, 2, \dots, d_2-1$, and the result will be tested to determine if it is a multiple of a divisor of $k+1$, greater than $(k+1)/d_2$. The range of values of z will be partitioned by three subsets. This arises from the observation that either $(z, k+1) = 1$, or $d \triangleq (z, k+1) > 1$ and if the latter condition is valid, either $d|i$, or $d \nmid i$.**

According to Theorem A7.2.7, if $(z, k+1) = 1$, then congruence $zx \equiv i \pmod{k+1}$ has a unique (modulo $k+1$) solution given by $x \equiv iz^{(k+1)-1} \pmod{k+1}$. The solution will not be acceptable if $(x, k+1) > (k+1)/d_2$.

According to Theorem A7.2.5, if $(z, k+1) \triangleq d > 1$ and $d|i$, then congruence $zx \equiv i \pmod{k+1}$ has exactly d solutions, given by $t + j(k+1)/d$ $/j=0, 1, \dots, d-1$; t is the solution of congruence $(z/d)x \equiv i/d \pmod{(k+1)/d}$. A solution, x , will

* If it exists.

** (a, b) denotes the greatest common divisor of a & b .

not be acceptable if $(x, k+1) > (k+1)/d_2$.

According to Theorem A7.2.5, if $(z, k+1) \hat{=} d > 1$ and $d \nmid i$, then congruence $zx \equiv i \pmod{k+1}$ has no solutions. **

The following theorem is based on the above idea.

Theorem 7.20: Consider an (n, k, m) type-B code* and let d_1 & d_2 be two consecutive divisors of $k+1$, such that $p \leq d_1 \leq m+1 < d_2 < k+1$, where p is the smallest prime factor of $k+1$. Let an initial array (IA) element i ($i \in [1, k]$) and let the greatest common divisor (gcd) between i & $k+1$ be e [$(i, k+1) \hat{=} e$]. Then IA column z ($z \in [1, m+1]$) contains up to d copies of element i , if i is divided by d , and none otherwise; d is the gcd between z & $k+1$ [$d \hat{=} (z, k+1)$].

Specifically, a column z which is relatively prime to $k+1$ [$(z, k+1) \hat{=} d = 1$], contains exactly one copy from each element i (where $i \in [1, k]$) which is such that $e \hat{=} (i, k+1) \leq (k+1)/d_2$ and no elements i for which $(i, k+1) \hat{=} e > (k+1)/d_2$.

A column z not relatively prime to $k+1$ [$(z, k+1) \hat{=} d > 1$], contains only elements i that are both multiples of d and such that $e/d \leq (k+1)/d_2$. There may be up to d copies of each multiple of d , along column z , depending on the number of j 's ($0 \leq j \leq d-1$) which satisfy inequality (7.23a).

$$\gcd\left(\varphi + j(k+1)/d, k+1\right) \leq (k+1)/d_2 \quad (7.23a)$$

$$\text{where, } \varphi \equiv (i/d)(z/d)^{\hat{=}(k+1)/d-1} \pmod{(k+1)/d} \quad (7.23b)$$

Proof: See Appendix 7.9 (§ A7.9.6., p. 464).

The last theorem is a contribution towards the determination of J (i.e. the minimum of $J_i / i = 1, 2, \dots, k$, where J_i is the number of syndromes checking on the i th error bit). It is possible to obtain an expression for J , for at least some partial cases (like, for example, $k+1=pq$, or $k+1=pqr$, or $k+1=p^a$, etc, where p , q & r are primes). Such an exercise though, is judged to be not worth exploring. This is so because most of the type-B codes with $p-2 < m < k-1$, are not self-orthogonal. The determination of which are and which are not self-orthogonal codes requires a (relatively simple)

* If it exists.

** (a, b) denotes the greatest common divisor of a & b .

computer-aided investigation, which can easily determine J.

For example, a simple Basic programme required about a minute of processing time in an Amstrad CPC6128 to conclude that the (344,174,5) type-B code is a J=5 self-orthogonal one. Similarly, it was found that the $(n_1, 174, 23)$, $(n_2, 174, 33)$ and $(n_3, 274, 9)$, are not self-orthogonal codes.

7.5 $(n, k, k-1)$ TYPE-B SELF-ORTHOGONAL CODES

The aim of this section is to investigate under what conditions an $(n, k, k-1)$ type-B code is a self-orthogonal one. The intention is to modify, for the (simpler) case of $m+1=k$, the results presented in Section 7.2.

The next theorem is an application of Lemma 7.2, for the case where $m = k-1$.

Theorem 7.21: Consider an (n, k, m) type-B code and its initial array (IA), with elements $a_{x,z}$, where $x=1, 2, \dots, n-k$ and $z=1, 2, \dots, m+1$, and with $k = \text{even}$.

A necessary and sufficient condition for the IA to have $m+1 = k$ columns [or, the same, for an $(n, k, k-1)$ type-B code to exist], is:

$$\left(a_{x,1}, k+1\right)^* = 1 \quad \text{for all } x=1, 2, \dots, n-k \quad (7.24)$$

Furthermore, the $(n, k, k-1)$ code is self-orthogonal iff the elements of the first column of the IA are incongruent to each other modulo any non-trivial divisor of $k+1$:

For all $x, y \in [1, n-k] \quad /x \neq y,$

$$a_{x,1} \not\equiv a_{y,1} \pmod{d} \quad \text{for all } d|(k+1) \quad /1 < d \leq k+1 \quad (7.25)$$

Proof: See Appendix 7.10 (§ A7.10.1., p. 466). ■

The theorem above is very useful in generating results and, in particular, in generating methods for the systematic construction of self-orthogonal type-B codes. It is valid though only for even values of k . The next theorem deals with all odd- k cases.

* (a, b) denotes the greatest common divisor of a & b .

Theorem 7.22: For every odd integer k ($k > 1$), there exists a $(k+2, k, k-1)$ type-B self-orthogonal code with two syndromes checking on each error bit ($J=2$). Such a code will also be called the k type-B4 code ($k = \text{odd} \ \& \ > 1$).

The following hold true for a k type-B4 code:

- i) There are exactly $\Phi(k+1)$ distinct initial arrays (IA_s) corresponding to the above code.
- ii) The two elements of the first column of the IA_s are x & $k+1-x$, where x is any positive integer relatively prime to, and not exceeding, $k+1$.
- iii) If $z \hat{=} (k+1)/2$, $a_{1,z} = a_{2,z} = z$.
- iv) For $k = \text{odd}$, there are no other self-orthogonal type-B codes, apart from the type-B4 ones.

Proof: See Appendix 7.10 (§ A7.10.2., p. 468). ■

Theorem 7.22 proved the existence of a class of single-error correcting ($J=2$), high-rate [$R = k/(k+2)$] self-orthogonal codes with $n_A = k(k+2)$, for every odd integer $k > 1$. The next theorem deals with even values of k .

Theorem 7.23: For every even positive integer k and every integer n in the range $[k+1, k+p-1]$, where p is the smallest prime factor of $k+1$, there exists an $(n, k, k-1)$ type-B self-orthogonal code with $J = n-k$ syndromes checking on each error bit. Such a code will also be called the (n, k) type-B5 code. The (n, k) type-B5 code is the only $(n, k, k-1)$ type-B code that is self-orthogonal. Furthermore, the set of first-column elements $\{a_{x,1} / x=1, 2, \dots, n-k\}$ is any subset of the set $B(\mu)$, defined for any $(\mu, k+1) = 1$, by:

$$B(\mu) \hat{=} \{b_j / j=1, 2, \dots, p-1, b_j \equiv \mu j \pmod{k+1}, 1 \leq b_j \leq k\} \quad (7.26)$$

Proof: See Appendix 7.10 (§ A7.10.3., p. 476). ■

Note that Theorem 7.23 proves that, if $k+1 = \text{odd}$ and $n \in [k+1, k+p-1]$ (p is the smallest prime factor of $k+1$), the (n, k) type-B5 code is the only $(n, k, k-1)$ type-B self-orthogonal code. Although the theorem provides ways for the construction of the IA , it does not specify if the proposal

generates all the appropriate IAs. It may be proved that μ in (7.26) may be replaced by μ_j , i.e. a separate value of μ for each b_j , $j=1,2,\dots,p-1$. Of course μ_j will be restricted to be relatively prime to $k+1$ and possibly it will be restricted otherwise, as well. The author believes, though, that this generalization will not generate more valid IAs because of duplications. Of course this is something to be proved but, because of the insignificance of the potential results, the effort on type-B codes will terminate with the following example.

Example 7.6: Let $k+1 = 5 \times 7 = 35$. According to Theorem 7.23 there exist $(n,34)$ self-orthogonal (SO) codes with $n \in [35,38]$ & $J = n-k$ (corresponding to $1 \leq J \leq 4$). Let $J=4$, hence $n = k+4 = 38$. Then, $R = 34/38 = 17/19$.

μ may assume any value, provided that $(\mu,35) = 1$. From (7.26): $B(1) = \{1,2,3,4\}$, $B(2) = \{2,4,6,8\}$, $B(16) = \{16,32,13,29\}$, etc. The elements of the B_s are supposed to be incongruent modulo any non-trivial divisor, d , of 35. Reduced (mod 5), the elements of $B(16)$ are 1,2,3,4, while those of $B(2)$ are 2,4,1,3. Reduced (mod 7) the elements of $B(16)$, or $B(2)$ are 2,4,6,1. Therefore, $B(1)$, or $B(2)$, or $B(16)$, or etc, can indeed provide the 1st column of the IA.

7.6 TYPE-C CODES: CYCLICALLY-DECODABLE TYPE-B CODES

A third (and final) 'arbitrary' restriction will be imposed on the codes. The previous 'arbitrary' restrictions were the one imposed on systematic convolutional codes by Definition 7.1 (type-A codes) and the one imposed on type-A codes by Definition 7.2 (type-B codes).

Threshold decodable codes are especially suitable for hardware implementation of their encoder & decoder. A linear encoder is made of shift-register (SR) stages and X-OR gates. The corresponding decoder (see Fig. 5.1) is made of a demultiplexer, a replica of the encoder, X-OR gates, majority gates (MGs) and the syndrome register (SYRE) (Fig. 7.1).

The MGs are the most expensive part of the decoder, hence it is reasonable for one to try and minimize their number. Hence, a reduced number of MGs is required, hopefully one, perhaps more, but definitely less than k . This is expected to be achieved by sharing the MGs and exploiting some property of the code. For example, after the decoding of $e_h^{(1)}$, an appropriate shift of the contents of the SYRE could bring the syndromes, checking on another bit, at the input of the gates used by $e_h^{(1)}$, and so on.

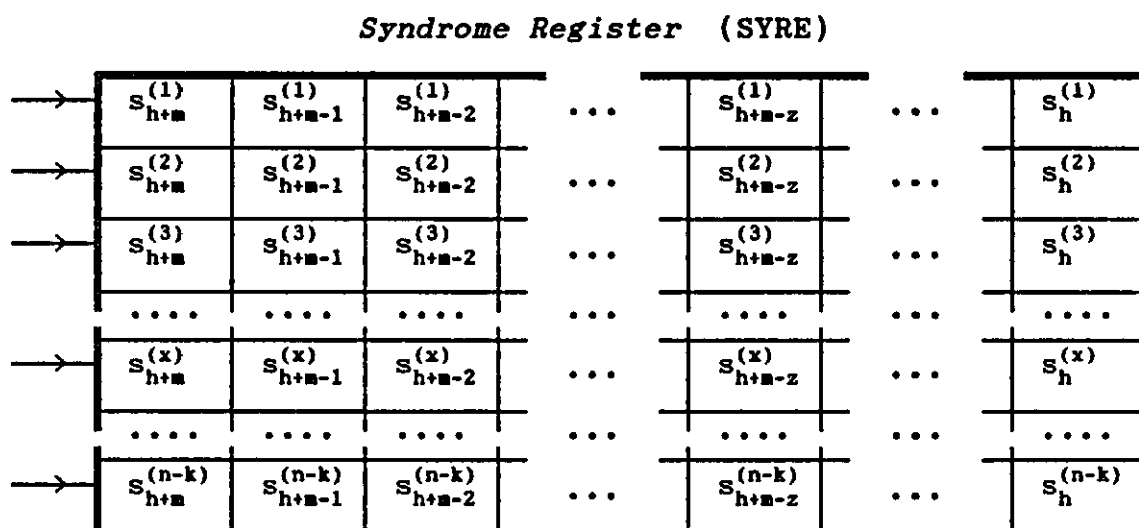


Figure 7.1: The contents of the SYRE at the time-unit of the estimation of e_h .

The type of shift envisaged is a vertical or horizontal one, by one position. This assumes that the extra hardware required to shift the contents of the SYRE, either vertically or horizontally or both, will not exceed the savings obtained by the reduction of the number of the MGs.

Definition 7.4: A code is *cyclically decodable* if there exist at least two error bits, $e_h^{(a)}$ & $e_h^{(\beta)}$ / $1 \leq a, \beta \leq k$ & $a \neq \beta$, such that the syndromes checking on $e_h^{(\beta)}$ can be brought into the positions (co-ordinates) of the syndromes checking on $e_h^{(a)}$, by a one-step uniform cyclic-shift of the syndrome-register (SYRE). A *one-step uniform shift* is the one where all the contents of the SYRE are shifted simultaneously by one position either to the right, or to the left, or up-

wards, or downwards. A shift is *cyclic* if the contents of the last stage are fed back into the first stage, and vice-versa, according to the direction of the shift.

7.6.1. Cyclically-Decodable SO Type-B Codes

The first theorem excludes the possibility of a horizontally-shifted SYRE:

Theorem 7.24: There are no horizontal-shift cyclically decodable type-B self-orthogonal codes.

Proof: See Appendix 7.11 (§ A7.11.1., p. 478).

Consider now the necessary conditions for the existence of vertical-shift SO type-B codes. In Appendix 7.11 (§ A7.11.2., p. 479), it was proved that these conditions are:

$$z(1)a_{x(1),1} \equiv z(2)a_{x(2),1} \equiv \dots \equiv z(J)a_{x(J),1} \pmod{k+1} \quad (7.27a)$$

$$z(1)a_{x(1)\pm 1,1} \equiv z(2)a_{x(2)\pm 1,1} \equiv \dots \equiv z(J)a_{x(J)\pm 1,1} \pmod{k+1} \quad (7.27b)$$

where $s_{h+z(j)-1}^{(x(j))} / j=1,2,\dots,J$ are the syndromes checking on $e_h^{(a)}$. (7.27) are $2J-2$ simultaneous congruences in J unknowns. A solution (if it exists) depends on the relation among the $a_{x(j),1}$ s, $z(j)$ s and $k+1$, and has to satisfy the corollary of Theorem 7.5, as well.

Since results do not seem to be easily obtainable, from (7.27), a more strict specification will be introduced:

Definition 7.5: An (n,k,m) type-B code will be said to be an (n,k,m) type-C code, if all its error bits are cyclically decodable*.

Before an attempt is made to quantify Definition 7.5, in terms of some relation among the elements of the 1st column of the IA, some properties of the SYRE will be proved.

Theorem 7.25: The following are necessary conditions for the existence of an (n,k,m) type-C code:

* Only vertical shift will be considered - see Theorem 7.24.

i) No row, or column, of the syndrome register contains more than one syndrome checking on the same error bit.

ii) The initial array is organized into $k/(n-k)$ groups of columns, called *cosets*. By necessity, $n-k$ must divide k . Each column of a coset is a cyclic shift of another column of that coset. The first column (= the one with the smallest column number) of a coset is called the *coset leader*. The *first coset* is the one with the 1st column as coset leader. If coset i contains J_i columns, then:

$$\sum_{i=1}^{k/(n-k)} J_i = m+1 \quad (7.28a)$$

$$J_i \leq n-k \quad /i=1,2,\dots,k/(n-k) \quad (7.28b)$$

iii) If a is the first element of the first column of the IA, there must exist a column β of the first coset, such that the elements of the first coset of the IA are congruent (mod $k+1$) to $a, \beta a, \beta^2 a, \dots, \beta^{n-k-1} a$. Also:

$$\beta^i \not\equiv 1 \pmod{k+1} \quad /i=1,2,\dots,n-k-1 \quad (7.29a)$$

$$\beta^{n-k} a \equiv a \pmod{k+1} \quad (7.29b)$$

Proof: See Appendix 7.11 (§ A7.11.3., p. 479).

The following lemma (proved in § A7.11.4., p. 483), is an elaboration on the last statement of the last theorem.

Lemma 7.3: Consider an (n,k,m) type-C code and let a denote the top element of the first column of its initial array (IA) and β ($1 < \beta \leq m+1$) be one of the 1st-coset columns. Then, the following are necessary conditions for the existence of such a code: *

$$1. \text{ If } s \hat{=} (\beta^{n-k-1}, k+1): a = i[(k+1)/s] \quad /i=1,2,\dots,s-1 \quad (7.30a)$$

$$2. \text{ If } r \hat{=} (a, k+1): \beta^{n-k} \equiv 1+i[(k+1)/r] \quad /i=0,1,\dots,r-1 \quad (7.30b)$$

$$3. \text{ If } (\beta, k+1) > 1: \quad n-k \leq (k+1)/(\beta, k+1) - 1 \quad (7.30c)$$

$$4. \text{ If } (\beta, k+1) = 1: \quad n-k \leq \phi(k+1) \quad (7.30d)$$

$$5. \text{ If } (a, k+1) = 1: \quad \text{Ord}_{k+1}(\beta) = n-k \quad /(\beta, k+1) = 1 \quad (7.30e)$$

* (a,b) denotes the greatest common divisor of a & b .

$$n-k \mid \Phi(k+1) \quad (7.30f)$$

$$(a_{j,1}, k+1) = 1 \quad /j=1,2,\dots,n-k \quad (7.30g)$$

7.6.2. Cyclically-Decodable Type-B_i Codes /i=1,2,3,4

In this paragraph the possibility of decoding cyclically a type-B_j self-orthogonal (SO) code will be discussed.

Consider a type-B₁ code. By Theorem 7.10, this is a $(2k, k, J-1)$ code, hence its IA has only one coset. Its first-column elements are the integers $1, 2, \dots, k$. Then, since by (7.30g) $(i, k+1) = 1$ / $i=1, 2, \dots, k$, $k+1$ must be a prime, p . As a consequence $(\beta, k+1) = (\alpha, k+1) = 1$, hence β must have order $n-k = k = p-1 \pmod{p}$. Since $\Phi(p) = p-1$ (see Theorem A7.1.15), β must be a primitive root \pmod{p} (whose existence is guaranteed by Theorem A7.3.3). Then, by Theorem 7.25, the 1st-column elements are $\alpha, \beta\alpha, \dots, \beta^{p-2}\alpha$. Finally, since there is only one coset, all the other columns will be cyclic shifts of the first one, hence, (see Definition 7.5) the code can be decoded cyclically.

Theorem 7.26: For every prime p and any $J \in [2, p-1]$, there exists a $(p-1, J)$ type-B₁ code* which is also a type-C code. The 1st column of its initial array is $\alpha, g\alpha, g^2\alpha, \dots, g^{p-2}\alpha$, where g is a primitive root \pmod{p} and $\alpha \in [1, p-1]$.

No other type-B₁ code, is also a type-C code.

The next theorem excludes all type-B₂, B₃ & B₄ codes.

Theorem 7.27: There is no type-B₂, B₃ or B₄ code, which is also a type-C code.

Proof: See Appendix 7.12 (§ A7.12.1., p. 484).

7.6.3. Cyclic Decoding of Type-B₄ Codes

Consider the decoding of the k type-B₄ code. By Theorem 7.22, this is a $J=2$ $(k+2, k, k-1)$ code. Hence its SYRE has

* See Theorem 7.10.

dimensions $(n-k) \times (m+1) = 2 \times k$ (see Fig. 7.1). From Theorem 7.1, syndrome bits $s_{\tau-1}^{(1)}$ / $\tau=1,2,\dots,k$, check on error bits $e_0^{(a[1,\tau])}$ / $\tau=1,2,\dots,k$, respectively. Since $a_{2,\sigma} = a_{1,k+1-\sigma}$ / $\sigma=1,2,\dots,k$, [by (A7.10.2)], it follows that syndrome bits $s_{\sigma-1}^{(2)}$ / $\sigma=1,2,\dots,k$ check on error bits $e_0^{(a[1,k+1-\sigma])}$ / $\sigma=1,2,\dots,k$, respectively.

The two syndrome bits, checking on a particular error bit, are located along the two rows of the SYRE. According to the above, $s_{\tau-1}^{(1)}$ & $s_{\sigma-1}^{(2)}$ check on the same error bit, if $\tau = k+1-\sigma$. Hence,

$$s_{\tau-1}^{(1)} \text{ \& } s_{k-\tau}^{(2)} \text{ check on } e_0^{(a[1,\tau])} \text{ / } \tau=1,2,\dots,k, \text{ respectively} \quad (7.31)$$

There are two basic ways to decode, the serial & the parallel. Strictly speaking, parallel decoding requires k MGs. From (7.31), for parallel decoding, the i th ($1 \leq i \leq k$) MG (used for the decoding of $e_0^{(a[1,i])}$) is connected to stages $(1,i)$ & $(2,k+1-i)$ of the SYRE (see Fig. 7.2).

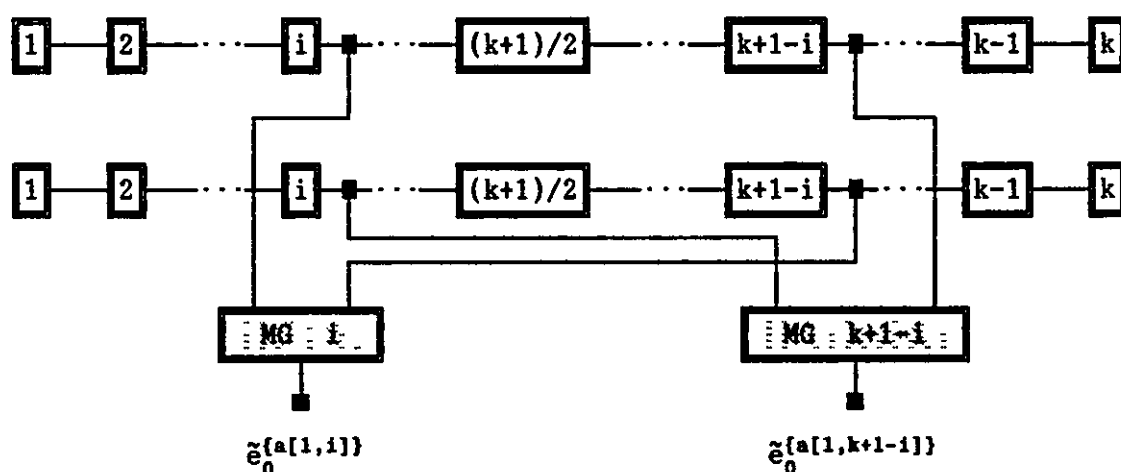


Figure 7.2: Parallel decoding of the k type-B4 code.

The idea behind type-C codes is to use fewer MGs, by performing non-parallel decoding. This can be achieved by exploiting some kind of cyclic structure in the IA.

Consider the i th & $(k+1-i)$ th MGs of Fig. 7.2. If the latter MG is dropped, after the decoding of $e_0^{(a[1,i])}$, one may shift $(2,i)$ to $(1,i)$ and $(1,k+1-i)$ to $(2,k+1-i)$ to decode $e_0^{(a[1,k+1-i])}$. For $i=1,2,\dots,(k-1)/2$, all error bits are decoded, except for $e_0^{(a[1,(k+1)/2])}$. This bit needs its own MG, which

will be used only once, before the vertical shift. Hence:

Theorem 7.28: A k type-B4 code can be decoded using $(k+1)/2$ majority gates (MGs). MG i [$1 \leq i \leq (k-1)/2$] is connected to stages $(1,i)$ & $(2,k+1-i)$ of the syndrome register (SYRE). Error bits $e_0^{[a[1,i]]}$ $/i=1,2,\dots,(k-1)/2$ are decoded in parallel from MG i $/i=1,2,\dots,(k-1)/2$, respectively. Error bits $e_0^{[a[1,k+1-i]]}$ $/i=1,2,\dots,(k-1)/2$ are decoded in parallel from MG i $/i=1,2,\dots,(k-1)/2$, respectively, after a vertical cyclic shift of the SYRE. Error bit $e_0^{[a[1,(k+1)/2]]}$ is decoded from MG $(k+1)/2$, with the first, or the second group. This gate is connected to the $[(k+1)/2]$ th stage of each of the two shift registers of the SYRE.

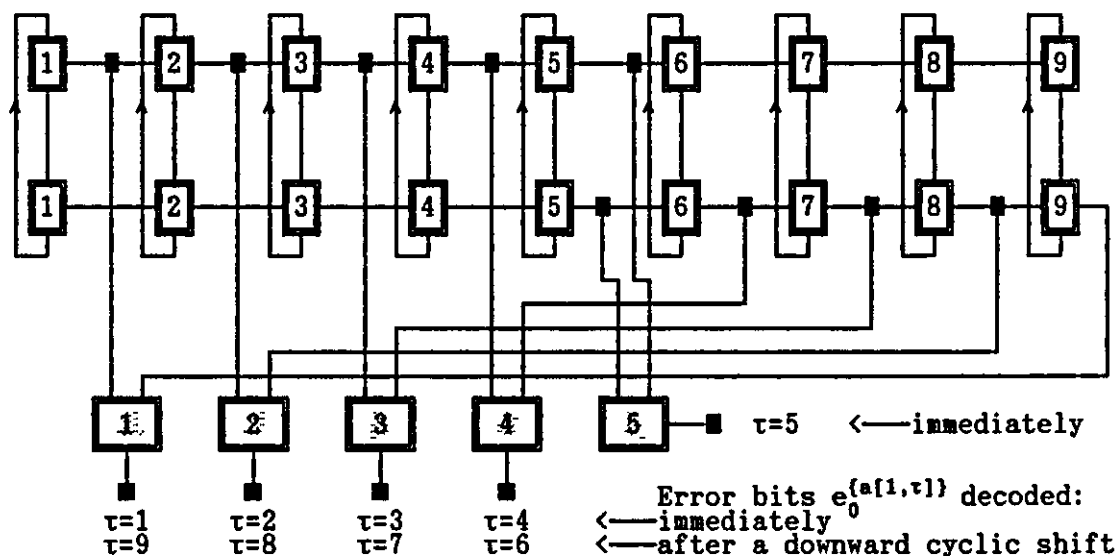


Figure 7.3: Vertically-cyclic decoding of the 9 type-B4 code.

Finally, the k type-B4 code can be decoded serially using only one MG, with inputs from stages $(1,1)$ & $(2,k)$. These stages decode $e_0^{[a[1,1]]}$ (see Fig. 7.3), but if the 1st SR is shifted leftward and the 2nd rightward, both by one position, then the MG will decode $e_0^{[a[1,2]]}$ (see Fig. 7.3), etc, while at the $(k-1)$ th shift the gate will decode $e_0^{[a[1,k]]}$.

Theorem 7.29: A k type-B4 code can be decoded serially using only one majority gate, which is connected to stages $(1,1)$ & $(2,k)$ of the syndrome register (SYRE). After the

decoding of the first bit, the top row of the SYRE is shifted cyclically leftward and the 2nd row cyclically rightward, both by one position, for the decoding of the 2nd bit. A complete cyclic shift decodes all bits.

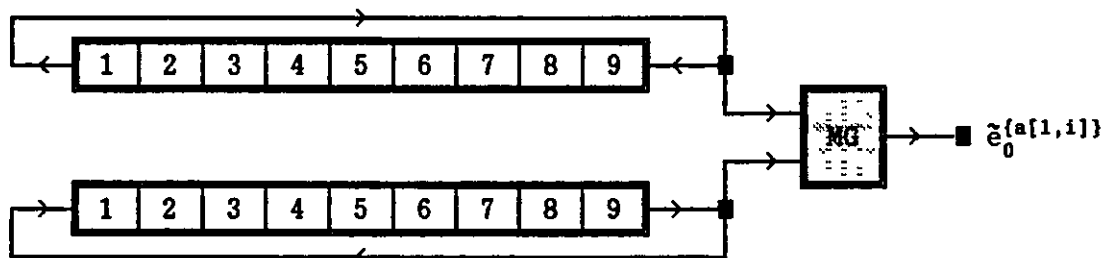


Figure 7.4: Horizontally-cyclic decoding of the 9 type-B4 code.

Note that the decoder of Fig. 7.4 is cyclic, but was not predicted by the work, so far, on type-C codes. This is so, because only uniform shifts were considered*. The decoder of the last figure does not use uniform shifts, because the two rows are shifted at opposite directions.

7.6.4. Type-C5 Codes

Let us consider now the possibility of cyclically decodable type-B5 codes.

Theorem 7.30: For every odd integer $k+1$, there exists a $(k+J, k, k-1)$ type-B self-orthogonal code [equivalently, a $(k+J, k)$ type-B5 code], which is also a type-C code if, and only if, there exists an integer $\beta \in [1, k]$, such that $a, a\beta, a\beta^2, \dots, a\beta^{J-1}$ are incongruent to each other modulo any non-trivial divisor, d , of $k+1$ and relatively prime to $k+1$. Such a code, if it exists, will be called the (k, J) type-C5 code. No other type-B5 code can also be a type-C code.

Proof: Assume that there exists $\beta \in [1, k]$ such that $a, a\beta, a\beta^2, \dots, a\beta^{J-1}$ are incongruent to each other modulo any non-trivial divisor, d , of $k+1$ and relatively prime to $k+1$. Then, by Theorem 7.21, there exists a $(k+J, k, k-1)$ type-B self-orthogonal code [which, by Theorem 7.23, is the $(k+J, k)$

* See Definition 7.4.

type-B5 code]. Furthermore, let the elements of the 1st column of its IA be $a, a\beta, a\beta^2, \dots, a\beta^{J-1}$.

Let us examine now if this code is also a type-C code. Let two columns x & y that have a common element in rows z & w , respectively. By the IA generation method* and the first-column elements, $a_{1,1} \equiv a\beta^1 \pmod{k+1}$:

$$a_{z,x} = a_{w,y} \longrightarrow xa_{z,1} \equiv ya_{w,1} \pmod{k+1} \longrightarrow$$

$$xa\beta^z \equiv ya\beta^w \pmod{k+1} \longrightarrow \text{(multiplying by } \beta\text{):}$$

$$xa\beta^{z+1} \equiv ya\beta^{w+1} \pmod{k+1}, \quad xa\beta^{z+2} \equiv ya\beta^{w+2} \pmod{k+1}, \text{ etc} \longrightarrow$$

If $a_{z,x} = a_{w,y}$, then: $a_{z+1,x} = a_{w+1,y}$, $a_{z+2,x} = a_{w+2,y}$, etc.

Hence if two columns have two equal elements then the one is a cyclic shift of the other. Since each row has k distinct elements (see Theorem 7.6), then there are exactly J columns with one common element, hence a cyclic shift of each other. Since there are k columns and $J|k$, then there are k/J such cosets. According to the discussion so far, on the relation between the SYRE and the IA, this code satisfies Definition 7.5, i.e. it is cyclically decodable.

Conversely, assume that for every odd integer $k+1$, there exists a $(k+J, k, k-1)$ type-B code, which is also a type-C code. This will have an IA with J rows and k columns. By Theorem 7.25 the only type-B codes that can also be type-C codes must have $a, a\beta, a\beta^2, \dots, a\beta^{J-1}$ as the first-column elements of the IA. Then, by Theorem 7.21, these J integers are relatively prime to $k+1$ and incongruent to each other $\pmod{k+1}$.

QED

It is necessary to elaborate on the above equivalent conditions and to obtain a solution, which will also verify the existence of the type-C5 codes.

Since it is desirable to obtain all type-C5 codes, the equivalent conditions on J must also be deduced. The following number-theoretic function will assist in obtaining the range of J . The author is not aware of any reference on it,

* See Definition 7.2 (p. 185).

in number theory, hence it can be claimed that it is a new and, possibly, widely useful function.

Definition 7.6: Given any integer m , with prime factors p_1, p_2, \dots, p_r , the *theta function* is denoted by $\theta(m)$ and is defined to be:

$$\theta(m) \triangleq \gcd(p_1-1, p_2-1, \dots, p_r-1) \quad (7.32)$$

Theorem 7.31: For every odd integer $k+1$, and every integer $J \geq 2$ such, and only such, that $J \mid \theta(k+1)$, there exists a (k, J) type-C5 code if, and only if, there exists $\beta \in [1, k] / (\beta, k+1) = 1$, such that $\text{Ord}_d(\beta) = J$, for every non-trivial divisor d , of $k+1$. The first-column elements of the initial array are then $a, a\beta, a\beta^2, \dots, a\beta^{J-1}$, where $a \in [1, k] / (a, k+1) = 1$. It will be said that a & β *generate** the (k, J) type-C5 code.

Proof: See Appendix 7.12 (§ A7.12.2., p. 486).

Hence, the existence of this class of codes depends on the existence of β . Thankfully (!) there is at least one solution, for β :

Theorem 7.32: For every odd integer $k+1$ and every integer $J \geq 2$, such that $J \mid \theta(k+1)$, there exists β , such that $\text{Ord}_d(\beta) = J$ for every non-trivial divisor d , of $k+1$. This β is given by:

$$\beta \equiv \sum_{i=1}^r g_i^{f(i)/J} \left[(k+1)/p_i^{a(i)} \right]^{f(i)} \pmod{k+1} \quad (7.33a)$$

$$\text{where:} \quad f(i) \triangleq p_i^{a(i)-1}(p_i-1) \quad / i=1, 2, \dots, r \quad (7.33b)$$

$$k+1 = \prod_{i=1}^r p_i^{a(i)} \quad / a(i) \geq 1, i=1, 2, \dots, r \quad (7.33c)$$

and, for $i=1, 2, \dots, r$:

$$g_i \triangleq \text{primitive root (mod } p_i): g_i^{p_i-1} \not\equiv 1 \pmod{p_i^2} \quad (7.33d)$$

Proof: See Appendix 7.12 (§ A7.12.3., p. 488).

* If there is such a β .

Hence, the last theorem guarantees the existence of type-C5 codes for any $k=\text{even}$ and any $J \mid \theta(k+1)$. This is the class of codes discovered by McQuilton and named 'cyclic' CSOCs. The following lemma elaborates on the above theorem, for the three most common cases.

Lemma 7.4: For every odd integer $k+1$, there exists a (k,J) type-C5 code generated by:

$$\text{i) If } k+1 = p, \text{ then } \beta \equiv g^{(p-1)/J} \pmod{p} \quad (7.34a) \\ \& J \mid p-1, \text{ where: } g \hat{=} \text{prim.root} \pmod{p}.$$

$$\text{ii) If } k+1 = p^a, \text{ then } \beta \equiv g^{f/J} \pmod{p^a} \quad (7.34b) \\ \text{and } J \mid p-1, \text{ where: } g^{p-1} \not\equiv 1 \pmod{p^2}, \\ g \hat{=} \text{prim. root} \pmod{p} \& f \hat{=} p^{a-1}(p-1).$$

$$\text{iii) If } k+1 = p_1 p_2, \text{ then } J \mid (p_1-1, p_2-1) \& \\ \beta \equiv g_1^{(p_1-1)/J} p_2^{p_1-1} + g_2^{(p_2-1)/J} p_1^{p_2-1} \pmod{p_1 p_2} \quad (7.34c) \\ \text{where: } g_i \hat{=} \text{primitive root} \pmod{p_i}.$$

In Appendix 7.12 (§ A7.12.4., p. 491), there are some examples of type-C5 codes. In particular, the IAs of the (22,11), the (24,4) & the (64,4) type-C5 codes are given, and the calculation of their β is illustrated. These codes are (33,22,21) (rate-2/3), (28,24,23) (rate-6/7) & (68,64,63) (rate-16/17) type-B self-orthogonal codes, respectively. Also, the (18,6) type-C5 code is considered (see Example A7.12.4., p. 493), together with the connections from the syndrome register to the three majority gates. The arrangement is such that the 18 bits are decoded in six steps, but in an 'arbitrary' order, as can be seen below:

$$\begin{array}{llllll} \text{From MG1:} & \tilde{e}_0^{(8)} & \tilde{e}_0^{(7)} & \tilde{e}_0^{(18)} & \tilde{e}_0^{(11)} & \tilde{e}_0^{(12)} & \tilde{e}_0^{(1)} \\ \text{From MG2:} & \tilde{e}_0^{(16)} & \tilde{e}_0^{(14)} & \tilde{e}_0^{(17)} & \tilde{e}_0^{(3)} & \tilde{e}_0^{(5)} & \tilde{e}_0^{(2)} \\ \text{From MG3:} & \tilde{e}_0^{(13)} & \tilde{e}_0^{(9)} & \tilde{e}_0^{(15)} & \tilde{e}_0^{(6)} & \tilde{e}_0^{(10)} & \tilde{e}_0^{(4)} \end{array}$$

It is 'natural' to require that the error bits are produced in 'order'. The 'order' should be such that the output serial bit-stream is $\tilde{u}_h^{(1)}, \tilde{u}_h^{(2)}, \dots, \tilde{u}_h^{(k)}$. Hence it would make

sense if, in the above example, MG1, MG2 & MG3 were to produce $\tilde{e}_h^{(1)}$, $\tilde{e}_h^{(2)}$ & $\tilde{e}_h^{(3)}$, respectively, at the first time-unit, $\tilde{e}_h^{(4)}$, $\tilde{e}_h^{(5)}$ & $\tilde{e}_h^{(6)}$, respectively, at the second time-unit, etc. This may be achieved if the message bits are suitably transposed, prior to entering the encoder, so that they appear in their 'natural' order at the output of the decoder. For the case of the above code, the transposition (or mapping) should be:

8	→	1	16	→	2	13	→	3
7	→	4	14	→	5	9	→	6
18	→	7	17	→	8	15	→	9
11	→	10	3	→	11	6	→	12
12	→	13	5	→	14	10	→	15
1	→	16	2	→	17	4	→	18

Note that this corresponds to adopting the same mapping for the 1st, 2nd & 4th columns of the IA (the three coset leaders) & hence for the rest of the IA. It is necessary, therefore, to use a different array for encoding:

Definition 7.7: Let $k+1$ be any odd integer, and $J \geq 2$ any integer such that $J \mid \theta(k+1)$. Consider the initial array (IA) corresponding to the (k, J) type-C5 code and let columns $c_1, c_2, \dots, c_{k/J}$ be the coset leaders. Then, the *encoding array* (EA), corresponding to the given IA, is a $J \times k$ array of integers $b(x, y) \mid 1 \leq b(x, y) \leq k, 1 \leq x \leq J \text{ \& \& } 1 \leq y \leq k$, defined by the following mapping:

$$b(x, c_z) \hat{=} z + (x-1)k/J \quad / x=1, 2, \dots, J \text{ \& \& } z=1, 2, \dots, k/J \quad (7.35)$$

The rest of the elements of the EA follow the above-defined mapping, so that the cyclic structure of the IA is passed into the EA.

Note that McQuilton [42] used a different mapping for the EA: $b(x, c_z) \hat{=} x + (z-1)J$. It seems though that this mapping results in a more unfavourable demand on the decoder. Since there is some kind of parallel decoding (in the case of Example A7.12.4, three bits are decoded at each time-unit), there must be a kind of parallel-to-serial conversion involved. One would like to keep the cost of the convertor down. If MG1 outputs $\tilde{e}_0^{(1)}, \tilde{e}_0^{(2)}, \dots, \tilde{e}_0^{(6)}$, as suggested by

McQuilton, then the output of MG2 must be delayed by six time-units, while the output of MG3, by 12 time-units. Hence, there is a need for a store of $6+12=18$ bits. On the other hand, if mapping (7.35) is used, the store needed is 1 for MG2 and 2 for MG3. *

In general, a (k,J) type-C5 code uses k/J MGs, each of which decodes J bits. With McQuilton's mapping MG2 needs a delay of J , MG3 a delay of $2J$, etc, hence the total delay is $(1+2+\dots+k/J-1)J$ stages. With mapping (7.35) the corresponding figure is $(1+2+\dots+k/J-1)$ stages, i.e. J times less.

7.7 PROPERTIES OF THE INITIAL ARRAY

The following theorems present some of the properties of the IA of type-C5 codes:

Theorem 7.33: For any (k,J) type-C5 code, generated by α & β [given by (7.33a)], with $J = \text{even}$:

$$\beta^{J/2} \equiv k \pmod{k+1} \quad (7.36)$$

Proof: See Appendix 7.14 (§ A7.14.1., p. 497). ■

Theorem 7.34: For any (k,J) type-C5 code, generated by $\alpha=1$ & β [given by (7.33a)], if $a_{x,z}$ $/x=1,2,\dots,J$ & $z=1,2,\dots,k$ are the elements of its initial array:

$$a_{J,z} = z \quad /z=1,2,\dots,k \quad (7.37a)$$

$$\text{For } J = \text{even:} \quad a_{J/2,z} = k+1-z \quad /z=1,2,\dots,k \quad (7.37b)$$

$$a_{x,z} + a_{x+J/2,z} = k+1 \quad /x=1,2,\dots,J/2 \text{ \& } z=1,2,\dots,k \quad (7.37c)$$

$$\sum_{x=1}^J a_{x,z} = (k+1)J/2 \quad /z=1,2,\dots,k \quad (7.37d)$$

Proof: See Appendix 7.14 (§ A7.14.2., p. 498). ■

Theorem 7.35: For any (k,J) type-C5 code, generated by $\alpha=1$ & β [given by (7.33a)], if $a_{x,z}$ $/x=1,2,\dots,J$ & $z=1,2,\dots$

* It all depends on the decoder configuration.

,k are the elements of its initial array:

$$\sum_{x=1}^J a_{x,z} \equiv 0 \pmod{k+1} \quad /z=1,2,\dots,k \quad (7.38a)$$

$$a_{x,z} + a_{x,k+1-z} = k+1 \quad /x=1,2,\dots,J \text{ \& } z=1,2,\dots,k \quad (7.38b)$$

$$\sum_{x=1}^J a_{x,z} + \sum_{x=1}^J a_{x,k+1-z} = J(k+1) \quad /z=1,2,\dots,k \quad (7.38c)$$

Proof: See Appendix 7.14 (§ A7.14.3., p. 499). ■

The examples, below, will help clarify the above results:

Example 7.7: Consider the (22,11) type-C5 code of Example A7.12.1 (p. 491). For this code, $k+1=23$ & $J=11=\text{odd}$. Note that $a_{11,z} = z$ $/z=1,2,\dots,22$ [as predicted by (7.37a)]. Consider $C(z)$, the sum of the elements of column z :

$$C(1) = 2+4+8+16+9+18+13+3+6+12+1 = 92 = 4 \times 23$$

$$C(5) = 10+20+17+11+22+21+19+15+7+14+5 = 161 = 7 \times 23$$

All the other columns will sum to one of the above two results, because there are two cosets, and 1 & 5 are their coset leaders. For instance, $C(3)+C(23-3) = C(3)+C(20) = (4+7) \times 23 = 11 \times 23 = J \times (k+1)$ [as predicted by (7.38c)]. ■

Example 7.8: Consider the (24,4) type-C5 code of Example A7.12.2 (p. 492). For this code, $k+1=25$ & $J=4$. Note that $a_{1,z}+a_{3,z} = a_{2,z}+a_{4,z} = 25 = k+1$ $/z=1,2,\dots,24$ [(7.37c)]. It is also easy to verify all results of Theorems 7.33 & 7.34. For instance, $C(1) = C(2) = \dots = C(24) = 50 = 25 \times 2 = (k+1)J/2$. ■

The next theorem relates the elements of the IA with quadratic residues*, for $J=\text{odd}$ codes.

Theorem 7.36: Consider any (k,J) type-C5 code, with $J=\text{odd}$, generated by $\alpha=1$ & β [given by (7.33a)]. For every prime factor, p_i , of $k+1$, a column of the initial array contains either multiples of p_i , or quadratic residues, or

* See Appendix 7.13 (p. 495), for an introduction to quadratic residues.

quadratic nonresidues (mod p_i). The first column contains always J distinct quadratic residues, modulo any p_i .

Proof: See Appendix 7.14 (§ A7.14.4., p. 500). ■

In Example A7.14.1 (§ A7.14.4., p. 502), the (340,5) type-C5 code illustrates the validity of the above results.

7.8 EFFECTIVE CONSTRAINT-LENGTH

The results of the previous section will be used now for the determination of the n_e of the type-C5 codes.

Theorem 7.37: If $a_{x,z}$ $/x=1,2,\dots,J$ & $z=1,2,\dots,k$ denote the elements of the initial array (IA) of the (k,J) type-C5 code, then the effective constraint-length, $n_e(i)$, corresponding to the feedback decoding of $e_h^{(i)}$, is given by:

$$n_e(i) = 1 + \sum_{j=1}^J a_{j,z} = 1 + (k+1)q(z) \quad (7.39)$$

where IA element $a_{1,z}$ is mapped into encoding array (EA) element i , via mapping (7.35) (p. 220) and $q(z)$ = integer, defined by (7.39).

Proof: According to Theorem 7.9, if

$$a[1,z_1] = a[2,z_2] = \dots = a[J,z_J] = a \quad (A)$$

$$\text{then,} \quad n_e(a) = 1 + z_1 + z_2 + \dots + z_J \quad (B)$$

Note that a refers to the IA, while the codes are encoded using the EA. Let then $a \longrightarrow i$, via mapping (7.35). In (B), z_j $/j=1,2,\dots,J$ are the column numbers of the columns that contain a . Then, all these columns belong to the same coset (see Theorem 7.25). Furthermore, from (7.37a), $a[J,z_j] = z_j$, $j=1,2,\dots,J$, hence the column numbers are also elements of the columns and hence of the coset. Since they are distinct, they are also the elements of the coset. Then, the sum of the column numbers equals the sum of the elements of any of these columns. Let $z \hat{=} z_1$ and use the fact that $a \longrightarrow i$, in

$$(B): n_z(i) = 1 + a_{1,z} + a_{2,z} + \cdots + a_{J,z}.$$

Finally, from (7.38a), the sum of the elements of a column is a multiple [say, $q(z)$] of $k+1$.

QED

Theorem 7.38: For any (k,J) type-C5 code, for $J = \text{even}$:

$$\text{For all } i=1,2,\dots,k: n_z(i) = n_z = 1 + (k+1)J/2 \quad (7.40)$$

Proof: The result follows from Theorem 7.37 & eqn (7.37d). ■

The above result was deduced, independently by the author, 6 months before its publication by McQuilton ([47])^{*}.

A closed-form expression for n_z , for $J = \text{odd}$, is impossible, for at least some cases, except for $J = 3$. The following theorem introduces upper & lower bounds.

Theorem 7.39: For any (k,J) type-C5 code, with $J = \text{odd}$:

$$n_z = 1 + (k+1)\text{MAX}_z\{q(z)\} = 1 + (k+1)\left[J - \text{MIN}_z\{q(z)\}\right] \quad (7.41)$$

$$1 + (k+1)(J+1)/2 \leq n_z \leq 1 + (k+1)(J-1) \quad (7.42)$$

$$\text{In particular, for } J = 3: n_z = 1 + 2(k+1) \quad (7.43)$$

Proof: See Appendix 7.15 (§ A7.15.1., p. 504). ■

Note that these bounds are as tight as such a bound can be without taking into account the particular properties of k & J . To illustrate this, consider the following examples:

Example 7.9: Let $k+1 = 67$ (= prime). Since $\theta(67) = 66$, $J = 2, 3, 6, 11, 22, 33, 66$. Let $J = 11$. From TABLE A7.3.1, $g=2$ is a primitive root (mod 67), and from Lemma 7.4, $\beta \equiv 2^{66/11} \equiv 2^6 \equiv 64 \pmod{67}$. The $q(z)$ s for the coset leaders (there are $66/11 = 6$ cosets) are: $q(1) = q(6) = q(7) = 5$ and $q(2) = q(3) = q(4) = 6$. Then $\text{MAX}\{q(z)\} = 6$, and from (7.41): $n_z = 1 + 67 \times 6 = 403$. The bounds are $1 + 67 \times 6 \leq n_z \leq 1 + 67 \times 10$ —> $403 \leq n_z \leq 671$. Hence, it meets the lower bound. ■

* At that time the result did not seem to be worthy of a publication.

Example 7.10: Let $k+1 = 71$ (= prime). Since $\theta(71) = 70$, $J = 2, 5, 7, 10, 14, 35, 70$. Let $J = 5$. From TABLE A7.3.1, $g=7$ is a primitive root (mod 71), and from Lemma 7.4, $\beta \equiv 7^{70/5} \equiv 7^{14} \equiv 54 \pmod{71}$. The $q(z)$ s for the coset leaders (there are $70/5 = 14$ cosets) are: $q(3) = 1$, $q(1) = q(2) = q(6) = q(7) = q(9) = q(18) = 2$, $q(11) = q(13) = q(14) = q(21) = q(22) = q(27) = 3$ and $q(42) = 4$. Then $\text{MAX}\{q(z)\} = 4$, & from (7.41): $n_{\mathbf{z}} = 1 + 71 \times 4 = 285$. Also, $1 + 71 \times 3 \leq n_{\mathbf{z}} \leq 1 + 71 \times 4 \implies 214 \leq n_{\mathbf{z}} \leq 285$. Hence, it meets the upper bound. ■

Note that, since $q(z) + q(k+1-z) = J$ $/z=1, 2, \dots, k$ [see (A7.15.1)], if for some column w , $q(w)=1$ then $q(k+1-w)=J-1$, which is the upper bound of $q(z)$ [see (A7.15.2)], hence $n_{\mathbf{z}} = 1 + (k+1)(J-1)$. This was the case with the last example, where $q(3) = 1$.

The following theorem is the last result that could be obtained on $n_{\mathbf{z}}$, for $J = \text{odd}$.

Theorem 7.40: Consider the $(p-1, (p-1)/2)$ type-C5 code, where p is any odd prime. If $p \equiv 3 \pmod{4}$, the code effective constraint-length, $n_{\mathbf{z}}$, equals one plus the sum of the quadratic nonresidues (mod p)*. A closed-form expression for $n_{\mathbf{z}}$ is equivalent to solving one of number theory's unsolved problems.

Proof: See Appendix 7.15 (§ A7.15.2., p. 505). ■

Of course, research may reveal more results, but it seems that the above theorem is a deterrent. This is so because (by Theorem 7.36) the sum of the elements of each column of the IA, is the sum of some quadratic residues, or nonresidues, or some multiples of a prime factor of $k+1$. Thus, the problem is made even more difficult, than above, for $R > 2/3$.

* For $(p-1)/2 = \text{even}$, $n_{\mathbf{z}} = 1 + p(p-1)/4$ - see Theorem 7.38.

7.9 EXISTENCE THEOREMS FOR 'CYCLIC' CSOCs

One important characteristic of the above class of codes is the infinite number of its membership. It would be useful, therefore, if a method is given for the construction of a code with one given parameter (J, R, n, k). The next theorem will facilitate the choice of the other code parameters.

Theorem 7.41: For any (k, J) type-C5 code, if $c \hat{=} k/J$:

$$R = c/(c+1) \quad (7.44a)$$

$$n = (c+1)J \quad (7.44b)$$

$$n_A = c(c+1)J^2 \quad (7.44c)$$

$$* \quad 1+(cJ+1)\lceil J/2 \rceil \leq n_E \leq 1+(cJ+1)(J-1) \quad (7.44d)$$

$$** \quad 1/(1-R) = c+1 \leq n_A/n_E \leq 2(c+1) = 2/(1-R) \quad (7.44e)$$

$$\text{For } J = \text{even}, (J/2)/n_E \approx 1/(cJ) = 1/k \quad (7.44f)$$

Proof: See Appendix 7.16 (p. 508). ■

Consider any odd integer $k+1$, and let $p_1 < p_2 < \dots < p_r$ be its prime factors. According to Theorem 7.31, J must divide all $p_i - 1$ $/i=1, 2, \dots, r$. Hence, there must exist integers q_i such that $p_i = Jq_i + 1$ $/i=1, 2, \dots, r$. So, if J is given, the prime factors of $k+1$ must be of the form $Jq_i + 1$ $/q_i=1, 2, 3, \dots$. Since $p_i = Jq_i + 1 = \text{odd}$, then $Jq_i = \text{even}$, and if $J = \text{odd}$, then by necessity $q_i = \text{even}$. There are no restrictions on the exponent of each p_i (in the prime factorization of $k+1$). Finally, if $J = 2$, because $2 \mid \theta(k+1)$, for all $k+1 = \text{odd}$, there is always a $(k, 2)$ type-C5 code. Hence:

Theorem 7.42: For every $k+1 = \text{odd}$, there exists a $(k, 2)$ type-C5 code. Its parameters are $R = c/(c+1)$, $n_A = 4c(c+1)$ & $n_E = 2(c+1)$, where $c \hat{=} k/2$. Given $J > 2$, a (k, J) type-C5 code is obtained by finding primes of the form $p = qJ + 1$ $/q = \text{integer}$ & $q = \text{even}$ if $J = \text{odd}$. Then any of the above primes can be a prime factor of $k+1$, each raised to any integer power. ■

Example 7.11: Let $J = 6$. Then $p = 6q + 1$ $/q = \text{integer}$. From the set $\{7, 13, 19, 25, 31, 37, 43, \dots\}$, all but 25 are primes. Let us select 7 & 19. Then $k+1 = 7^a \times 19^b$, for any positive

* If $J = \text{even}$, the effective constraint-length always meets the lower bound.

** If $J = \text{even}$, the ratio always meets the upper bound - bounds are approximate.

integers a & b . Obviously, $J = 6 \mid \theta(k+1) = (6, 18) = 6$.

Let $J = 7$. Then $p=7q+1$ / $q=\text{even}$. From $\{15, 29, 43, 57, 71, \dots\}$, all but 15 & 57 are primes. If 29 & 71 are selected, $k+1 = 29^a \times 71^b$, for any positive integers a & b . Note that $J = 7 \mid \theta(k+1) = (28, 70) = 14$.

The case of given k is covered by Theorem 7.31 (p. 218), which introduced this class of codes. The case of given rate is equivalent to the case of given k/J [see (7.44a)], but a useful construction-method is very difficult (if not impossible) to obtain. The same applies to the case of given code-length, n . These last two cases will be covered in the next chapter, by computer-search programmes.

7.10 CONCLUSIONS

McQuilton [42] discovered a class of $(n, k, k-1)$ systematic self-orthogonal (SO) codes which can be decoded cyclically using $k/(n-k)$ majority gates. His work, which concentrated on the proof of their existence, was studied and it was concluded that the class of codes he discovered is based on three 'ideas': Each syndrome bit checks on exactly one error bit from each block; the error-bit numbers of each block are related via a congruence (mod $k+1$); and the codes are cyclically decodable.

In this chapter, the widest possible approach was followed. The aim was to illustrate the process of systematically designing SO codes, by obtaining the necessary & sufficient conditions so that a general (n, k, m) systematic convolutional code is SO. Since a general solution is not feasible, a minimum number of restrictions were to be imposed, in stages, and only when confronted with a seemingly unsolvable problem.

The effort started with an 'alternative' representation of the codes, via the so-called *initial array* (IA). This is an $(n-k) \times (m+1)$ array of cells with integers in the range $[1, k]$, which represent the error bits checked by each syn-

drome bit. Cell (j,i) contains the numbers of the error bits, from $(e^a)_{h-1+i}$, checked by $s_h^{(j)}$. Then, the problem is reduced to determining these integers, so that the code is SO. Because the IA was very complicated, the 'population' of each cell was restricted to one. These codes (with each syndrome bit checking exactly one error bit from each block) were named *type-A codes* and the necessary & sufficient conditions for self-orthogonality were deduced (Theorem 7.2). Other possible restrictions on the cell population, are up to one element/cell, two elements/cell, etc.

A restriction placed on the type-A codes, generates the *type-B codes*. Specifically, if the j th element of the IA is $a_{j,1}$, then, $a_{j,1} \equiv ia_{j,1} \pmod{k+1}$. In this way, the problem is reduced to that of obtaining the 1st column of the IA. Theorem 7.3 proved that for an (n,k,m) type-B code, it is necessary that $m \leq (k+1)/(k+1, a_{j,1}) - 2$, for $j=1,2,\dots,n-k$.^{*} Finally, the corollary of Theorem 7.5 gave the necessary and sufficient conditions, on the 1st column of the IA, so that the code is SO. Thereafter, some new classes of SO codes were discovered.

Type-B1 codes are $(2k,k,J-1)$ type-B SO codes with exactly J syndromes checking on each error bit. These codes exist for any even k and any $J \in [2,p-1]$, where p is the smallest prime factor of $k+1$. Theorem 7.11 proved that they also meet the lower bound on the effective constraint-length.

Other classes of codes with $m < p-1$, but of rate $> 1/2$, were obtained by deleting rows from the IA of type-B1 codes (see Theorem 7.15). Unfortunately, the conditions were too general for a clear picture, about the 'quality' of these codes, to emerge. For this reason, some special cases were considered. *Type-B2 codes* are $(p^2-1+J(p+1), p^2-1, p-2)$ type-B SO codes, with exactly J syndromes checking on each error bit, where p is any odd prime and $J \in [2,p-1]$ (see Theorem 7.17). *Type-B3 codes* are $(pq-1+J(q+2), pq-1, p-2)$ type-B SO codes with at least J syndromes checking on each error bit, where p & q are any odd primes, with $p < q < 2p$ and $2 \leq J \leq (q-1)/2$ (see Theorem 7.18). Instructions for the construction of the IA, for each of these three classes, are also provided. Theorems 7.19 & 7.20 make some progress towards obtaining SO type-B

* (a,b) denotes the greatest common divisor of a & b .

codes, with $m \geq p-1$, but no concrete general results were obtained. Nevertheless, some SO codes were discovered, with the assistance of a simple computer programme.

The last effort on type-B codes was for $m=k-1$, and resulted in two more classes of codes. *Type-B4 codes* are $(k+2, k, k-1)$ type-B SO codes with exactly two syndromes checking on each error bit, where k is any odd integer (see Theorem 7.22). *Type-B5 codes* are $(n, k, k-1)$ type-B SO codes, with exactly $J=n-k$ syndromes checking on each error bit, where k is any even integer and $n \in [k+1, k+p-1]$ (p is the smallest prime factor of $k+1$) (see Theorem 7.23).

Type-C codes were defined to be any type-B codes which are cyclically decodable (see Definition 7.5). Necessary conditions on the code parameters and the elements of the first column of the IA were derived. These codes are generated by two integers, α & β , which are relatively prime to $k+1$ and less than $k+1$; (β has to satisfy some other conditions, as well - see Theorem 7.25). These results were used to test the various classes of type-B codes, for 'cyclic decodability'. The only type-B1 codes which satisfied the type-C requirements are those with $k+1=\text{prime}$ (see Theorem 7.26). It was proved also that type-B2, B3 & B4 codes cannot be type-C codes (see Theorem 7.27). It was shown, though, that there were alternative techniques for cyclically decoding the type-B4 codes (see Figs 7.2, 7.3 & 7.4).

Type-C5 codes are (k, J) type-B5 codes, which are cyclically decodable, using k/J majority gates, where k is any odd integer and $J \mid \theta(k+1)$. $\theta(k+1)$ is a number-theoretic function, introduced by the author (see Definition 7.6). Hence, there are type-B5 codes which are not type-C [those with $J \leq \theta(k+1)$, but $J \nmid \theta(k+1)$]. Theorem 7.32 provides instructions for the construction of the IA.

McQuilton's codes are the type-B4 codes and the type-C5 codes (with $\alpha=1$).

Finally, it was illustrated that the IA has to be mapped to the *encoding array* (EA), so that the bits are decoded in their natural order (see Definition 7.7). This was proposed by McQuilton, but it was shown that his mapping would result in an unfavourable demand on the memory of the parallel-to-

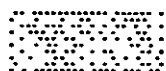
serial converter of the decoder. Another mapping was proposed instead, which requires J times less storage.

A number of properties of the IA of type-C5 codes (with $\alpha=1$), some of them new, were also proved. So, the sum of the elements of any column is a multiple of $k+1$, while the sum of the elements of any two columns z & $k+1-z$ is $J(k+1)$. The most important conclusion, though, is that the IA is much more 'predictable', for $J = \text{even}$, than it is for $J = \text{odd}$. So, for $J = \text{even}$ only, it was proved that the sum of the elements of any column is $(k+1)J/2$. On the other hand, for $J = \text{odd}$, the last result is not valid. Finally, Theorem 7.36 linked the IA with quadratic residues. So, the elements of the first column of the IA are quadratic residues modulo any prime factor, p , of $k+1$. The elements of the rest of the columns have the same quadratic character*, $(z|p)$, as the column number z , hence they are either multiples of p , or quadratic residues, or nonresidues (mod p), for any $p \mid k+1$.

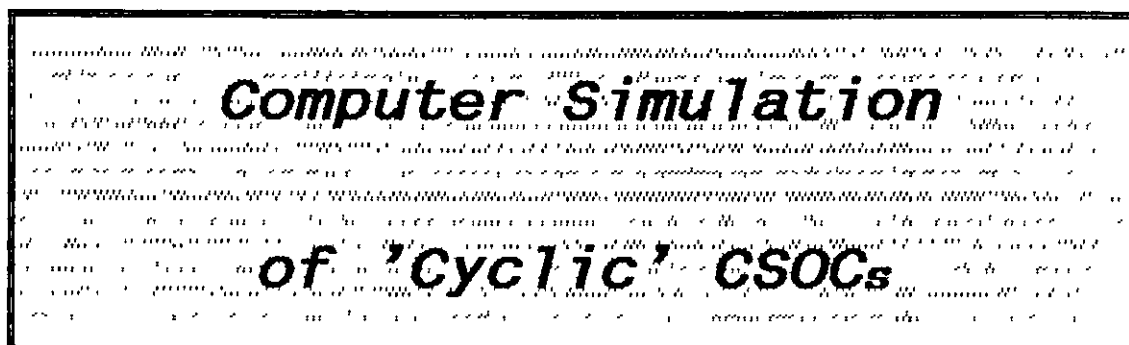
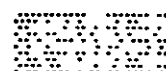
The above results were used to obtain expressions for the effective constraint-length of the type-C5 codes. For the reasons mentioned above, this was possible for the $J = \text{even}$ codes [$n_{\text{e}} = 1+(k+1)J/2$], but impossible for the $J = \text{odd}$ codes, exactly because the sum of the elements of a column of the IA is not known. Of course, one is entitled to ask if this is not known to the author, or if in general it is a result impossible to obtain. Again, there is no proof either for, or against, but for a special case. It was proved that for rate $2/3$ type-C5 codes, with $k+1 = p = \text{prime}$ & $J = \text{odd}$, $n_{\text{e}}-1$ equals the sum of quadratic nonresidues (mod p). This implies that any closed-form expression for $n_{\text{e}}-1$, would also solve an unsolved number-theoretic problem. It seems also that for the rest of the $J = \text{odd}$ cases, the problem will be even more difficult to tackle. For the $J = \text{odd}$ cases, bounds on n_{e} were obtained, which were as tight as they can be. The lower bound coincides with the n_{e} for the $J = \text{even}$ codes, hence the latter are at least as good as the $J = \text{odd}$ codes.

Concerning the quality of $J=\text{even}$ codes, $n_{\text{A}}/n_{\text{e}} \approx 2/(1-R)$, while $(J/2)/n_{\text{e}} \approx 1/k$. So, the rate- $1/2$ codes are the 'best' (with $n_{\text{A}}/n_{\text{e}} \approx 4$), while shorter codes have a better relative error-correcting capability ($\approx 1/k$) (see Theorem 7.41).

* The quadratic character is defined in Appendix 7.13.



CHAPTER 8



Chapter 8 concludes this thesis with the results of the computer simulation of the type-C5 codes.

Because a great number of codes was to be simulated, it was decided that a separate computer programme for each one of them would have been an exercise prone to errors, as well as a tedious one. Hence, a separate library of routines was built which, given the code parameters k & J , generated and stored in arrays all the data necessary for the encoding and decoding of the code (See Appendix 8.1). As a consequence, the simulation programmes were designed for the general (k, J) type-C5 code (see Section 8.2). Also, a number of sub-routines had to be designed for the monitoring, processing and presentation of the results of simulation. Section 8.3 discusses the choice of performance data and their statistical confidence. Section 8.4 presents & discusses the code-performance data (net coding-gain & error-extension ratio) under feedback-decoding and with the use of the nominal syndrome threshold. The gains obtained from the use of the optimum threshold are discussed in Section 8.5. The error-propagation effect is analysed in Section 8.6, while the last section looks at the unequal error-protection performance of the codes.

8.1 COMPUTER GENERATION OF 'CYCLIC' CSOC.

The decoding of a (k,J) type-C5 code is done via the encoding array (EA) and the syndrome array (SA), both of which are generated by the initial array (IA). The latter is generated by α & β , of which α is taken to be 1, while β is calculated from eqn (7.33a). This requires, the prime decomposition of $k+1$, the primitive roots of the prime factors of $k+1$ and the ability to raise an integer to an exponent and then obtain the least positive residue modulo some integer m , without overflow. The last requirement also implies the ability to form the product of two integers and reduce them $(\text{mod } m)$ and this, in turn, to form the sum of two integers and reduce them $(\text{mod } m)$, without overflow. Also, a routine for the calculation of the greatest common divisor between two integers is required, in order to calculate $\theta(k+1)$.

Once the above-outlined number-theoretic library is developed, the generation of the IA, EA & SA becomes a straightforward task. The basic *flow-charts* of the number-theoretic routines are given in Appendix 8.1, while the associated FORTRAN programmes in Appendix 8.2.

The routine which returns the least positive residue of $\alpha^\beta \pmod{m}$, without overflow, is the most useful of the library. It was designed to operate efficiently over all integers $\alpha, \beta, m \in [1, \text{MAXIN}]^*$. From the rest, the one returning the smallest primitive root $(\text{mod } m)$, for $m \in [1, \text{MAXIN}]$, is the most sophisticated.

8.2 CHANNEL SIMULATION & S/W IMPLEMENTATION OF THE DECODER

The codes will be tested over the AWGN channel, with a binary PSK modulator and coherent demodulation with hard decisions. This model is briefly discussed in § 1.1.2. **

The objective of the simulation is the counting of bit decoding-errors, at various signal-to-noise power ratios, Γ . According to statement (A1.2.13) an error occurs, if " $s_0(t)$ is transmitted and $n_c < -E$, or $s_1(t)$ is transmitted and $n_c \geq +E$ ". E is the received energy per bit and n_c is a zero-mean

* MAXIN denotes the maximum integer of the computer.

** See, also, Appendix 1.2 (p. 285).

Gaussian random variable with variance $\sigma^2 = E\tilde{n}/2$, where $\tilde{n}/2$ is the double-sided noise power spectral density. From (A1.2.14), the probability of error, P_e , is $P(n_c > E)$. For the purpose of collecting results about errors, it seems obvious then that generality is not lost if only $s_1(t)$ is considered. Then an error occurs only if $n_c > E$. *

Let binary 0 correspond to $s_1(t)$. Then, the hard-decision output is 0 if $n_c < E$ (no error) and 1 if $n_c > E$ (error). The decoder is expected to invert all 1s and produce an all-zero output. Hence, the number of decoding errors is the number of 1s, at the decoder output. Furthermore, it is assumed that $E = 0.5$. Then, $\sigma^2 = \tilde{n}/4$, so n_c is a zero-mean Gaussian random variable with standard deviation $\sigma = \sqrt{\tilde{n}/2}$. A sample from such a variable is returned by calling the appropriate random-number generator [$n_c = \text{G05DDF}(0, \sigma)$ - a NAG Library routine] (see Fig. A8.3.1, p. 535). **

The decoder is implemented using the majority-logic circuit of Fig. 5.1, as a model. Its operation requires the use of the following arrays: a) IRA ($k \times k$), for the last constraint-length of received message-bits; b) JRA ($1 \times J$), for the current block of received parity-bits; c) ISR ($k \times J$), for the last constraint-length of syndrome-bits; d) JAR ($J \times k$), for the encoding array & e) KAR ($J \times k$), for the syndrome array. The last two arrays are returned by a call to subroutine *CODAR2* (see Fig. A8.1.8, p. 518). After the initializations the decoder shifts IRA & ISR, to make space for the current blocks, stores the data, counts the channel errors so far, calculates the syndromes, collects the syndromes checking on each error bit, estimates the error bits, counts the decoding errors so far and resets the syndrome register, as appropriate. The above is repeated for the specified number of blocks to be decoded and subsequently for each of the channel error-rates to be considered. **

The above-mentioned straightforward approach to decoder implementation is inefficient, because it uses one computer-word to store one bit. As a result, attempts to simulate long codes ($k \geq 160$) hit the computer-memory limit. An alternative implementation uses bit-manipulation commands

* The error probability remains as above.

** See Appendix 8.3 (p. 533), for a discussion.

to store b bits of data in one word ($b=60$, for the CDC-7600 mainframe, used). The above-mentioned approach requires a very sophisticated programme because the decoder, for the general (k,J) type-C5 code, has to be assembled using a minimum number of $1 \times b$ subarrays. Complications arise because of word-boundary conditions, during the shift (vertical or horizontal) of the arrays, or the writing-in, or reading-from, selected bit-positions. The resulting programmes* allowed the simulation of longer codes (by a factor of \sqrt{b}) and improved the processing time by a factor of 2-3. Examples A8.3.1 & A8.3.2 (pp. 536-41) illustrate the above technique, as well as the support required, for such a decoder to operate efficiently. § A8.4.2. (p. 543), lists the corresponding (complete) FORTRAN programme.

The above-mentioned programmes return the net coding-gain, the probability of decoding error, etc. Variations of this programme were 'enriched' with a choice between feedback & definite decoding, as well as transmitter feedback (or 'genie' decoding). In addition, the autocorrelation function of the decoder output error-sequence, as well as the error performance of each coset, were also available.

The main programme makes use of a number of subroutines for the processing & presentation of results. This includes, the calculation of the net coding-gain, the inversion of eqn $P_e = \frac{1}{2}\text{erfc}(\sqrt{\Gamma})$, the calculation of the BSC capacity, the calculation of the theoretical probability of decoding error, the ordering of data in descending order, the determination of the $C(N,t)$ combinations of N things taken t at a time, etc. Their flow-charts are explained in Appendix 8.5.

In order to facilitate the choice of suitable type-C5 codes, three subroutines were designed that select a code that best matches given code parameters (like k , J & rate). For example, given positive integers c & k , subroutine IORD4 (see § A8.7.2, p. 564) returns the type-C5 code with rate $c/(c+1)$ and information-block length as close to k as possible. Appendix 8.7 presents & discusses these routines.

* The new decoder implementations were tested against the older ones, for correctness.

8.3 PERFORMANCE DATA

The test to which a given code is subjected is the delivery at the decoder I/P of blocks of corrupted data. The decoder is expected to correct them, but it does not always do so. The three fundamental parameters obtained are, the total number, N_b , of bits considered (*sample-size*), the total number of channel errors, N_{ce} , and the total number of decoder errors, N_{de} . For a given N_b , the decoder is subjected to N_{ce} errors and its performance is assessed by counting the number of failures, N_{de} . As is the case, though, with all performance-data graphs & tables, a degree of normalization is required. The usual way, for error-correcting codes, is a graph of the probability of decoding error (in log-scale), P_d , versus $10\log\Gamma$, where Γ is the energy per information-bit over \tilde{n}^* . The simulation curves are compared with the corresponding theoretical curve, for uncoded transmission.

It has been mentioned already (see Section 1.4), that the *net coding-gain*, G , is the most 'fair' measure of the effectiveness of a code. G may be obtained from the above-mentioned graph by measuring the dB difference between the coded & uncoded cases, for a given error rate. This error rate is the P_d of the coded system and the probability of channel error, P_e , of the uncoded system. Furthermore, P_e corresponds to a specific Γ . Hence, an alternative representation is G vs Γ , or G vs P_e .

8.3.1. Estimation of G , P_e & P_d

What one gets from a simulation programme, such as the one described above, are some data from a sample (of size N_b), the sample taken from a *hypothetical population* (i.e. one which can be conceived, without being able to attain in practice). Such a population is taken to have infinite size.

The 'population' mentioned, is the set of all bits at the decoder input. Due to the adoption of an all-zero transmission, the population average is also the probability of channel error, P_e . A second population is the set of all bits at the decoder O/P^{**}. Similarly, its average is P_d .

* This is the one-sided noise power spectral density.

** Strictly speaking, the 2nd population is obtained from the 1st one & the decoder.

To estimate the population statistics, the sample ones will be used. The sample mean is known to be the best estimate of the population mean. Hence:

$$\tilde{P}_e = N_{ce}/N_b \quad (8.1a)$$

$$\tilde{P}_d = N_{de}/N_b \quad (8.1b)$$

The corresponding best estimate of G is now required. From eqns (1.19) & (1.18), $G = 10\log(\Gamma'/\Gamma)$, where Γ' is the ' Γ quantity' for uncoded transmission with error rate P_d , i.e. $\Gamma' \triangleq [\text{erfc}^{-1}(2P_d)]^2$. Since the best estimate of P_d is \tilde{P}_d , then (intuitively) the best estimate of Γ' is:

$$\tilde{\Gamma}' = [\text{erfc}^{-1}(2\tilde{P}_d)]^2 \quad (8.2)$$

Γ is the energy per information-bit over \tilde{n} , for the coded case. From eqn (1.15):

$$\tilde{\Gamma} = [\text{erfc}^{-1}(2\tilde{P}_e)]^2/R \quad (8.3)$$

8.3.2. Confidence Intervals

Since it is attempted to estimate the population mean from the sample mean, one has to declare one's confidence on the accuracy of the experiment. The sample mean is a random variable. It can be proved (see Erricker [49], p. 196) that if the sample size is sufficiently large (≥ 30) then the sample means are normally distributed random variables. Then, it may be proved* that if \tilde{P}_x is the sample mean, the probability that P_x lies in the interval below, is 99%.

$$\tilde{P}_x - 2.58\sqrt{[\tilde{P}_x(1-\tilde{P}_x)/N_b]} < P_x < \tilde{P}_x + 2.58\sqrt{[\tilde{P}_x(1-\tilde{P}_x)/N_b]}^{**} \quad (8.4)$$

8.3.3. Presentation of Results

From the preceding discussion, it is obvious that confidence intervals are introduced not only for P_d , but also for P_e . The random number generator may be assumed to be the hypothetical population generator. A sample (of size N_b) is taken from its O/P. In response to this, a decoder O/P bit stream is generated, from which P_d is estimated. The decoder responds in a deterministic way, given the sample of the input bit stream. Hence, the only uncertainty lies with the

* See Appendix 8.9 (p. 569).

** 2.58 is replaced by 1.96 for 95% confidence, by 1.645 for 90% and by 0.6745 for 50%.

relation between the sample and the I/P bit stream, i.e. between \tilde{P}_e and P_e . This though generates an uncertainty between \tilde{P}_d & P_d and an uncertainty between \tilde{G} & G , as well as $\tilde{\Gamma}$ & Γ . Hence, the confidence intervals should be two-dimensional. One may test the code against the given random-number generator settings, in order to avoid the two-dimensional confidence intervals. But, as will be seen later on, the uncertainty on the channel actual error-rate is insignificant.

From Fig. A8.3.1, $n_c = G05DDF(0, \sigma)$, hence, the channel is chosen via σ . The uncertainty between σ & $\tilde{\sigma}$ generates the uncertainty between P_e and \tilde{P}_e . Since $\sigma^2 = E\tilde{n}/2$ [see equation (A1.2.11)] and since $E = 0.5$ (see Section 8.2), then $\sigma^2 = \tilde{n}/4$. Since $E = \text{energy/bit} \longrightarrow E/R = \text{energy/information bit} \longrightarrow \Gamma = (E/R)/\tilde{n} = (0.5/R)/(4\sigma^2)$:

$$\tilde{\Gamma} = 1/(8\tilde{\sigma}^2 R) \quad (8.5)$$

The confidence intervals on $\tilde{\Gamma}$ are obtained, though, from those on \tilde{P}_e , via eqn (8.3).

Consider now \tilde{G} . From (1.19) and the discussion so far:

$$\tilde{G} = 10 \log(\tilde{\Gamma}'/\Gamma) \quad (8.6)$$

$$\text{where} \quad \tilde{\Gamma}' = [\text{erfc}^{-1}(2\tilde{P}_d)]^2 \quad (8.2)$$

$$\text{and} \quad \tilde{\Gamma} = [\text{erfc}^{-1}(2\tilde{P}_e)]^2/R \quad (8.3)$$

The major uncertainty in (8.6) is about P_d . To obtain therefore the confidence limits on \tilde{G} , one has to substitute those for P_d in (8.2) and then in (8.6). From (8.4),

$$\tilde{P}_d \pm 2.58\sqrt{[\tilde{P}_d(1-\tilde{P}_d)/Nb]} \quad (8.7)$$

are the confidence limits on P_d . Hence, \tilde{P}_d is obtained from Nde/Nb , substituted in (8.7) and these three values are used in (8.2) to obtain three values for $\tilde{\Gamma}'$, and finally in (8.6) to obtain three values of \tilde{G} (average & 99% confidence limits).

8.4 CODING GAIN AND OTHER PERFORMANCE CRITERIA UNDER FD

A number of type-C5 codes were simulated over the binary symmetric channel (BSC) using feedback decoding. The simulation programme produced figures for the net coding-gain, G , based on the obtained probability of decoding error, P_d . G is plotted against Γ , the signal-to-noise ratio per information-bit. The minimum Γ used was such that the BSC capacity was not exceeded.*

Consider now the reliability of the experimental data. From relation (8.7), the 99% confidence interval on the estimated P is $P\{1 \pm 2.58\sqrt{[P(1-P)/N]/P}\}$. If d is the number of errors and N is the sample-size, then $P = d/N$ and the relative width of the interval is $2.58\sqrt{[P(1-P)/(NP^2)]} = 2.58\sqrt{[(1-d/N)/d]} = 2.58\sqrt{(1/d - 1/N)}$. If this is to be small, say less than 0.1, then $2.58\sqrt{(1/d - 1/N)} < 0.1$, or $1/d - 1/N < 1.5 \times 10^{-3}$, or $d > (1.5 \times 10^{-3} + 1/N)^{-1}$. If at least 10,000 blocks are considered and the smallest information-block length tested is $k=12$, then $1/N$ is at most $1/120,000 = 8.3 \times 10^{-6}$. Then, $d > (1.5 \times 10^{-3})^{-1} \approx 660$. So, N must be such that at least 660 decoding errors occur (if the relative width of the confidence interval is to be less than 0.1). On the other hand, the computer processing-time limitations were such that N could not exceed 10^7 . For example, to obtain one point of the P_d versus Γ graph for the (1320,6) type-C5 code, 740 secs of processing time were required** for a 10,000-block simulation (13,260,000 error bits were injected). With such a limit on N , the minimum decoding-error probability which could, confidently, be estimated was of the order of $660/10^7 \approx 7 \times 10^{-5}$, i.e. not small enough. For the code performance to be evaluated at useful channel error-rates, small values of d had to be accepted. From above, the relative width of the 99% confidence interval is $\approx 2.58/\sqrt{d}$. This is 1, for $d = 2.58^2 \approx 7$, which means that the corresponding 99% confidence interval ranges from 0 to $2\tilde{P}_d$.

\tilde{P}_d is used to estimate G , the net coding-gain. For high values of Γ (i.e. for small \tilde{P}_d), d may be small enough so that the confidence interval of \tilde{P}_d to start from $P_d = 0$, to which the asymptotic coding-gain, G_a , corresponds (see Sec-

* See Appendix 6.2 (p. 418) & relation (8.3).

** On a CDC-7600 mainframe.

tion 1.4). Once though d reaches 20, the relative confidence interval of \tilde{P}_d is about 1 ± 0.5 . The corresponding range for G is not wide any more because of the non-critical dependence of G on P_d , for high Γ s, since G tends asymptotically to G_a , as $\Gamma \rightarrow +\infty$.

The uncertainty on P_e is smaller than that on P_d , because $P_d < P_e$ for at least those values of P_e which 'lack' confidence. The lowest channel error rate used, must be high enough to produce at least one decoding error. Since the code performance is very good at very low values of P_e , the number of channel errors injected must be at least two orders of magnitude greater than the number of decoding errors. Furthermore, care was taken to increase the number of blocks transmitted, if the expected number of channel errors was less than, say, 350. In the extreme case where this was true, the relative width of the confidence interval on \tilde{P}_e is $2.58/\sqrt{350} = 0.14$. Therefore, the uncertainty on P_e , and hence on Γ , may be ignored exactly because the experiments were designed so that this uncertainty is insignificant.

G vs Γ graphs of type-C5 codes of rates between $1/2$ & $39/40$ are presented below and in appendices. The range of Γ is between 3 & 8 dB, while the net coding-gains obtained were between -2 & 3 dB. Figs 8.3-5 (see also Figs A8.10.3-5, in Appendix 8.10, pp. 572-4) contain $G=f(\Gamma)$ graphs of codes of the same rate.

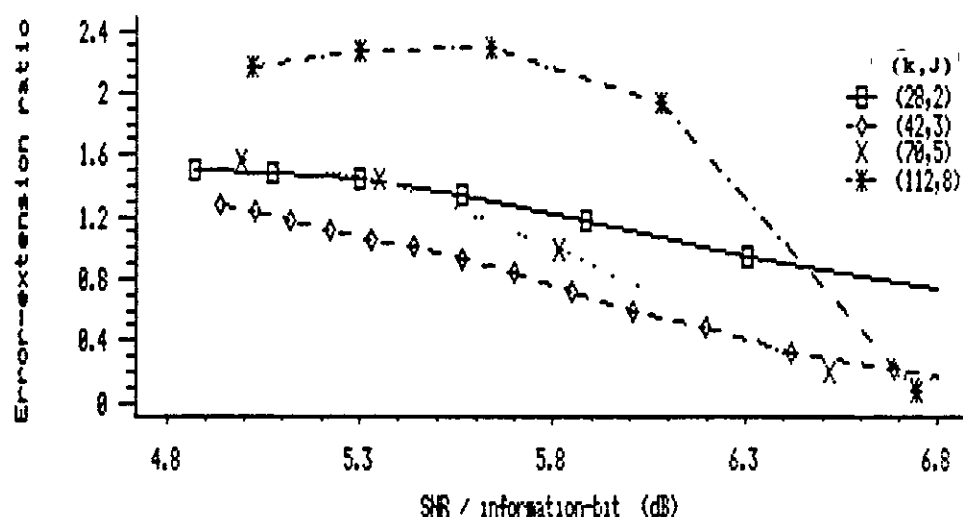


Figure 8.1: EER vs Γ for codes of rate 14/15. *

* EER = error extension-ratio.

Γ = signal-to-noise power-ratio per information-bit.

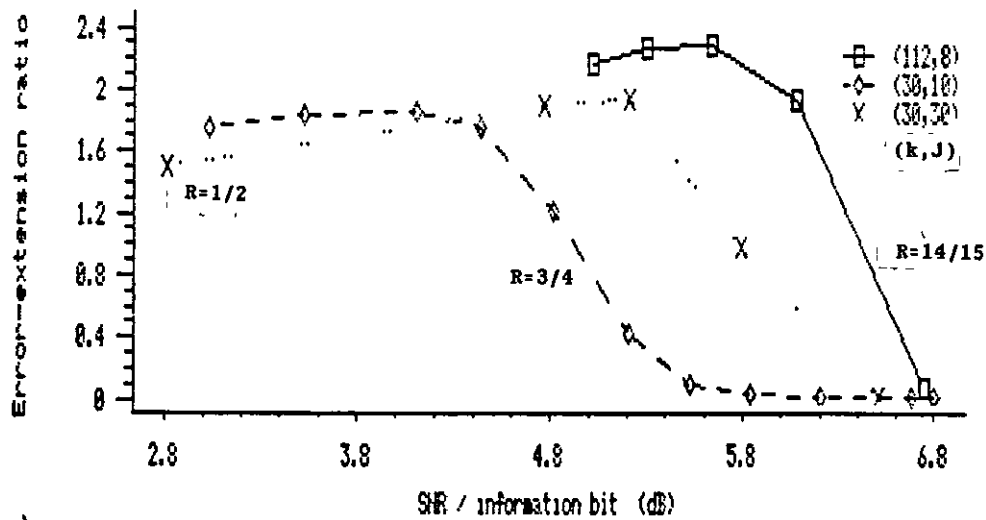


Figure 8.2: EER vs Γ for codes with large J. *

As a first conclusion, as Γ decreases towards its value corresponding to the channel capacity (3-4 dB), G tends asymptotically to a negative value of a few dBs. It is anticipated that the syndrome-feedback mechanism prevents the further deterioration of G. This is expected to be so because, at low Γ s too many channel errors cause even more decoding errors which are then fed back to the syndrome register and cancel, by accident, some of the incoming channel errors. This will be verified at a later section, when the error-propagation effect is studied. Figs 8.1 & 8.2 (and A8.10.1-2, p. 571), show that the error-extension ratio, EER ($\hat{=} P_d/P_e$), stabilizes around 1.5-2.5 and even decreases for

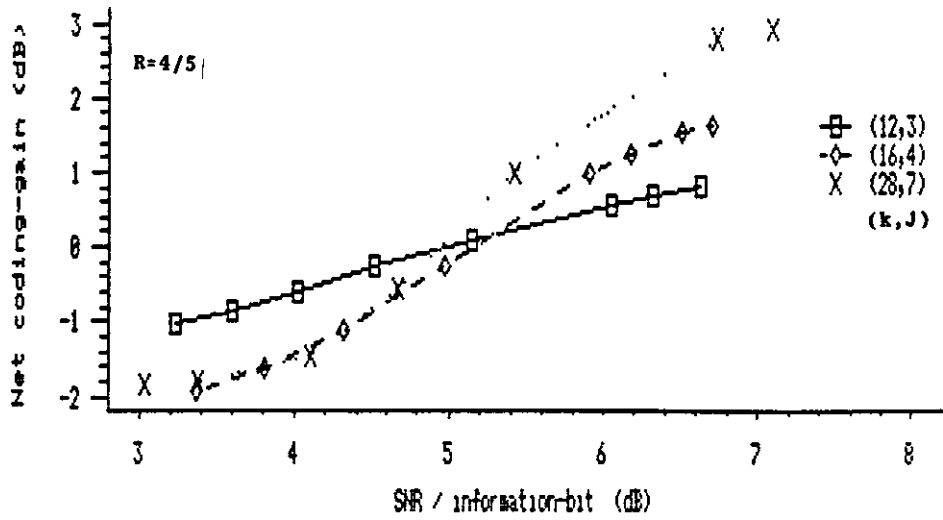


Figure 8.3: Net coding-gain vs Γ for codes of rate 4/5. *

* EER = error extension-ratio.
 Γ = signal-to-noise power-ratio per information-bit.

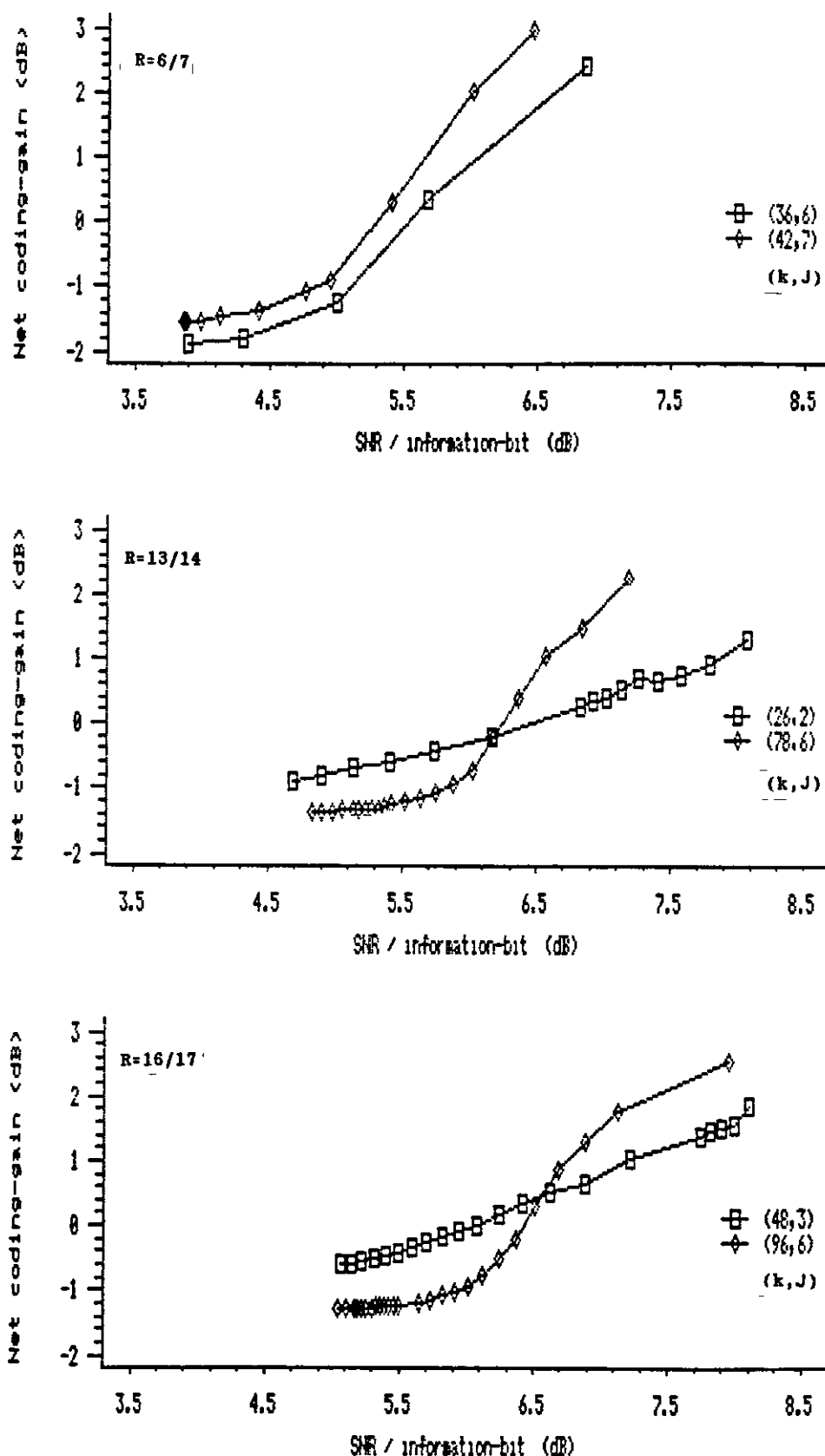


Figure 8.4: Net coding-gain vs Γ , for codes of rate 6/7 (top), $R=13/14$ (middle) and $R=16/17$ (bottom). *

* Γ = signal-to-noise power-ratio per information-bit.

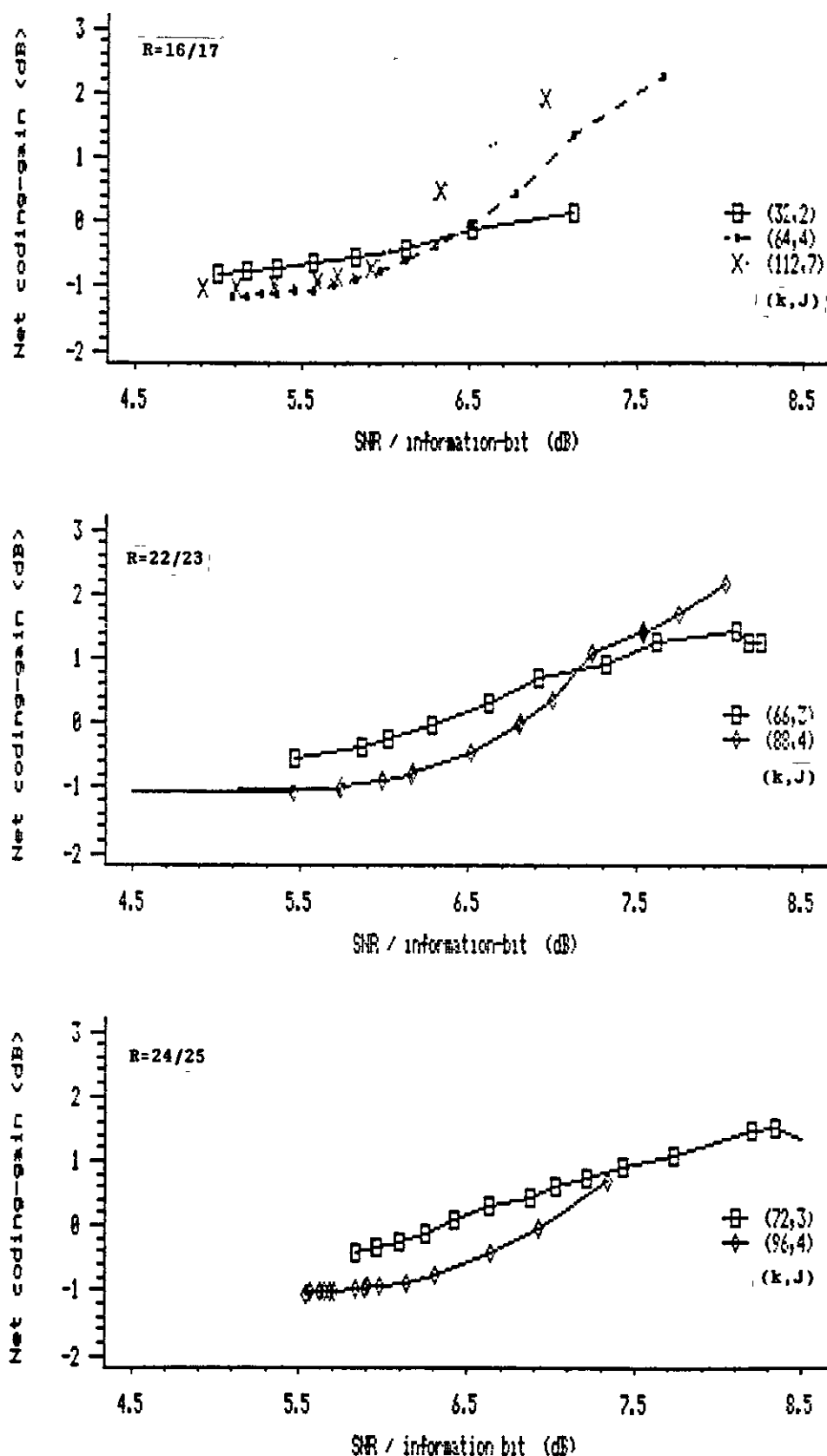


Figure 8.5: Net coding-gain vs Γ , for codes of rate 16/17 (top), $R=22/23$ (middle) and $R=24/25$ (bottom). *

* Γ = signal-to-noise power-ratio per information-bit.

codes with high J_s (≥ 8) (with every decoding error, J errors are injected into the syndrome register).

A second conclusion, that can easily be drawn, is that for codes of the same rate the $G=f(\Gamma)$ characteristic has a higher slope for 'longer' codes. This reveals that these codes achieve a higher net coding-gain at high Γ_s but perform worse for low Γ_s . The slope-difference increases with the 'length'-difference. What is not known, at this stage, is which of J or k is responsible for the higher slope of the 'longer' codes? This can be deduced, though, from constant- J and/or constant- k graphs.

The asymptotic coding gain G_a , for codes with rate R and J orthogonal check-sums is $R(\lfloor J/2 \rfloor + 1)$. Hence, for two codes of the same rate, with J & $J+8$ orthogonal check-sums, respectively, the $G=f(\Gamma)$ characteristic is expected to be $10\log[(\lfloor (J+8)/2 \rfloor + 1)/(\lfloor J/2 \rfloor + 1)]$ dB higher, at high Γ_s , for the 'longer' code. For codes with the same error-correcting capability $\lfloor J/2 \rfloor$, ($J=\text{even}$ & $\delta=1$), the two characteristics are expected to start from the same point (at high Γ_s), but the $J+1$ code is expected to do better for moderate values of Γ , because of the extra check-sum. For low Γ_s the difference is expected to disappear, or at least be reduced, because the $J+1$ code corrupts its syndrome register with one extra error, with every erroneous decoding. From Fig. 8.4 (top) it may be seen that the $J=7$ code has an advantage of about 1 dB at moderate values of Γ ; this advantage is reduced to about 0.5 dB at low Γ_s (the two codes have rate $6/7$, hence they are expected to have $G_a = 5.4$ dB).

On the other hand, for $J=\text{odd}$, two codes with the same rate but with J & $J+1$ orthogonal check-sums have different error-correcting capabilities $[(J-1)/2$ & $(J+1)/2$, respectively], hence they are expected to start from different points, at high Γ_s . For $J=3$, the codes have G_a 's that differ by 1.8 dB. From Fig 8.5, the (88,4) code has an advantage of about 0.8 dB, over the (66,3) code, for $\Gamma = 8$ dB.

In order to explain some of the observations, it is necessary to compare codes with the same J (see Figs 8.6 & 8.7, as well as Figs A8.10.6-8, pp. 574-6).

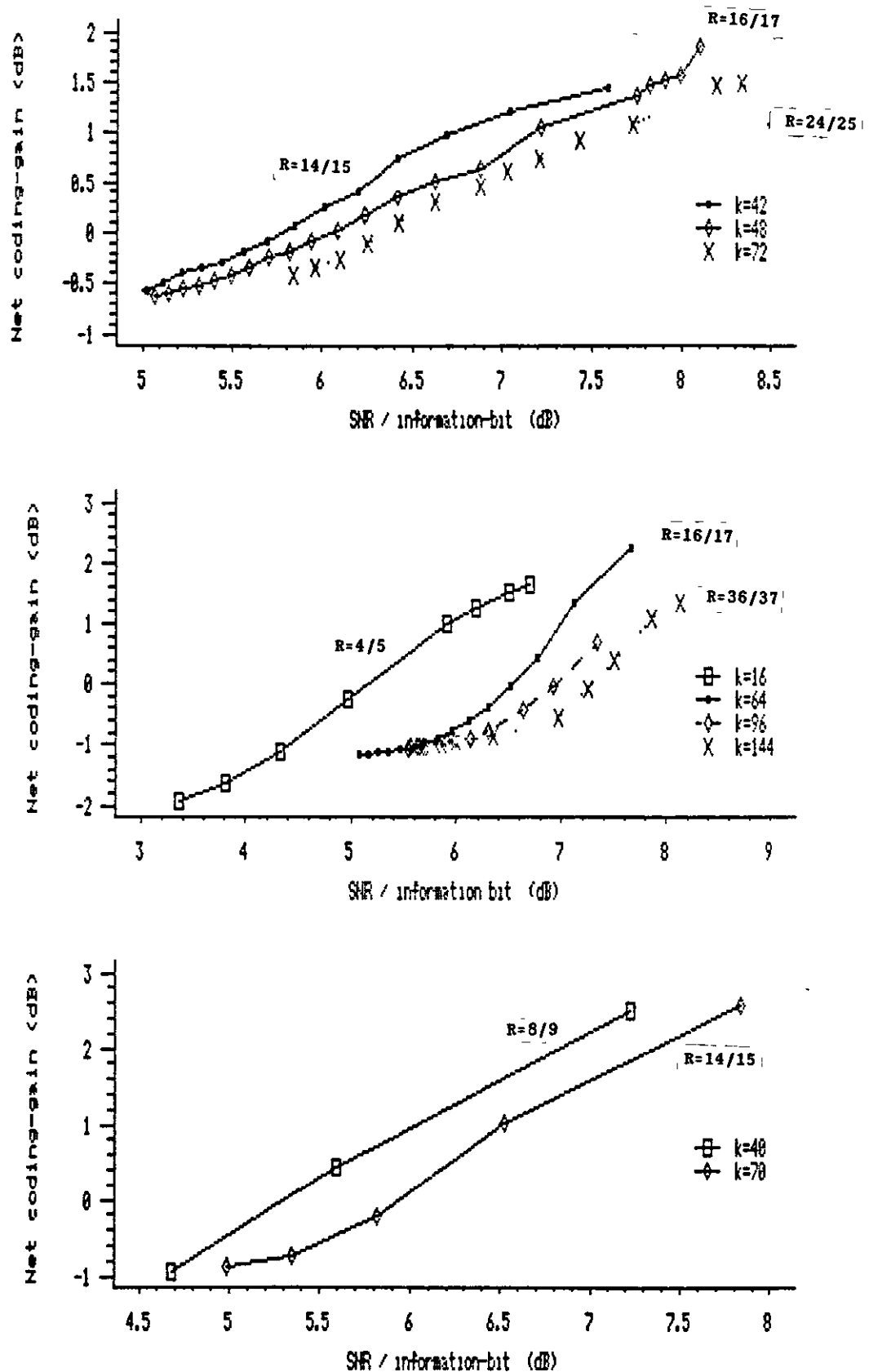


Figure 8.6: Net coding-gain vs Γ , for codes with $J=3$ (top), $J=4$ (middle) and $J=5$ (bottom). *

* Γ = signal-to-noise power-ratio per information-bit.

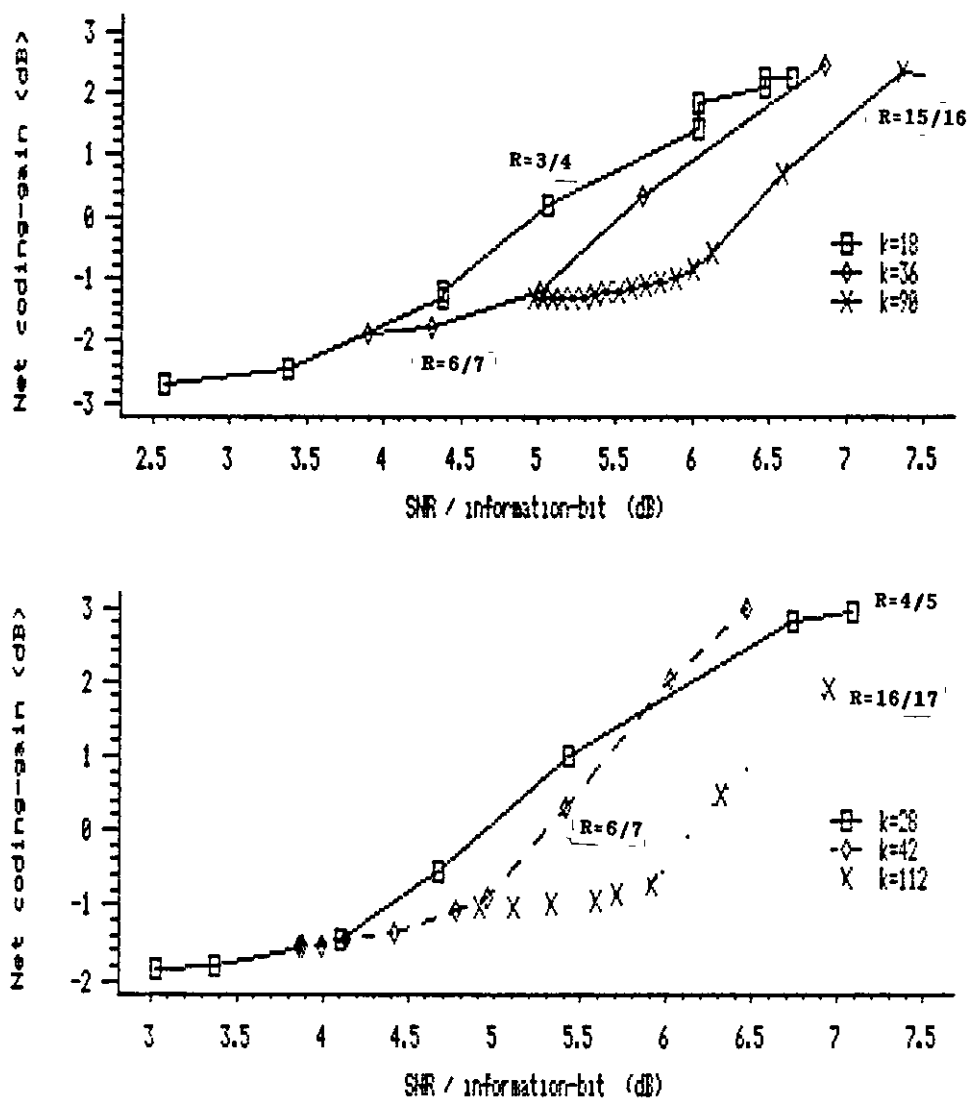


Figure 8.7: Net coding-gain vs Γ , for codes with $J=6$ (top), and $J=7$ (middle). *

A third conclusion is that the value of J seems to be responsible for the slope of the graph. Codes with the same J have the same slope, while the characteristic of longer codes is displaced towards higher values of Γ . To state it differently, a longer code achieves the same G for higher values of Γ . For example, for the (40,5) & (70,5) codes, the latter requires an extra 0.5 dB in Γ to achieve the same G of 2 dB. The (16,4), (64,4) & (144,4) codes, achieve a net-coding gain of 1 dB at $\Gamma = 6, 7$ & 8 dB, respectively.

Figs 8.8 & 8.9 contain graphs of codes with the same k (see also Figs A8.10.9 & A8.10.10). It is obvious, from these graphs, that codes with larger J have a steeper slope

* Γ = signal-to-noise power-ratio per information-bit.

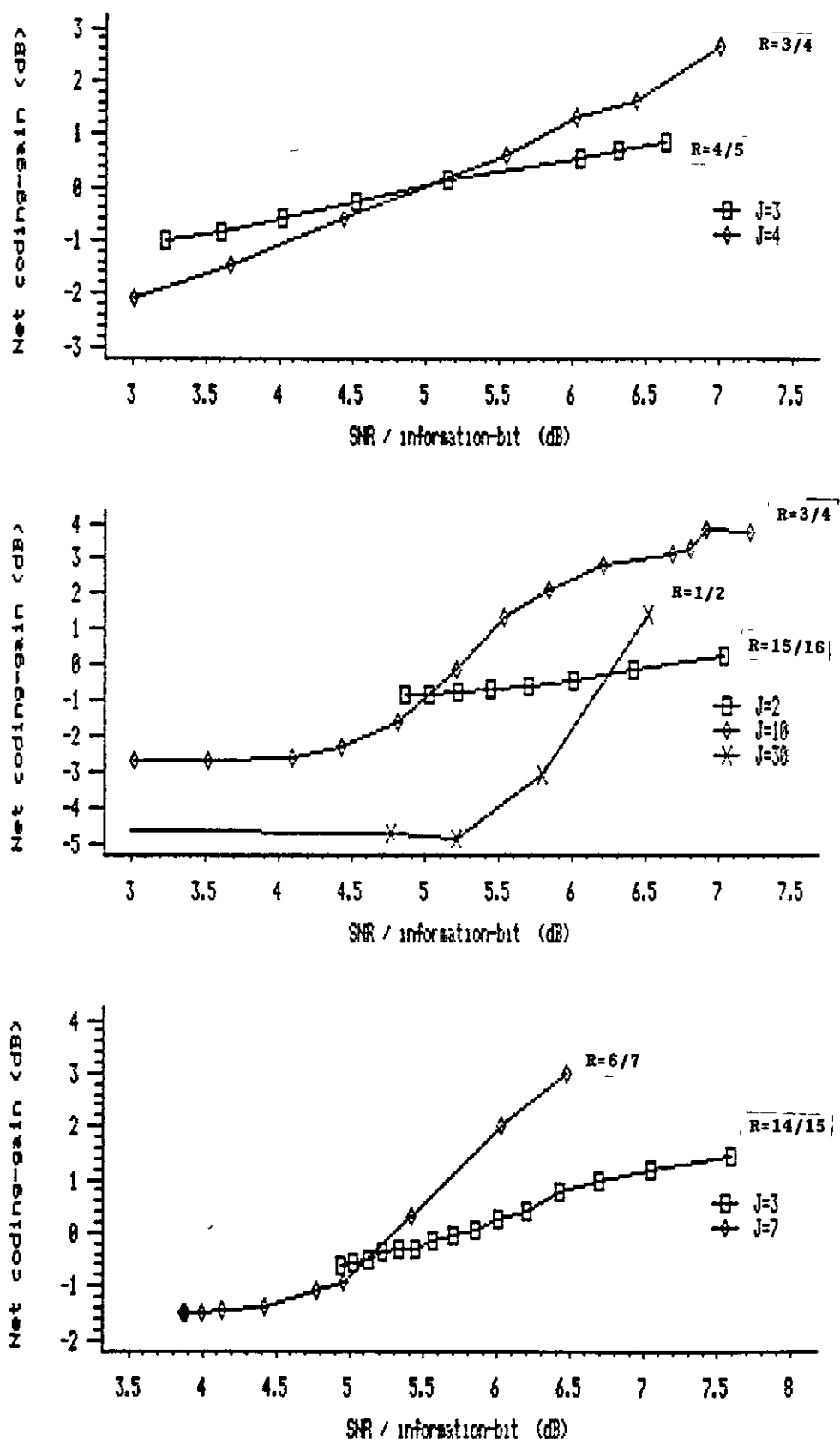


Figure 8.8: Net coding-gain vs Γ , for codes with $k=12$ (top), $k=30$ (middle) and $k=42$ (bottom). *

* Γ = signal-to-noise power-ratio per information-bit.

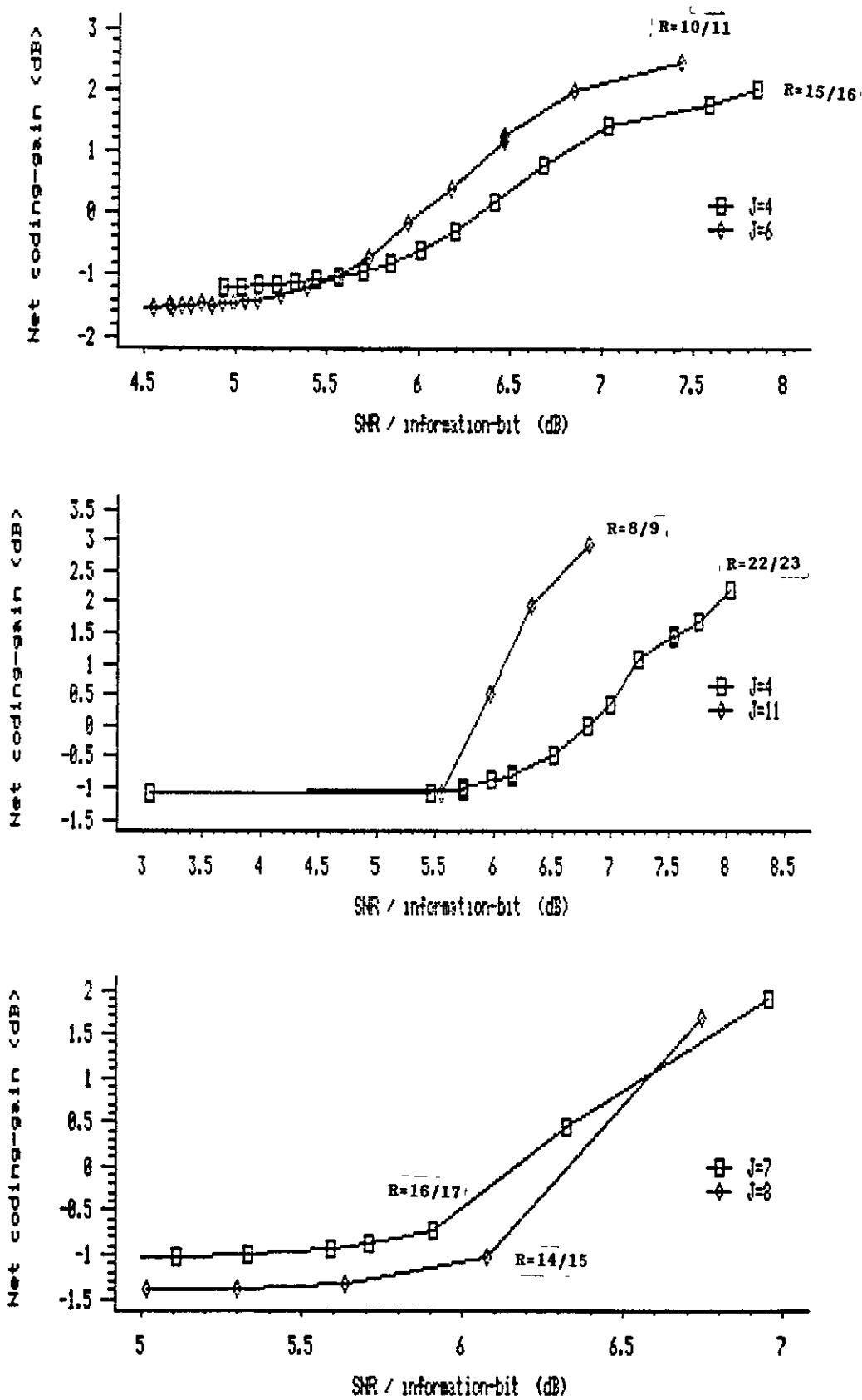


Figure 8.9: Net coding-gain vs Γ , for codes with $k=60$ (top), $k=88$ (middle) and $k=112$ (bottom). *

* Γ = signal-to-noise power-ratio per information-bit.

for moderate to high values of Γ , but end with a worse value of G for low Γ s. For example, the (12,4) code is better than the (12,3) for $\Gamma > 5$ dB. At $\Gamma = 6.5$ dB it offers one extra dB (their G_a is 3.5 & 2.0 dB, respectively). The (42,7) code is better than the (42,3) for $\Gamma > 5$ dB. At $\Gamma = 6.5$ dB it offers 2.5 extra dBs (their G_a is 5.4 & 2.7 dB, respectively). The (60,6) code is better than the (60,4) code for $\Gamma > 5.5$ dB, offering 0.8 extra dB at $\Gamma = 7$ dB. The (88,11) code is better than the (88,4) code for $\Gamma > 5.5$ dB, offering 2.9 extra dBs at $\Gamma = 7$ dB.

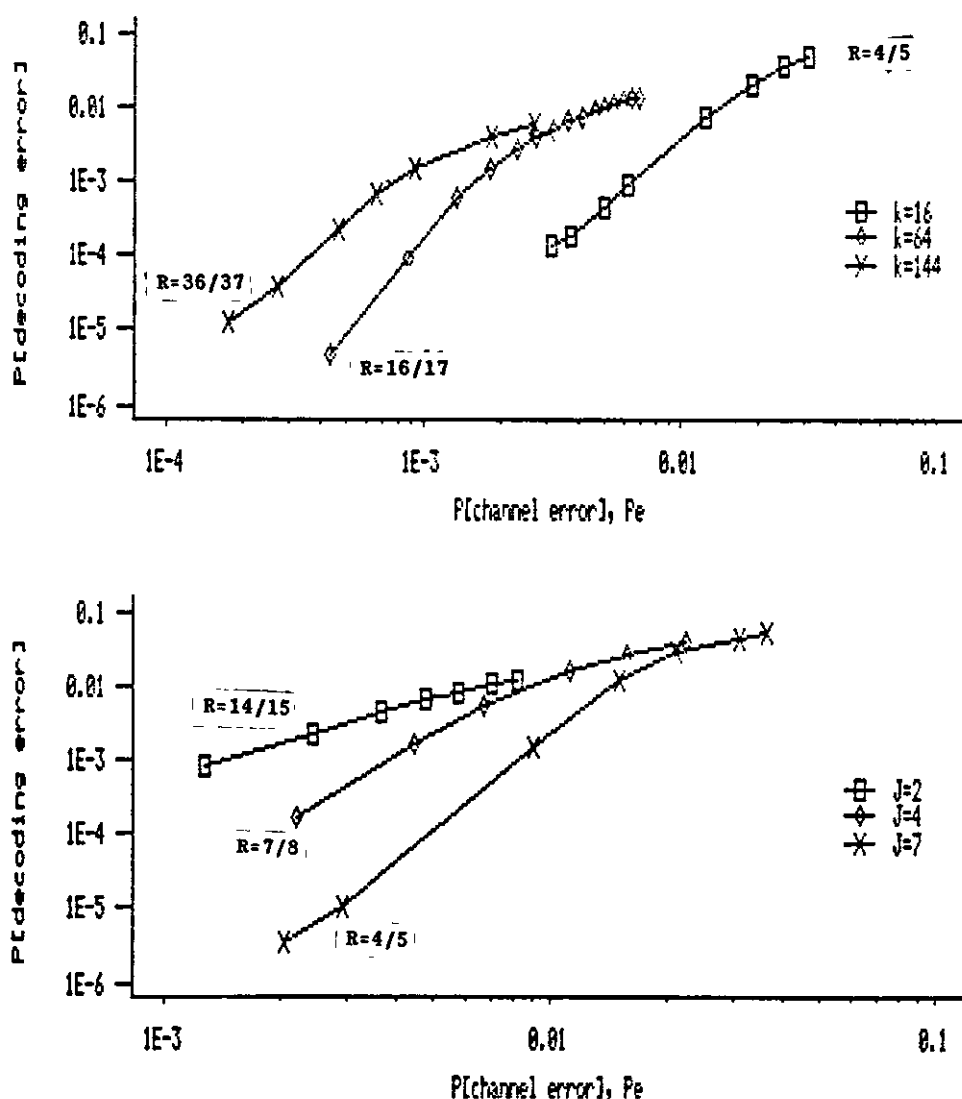


Figure 8.10: Probability of decoding error, P_d , vs probability of channel error, P_e , for $J=4$ codes (top) & $k=28$ codes (bottom).

Figs 8.10 contain P_d vs P_e graphs for $J=4$ codes & $k=28$ codes, respectively. The constant- J graphs have approximately the same slope, and are displaced along the Γ -axis. The constant- k graphs have different slopes. Hence, the earlier findings about the relation between J and the rate at which the code-performance deteriorates, as channel conditions worsen, were not the result of the normalization of signal-to-noise ratio to the code rate, but the working of the syndrome-feedback mechanism.

A (fourth) conclusion can be reached about the displacement of the constant- J graphs towards smaller P_e s (greater Γ s). As k increases, this must be (at least partly) due to the worsening relative error-correcting capability $t_r \hat{=} \lfloor J/2 \rfloor / [(k+J)k]$. For $J=4$, the code guarantees to correct any two errors within one constraint-length, $n_A = k(k+4)$. * For $P_e = p$, the probability of three errors in $N \hat{=} n_A$ bits is $p^3(1-p)^{N-3}C(N,3) \approx (p^3/6)[N!/(N-3)!] \approx (p^3/6)N^3$; hence this probability increases as the cube of the actual constraint-length. Fig. 8.11 contains the graph of P_d vs $QE \hat{=} P_e/t_r$ for $J=4$ codes; the longer codes perform better at the same QE . This reversal of roles is due to the better noise-averaging performance of longer codes (see § 1.2.2., p. 9).

From the groups of graphs, considered so far, one may conclude that as J increases so does the slope of the $G=f(\Gamma)$

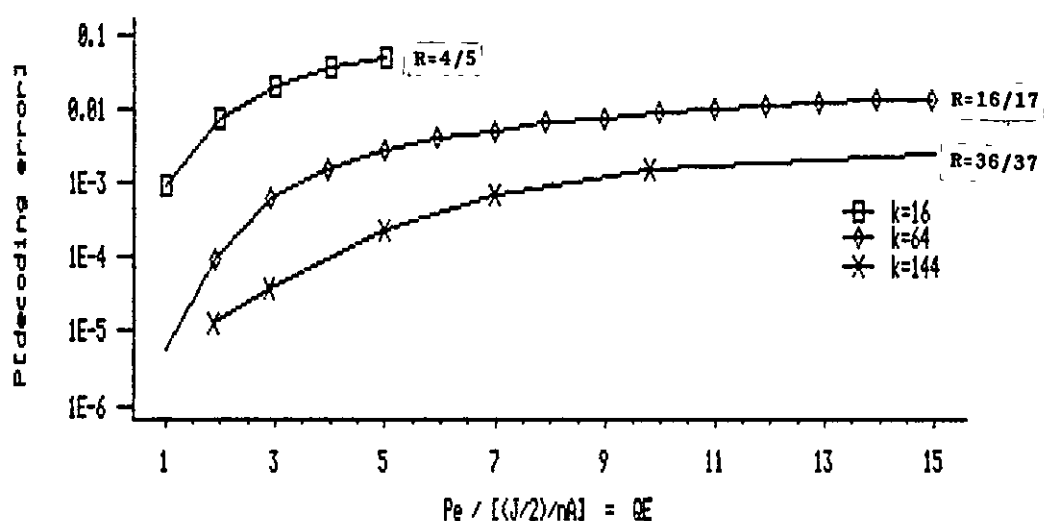


Figure 8.11: P_d vs QE , for $J=4$ codes.

* In fact, within one effective constraint-length, which equals $1+2(k+1)$.

graph. For constant-R codes, $G_a [\approx R(J+2)/2]$ increases with J . For constant-J codes, G_a increases with R (hence with k). For constant-k codes, G_a increases with J . Hence, high-J codes are expected to offer high coding-gains at large values of Γ . As Γ decreases, though, P_d increases and the syndrome-register feedback mechanism starts injecting errors at a rate which is proportional to J . Hence, for a given high value of Γ , the G of high-J codes is greater, but as Γ decreases it deteriorates faster.

A fifth conclusion concerns the range of values of Γ , for which a code is useful, i.e. that value Γ_0 , of Γ , for which $G = 0$. Fig. 8.12 contains graphs of Γ_0 vs k , for $J = 3-7$. For a given J , longer codes are useful at higher values of Γ . For a given k , the higher-J codes are useful over a wider range of Γ s.

Long codes were expensive to simulate. Some results were obtained for eight such codes (see Fig. 8.13). As expected, they are weak even at normally high values of Γ . This is so because their relative error-correcting capability is very low. For the (996,12), $t_r = 6/[996 \times (996+12)] \approx 6 \times 10^{-6}$, i.e. it guarantees to correct the equivalent of one error in 167,328 consecutive bits. In effect, what matters is the effective constraint-length which (for J =even, by Theorem 7.38) is $n_E = 1 + (k+1)J/2 = 5,983$, hence the code guaran-

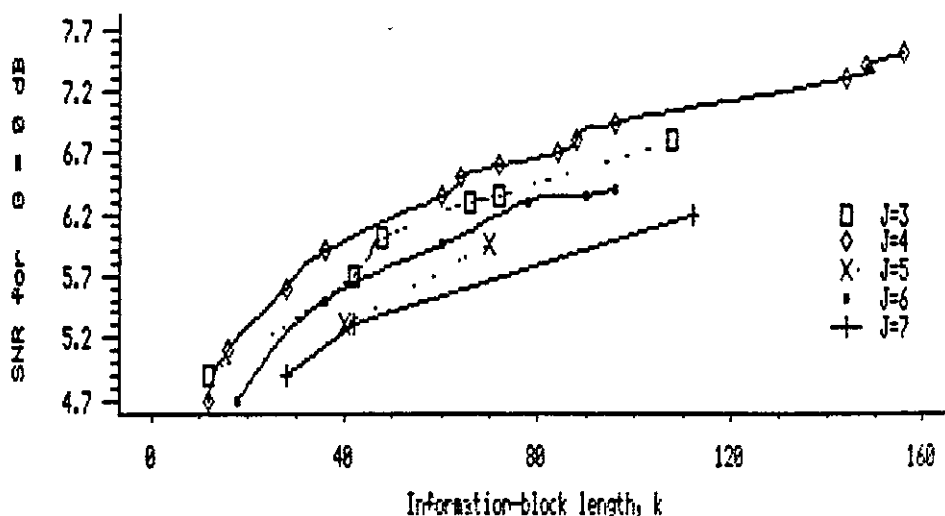


Figure 8.12: Γ_0 vs k , for codes of various J s. *

* Γ_0 = signal-to-noise power-ratio per information-bit, at which net coding-gain = 0 dB.

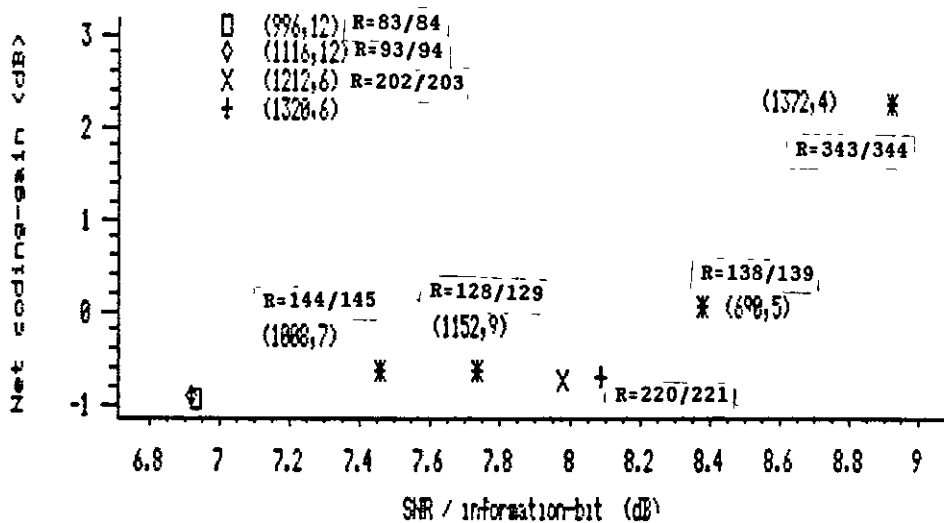


Figure 8.13: Net coding-gain vs Γ , for long codes. *

tees to correct any 6 errors in 5,983 selected bits. For the channel error rate used ($P_e = p = 8.9 \times 10^{-4}$) the probability of 7 errors is $= p^7(1-p)^{5976}C(5983, 7) = p^7(1-p)^{5976} \times 5.43 \times 10^{22} = 0.117$, the probability of 8 errors is $0.117p(5983-7)/(1-p)/8 = 0.078$, etc. Hence, even at high Γ s the guaranteed error-correcting capability of the code is frequently exceeded. On the other hand, the asymptotic coding-gain for these very high-rate codes is virtually $\lfloor J/2 \rfloor + 1$ (because $R \approx 1$), hence they are expected to offer high coding-gains at very high Γ s [$G_a = 8.4$ dB, for the (996, 12) code].

8.5 OPTIMUM THRESHOLD UNDER FEEDBACK DECODING

A number of type-C5 codes were simulated, with a threshold different than the nominal one ($\lfloor J/2 \rfloor$), for various channel error rates (P_e). For this, and other applications, P_e was normalized to the guaranteed error-correcting capability, $t_r \triangleq \lfloor J/2 \rfloor / n_A$: $QE \triangleq P_e / t_r$. The graphs obtained of the net coding-gain, G , vs QE , for various thresholds $T = \lfloor J/2 \rfloor + dT$ are shown in Figs 8.14 & 8.15.

Consider the (36, 4) code [Fig. 8.14 (top)]. Its nominal threshold is 2 and this is expected to be the optimum for $P_e < 1/(C_{Tn})^2$ [see (6.60)]. For this code, $\beta=31$ and the 1st-column elements of the IA are 31, 36, 6 & 1, while the 2nd-

* Γ = signal-to-noise power-ratio per information-bit.

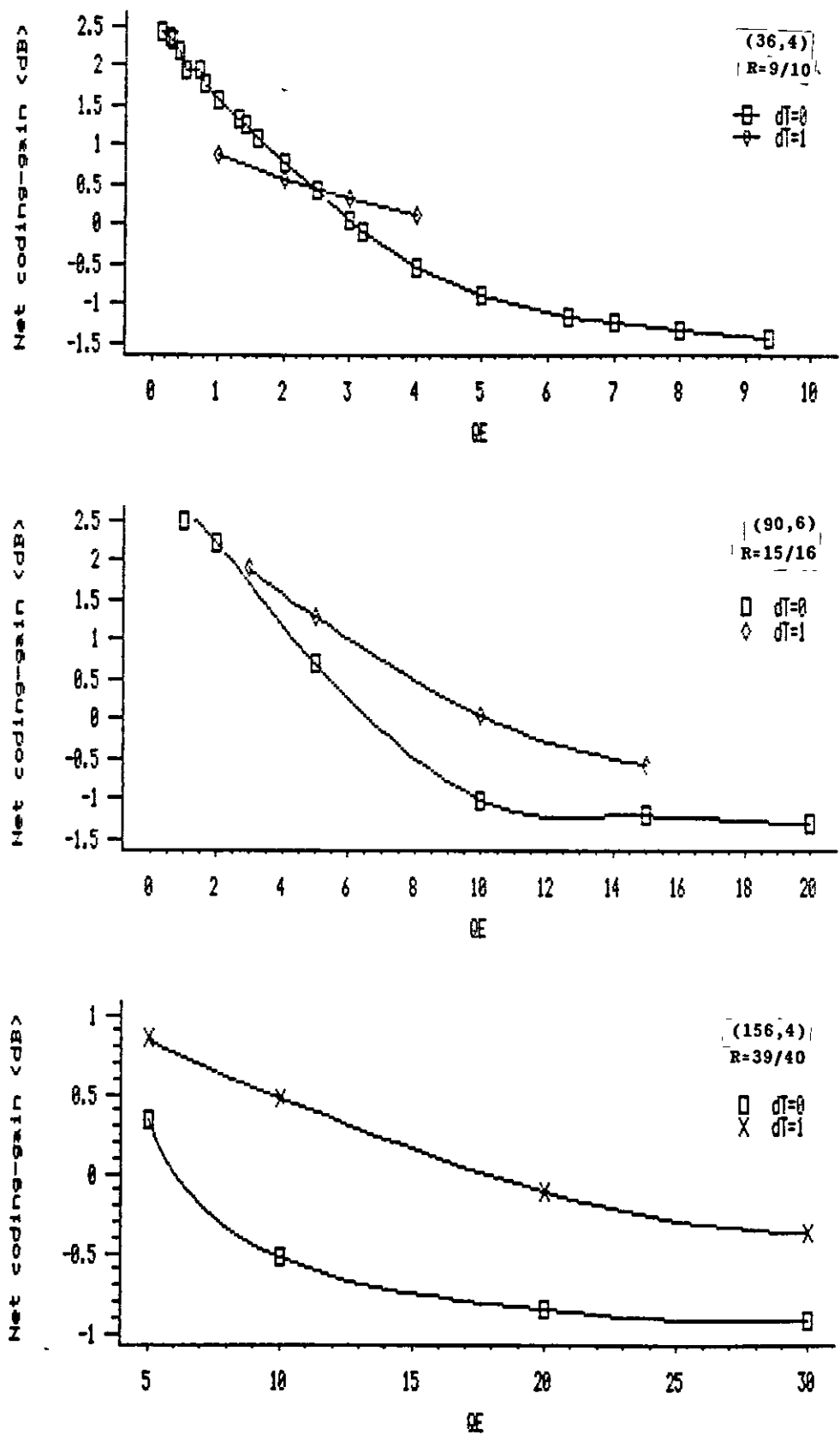


Figure 8.14: Net coding-gain vs QE, under FD & thresholds $T = T_n + dT$ ($dT = 0,1$), for the (36,4) code (top), the (90,6) code (middle) & the (156,4) code (bottom). *

* $QE = (\text{channel error-rate}) / (\text{error-correcting capability/actual constraint-length}).$

column elements are 25,35,12 & 2. Then, for the 1st coset $(C_{Tn})^2 = (31 \times 36 + 31 \times 6 + 31 \times 1 + 36 \times 6 + 36 \times 1 + 6 \times 1) / 6 = 265.2$, while for the 2nd coset $(C_{Tn})^2 = 289.8$. So, for the decoding of the 1st coset $T = 3$, for $P_e > 1/265.2$, or $QE > 2.7$, while for the 2nd coset $T = 3$, for $QE > 2.5$, etc. These predictions are verified from the graph of Fig. 8.14, where $T=3$ becomes better than $T=2$ from about $QE=2.5$. Note also that when the nominal threshold ceases to be useful ($G=0\text{dB}$), the $T=3$ one offers $G = 0.3 \text{ dB}$, while when the latter seizes to be useful, the nominal one has deteriorated to $G = -0.9 \text{ dB}$.

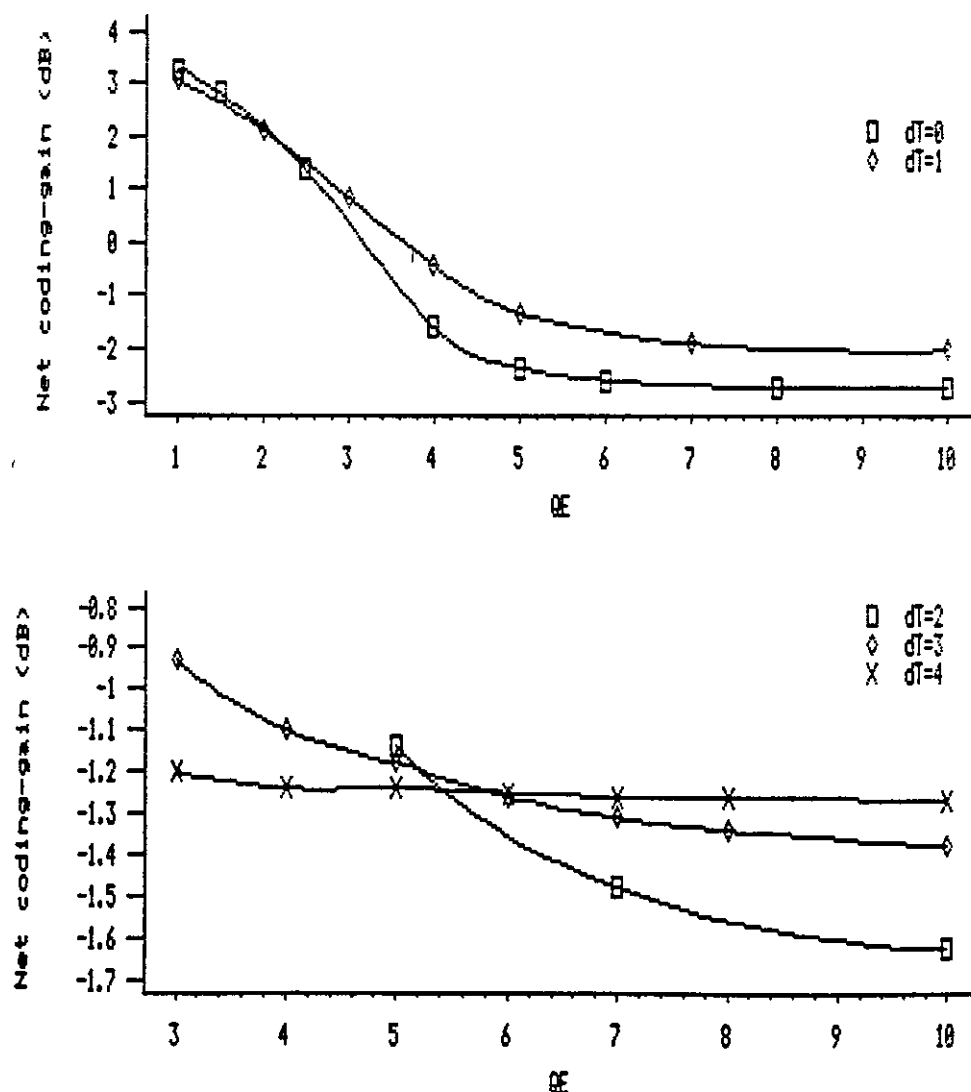


Figure 8.15: Net coding-gain vs QE, for the (30,10) code under FD & thresholds $T = T_n + dT$, where $dT = 0, 1$ (middle), & $dT = 2, 3, 4$ (bottom). *

* $QE = (\text{channel error-rate}) / (\text{error-correcting capability/actual constraint-length})$.

For the (90,6) code, the $T=4$ threshold offers 0.9 dB extra when the $T=3$ threshold results in $G = 0$ dB. When the former returns $G = 0$ dB, the latter has reached $G = -1$ dB.

For the (156,4) code, the $T=3$ threshold offers 0.8 dB extra, when the $T=2$ threshold results in $G = 0$ dB. When the former returns $G = 0$ dB, the latter has reached $G = -0.8$ dB. For this code, $\beta = 129$, and the 1st column of the IA is 129,156,28 & 1, giving $(C_{Tn})^2 = 4736.17$. Hence, the optimum threshold for the 1st coset is 3, for $QE \approx 2.64$.

Fig. 8.15 shows that further increase of the threshold, beyond $\lceil J/2 \rceil + 1$, does improve the code performance but that it is likely that this will happen at channel error-rates at which the code is useless, anyway. Note that the G vs QE graph for the (30,10) code with $T = 9$, is virtually flat and tends to -1.25 dB. This is expected since the probability of decoding error remains $\leq P_e$ with the optimum threshold, hence, as P_e increases, $P_d \rightarrow P_e$ and $G \rightarrow 10\log R^*$. Hence, with the optimum threshold, $G \geq 10\log(3/4) = -1.25$ dB.

8.6 ERROR PROPAGATION

Error propagation is studied by obtaining various results both for 'normal' feedback-decoding (FD or DE f/b) and for

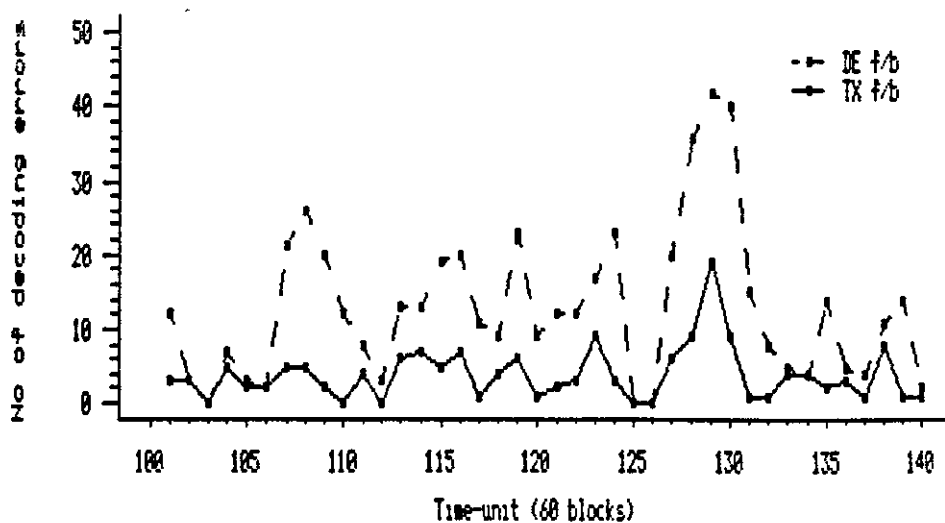


Figure 8.16: Decoder-output error-sequence of the (60,6) code, at $QE = 5$, with $b = 60$.

* See equations (8.2), (8.3) & (8.6)

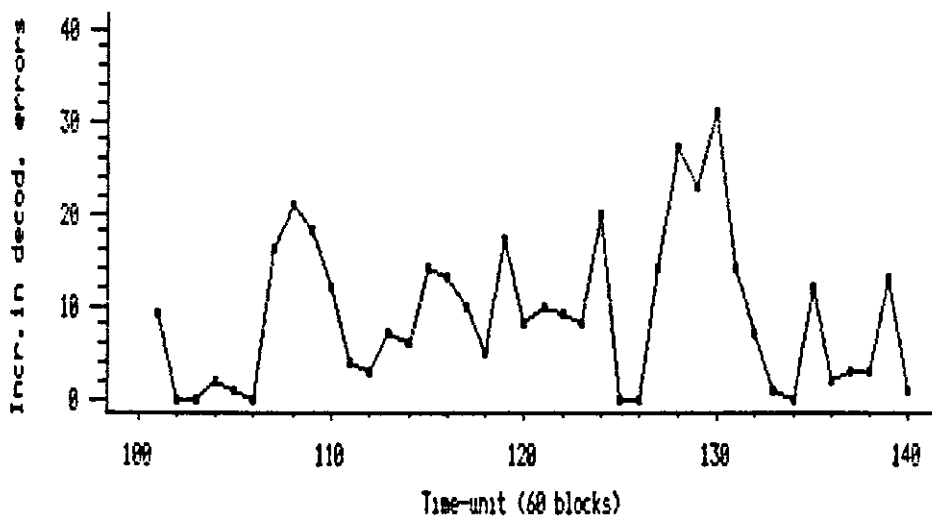


Figure 8.17: Decoding errors, due to incorrect syndrome-resetting, for the (60,6) code, with $QE = 5$ & $b = 60$.

what has been called 'genie decoding'.* Under this mode, which does not exist in the real world, the syndrome register is reset using transmitter feedback ($TX\ f/b$), i.e. the true value of each error bit, instead of the decoder's estimate of it. For some applications, the *decoder output error-sequence* is considered. This is the sequence of numbers representing the number of decoding errors per b (k-bit) blocks ($b=1,2,3,\dots$).

Fig. 8.16 shows the number of decoding errors, per constraint-length, for the (60,6) code both under TX & $DE\ f/b$, while Fig. 8.17 displays the difference between the two time-sequences, i.e. the decoding errors due to error propagation.** The various 'peaks' in the $TX\ f/b$ sequence are due to corresponding peaks in the channel error-sequence. The resulting decoding-errors are fed back in the syndrome register and cause even more decoding errors, when the $DE\ f/b$ mode is employed. Notice, for instance, that at time-unit 107 (107th constraint-length), an undisclosed number of channel errors cause 5 decoding errors, under $TX\ f/b$, but 21 under $DE\ f/b$, although both modes produced 1 error during the previous time-unit. Although, (and obviously due to channel conditions) under $TX\ f/b$, 5 decoding errors occur during the next time-unit, (incorrect) decoder f/b adds another 19 decoding errors. During the next 4 time-units,

* See p. 157.

** See also Fig. A8.11.1 (p. 578).

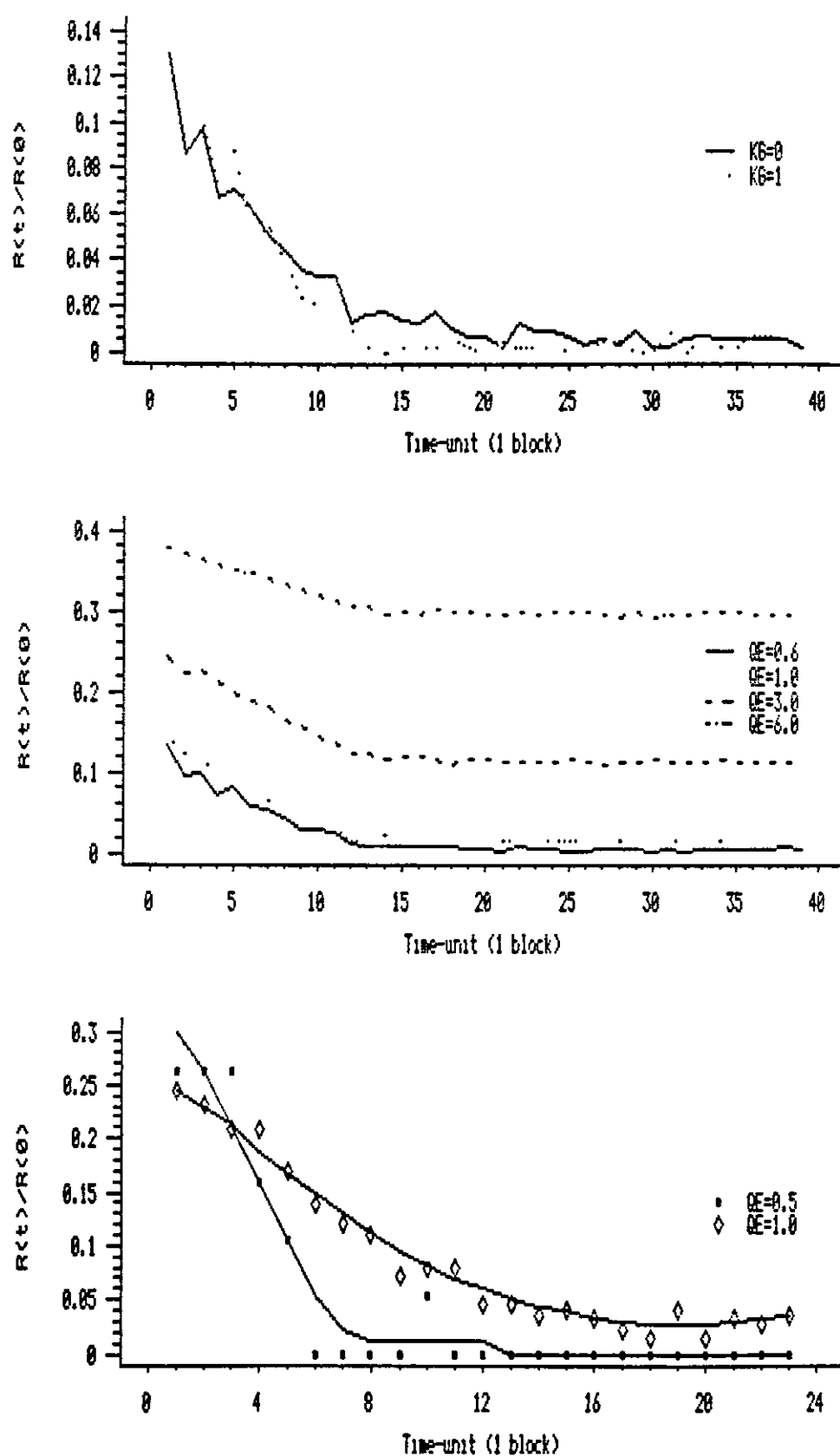


Figure 8.18: Normalized autocorrelation function of the decoder-output error-sequence, with $b = 1$, for the (12,3) (top & middle) and the (12,4) (bottom), code.

the channel 'calms down' and TX f/b produces 1,0,1 & 0 errors, but DE f/b produces 20,12,8 & 3 errors, respectively. At this channel-error rate ($\Gamma = 5.9$ dB), a few decoder failures (1-5 per constraint-length) generate many times more decoding errors, which need a considerable amount of time (3-8 constraint-lengths) to clear-off the decoder. *

Another way to study the error-propagation effect is via the autocorrelation function, $R(\tau)$, of the decoder-output error-sequence. If $D(i)$ denotes the number of decoding errors in the i th b-block part of the sequence, then $R(\tau) \triangleq E[D(i)D(i+\tau)]$. If there is no correlation between $D(i)$ & $D(i+\tau)$, then $R(\tau) = E[D(i)]^2 = E[D]^2$,** where $E[D]$ is the expected number of decoding errors per b blocks. It is expected that $R(\tau) \rightarrow E[D]^2$, as $\tau \rightarrow +\infty$. On the other hand, $R(0) = E[D^2]$. Frequently, it is suitable to study the normalized version of $R(\tau)$, $R_n(\tau) \triangleq R(\tau)/E[D^2]$, and since $R_n(0) = 1$, the $\tau=0$ value is omitted. $R_n(\tau)$ is expected to decrease as τ increases, until it reaches a minimum of $E[D]^2/E[D^2]$, for some value of τ corresponding to, roughly, one constraint-length. $R_n(\tau)$ is expected to be steeper at lower channel error-rates, reaching thus its minimum value faster than at higher channel error-rates.

Fig. 8.18 (top) shows $R_n(\tau)$ / $\tau=1-39$, for the (12,3) code, for two channel error-sequences at $QE=0.6$. Note that both sequences produce a similar 'picture', even though at such a low QE the number of decoding errors is low. As expected, $R_n(\tau)$ decreases towards a minimum (of about 0.005), which is reached for $\tau \approx 13$. There is no correlation among decoding errors that occur more than one constraint-length apart ($b=1$). This is the case at higher QEs, as well [see Fig. 8.18 (middle)]; note that as QE increases, $R_n(\tau)$ stabilizes at higher values, as expected [see Fig. A8.11.2 (p. 579)].

Fig. 8.18 (bottom) shows that the correlation length for the (12,4) code at $QE=0.5$ is ≈ 8 , while at $QE=1.0$ it is ≈ 17 . The two characteristics crossover because they have been normalized to different quantities.

It can be instructive to consider the autocovariance

* Both sequences were obtained using the same channel-noise samples.

** D is assumed to be a stationary random process.

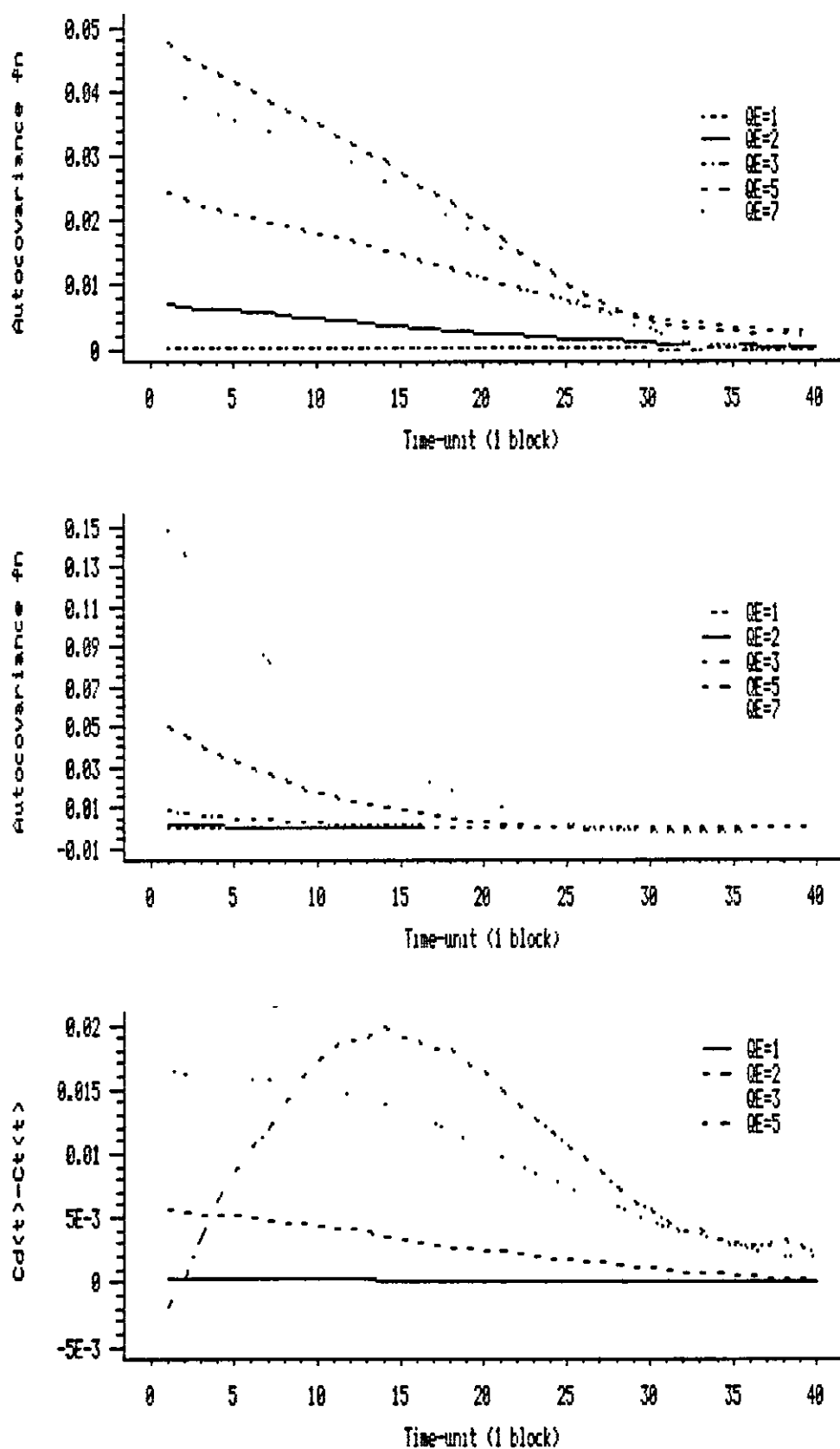


Figure 8.19: For the (28,4) code, with $b=1$, $C_D(\tau)$ vs τ with DE f/b (top), $C_T(\tau)$ vs τ with TX f/b (middle) & $C_D(\tau) - C_T(\tau)$ vs τ (bottom).

function, $C(\tau) \triangleq R(\tau) - E[D]^2$, or $C_n(\tau) \triangleq C(\tau)/C(0)$ [note that $C(0) = R(0) - E[D]^2 = E[D^2] - E[D]^2 = \sigma_D^2$]. This is expected to be flat and low at low QEs, but impulsive at higher QEs.

Consider, at first, the autocovariance function (fn) of the (28,4) code with TX f/b [Fig. 8.19 (middle)]. Since $C(\tau) = E[D(i)D(i+\tau)] - E[D(i)]E[D(i+\tau)] = E[D(i)D(i+\tau)] - E[D]^2$, the more positive the $C(\tau)$ is, the greater the correlation between the decoding errors in blocks i & $i+\tau$. Although there is no feedback of the current decoding-errors to the next ones, the graphs show a correlation which increases with the channel error-rate. This is not a paradox, but the result of the random fluctuations of the rate of channel errors. When the decoder receives many* errors, over a constraint-length, these errors affect the decoding of the current and the subsequent 27 blocks, because they reside in the syndrome register. Of course, with every block decoded, and due to perfect syndrome-resetting (TX f/b), the decoder is gradually unloaded of them. Hence, $C(\tau)$ is expected to decrease 'smoothly', as τ increases, and become ≈ 0 for $\tau \approx 28$. For a given τ , $C(\tau)$ is expected to increase with QE. This reasoning is verified by the results. The graphs for DE f/b are expected to be similar [see Fig. 8.19 (top)].

A graph of the difference between the DE-f/b $C(\tau)$ & the TX-f/b $C(\tau)$, would be a good measure of the correlation of decoding errors due to incorrect syndrome-resetting [see Fig. 8.19 (bottom)]. For QE=1, there is no visible correlation, but for QE = 2 & 3 there is a strong one, especially for $\tau \leq 14$ ($\frac{1}{2}$ of the decoder's memory). For QE=5, the correlation, due to error propagation, reaches a sharp maximum for $\tau = 14$, most probably because this is where the 'average' decoding-error is fed back (the centre of the decoder's memory). See also Fig. A8.11.3 (top), p. 580.

Fig. 8.20 shows $R(\tau)$ with DE f/b, normalized to $R(\tau)$ with TX f/b. This time, the lower-QE graphs are more 'swollen' than the higher-QE ones. This reversal is not due to correlations at low channel error-rates, but due to the average number of errors. Note that, since

$$R(\tau) = C(\tau) + E[D]^2 = E[D]^2 \left[C(\tau)/E[D]^2 + 1 \right] \quad \text{then:}$$

* More than the average.

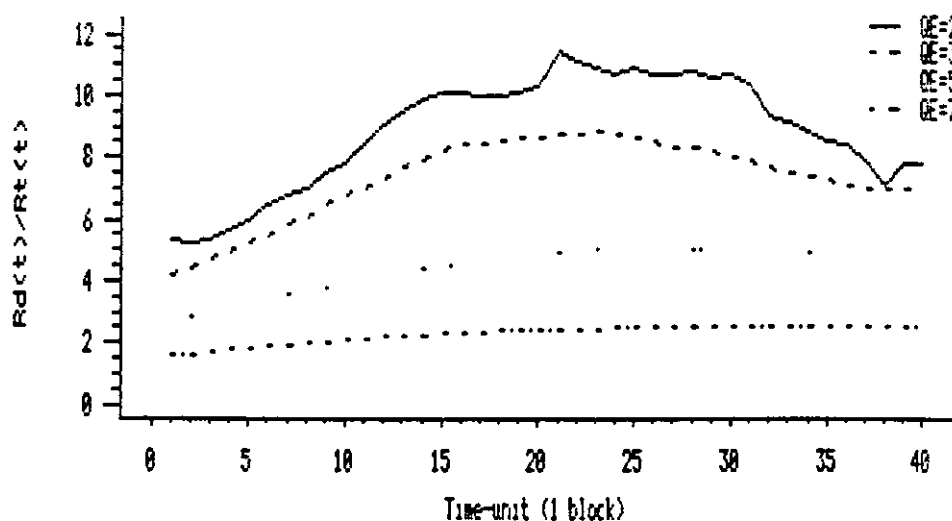


Figure 8.20: Autocorrelation fn with DE f/b, normalized to the TX-f/b autocorrelation fn, for the (28,4) code.

$$R_D(\tau)/R_T(\tau) = \left(E[D_D]/E[D_T] \right)^2 \left[C_D(\tau)/E[D_D]^2 + 1 \right] / \left[C_T(\tau)/E[D_T]^2 + 1 \right]$$

Then, as QE increases, $E[D_D]$ & $E[D_T]$ become strong, enough to 'cover' the variations of $C(\tau)$ with τ , hence the ratio tends to $(E[D_D]/E[D_T])^2$ with QE increasing, as has been verified by the simulation results.

Figs A8.11.3 (middle & top) show the normalized autocorrelation fn for the (60,6) code, with DE & TX f/b.

Another way to conclude about the effects of error propagation is via graphs of the difference between the net coding-gains of DE & TX f/b. Figs 8.21 & 8.22* verify earlier conclusions about the effect of the decoding f/b mechanism. Notice that at high Γ_s , there is virtually no loss due to incorrect syndrome-resetting. While the TX-f/b characteristic decreases almost linearly with Γ , the DE-f/b one decreases fast, initially (due to the extra errors fed back in the syndrome register), but then 'flatens out' because the decoding errors stop increasing the number of errors the decoder has to cope with, and start cancelling them.

The amount of signal power-loss and the Γ at which this occurs, depend on the code, but it seems to be between 1 & 2 dBs, at Γ_s between 4.5 & 7 dB [see also Fig. A8.11.6 (top)].

Similar information may be obtained from graphs of the % increase in decoding errors due to DE f/b, relative to TX

* See also Figures A8.11.4-5 (pp. 581-2).

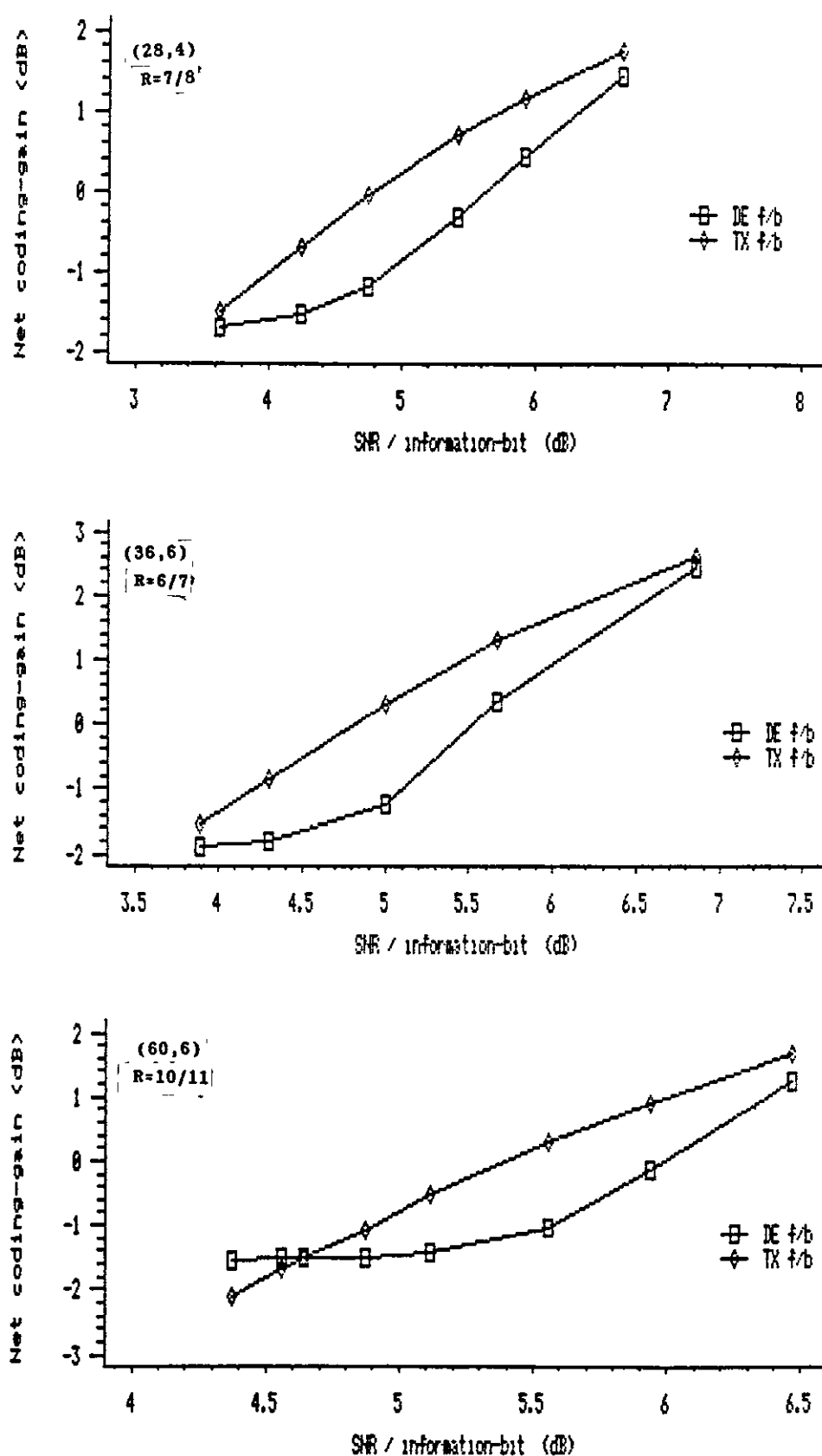


Figure 8.21: Net coding-gain vs Γ , with DE & TX f/b, for the (28,4) code (top), the (36,6) code (middle) & the (60,6) code (bottom). *

* Γ = signal-to-noise power-ratio per information-bit.

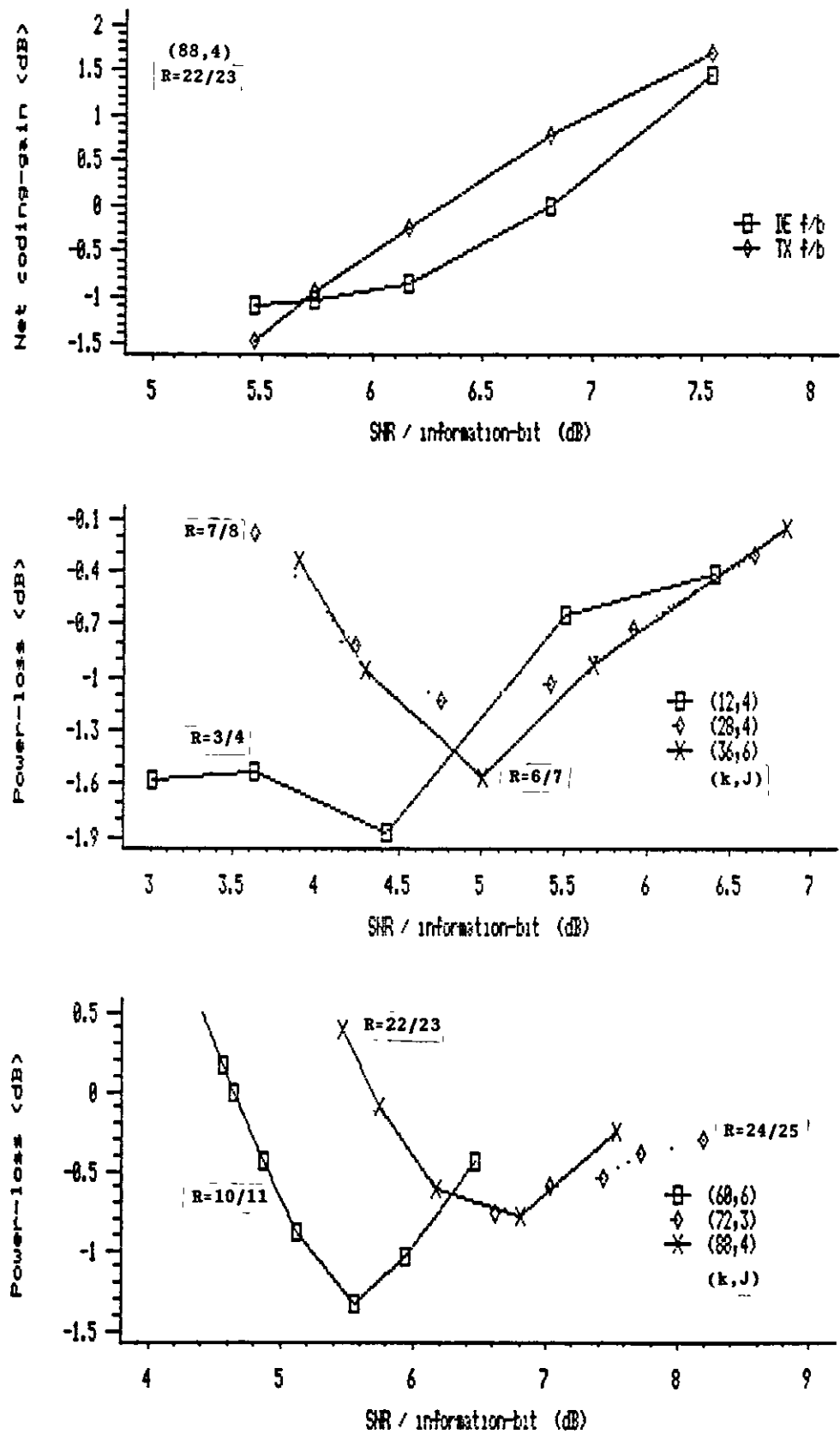


Figure 8.22: G vs Γ , with DE & TX f/b, for the (88,4) code (top); power-loss vs Γ due to DE f/b (middle & bottom). *

* Γ = signal-to-noise power-ratio per information-bit.

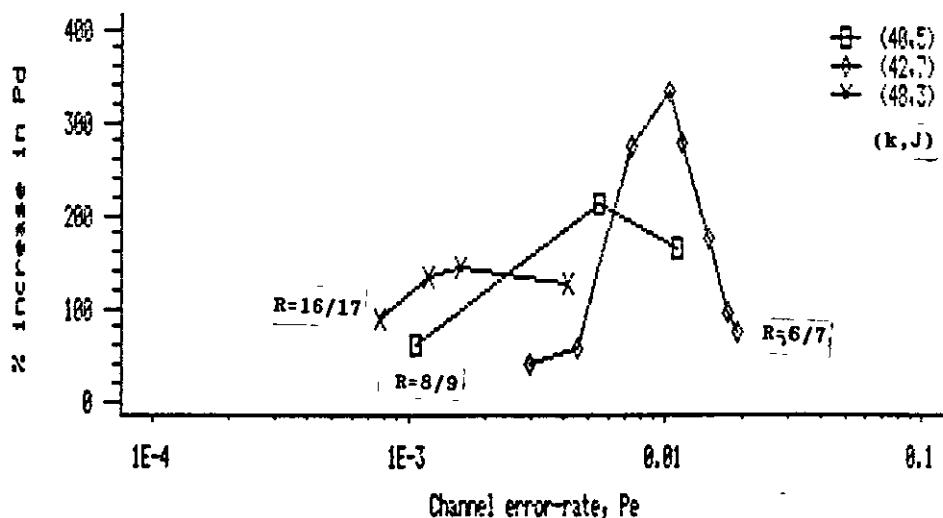


Figure 8.23: % increase in P_d due to DE f/b vs P_e .

f/b (see Figs 8.23 & A8.11.6). At worst, P_d increases between 150% & 300%. This occurs at channel error-rates between 10^{-3} & 10^{-2} . It is evident that the worst loss increases with J ($\approx 150\%$ for $J=3$, $\approx 170\text{--}200\%$ for $J=4$ & 5 , $\approx 300\%$ for $J=7$) and also that it occurs at higher channel error-rates, for shorter codes. Most probably, this is due to the ratio J/k , the rate at which J errors (due to decoder failure) are injected into a syndrome register of k blocks.

8.7 UNEQUAL ERROR-PROTECTION

The probability of decoding error, P_d (see Chapter 6), depends on the sizes of the set of syndromes checking on each error-bit. Since this set is special for each bit to be decoded, then P_d is expected to be different for each one of them. For type-C5 codes, the 'sizes' are the elements of the corresponding columns of the IA, hence identical for the J bits of a coset.

Fig. 8.24 (top), shows the number of decoding errors* per coset, for the (40,5) code, with TX f/b. Notice the close agreement between the simulation & the theoretical results**. Figs A8.12.1 (top & bottom) (p. 584) show the results for $QE = 1$ & 10 .

From reln (6.61) (p. 177) it seems that, for a given er-

* 10,000 blocks were considered, at $QE=4.99$.

** Equation (6.38b) was used for the theoretical calculations.

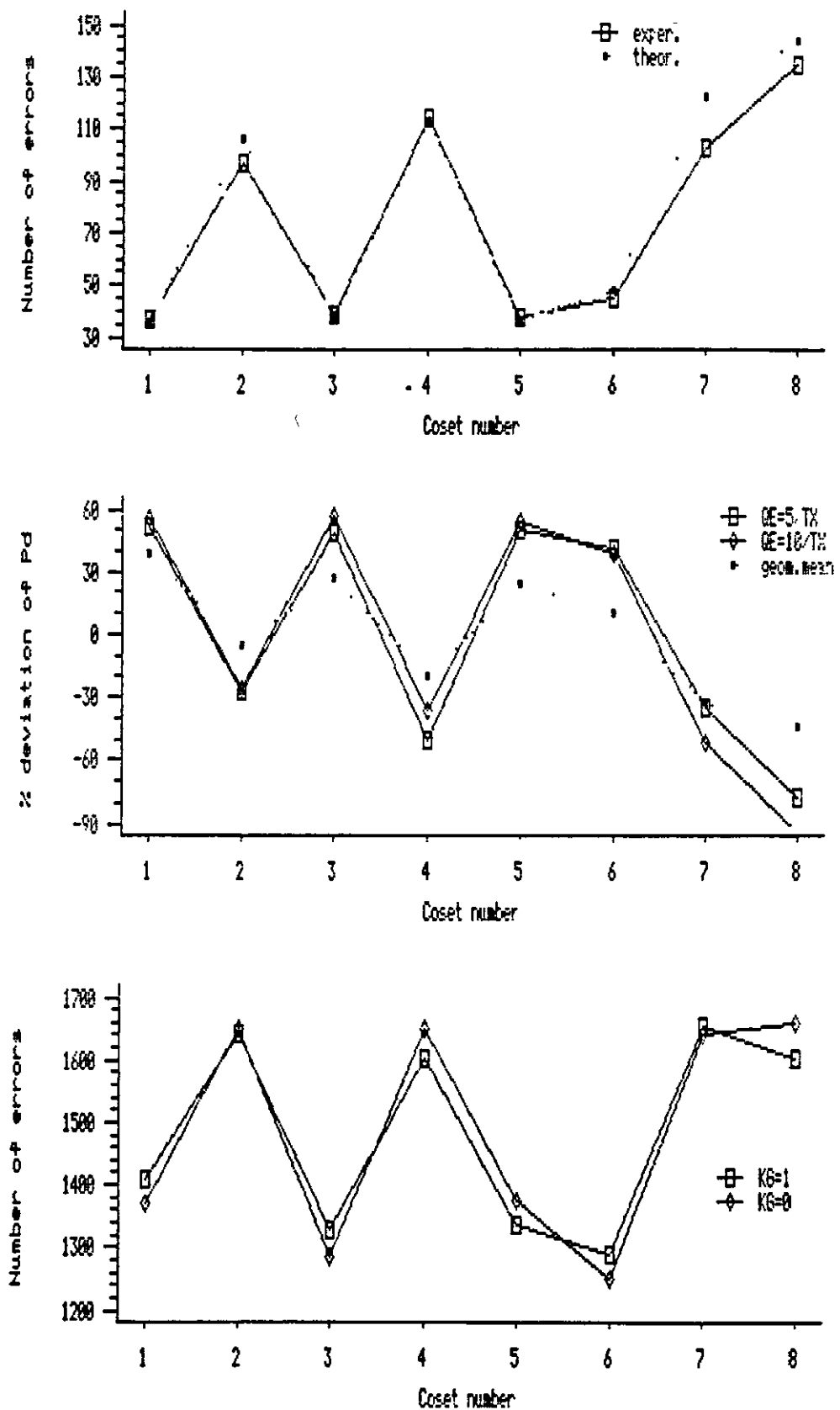


Figure 8.24: For the (40,5) code: No of decod. errors (QE=5 - TX f/b - top); % deviation from the average (TX f/b - mid.); No of decod. errors (DE f/b - QE=10 - bot.).

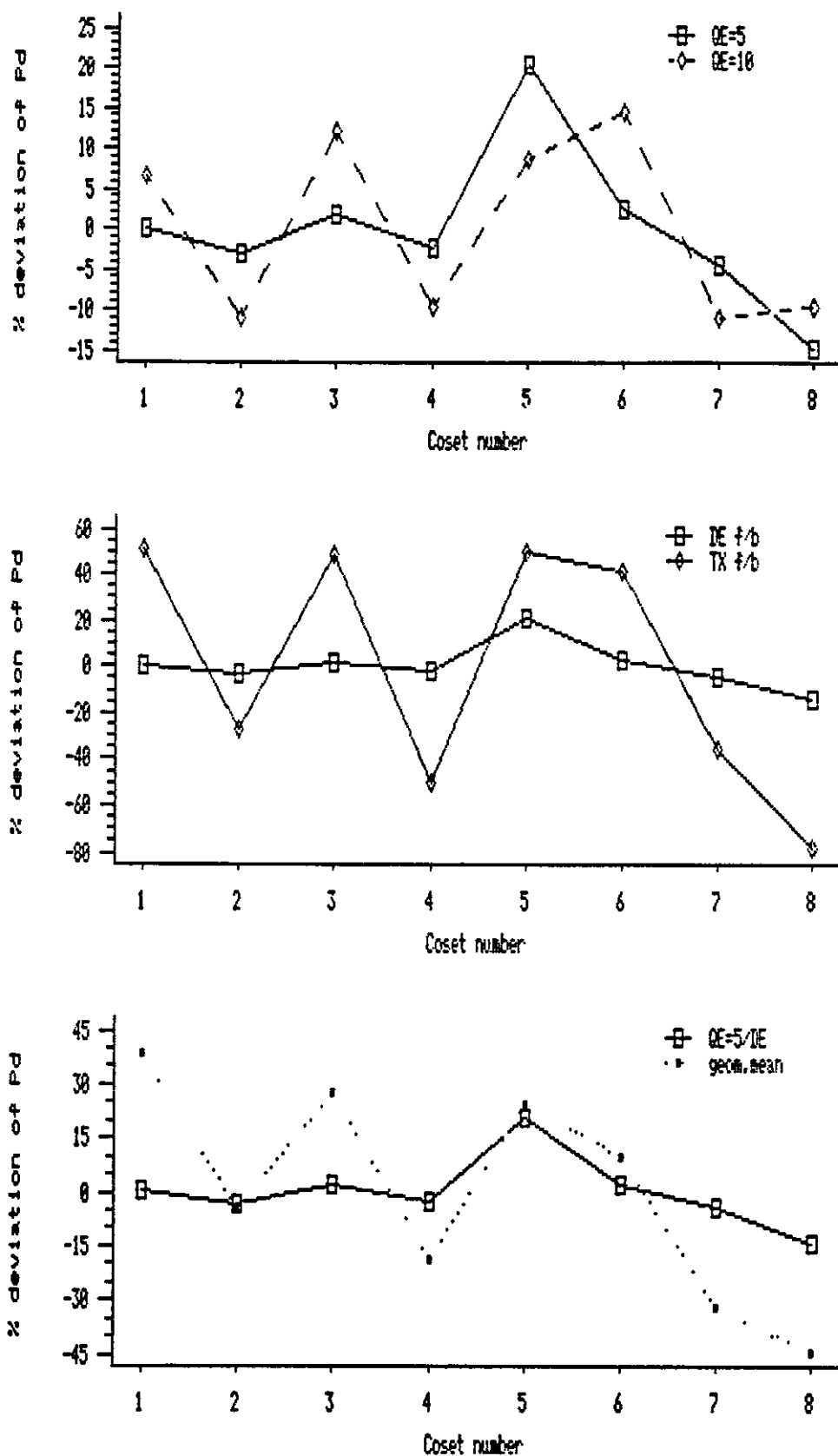


Figure 8.25: % deviation of the coset-distribution from the average, for the (40,5) code: DE f/b (top); QE=5 (middle); QE=5 & DE f/b (bottom).

ror bit, P_d increases with C_{T_n} , the T_n th generalized mean of the syndrome sizes of the corresponding coset, where T_n is the nominal threshold ($\lceil J/2 \rceil$). To simplify this, and since the generalized means are very close to each other, the geometric mean C_j is considered. Fig. 8.24 (middle) shows the % deviation from the average, of the P_d of each coset, with TX f/b, at QE = 5 & 10. Also plotted is the geometric mean of the syndrome sizes. Notice that the latter follows very reliably the coset P_d -distribution. Note also that the % deviation, for the two different channel error-rates, is virtually the same.

Consider now the number of decoding errors, per coset, with DE f/b, from two different sequence of noise samples* [Fig. 8.24 (bottom)]. The consistency indicates that the performance of each coset is not the result of statistical fluctuations.

Fig. 8.25 (top) indicates that with DE f/b, the various cosets have approximately the same behaviour, at various channel error-rates. Fig. 8.25 (middle) shows that the syndrome f/b mechanism moderates the non-uniformity of the coset P_d -distribution. Nevertheless, the geometric mean still indicates the cosets that can offer better error-protection [Fig. 8.25 (bottom)]. See also Fig. A8.12.2 (top).

For comparison, the number of decoding errors per coset, under definite decoding, has been plotted in Fig. A8.12.2 (bottom). As expected, the distribution is 'flat'.

8.8 CONCLUSIONS

A great number of the codes discovered by McQuilton [42] were tested over a computer-simulated BSC. The FORTRAN programmes used, required a minimal input for each run, which could, reliably, produce a number of points for the $G = f(\Gamma)$ graph**, of any type-C5 code. Approximately twenty versions of the *main programme* were used to generate a variety of performance results. The input data required, consisted of the code parameters k & J , the minimum number of blocks

* 10,000 blocks each, at QE = 10.

** G = net coding-gain; Γ = signal-to-noise power-ratio per information-bit.

(NDMB) to be decoded, the minimum number of channel errors for each point* and the channel error-rates the code was to be tested on. The basic returns from the programme were the number of channel & decoding errors. These were processed to produce results like G , P_d , P_e , Γ , EER, etc. Variations of the main programme could implement feedback & definite decoding, as well as 'genie' decoding. Other programmes could produce results for each coset, calculate autocorrelation functions, etc.

Because the type-C5 codes have a large constraint-length $[k(k+J)]$, attempts to simulate $k \geq 160$ -codes hit the computer-memory limit [the decoder memory is $k^2 + J + kJ$, to which the encoding & syndrome arrays have to be added]. In order to test *long codes*, a special decoder-implementation was used which stored b bits of data in one word (b is the computer's word-length). The problem to be solved was not an easy one, because the decoder memory, IRA, of general dimensions $k \times k$, had to be composed of a minimal number of sub-arrays of dimensions $1 \times b$, and horizontal & vertical cyclic-shifts of the contents of IRA to be possible, as well as writing in, and reading of, selected bit-positions. This decoder allowed the simulation of codes with a k longer by a factor of $\approx \sqrt{b}$ and decreased the processing time by a factor of 2 to 3 (because the bit operations are faster).

For the general decoder to operate, the encoding & syndrome arrays have to be supplied, given k & J . This required the support of a *number-theoretic library* of subroutines [including prime decomposition, primitive roots & the calculation of elements of order $J \bmod (k+1)$]. These subroutines had to operate safely** over the entire range of the computer's positive integers. Furthermore, the presentation of the various results required the support of another library of subroutines which could, calculate the net coding-gain, invert eqn $P_e = \frac{1}{2} \text{erfc}(\sqrt{\Gamma})$, calculate theoretical results (for comparison), etc.

If the simulation results were to be useful & reliable, the quality of the programmes & the confidence on their integrity should be unlimited. To this end, all routines were tested under extreme conditions and optimized (both for

* NDMB was increased, if necessary.

** Without overflow.

processing-time & memory-requirement). The main programmes were built in stages, each of which was exhaustively tested; whenever possible, tasks were diverted to subroutines. Simple programmes were built first, and tested manually for accuracy; they were subsequently used as the measure of the integrity of a more sophisticated programme. Extra features were added only when the older & the new versions, produced identical results.

All data to be generated by the programmes, stemmed from the observed numbers of channel & decoder errors. The confidence on these data depends on their number. The $G=f(\Gamma)$ graph suffers from an uncertainty on both Γ & G . The uncertainty on G originates from both P_e & P_d . With d errors, the relative width of the 99% confidence-interval is $\approx 2.58/\sqrt{d}$, for a large-enough sample-size (which was the case). Care was taken to generate at least 300-400 channel errors, so that the uncertainty on P_e (& Γ) to be insignificant. For the 'difficult' case of high Γ_s , P_d is 1-2 orders of magnitude less than P_e , while the computer processing-time limitations did not permit runs of more than 10^7 bits. Inevitably, the high- Γ results suffer from uncertainty, on P_d .

From the graphs produced it was concluded that the $G=f(\Gamma)$ relation would have been an almost linear one, had it not been for the effect of *incorrect syndrome-resetting*, due to decoding errors. As Γ decreases, this mechanism initially accelerates the power loss, but it subsequently puts a brake on it and eventually stops it, because of the overwhelming (and accidental) error-cancellation in the syndrome register*. The worst power-loss is of the order of 1-2 dB, which occurs at Γ_s between 5-7 dB (depending on the code). The corresponding increase in decoding errors increases with J and is of the order of 150-300% (for the codes tested). The worst error-propagation occurs at a channel error-rate, P_e , which is lower for longer codes; in general, this P_e seems to decrease with J/k , the rate at which the decoder output adds errors to the syndrome register.

The slope of the $G=f(\Gamma)$ characteristic increases with J . A large J , is the main contributor to a high asymptotic net

* Due to the feedback of erroneous decoding-decisions.

coding-gain, while at low Γ_s it floods the syndrome register with many errors, hence it contributes to a lower G . The importance of J is such that (at moderate to high Γ_s), if increased by 1, G may increase by 0.5-1 dB.

The effect of an increase in k is the weakening of the code *relative error-correcting capability* [$\lfloor J/2 \rfloor$ errors in $k(k+J)$ bits], but also a better *noise-averaging capability*. Hence, for a given k high- J codes are useful over a wider range of Γ_s , while for a given J long codes are useful at higher Γ_s .

The best G_s (2-3 dB) were returned from high- J (≥ 7) codes, at $\Gamma > 6$ dB.

The *correlation* among decoding errors is partly due to the random channel-fluctuations which are stored in the decoder memory; nevertheless, at moderate-to-low Γ_s correlations are mainly due to *error propagation*. In such a case, a decoder failure at time τ , greatly affects the decoding at times $\tau+1, \tau+2, \dots, \tau+\frac{1}{2}k$; on occasion the $(\tau+\frac{1}{2}k)$ th block is affected more than the rest. From the results displayed, it seems that the autocovariance function of the decoder output error sequence (with 1-3 blocks per time-unit) is the best indicator of correlations; the error propagation can be studied by the *autocovariance difference* between the DE-f/b & the TX-f/b modes.

The codes offer *unequal error-protection* to each coset. A reliable indicator of the protection per coset is the *geometric mean* of the associated syndrome sizes. Deviations from the average P_d , were up to $\pm 20\%$, for the (40,5) code. Incorrect decoder-feedback tends to neutralize these deviations, but this is expected to occur at moderate-to-low Γ_s .

Some of the codes were tested, under FD and with a *syndrome threshold*, T , different than T_n , the nominal one ($T_n \hat{=} \lfloor J/2 \rfloor$). As predicted in Chapter 6 (see § 6.3.4., p. 175), performance was improved with an increase of T_n by 1, when the channel error-rate exceeded $1/(C_{T_n})^2$ *. When, with the nominal threshold, G fell to 0 dB, the use of a T_n+1 threshold increased G up to 1 dB. Furthermore, the use of the optimum threshold increased the range of Γ_s over which the code is useful and kept G above 10logR dB.

* C_μ is the μ th *generalized mean* of the syndrome sizes.



Conclusions & Further Work

Codes & decoding techniques are tested over a binary-in, binary-out, *channel model* with, or without, memory. The BSC* is the most suitable one because of its simplicity & widespread use; it is also argued that it is (usually) more 'demanding' than a bursty channel, so that any performance data obtained over the latter are expected to paint a more optimistic picture. A BSC can be realized with a binary PSK modulator with hard-decision coherent demodulation and transmission over the additive white Gaussian-noise channel. If E is the energy per information bit at the demodulator input and $\bar{n}/2$ is the double-sided noise power spectral-density, then the probability of a bit in error at the demodulator output is $P_e = \frac{1}{2}\text{erfc}[\sqrt{(R\Gamma)}]$, where $\Gamma \triangleq E/\bar{n}$. The performance of a coding scheme is usually assessed via the decoding-error probability (P_d) vs P_e relation, or via the P_d vs Γ one; an uncoded system operating at an increased signal-to-noise power ratio, Γ , is used for reference. The extra signal-power, G , required by the uncoded system so that it matches the error performance of the coded one, is called the *net coding-gain* of the coded system; the $G=f(\Gamma)$ relation is one of the best measures of its performance.

The current channel block of an (n,k,m) *convolutional code* (CC) depends on the current and the past m information blocks.** The minimal encoder for a linear (n,k,m) CC is, usually, the *normal encoder*, a k -in, n -out, linear-sequential circuit (LSC) made of k shift registers (SRs) of lengths between 0 & m and up to n X-OR gates. The code is completely described by the kn transfer functions of the encoder, called the *generator polynomials*. If the high-

* Binary symmetric channel - a memoryless channel.

** A block code is a CC with $m=0$.

degree polynomials correspond to one or two of the encoder's outputs, the *type-II* encoder may be the minimal one.

Although the CC generates a single codeword, a relationship (developed by the author) is possible, between channel blocks h to $h+z$, and information blocks $h-m$ to $h+z$, via a finite-dimensioned 'generator' matrix $[G]_z^h$ (where $h \geq m$ and $z \geq 0$). It was then proved that the CC generator matrix, G , may be defined as the limit of $[G]_z^0$, as $z \rightarrow +\infty$.

CCs may be described either via the *matrix approach*, or via the *polynomial approach*. The former, groups the bits of a block into a vector and results in infinite-dimensioned matrices. The latter, groups the bits of a port into a polynomial, making use of the concept of transfer functions and of finite-dimensioned matrices.

The *parity-check matrix*, H , was proved (by the author) to be equal to the limit of $[H]_z$, as z tends to $+\infty$, where $[G]_z^0[H]_z^T = 0$ / $z \geq 0$. H is used to define the *syndrome*, $s \triangleq rH^T$. For an additive channel, $s = eH^T$, which means that the syndromes are linear combinations of the current and the past m error blocks.

The CC maximum-likelihood decoding technique is based on the encoder's *trellis-diagram*, which is its state-transition diagram expanded in time. This technique suffers from processing-time limitations, arising from the complexity of the trellis. The decoder, for each received block, and for each of the states of the trellis, has to calculate 2^k metrics, choose the best and store it, together with the sequence of information bits leading to this state, at that time-unit. Since, for an encoder of total memory M , there are 2^M states, hardware limitations do not permit the employment of long codes ($2^M 2^k$ calculations per block time-unit & storing of 2^M information sequences). A quantitative relation between trellis-complexity and code-parameters, may assist in the design/choice of more suitable codes.

The memory of a normal encoder (and of any similar LSC) may be partitioned into three groups: The FEG (which will store the next I/P block), the REG (which loses its contents with each transition) and the CEG (the rest). This

approach helped to develop relations between trellis-complexity and code-parameters. For instance, if f denotes the number of rows of $G(D)^*$ with 1s & 0s only, the number of transitions into (or out of) a state is 2^{k-f} , while each transition corresponds to 2^f different input-blocks. There are 2^{k-f} self-loops, whose existence-conditions are given by Theorem 3.5 (p. 63).

In an effort to use longer codes and trellis decoding, the idea of a *constrained trellis*** arises. This is obtained from the ordinary one, by imposing a limit, t , on the sum of the Hamming weights of the current state & input.

Using the memory partition of a *normal LSC* and the encoder parameters, Theorem 4.21 (p. 107) provides expressions about the number of states of a given weight, the number of transitions from a given state to a state of a given weight, etc, as well as existence conditions.

The *simplified trellis* is obtained, from the constrained one, by removing all states with a single transition, introducing thus *long transitions*. Theorems 4.26 & 4.27 used a refinement of the memory-partition technique, mentioned above, to produce expressions about the number of states of a given weight w , from which a transition of a given length β may start, as well as the number of these transitions. These results are expressed in terms of k , M , t , β , w & the *memory-distribution function* F , where $F(i)$ is the number of encoder SRs of length $\leq i$.

For the special case of $t=1$, the simplified trellis has only one state, S_0 , and transitions of length 1, $1+M_1$, $1+M_2, \dots, 1+M_k$, where M_i is the length of the i th SR ($1 \leq i \leq k$).

Similar results were obtained for the special cases of $t=2$, LSC with equal-length SRs & a 1-SR LSC (see Sec. 4.7).

A decoding technique, based on the constrained trellis, has been proposed by Reed & Truong [24]. The encoder trellis cannot be constrained because all possible channel-sequences, v , are equally probable. This is not the case with e , though, whose frequency of appearance decreases exponentially with its Hamming weight. The syndrome eqn was solved, $e = [z, zP+s]$ and z (the error on the message bits) had to be

* The $k \times n$ matrix of polynomial generators.

** The idea is due to Reed & Truong [24].

chosen so that e was minimized. To this end, the trellis corresponding to P was constrained and driven by s , using the Viterbi algorithm. The simplified trellis* has a reduced complexity and hence the potential to simplify decoding.

Majority-logic is a, syndrome-decoding, technique applicable to systematic CCs with a special structure. According to the decoding rule, if J syndromes are orthogonal on an error bit, then this is estimated to be 1, only if more than $T = \lceil J/2 \rceil$ syndromes are 1. If all syndromes checking on an error bit are orthogonal on it, the code is *self-orthogonal*, and then $J = d_{\min} - 1$.

It was proved by Massey [18] that $T = \lceil J/2 \rceil$ guarantees correct decoding, provided that no more than $\lfloor J/2 \rfloor$ errors have occurred among the n_e^{**} error bits checked by the J syndromes. It has been argued by the author (see Chapter 6), that as P_e increases, the requirement of no more than $\lfloor J/2 \rfloor$ errors becomes hard to satisfy. If the probability of decoding error, P_d , is used as the performance criterion, then the value of T that minimizes P_d , increases with P_e , from a minimum of $\lceil J/2 \rceil$ to a maximum of J . For syndromes of size c (usually, the case of DD), it was proved that $T_0 = \left\lceil \frac{1}{2}J + \frac{1}{2} \ln[P_e/(1-P_e)] / \ln[P/(1-P)] \right\rceil$ is the optimum threshold, where $P = \frac{1}{2}[1 - (1 - 2P_e)^c]$. It was proved, also, that $T_0 = \lceil J/2 \rceil$, for $P_e < 1/c^2$.

The case of J syndromes of various sizes, c_i , results in 'cumbersome' expressions with multiple summations of products of probabilities. It was argued, though, that there must be an 'average' syndrome-size, which could play the role of c . To this end, the concept of the μ th generalized mean, X_μ , of the quantities x_1, x_2, \dots, x_J was defined (by the author), as the μ th root of the arithmetic mean of all the distinct products of x_i 's, taken μ at a time [$C(J, \mu)$ such products]. It was proved that X_1 is the arithmetic mean & X_J the geometric mean. It was also found that $X_1 > X_2 > \dots > X_J$. If A_μ is the μ th generalized mean of $K_i \triangleq P_i/(1-P_i)$, where $P_i = \frac{1}{2}[1 - (1 - 2P_e)^{c_i}]$ ($i=1, 2, \dots, J$), then the analysis resulted in $T_0 = \left\lceil \frac{\ln[P_e/(1-P_e)] + J \ln A_{J-T_0}}{\ln(A_{J-T_0} A_{T_0})} \right\rceil$, an eqn that can be solved fast, by trial and error, if the bounds and approximations on T_0 are used (see § 6.3.3., p. 171). The bounds

* Proposed by the author.

** Effective constraint-length.

obtained (upper & lower) differ by 0, 1 or 2 (in all cases considered). $T_0 \approx \lceil \frac{1}{2}J + \frac{1}{2}\ln[P_e/(1-P_e)] / \ln A_{T_n} \rceil$, is expected to be accurate for most cases of practical interest ($T_n \triangleq \lceil J/2 \rceil$). It was shown that T_0 becomes T_n+1 , at $P_e \approx 1/(C_{T_n})^2$, where C_μ is the μ th generalized mean of the syndrome sizes (hence it represents the average syndrome-size).

McQuilton [42] discovered a class of cyclically-decodable convolutional self-orthogonal codes (CCSOCs). A study of his approach concluded that his results were the partial solution to the problem of, constructing self-orthogonal (SO) CCs under three 'arbitrary' constraints. It was thereafter concluded that an alternative approach to the problem, of systematically constructing (n,k,m) systematic SO CCs, may be via an $(n-k) \times (m+1)$ array of cells of integers in the range $[1,k]$ [called the *initial array (IA)*]. The j th cell gives the numbers of the error-bits, from $(e^m)_{h-1+1}$, checked by $s_h^{(j)}$. Necessary and sufficient conditions can then be developed on these numbers, so that the code is SO. Restrictions are imposed, if a practical solution seems unfeasible. To this end, the population of each cell was restricted to one* and necessary & sufficient conditions on the elements of the IA were developed (Theorem 7.5). Subsequently, congruence $a_{i,j} \equiv ja_{i,1} \pmod{k+1}$ * was used to reduce the problem to that of determining $n-k$ integers $(a_{i,1} / i=1,2,\dots,n-k)$ so that the code is SO (*type-B codes*).

The insistence on discovering the necessary & sufficient conditions, for a systematic CC to be SO, may make the problem more difficult, but it has the potential to discover all such classes of codes. For this reason, the introduction of the restrictions was delayed as much as possible, so that general results can be developed. Some new classes of systematic SO CCs were discovered.

Type-B1 are $(2k,k,J-1)$ codes ($k=\text{even} \ \& \ J < p$).** *Type-B2* are $(p^2-1+J(p+1), p^2-1, p-2)$ codes ($p=\text{odd prime} \ \& \ J < p$).** *Type-B3* are $(pq-1+J(q+2), pq-1, p-2)$ codes (p,q are odd primes with $p < q < 2p$ and $J < q/2$).** *Type-B5* are $(k+J, k, k-1)$ codes ($k=\text{even} \ \& \ J < p$).** *Type-B4* are $(k+2, k, k-1)$ codes with 2 syndromes checking on each error bit ($k=\text{odd}$). Some other type-B

* One of the three properties of the CCSOCs.

** J syndromes check on each error bit - p is the smallest prime factor of $k+1$.

SO codes were discovered via a simple computer-search.

Type-C codes were defined to be any type-B codes whose error bits are cyclically decodable*. Necessary conditions for the existence of type-C codes restricted the freedom of choice on both the code parameters and the IA elements. Only type-B1 (with $k+1=\text{prime}$) & type-B5 [with J a divisor of $\theta(k+1)$, where $\theta(k+1)$ is a number-theoretic function introduced by the author] can be of type-C. Type-B4 codes can also be decoded in some cyclic manner. Finally, the IA has to be mapped into another array so that bits are decoded in their 'natural' order. More than one ways can be proposed, the final choice being dependent on the particular decoder-implementation adopted. Type-C5 codes can be decoded with k/J majority gates, while type-B4 with just one (other implementations are possible). The CCSOCs discovered by McQuilton, are the type-C5 & the type-B4 codes.

It was proved that the only necessary & sufficient condition for the existence of type-C5 codes is the existence of an element $\beta (= a_{1,1})$, which has order J modulo any non-trivial divisor of $k+1$. A formula for the calculation of β has been obtained. The IA of type-C codes has a rich mathematical structure which is less 'predictable' if $J=\text{odd}$. It was proved that for $J=\text{odd}$ codes the 1st column of the IA is made of quadratic residues modulo any prime factor, p , of $k+1$, while the quadratic character of the elements of the z th column is $(z|p)$. For $J=\text{even}$, the effective constraint-length is $n_g=1+(k+1)J/2$. For $J=\text{odd}$ though, it was proved that $(n_g-1)/(k+1) \in [(J+1)/2, J-1]$, hence the $J=\text{even}$ codes are at least as good as the $J=\text{odd}$ codes. For the special case of rate-2/3 codes, with $k+1=\text{prime}$, n_g-1 equals the sum of quadratic non-residues (mod p)**. It is anticipated that n_g does not have a closed-form expression, for $J=\text{odd}$.

A large number of type-C5 codes were tested over a computer-simulated BSC. The best *net coding-gains* (2-3 dB) were returned by high- J (≥ 7) codes, while the worst *error-extension ratio* did not exceed 2.5. The *slope of the $G=f(\Gamma)$ characteristic* increases with J (because of an increased asymptotic G), while an *increase in k* displaces the characteristic towards higher values of Γ (due to the weakening

* One of the three properties of the CCSOCs.

** A closed-form expression for this sum, remains an unsolved mathematical problem.

relative error-correcting capability, $\lfloor J/2 \rfloor / [k(k+J)]$).

Incorrect syndrome-resetting resulted in a worst power-loss of 1-2 dB (at Γ_s between 5-7 dB). The $G=f(\Gamma)$ characteristic would have been an almost linear one, without error propagation. This effect accelerates the power loss at high Γ_s , but does the opposite for moderate values of Γ and eventually stops it. Interesting results about error propagation can be obtained from the autocovariance function, $C(\tau)$, of the decoder-output error-sequence (the number of decoding errors per b blocks) with a small value of b (1-3). The $C(\tau)$ -difference between (normal) feedback (f/b) decoding and decoding with perfect f/b ('genie' decoding) seems to be a very sensitive indicator of the 'character' of this effect, at various channel error-rates.

The codes offer *unequal error-protection* per coset. A reliable indicator of the error-performance of each coset is the geometric mean of the syndrome-sizes associated with it. Error propagation, though, tends to neutralize this effect.

The use of the *optimum threshold* improved the code performance by 0.5-1 dBs, when G fell to 0 dB with the nominal threshold. It also extended the range of Γ_s over which the code is useful ($G > 0$). As predicted, the nominal threshold had to increase by 1, when P_e exceeded $1/(C_{tn})^2$.

The decision to design and build simulation programmes for the general type-C5 code was a correct one, both because it saved time, but also because it prevented any uncertainty about the correctness of the simulation results. The use of one computer word to store b bits of data, not only reduced (by a factor of b) the total storage required, but also improved the processing time by a factor of 2-3. The main problem there was the design of support routines which could efficiently permit the synthesis of a $k \times k$ array (the main decoder-memory), using a minimum number of small $1 \times b$ sub-arrays; this had to be done in such a way that cyclic shifts (vertical or horizontal) and writing in, or reading from, selected bit-positions was possible. The software implementation of the decoder required the use of a number-theoretic library, whose most powerful routine was the one which could

(efficiently) calculate $a^b \pmod m$, without overflow, however large a , b & m may be. Finally, it is worth mentioning that type-C5 codes constitute a very large family; to facilitate a search, subroutines were developed which return the type-C5 code best matching the code parameters k & J , in a specified priority (k , J or R).

Further work is required on the results of Chapters 4, 6 & 7. The decoding algorithm, using a *simplified trellis*, must be improved & generalized*. The algebra linking the *trellis complexity* with the code parameters, must be used in an effort to select, and attempt to design, codes which are more suitable for trellis decoding. Simulation results are required in order to determine the performance degradation of various codes, as a function of the weight-constraint, t , of the trellis. Also, the 'complexity-gain' of the constrained trellis can be determined by simulating codes of different capabilities, but adjusting t so that they end-up with the same decoding complexity.

Simulation results for other classes of codes, using the *optimum threshold*, would be very useful in an effort to determine gain-returns against code-parameters (length, rate, J , etc). It is expected that large- J codes may have to increase their threshold by 1, more than once, as Γ deteriorates. In addition, a threshold-adjusting mechanism is required for channels with memory (for instance, the number of estimated-errors per constraint-length); simulation-results could prove the optimum threshold to be a more valuable proposition for *bursty channels*. A problem-area, though, is the synchronization between threshold adjustments and channel 'relapses'. It is possible that the code parameters, especially its length, are critical in this respect.

New *code designs* can be pursued from the general results of Chapter 7. For example, the consideration of an IA with two numbers per cell and a cyclic-decodability property, or the computer-aided 'shortening' of type-C5 codes, could produce some useful results. The performance of the new codes discovered (type-B1-5) should, also, be assessed.

* Beyond $t=1$ - see Note 4.8 & Example 4.4 (pp. 123-6).

