Turbo Coding, Turbo Equalisation and Space-Time Coding
for Transmission over Wireless Channels

by

©L. Hanzo, T. H. Liew, B. L. Yeap
Department of Electronics and Computer Science,
University of Southampton, UK

We dedicate this monograph to Rita, Ching Ching, Khar Yee and to our parents as well as to the numerous contributors of this field, many of whom are listed in the Author Index
Dear Martin, dear Zöe, would you be so kind as to finding a little space somewhere in the Prelims for the contributors listed in the style seen below?

Contributors of the book:
Chapter 4: J.P. Woddard, L. Hanzo
Chapter 5: M. Breiling, L. Hanzo
Chapter 7: T.H. Liew, L.L. Yang, L. Hanzo
Chapter 8: S.X. Ng, L. Hanzo
Contents

Acknowledgments xvi

1 Historical Perspective, Motivation and Outline 1
  1.1 A Historical Perspective on Channel Coding 1
  1.2 Motivation of the Book 3
  1.3 Organisation of the Book 5

I Convolutional and Block Coding 11

2 Convolutional Channel Coding 13
  2.1 Brief Channel Coding History 13
  2.2 Convolutional Encoding 14
  2.3 State and Trellis Transitions 16
  2.4 The Viterbi Algorithm 18
    2.4.1 Error-free Hard-decision Viterbi Decoding 18
    2.4.2 Erroneous Hard-decision Viterbi Decoding 20
    2.4.3 Error-free Soft-decision Viterbi Decoding 23
  2.5 Summary and Conclusions 25

3 Block Coding 27
  3.1 Introduction 27
  3.2 Finite Fields 28
    3.2.1 Definitions 28
    3.2.2 Galois Field Construction 31
    3.2.3 Galois Field Arithmetic 32
  3.3 Reed-Solomon and Bose-Chaudhuri-Hocquenghem Block Codes 35
    3.3.1 Definitions 35
    3.3.2 RS Encoding 36
    3.3.3 RS Encoding Example 38
    3.3.4 Linear Shift-register Circuits for Cyclic Encoders 41
      3.3.4.1 Polynomial Multiplication 41
      3.3.4.2 Systematic Cyclic Shift-register Encoding Example 42
    3.3.5 RS Decoding 45
### CONTENTS

3.3.5.1 Formulation of the Key Equations ........................................ 45
3.3.5.2 Peterson-Gorenstein-Zierler Decoder .................................. 49
3.3.5.3 PGZ Decoding Example .................................................... 51
3.3.5.4 Berlekamp–Massey Algorithm .......................................... 56
3.3.5.5 Berlekamp–Massey Decoding Example ................................. 62
3.3.5.6 Computation of the Error Magnitudes by the Forney Algorithm ... 66
3.3.5.7 Forney Algorithm Example .............................................. 69
3.3.5.8 Error Evaluator Polynomial Computation ............................... 71
3.4 Summary and Conclusions ....................................................... 74

4 Soft Decoding and Performance of BCH Codes ............................... 75
4.1 Introduction .............................................................................. 75
4.2 BCH codes ............................................................................. 75
  4.2.1 BCH Encoder .................................................................... 77
  4.2.2 State and Trellis Diagrams .................................................. 79
4.3 Trellis Decoding ...................................................................... 81
  4.3.1 Introduction ...................................................................... 81
  4.3.2 Viterbi Algorithm ............................................................... 81
  4.3.3 Hard-decision Viterbi Decoding .......................................... 84
    4.3.3.1 Correct Hard-decision Decoding .................................... 84
    4.3.3.2 Incorrect Hard-decision Decoding ................................. 85
  4.3.4 Soft-decision Viterbi Decoding ........................................... 86
  4.3.5 Simulation Results .............................................................. 87
    4.3.5.1 The Berlekamp–Massey Algorithm .................................. 87
    4.3.5.2 Hard-decision Viterbi Decoding ..................................... 89
    4.3.5.3 Soft-decision Viterbi Decoding ....................................... 89
  4.3.6 Conclusion on Block Coding ................................................. 89
4.4 Soft input Algebraic Decoding .................................................. 95
  4.4.1 Introduction ...................................................................... 95
  4.4.2 Chase Algorithms ............................................................ 96
    4.4.2.1 Chase Algorithm 1 .................................................... 99
    4.4.2.2 Chase Algorithm 2 .................................................... 101
  4.4.3 Simulation Results .............................................................. 101
4.5 Summary and Conclusions ....................................................... 102

II Turbo Convolutional and Turbo Block Coding .............................. 105

5 Turbo Convolutional Coding ......................................................... 107
  5.1 Introduction ........................................................................... 107
  5.2 Turbo Encoder ....................................................................... 108
  5.3 Turbo Decoder ....................................................................... 110
    5.3.1 Introduction .................................................................... 110
    5.3.2 Log Likelihood Ratios ...................................................... 111
    5.3.3 The Maximum A-Posteriori Algorithm ............................. 114
      5.3.3.1 Introduction and Mathematical Preliminaries .............. 114
      5.3.3.2 Forward Recursive Calculation of the \( \alpha_k(s) \) Values ... 118
      5.3.3.3 Backward Recursive Calculation of the \( \beta_k(s) \) Values ... 118
      5.3.3.4 Calculation of the \( \gamma_k(s, s) \) Values ...................... 120
CONTENTS

5.3.3 Summary of the MAP Algorithm . . . . . . . . . . . . . . . . 122
5.3.4 Iterative Turbo Decoding Principles . . . . . . . . . . . . . . . . . . . . 123
5.3.4.1 Turbo Decoding Mathematical Preliminaries . . . . . . . . . . 123
5.3.4.2 Iterative Turbo Decoding . . . . . . . . . . . . . . . . . . . . 126
5.3.5 Modifications of the MAP algorithm . . . . . . . . . . . . . . . . . . . . 129
5.3.5.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . 129
5.3.5.2 Mathematical Description of the Max-Log-MAP Algorithm . . 130
5.3.5.3 Correcting the Approximation — the Log-MAP Algorithm . . 133
5.3.6 The Soft-output Viterbi Algorithm . . . . . . . . . . . . . . . . . . . . . 133
5.3.6.1 Mathematical Description of the Soft-output Viterbi Algorithm 133
5.3.6.2 Implementation of the SOVA . . . . . . . . . . . . . . . . . . . 136
5.3.7 Turbo Decoding Example . . . . . . . . . . . . . . . . . . . . . . . . . 137
5.3.8 Comparison of the Component Decoder Algorithms . . . . . . . . . . . 147
5.3.9 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 150
5.4 Turbo-coded BPSK Performance over Gaussian Channels . . . . . . . . . . . . 150
5.4.1 Effect of the Number of Iterations Used . . . . . . . . . . . . . . . . . . 151
5.4.2 Effects of Puncturing . . . . . . . . . . . . . . . . . . . . . . . . . . . . 152
5.4.3 Effect of the Component Decoder Used . . . . . . . . . . . . . . . . . . 153
5.4.4 Effect of the Frame Length of the Code . . . . . . . . . . . . . . . . . . 155
5.4.5 The Component Codes . . . . . . . . . . . . . . . . . . . . . . . . . . . 156
5.4.6 Effect of the Interleaver . . . . . . . . . . . . . . . . . . . . . . . . . . . 158
5.4.7 Effect of Estimating the Channel Reliability Value $L_c$ . . . . . . . . . . 162
5.5 Turbo Coding Performance over Rayleigh Channels . . . . . . . . . . . . . . 167
5.5.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 167
5.5.2 Performance over Perfectly Interleaved Narrowband Rayleigh Channels . 168
5.5.3 Performance over Correlated Narrowband Rayleigh Channels . . . . . . 170
5.6 Summary and Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171
6 The Super-Trellis Structure of Convolutional Turbo Codes 173
6.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 173
6.2 Non-iterative Turbo Decoding Example . . . . . . . . . . . . . . . . . . . . . 174
6.2.1 Decoding Convolutional Codes . . . . . . . . . . . . . . . . . . . . . . 174
6.2.2 Decoding Turbo Codes . . . . . . . . . . . . . . . . . . . . . . . . . . . 177
6.2.2.1 Non-iterative Turbo Decoding Performance . . . . . . . . . . . 182
6.3 System Model and Terminology . . . . . . . . . . . . . . . . . . . . . . . . . 182
6.4 Introducing the Turbo Code Super-trellis . . . . . . . . . . . . . . . . . . . . 186
6.4.1 Turbo Encoder Super-states . . . . . . . . . . . . . . . . . . . . . . . . 186
6.4.2 Turbo Encoder Super-trellis . . . . . . . . . . . . . . . . . . . . . . . . 189
6.4.3 Generalised Definition of the Turbo Encoder Super-states . . . . . . . 189
6.4.4 Example of a Super-trellis . . . . . . . . . . . . . . . . . . . . . . . . . 192
6.5 Complexity of the Turbo Code Super-trellis . . . . . . . . . . . . . . . . . . . 197
6.5.1 Rectangular Interleavers . . . . . . . . . . . . . . . . . . . . . . . . . . 197
6.5.2 Uniform Interleaver . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 198
6.6 Optimum Decoding of Turbo Codes . . . . . . . . . . . . . . . . . . . . . . . 199
6.6.1 Comparison to Iterative Decoding . . . . . . . . . . . . . . . . . . . . . 200
6.6.2 Comparison to Conventional Convolutional Codes . . . . . . . . . . . . 204
6.7 Optimum Decoding of Turbo Codes: Discussion of the Results . . . . . . . . . 205
6.8 Summary and Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 209
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>RRNS Encoder</td>
<td>282</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Non-systematic RRNS Code</td>
<td>282</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Systematic RRNS Code</td>
<td>284</td>
</tr>
<tr>
<td>8.5.2.1</td>
<td>Modified Systematic RRNS Code</td>
<td>286</td>
</tr>
<tr>
<td>8.6</td>
<td>RRNS Decoder</td>
<td>286</td>
</tr>
<tr>
<td>8.7</td>
<td>Soft-input and Soft-output RRNS Decoder</td>
<td>288</td>
</tr>
<tr>
<td>8.7.1</td>
<td>Soft-input RRNS Decoder</td>
<td>288</td>
</tr>
<tr>
<td>8.7.2</td>
<td>Soft-output RRNS Decoder</td>
<td>290</td>
</tr>
<tr>
<td>8.7.3</td>
<td>Algorithm Implementation</td>
<td>293</td>
</tr>
<tr>
<td>8.8</td>
<td>Complexity</td>
<td>294</td>
</tr>
<tr>
<td>8.9</td>
<td>Simulation Results</td>
<td>298</td>
</tr>
<tr>
<td>8.9.1</td>
<td>Hard-decision Decoding</td>
<td>299</td>
</tr>
<tr>
<td>8.9.1.1</td>
<td>Encoder Types</td>
<td>299</td>
</tr>
<tr>
<td>8.9.1.2</td>
<td>Comparison of Redundant Residue Number System Codes and Reed-Solomon Codes</td>
<td>301</td>
</tr>
<tr>
<td>8.9.1.3</td>
<td>Comparison Between Different Error Correction Capabilities</td>
<td>302</td>
</tr>
<tr>
<td>8.9.2</td>
<td>Soft-decision Decoding</td>
<td>304</td>
</tr>
<tr>
<td>8.9.2.1</td>
<td>Effect of the Number of Test Positions</td>
<td>304</td>
</tr>
<tr>
<td>8.9.2.2</td>
<td>Soft-decision RRNS(10,8) Decoder</td>
<td>305</td>
</tr>
<tr>
<td>8.9.3</td>
<td>Turbo RRNS Decoding</td>
<td>305</td>
</tr>
<tr>
<td>8.9.3.1</td>
<td>Algorithm Comparison</td>
<td>305</td>
</tr>
<tr>
<td>8.9.3.2</td>
<td>Number of Iterations Used</td>
<td>308</td>
</tr>
<tr>
<td>8.9.3.3</td>
<td>Imperfect Estimation of the Channel Reliability Value $L_c$</td>
<td>309</td>
</tr>
<tr>
<td>8.9.3.4</td>
<td>The Effect of the Turbo Interleaver</td>
<td>310</td>
</tr>
<tr>
<td>8.9.3.5</td>
<td>The Effect of the Number of Bits Per Symbol</td>
<td>312</td>
</tr>
<tr>
<td>8.9.3.6</td>
<td>Coding Gain versus Estimated Complexity</td>
<td>313</td>
</tr>
<tr>
<td>8.10</td>
<td>Summary and Conclusions</td>
<td>314</td>
</tr>
</tbody>
</table>

| III | Coded Modulation: TCM, TTCM, BICM, BICM-ID | 317 |

| 9 | Coded Modulation Theory and Performance | 319 |
| 9.1 | Introduction | 319 |
| 9.2 | Trellis-Coded Modulation | 320 |
| 9.2.1 | TCM Principle | 321 |
| 9.2.2 | Optimum TCM Codes | 325 |
| 9.2.3 | TCM Code Design for Fading Channels | 327 |
| 9.2.4 | Set Partitioning | 329 |
| 9.3 | The Symbol-based MAP Algorithm | 330 |
| 9.3.1 | Problem Description | 330 |
| 9.3.2 | Detailed Description of the Symbol-based MAP Algorithm | 332 |
| 9.3.3 | Symbol-based MAP Algorithm Summary | 335 |
| 9.4 | Turbo Trellis-coded Modulation | 337 |
| 9.4.1 | TTCM Encoder | 337 |
| 9.4.2 | TTCM Decoder | 339 |
| 9.5 | Bit-interleaved Coded Modulation | 343 |
| 9.5.1 | BICM Principle | 343 |
| 9.5.2 | BICM Coding Example | 346 |
| 9.6 | Bit-Interleaved Coded Modulation Using Iterative Decoding | 347 |
CONTENTS

9.6.1 Labelling Method ......................................................... 347
9.6.2 Interleaver Design ....................................................... 351
9.6.3 BICM-ID Coding Example .............................................. 351
9.7 Coded Modulation Performance ........................................... 354
  9.7.1 Introduction ............................................................. 354
  9.7.2 Coded Modulation in Narrowband Channels ...................... 354
    9.7.2.1 System Overview .................................................. 354
    9.7.2.2 Simulation Results and Discussions .......................... 357
      9.7.2.2.1 Coded Modulation Performance over AWGN Channels 357
      9.7.2.2.2 Performance over Uncorrelated Narrowband Rayleigh Fading Channels ....................... 359
    9.7.2.2.3 Coding Gain versus Complexity and Interleaver Block Length ............................... 361
    9.7.2.3 Conclusion .......................................................... 366
  9.7.3 Coded Modulation in Wideband Channels .......................... 367
    9.7.3.1 Inter-symbol Interference ...................................... 367
    9.7.3.2 Decision Feedback Equaliser ................................... 368
      9.7.3.2.1 Decision Feedback Equaliser Principle .................. 368
      9.7.3.2.2 Equalizer Signal To Noise Ratio Loss .................... 370
    9.7.3.3 Decision Feedback Equalizer Aided Adaptive Coded Modulation 371
      9.7.3.3.1 Introduction .................................................. 371
      9.7.3.3.2 System Overview .............................................. 372
      9.7.3.3.3 Fixed-mode Performance .................................... 375
      9.7.3.3.4 System I and System II Performance ...................... 376
      9.7.3.3.5 Overall Performance ....................................... 380
      9.7.3.3.6 Conclusions .................................................... 382
    9.7.3.4 Orthogonal Frequency Division Multiplexing .................. 383
      9.7.3.4.1 Orthogonal Frequency Division Multiplexing Principle 383
    9.7.3.5 Orthogonal Frequency Division Multiplexing Aided Coded Modulation ......................... 386
      9.7.3.5.1 Introduction .................................................. 386
      9.7.3.5.2 System Overview .............................................. 386
      9.7.3.5.3 Simulation Parameters ...................................... 387
      9.7.3.5.4 Simulation Results and Discussions ...................... 388
      9.7.3.5.5 Conclusions .................................................... 389
  9.8 Summary and Conclusions .............................................. 390

IV Space-Time Block and Space-Time Trellis Coding ................................................. 393

10 Space-time Block Codes .................................................. 395
  10.1 Introduction ........................................................... 395
  10.2 Background .................................................................. 396
    10.2.1 Maximum Ratio Combining ...................................... 396
  10.3 Space-time Block Codes .................................................. 398
    10.3.1 A Twin-transmitter-based Space-time Block Code .................. 398
      10.3.1.1 The Space-time Code G₂ Using One Receiver ............... 399
      10.3.1.2 The Space-time Code G₂ Using Two Receivers .............. 401
    10.3.2 Other Space-time Block Codes ..................................... 403
CONTENTS xiii

10.3.3 MAP Decoding of Space-time Block Codes ......................... 404
10.4 Channel-coded Space-time Block Codes ............................. 406
  10.4.1 System Overview .................................................. 407
  10.4.2 Channel Codec Parameters ........................................ 408
  10.4.3 Complexity Issues and Memory Requirements ..................... 411
10.5 Performance Results .................................................... 414
  10.5.1 Performance Comparison of Various Space-time Block Codes Without
       Channel Codes .......................................................... 415
     10.5.1.1 Maximum Ratio Combining and the Space-time Code $G_2$ ....... 415
     10.5.1.2 Performance of 1 BPS Schemes .................................. 415
     10.5.1.3 Performance of 2 BPS Schemes .................................. 417
     10.5.1.4 Performance of 3 BPS Schemes .................................. 418
     10.5.1.5 Channel-coded Space-time Block Codes .......................... 420
  10.5.2 Mapping Binary Channel Codes to Multilevel Modulation .......... 421
     10.5.2.1 Turbo Convolutional Codes: Data and Parity Bit Mapping ....... 422
     10.5.2.2 Turbo Convolutional Codes: Interleaver Effects ................ 425
     10.5.2.3 Turbo BCH Codes .............................................. 428
     10.5.2.4 Convolutional Codes ........................................... 430
  10.5.3 Performance Comparison of Various Channel Codecs Using the $G_2$
       Space-time Code and Multilevel Modulation .......................... 430
     10.5.3.1 Comparison of Turbo Convolutional Codes ...................... 431
     10.5.3.2 Comparison of Different-rate TC(2,1,4) Codes ................. 432
     10.5.3.3 Convolutional Codes ........................................... 433
     10.5.3.4 $G_2$-coded Channel Codec Comparison: Throughput of 2 BPS ... 434
     10.5.3.5 $G_2$-coded Channel Codec Comparison: Throughput of 3 BPS ... 435
     10.5.3.6 Comparison of $G_2$-coded High-rate TC and TBCH Codes .......... 436
     10.5.3.7 Comparison of High-rate TC and Convolutional Codes .......... 437
  10.5.4 Coding Gain versus Complexity .................................... 438
     10.5.4.1 Complexity Comparison of Turbo Convolutional Codes ........... 438
     10.5.4.2 Complexity Comparison of Channel Codes ....................... 440
10.6 Summary and Conclusions .............................................. 442

11 Space-Time Trellis Codes .............................................. 445
  11.1 Introduction ........................................................... 445
  11.2 Space-time Trellis Codes ............................................. 446
     11.2.1 The 4-state, 4PSK Space-time Trellis Encoder ..................... 446
     11.2.1.1 The 4-state, 4PSK Space-time Trellis Decoder ................. 448
     11.2.2 Other Space-time Trellis Codes .................................. 449
  11.3 Space-time-coded Transmission over Wideband Channels ............. 450
     11.3.1 System Overview .................................................. 454
     11.3.2 Space-time and Channel Codec Parameters ........................ 455
     11.3.3 Complexity Issues ................................................ 457
  11.4 Simulation Results .................................................... 457
     11.4.1 Space-time Coding Comparison: Throughput of 2 BPS ............... 459
     11.4.2 Space-time Coding Comparison: Throughput of 3 BPS ................ 463
     11.4.3 The Effect of Maximum Doppler Frequency .......................... 466
     11.4.4 The Effect of Delay Spreads ...................................... 467
     11.4.5 Delay Non-sensitive System ...................................... 471
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.4.6</td>
<td>The Wireless Asynchronous Transfer Mode System</td>
</tr>
<tr>
<td>11.4.6.1</td>
<td>Channel-coded Space-time Codes: Throughput of 1 BPS</td>
</tr>
<tr>
<td>11.4.6.2</td>
<td>Channel-coded Space-time Codes: Throughput of 2 BPS</td>
</tr>
<tr>
<td>11.5</td>
<td>Space-time-coded Adaptive Modulation for OFDM</td>
</tr>
<tr>
<td>11.5.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>11.5.2</td>
<td>Turbo-coded and Space-time-coded AOOFDM</td>
</tr>
<tr>
<td>11.5.3</td>
<td>Simulation Results</td>
</tr>
<tr>
<td>11.5.3.1</td>
<td>Space-time-coded AOOFDM</td>
</tr>
<tr>
<td>11.5.3.2</td>
<td>Turbo- and Space-time-coded AOOFDM</td>
</tr>
<tr>
<td>11.6</td>
<td>Summary and Conclusions</td>
</tr>
<tr>
<td>12</td>
<td>Turbo-coded Adaptive QAM versus Space-time Trellis Coding</td>
</tr>
<tr>
<td>12.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>12.2</td>
<td>System Overview</td>
</tr>
<tr>
<td>12.2.1</td>
<td>SISO Equaliser and AQAM</td>
</tr>
<tr>
<td>12.2.2</td>
<td>MIMO Equaliser</td>
</tr>
<tr>
<td>12.3</td>
<td>Simulation Parameters</td>
</tr>
<tr>
<td>12.4</td>
<td>Simulation Results</td>
</tr>
<tr>
<td>12.4.1</td>
<td>Turbo-coded Fixed Modulation Mode Performance</td>
</tr>
<tr>
<td>12.4.2</td>
<td>Space-time Trellis Code Performance</td>
</tr>
<tr>
<td>12.4.3</td>
<td>Adaptive Quadrature Amplitude Modulation Performance</td>
</tr>
<tr>
<td>12.5</td>
<td>Summary and Conclusions</td>
</tr>
<tr>
<td>13</td>
<td>Turbo Equalisation</td>
</tr>
<tr>
<td>13.1</td>
<td>Motivation</td>
</tr>
<tr>
<td>13.2</td>
<td>The Mobile Radio Channel</td>
</tr>
<tr>
<td>13.3</td>
<td>Continuous Phase Modulation Theory</td>
</tr>
<tr>
<td>13.4</td>
<td>Digital Frequency Modulation Systems</td>
</tr>
<tr>
<td>13.5</td>
<td>State Representation</td>
</tr>
<tr>
<td>13.5.1</td>
<td>Minimum Shift Keying</td>
</tr>
<tr>
<td>13.5.2</td>
<td>Gaussian Minimum Shift Keying</td>
</tr>
<tr>
<td>13.6</td>
<td>Spectral Performance</td>
</tr>
<tr>
<td>13.6.1</td>
<td>Power Spectral Density</td>
</tr>
<tr>
<td>13.6.2</td>
<td>Fractional Out-of-Band Power</td>
</tr>
<tr>
<td>13.7</td>
<td>Construction of Trellis-based Equaliser States</td>
</tr>
<tr>
<td>13.8</td>
<td>Soft-output GMSK Equaliser and Turbo Coding</td>
</tr>
<tr>
<td>13.8.1</td>
<td>Background and Motivation</td>
</tr>
<tr>
<td>13.8.2</td>
<td>Soft-output GMSK Equaliser</td>
</tr>
<tr>
<td>13.8.3</td>
<td>The Log-MAP Algorithm</td>
</tr>
<tr>
<td>13.8.4</td>
<td>Summary of the Log-MAP Algorithm</td>
</tr>
<tr>
<td>13.8.5</td>
<td>Complexity of Turbo Decoding and Convolutional Decoding</td>
</tr>
<tr>
<td>13.8.6</td>
<td>System Parameters</td>
</tr>
<tr>
<td>13.8.7</td>
<td>Turbo Coding Performance Results</td>
</tr>
<tr>
<td>13.9</td>
<td>Summary and Conclusions</td>
</tr>
</tbody>
</table>
14 Turbo Equalisation for Partial-response Systems 553
14.1 Motivation ........................................... 556
14.2 Principle of Turbo Equalisation Using Single/Multiple Decoder(s) .... 557
14.3 Soft-in/Soft-out Equaliser for Turbo Equalisation ......................... 561
14.4 Soft-in/Soft-out Decoder for Turbo Equalisation .......................... 561
14.5 Turbo Equalisation Example .................................. 565
14.6 Summary of Turbo Equalisation .................................. 581
14.7 Performance of Coded GMSK Systems using Turbo Equalisation .... 583
  14.7.1 Convolutional-coded GMSK System .......................... 583
  14.7.2 Convolutional-coding-based Turbo-coded GMSK System ........ 585
  14.7.3 BCH-coding-based Turbo-coded GMSK System .................. 587
14.8 Discussion of Results ........................................ 589
14.9 Summary and Conclusions ...................................... 593

15 Turbo Equalisation Performance Bound 595
15.1 Motivation ........................................... 596
15.2 Parallel Concatenated Code Analysis .................................. 596
15.3 Serial Concatenated Code Analysis ................................... 603
15.4 Enumerating the Weight Distribution of the Convolutional Code .... 607
15.5 Recursive Properties of the MSK, GMSK and DPSK Modulators ....... 612
15.6 Analytical Model of Coded DPSK Systems ................................ 615
15.7 Theoretical and Simulation Performance of Coded DPSK Systems .... 617
15.8 Summary and Conclusions ...................................... 619

16 Comparative Study of Turbo Equalisers 623
16.1 Motivation ........................................... 623
16.2 System overview .......................................... 624
16.3 Simulation Parameters ........................................ 624
16.4 Results and Discussion ...................................... 627
  16.4.1 Five-path Gaussian Channel .................................. 627
  16.4.2 Equally Weighted Five-path Rayleigh Fading Channel .......... 631
16.5 Non-iterative Joint Channel Equalisation and Decoding ............... 639
  16.5.1 Motivation ........................................... 639
  16.5.2 Non-iterative Turbo Equalisation ................................ 639
  16.5.3 Non-iterative Joint Equalisation/Decoding...................... 641
    Using a $2 \times N$ Interleaver ................................... 641
  16.5.4 Non-iterative Turbo Equaliser Performance ..................... 642
    16.5.4.1 Effect of Interleaver Depth ................................ 643
    16.5.4.2 The M-algorithm .......................................... 647
16.6 Summary and Conclusions ...................................... 649

17 Reduced-complexity Turbo Equaliser 651
17.1 Motivation ........................................... 651
17.2 Complexity of the Multilevel Full-response Turbo Equaliser .......... 652
17.3 System Model ............................................. 654
17.4 In-phase/Quadrature-phase Equaliser Principle ........................... 656
17.5 Overview of the Reduced-complexity Turbo Equaliser ................... 658
  17.5.1 Conversion of the DFE Symbol Estimates to LLRs ................ 658
  17.5.2 Conversion of the Decoder A-Posteriori LLRs into Symbols .... 660
17.5.3 Decoupling Operation ........................................... 664
17.6 Complexity of the In-phase/Quadrature-phase Turbo Equaliser .... 664
17.7 System Parameters .................................................... 666
17.8 System Performance ................................................ 667
  17.8.1 4-QAM System ................................................... 667
  17.8.2 16-QAM System .................................................. 670
  17.8.3 64-QAM System .................................................. 673
17.9 Summary and Conclusions ......................................... 674

18 Turbo Equalisation for Space-time Trellis-coded Systems 677
  18.1 Introduction ....................................................... 677
  18.2 System Overview .................................................. 678
  18.3 Principle of In-phase/Quadrature-phase Turbo Equalisation ........ 679
  18.4 Complexity Analysis .............................................. 682
  18.5 Results and Discussion ......................................... 683
    18.5.1 Performance versus Complexity Trade-off .................. 688
    18.5.2 Performance of STTC Systems over Channels with Long Delays 695
  18.6 Summary and Conclusions ....................................... 696

19 Summary and Conclusions 699
  19.1 Summary of the Book ............................................. 699
  19.2 Concluding Remarks ............................................. 709

Bibliography 713

Subject Index 729

Author Index 729

About the Authors 729

Other Related Wiley and IEEE Press Books 731
Acknowledgments

We are indebted to our many colleagues who have enhanced our understanding of the subject, in particular to Prof. Emeritus Raymond Steele. These colleagues and valued friends, too numerous to be mentioned, have influenced our views concerning various aspects of wireless multimedia communications. We thank them for the enlightenment gained from our collaborations on various projects, papers and books. We are grateful to Steve Braithwaite, Jan Brecht, Jon Blogh, Marco Breiling, Marco del Buono, Sheng Chen, Peter Cherriman, Stanley Chia, Byoung Jo Choi, Joseph Cheung, Sheyam Lal Dhomeja, Dirk Didascalou, Lim Dongmin, Stephan Ernst, Peter Fortune, Eddie Green, David Greenwood, Hee Thong How, Thomas Keller, Ee Lin Kuan, W. H. Lam, C. C. Lee, Xiao Lin, Chee Siong Lee, Tong-Hooi Liew, Matthias Münster, Vincent Roger-Marchart, Jason Ng, Michael Ng, M. A. Nofal, Jeff Reeve, Redwan Salami, Clare Somerville, Rob Stedman, David Stewart, Jürgen Streit, Jeff Torrance, Spyros Vlahoyiannatos, William Webb, Stephan Weiss, John Williams, Jason Woodard, Choong Hin Wong, Henry Wong, James Wong, Lie-Liang Yang, Bee-Leong Yeap, Mong-Suan Yee, Kai Yen, Andy Yuen, and many others with whom we enjoyed an association.

We also acknowledge our valuable associations with the Virtual Centre of Excellence (VCE) in Mobile Communications, in particular with its chief executive, Dr Walter Tuttlebee, and other leading members of the VCE, namely Dr Keith Baughan, Prof. Hamid Aghvami, Prof. Ed Candy, Prof. John Dunlop, Prof. Barry Evans, Prof. Peter Grant, Dr Mike Barnard, Prof. Joseph McGeehan, Dr Steve McLaughlin and many other valued colleagues. Our sincere thanks are also due to the EPSRC, UK for supporting our research. We would also like to thank Dr Joao Da Silva, Dr Jorge Pereira, Dr Bartholome Arroyo, Dr Bernard Barani, Dr Demosthenes Ikonomou, Dr Fabrizio Sestini and other valued colleagues from the Commission of the European Communities, Brussels, Belgium, as well as Andy Aftelak, Mike Philips, Andy Wilton, Luis Lopes and Paul Crichton from Motorola ECID, Swindon, UK, for sponsoring some of our recent research. Further thanks are due to Tim Wilkinson and Ian Johnson at HP in Bristol, UK for funding some of our research efforts.

We feel particularly indebted to Rita Hanzo as well as Denise Harvey for their skilful assistance in typesetting the manuscript in LaTeX. Without the kind support of Mark Hammond, Sarah Hinton, Zöe Pinnock and their colleagues at the Wiley editorial office in Chichester, UK this monograph would never have materialised. Finally, our sincere gratitude is due to the numerous authors listed in the Author Index — as well as to those whose work was not cited due to space limitations — for their contributions to the state of the art, without whom this book would not have materialised.

Lajos Hanzo, Tong Hooi Liew and Bee Leong Yeap
Department of Electronics and Computer Science
University of Southampton
Chapter 1

Historical Perspective, Motivation and Outline

1.1 A Historical Perspective on Channel Coding

The history of channel coding or Forward Error Correction (FEC) coding dates back to Shannon’s pioneering work [1] in 1948, predicting that arbitrarily reliable communications are achievable with the aid of channel coding, upon adding redundant information to the transmitted messages. However, Shannon refrained from proposing explicit channel coding schemes for practical implementations. Furthermore, although the amount of redundancy added increases as the associated information delay increases, he did not specify the maximum delay that may have to be tolerated, in order to be able to communicate near the Shannonian limit. In recent years researchers have been endeavouring to reduce the amount of latency inflicted for example by a turbo codec’s interleaver that has to be tolerated for the sake of attaining a given target performance.

Historically, one of the first practical FEC codes was the single error correcting Hamming code [2], which was a block code proposed in 1950. Convolutional FEC codes date back to 1955 [3], which were discovered by Elias, while Wozencraft and Reiffen [4,5], as well as Fano [6] and Massey [7], proposed various algorithms for their decoding. A major milestone in the history of convolutional error correction coding was the invention of a maximum likelihood sequence estimation algorithm by Viterbi [8] in 1967. A classic interpretation of the Viterbi Algorithm (VA) can be found, for example, in Forney’s often-quoted paper [9]. One of the first practical applications of convolutional codes was proposed by Heller and Jacobs [10] during the 1970s.

We note here that the VA does not result in minimum Bit Error Rate (BER), rather it finds the most likely sequence of transmitted bits. However, it performs close to the minimum possible BER, which can be achieved only with the aid of an extremely complex full-search algorithm evaluating the probability of all possible $2^n$ binary strings of a $k$-bit message. The minimum BER decoding algorithm was proposed in 1974 by Bahl et al. [11], which was termed the Maximum A-Posteriori (MAP) algorithm. Although the MAP algorithm slightly outperforms the VA in BER terms, because of its significantly higher complexity it was rarely used in practice, until turbo codes were contrived by Berrou et al. in 1993 [12, 13].

Focusing our attention on block codes, the single error correcting Hamming block code was too weak for practical applications. An important practical milestone was the discovery
of the family of multiple error correcting Bose–Chaudhuri–Hocquenghem (BCH) binary block codes [14] in 1959 and in 1960 [15, 16]. In 1960, Peterson [17] recognised that these codes exhibit a cyclic structure, implying that all cyclically shifted versions of a legitimate codeword are also legitimate codewords. The first method for constructing trellises for linear block codes was proposed by Wolf [18] in 1978. Owing to the associated high complexity, there was only limited research in trellis decoding of linear block codes [19, 20]. It was in 1988, when Forney [21] showed that some block codes have relatively simple trellis structures. Motivated by Forney’s work, Honary, Markarian and Farrell et al. [19,22–25] as well as Lin and Kasami et al. [20,26,27] proposed various methods for reducing the associated complexity. The Chase algorithm [28] is one of the most popular techniques proposed for near maximum likelihood decoding of block codes.

Furthermore, in 1961 Gorenstein and Zierler [29] extended the binary coding theory to treat non-binary codes as well, where code symbols were constituted by a number of bits, and this led to the birth of burst-error correcting codes. They also contrived a combination of algorithms, which is referred to as the Peterson–Gorenstein–Zierler (PGZ) algorithm. In 1960 a prominent non-binary subset of BCH codes was discovered by Reed and Solomon [30]; they were named Reed–Solomon (RS) codes after their inventors. These codes exhibit certain optimality properties, since their codewords have the highest possible minimum distance between the legitimate codewords for a given code rate. This, however, does not necessarily guarantee attaining the lowest possible BER. The PGZ decoder can also be invoked for decoding non-binary RS codes. A range of powerful decoding algorithms for RS codes was found by Berlekamp [31, 32] and Massey [33,34]. Various soft-decision decoding algorithms were proposed for the soft decoding of RS codes by Sweeney [35–37] and Honary [19]. In recent years RS codes have found practical applications, for example, in Compact Disc (CD) players, in deep-space scenarios [38], and in the family of Digital Video Broadcasting (DVB) schemes [39], which were standardised by the European Telecommunications Standardisation Institute (ETSI).

Inspired by the ancient theory of Residue Number Systems (RNS) [40–42], which constitute a promising number system for supporting fast arithmetic operations [40,41], a novel class of non-binary codes referred to as Redundant Residue Number System (RRNS) codes were introduced in 1967. An RRNS code is a maximum–minimum distance block code, exhibiting similar distance properties to RS codes. Watson and Hastings [42] as well as Krishna et al. [43, 44] exploited the properties of the RRNS for detecting or correcting a single error and also for detecting multiple errors. Recently, the soft decoding of RRNS codes was proposed in [45].

During the early 1970s, FEC codes were incorporated in various deep-space and satellite communications systems, and in the 1980s they also became common in virtually all cellular mobile radio systems. However, for a long time FEC codes and modulation have been treated as distinct subjects in communication systems. By integrating FEC and modulation, in 1987 Ungerboeck [46–48] proposed Trellis Coded Modulation (TCM), which is capable of achieving significant coding gains over power and band-limited transmission media. A further historic breakthrough was the invention of turbo codes by Berrou, Glavieux, and Thitimajshima [12, 13] in 1993, which facilitate the operation of communications systems near the Shannonian limits. Turbo coding is based on a composite codec constituted by two parallel concatenated codecs. Since its recent invention turbo coding has evolved at an unprecedented rate and has reached a state of maturity within just a few years due to the intensive research efforts of the turbo coding community. As a result of this dramatic evolution, turbo coding has also found its way into standardised systems, such as for example the recently ratified third-generation (3G) mobile radio systems [49]. Even more impressive performance gains can be attained with the aid of turbo coding in the context of video broadcast systems, where the associated system delay is less
critical than in delay-sensitive interactive systems.

More specifically, in their proposed scheme Berrou et al. [12, 13] used a parallel concatenation of two Recursive Systematic Convolutional (RSC) codes, accommodating the turbo interleaver between the two encoders. At the decoder an iterative structure using a modified version of the classic minimum BER MAP invented by Bahl et al. [11] was invoked by Berrou et al., in order to decode these parallel concatenated codes. Again, since 1993 a large amount of work has been carried out in the area, aiming for example to reduce the associated decoder complexity. Practical reduced-complexity decoders are for example the Max-Log-MAP algorithm proposed by Koch and Baier [50], as well as by Erfanian et al. [51], the Log-MAP algorithm suggested by Robertson, Villebrun and Hoeher [52], and the SOVA advocated by Hagenauer as well as Hoeher [53,54]. Le Goff, Glavieux and Berrou [55], Wachsmann and Huber [56] as well as Robertson and Woz [57] suggested the use of these codes in conjunction with bandwidth-efficient modulation schemes. Further advances in understanding the excellent performance of the codes are due, for example, to Benedetto and Montorsi [58, 59] and Perez, Seghers and Costello [60]. During the mid-1990s Hagenauer, Offer and Papke [61], as well as Pyndiah [62], extended the turbo concept to parallel concatenated block codes as well. Nickl et al. show in [63] that Shannon’s limit can be approached within 0.27 dB by employing a simple turbo Hamming code. In [64] Acikel and Ryan proposed an efficient procedure for designing the puncturing patterns for high-rate turbo convolutional codes. Jung and Nasshan [65,66] characterised the achievable turbo-coded performance under the constraints of short transmission frame lengths, which is characteristic of interactive speech systems. In collaboration with Blanz they also applied turbo codes to a CDMA system using joint detection and antenna diversity [67]. Barbulescu and Pietrobon addressed the issues of interleaver design [68]. The tutorial paper by Sklar [69] is also highly recommended as background reading.

Driven by the urge to support high data rates for a wide range of bearer services, Tarokh, Seshadri and Calderbank [70] proposed space-time trellis codes in 1998. By jointly designing the FEC, modulation, transmit diversity and optional receive diversity scheme, they increased the throughput of band-limited wireless channels. A few months later, Alamouti [71] invented a low-complexity space-time block code, which offers significantly lower complexity at the cost of a slight performance degradation. Alamouti’s invention motivated Tarokh et al. [72,73] to generalise Alamouti’s scheme to an arbitrary number of transmitter antennas. Then, Tarokh et al., Bauch et al. [74,75], Agrawal et al. [76], Li et al. [77,78] and Naguib et al. [79] extended the research of space-time codes from considering narrowband channels to dispersive channels [70,71,73,79,80].

In Figure 1.1, we show the evolution of channel coding research over the past 50 years since Shannon’s legendary contribution [1]. These milestones have been incorporated also in the range of monographs and textbooks summarised in Figure 1.2. At the time of writing, the Shannon limit has been approached within 0.27 dB [63] over Gaussian channels. Also at the time of writing the challenge is to contrive FEC schemes which are capable of achieving a performance near the capacity of wireless channels.

1.2 Motivation of the Book

The design of an attractive channel coding and modulation scheme depends on a range of contradictory factors, which are portrayed in Figure 1.3. The message of this illustration is multi-fold. For example, given a certain transmission channel, it is always feasible to design a coding and modulation (‘codulation’) system, which can further reduce the BER achieved. This typically implies, however, further investments and/or penalties in terms of the required increased imple-
Figure 1.1: A brief history of channel coding.
mentational complexity and coding/interleaving delay as well as reduced effective throughput. Different solutions accrue when optimising different codec features. For example, in many applications the most important codec parameter is the achievable coding gain, which quantifies the amount of bit-energy reduction attained by a codec at a certain target BER. Naturally, transmitted power reduction is extremely important in battery-powered devices. This transmitted power reduction is only achievable at the cost of an increased implementational complexity, which itself typically increases the power consumption and hence erodes some of the power gain.

Viewing this system optimisation problem from a different perspective, it is feasible to transmit at a higher bit rate in a given fixed bandwidth by increasing the number of bits per modulated symbol. However, when aiming for a given target BER, the channel coding rate has to be reduced, in order to increase the transmission integrity. Naturally, this reduces the effective throughput of the system and results in an overall increased system complexity. When the channel’s characteristic and the associated bit error statistics change, different solutions may become more attractive. This is because Gaussian channels, narrowband and wideband Rayleigh fading or various Nakagami fading channels inflict different impairments. These design trade-offs constitute the subject of this monograph.

Our intention with the book is multi-fold:

1) First, we would like to pay tribute to all researchers, colleagues and valued friends who contributed to the field. Hence this book is dedicated to them, since without their quest for better coding solutions to communications problems this monograph could not have been conceived. They are too numerous to name here, hence they appear in the author index of the book.

2) The invention of turbo coding not only assisted in attaining a performance approaching the Shannonian limits of channel coding for transmissions over Gaussian channels, but also revitalised channel coding research. In other words, turbo coding opened a new chapter in the design of iterative detection-assisted communications systems, such as turbo trellis coding schemes, turbo channel equalisers, etc. Similarly dramatic advances have been attained with the advent of space-time coding, when communicating over dispersive, fading wireless channels. Recent trends indicate that better overall system performance may be attained by jointly optimising a number of system components, such as channel coding, channel equalisation, transmit and received diversity and the modulation scheme, than in case of individually optimising the system components. This is the main objective of this monograph.

3) Since at the time of writing no joint treatment of the subjects covered by this book exists, it is timely to compile the most recent advances in the field. Hence it is our hope that the conception of this monograph on the topic will present an adequate portrayal of the last decade of research and spur this innovation process by stimulating further research in the coding and communications community.

1.3 Organisation of the Book

Below, we present the outline and rationale of the book:
Shannon limit [1] (1948)

1950

1960
  Reed & Solomon, Polynomial codes over certain finite fields [30]
  Peterson, Error correcting codes [84]
  Wozencraft & Reiflen, Sequential decoding [5]
  Shannon, Mathematical theory of communication [91]
  Massey, Threshold decoding [7]

  Szabo & Tanaka, Residue arithmetic & its appl. to computer technology [41]
  Berlekamp, Algebraic coding theory [32]
  Kasami, Combinational mathematics and its applications [83]

1970
  Peterson & Weldon, Error correcting codes [82]
  Blake, Algebraic coding theory: history and development [87]

  Macwilliams & Sloane, The theory of error correcting codes [85]

1980
  Clark & Cain, Error correction coding for digital communications [88]
  Pless, Introduction to the theory of error-correcting codes [89]
  Blahut, Theory and practice of error control codes [90]
  Lidl & Niederreiter, Finite fields [95]
  Lin & Costello, Error control coding: fundamentals and applications [96]
  Michelson & Levesque, Error control techniques for digital communication [97]

  Sklar, Digital communications fundamentals and applications [86]
  Sweeney, Error Control Coding: An Introduction [103]

1990
  Hoffman et al., Coding theory [98]
  Huber, Trelliscodierung [99]
  Anderson & Mohan, Source and channel coding - an algorithmic approach [100]
  Wicker, Error control systems for digital communication and storage [101]
  Proakis, Digital communications [102]
  Honary & Markarian, Trellis decoding of block codes [19]
  S. Lin et al., Trellises & trellis-based decoding alg. for linear block codes [20]
  Schlegel, Trellis coding [48]
  Heegard & Wicker, Turbo coding [92]

2000
  Bossert, Channel coding for telecommunications [93]
  Vucetic & Yuan, Turbo codes principles and applications [94]
  Hanzo, Liew & Yeap, Turbo coding, turbo equalisation & space-time coding, 2002

Figure 1.2: Milestones in channel coding.
1.3. ORGANISATION OF THE BOOK

- **Chapter 2:** For the sake of completeness and wider reader appeal virtually no prior knowledge is assumed in the field of channel coding. Hence in Chapter 2 we commence our discourse by introducing the family of convolutional codes and the hard- as well as soft-decision Viterbi algorithm in simple conceptual terms with the aid of worked examples.

- **Chapter 3:** This chapter provides a rudimentary introduction to the most prominent classes of block codes, namely to Reed–Solomon (RS) and Bose–Chaudhuri–Hocquenghem (BCH) codes. A range of algebraic decoding techniques are also reviewed and worked examples are included.

- **Chapter 4:** Based on the simple Viterbi decoding concepts introduced in Chapter 2, in this chapter an overview of the family of conventional binary BCH codes is given, with special emphasis on their trellis decoding. In parallel to our elaborations in Chapter 2 on the context of convolutional codes, the Viterbi decoding of binary BCH codes is detailed with the aid of worked examples. These discussions are followed by the simulation-based performance characterisation of various BCH codes employing both hard-decision and soft-decision decoding methods. The classic Chase algorithm is introduced and its performance is investigated.

- **Chapter 5:** This chapter introduces the concept of turbo convolutional codes and gives a detailed discourse on the Maximum A-Posteriori (MAP) algorithm and its computationally less demanding counterparts, namely the Log-MAP and Max-Log-MAP algorithms. The Soft-Output Viterbi Algorithm (SOVA) is also highlighted and its concept is augmented with the aid of a detailed worked example. Then the effects of the various turbo codec parameters are investigated, namely that of the number of iterations, the puncturing patterns used, the component decoders, the influence of the interleaver depth, which is related to the codeword length, etc. The various codecs’ performance is studied also when communicating over Rayleigh fading channels.

- **Chapter 6:** While in Chapter 5 we invoked iterative turbo decoders, in this chapter a super-trellis is constructed from the two constituent convolutional codes’ trellises and the maximum likelihood codeword is output in a single, but implementationally complex decoding step, without iterations. The advantage of the associated super-trellis is that it allows us to explore the trellis describing the construction of turbo codes and also to relate turbo codes to high-constraint-length convolutional codes of the same decoding complexity.
• **Chapter 7:** The concept of turbo codes using BCH codes as component codes is introduced. A detailed derivation of the MAP algorithm is given, building on the concepts introduced in Chapter 5 in the context of convolutional turbo codes, but this time cast in the framework of turbo BCH codes. Then, the MAP algorithm is modified in order to highlight the concept of the Max-Log-MAP and Log-MAP algorithms, again, with reference to binary turbo BCH codes. Furthermore, the SOVA-based binary BCH decoding algorithm is introduced. Then a simple turbo decoding example is given, highlighting how iterative decoding assists in correcting multiple errors. We also describe a novel MAP algorithm for decoding extended BCH codes. Finally, we show the effects of the various coding parameters on the performance of turbo BCH codes.

• **Chapter 8:** The concept of Residue Number Systems (RNS) is introduced and extended to Redundant Residue Number Systems (RRNS), introducing the family of RRNS codes. Some coding-theoretic aspects of RRNS codes is investigated, demonstrating that RRNS codes exhibit similar distance properties to RS codes. A procedure for multiple-error correction is then given. Different bit-to-symbol mapping methods are highlighted, yielding non-systematic and systematic RRNS codes. A novel bit-to-symbol mapping method is introduced, which results in efficient systematic RRNS codes. The classic Chase algorithm is then modified in order to create a Soft-Input Soft-Output (SISO) RRNS decoder. This enables us to implement the iterative decoding of turbo RRNS codes. Finally, simulation results are given for various RRNS codes, employing hard-decision and soft-decision decoding methods. The performance of the RRNS codes is compared to that of RS codes and the performance of turbo RRNS codes is studied.

• **Chapter 9:** Our previous discussions on various channel coding schemes evolves to the family of joint coding and modulation-based arrangements, which are often referred to as coded modulation schemes. Specifically, Trellis-Coded Modulation (TCM), Turbo Trellis-Coded Modulation (TTCM), Bit-Interleaved Coded Modulation (BICM) as well as iterative joint decoding and demodulation-assisted BICM (BICM-ID) will be studied and compared under various narrowband and wideband propagation conditions.

• **Chapter 10:** Space-time block codes are introduced. The derivation of the MAP decoding of space-time block codes is then given. A system is proposed by concatenating space-time block codes and various channel codes. The complexity and memory requirements of various channel decoders are derived, enabling us to compare the performance of the proposed channel codes by considering their decoder complexity. Our simulation results related to space-time block codes using no channel coding are presented first. Then, we investigate the effect of mapping data and parity bits from binary channel codes to non-binary modulation schemes. Finally, we compare our simulation results for various channel codes concatenated with a simple space-time block code. Our performance comparisons are conducted by also considering the complexity of the associated channel decoder.

• **Chapter 11:** The encoding process of space-time trellis codes is highlighted. This is followed by employing an Orthogonal Frequency Division Multiplexing (OFDM) modem in conjunction with space-time codes over wideband channels. Turbo codes and RS codes are concatenated with space-time codes in order to improve their performance. Then, the performance of the advocated space-time block code and space-time trellis codes is compared. Their complexity is also considered in comparing both schemes. The effect of delay spread and maximum Doppler frequency on the performance of the space-time codes is investigated. A Signal to Interference Ratio (SIR) related term is defined in the context
of dispersive channels for the advocated space-time block code, and we will show how the SIR affects the performance of the system. In our last section, we propose space-time-coded Adaptive OFDM (AOOFDM). We then show by employing multiple antennas that with the advent of space-time coding, the wideband fading channels have been converted to AWGN-like channels.

- **Chapter 12:** The discussions of Chapters 10 and 11 were centred around the topic of employing multiple-transmitter, multiple-receiver (MIMO) based transmit and receive-diversity assisted space-time coding schemes. These arrangements have the potential of significantly mitigating the hostile fading wireless channel’s near-instantaneous channel quality fluctuations. Hence these space-time codecs can be advantageously combined with powerful channel codecs originally designed for Gaussian channels. As a lower-complexity design alternative, this chapter introduces the concept of near-instantaneously Adaptive Quadrature Amplitude Modulation (AQAM), combined with near-instantaneously adaptive turbo channel coding. These adaptive schemes are capable of mitigating the wireless channel’s quality fluctuations by near-instantaneously adapting both the modulation mode used as well as the coding rate of the channel codec invoked. The design and performance study of these novel schemes constitutes the topic of Chapter 12.

- **Chapter 13:** This chapter focuses on the portrayal of partial-response modulation schemes, which exhibit impressive performance gains in the context of joint iterative, joint channel equalisation and channel decoding. This joint iterative receiver principle is termed turbo equalisation. An overview of Soft-In/Soft-Out (SISO) algorithms, namely that of the MAP algorithm and Log-MAP algorithm, is presented in the context of GMSK channel equalisation, since these algorithms are used in the investigated joint channel equaliser and turbo decoder scheme.

- **Chapter 14:** Based on the introductory concepts of Chapter 13, in this chapter the detailed principles of iterative joint channel equalisation and channel decoding techniques known as turbo equalisation are introduced. This technique is invoked in order to overcome the unintentional Inter-Symbol Interference (ISI) and Controlled Inter-Symbol Interference (CISI) introduced by the channel and the modulator, respectively. Modifications of the SISO algorithms employed in the equaliser and decoder are also portrayed, in order to generate information related not only to the source bits but also to the parity bits of the codewords. The performance of coded systems employing turbo equalisation is analysed. Three classes of encoders are utilised, namely convolutional codes, convolutional-coding-based turbo codes and BCH-coding-based turbo codes.

- **Chapter 15:** Theoretical models are devised for the coded schemes in order to derive the maximum likelihood bound of the system. These models are based on the Serial Concatenated Convolutional Code (SCCC) analysis presented in reference [104]. Essentially, this analysis can be employed since the modulator could be represented accurately as a rate $R = 1$ convolutional encoder. Apart from convolutional-coded systems, turbo-coded schemes are also considered. Therefore the theoretical concept of Parallel Concatenated Convolutional Codes (PCCC) [59] is utilised in conjunction with the SCCC principles in order to determine the Maximum Likelihood (ML) bound of the turbo-coded systems, which are modelled as hybrid codes consisting of a parallel concatenated convolutional code, serially linked with another convolutional code. An abstract interleaver from reference [59] — termed the uniform interleaver — is also utilised, in order to reduce the
complexity associated with determining all the possible interleaver permutations.

- **Chapter 16**: A comparative study of coded BPSK systems, employing high-rate channel encoders, is presented. The objective of this study is to investigate the performance of turbo equalisers in systems employing different classes of codes for high code rates of $R = \frac{3}{4}$ and $R = \frac{2}{3}$, since known turbo equalisation results have only been presented for turbo equalisers using convolutional codes and convolutional-based turbo codes for code rates of $R = \frac{1}{2}$ and $R = \frac{1}{3}$ [105, 106]. Specifically, convolutional codes, convolutional-coding-based turbo codes, and Bose–Chaudhuri–Hocquengham (BCH)-coding-based [14, 15] turbo codes are employed in this study.

- **Chapter 17**: A novel reduced-complexity trellis-based equaliser is presented. In each turbo equalisation iteration the decoder generates information which reflects the reliability of the source and parity bits of the codeword. With successive iteration, the reliability of this information improves. This decoder information is exploited in order to decompose the received signal such that each quadrature component consists of the in-phase or quadrature-phase component signals. Therefore, the equaliser only has to consider the possible in-phase or quadrature-phase components, which is a smaller set of signals than all of their possible combinations.

- **Chapter 18**: For transmissions over wideband fading channels and fast fading channels, space-time trellis coding (STTC) is a more appropriate diversity technique than space-time block coding. STTC [70] relies on the joint design of channel coding, modulation, transmit diversity and the optional receiver diversity schemes. The decoding operation is performed by using a maximum likelihood detector. This is an effective scheme, since it combines the benefits of Forward Error Correction (FEC) coding and transmit diversity, in order to obtain performance gains. However, the cost of this is the additional computational complexity, which increases as a function of bandwidth efficiency (bits/s/Hz) and the required diversity order. In this chapter STTC is investigated for transmission over wideband fading channels.

- **Chapter 19**: This chapter provides a brief summary of the book.

It is our hope that this book portrays the range of contradictory system design trade-offs associated with the conception of channel coding arrangements in an unbiased fashion and that readers will be able to glean information from it in order to solve their own particular channel coding and communications problem. Most of all, however, we hope that they will find it an enjoyable and informative read, providing them with intellectual stimulation.

Lajos Hanzo, Tong Hooi Liew and Bee Leong Yeap

*Department of Electronics and Computer Science*
*University of Southampton*
Part I

Convolutional and Block Coding
Chapter 2

Convolutional Channel Coding

2.1 Brief Channel Coding History

In this chapter a rudimentary introduction to convolutional coding is offered to those readers who are not familiar with the associated concepts. Readers who are familiar with the basic concepts of convolutional coding may proceed to the chapter of their immediate interest.

The history of channel coding or Forward Error Correction (FEC) coding dates back to Shannon’s pioneering work in which he predicted that arbitrarily reliable communications are achievable by redundant FEC coding, although he refrained from proposing explicit schemes for practical implementations. Historically, one of the first practical codes was the single error correcting Hamming code [2], which was a block code proposed in 1950. Convolutional FEC codes date back to 1955 [3], which were discovered by Elias, whereas Wozencraft and Reiffen [4, 5], as well as Fano [6] and Massey [7], proposed various algorithms for their decoding. A major milestone in the history of convolutional error correction coding was the invention of a maximum likelihood sequence estimation algorithm by Viterbi [8] in 1967. A classic interpretation of the Viterbi Algorithm (VA) can be found, for example, in Forney’s often-quoted paper [9], and one of the first applications was proposed by Heller and Jacobs [10].

We note, however, that the VA does not result in minimum Bit Error Rate (BER). The minimum BER decoding algorithm was proposed in 1974 by Bahl et al. [11], which was termed the Maximum A-Posteriori (MAP) algorithm. Although the MAP algorithm slightly outperforms the VA in BER terms, because of its significantly higher complexity it was rarely used, until turbo codes were contrived [12].

During the early 1970s, FEC codes were incorporated in various deep-space and satellite communications systems, and in the 1980s they also became common in virtually all cellular mobile radio systems. A further historic breakthrough was the invention of the turbo codes by Berrou, Glavieux, and Thitimajshima [12] in 1993, which facilitates the operation of communications systems near the Shannonian limits.

Focusing our attention on block codes, the single error correcting Hamming block code was too weak, however, for practical applications. An important practical milestone was the discovery of the family of multiple error correcting Bose–Chaudhuri–Hocquenghem (BCH) binary block codes [14] in 1959 and in 1960 [15, 16]. In 1960, Peterson [17] recognised that these codes exhibit a cyclic structure, implying that all cyclically shifted versions of a legitimate codeword are also legitimate codewords. Furthermore, in 1961 Gorenstein and Zierler [29] extended the
binary coding theory to treat non-binary codes as well, where code symbols were constituted by a number of bits, and this led to the birth of burst-error correcting codes. They also contrived a combination of algorithms, which are referred to as the Peterson–Gorenstein–Zierler (PGZ) algorithm. We will elaborate on this algorithm later in this chapter. In 1960 a prominent non-binary subset of BCH codes were discovered by Reed and Solomon [30]; they were named Reed–Solomon (RS) codes after their inventors. These codes exhibit certain optimality properties, and they will also be treated in more depth in this chapter. We will show that the PGZ decoder can also be invoked for decoding non-binary RS codes.

A range of powerful decoding algorithms for RS codes was found by Berlekamp [31, 32] and Massey [33, 34], which also constitutes the subject of this chapter. In recent years, these codes have found practical applications, for example, in Compact Disc (CD) players, in deep-space scenarios [38], and in the family of Digital Video Broadcasting (DVB) schemes, which were standardised by the European Telecommunications Standardization Institute (ETSI). We now consider the conceptually less complex class of convolutional codes, which will be followed by our discussions on block coding.

### 2.2 Convolutional Encoding

Both block codes and Convolutional Codes (CCs) can be classified as systematic or non-systematic codes, where the terminology suggests that in systematic codes the original information bits or symbols constitute part of the encoded codeword and hence they can be recognised explicitly at the output of the encoder. Their encoders can typically be implemented by the help of linear shift-register circuitries, an example of which can be seen in Figure 2.1. The figure will be explored in more depth after introducing some of the basic convolutional coding parameters.

Specifically, in general a $k$-bit information symbol is entered into the encoder, constituted by $K$ shift-register stages. In our example of Figure 2.1, the corresponding two shift-register stages are $s_1$ and $s_2$. In general, the number of shift-register stages $K$ is referred to as the constraint length of the code. An alternative terminology is to refer to this code as a memory three code, implying that the memory of the CC is given by $K + 1$. The current shift-register state $s_1, s_2$ plus the incoming bit $b_i$ determine the next state of this state machine. The number of output bits is typically denoted by $n$, while the coding rate by $R = k/n$, implying that $R \leq 1$. In order to fully specify the code, we also have to stipulate the generator polynomial, which describes the topology of the modulo-2 gates generating the output bits of the convolutional encoder. For generating $n$ bits, $n$ generator polynomials are necessary. In general, a CC is denoted as a $CC(n, k, K)$ scheme, and given the $n$ generator polynomials, the code is fully specified.

Once a specific bit enters the encoder’s shift register in Figure 2.1, it has to traverse through the register, and hence the register’s sequence of state transitions is not arbitrary. Furthermore, the modulo-2 gates impose additional constraints concerning the output bit-stream. Because of these constraints, the legitimate transmitted sequences are restricted to certain bit patterns, and if there are transmission errors, the decoder will conclude that such an encoded sequence could not have been generated by the encoder and that it must be due to channel errors. In this case, the decoder will attempt to choose the most resemblent legitimately encoded sequence and output the corresponding bit-stream as the decoded string. These processes will be elaborated on in more detail later in the chapter.

The $n$ generator polynomials $g_1, g_2, \ldots, g_n$ are described by the specific connections to the register stages. Upon clocking the shift register, a new information bit is inserted in the register, while the bits constituting the states of this state machine move to the next register stage and the last bit is shifted out of the register. The generator polynomials are constituted by a binary
pattern, indicating the presence or absence of a specific link from a shift register stage by a binary one or zero, respectively. For example, in Figure 2.1 we observe that the generator polynomials are constituted by:
\[ g_1 = [1 \ 0 \ 0] \quad \text{and} \quad g_2 = [1 \ 1 \ 1], \] (2.1)
or, in an equivalent polynomial representation, as:
\[ g_1(z) = 1 + 0 \cdot z^1 + 0 \cdot z^2 \quad \text{and} \quad g_2(z) = 1 + z + z^2. \] (2.2)

We note that in a non-systematic CC, \( g_1 \) would also have more than one non-zero term. It is intuitively expected that the more constraints are imposed by the encoder, the more powerful the code becomes, facilitating the correction of a higher number of bits, which renders non-systematic CCs typically more powerful than their systematic counterparts.

Again, in a simple approach, we will demonstrate the encoding and decoding principles in the context of the systematic code specified as \( (k = 1) \), half-rate \( (R = k/n = 1/2) \), CC(2, 1, 2), with a memory of three binary stages \( (K = 2) \). These concepts can then be extended to arbitrary codecs. At the commencement of the encoding, the shift register is typically cleared by setting it to the all-zero state, before the information bits are input to it. Figure 2.1 demonstrates the encoder’s operation for the duration of the first ten clock cycles, tabulating the input bits, the

---

**Figure 2.1:** Systematic half-rate, constraint-length two convolutional encoder CC(2, 1, 2).
Figure 2.2: State transition diagram of the $C'C(2, 1, 2)$ systematic code, where broken lines indicate transitions due to an input one, while continuous lines correspond to input zeros.

shift-register states $s_1, s_2$, and the corresponding output bits. At this stage, the uninitiated reader is requested to follow the operations summarised in the figure before proceeding to the next stage of operations.

2.3 State and Trellis Transitions

An often-used technique for characterising the operations of a state machine, such as our convolutional encoder, is to refer to the state transition diagram of Figure 2.2. Given that there are two bits in the shift register at any moment, there are four possible states in the state machine and the state transitions are governed by the incoming bit $b_i$. A state transition due to a logical zero is indicated by a continuous line in the figure, while a transition activated by a logical one is represented by a broken line. The inherent constraints imposed by the encoder manifest themselves here in that from any state there are only two legitimate state transitions, depending on the binary input bit. Similarly, in each state there are two merging paths. It is readily seen from the encoder circuit of Figure 2.1 that, for example, from state $(s_1, s_2)=(1,1)$ a logical one input results in a transition to $(1,1)$, while an input zero leads to state $(0,1)$. The remaining transitions can also be readily checked by the reader. A further feature of this figure is that the associated encoded output bits are also plotted in the boxes associated with each of the transitions. Hence,
Another simple way of characterising the encoder is to portray its trellis diagram, which is depicted in Figure 2.3. At the left of the figure, the four legitimate encoder states are portrayed. Commencing operations from the all-zero register state (0,0) allows us to mirror the encoder’s actions seen in Figures 2.1 and 2.2 also in the trellis diagram, using the same input bit-stream. As before, the state transitions are governed by the incoming bits $b_i$ and a state transition due to a logical zero is indicated by a continuous line, while a transition activated by a logical one is represented by a broken line. 

Again, the inherent constraints imposed by the encoder manifest themselves here in that from any state there are only two legitimate state transitions, depending on the binary input bit, and in each state there are two merging paths. Given our specific input bit-stream, it is readily seen from the encoder circuit of Figure 2.1 and the state transition diagram of Figure 2.2 that, for example, from state $(s_1, s_2) = (0,0)$ a logical zero input bit results in a transition to (0,0), while an input one leads to state (1,0). The remaining transitions shown in the figure are associated with our specific input bit-stream, which can be readily explored by the reader. As before, the associated output bits are indicated in the boxes along each of the transitions. Hence, the trellis diagram gives a similarly unambiguous description of the encoder’s operations to the state diagram of Figure 2.2. Armed with the above description of CCs, we are now ready to give an informal description of the maximum likelihood Viterbi algorithm in the next section.


**2.4 The Viterbi Algorithm**

### 2.4.1 Error-free Hard-decision Viterbi Decoding

Given the received bit-stream, the decoder has to arrive at the best possible estimate of the original uncoded information sequence. Hence, the previously mentioned constraints imposed by the encoder on the legitimate bit sequences have to be exploited in order to eradicate illegitimate sequences and thereby remove the transmission errors. For the sake of computational simplicity, let us assume that the all-zero bit-stream has been transmitted and the received sequence of Figure 2.4 has been detected by the demodulator, which has been passed to the FEC decoder. We note...
here that if the demodulator carries out a binary decision concerning the received bit, this operation is referred to as hard-decision demodulation. By contrast, if the demodulator refrains from making a binary decision and instead it outputs a more finely graded multilevel confidence measure concerning the probability of a binary one and a binary zero, then it is said to invoke soft-decision demodulation.

For the time being we will consider only hard-decision demodulation. The decoder now has to compare the received bits with all the legitimate sequences of the trellis diagram of Figure 2.4 and quantify the probability of each of the associated paths, which ultimately assigns a probability-related quantity to specific decoded sequences, as we will show below.

Referring to Figure 2.4 and beginning the decoding operations from the all-zero state, we compute the Hamming distance of the received two-bit symbol with respect to both of the legitimate encoded sequences of the trellis diagram for the first trellis section (i.e. for the first trellis transition), noting that the Hamming distance is given by the number of different bit positions between two binary sequences. For example, for the first two-bit received symbol 10, the associated Hamming distances are 1 with respect to both the 00 and the 11 encoded sequences. Thus, at this stage the decoder is unable to express any preference as to whether 00 or 11 was the more likely transmitted symbol. We also note these Hamming distances in the trellis diagram of Figure 2.4, indicated at the top of the nodes corresponding to the new encoder states we arrived at, as a consequence of the state transitions due to a logical one and zero, respectively. These Hamming distances are known in the context of Viterbi decoding as the branch metric. The power of the Viterbi decoding algorithm accrues from the fact that it carries out a maximum likelihood sequence estimation, as opposed to arriving at symbol-by-symbol decisions, and thereby exploits the constraints imposed by the encoder on the legitimate encoded sequences. Hence, the branch metrics will be accumulated over a number of consecutive trellis stages before a decision as to the most likely encoder path and information sequence can be released.

Proceeding to the next received two-bit symbol, namely 00, the operations of the decoder are identical; that is, the Hamming distance between the encoded symbols of all four legitimate paths and the received symbol is computed. These distances yield the new branch metrics associated with the second trellis stage. By now the encoded symbols of two original input bits have been received, and this is why there are now four possible trellis states in which the decoder may reside. The branch metrics computed for these four legitimate transitions from top to bottom are 0, 2, 1 and 1, respectively. These are now added to the previous branch metrics of 1 in order to generate the path metrics of 1, 2, 3 and 2, respectively, quantifying the probability of each legitimate trellis path in terms of the accumulated Hamming distance. A low Hamming distance indicates a high similarity between the received sequence and the encoded sequence concerned, which is characteristic of the most likely encoded sequence, since the probability of a high number of errors is exponentially decreasing with the number of errors.

Returning to Figure 2.4 again, the corresponding accumulated Hamming distances or branch metrics from top to bottom are 1, 2, 3 and 2, respectively. At this stage, we can observe that the top branch has the lowest branch metric and hence it is the most likely encountered encoder path. The reader knows this, but the decoder can only quantify the probability of the corresponding paths and thus it cannot be sure of the validity of its decision. The other three encoder paths and their associated information bits also have a finite probability.

Continuing the legitimate paths of Figure 2.4 further, at trellis stage three the received sequence of 10 is compared to the four legitimate two-bit encoded symbols and the associated path metrics now become dependent on the actual path followed, since at this stage there are merging paths. For example, at the top node we witness the merger of the 00, 00, 00 path with the 11, 01, 01 path, where the associated original information bits were 0,0,0 and 1,0,0, respectively. On
the basis of the associated path metrics, the decoder may ‘suspect’ that the former one was the original information sequence, but it still refrains from carrying out a decision. Considering the two merging paths, future input bits would affect both of these in an identical fashion, resulting in an indistinguishable future set of transitions. Their path metrics will therefore also evolve identically, suggesting that it is pointless to keep track of both of the merging paths, since the one with the lower metric will always remain the more likely encoder path. This is reflected in Figure 2.4 by referring to the path exhibiting the lower metric as the survivor path, while the higher metric merging path will be discarded.

We also note that at the bottom node of trellis stage three we ended up with two identical path metrics, namely 3, and in this case a random decision must be made as to which one becomes the survivor. This event is indicated in Figure 2.4 by the arrow. In this particular example, this decision does not affect the final outcome of the decoder’s decision, since the top path appears to be the lowest metric path. Nonetheless, in some situations such random decisions will influence the decoded bit sequence and may indeed determine whether decoding errors are encountered. It is plausible that the confidence in the decoder’s decision is increased, as the accumulation of the branch metrics continues. Indeed, one may argue that the ‘best’ decision can be taken upon receiving the complete information sequence. However, deferring decisions for so long may not be acceptable in latency terms, in particular in delay-sensitive interactive speech or video communications. Nor is it necessary in practical terms, since experience shows that the decoder’s bit error rate is virtually unaffected by curtailing the decision interval to about five times the encoder’s memory, which was three in our example.

In our example, the received bit sequence does not contain any more transmission errors, and so it is plausible that the winning path remains the one at the top of Figure 2.4 and the associated branch metric of 2 reflects the actual number of transmission errors. We are now ready to release the error-free decoded sequence, namely the all-zero sequence, as seen explicitly in terms of the corresponding binary bits at the bottom of Figure 2.4. The corresponding winning path was drawn in bold in the figure.

### 2.4.2 Erroneous Hard-decision Viterbi Decoding

Following the above double-error correction scenario, below we consider another instructive example where the number of transmission errors remains two and even their separation is increased. Yet the decoder may become unable to correct the errors, depending on the outcome of a random decision. This is demonstrated in Figure 2.5 at stage four of the top path. Observe furthermore that the corresponding received bits and path metrics of Figure 2.4 are also indicated in Figure 2.5, but they are crossed out and superseded by the appropriately updated values according to the current received pattern. Depending on the actual choice of the survivor path at stage four, the first decoded bit may become a logical one, as indicated at the bottom of Figure 2.5. The accumulated Hamming distance becomes 2, regardless of the random choice of the survivor path, which indicates that in the case of decoding errors the path metric is not a reliable measure of the actual number of errors encountered. This will become even more evident in our next example.

Let us now consider a scenario in which there are more than two transmission errors in the received sequence, as seen in Figure 2.6. Furthermore, the bit errors are more concentrated, forming a burst of errors, rather than remaining isolated error events. In this example, we show that the decoder becomes ‘overloaded’ by the plethora of errors, and hence it will opt for an erroneous trellis path, associated with the wrong decoded bits.

Observe in Figure 2.4 that up to trellis stage three the lowest-metric path is the one at the
2.4. THE VITERBI ALGORITHM

Figure 2.5: Trellis-diagram-based Viterbi decoding of the CC(2, 1, 2) systematic code, where broken lines indicate transitions due to an input one, while continuous lines correspond to input zeros — erroneous hard-decision decoding of two isolated bit errors.
CHAPTER 2. CONVOLUTIONAL CHANNEL CODING

Figure 2.6: Trellis-diagram-based Viterbi decoding of the CC(2, 1, 2) systematic code, where broken lines indicate transitions due to an input one, while continuous lines correspond to input zeros — erroneous hard-decision decoding of burst errors.

top, which is associated with the error-free all-zero sequence. However, the double error in the fourth symbol of Figure 2.6 results in a ‘dramatic’ turn of events, since the survivor path deviates from the error-free all-zero sequence. Because of the specific received sequence encountered, the path metric remains 0 up to trellis stage \( J = 11 \), the last stage, accumulating a total of zero Hamming distance, despite actually encountering a total of six transmission errors, resulting in four decoding errors at the bottom of Figure 2.6. Again, the winning path was drawn in bold in Figure 2.4.

From this experience we can infer two observations. First, the high-error rate scenario encountered is less likely than the previously considered double-error case, but it has a finite probability and hence it may be encountered in practice. Second, since the decoder carries out a maximum likelihood decision, in such cases it will opt for the wrong decoded sequence, in which case the accumulated path metric will not correctly reflect the number of transmission errors encountered. We therefore conclude that, in contrast to block codes, CCs do not possess an ability to monitor the number of transmission errors encountered.
2.4.3 Error-free Soft-decision Viterbi Decoding

Having considered a number of hard-decision decoding scenarios, let us now demonstrate the added power of soft-decision decoding. Recall from our earlier discussions that if the demodulator refrains from making a binary decision and instead it outputs a finely graded soft-decision confidence measure related to the probability of a binary one and a binary zero, respectively, then it is said to invoke soft-decision demodulation. As an example, we may invoke an eight-level soft-decision demodulator output. This provides a more accurate indication of whether the demodulator’s decision concerning a certain demodulated bit is a high- or low-reliability one. This clearly supplies the Viterbi decoder with substantially more information than the previous binary zero/one decisions. Hence, a higher error correction capability will be achieved, as we will demonstrate in Figure 2.7.

Specifically, let us assume that on our eight-level confidence scale +4 indicates the highest possible confidence concerning the demodulator’s decision for a binary one and -4 the lowest possible confidence. In fact, if the demodulator outputs -4, the low confidence in a logical one implies a high confidence in a logical zero, and conversely, the demodulator output of +4 implies a very low probability of a binary zero. Bearing this eight-level confidence scale of \([-4, -3, \ldots +3, +4]\) in mind, the previously erroneously decoded double-error scenario of Figure 2.5 can now be revisited in Figure 2.7, where we will demonstrate that the more powerful soft-decision decoding allows us to remove all transmission errors.

Let us first consider the soft-decision metrics provided by the demodulator, which now replace the previously used hard-decision values at the bottom of Figure 2.5, appearing crossed out in Figure 2.7. For example, the first two values appear to be a high-confidence one and zero, respectively. The second two values are relatively high-confidence zeros, whereas the previously erroneously demodulated third symbol, namely 01, is now represented by the confidence values of -2,+1, indicating that these bits may well have been either one or zero. The rest of the soft-decision metrics appear to be of higher value, apart from the last-but-one.

The computation of the branch metrics and path metrics now has to be slightly modified. Previously, we were accumulating only ‘penalties’ in terms of the Hamming distances encountered. By contrast, in soft-decision decoding we will have to accumulate both penalties and credits, since we now consider the possibility of all demodulated values being both a binary one and a zero and quantify their associated probabilities using the soft-decision (SD) metrics. Explicitly, in Figure 2.7 we replace the crossed-out hard-decision metrics by the corresponding soft-decision metrics.

Considering trellis stage one and the 00 encoded symbol, the first SD metric of +3 does not tally well with the bit zero; rather, it indicates a strong probability of a one; hence, we accumulate a ‘penalty’ of -3. The second SD metric, however, indicates a strong probability of a zero, earning a credit of +3, which cancels the previous -3 penalty, yielding a SD branch metric of 0. Similar arguments are valid for the trellis branch from (0,0) to (1,0), which is associated with the encoded symbol 11, also yielding a SD branch metric of 0. During stage two, the received SD metrics of -2,-3 suggest a high probability of two zeros, earning an added credit of +5 along the branch (0,0) to (0,0). By contrast, these SD values do not tally well with the encoded symbol of 11 along the transition of (0,0) to (1,0), yielding a penalty of -5. During the third stage along the path from (0,0) to (0,0), we encounter a penalty of -1 and a credit of +2, bringing the total credits for the all-zero path to +6.

At this stage of the hard-decision decoder of Figure 2.7, we encountered a random decision, which now will not be necessary, since the merging path has a lower credit of +2. Clearly, at trellis stage nine we have a total credit of +29, allowing the decoder to release the correct original all-zero information sequence.
24 CHAPTER 2. CONVOLUTIONAL CHANNEL CODING

Figure 2.7: Trellis-diagram-based Viterbi decoding of the $CC(2, 1, 2)$ systematic code, where broken lines indicate transitions due to an input one, while continuous lines correspond to input zeros — error-free soft-decision decoding of two isolated bit errors.
2.5 Summary and Conclusions

This brief chapter commenced by providing a historical perspective on convolution coding. The convolutional encoder has been characterised with the aid of its state transition diagram and trellis diagram. Then the classic Viterbi algorithm has been introduced in the context of a simple decoding example, considering both hard- and soft-decision-based scenarios. In the next chapter, we focus our attention on the family of block codes, which have found numerous applications in both standard and proprietary wireless communications systems. A variety of video schemes characterised in this monograph have also opted for employing Bose–Chaudhuri–Hocquenghem codes.
Chapter 3

Block Coding

3.1 Introduction

Having studied the basics of convolutional coding in the previous chapter, we now focus our attention on methods of combating transmission errors with the aid of block codes. The codes discussed in this chapter have found favour in numerous standard communications systems and hence they are historically important. While a basic understanding of the construction of these codes is important for equipping the reader with the necessary background for studying turbo Bose–Chaudhuri–Hocquenghem codes, for example, the full appreciation of the associated so-called algebraic decoding techniques discussed in this this chapter is not necessary.

After demodulating the received signal, transmission errors can be removed from the digital information if their number does not exceed the error correcting power of the error correction code used. The employment of Forward Error Correction (FEC) coding techniques becomes important for transmissions over hostile mobile radio channels, where the violent channel fading precipitates bursts of error. This is particularly true when using vulnerable multilevel modulation schemes, such as Quadrature Amplitude Modulation (QAM). This chapter addresses issues of Reed–Solomon (RS) and Bose–Chaudhuri–Hocquenghem (BCH) coding in order to provide a self-contained reference for readers who want to delve into the theory of FEC coding.

The theory and practice of FEC coding has been well documented in classic references [32, 82, 84, 87–90, 96, 97]; hence for an in-depth treatment, the reader is referred to these sources. Both convolutional and block FEC codes have been successfully utilised in various communications systems [108, 109]. In this brief overview, to a certain extent we will follow the philosophy of these references, which are also recommended for a more detailed discourse on the subject. Since the applications described in this book often use the family of RS and BCH block codes, in this chapter we concentrate on their characterisation. We will draw the reader’s attention to any differences between them, as they arise during our discussions. Both RS and BCH codes are defined over the mathematical structure of finite fields; therefore we briefly consider their construction. For a more detailed discourse on finite fields, the reader is referred to Lidl’s work [95].


3.2 Finite Fields

3.2.1 Definitions

Loosely speaking, an algebraic field is any arithmetic structure in which addition, subtraction, multiplication and division are defined and the associative, distributive and commutative laws apply. In conventional algebraic fields, like the rational, real or complex field, these operations are trivial, but in other fields they are somewhat different, such as modulo or modulo polynomial operations.

A more formal field definition can be formulated as follows [90]. An algebraic field is a set of elements that has addition and multiplication defined over it, such that the set is closed under both addition and multiplication; that is, the results of these operations are also elements in the same set. Furthermore, both addition and multiplication are associative and commutative. There is a zero element, such that $a + 0 = a$, and an additive inverse element, such that $a + (-a) = 0$. Subtraction is defined as $a - b = a + (-b)$. There is a one element, such that $1 \cdot a = a$, and a multiplicative inverse element for which we have $a \cdot (a^{-1}) = 1$. Division is defined as $a/b = a \cdot (b^{-1})$.

Digital Signal Processing (DSP) has been historically studied in the real algebraic field constituted by all finite and infinite decimals, or in the complex algebraic field with elements of the form $a + jb$, where $a$ and $b$ are real. In both the real and complex fields, the operations are fairly straightforward and well known, and the number of elements is infinite; consequently, they are infinite algebraic fields.

Finite algebraic fields or shortly finite fields are constituted by a finite number $q$ of elements if there exists a field with $q$ elements. They are also referred to as Galois Fields $GF(q)$. Every field must have a zero and a one element. Consequently, the smallest field is $GF(2)$ with mod 2 addition and multiplication, because the field must be closed under the operations. That is, their result must be a field element, too. If the number of elements in a field is a prime $p$, the $GF(p)$ constructed from the elements $\{0, 1, 2, \ldots, (p-1)\}$ is called a prime field with modulop operations.

**Example 1:** Galois prime-field modulo operation.

As an illustrative example, let us briefly consider the basic operations, namely, addition and multiplication, over the prime field $GF(7)$. The prime field $GF(7)$ is constituted by the elements $\{0, 1, 2, 3, 4, 5, 6\}$, and the corresponding modulo $-7$ addition and multiplication tables are summarised in Table 3.1.

Since every field must contain the unity element, we can define the order $n$ of a field element $a$, such that $a^n = 1$. In other words, to determine the order of a field element the
3.2. FINITE FIELDS

Exponent $n$ yielding $\alpha^n = 1$ has to be found. Every $GF(p)$ has at least one element $\alpha$ of order $(p - 1)$, which is called a primitive element and which always exists in a $GF(p)$ [32]. Hence, $\alpha^{(p-1)} = 1$ must apply. For practical FEC applications we must have sufficiently large finite fields, constituted by many field elements, since only this finite number of elements can be used in order to represent our codewords.

Example 2: Extension field construction example.

In order to augment the concept of extension fields, let us consider first the simple example of extending the real field to the complex field. Thus, we attempt to find a second-order polynomial over the real field that cannot be factorised over it, which is also referred to in finite field parlance as an irreducible polynomial. For example, it is well understood that the polynomial $x^2 + 1$ cannot be factorised over the real field because it has no real zeros. However, one can define a special zero $j$ of the polynomial $x^2 + 1$, which we refer to as an imaginary number, for which $j^2 = -1$ and $x^2 + 1 = (x+j)(x-j)$. This way the real number system has been extended to the complex field, which has complex elements $\alpha_1 + j\alpha_2$, where $\alpha_1$ and $\alpha_2$ are real. The new number system derived from the real field fulfills all the criteria necessary for an extension field.

The elements $\alpha_1 + j\alpha_2$ of the complex field can also be interpreted as first-order polynomials $\alpha_1 x^0 + \alpha_2 x^1$ having real coefficients, if we replace $j$ by $x$. In this context, multiplication of the field elements $(\alpha_1 + \alpha_2 x)$ and $(\alpha_3 + \alpha_4 x)$ is equivalent to modulo polynomial multiplication of the field elements:

$$
\alpha_5 + \alpha_6 x = (\alpha_1 + \alpha_2 x)(\alpha_3 + \alpha_4 x) \pmod{x^2 + 1} \\
= \alpha_1\alpha_3 + \alpha_2\alpha_3 x + \alpha_1\alpha_4 x + \alpha_2\alpha_4 x^2 \pmod{x^2 + 1} \\
= \alpha_1\alpha_3 + (\alpha_2\alpha_3 + \alpha_1\alpha_4)x + \alpha_2\alpha_4 x^2 \pmod{x^2 + 1},
$$

where $\alpha_1 \ldots \alpha_6$ are arbitrary elements of the original real field. But since $x^2 \pmod{x^2 + 1} = \text{Remainder}(x^2 : (x^2 + 1)) = -1$, this polynomial formulation delivers identical results to those given by the complex number formulation, where $j^2 = -1$. So we may write:

$$
\alpha_5 + \alpha_6 x = (\alpha_1\alpha_3 - \alpha_2\alpha_4) + (\alpha_2\alpha_3 + \alpha_1\alpha_4)x \\
\alpha_5 = \alpha_1\alpha_3 - \alpha_2\alpha_4 \quad (3.3)
$$

Pursuing this polynomial representation, we find that the general framework for extending the finite prime field $GF(p)$ is to have $m$ components rather than just two.

Explicitly, a convenient way of generating large algebraic fields is to extend the prime field $GF(p)$ to a so-called extension field $GF(p^m)$. In general, the elements of the extension field $GF(p^m)$ are all the possible $m$ dimensional vectors, where all $m$ vector coordinates are elements of the original prime field $GF(p)$, and $m$ is an integer. For example, an extension field $GF(p^m)$ of the original prime field $GF(p)$ with $p = 2$ elements $\{0,1\}$ contains all the possible combinations of $m$ number of $GF(p)$ elements, which simply means that there are $p^m$ number of extension field elements. Consequently, an element of the extension field $GF(p^m)$ can be written as a polynomial of order $m-1$, with coefficients from $GF(p)$:

$$
GF(p^m) = \{(a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots + a_{m-1} x^{m-1}) \}
$$
with
\[ \{a_0, a_1, \ldots, a_{m-1}\} \in \{0, 1 \ldots (p - 1)\}. \]

The operations in the extension field \(GF(p^m)\) are \textit{modulo polynomial} operations rather than conventional modulo operations. Hence, the addition is carried out as the addition of two polynomials:
\[
c(x) = a(x) + b(x) \\
= a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1} + b_0 + b_1 x + b_2 x^2 + \ldots + b_{m-1} x^{m-1} \\
= (a_0 + b_0) + (a_1 + b_1)x^1 + (a_2 + b_2)x^2 + \ldots + (a_{m-1} + b_{m-1})x^{m-1}.
\]

(3.5)

(3.6)

(3.7)

Since
\[ \{a_0, a_1, \ldots, a_{m-1}\} \in \{0, 1, \ldots, (p - 1)\} \]

and
\[ \{b_0, b_1, \ldots, b_{m-1}\} \in \{0, 1, \ldots, (p - 1)\}, \]

the component-wise addition of the polynomial coefficients must be \textit{mod} \(p\) addition, so that the coefficients of the result polynomial are also elements of \(GF(p)\).

The definition of the modulo polynomial multiplication is somewhat more complicated. First, one has to find an irreducible polynomial \(p(x)\) of degree \(m\), which cannot be factorised over \(GF(p)\). In other words, \(p(x)\) must be a polynomial with coefficients from \(GF(p)\), and it must not have any zeros; that is, it cannot be factorised into polynomials of lower order with coefficients from \(GF(p)\). Recall, for example, that the polynomial \(x^2 + 1\) was irreducible over the real field. More formally, the polynomial \(p(x)\) is \textit{irreducible} if and only if it is divisible exclusively by \(\alpha p(x)\) or by \(\alpha\), where \(\alpha\) is an arbitrary field element in \(GF(p)\). Whenever the highest-order coefficient of an irreducible polynomial is equal to one, the polynomial is called a \textit{prime} polynomial.

Once we have found an appropriate prime polynomial \(p(x)\), the multiplication of two extension field elements over the extension field is modulo polynomial multiplication of the polynomial representations of the field elements. Explicitly, the product of the two extension field elements given by their polynomial representations must be divided by \(p(x)\), and the remainder must be retained as the final result. This is to ensure that the result is also an element of the extension field (i.e. a polynomial with an order less than \(p\)). Then the extension field is declared closed under the operations, yielding results that are elements of the original field. Accordingly, the nature of the extension field depends on the choice of the prime polynomial \(p(x)\). Sometimes there exist several prime polynomials, and some of them result in somewhat more advantageous extension field construction than others. Sometimes there is no prime polynomial of a given degree \(m\) over a given prime field \(GF(p)\); consequently, \(GF(p)\) cannot be extended to \(GF(p^m)\). At this stage, we note that we have constructed a mathematical environment necessary for representing our information carrying signals (i.e. codewords), and defined the operations over the extension field.

Since the source data to be encoded are usually binary, we represent the non-binary information symbols of the extension field as a sequence of \(m\) bits. Logically, our prime field is the
3.2. FINITE FIELDS

binary field $GF(2)$ with elements $\{0, 1\}$ and $p = 2$. Then we extend $GF(2)$ to $GF(2^m)$ in order to generate a sufficiently large working field for signal processing, where an $m$-bit non-binary information symbol constitutes a field element of $GF(2^m)$. If, for example, an information symbol is constituted by one byte of information (i.e. $m = 8$), the extension field $GF(2^m) = GF(256)$ contains 256 different field elements. The appropriate prime polynomial can be chosen, for example, from Table 4.7 on p. 409 of reference \[109\], where primitive polynomials of degree less than 25 over $GF(2)$ are listed. These prime polynomials allow us to construct any arbitrary extension field from $GF(2^2)$ to $GF(2^{25})$.

It is useful for our further discourse to represent the field elements of $GF(q = 2^m)$ each as a unique power of an element $\alpha$, which we refer to as the primitive element. The primitive element $\alpha$ was earlier defined to be the one that fulfills the condition $\alpha^{q-1} = \alpha^{2^m-1} = 1$. With this notation the elements of the extension field $GF(q = 2^m)$ can be written as $\{0, 1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{q-2}\}$, and their polynomial representation is given by the remainder of $x^n$ upon division by the prime polynomial $p(x)$:

$$\alpha^n = \text{Remainder } \{x^n/p(x)\}.$$ (3.8)

This relationship will become clearer in the practical example of the next subsection.

3.2.2 Galois Field Construction

Example 3: $GF(16)$ construction example.

As an example, let us consider the construction of $GF(2^4) = GF(16)$ based on the prime polynomial $p(x) = x^4 + x + 1$ [109], p. 409, where $m = 4$. Each extension field element $\alpha$ can be represented as a polynomial of degree three with coefficients over $GF(2)$. Equivalently, each extension field element can be described by the help of the binary representation of the coefficients of the polynomial or by means of the decimal value of the binary representation.

Let us proceed with the derivation of all the representations mentioned above, which are summarised in Table 3.2. Since every field must contain the zero and one element, we have:

$$0 = 0$$

$$\alpha^0 = 1.$$  

Further plausible assignments are:

$$\alpha^1 = x$$

$$\alpha^2 = x^2$$

$$\alpha^3 = x^3,$$

because the remainders of $x$, $x^2$ and $x^3$ upon division by the primitive polynomial $p(x) = x^4 + x + 1$ are themselves. However, the polynomial representation of $x^4$ cannot be derived without polynomial division:

$$\alpha^4 = \text{Rem } \{x^4 : (x^4 + x + 1)\}_{p(x)=x^4+x+1}$$
### Table 3.2: Different representations of $GF(16)$ elements ($\alpha^{15} \equiv \alpha^0 \equiv 1$) generated using the prime polynomial $p(x) = x^4 + x + 1$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>$0$</td>
<td>$0000$</td>
<td>$0$</td>
</tr>
<tr>
<td>$\alpha^0$</td>
<td>$1$</td>
<td>$0001$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\alpha^1$</td>
<td>$x$</td>
<td>$0010$</td>
<td>$2$</td>
</tr>
<tr>
<td>$\alpha^2$</td>
<td>$x^2$</td>
<td>$0100$</td>
<td>$4$</td>
</tr>
<tr>
<td>$\alpha^3$</td>
<td>$x^3$</td>
<td>$1000$</td>
<td>$8$</td>
</tr>
<tr>
<td>$\alpha^4$</td>
<td>$x + 1$</td>
<td>$0011$</td>
<td>$3$</td>
</tr>
<tr>
<td>$\alpha^5$</td>
<td>$x^2 + x$</td>
<td>$0110$</td>
<td>$6$</td>
</tr>
<tr>
<td>$\alpha^6$</td>
<td>$x^3 + x^2$</td>
<td>$1100$</td>
<td>$12$</td>
</tr>
<tr>
<td>$\alpha^7$</td>
<td>$x^3 + x + 1$</td>
<td>$1011$</td>
<td>$11$</td>
</tr>
<tr>
<td>$\alpha^8$</td>
<td>$x^2 + 1$</td>
<td>$0101$</td>
<td>$5$</td>
</tr>
<tr>
<td>$\alpha^9$</td>
<td>$x^3 + x$</td>
<td>$1010$</td>
<td>$10$</td>
</tr>
<tr>
<td>$\alpha^{10}$</td>
<td>$x^2 + x + 1$</td>
<td>$0111$</td>
<td>$7$</td>
</tr>
<tr>
<td>$\alpha^{11}$</td>
<td>$x^3 + x^2 + x$</td>
<td>$1110$</td>
<td>$14$</td>
</tr>
<tr>
<td>$\alpha^{12}$</td>
<td>$x^3 + x^2 + x + 1$</td>
<td>$1111$</td>
<td>$15$</td>
</tr>
<tr>
<td>$\alpha^{13}$</td>
<td>$x^3 + x^2 + 1$</td>
<td>$1101$</td>
<td>$13$</td>
</tr>
<tr>
<td>$\alpha^{14}$</td>
<td>$x^5 + 1$</td>
<td>$1001$</td>
<td>$9$</td>
</tr>
</tbody>
</table>

The polynomial division:

\[
x^4 : (x^4 + x + 1) = 1
\]  

\[
-(x^4 + x + 1)
\]  

The remainder:

\[
-(x + 1).
\]

Hence we have:

\[
\alpha^4 = x + 1.
\]

Following these steps the first two columns of Table 3.2 can easily be filled in. The binary notation simply comprises the coefficients of the polynomial representation, whereas the decimal notation is the decimal number computed from the binary representation.

### 3.2.3 Galois Field Arithmetic

Multiplication in the extension field is carried out by multiplying the polynomial representations of the elements, dividing the result by the prime polynomial $p(x) = x^4 + x + 1$, and finally taking the remainder. For example, for the field elements:

\[
\alpha^4 \equiv x + 1
\]

\[
\alpha^7 \equiv x^3 + x + 1
\]
we have:

\[
\alpha^4 \cdot \alpha^7 = (x + 1)(x^3 + x + 1) \mod_{p(\mathbb{F}_2)} = x^4 + x + 1
\]

Then the polynomial division is computed as follows:

\[
(x^4 + x^3 + x^2 + 0 + 1) : (x^4 + x + 1) = 1
\]

The remainder:

\[
x^3 + x^2 + x = x^3 + x^2 + x.
\]

Therefore, the required product is given by:

\[
\alpha^4 \cdot \alpha^7 = x^3 + x^2 + x \equiv \alpha^{11}.
\]

From Table 3.2 it is readily recognised that the exponential representation of the field elements allows us to simply add the exponents without referring back to their polynomial representations. Whenever the exponent happens to be larger than \( q - 1 = 15 \), it is simply collapsed back into the finite field by taking its value modulo \( q - 1 \), that is modulo 15. For example:

\[
\alpha^{12} \cdot \alpha^6 = \alpha^{18} = \alpha^{15} \cdot \alpha^3 = 1, \alpha^3 = \alpha^3.
\]

The addition of two field elements is also carried out easily by referring to the polynomial representation by the help of a component-wise addition, as follows:

\[
\alpha^{11} + \alpha^7 \equiv (x^3 + x^2 + x + 0) + (x^3 + 0 + x + 1) = x^2 + 1 \equiv \alpha^8.
\]

This is also equivalent to the modulo-2 addition of the binary representations:

\[
\alpha^{11} + \alpha^7 \equiv \begin{array}{c}
1110 \\
+ \ 1011 \\
0101 \equiv x^2 + 1 \equiv \alpha^8.
\end{array}
\]

The fastest way to compute \( GF(16) \) addition of exponentially represented field elements is to use the precomputed \( GF(16) \) addition table, namely Table 3.3. With our background in finite fields, we can now proceed to define and characterise the important so-called maximum–minimum distance family — a term to be clarified during our later discussion — of non-binary block codes referred to as RS codes and their binary subclass, BCH codes, which are often used in our prototype systems.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>α⁰</td>
<td>0001</td>
<td>1</td>
<td>0</td>
<td>α⁴</td>
<td>α¹</td>
<td>α³</td>
<td>α²</td>
<td>α¹⁰</td>
<td>α⁸</td>
<td>α⁵</td>
<td>α¹⁴</td>
<td>α³</td>
<td>α¹⁰</td>
<td>α⁷</td>
<td>α⁶</td>
<td>α¹³</td>
<td>α¹¹</td>
</tr>
<tr>
<td>α¹</td>
<td>0010</td>
<td>2</td>
<td>α⁴</td>
<td>0</td>
<td>α⁰</td>
<td>α⁵</td>
<td>α¹⁰</td>
<td>α²</td>
<td>α⁸</td>
<td>α⁹</td>
<td>α⁷</td>
<td>α³</td>
<td>α¹⁴</td>
<td>α¹²</td>
<td>α⁶</td>
<td>α¹³</td>
<td>α¹¹</td>
</tr>
<tr>
<td>α²</td>
<td>0011</td>
<td>3</td>
<td>α¹</td>
<td>α⁰</td>
<td>0</td>
<td>α¹⁰</td>
<td>α³</td>
<td>α⁸</td>
<td>α²</td>
<td>α⁴</td>
<td>α⁰</td>
<td>α¹⁴</td>
<td>α¹²</td>
<td>α³</td>
<td>α¹³</td>
<td>α¹¹</td>
<td>α¹⁰</td>
</tr>
<tr>
<td>α³</td>
<td>0100</td>
<td>4</td>
<td>α⁸</td>
<td>α⁵</td>
<td>α¹⁰</td>
<td>0</td>
<td>α⁰</td>
<td>α¹</td>
<td>α⁴</td>
<td>α¹³</td>
<td>α⁶</td>
<td>α¹²</td>
<td>α³</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α¹⁴</td>
<td>α¹⁰</td>
</tr>
<tr>
<td>α⁴</td>
<td>0101</td>
<td>5</td>
<td>α²</td>
<td>α¹⁰</td>
<td>α³</td>
<td>α⁰</td>
<td>0</td>
<td>α⁴</td>
<td>α¹</td>
<td>α³</td>
<td>α¹³</td>
<td>α⁶</td>
<td>α¹²</td>
<td>α³</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α¹⁴</td>
</tr>
<tr>
<td>α⁵</td>
<td>0110</td>
<td>6</td>
<td>α¹⁰</td>
<td>α²</td>
<td>α⁸</td>
<td>α¹</td>
<td>0</td>
<td>α⁰</td>
<td>α¹¹</td>
<td>α¹²</td>
<td>α⁶</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
<td>α¹¹</td>
</tr>
<tr>
<td>α⁶</td>
<td>1000</td>
<td>8</td>
<td>α¹⁴</td>
<td>α⁹</td>
<td>α⁶</td>
<td>α¹³</td>
<td>α¹²</td>
<td>α⁶</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
</tr>
<tr>
<td>α⁷</td>
<td>1001</td>
<td>9</td>
<td>α³</td>
<td>α⁰</td>
<td>α⁶</td>
<td>α¹²</td>
<td>α¹¹</td>
<td>α⁰</td>
<td>α⁴</td>
<td>α³</td>
<td>α⁰</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
<td>α¹¹</td>
</tr>
<tr>
<td>α⁸</td>
<td>1010</td>
<td>A</td>
<td>α⁶</td>
<td>α³</td>
<td>α¹⁴</td>
<td>α¹¹</td>
<td>α¹²</td>
<td>α⁶</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
</tr>
<tr>
<td>α⁹</td>
<td>1011</td>
<td>B</td>
<td>α⁹</td>
<td>α¹⁴</td>
<td>α³</td>
<td>α¹²</td>
<td>α¹¹</td>
<td>α⁰</td>
<td>α⁴</td>
<td>α³</td>
<td>α⁰</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
<td>α¹¹</td>
</tr>
<tr>
<td>α¹⁰</td>
<td>1100</td>
<td>C</td>
<td>α³</td>
<td>α¹¹</td>
<td>α¹²</td>
<td>α⁹</td>
<td>α⁰</td>
<td>α⁷</td>
<td>α³</td>
<td>α⁰</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
<td>α¹¹</td>
<td>α¹⁰</td>
</tr>
<tr>
<td>α¹¹</td>
<td>1101</td>
<td>D</td>
<td>α⁶</td>
<td>α¹⁴</td>
<td>α³</td>
<td>α¹²</td>
<td>α⁹</td>
<td>α³</td>
<td>α⁰</td>
<td>α⁷</td>
<td>α³</td>
<td>α⁰</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
</tr>
<tr>
<td>α¹²</td>
<td>1110</td>
<td>E</td>
<td>α¹²</td>
<td>α⁵</td>
<td>α³</td>
<td>α⁰</td>
<td>α⁷</td>
<td>α³</td>
<td>α⁰</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
<td>α¹¹</td>
<td>α¹⁰</td>
<td>α⁹</td>
</tr>
<tr>
<td>α¹³</td>
<td>1111</td>
<td>F</td>
<td>α¹³</td>
<td>α⁶</td>
<td>α⁰</td>
<td>α⁷</td>
<td>α³</td>
<td>α⁰</td>
<td>α¹³</td>
<td>α⁷</td>
<td>α⁰</td>
<td>α³</td>
<td>α¹³</td>
<td>α¹¹</td>
<td>α¹⁰</td>
<td>α⁹</td>
<td>α⁰</td>
</tr>
</tbody>
</table>
3.3 Reed-Solomon and Bose-Chaudhuri-Hocquenghem Block Codes

3.3.1 Definitions

As mentioned above, RS codes represent a non-binary subclass of (BCH) multiple error-correcting codes. Because of their non-binary nature, RS codes pack the information represented by \( m \) consecutive bits into non-binary symbols, which are elements in the extension field \( GF(2^m) \) of \( GF(2) \). In general, an RS code is defined as a block of \( n \) non-binary symbols over \( GF(2^m) \), constructed from \( k \) original information symbols by adding \( n-k=2t \) number of redundancy symbols from the same extension field, giving \( n=k+2t \). This code is often written as \( RS(n,k,t) \) over \( GF(2^m) \).

RS codes are cyclic codes, implying that any cyclically shifted version of a codeword is also a codeword of the same code. A further property of cyclic codes is that all of the codewords can be generated as a linear combination of any codeword and its cyclically shifted versions. The distance properties of a code are crucial in assessing its error correction capability. The minimum distance of a code is the minimum number of positions in which two arbitrary codewords differ. If we define the weight of a codeword as the number of non-zero symbol positions, then the minimum distance of a code is actually the weight of its minimum weight codeword, since the ‘all-zero’ word is always a codeword in a linear code. It is plausible that if in an \( (n, k) \) code less than half the minimum distance number of symbols are in error, it can be uniquely recognised which \( k \) symbol’s long information message has been sent. This is because the received message is still closer to the transmitted one than to any other one. On the other hand, if more than half the minimum distance number of symbols are in error, the decoder decides erroneously on the basis of the nearest legitimate codeword.

The Singleton bound imposes an upper limit on the maximum achievable minimum distance of a code on the following basis. If in a codeword one information symbol is changed, the highest possible distance for the newly computed codeword from the original one will be \( d=(n-k)+1 \), provided that all the \( n-k \) parity symbols also changed as a result, which is an extreme assumption. Consequently, for the code’s minimum distance we have \( d_{min} \leq (n-k)+1 \). RS codes are one of the very few codes which reach the maximum possible minimum distance of \( d=(n-k)+1 \). This is why they are referred to as maximum-minimum distance codes. In general, an \( RS(n,k,t) \) code can correct up to \( t=(n-k)/2 \) symbol errors, or in other words a \( t \) error correcting code must contain \( 2t=(n-k) \) number of redundancy symbols. Therefore, the minimum distance between codewords must be \( d_{min} \geq 2t+1 \). Substituting \( (n-k) = 2t \) into \( d_{min} \geq 2t+1 \) and taking into account that previously we showed that \( d_{min} \leq (n-k)+1 \), we find that the minimum distance of RS codes is exactly \( d_{min} = 2t+1 \).

Before we proceed to define the RS encoding rules, the generator polynomial has to be introduced, which is defined to be the polynomial of order \( 2t \), which has its zeros at any \( 2t \) out of the \( 2^m \) possible field elements. For the sake of simplicity, but without any loss of generality, we always use the \( GF \) elements \( \alpha, \alpha^2, \ldots, \alpha^{2t} \) in order to determine the generator polynomial of an RS code:

\[
g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{2t}) = \prod_{j=1}^{2t} (x - \alpha^j) = \sum_{j=0}^{2t} g_jx^j. \tag{3.12}
\]

In the next subsection, we highlight two algorithms using the above-mentioned generator polynomial for the encoding of the information symbols in order to generate RS-coded codewords.
3.3.2 RS Encoding

Since RS codes are cyclic, any cyclic encoding method can be used for their coding. When opting for the non-systematic encoding rule, the information symbols are not explicitly recognisable in the encoded codeword — hence the terminology. Non-systematic cyclic encoders generate the encoded word by multiplying the information polynomial $i(x)$ by the generator polynomial $g(x)$ using modulo polynomial algebra as follows:

$$c(x) = i(x) \cdot g(x) \quad (3.13)$$

$$i(x) = i_1 x + i_2 x^2 + \ldots + i_k x^k = \sum_{j=1}^{k} i_j x^j \quad (3.14)$$

$$c(x) = c_1 x + c_2 x^2 + \ldots + c_n x^n = \sum_{j=1}^{n} c_j x^j \quad (3.15)$$

where the information polynomial $i(x)$ is of order $k$ and the encoded codeword polynomial $c(x)$ is of order $n = k + 2t$. The coefficients of the polynomials are the non-binary information carrying symbols, which are elements in $GF(2^m)$, and the powers of $x$ can be thought of as place markers for the symbols in the codeword. Again, since the polynomial $c(x)$ does not contain explicitly the original $k$ information symbols, this is called a non-systematic encoder. The codeword is then modulated and sent over a non-ideal transmission medium, where the code symbols or polynomial coefficients may become corrupted.

The channel effects can be modelled by the help of an additive error polynomial $e(x)$ of order $n$ as follows:

$$r(x) = c(x) + e(x) = c_1 x + c_2 x^2 + \ldots + c_n x^n + e_1 x + e_2 x^2 + \ldots + e_n x^n, \quad (3.16)$$

where $r(x)$ is the corrupted received polynomial. The decoder has to determine up to $t$ error positions and compute $t$ error locations. In other words, the decoder has to compute $t + t = 2t$ unknowns from the $2t$ redundancy symbols and correct the errors to produce the error-free codeword $c(x)$. After error correction, the information symbols must be recovered by the help of the inverse operation of the encoding, using the following simple decoding rule:

$$i(x) = \frac{c(x)}{g(x)} \quad (3.17)$$

However, if there are more than $t$ transmission errors in the received polynomial $r(x)$, the decoder fails to produce the error-free codeword polynomial $c(x)$. This is plausible, since owing to its excessive corruption the received codeword will become more similar to another legitimate codeword rather than the transmitted one. In algebraic terms, this is equivalent to saying that we cannot determine more than $t$ unknown error positions and error values, when using $2 \times t$ redundancy symbols. We note that in the case of binary BCH codes it is plausible that having determined the error positions, the corresponding bits are simply inverted in order to correct them, while for non-binary RS codes a more complicated procedure will have to be employed. These statements will be made more explicit during our further discourse.

The systematic RS encoding rule is somewhat more sophisticated than the non-systematic one, but the original information symbols are simply copied into the encoded word. This property is often attractive, because in some applications it is advantageous to know where the original information symbols reside in the encoded block. Explicitly, when a received codeword is deemed
to be overwhelmed by transmission errors, a non-systematic RS or BCH code has no other option than to attempt to compute the error locations and ‘magnitudes’ for their correction, but this operation will be flawed by the plethora of transmission errors, and hence the decoding operation may actually corrupt more received symbols by carrying out a flawed decoding operation.

By contrast, in systematic RS or BCH codecs, instead of erroneously computing the error locations and ‘magnitudes’ for their correction, this ‘code overload’ condition can be detected. Hence, this flawed action can be avoided by simply extracting the \( k \) original information symbols from \( c(x) \). Because of these differences, non-systematic codes usually result in a Bit Error Rate (BER) increase under hostile channel conditions, while powerful systematic codes can maintain a similar BER performance to the uncoded case under similar circumstances.

Again, in systematic RS codes the first \( k \) encoded symbols are chosen to be the original \( k \) information symbols. We simply multiply the information polynomial \( i(x) \) by \( x^{n-k} \) in order to shift it into the highest-order position of the codeword \( c(x) \). Then we choose the parity symbols constituted by the polynomial \( p(x) \) according to the systematic encoding rule in order to result in a legitimate codeword. Legitimate in this sense means that the remainder of the encoded word \( c(x) \) upon division by the generator polynomial \( g(x) \) is zero. Using the codeword \( c(x) \) hosting the shifted information word \( i(x) \) plus the parity segment \( p(x) \), we have:

\[
c(x) = x^{(n-k)} \cdot i(x) + p(x),
\]

and according to the above definition of \( c(x) \), we have:

\[
\text{Rem} \left\{ \frac{c(x)}{g(x)} \right\} = 0 \quad (3.19)
\]

\[
\text{Rem} \left\{ \frac{x^{(n-k)} \cdot i(x) + p(x)}{g(x)} \right\} = 0 \quad (3.20)
\]

\[
\text{Rem} \left\{ \frac{x^{(n-k)} \cdot i(x)}{g(x)} \right\} + \text{Rem} \left\{ \frac{p(x)}{g(x)} \right\} = 0. \quad (3.21)
\]

Since the order of the parity polynomial \( p(x) \) is less than \( (n-k) \) and the order of \( g(x) \) is \( (n-k) \), we have:

\[
\text{Rem} \left\{ \frac{p(x)}{g(x)} \right\} = p(x). \quad (3.22)
\]

By substituting Equation 3.22 into Equation 3.21 and rearranging it, we get:

\[
- \text{Rem} \left\{ \frac{x^{(n-k)} \cdot i(x)}{g(x)} \right\} = p(x). \quad (3.23)
\]

Hence, if we substitute Equation 3.23 into Equation 3.18 and take into account that over GFs addition and subtraction are the same, the systematic encoding rule is as follows:

\[
c(x) = x^{(n-k)} \cdot i(x) + \text{Rem} \left\{ \frac{x^{(n-k)} \cdot i(x)}{g(x)} \right\} . \quad (3.24)
\]

The error correction ensues in a completely equivalent manner to that of the non-systematic decoder, which will be the subject of our later discussion, but recovering the information symbols from the corrected received codeword is simpler. Namely, the first \( k \) information symbols of an \( n \)-symbol long codeword are retained as corrected decoded symbols. Let us now revise these definitions and basic operations with reference to the following example.
### 3.3.3 RS Encoding Example

**Example 4:**
Systematic and non-systematic $RS(12, 8, 2)$ encoding example.

Let us consider a low-complexity double-error correcting RS code, namely the $RS(12, 8, 2)$ code over $GF(16)$, and demonstrate the operation of both the systematic and non-systematic encoder. We begin our example with the determination of the generator polynomial $g(x)$, which is a polynomial of order $2t = 4$, having zeros at the first four elements of $GF(16)$, namely at $\alpha^1, \alpha^2, \alpha^3$ and $\alpha^4$. Recall, however, that we could have opted for the last four $GF(16)$ elements or any other four elements of it. Remember furthermore that multiplication is based on adding the exponents of the exponential representations, while addition is conveniently carried out using Table 3.3. Then the generator polynomial is arrived at as follows:

$$
g(x) = (x - \alpha^1)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)
= (x^2 - \alpha^4 x + \alpha^3)(x^2 - \alpha^3 x + \alpha^4) + \alpha^3 x + \alpha^4
= (x^2 - (\alpha^1 + \alpha^2) x + \alpha^3)(x^2 - (\alpha^3 + \alpha^4) x + \alpha^7)
= (x^2 + \alpha^5 x + \alpha^3)(x^2 + \alpha^7 x + \alpha^7)
= x^4 + \alpha^7 x^3 + \alpha^7 x^2 + \alpha^5 x^3 + \alpha^{12} x^2 + \alpha^{12} x + \alpha^{10} x + \alpha^{10}
= x^4 + (\alpha^7 + \alpha^5 + \alpha^{12} + \alpha^3) x^3 + (\alpha^7 + \alpha^{12} + \alpha^3) x^2 + (\alpha^{12} + \alpha^{10}) x + \alpha^{10}
= x^4 + \alpha^{13} x^3 + \alpha^6 x^2 + \alpha^3 x + \alpha^{10}.
$$

Let us now compute the codeword polynomial $c(x)$ for the $RS(12, 8, 2)$ double-error correcting code, when the ‘all-one’ information polynomial $i(x) = 11\ldots 11$ is to be encoded. In hexadecimal format, this can be expressed as $i(x) = F F F F F F F F \#H$. Since $GF(2^m) = GF(16)$ for $m = 4$, $8.4 = 32$ bits; that is, 8 hexadecimal symbols must be encoded into 12 hexadecimal symbols. The exponential representation of the $1111 = F \#H$ hexadecimal symbol is $\alpha^{12}$, as seen in Table 3.2. Hence, the seventh-order information polynomial is given by:

$$
i(x) = \alpha^{12} x^7 + \alpha^{12} x^6 + \alpha^{12} x^5 + \alpha^{12} x^4 + \alpha^{12} x^3 + \alpha^{12} x^2 + \alpha^{12} x + \alpha^{12}.
$$

Then the non-systematic encoder simply computes the product of the information poly-
3.3. RS AND BCH CODES

where:

\[ c(x) = g(x)i(x) \]  \hspace{1cm} (3.26)
\[ c(x) = (x^4 + a^{13}x^3 + a^6x^2 + a^3x + a^{10}) \]
\[ \cdot (a^{12}x^7 + a^{12}x^6 + a^{12}x^5 + a^{12}x^4 + a^{12}x^3 + a^{12}x^2 + a^{12}x + a^{12}) \]
\[ = a^{12}x^{11} + a^{12}x^{10} + a^{12}x^9 + a^{12}x^8 + a^{12}x^7 + a^{12}x^6 + a^{12}x^5 + a^{10}x^{10} + a^{10}x^9 + a^{10}x^8 + a^{10}x^7 + a^{10}x^6 + a^{10}x^5 + a^{7}x^7 + a^7x^6 + a^7x^5 + a^7x^4 + a^7x^3 + a^7x^2 + a^7x + a^7 \]
\[ = a^{12}x^{11} + (a^{12} + a^{10})x^{10} + (a^{12} + a^{10} + a^3)x^9 + (a^{12} + a^{10} + a^3 + 1)x^8 + (a^{12} + a^{10} + a^3 + 1 + a^7)x^7 + (a^{12} + a^{10} + a^3 + 1 + a^7)x^6 + (a^{12} + a^{10} + a^3 + 1 + a^7)x^5 + (a^{12} + a^{10} + a^3 + 1 + a^7)x^4 + (a^{10} + a^3 + 1 + a^7)x^3 + (a^3 + 1 + a^7)x^2 + (1 + a^7)x + a^7 \]
\[ = a^{12}x^{11} + a^{12}x^7 + a^3x^6 + a^8x^5 + a^3x^4 + a^4x^3 + a^1x^2 + a^8x + a^{11} + a^{14}x^3 + a^2x^2 + a^9x + a^6, \]

Here it becomes clear that when using the non-systematic encoding rule, the original information polynomial \( i(x) \) cannot be directly recognised in \( c(x) \).

We argued above that in the case of systematic RS and BCH codes the BER can be kept lower than that of the non-systematic codes if a small additional signal processing complexity is tolerated. Thus, from now on we concentrate our attention on systematic codes. In order to compute the systematic codeword \( c(x) \), Equations 3.18–3.22 must be used:

\[ c(x) = x^4 \cdot i(x) + p(x), \]

where:

\[ p(x) = \text{Rem} \left\{ \frac{x^4 \cdot i(x)}{g(x)} \right\}. \]

The quotient \( q(x) \) and remainder \( p(x) \) of the above polynomial division are computed in Table 3.4, and the reader may find it beneficial at this stage to work through this example. Although the quotient polynomial is not necessary for our further elaborations, it is delivered by these operations, while the remainder appears in the bottom line of the table, which are given by:

\[ q(x) = a^{12}x^7 + a^3x^6 + a^8x^5 + a^3x^4 + a^4x^3 + a^1x^2 + a^8x + a^{11} \]
\[ p(x) = a^{14}x^3 + a^2x^2 + a^9x + a^6, \]

where:

\[ \frac{x^4i(x)}{g(x)} = q(x)g(x) + p(x) \]
\[
x^4, \tilde{g}(x) = \frac{\alpha_1^2 x^{11} + \alpha_1^2 x^{10} + \alpha_1^2 x^9 + \alpha_1^2 x^8 + \alpha_1^2 x^7 + \alpha_1^2 x^6 + \alpha_1^2 x^5 + \alpha_1^2 x^4}{x^4 + \alpha_1^1 x^3 + \alpha_1^1 x^2 + \alpha_1^1 x + \alpha_1^1}
\]

Table 3.4: Systematic RS(12,8,2) encoding example using polynomial division.
3.3. RS AND BCH CODES

3.3.4 Linear Shift-register Circuits for Cyclic Encoders

3.3.4.1 Polynomial Multiplication

Since RS codes constitute a subclass of cyclic codes, any cyclic encoding circuitry can be used for RS and BCH encoding. The non-systematic RS or BCH encoder can be implemented as a Linear Shift Register (LSR) depicted in Figure 3.1, which multiplies the information polynomial $i(x)$ by the fixed generator polynomial $g(x)$ as follows:

$$c(x) = g(x) \cdot i(x)$$

where:

$$i(x) = i_{k-1}x^{k-1} + i_{k-2}x^{k-2} + \ldots + i_1x + i_0$$

$$g(x) = g_{2t}x^{2t} + g_{2t-1}x^{2t-1} + \ldots + g_1x + g_0.$$  

Figure 3.1: LSR circuit for multiplying polynomials in non-systematic RS and BCH encoders.

The prerequisites for the circuit to carry out proper polynomial multiplications are as follows:

$$g(x) = g_{2t}x^{2t} + g_{2t-1}x^{2t-1} + \ldots + g_1x + g_0.$$  

$$i(x) = i_{k-1}x^{k-1} + i_{k-2}x^{k-2} + \ldots + i_1x + i_0.$$  

Since we now know the parity polynomial $p(x)$, the systematic codeword $c(x)$ is known from Equation 3.18 as well:

$$c(x) = x^4i(x) + p(x)$$

$$c(x) = \alpha^{12}x^{11} + \alpha^{12}x^{10} + \alpha^{12}x^9 + \alpha^{12}x^8 + \alpha^{12}x^7 + \alpha^{12}x^6 + \alpha^{12}x^5 + \alpha^{12}x^4 + \alpha^{12}x^3 + \alpha^{12}x^2 + \alpha^{12}x + \alpha^{12}.$$  

$$(3.30)$$

If there are no transmission errors in the received polynomial $r(x)$, then we have $r(x) = c(x)$, and decoding simply ensues by taking the first $k = 8$ information symbols of $r(x)$, concluding Example 4. Let us continue our discourse by considering an implementation-ally attractive Linear Shift Register (LSR) encoder structure in the next subsection.
CHAPTER 3. BLOCK CODING

42

CHAPTER 3. BLOCK CODING

42

42 CHAPTER 3. BLOCK CODING

T

- 

+ α

3

2

T

- 

+ α

6

2

T

- 

+ α

10

2

T

- 

+ α

13

2

p(x)

i(x)

Figure 3.2: RS(12, 8, 2) systematic encoder using polynomial division.

1) The $LSR$ must be cleared before two polynomials are multiplied.

2) The $k$ symbols of the information polynomial $i(x)$ must be followed by $2t$ zeros.

3) The highest-order $i(x)$ coefficients must be entered into the $LSR$ first.

When the first information symbol $i_{k-1}$ appears at the input of the $LSR$ of Figure 3.1, its output is $i_{k-1} \cdot g_{2t} \cdot x^{(k-1+2t)}$, since there is no contribution from its internal stages because its contents were cleared before entering a new codeword. After one clock pulse, the output is $(i_{k-2} \cdot g_{2t} + i_{k-1} \cdot g_{2t-1})x^{(k-2+2t)}$, and so on. After $(2t + k - 1)$ clock pulses, the $LSR$ contains $0, 0, \ldots, 0, i_0, i_1$, and the output is $(i_0 \cdot g_1 + i_1 \cdot g_0) x$, while the last non-zero output is $i_0 \cdot g_0$. Consequently, the product polynomial at the $LSR$’s output is given by:

$$c(x) = i(x) \cdot g(x) = i_{k-1} \cdot g_{2t} \cdot x^{k-1+2t} + (i_{k-2} \cdot g_{2t} + i_{k-1} \cdot g_{2t-1})x^{k-2+2t} + (i_{k-3} \cdot g_{2t} + i_{k-2} \cdot g_{2t-1} + i_{k-3} \cdot g_{2t-2})x^{k-3+2t} + \ldots + (i_0 \cdot g_1 + i_1 \cdot g_0) x + i_0 \cdot g_0.$$  \hspace{1cm} (3.32)

\[3.3.4.2 \ \text{Systematic Cyclic Shift-register Encoding Example}\]

In this subsection we demonstrate how $LSR$ circuits can be used in order to carry out the polynomial division necessary for computing the parity polynomial $p(x)$ of systematic RS or BCH codes. Let us attempt to highlight the operation of the $LSR$ division circuit by referring to the example computed for the systematic $RS(12, 8)$ encoder previously.

Example 5:

$LSR$ polynomial division example for the $RS(12, 8) GF(16)$ code used for the generation of the parity polynomial $p(x)$ in cyclic systematic encoders.

The corresponding division circuit is depicted in Figure 3.2, where the generator polynomial of Equation 3.25 was used. Note that the highest-order information polynomial coefficients must be entered in the $LSR$ first. Hence, $i(x)$ has to be arranged with the high-order coefficients at its right, ready for entering the $LSR$ of Figure 3.2 from the left.
### Table 3.5: List of LSR internal states for RS(12, 8) systematic encoder.

<table>
<thead>
<tr>
<th>No. of shifts $j$</th>
<th>LSR content after $j$ shifts</th>
<th>Output symbol after $j$ shifts</th>
<th>Feedback symbol</th>
<th>Input symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td>1</td>
<td>$\alpha^{12} + \alpha^2$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td>2</td>
<td>$\alpha^{12} + \alpha^{13}$</td>
<td>$\alpha^{12} + \alpha^{12}$</td>
<td>$0 + 0 = 0$</td>
<td>$0 + 0 = 0$</td>
</tr>
<tr>
<td>3</td>
<td>$\alpha^{12} + \alpha^{13}$</td>
<td>$\alpha^{12} + \alpha^{13}$</td>
<td>$0 + 0 = 0$</td>
<td>$0 + 0 = 0$</td>
</tr>
<tr>
<td>4</td>
<td>$\alpha^{12} + \alpha^{13}$</td>
<td>$\alpha^{12} + \alpha^{12}$</td>
<td>$\alpha^{12} + \alpha^{13}$</td>
<td>$\alpha^{12}$</td>
</tr>
<tr>
<td>5</td>
<td>$\alpha^{12} + \alpha^7 = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^7 = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{10} = \alpha^3$</td>
<td>$\alpha^{12}$</td>
</tr>
<tr>
<td>6</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{10} = \alpha^3$</td>
<td>$\alpha^{12}$</td>
</tr>
<tr>
<td>7</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{10} = \alpha^3$</td>
<td>$\alpha^{12}$</td>
</tr>
<tr>
<td>8</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{10} = \alpha^3$</td>
<td>$\alpha^{12}$</td>
</tr>
<tr>
<td>9</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{10} = \alpha^3$</td>
<td>$\alpha^{12}$</td>
</tr>
<tr>
<td>10</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{10} = \alpha^3$</td>
<td>$\alpha^{12}$</td>
</tr>
<tr>
<td>11</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{10} = \alpha^3$</td>
<td>$\alpha^{12}$</td>
</tr>
<tr>
<td>12</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{13} = \alpha^2$</td>
<td>$\alpha^{12} + \alpha^{10} = \alpha^3$</td>
<td>$\alpha^{12}$</td>
</tr>
</tbody>
</table>
By contrast, in the previously invoked polynomial division algorithm, \( i(x) \) was arranged with the high-order coefficients at the left. In order to demonstrate the operation of the LSR encoder, we listed its contents during each clock cycle in Table 3.5, which mimics the operation carried out during the polynomial division portrayed in Table 3.4. Explicitly, the feedback loop of the schematic in Figure 3.2 carries out the required multiplications by the generator polynomial after each division step, while the modulo-2 additions correspond to subtracting the terms of the same order during the polynomial division, where subtraction and addition are identical in our modulo algebra. Careful inspection of Tables 3.4 and 3.5 reveals the inherent analogy of the associated operations. The quotient polynomial exits the encoder at its output after 12 clock cycles, which is seen in the central column of Table 3.5, while the remainder appears at the bottom left section of the table. Observe in the table that the first \( 2^t = 4 \) shifts of the LSR generate no output and consequently have no equivalent in the polynomial division algorithm. However, the following output symbols are identical to the quotient coefficients of \( q(x) \) in the polynomial division algorithm of Table 3.4, and the LSR content after the 12th clockpulse or shift is exactly the remainder polynomial \( p(x) \) computed by polynomial division. The field additions at the inputs of the LSR cells are equivalent to subtracting the terms of identical order in the polynomial division, which actually represent the updating of the dividend polynomial in the course of the polynomial division after computing a new quotient coefficient.

Following similar generalised arguments, an arbitrary systematic encoder can be implemented by the help of the division circuit of Figure 3.3. As in the case of the multiplier circuit, the LSR must again be cleared initially. Then the output of the LSR is zero for the first \( 2^t \) clockpulses, and the first non-zero output is the first coefficient of the quotient, as we have seen in Table 3.5 for the RS(12, 8, 2) encoder. Again, as we have demonstrated in our systematic RS(12, 8) encoder example, for each quotient coefficient \( q_j \), the product polynomial \( q_j g(x) \) must be subtracted from the dividend polynomial, which is contained in the LSR. This is carried out by arranging for the current quotient coefficient \( q_j \), multiplied by \( g(x) \), to manipulate cells of the LSR by the help of the modulo gates in order to update its contents to hold the current dividend polynomial. After \( n \) clockpulses, the entire quotient has appeared at the LSR’s output, and the current dividend polynomial in the LSR has an order which is lower than that of \( g(x) \). Hence, it cannot be divided by \( g(x) \); it is the remainder of the polynomial division operation. It therefore constitutes the required parity polynomial.

After this short overview of the analogies between GF field operations and their LSR implementations, we proceed with the outline of decoding and error correction algorithms for RS and BCH codes. We emphasise at this point that LSR implementations of GF arithmetics are...
usually computationally quite effective, though conceptually often a bit mysterious. However, one often accepts a conceptually more cumbersome solution for the sake of computational efficiency. Following the above portrayal of RS and BCH encoding techniques, let us now consider the corresponding decoding algorithms.

### 3.3.5 RS Decoding

#### 3.3.5.1 Formulation of the Key Equations

Since RS codes are non-binary BCH codes and BCH codes are cyclic codes, any cyclic decoding technique can be deployed for their decoding. In this section we follow the philosophies of Peterson [82, 84], Clark and Cain [88], Blahut [90], Berlekamp [32], Lin and Costello [96], and a number of other prestigious authors, who have contributed substantially to the better understanding of algebraic decoding techniques and derive a set of non-linear equations to compute the error locations and magnitudes, which can be solved by matrix inversion. This approach was originally suggested for binary BCH codes by Peterson [17], extended for non-binary codes by Gorenstein and Zierler [29], and refined again by Peterson [82, 84].

Let us now attempt to explore what information is given by the parity symbols represented by the coefficients of \( p(x) \) about the location and magnitude of error symbols represented by the error polynomial \( e(x) \), which are corrupting the encoded codeword \( c(x) \) into the received codeword \( r(x) \). Since the encoded codeword is given by:

\[
c(x) = i(x) \cdot x^{n-k} + p(x)
\]

and the error polynomial \( e(x) \) is added component-wise in the GF:

\[
r(x) = c(x) + e(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots + c_1x + c_0 + e_{n-1}x^{n-1} + e_{n-2}x^{n-2} + \ldots + e_1 + e_0.
\] (3.33)

May we remind the reader at this point that an RS\((n, k, t)\) code over \( GF(2^m) \) contains in a codeword \( c(x) \) \( n \) number of \( m \)-ary codeword symbols, constituted by \( k \) information and \( (n-k) = 2t \) parity symbols of the polynomials \( i(x) \) and \( p(x) \), respectively. This code can correct up to \( t \) symbol errors, from which we suppose that \( e(x) \) contains at most \( t \) non-zero coefficients. Otherwise, the error correction algorithm fails to compute the proper error positions and magnitudes. If we knew the error polynomial \( e(x) \), we could remove its effect from the received codeword \( r(x) \) resulting into \( c(x) \), the error-free codeword, and we could recover the information \( i(x) \) sent. Since effectively \( e(x) \) carries merely \( 2t \) unknowns in the form of \( t \) unknown error positions and \( t \) unknown error magnitudes, it would be sufficient to know the value of \( e(x) \) at \( 2t \) field elements to determine the \( 2t \) unknowns, required for error correction.

Recall that the generator polynomial has been defined as:

\[
g(x) = \prod_{j=1}^{2t} (x - \alpha^j),
\]

with zeros at the first \( 2t \) number of GF elements, and:

\[
c(x) = i(x) \cdot g(x).
\]
Therefore, \( c(x) = 0 \) at all zeros of \( g(x) \) in the case of systematic encoders. Hence, if we evaluate the received polynomial \( r(x) \) at the zeros of \( g(x) \), which are the first \( 2t \) \( GF \) elements, we arrive at:

\[
\begin{align*}
    r(x)|_{\alpha^1 \ldots \alpha^{2t}} &= [c(x) + e(x)]|_{\alpha^1 \ldots \alpha^{2t}} \\
    &= [i(x) \cdot g(x) + e(x)]|_{\alpha^1 \ldots \alpha^{2t}} \\
    &= [i(x) \cdot g(x)]|_{\alpha^1 \ldots \alpha^{2t}} + e(x)|_{\alpha^1 \ldots \alpha^{2t}} \\
    &= 0 + e(x)|_{\alpha^1 \ldots \alpha^{2t}}. 
\end{align*}
\]  

(3.34)

Consequently, the received polynomial \( r(x) \) evaluated at the zeros of \( g(x) \), that is at the first \( 2t \) \( GF \) elements, provides us with the \( 2t \) values of the error polynomial \( e(x) \), which are necessary for computation of the \( t \) error positions and \( t \) error magnitudes. These \( 2t \) characteristic values are called the syndromes \( S_1 \ldots S_{2t} \), which can be viewed as the algebraic equivalents of the symptoms of an illness.

If we now assume that actually \( v \leq t \) errors have occurred and the error positions are \( P_1 \ldots P_v \), while the error magnitudes are \( M_1 \ldots M_v \), then an actual transmission error, that is each non-zero component of the error polynomial \( e(x) \), is characterised by the pair of \( GF \) elements \( (P_j, M_j) \). The \( 2t \) equations to be solved for error correction are as follows:

\[
\begin{align*}
S_1 &= r(\alpha) = (e_{n-1} x^{n-1} + e_{n-2} x^{n-2} + \ldots + e_1 x + e_0)|_{x=\alpha} \\
S_2 &= r(\alpha^2) = (e_{n-1} x^{n-1} + e_{n-2} x^{n-2} + \ldots + e_1 x + e_0)|_{x=\alpha^2} \\
& \quad \vdots \\
S_{2t} &= r(\alpha^{2t}) = (e_{n-1} x^{n-1} + e_{n-2} x^{n-2} + \ldots + e_1 x + e_0)|_{x=\alpha^{2t}}. 
\end{align*}
\]  

(3.35)

There are only \( v \leq t \) non-zero error magnitudes \( M_1 \ldots M_v \) with the corresponding positions \( P_1 \ldots P_v \), which can be thought of as place markers for the error magnitudes, and all the positions \( P_1 \ldots P_v \) and magnitudes \( M_1 \ldots M_v \) are \( GF(2^n) \) elements, both of which can be expressed as powers of the primitive element \( \alpha \). Wherever on the right-hand side (RHS) of the first equation in the set of Equations 3.35 there is a non-zero error magnitude \( M_j \neq 0, j = 1 \ldots v, x = \alpha \) is substituted. In the corresponding terms of the second equation, \( x = \alpha^2 \) is substituted, and so forth, while in the last equation \( x = \alpha^{2t} \) is used. Regardless of the actual error positions, the non-zero terms in the syndrome equations \( S_1 \ldots S_{2t} \) are always ordered in corresponding columns above each other with appropriately increasing powers of the original non-zero error positions. This is so whether they are expressed as powers of the primitive element \( \alpha \) or that of the error positions \( P_1 \ldots P_v \). When formulating this equation in terms of the error positions, we arrive at:

\[
\begin{align*}
S_1 &= r(\alpha) = M_1 P_1 + M_2 P_2 + \ldots + M_v P_v = \sum_{i=1}^{v} M_i P_i \\
S_2 &= r(\alpha^2) = M_1 P_1^2 + M_2 P_2^2 + \ldots + M_v P_v^2 = \sum_{i=1}^{v} M_i P_i^2 \\
& \quad \vdots \\
S_{2t} &= r(\alpha^{2t}) = M_1 P_1^{2t} + M_2 P_2^{2t} + \ldots + M_v P_v^{2t} = \sum_{i=1}^{v} M_i P_i^{2t}. 
\end{align*}
\]  

(3.36)
3.3. RS AND BCH CODES

Equation 3.36 can also be conveniently expressed in matrix form, as given below:

\[
\begin{bmatrix}
S_1 \\
S_2 \\
\vdots \\
S_{2t}
\end{bmatrix}
= 
\begin{bmatrix}
P_1 & P_2 & P_3 & \cdots & P_v \\
P_1^2 & P_2^2 & P_3^2 & \cdots & P_v^2 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
P_1^{2t} & P_2^{2t} & P_3^{2t} & \cdots & P_v^{2t}
\end{bmatrix}
\begin{bmatrix}
M_1 \\
M_2 \\
\vdots \\
M_v
\end{bmatrix}
\]

(3.37)

However, this set of equations is non-linear, and hence a direct solution appears to be too complicated, since in general there are many solutions. All the solutions must be found, and the most likely error pattern is the one with the lowest number of errors, which in fact minimises the probability of errors and the BER.

Peterson suggested a simple method [17, 87] for binary BCH codes, which has been generalised by Gorenstein and Zierler [29, 87] for non-binary RS codes. Hence, the corresponding algorithm is referred to as the Peterson–Gorenstein–Zierler (PGZ) decoder.

Their approach was based on the introduction of the error locator polynomial \( L(x) \), which can be computed from the syndromes. The error locator polynomial can then be employed in order to linearise the set of equations, resulting in a more tractable solution for the error locations. Some authors define \( L(x) \) as the polynomial with zeros at the error locations [97].

More frequently, however, [90, 96] it is supposed to have zeros at the multiplicative inverses of the error positions [90, 96], as suggested by the following equation:

\[
L(x) = (1 - x.P_1)(1 - x.P_2)\cdots(1 - x.P_v) = \prod_{j=1}^{v}(1 - x.P_j)
\]

(3.38)

\[
L(x) = L_v.x^v + L_{v-1}.x^{v-1} + \ldots + L_1.x + 1 = \sum_{j=0}^{v} L_j.x^j.
\]

(3.39)

Clearly, \( L(x) = 0 \) for \( x = P_1^{-1}, P_2^{-1}, \ldots, P_v^{-1} \). If \( L(x) \), that is its coefficients or zeros, were known, we would know the error positions; therefore, we try to determine \( L(x) \) from \( S_1 \ldots S_{2t} \) [82,90]. Upon multiplying Equation 3.39 by \( M_i.P_i^{j+v} \), we get:

\[
L(x).M_i.P_i^{j+v} = M_i.P_i^{j+v}.(L_v.x^v + L_{v-1}.x^{v-1} + \ldots + L_1.x + 1)
\]

(3.40)

and on substituting \( x = P_i^{-1} \) into Equation 3.40 we arrive at:

\[
0 = M_i.P_i^{j+v}.(L_v.P_i^{-v} + L_{v-1}.P_i^{-v-1} + \ldots + L_1.P_i^{-1} + 1),
\]

(3.41)

\[
0 = M_i(L_v.P_i^j + L_{v-1}.P_i^{j-1} + \ldots + L_1.P_i^{j+v-1} + P_i^{j+v}).
\]

(3.42)

There exists such an equation for all \( i = 1 \ldots v \) and all \( j \). If we sum the equations for \( i = 1 \ldots v \), for each \( j, j = 1 \ldots 2t \), we get an equation of the form of Equation 3.43:

\[
\sum_{i=1}^{v} M_i(L_v.P_i^j + L_{v-1}.P_i^{j-1} + \ldots + L_1.P_i^{j+v-1} + P_i^{j+v}) = 0.
\]

(3.43)
Equivalently:

$$
\sum_{i=1}^{v} M_i L_i P_{i}^j + \sum_{i=1}^{v} M_i L_{i-1} P_{i-1}^j + \ldots + \sum_{i=1}^{v} M_i P_{i+v}^j = 0. \quad (3.44)
$$

If we compare Equation 3.44 with Equation 3.36, we can recognise the syndromes in the sums. Therefore, we arrive at:

$$
L_v S_j + L_{v-1} S_{j+1} + \ldots + L_1 S_{j+v-1} + S_{j+v} = 0. \quad (3.45)
$$

The highest syndrome index is \( j + v \), but since only the first \( 2 \cdot t \) syndromes \( S_1 \ldots S_{2t} \) are specified, and since \( v \leq t \), the condition \( 1 \leq j \leq t \) must be fulfilled. After rearranging Equation 3.45 we get a set of linear equations for the unknown coefficients \( L_1 \ldots L_v \) as a function of the known syndromes \( S_1 \ldots S_{2t} \), which is in fact the key equation for correcting RS or BCH codes. Any algorithm that delivers a solution to this set of equations can also be employed for correcting errors in RS codes. From Equation 3.45 we can also write:

$$
L_v S_j + L_{v-1} S_{j+1} + \ldots + L_1 S_{j+v-1} = -S_{j+v} \quad \text{for } j = 1 \ldots v. \quad (3.46)
$$

The key equation is more easily understood in a matrix form:

$$
\begin{bmatrix}
S_1 & S_2 & S_3 & \ldots & S_{v-1} & S_v \\
S_2 & S_3 & S_4 & \ldots & S_v & S_{v-1} \\
\vdots & & & & & \\
S_v & S_{v+1} & S_{v+2} & \ldots & S_{2v-2} & S_{2v-1}
\end{bmatrix}
\begin{bmatrix}
L_v \\
L_{v-1} \\
\vdots \\
L_1
\end{bmatrix}
= 
\begin{bmatrix}
-S_{v+1} \\
-S_{v+2} \\
\vdots \\
-S_{2v}
\end{bmatrix} \quad (3.47)
$$

$$
S_i L = S. \quad (3.48)
$$

Equation 3.47 can be solved for the unknown coefficients of the error locator polynomial \( L(x) \) (i.e. for the error positions \( P_1 \ldots P_v \)) if the matrix of syndromes is non-singular. Pless [89] p. 97 showed that the matrix \( S \) has the form of the Vandermonde matrix, which plays a prominent role in the theory of error correction coding. \( S \) can be shown to be non-singular if it is of dimension \( v \times v \), where \( v \) is the actual rather than the maximum number of errors that occurred, while it is singular if the dimension of \( S \) is greater than \( v \) [29, 82]. This theorem provides the basis for determining the actual number of errors \( v \) and determining the error positions \( P_1 \ldots P_v \).

Before we proceed with our discourse on the various solutions to the key equation, we formulate it in different ways, from which the various solutions accrue. Notice that Equation 3.46 can also be interpreted in the form of a set of recursive formulae, which led to Massey’s solution by the synthesis of an autoregressive filter [34]. Namely, if we assume that the coefficients \( L_1 \ldots L_v \) are known for \( j = 1 \ldots v \), Equation 3.46 generates recursively the next syndrome from the previous \( v \) number of syndromes as follows:

$$
\begin{align*}
\begin{array}{rcl}
j = 1 & -S_{v+1} &=& L_v S_1 + L_{v-1} S_2 + L_{v-2} S_3 + \ldots + L_1 S_v \\
j = 2 & -S_{v+2} &=& L_v S_2 + L_{v-1} S_3 + L_{v-2} S_4 + \ldots + L_1 S_{v+1} \\
& \vdots & \vdots \\
j = v & -S_{2v} &=& L_v S_v + L_{v-1} S_{v+1} + L_{v-2} S_{v+2} + \ldots + L_1 S_{2v-1}.
\end{array}
\end{align*}
\quad (3.49)
$$
This set of equations can also be written as:

\[ S_j = - \sum_{n=1}^{v} L_n S_{j-n} \quad \text{for} \quad j = (v+1), (v+2), \ldots, 2v, \tag{3.50} \]

which is reminiscent of a convolutional expression and hence can be implemented by the help of a linear feedback shift register or autoregressive filter having taps \( L_1 \ldots L_v \), and with \( S_j \) fed back into the shift register. Massey’s approach constitutes one of the most computationally effective alternatives to the solution of the key equations, in particular when the codeword length is high and the number of correctible errors is also high. Hence, we will demonstrate how it can be invoked after describing the conceptually simpler PGZ decoding algorithm.

A plethora of other solutions for the key equation in Equation 3.50 are also possible because in this form it is equivalent to the problem found, for example, in spectral estimation in speech coding, when solving a set of recursive equations of the same form for finding the prediction filter’s coefficients. The most powerful solution to the set of equations accrues, if one recognises that the matrix of syndromes in Equation 3.47 can be shown to have both Vandermonde structure, and its values along the main diagonal are identical. In other words, it exhibits a symmetric Toeplitz structure. Efficient special algorithms for the solution of Toeplitz-type matrix equations have been proposed by Levinson, Robinson, Durbin, Berlekamp, Massey, Trench, Burg and Schur in the literature for various prediction and equalisation problems. All these techniques can be invoked for solving the key equations in Equation 3.50 when ensuring that all the operations are carried out over the previously described finite field \( GF(2^m) \). Excellent discourses on the topic have been published in the literature by Makhoul [110], Blahut [111] pp. 352–387 and Schur [112].

Having derived various representations of the key equation for error correction (see Equations 3.46–3.50), it is instructive to continue with the description of the conceptually most simple solution. This was originally proposed by Peterson [17] for binary BCH codes, and it was extended by Gorenstein and Zierler [29] for non-binary RS codes on the basis of inverting the matrix of syndromes.

### 3.3.5.2 Peterson-Gorenstein-Zierler Decoder

As mentioned above, a number of solutions to the key equations have been suggested in [17, 29, 90, 97]. The following set of equations can be derived from Equation 3.48, whose solution is based on inverting the matrix of syndromes, as shown below:

\[ L = S^{-1} \cdot S. \tag{3.51} \]

The solution is based on the following theorem. The Vandermonde matrix \( S \) constituted by the syndromes is non-singular and can be inverted if its dimension is \( v \times v \), but it is singular and cannot be inverted if its dimension is greater than \( v \), where \( v \) is the actual number of errors that occurred.

We have to determine \( v \) to be able to invert \( S \) in order to solve Equation 3.51. Initially, we set \( v = t \), since \( t \) is the maximum possible number of errors, and we compute the determinant of \( S \). If \( \det(S) = 0 \), then \( v = t \), and we can proceed with the matrix inversion. Otherwise, we decrement \( v \) by one and tentatively try \( v = t - 1 \), and so forth, down to \( v = 0 \), until \( \det(S) \neq 0 \) is found. If we have found the specific \( v \) value, for which \( \det(S) \neq 0 \), we compute \( S^{-1} \) by matrix inversion and derive \( L = S^{-1} \cdot S \).

Now the zeros of the error locator polynomial are determined by trial and error. This is carried out by substituting all non-zero field elements into \( L(x) \), and finding those for which we
Figure 3.4: Peterson-Gorenstein-Zierler decoding flowchart.
have $L(x) = 0$. This method is called the Chien search [113]. The error positions $P_1 \ldots P_v$ can then be found by determining the multiplicative inverses of the zeros of $L(x)$. With this step, the solution is complete for binary BCH codes; the bits at the computed error positions must be inverted for error correction.

For non-binary RS codes the error magnitudes $M_1 \ldots M_v$ must be determined in a further step. This is relatively straightforward now from the syndrome Equation 3.37; simply the known matrix of error positions $P$ has to be inverted to compute the vector $M$ of error magnitudes:

$$M = P^{-1} \cdot S.$$  \hspace{1cm} (3.52)

Similarly to the matrix of syndromes $S$, $P$ can also be shown to have Vandermonde structure and consequently is non-singular and can be inverted, if exactly $v$ errors have occurred. Although Equation 3.52 represents a set of $2v$ equations, it is sufficient to solve $v$ of them for the $v$ unknown error magnitudes $M_1 \ldots M_v$. The PGZ method of RS decoding is summarised in the flowchart of Figure 3.4. Observe in the figure that the error correction problem was converted into a problem of inverting two matrices.

Before providing a worked numerical example for PGZ decoding, we briefly allude to the error detection capability of the RS and BCH codes. Explicitly, it is a very attractive property of RS and BCH codes that after error correction the syndromes can be recomputed in order to check whether the code was capable of removing the transmission errors. This is necessary, because without it the received codeword could have been more similar to some other legitimate codeword, and hence the decoder might have concluded that this other, more similar codeword was transmitted. If the error correction action of the decoder was successful, all recomputed syndromes must be zero. If, however, there are errors after the error correction, a systematic decoder can at least separate the original information part of the codeword. This measure allows the decoder to minimize the ‘damage’, which would have been more catastrophic had the decoder attempted to correct the received symbols in the wrong positions, thereby actually corrupting potentially correct symbols.

In order to appreciate the details of the PGZ decoding algorithm, we now offer a worked example using the previously invoked RS(12, 8, 2) code.

### 3.3.5.3 PGZ Decoding Example

**Example 6:**

Consider the double-error correcting RS(12, 8, 2) code over $GF(2^4)$ and carry out error correction by matrix inversion, using the PGZ decoder.

Let us assume that the ‘all-one’ information sequence has been sent in the form of the systematically encoded codeword of Section 3.3.3:

$$i(x) = \alpha^{12}.x^{11} + \alpha^{12}.x^{10} + \alpha^{12}.x^9 + \alpha^{12}.x^8 + \alpha^{12}.x^7 + \alpha^{12}.x^6 + \alpha^{12}.x^5 + \alpha^{12}.x^4 + \alpha^{12}.x^3 + \alpha^{12}.x^2 + \alpha^{12}.x + \alpha^{12}.$$  \hspace{1cm} (3.53)

Let us also assume that two errors have occurred in $c(x)$ in positions 11 and 3, resulting in
the following received polynomial:

\[ r(x) = \alpha^4 x^{11} + \alpha^{12} x^{10} + \alpha^{12} x^9 + \alpha^{12} x^8 \]

\[ + \alpha^{12} x^7 + \alpha^{12} x^6 + \alpha^{12} x^5 + \alpha^{12} x^4 \]

\[ + \alpha^5 x^3 + \alpha^2 x^2 + \alpha^0 x + \alpha^6. \]

Since \( r(x) = c(x) + e(x) \), based on Table 3.3, we infer that \( e(x) \) contains two non-zero terms, in position 11, where the systematic data symbol was \( \alpha^{12} \) and hence the additive error was:

\[ e_{11} = r_{11} - c_{11} = \alpha^4 + \alpha^{12} = \alpha^6 \]

and in position 3, where the parity symbol was \( \alpha^{14} \), requiring an error magnitude of:

\[ e_3 = r_3 - c_3 = \alpha^5 + \alpha^{14} = \alpha^{12}. \]

Hence:

\[ e(x) = \alpha^6 x^{11} + 0 + 0 + 0 + 0 + 0 + 0 + \alpha^{12} x^3 + 0 + 0 + 0. \]

This exponential representation is easily translated into bit patterns by referring to Table 3.2, which contains the various representations of \( GF(2^4) \) elements.

**Syndrome calculation:** Since the syndromes are the received polynomials evaluated at the \( GF(2^4) \) elements \( \alpha^1 \ldots \alpha^{2^4} = \alpha^1 \ldots \alpha^4 \), we have:

\[
S_1 = r(\alpha^1) \\
= \alpha^4 \alpha^{11} + \alpha^{12} \alpha^{10} + \alpha^{12} \alpha^9 + \alpha^{12} \alpha^8 + \alpha^{12} \alpha^7 + \alpha^{12} \alpha^6 \\
+ \alpha^{12} \alpha^5 + \alpha^{12} \alpha^4 + \alpha^5 \alpha^3 + \alpha^2 \alpha^2 + \alpha^0 \alpha^1 + \alpha^6 \\
= \underbrace{\alpha^9 + \alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + \alpha^8 + \alpha^4 + \alpha^1 + \alpha^6}_{\text{Syndrome calculation}} \\
= \underbrace{\alpha^9 + \alpha^{11} + \alpha^0}_{\text{Syndrome calculation}} \\
= \alpha^2 + \alpha^0 \\
S_1 = \alpha^8
\]
\[ S_2 = r(\alpha^2) = \alpha^4 \alpha^{22} + \alpha^{12} \alpha^{20} + \alpha^{12} \alpha^{18} + \alpha^{12} \alpha^{16} + \alpha^{12} \alpha^{14} + \alpha^{12} \alpha^{10} + \alpha^{12} \alpha^8 + \alpha^5 \alpha^6 + \alpha^2 \alpha^4 + \alpha^0 \alpha^2 + \alpha^6 \]

\[ = \alpha^{11} + \alpha^2 + \alpha^0 + \alpha^{13} + \alpha^{11} + \alpha^9 + \alpha^7 + \alpha^5 + \alpha^{11} + \alpha^6 + \alpha^2 + \alpha^6 \]

\[ = \alpha^6 + \alpha^0 + \alpha^3 \]

\[ = \alpha^{13} + \alpha^3 \]

\[ S_2 = \alpha^8 \]

\[ S_3 = r(\alpha^3) = \alpha^4 \alpha^{33} + \alpha^{12} \alpha^{30} + \alpha^{12} \alpha^{27} + \alpha^{12} \alpha^{24} + \alpha^{12} \alpha^{21} + \alpha^{12} \alpha^{18} + \alpha^{12} \alpha^{15} + \alpha^{12} \alpha^{12} + \alpha^5 \alpha^9 + \alpha^2 \alpha^6 + \alpha^0 \alpha^3 + \alpha^6 \]

\[ = \alpha^7 + \alpha^{12} + \alpha^9 + \alpha^6 + \alpha^3 + \alpha^0 + \alpha^{12} + \alpha^9 + \alpha^{14} + \alpha^8 + \alpha^3 + \alpha^6 \]

\[ = \alpha^9 + \alpha^6 \]

\[ S_3 = \alpha^5 \]

\[ S_4 = r(\alpha^4) = \alpha^4 \alpha^{44} + \alpha^{12} \alpha^{40} + \alpha^{12} \alpha^{36} + \alpha^{12} \alpha^{32} + \alpha^{12} \alpha^{28} + \alpha^{12} \alpha^{24} + \alpha^{12} \alpha^{20} + \alpha^{12} \alpha^{16} + \alpha^5 \alpha^{12} + \alpha^2 \alpha^8 + \alpha^0 \alpha^4 + \alpha^6 \]

\[ = \alpha^3 + \alpha^7 + \alpha^3 + \alpha^{14} + \alpha^{10} + \alpha^6 + \alpha^2 + \alpha^{13} + \alpha^2 + \alpha^{10} + \alpha^4 + \alpha^6 \]

\[ = \alpha^1 + \alpha^{11} \]

\[ S_4 = \alpha^6 \]

Since we have no information on the actual number of errors \( v \) in the receiver, we must determine \( v \) first. Let us suppose initially \( v = t = 2 \). The key equation to be solved according to Equation 3.47 is as follows:

\[
\begin{bmatrix}
S_1 & S_2 \\
S_2 & S_3
\end{bmatrix}
\begin{bmatrix}
L_2 \\
L_1
\end{bmatrix}
= -
\begin{bmatrix}
S_3 \\
S_4
\end{bmatrix}.
\]
Let us now compute the determinant of $S$, as portrayed in:

$$\det(S) = \det \begin{vmatrix} S_1 & S_2 \\ S_2 & S_3 \end{vmatrix} = (S_1 S_3 - S_2^2).$$

By substituting the syndromes and using Table 3.3 we arrive at:

$$S_1 = \alpha^8, \ S_2 = \alpha^8, \ S_3 = \alpha^5, \ S_4 = \alpha^6$$

$$\det(S) = (\alpha^8 \cdot \alpha^5 - (\alpha^8)^2) = \alpha^{13} - \alpha^1 = \alpha^{12} \neq 0.$$ 

Since $\det(S) \neq 0$, $v = 2$ errors occurred, and we can compute $S^{-1}$.

Let us use Gauss–Jordan elimination \[114\] for the computation of $S^{-1}$. This can be achieved by transforming both the original matrix $S$, which has to be inverted, and the unit matrix in the same way, until the original $S$ is transformed to a unit matrix. The transformed unit matrix then becomes inverted matrix $S^{-1}$. The operations are naturally $GF(2^4)$ operations, and we commence the transformations with the following matrices:

$$S = \begin{bmatrix} \alpha^8 & \alpha^8 \\ \alpha^8 & \alpha^5 \end{bmatrix}, \quad I = \begin{bmatrix} \alpha^0 & 0 \\ 0 & \alpha^0 \end{bmatrix}.$$ 

After adding row 1 of both matrices to their second rows, we get:

$$\begin{bmatrix} \alpha^8 \\ \alpha^8 + \alpha^8 = 0 \end{bmatrix} = \alpha^8 + \alpha^8 = \alpha^4$$

In a second step we carry out the assignment row 1 = row 1 \cdot \alpha^{11} + row 2, yielding:

$$\begin{bmatrix} \alpha^4 \\ 0 \end{bmatrix} = \alpha^{11} + \alpha^0 = \alpha^{12} \alpha^0$$

Finally, we compute row 1 = row 1 \cdot \alpha^{11} and row 2 = row 2 \cdot \alpha^{11} in order to render $S$ a unit matrix, as seen in:

$$\begin{bmatrix} \alpha^0 \\ 0 \end{bmatrix} = \alpha^8 \alpha^{11}$$

Therefore, the inverted matrix becomes:

$$S^{-1} = \begin{bmatrix} \alpha^8 & \alpha^{11} \\ \alpha^{11} & \alpha^{11} \end{bmatrix}$$

and hence:

$$L = S^{-1} \cdot S$$

which gives:

$$\begin{bmatrix} L_2 \\ L_1 \end{bmatrix} = \begin{bmatrix} \alpha^8 & \alpha^{11} \\ \alpha^{11} & \alpha^{11} \end{bmatrix} \cdot \begin{bmatrix} \alpha^5 \\ \alpha^6 \end{bmatrix} = \begin{bmatrix} \alpha^{13} + \alpha^2 \\ \alpha^1 + \alpha^2 \end{bmatrix} = \begin{bmatrix} \alpha^{14} \end{bmatrix}.$$
Now we can compute the error positions from $L(x)$, where:

$$L(x) = \alpha^{14}x^2 + \alpha^5x + 1,$$

by trying all $GF(2^4)$ elements according to the Chien search:

$$L(\alpha^0) = \alpha^{14}\alpha^0 + \alpha^{5}\alpha^0 + 1 = \alpha^{14} + \alpha^5 + \alpha^0 = \alpha^{11}$$

$$L(\alpha^1) = \alpha^{14}\alpha^2 + \alpha^{5}\alpha^1 + 1 = \alpha^{14} + \alpha^6 + \alpha^0 = \alpha^{11} + \alpha^0 = \alpha^{12}$$

$$L(\alpha^2) = \alpha^{14}\alpha^4 + \alpha^{5}\alpha^2 + 1 = \alpha^{3} + \alpha^7 + \alpha^0 = \alpha^{4} + \alpha^0 = \alpha^{1}$$

$$L(\alpha^3) = \alpha^{14}\alpha^6 + \alpha^{5}\alpha^3 + 1 = \alpha^{5} + \alpha^8 + \alpha^0 = \alpha^4 + \alpha^0 = \alpha^{1}$$

$$L(\alpha^4) = \alpha^{14}\alpha^8 + \alpha^{5}\alpha^4 + 1 = \alpha^{7} + \alpha^9 + \alpha^0 = \alpha^0 + \alpha^0 = 0$$

$$L(\alpha^5) = \alpha^{14}\alpha^{10} + \alpha^{5}\alpha^5 + 1 = \alpha^{9} + \alpha^{10} + \alpha^0 = \alpha^{13} + \alpha^0 = \alpha^{6}$$

$$L(\alpha^6) = \alpha^{14}\alpha^{12} + \alpha^{5}\alpha^6 + 1 = \alpha^{11} + \alpha^{11} + \alpha^0 = 0 + \alpha^0 = \alpha^0$$

$$L(\alpha^7) = \alpha^{14}\alpha^{14} + \alpha^{5}\alpha^7 + 1 = \alpha^{13} + \alpha^{12} + \alpha^0 = \alpha^{1} + \alpha^0 = \alpha^{4}$$

$$L(\alpha^8) = \alpha^{14}\alpha^{16} + \alpha^{5}\alpha^8 + 1 = \alpha^{0} + \alpha^{13} + \alpha^0 = 0 + \alpha^{13} = \alpha^{13}$$

$$L(\alpha^9) = \alpha^{14}\alpha^{18} + \alpha^{5}\alpha^9 + 1 = \alpha^{2} + \alpha^{14} + \alpha^0 = \alpha^{13} + \alpha^0 = \alpha^{6}$$

$$L(\alpha^{10}) = \alpha^{14}\alpha^{20} + \alpha^{5}\alpha^{10} + 1 = \alpha^{4} + \alpha^{9} + \alpha^0 = \alpha^{4} + 0 = \alpha^{4}$$

$$L(\alpha^{11}) = \alpha^{14}\alpha^{22} + \alpha^{5}\alpha^{11} + 1 = \alpha^{6} + \alpha^{1} + \alpha^0 = \alpha^{11} + \alpha^0 = \alpha^{12}$$

$$L(\alpha^{12}) = \alpha^{14}\alpha^{24} + \alpha^{5}\alpha^{12} + 1 = \alpha^{8} + \alpha^{2} + \alpha^0 = \alpha^{0} + \alpha^0 = 0$$

$$L(\alpha^{13}) = \alpha^{14}\alpha^{26} + \alpha^{5}\alpha^{13} + 1 = \alpha^{10} + \alpha^{3} + \alpha^0 = \alpha^{12} + \alpha^0 = \alpha^{11}$$

$$L(\alpha^{14}) = \alpha^{14}\alpha^{28} + \alpha^{5}\alpha^{14} + 1 = \alpha^{12} + \alpha^{4} + \alpha^0 = \alpha^{6} + \alpha^0 = \alpha^{13}.$$

Since the error locator polynomial has its zeros at the inverse error positions, their multiplicative inverses have to be found in order to determine the actual error positions:

$$\begin{align*}
(\alpha^3)^{-1} & = \alpha^{11} = P_1 \\
(\alpha^{12})^{-1} & = \alpha^{3} = P_2,
\end{align*}$$

which are indeed the positions, where the $e(x)$ symbols have been corrupted by $e(x)$.

If we had a binary BCH code, the error correction would simply be the inversion of the bits in positions $P_1 = \alpha^{11}$ and $P_2 = \alpha^{3}$. For the non-binary $Rs(12, 8, 2)$ code over $GF(2^4)$, the error magnitudes still have to be determined. Now we know that there are $\nu = 2$ errors and so the matrix $P$ of error positions in Equation 3.37 can be inverted and
Equation 3.52 can be solved for the error positions \(M_1, \ldots, M_v\). But since \(v = 2\), there are only two equations to be solved for \(M_1\) and \(M_2\), so we simply substitute \(P_1\) and \(P_2\) into Equation 3.36, which gives:

\[
S_1 = M_1 P_1 + M_2 P_2 \\
S_2 = M_1 P_1^2 + M_2 P_2^2 \\
\alpha^8 = M_1 \alpha^{11} + M_2 \alpha^3 \\
\alpha^8 = M_1 \alpha^7 + M_2 \alpha^6.
\]

From the first equation we can express \(M_1\) and substitute it into the second one as follows:

\[
M_1 = \frac{\alpha^8 + M_2 \alpha^3}{\alpha^{11}} = (\alpha^8 + M_2 \alpha^3) \alpha^4 \\
M_1 = \alpha^{12} + M_2 \alpha^7 \\
\alpha^8 = (\alpha^{12} + M_2 \alpha^7) \alpha^7 + M_2 \alpha^6 \\
\alpha^8 = \alpha^4 + M_2 \alpha^{14} + M_2 \alpha^6 \\
\alpha^8 - \alpha^4 = M_2 (\alpha^{14} + \alpha^6)
\]

\[
M_2 = \frac{\alpha^8 + \alpha^4}{\alpha^{14} + \alpha^6} = \frac{\alpha^5}{\alpha^8} = \alpha^5 \alpha^7 = \alpha^{12} \\
M_1 = \alpha^{12} + M_2 \alpha^7 = \alpha^{12} + \alpha^{12} \alpha^7 = \alpha^{12} + \alpha^4 = \alpha^6.
\]

Now we are able to correct the errors by the help of the pairs:

\[
(P_1, M_1) = (\alpha^{11}, \alpha^6) \\
(P_2, M_2) = (\alpha^3, \alpha^{12}),
\]

if we simply add the error polynomial \(e(x) = \alpha^6 x^{11} + \alpha^{12} x^3\) to the received polynomial \(r(x)\) to recover the error-free codeword polynomial \(c(x)\), carrying the original information \(i(x)\).

As we have seen, the PGZ method involves the inversion of two matrices, one to compute the error positions \(P_1, \ldots, P_v\), and one to determine the error magnitudes \(M_1, \ldots, M_v\). For short codes over small GFs with low values of the \(n, k, t\) and \(m\) parameters, the computational complexity is relatively low. This is because the number of multiplications is proportional to \(t^3\). However, for higher \(t\) values the matrix inversions have to be circumvented. The first matrix inversion can, for example, be substituted by the algorithms suggested by Berlekamp [32] and Massey [34, 87], which pursue similar approaches. For the efficient computation of the error magnitudes, Forney has proposed a method [115] that can conveniently substitute the second matrix inversion. Let us now embark on highlighting the principles of these techniques, noting that at this stage we can proceed to the next chapter without jeopardising the seamless flow of thought.

### 3.3.5.4 Berlekamp–Massey Algorithm [32, 34, 82, 88, 90, 96, 97, 108, 109]

In Section 3.3.5.1 we formulated the key equation first in matrix form and showed a solution based on matrix inversion. We also mentioned that by exploiting the Toeplitz structure of the
3.3. RS AND BCH CODES

Figure 3.5: LFSR design for generating the syndromes $S_1 \ldots S_{2t}$ where $S_j = -\sum_{n=1}^{v} L_n \cdot S_{j-n}$ for $j = (v+1), \ldots, 2v$.

syndrome matrix $S$, more efficient methods can be contrived. Instead of pursuing this matrix approach, we now concentrate on the form of the key equation described by Equation 3.50, which suggests the design of an autoregressive filter or Linear Feedback Shift Register (LFSR) in order to generate the required sequence of syndromes $S_1 \ldots S_{2t}$. Berlekamp suggested an efficient iterative heuristic solution to the problem [32], p. 180, which can be better understood in Massey’s original interpretation [34]. However, even Massey’s approach is somewhat abstract, and several authors have attempted to give their own slants on its justification [82, 88, 90, 96, 97]. Some others find it instructive to introduce it after explaining a simpler algorithm, the Euclidean algorithm [116]. In this treatise, we will attempt to make the Berlekamp–Massey (BM) algorithm plausible by the help of a LFSR approach [90, 117], which shows that it is possible to design the taps of a LFSR, constituted in our case by the coefficients $L_0 \ldots L_v$ of the error locator polynomial $L(x)$ so that it produces a required sequence, which is represented by the syndromes $S_1 \ldots S_{2t}$ in our case. In a conventional signal processing context, this would be equivalent to designing the coefficients of a filter such that it would generate a required output sequence. However, in our current context all operations are over the finite GF.

According to this approach, Equation 3.50 can be modelled by the LFSR depicted in Figure 3.5. Clearly, our objective is to determine the length $v$ and the feedback tap values $L_n$ of the LFSR, such that it recursively generates the already known syndromes. Naturally, there is only a limited number of LFSR designs, which produce the required sequence $S_1 \ldots S_{2t}$, and an error polynomial is associated with each possible LFSR design. Note that the order of the error locator polynomial predetermines the number of errors per codeword. The probability of receiving a transmitted codeword with one symbol error is always lower than the joint probability of having a double error within the codeword. In general terms, the most likely error pattern is the one with the lowest number of errors per codeword, which is equivalent to saying that we are looking for the minimum length LFSR design; that is, for the minimum order error locator polynomial $L(x)$.

Since the BM algorithm is iterative, it is instructive to start from iteration one ($i = 1$), where the shortest possible LFSR length to produce the first syndrome $S_1$ is $l^{(i)} = l^{(1)} = 1$ and the corresponding LFSR connection polynomial, that is the error locator polynomial at $i = 1$, is $L^i(x) = L^1(x) = 1$. Let us now assume tentatively that this polynomial also produces the second syndrome $S_2$, and let us attempt to compute the first estimated syndrome $S_{2e}$ produced by this tentative feedback connection polynomial $L^1(x) = 1$ at $i = 1$ by referring to Equation 3.50.
and Figure 3.5:

\[
S_{i+1} = - \sum_{n=1}^{l(i)} L_n^{(i)} S_{i+1-n}.
\]  

(3.53)

Since at the current stage of iteration we have \(l(i) = 1, j = 2, L_1^{(1)} = 1\), the estimated syndrome is given by:

\[
S_{2e} = -1, \quad S_1 = -S_1,
\]

which means that the second estimated syndrome \(S_{2e}\) is actually approximated by the first one. This might be true in some special cases, but not in general, and so-called discrepancy or error term \(d(i)\) is generated in order to check whether the present LFSR design adequately produces the next syndrome. This discrepancy is logically the difference of the required precomputed syndrome \(S_{i+1}\) and its estimate \(S_{(i+1)e}\), which is given by:

\[
d(i) = S_{i+1} - S_{(i+1)e} = S_{i+1} + \sum_{n=1}^{l(i)} L_n^{(i)} S_{i+1-n} = \sum_{n=0}^{l(i)} L_n^{(i)} S_{i+1-n} - \sum_{n=1}^{l(i)-1} L_n^{(i)} S_{i+1-n}.
\]  

(3.54)

Upon exploiting that \(L_0 \neq 1\), for the iteration index \(i = 1\) from Equation 3.54, we have:

\[
d^{(1)} = L_0^{(1)} S_2 + L_1^{(1)} S_1 = S_2 + S_1.
\]

If the current discrepancy or syndrome estimation error is \(d^{(1)} = 0\), then the present LFSR design correctly produces the next syndrome \(S_2\). This means that the internal variables \(l(i), L(i)\), \(x\), \(i\) of the iterative process must remain tentatively unaltered for the duration of the next iteration. In other words, the present LFSR design will be initially employed in the next iteration as a trial design in order to estimate the next precomputed syndrome. Therefore, the following ‘update’ has to take place:

\[
l^{(2)} := l^{(1)} = 1
\]

\[
L^{(2)}(x) := L^{(1)}(x)
\]

\[
i := i + 1 = 2.
\]

The above statements are also valid at other stages of the syndrome iteration and hence in general, if \(d^{(i)} = 0\), then the following assignments become effective:

\[
l^{(i+1)} := l^{(i)}
\]

\[
L^{(i+1)}(x) := L^{(i)}(x)
\]

\[
i := i + 1.
\]

However, if \(d^{(i)} \neq 0\), the LFSR design must be modified until a solution is found, where \(d^{(i)} = 0\). This means that the LFSR must be lengthened by the minimum possible number of stages and another connection polynomial has to be found, which produces the next desired syndrome \(S_{i+1}\) with an estimation error or discrepancy of \(d_i = 0\). Furthermore, all the previous syndromes \(S_1 \ldots S_i\) have to be also properly generated.

One possible way of doing this is to remember the last case at iteration \(m\), when the LFSR has failed to produce the next syndrome \(S_m\), and use the \(m\)th LFSR design with its associated
discrepancy of \( d_m \) in order to modify the present design as follows. The linearity of the LFSR circuitry allows us to invoke superposition, where the non-zero discrepancy of the current LFSR design can be cancelled by appropriately scaling the \( m \)th non-zero discrepancy and superimposing it on the current non-zero estimation error. This can be achieved by actually superimposing two LFSR designs exhibiting non-zero estimation errors. In terms of discrepancies we have:

\[
d(i) - d(i) \cdot \frac{d(m)}{d(m)} = 0,
\]

which suggests a solution for the choice of the proper connection polynomial in an equivalent form:

\[
L^{(i+1)}(x) = L^{(i)}(x) - x^{i-m} \cdot \frac{d(i)}{d(m)} \cdot L^{(m)}(x).
\]

In order to understand Equation 3.56, we emphasise again that two LFSRs can be superimposed to produce the required syndrome with zero discrepancy, since they are linear circuits, where superposition applies. Consequently, we can have a separate auxiliary LFSR of length \( l^{(m)} \) with connection polynomial \( L^{(m)}(x) \) and discrepancy \( d_m \), the coefficients of which are scaled by the factor \( \frac{d(i)}{d(m)} \) in order to compensate properly for the non-zero discrepancy. Note, however, that the auxiliary connection polynomial \( L^{(m)}(x) \) has to be shifted to the required stage of the LFSR. This can be carried out using the multiplicative factor \( x^{(i-m)} \). Having been shifted by \((i-m)\) positions, the output of the appropriately positioned auxiliary connection polynomial \( x^{(i-m)} \cdot L^{(m)}(x) \) is added to that of the LFSR design \( L^{(i)}(x) \), cancelling the undesirable non-zero discrepancy at this iteration.

We note that because of its shift by \( x^{(i-m)} \) the connection polynomial \( L^{(m)}(x) \) actually contributes to \( L^{(i+1)}(x) \) only in coefficients with indices in excess of \((i-m)\), which implies that the first \( m \) number of \( L^{(i)}(x) \) coefficients are not altered by this operation, while those above the index \( m \) must be altered to result in \( d(i) = 0 \). Since we have chosen the most recent \( m \)th iteration, at which \( d(m) \neq 0 \) occurred and consequently LFSR lengthening was necessary, the LFSR length \( l^{(m+1)} \) is higher than \( l^{(m)} \), but only by the minimum required number of stages. Hence, we have a minimum length LFSR design. Because of the linearity of the LFSR circuit, the auxiliary LFSR can be physically merged with the main LFSR. In order to assist in future modifications of the LFSR, when a non-zero discrepancy is produced, the most recent auxiliary LFSR with \( d(m) \neq 0 \) must be stored as well.

Since we have made the inner workings of the BM algorithm plausible, we now attempt to summarise formally the set of iterative steps to be carried out to design the LFSR. The generation of the appropriate LFSR is synonymous with the determination of the minimum length error locator polynomial \( L(x) \), which generates the required precomputed sequence of syndromes. The algorithm had originally been stated in the form of a number of theorems, lemmas and corollaries, for which the rigorous proofs can be found in Massey’s original paper [34], but only its essence is presented here.

**Theorem 1:** An LFSR of length \( l^{(i)} \), which generates the required syndromes \( S_1, S_2, \ldots, S_{i-1} \) and the required syndrome sequence \( S_1, S_2, \ldots, S_i \), does not have to be lengthened; hence, \( l^{(i+1)} = l^{(i)} \) is satisfied. Conversely, the LFSR of length \( l^{(i)} \) that generates \( S_1, S_2, \ldots, S_{i-1} \), but fails to generate the required syndrome sequence \( S_1, S_2, \ldots, S_i \), has to be lengthened. In general, the LFSR length has to obey \( l^{(i+1)} := \text{MAX} \{l^{(i)}, (i + 1 - l^{(i)})\} \). Thus, the LFSR has...
to be lengthened if and only if:

\[ i + 1 - l(i) > l(i) \]  
\[ \text{or } i + 1 > 2l(i), \]  
\[ i.e., \ i \geq 2l(i), \]  

and the increased LFSR length is given by:

\[ l(i+1) = i + 1 - l(i). \]

Following the above introductory elaborations, the BM algorithm can be more formally summarised following Blahut’s interpretation [90] as seen in Figure 3.6. The various stages of processing are numbered in the flowchart, and these steps are listed below.

1 Set the initial conditions:
- Iteration index: \( i = 0 \)
- LFSR length: \( l(1) = 0 \)
- Connection polynomial: \( L(1)(x) = 1 \)
- Auxiliary connection polynomial: \( A(1)(x) = 1 \)

For \( i := 0 \ldots 2t - 1 \) apply the set of recursive relations specified by Steps 2 . . . 11.

2 Check whether we reached the end of iteration, that is whether \( i = 2t \), and if so, branch to Step 13.

3 Compute the discrepancy or estimation error associated with the generation of the next syndrome from Equation 3.54:

\[ d(i) = \sum_{n=0}^{l(i)} L_{H}^{(i)} \cdot S_{i+1-n} \]

and go to Step 4.

4 Check whether the discrepancy is zero, and if \( d(i) = 0 \), then the present LFSR design having a length of \( l(i) \) and connection polynomial \( L(i)(x) \) does produce the next syndrome \( S_{i+1} \). Hence, go to Step 5; otherwise go to Step 6.

5 Simply shift the auxiliary LFSR by one position using the following operation:

\[ A^{(i)}(x) := x \cdot A^{(i)}(x) \]

and go to Step 12.

6 Since \( d(i) \neq 0 \), we correct the temporary connection polynomial \( T^{(i)}(x) \) by adding the properly shifted, normalised and scaled auxiliary connection polynomial \( A^{(i)}(x) \) to the current connection polynomial \( L^{(i)}(x) \), as seen below:

\[ T^{(i)}(x) = L^{(i)}(x) - d^{(i)} \cdot x \cdot A^{(i)}(x). \]

7 Check whether the LFSR has to be lengthened, by using Theorem 1 and Equation 3.59. If \( 2l(i) \leq i \), go to Step 8; otherwise go to Step 9.
Figure 3.6: The BM algorithm: computation of the error locator polynomial $L(x)$. 
8 Update the connection polynomial according to the following formula:

\[ L^{(i)}(x) := T^{(i)}(x) \]  

(3.63)

and go to Step 5.

9 Normalise the most recent connection polynomial \( L(x) \) by dividing it with \( d^{(i)} \neq 0 \) and store the results in the auxiliary LFSR \( A^{(i)}(x) \), as follows:

\[ A^{(i)}(x) := \frac{L^{(i)}(x)}{d^{(i)}} \]  

(3.64)

and go to Step 10.

10 Now one can overwrite \( L^{(i)}(x) \), since it has been normalised and stored in \( A^{(i)}(x) \) during Step 9; hence, update the connection polynomial \( L^{(i)}(x) \) according to:

\[ L^{(i)}(x) := T^{(i)}(x) \]  

(3.65)

and go to Step 11.

11 Since according to Step 7 the LFSR must be lengthened, we update the LFSR length by using Theorem 1 and Equation 3.60, yielding:

\[ i^{(i+1)} = i + 1 - i^{(i)} \]  

(3.66)

and go to Step 12.

12 Increment the iteration index \( i \) and go to Step 2.

13 Invoke the Chien search in order to find the zeros of \( L(x) \) by tentatively substituting all possible GF elements into \( L(x) \) and finding those that render \( L(x) \) zero.

14 Invert the zeros of \( L(x) \) over the given GF, since the zeros of \( L(x) \) are the inverse error locations, and hence find the error positions \( P_1 \ldots P_v \).

Having formalised the BM algorithm, let us now augment our exposition by working through our \( RS(12, 8, 2) \) example assuming the same error pattern, as in our PGZ decoder example.

### 3.3.5.5 Berlekamp–Massey Decoding Example

In order to familiarise ourselves with the somewhat heuristic nature of the BM algorithm, we return to our example in Section 3.3.5.3 and solve the same problem using the BM algorithm instead of the PGZ algorithm.

**Example 7:**

Computation of the error locator polynomial and error positions in the \( RS(12, 8) \) \( GF(2^m) \) code employing the BM algorithm.

The syndromes from Section 3.3.5.3 are as follows:

\[ S_1 = \alpha^8, \quad S_2 = \alpha^8, \quad S_3 = \alpha^5, \quad S_4 = \alpha^6. \]

Let us now follow Steps 1-14 of the formally stated BM algorithm:
3.3. RS AND BCH CODES

1 Initialisation:

\[ i = 0, \quad l = 0, \quad A(x) = 1, \quad L(x) = L_0 = 1. \]

Now we work through the flowchart of Figure 3.6 2t = 4 times:

2 End of iteration? No, because 0 ≠ 4; hence, go to Step 3.

3 Compute the discrepancy:

\[ d = \sum_{n=0}^{l-1} L_n.S_{i+1-n} = L_0.S_1 = 1.\alpha^8 = \alpha^8. \]

4 Check if \( d = 0 \)? Since \( d = \alpha^8 \neq 0 \), a new LFSR design is required; hence, go to Step 6.

6 The new temporary connection polynomial is given by:

\[ T(x) = L(x) - d.x.A(x) = 1 - \alpha^8.x, \]

go to Step 7.

7 Check whether the LFSR has to be lengthened. Since \( 2l = 0 \) and \( i = 0 \), \( 2l \leq i \) is true and the LFSR must be lengthened; hence, go to Step 9.

9 Store the most recent connection polynomial \( L(x) \) after normalising it by its discrepancy \( d \) for later use as auxiliary LFSR \( A(x) \):

\[ A(x) = \frac{L(x)}{d} = \frac{1}{\alpha^8} = \alpha^7 \]

and go to Step 10.

10 Update the connection polynomial \( L(x) \) with the contents of the temporary connection polynomial as follows:

\[ L(x) = T(x) = 1 - \alpha^8.x \]

and go to Step 11.

11 Update the LFSR length according to:

\[ l = i + 1 - l = 0 + 1 - 0 = 1 \]

and go to Step 12.

12 Increment the iteration index by assigning \( i := i + 1 = 0 + 1 = 1 \) and go to Step 2:

\[ i = 1, \quad l = 1, \quad L(x) = 1 - \alpha^8.x, \quad A(x) = \alpha^7. \]

2 Continue iterating, and go to Step 3.

3

\[
\begin{align*}
d &= \sum_{n=0}^{l-1} L_n.S_{i+1-n} = L_0.S_2 + L_1.S_1 \\
&= 1.\alpha^8 + (-\alpha^8).\alpha^8 = \alpha^8 + \alpha^1 = \alpha^{10}. \end{align*}
\]
Since $d \neq 0$, go to Step 6.

6 Compute the temporary corrected LFSR connection polynomial:

$$T(x) = L(x) - d.x.A(x) = 1 - \alpha^8.x - \alpha^{10}.x.\alpha^7$$

$$= 1 + (\alpha^8 + \alpha^2).x = 1 + \alpha^0.x$$

and go to Step 7.

7 Since $2l = 2$ and $i = 1, 2l \leq i$ is false and the LFSR does not have to be lengthened; therefore, go to Step 8.

8 Update the connection polynomial by the temporary connection polynomial:

$$L(x) = T(x) = 1 + \alpha^0.x,$$

and go to Step 5.

5 The contents of the auxiliary LFSR remain unchanged; they must simply be shifted:

$$A(x) = A(x).x = \alpha^7.x.$$

Go to Step 12.

12 $i := i + 1 = 2$ and go to Step 2:

$$i = 2, l = 1, L(x) = 1 + \alpha^0.x, A(x) = \alpha^7.x.$$

2 Carry on iterating, and go to Step 3.

3

$$d = L_0.S_3 + L_1.S_2 = 1.\alpha^5 + \alpha^0.\alpha^8 = \alpha^5 + \alpha^8 = \alpha^4;$$

go to Step 4.

4 Since $d = \alpha^4 \neq 0$, go to Step 6.

6

$$T(x) = L(x) - d.x.A(x) = 1 + \alpha^0.x - \alpha^4.x.\alpha^7.x = 1 + \alpha^0.x + \alpha^{11}.x^2;$$

go to Step 7.

7 Since $2l \leq 2$ is true, because $2 \leq 2$, go to Step 9.

9 The LFSR must be lengthened; hence, we store the most recent normalised connection polynomial into the auxiliary LFSR:

$$A(x) = \frac{L(x)}{d} = \frac{1 + \alpha^0.x}{\alpha^4} = \alpha^{11} + \alpha^{11}.x;$$

go to Step 10.
10 Update the LFSR connection polynomial:

\[ L(x) = T(x) = 1 + \alpha^0.x + \alpha^{11}.x^2; \]

go to Step 11.

11 Update the LFSR length:

\[ l = i + 1 - l = 2 + 1 - 1 = 2; \]

go to Step 12.

12 \( i := i + 1 = 3 \); go to Step 2:

\[ i = 3, \quad l = 2, \quad L(x) = 1 + \alpha^0.x + \alpha^{11}.x^2, \quad A(x) = \alpha^{11} + \alpha^{11}.x. \]

2 Continue iterating, go to Step 3.

3

\[ d = L_0.S_4 + L_1.S_3 + L_2.S_1 = 1.\alpha^6 + \alpha^0.\alpha^5 + \alpha^{11}.\alpha^8 = \alpha^6 + \alpha^5 + \alpha^4 \]

\[ d = \alpha^9 + \alpha^4 = \alpha^{14}; \]

go to Step 4.

4 Since \( d = \alpha^{14} \neq 0 \), go to Step 6.

6 Compute the new temporary connection polynomial:

\[
T(x) = L(x) - d.x.A(x) \\
= 1 + \alpha^0.x + \alpha^{11}.x^2 - \alpha^{14}.x.(\alpha^{11} + \alpha^{11}.x) \\
= 1 + \alpha^0.x + \alpha^{11}.x^2 - \alpha^{10}.x - \alpha^{10}.x^2 \\
= 1 + (\alpha^0 + \alpha^{10}).x + (\alpha^{11} + \alpha^{10}).x^2 \\
= 1 + \alpha^5.x + \alpha^{14}.x^2;
\]

go to Step 7.

7 Because \( 2l < i \) is false, since \( 4 > 3 \), the LFSR does not have to be lengthened; go to Step 8.

8 Simply the temporary connection polynomial is stored in \( L(x) \):

\[ L(x) = T(x) = 1 + \alpha^5.x + \alpha^{14}.x^2; \]

go to Step 5.

5

\[ A(x) = x.A(x) = x.(\alpha^{11} + \alpha^{11}.x) = \alpha^{11}.x + \alpha^{11}.x^2; \]

go to Step 12.

12 \( i := i + 1 = 3 + 1 = 4 \); go to Step 2.
Since \( i = 2t \) is true, we have computed the final error locator polynomial \( L(x) \), which is identical to that computed in Section 3.3.5.3 using the PGZ decoder.

We can therefore use the error positions, determined previously by Chien search. The error positions computed from this second-order connection polynomial \( L(x) \) are: \( P_1 = \alpha^{11} \), \( P_2 = \alpha^3 \). Having determined the error positions, all we have to do for the sake of error correction now is to compute the error magnitudes. This was achieved by the PGZ decoder using matrix inversion, but for large matrices this becomes a computationally demanding operation. Fortunately, it can be circumvented, for example by the Forney algorithm, which will be the subject of the next subsection.

### 3.3.5.6 Computation of the Error Magnitudes by the Forney Algorithm \([32, 82, 88, 90, 97, 108, 109, 115]\)

The Forney algorithm has been described in a number of classic references \([32, 34, 82, 88, 90, 96, 97, 108, 109]\), and here we follow their philosophy. Once the error locator polynomial has been computed to give the error positions \( P_1 \ldots P_v \), we concentrate on the determination of the error magnitudes \( M_1 \ldots M_v \), which can be computed from the so-called error evaluator polynomial \( E(x) \), defined as follows:

\[
E(x) = S(x)L(x),
\]

where \( L(x) \) is the error locator polynomial from Equation 3.68:

\[
L(x) = \prod_{l=1}^{v} (1 - xP_l)
\]

and \( S(x) \) is the so-called syndrome polynomial defined here as:

\[
S(x) = \sum_{j=1}^{2t} S_j x^j.
\]

By substituting the syndromes \( S_1 \ldots S_{2t} \) from Equation 3.36 in the above equation, we arrive at:

\[
S(x) = \sum_{j=1}^{2t} \left( \sum_{i=1}^{v} M_i P_i^j \right) x^j.
\]

The error evaluator polynomial \( E(x) \) depends on both the error positions and the error magnitudes, as opposed to the error locator \( L(x) \), which only depends on the error positions. This fact also reflects the parallelism to the PGZ decoder, where we had two matrix inversions to carry out. The first one has been circumvented by the BM algorithm to determine the error locator polynomial \( L(x) \), that is the error positions. The second matrix inversion is substituted by the Forney algorithm through the computation of the error evaluator polynomial by using the precomputed error positions \( P_1 \ldots P_v \) as well.

Since the decoder is capable of computing only the first \( 2t \) syndromes \( S_1 \ldots S_{2t} \) for a \( t \) error correcting code, the error evaluator polynomial has to be defined in \( \mod x^{2t} \). This is because, given the code construction, there are \( 2t \) parity symbols, allowing the determination of \( t \) error positions and \( t \) error magnitudes. Hence we have:

\[
E(x) = S(x) L(x) \mod x^{2t},
\]
where \( L \) terms of the error positions \( P \), which is a closed-form equation for the computation of the error evaluator polynomial which is the key equation to be solved for the unknown error evaluator polynomial in order

\[ E(x) = \left( \sum_{j=1}^{2^t} \sum_{i=1}^{v} M_i P_i^j x^j \right) \prod_{l=1}^{v} (1 - x P_l) \pmod{m(x^{2^t})}. \]  

(3.72)

By changing the order of summations and rearranging, we get:

\[ E(x) = \left( \sum_{i=1}^{v} M_i \sum_{j=1}^{2^t} P_i^j x^j \right) \prod_{l=1}^{v} (1 - x P_l) \pmod{m(x^{2^t})} \]
\[ = \left( \sum_{i=1}^{v} M_i P_i x \sum_{j=1}^{2^t} (P_i x)^{j-1} \right) (1 - x P_i) \prod_{l \neq i} (1 - x P_l) \pmod{m(x^{2^t})} \]
\[ = x \sum_{i=1}^{v} M_i P_i [(1 - x P_i) \sum_{j=1}^{2^t} (P_i x)^{j-1}] \prod_{l \neq i} (1 - x P_l) \pmod{m(x^{2^t})}. \]  

(3.73)

If we now expand the square bracketed term, we arrive at:

\[ (1 - x P_i) \sum_{j=1}^{2^t} (P_i x)^{j-1} = (1 - x P_i)[1 + P_i x + (P_i x)^2 + \ldots + (P_i x)^{2^t-1}] \]
\[ = 1 + P_i x + (P_i x)^2 + (P_i x)^3 + \ldots + (P_i x)^{2^t-1} \]
\[ = P_i x - (P_i x)^2 - (P_i x)^3 - \ldots - (P_i x)^{2^t-1} - (P_i x)^{2^t} \]
\[ = 1 - (P_i x)^{2^t}. \]  

(3.74)

If we substitute this simplification according to Equation 3.74 into Equation 3.73, we get:

\[ E(x) = x \sum_{i=1}^{v} M_i P_i \left[ 1 - (P_i x)^{2^t} \right] \prod_{l \neq i} (1 - x P_l) \pmod{m(x^{2^t})}. \]  

(3.75)

Since \((P_i x)^{2^t}\) in the square brackets gives zero in \( \pmod{m(x^{2^t})}\), Equation 3.75 yields:

\[ E(x) = x \sum_{i=1}^{v} M_i P_i \prod_{l=1}^{v} (1 - x P_l), \]  

(3.76)

which is a closed-form equation for the computation of the error evaluator polynomial \( E(x) \) in terms of the error positions \( P_i \ldots P_v \) and error magnitudes \( M_1 \ldots M_v \), enabling us to determine the error magnitudes by the help of the Forney algorithm [115].

**Forney algorithm:** If one evaluates the error evaluator polynomial at the inverse error positions \( P_i^{-1} \ldots P_v^{-1} \), the error magnitudes are given by:

\[ M_l = \frac{E(P_i^{-1})}{\prod_{j \neq l}(1 - P_j P_i^{-1})} = -\frac{E(P_i^{-1})}{P_i^{-1} L'(P_i^{-1})}, \]

(3.77)

where \( L' \) stands for the derivative of the error locator polynomial \( L(x) \) with respect to \( x \).
Proof of the Forney algorithm: Let us substitute \( x = P_l^{-1} \) into Equation 3.76, which yields:

\[
E(P_l^{-1}) = \sum_{i=1}^{v} M_i P_i P_l^{-1} \prod_{j=1, j \neq i}^{v} (1 - P_l^{-1} P_j)
\]

\[
= M_1 P_1 P_l^{-1} \prod_{j=1, j \neq 1}^{v} (1 - P_l^{-1} P_j)
\]

\[
+ M_2 P_2 P_l^{-1} \prod_{j=1, j \neq 2}^{v} (1 - P_l^{-1} P_j)
\]

\[
+ \cdots
\]

\[
+ M_v P_v P_l^{-1} \prod_{j=1, j \neq v}^{v} (1 - P_l^{-1} P_j)
\]

(3.78)

Observe in the above equation that all the square bracketed terms contain all but one combination of the factors \((1 - P_l^{-1} P_j)\), including \((1 - P_l^{-1} P_1)\), which is zero. There is, however, one such square bracketed term, where the missing factor happens to be \((1 - P_l^{-1} P_l) = 0\). Taking this into account, we arrive at:

\[
E(P_l^{-1}) = M_l P_l P_l^{-1} \prod_{j=1, j \neq l}^{v} (1 - P_l^{-1} P_j),
\]

(3.80)

or after expressing the error magnitude \( M_l \) from the above equation we get:

\[
M_l = \frac{E(P_l^{-1})}{\prod_{j=1, j \neq l}^{v} (1 - P_l^{-1} P_j)},
\]

(3.81)

which is the first form of the Forney algorithm.

The second form of the Forney algorithm in Equation 3.78 can be proved by computing the derivative \( L'(x) \) of the error locator polynomial \( L(x) \), which ensues as follows:

\[
L(x) = (1 - xP_1)(1 - xP_2) \cdots (1 - xP_v) = \prod_{j=1}^{v} (1 - xP_j)
\]
3.3. RS AND BCH CODES

\[ L'(x) = \frac{d}{dx} L(x) = -P_1(1 - xP_2) \ldots (1 - xP_v) \]
\[ + (1 - xP_1)(1 - xP_3) \ldots (1 - xP_v) \]
\[ + (1 - xP_1)(1 - xP_2) \ldots (1 - xP_v) \]
\[ \vdots \]
\[ + (1 - xP_1)(1 - xP_2) \ldots (1 - xP_{v-1}) \]  
(3.82)

\[ L'(x) = -\sum_{i=1}^{v} P_i \prod_{j=1}^{v} (1 - xP_j). \]  
(3.83)

By evaluating \( L'(x) \) from Equation 3.83 at \( P_{l-1} \) we get:

\[ L'(P_{l-1}) = -\sum_{i=1}^{v} P_i \prod_{j=1 \atop j \neq i}^{v} (1 - P_{l-1}P_j), \]  
(3.84)

where similarly to Equation 3.79 we have only one non-zero term left in the summation. Thus, we arrive at:

\[ L'(P_{l-1}) = -P_l \prod_{j=1 \atop j \neq l}^{v} (1 - P_{l-1}P_j). \]  
(3.85)

Hence, the denominator of Equation 3.78 can be formulated as follows:

\[ \prod_{j=1 \atop j \neq l}^{v} (1 - P_{l-1}P_j) = -P_{l-1} L'(P_{l-1}), \]  
(3.86)

which proves the second form of the algorithm in Equation 3.78.

Let us now compute the error magnitudes in our standard example of the RS(12, 8) code.

3.3.5.7 Forney Algorithm Example

**Example 8:**
Computation of the error evaluator polynomial and error magnitudes in an RS(12, 8) GF(2^4) code using the Forney algorithm.

The error locator polynomial from Section 3.3.5.5 is given by:

\[ S(x) = \sum_{j=1}^{2t} S_j x^j = S_1 x + S_2 x^2 + S_3 x^3 + S_4 x^4 \]
\[ S(x) = \alpha^8 x + \alpha^8 x^2 + \alpha^8 x^3 + \alpha^6 x^4. \]
The error evaluator polynomial \( E(x) = L(x)S(x) \) from the definition of Equation 3.71 is computed as follows:

\[
E(x) = (1 + \alpha^5 x + \alpha^{14} x^2)(\alpha^8 x + \alpha^8 x^2 + \alpha^5 x^3 + \alpha^6 x^4) \pmod{x^4}
\]

\[
= \alpha^8 x + \alpha^8 x^2 + \alpha^5 x^3 + \alpha^6 x^4 + \alpha^{13} x^2 + \alpha^{13} x^3 + \alpha^{10} x^4 + \alpha^{11} x^5 + \alpha^7 x^3 + \alpha^7 x^4 + \alpha^4 x^5 + \alpha^5 x^6
\]

\[
E(x) = \alpha^8 x + (\alpha^8 + \alpha^{13})x^2 + (\alpha^5 + \alpha^{13} + \alpha^7)x^3 + (\alpha^6 + \alpha^{10} + \alpha^7)x^4 + (\alpha^{11} + \alpha^4)x^5 + \alpha^5 x^6
\]

\[
= (\alpha^8 x + \alpha^3 x^2 + 0.x^3 + 0.x^4 + \alpha^{13} x^5 + \alpha^5 x^6) \pmod{x^4}
\]

\[
= \alpha^3 x^2 + \alpha^8 x.
\]

The error magnitudes can be computed from \( E(x) \) by Forney’s algorithm using Equation 3.77:

\[
M_1 = \frac{E(P^{-1}_1)}{\prod_{j \neq l}(1 - P_j P^{-1}_i)}.
\] (3.87)

The error positions are: \( P_1 = \alpha^{11}, P^{-1}_1 = \alpha^4, P_2 = \alpha^3, P^{-1}_2 = \alpha^{12} \); hence, we have:

\[
M_1 = \frac{E(P^{-1}_1)}{(1 - P_2 P^{-1}_1)} = \frac{\alpha^3(\alpha^4)^2 + \alpha^8 \alpha^4}{1 - \alpha^3 \alpha^4} = \frac{\alpha^{11} + \alpha^{12}}{1 - \alpha^7} = \frac{\alpha^0}{\alpha^5} = \alpha^6
\]

\[
M_2 = \frac{E(P^{-1}_2)}{(1 - P_1 P^{-1}_2)} = \frac{\alpha^3(\alpha^{12})^2 + \alpha^8 \alpha^{12}}{1 - \alpha^{13} \alpha^{12}} = \frac{\alpha^{12} + \alpha^5}{1 - \alpha^8} = \frac{\alpha^{14}}{\alpha^7} = \alpha^{12}.
\]

The error magnitudes computed by the Forney algorithm are identical to those computed by the PGZ decoder. Thus, the quantities required for error correction are given by:

\[
(P_1, M_1) = (\alpha^{11}, \alpha^6)
\]

\[
(P_2, M_2) = (\alpha^3, \alpha^{12}).
\]

In order to compute the error-free information symbols, one simply has to add the error magnitudes to the received symbols at the error positions computed, as we have shown in our PGZ decoding example.

Clark and Cain [88] have noted that since \( E(x) = L(x).S(x) \), \( E(x) \) can be computed in the same iterative loop as \( L(x) \), which was portrayed using the BM algorithm in Figure 3.6. In order to achieve this, however, the original BM algorithm’s flowchart in Figure 3.7 has to be properly initialised and slightly modified, as it will be elaborated on below. If we associate a separate normalised LFSR correction term \( C(x) \) with \( E(x) \), which is the counterpart of the auxiliary LFSR connection polynomial \( A(x) \) in the BM algorithm, then the correction formula for \( E(x) \) is identical to that of \( A(x) \) in Equation 3.62, yielding:

\[
E^{(i+1)}(x) = E^{(i)}(x) - d^{(i)}.x.C(x).
\] (3.88)
Then, by using a second temporary storage polynomial $T_2(x)$, Steps 6, 8, 5, 9 and 10 of the flowchart in Figure 3.6 are extended by the corresponding operations for $T_2(x)$, $E(x)$ and $C(x)$, as follows:

**Step 6b:**

$$T_2^{(i)}(x) := E^{(i)}(x) - a^{(i)} . x. C(x)$$  \hspace{1cm} (3.89)

**Step 8b:**

$$E^{(i)}(x) := T_2^{(i)}(x)$$ \hspace{1cm} (3.90)

**Step 5b:**

$$C^{(i)}(x) := x . C^{(i)}(x)$$ \hspace{1cm} (3.91)

**Step 9b:**

$$C^{(i)}(x) := E^{(i)}(x)/d^{(i)}$$ \hspace{1cm} (3.92)

**Step 10b:**

$$E^{(i)}(x) := T_2^{(i)}(x).$$ \hspace{1cm} (3.93)

The complete BM flowchart with these final refinements is depicted in Figure 3.7.

We now round off our discussion of the BM algorithm with the computation of the error evaluator polynomial $E(x)$ for our $RS(12, 8)$ $GF(2^4)$ example using the flowchart of Figure 3.7.

### 3.3.5.8 Error Evaluator Polynomial Computation

**Example 9:**

Computation of the error evaluator polynomial $E(x)$ in an $RS(12, 8)$ $GF(2^4)$ systematic code.

1. $E(x) = 0$, $C(x) = 1$, $l = 0$, $i = 0$; go to Step 2.
2. $0 \neq 4$; go to Step 3.
3. $d = L_0.S_1 = 1.\alpha^8 = \alpha^8$; go to Step 4.
4. $d \neq 0$; go to Step 6.
5. $T_2(x) = E(x) - d . x . C(x) = 0 - \alpha^8 . x . 1 = \alpha^8 . x$; go to Step 7.
6. $0 \leq 0$; go to Step 9.
7. $C(x) = E(x)/d = 0/\alpha^8 = 0$; go to Step 10.
8. $E(x) = T_2(x) = \alpha^8 . x$; go to Step 11.
9. $l = i + 1 - l = 0 + 1 - 0 = 1$; go to Step 12.
10. $i = i + 1 = 0 + 1 = 1$; go to Step 2.
11. $i = 1$, $l = 1$, $E(x) = \alpha^8 . x$, $C(x) = 0$; go to Step 11.
12. $2 \neq 4$; go to Step 3.
13. $d = L_0.S_2 + L_1.S_1 = 1.\alpha^8 + \alpha^8 \alpha^8 = \alpha^8 + \alpha^1 = \alpha^{10}$; go to Step 4.
Figure 3.7: The BM algorithm: computation of both the error locator polynomial $L(x)$ and the error evaluator polynomial $E(x)$. 

$\begin{align*}
L(x) &= 1, \quad A(x) = 1, \quad E(x) = 0, \\
C(x) &= 1, \quad t = 0, \quad i = 0
\end{align*}$
3.3. RS AND BCH CODES

4  

\[ d \neq 0; \text{ go to Step 6.} \]

6b  

\[ T_2(x) = E(x) - d \cdot x \cdot C(x) = \alpha^8 \cdot x - \alpha^{10} \cdot x \cdot 0 = \alpha^8 \cdot x; \text{ go to Step 7.} \]

7  

\[ 2l \leq i \text{ is false; go to Step 8.} \]

8b  

\[ E(x) = T_2(x) = \alpha^8 \cdot x; \text{ go to Step 5.} \]

5b  

\[ C(x) = x \cdot C(x) = 0; \text{ go to Step 12.} \]

12  

\[ i = i + 1 = 2; \text{ go to Step 2.} \]

\[ i = 2, l = 0, E(x) = \alpha^8 \cdot x, C(x) = 0; \]

2  

\[ 2 \neq 4; \text{ go to Step 3.} \]

3  

\[ d = L_0 \cdot S_4 + L_1 \cdot S_2 = 1 \cdot \alpha^5 + \alpha^0 \cdot \alpha^8 = \alpha^5 + \alpha^8 = \alpha^4; \text{ go to Step 4.} \]

4  

\[ d \neq 0; \text{ go to Step 6.} \]

6b  

\[ T_2(x) = E(x) - d \cdot x \cdot C(x) = \alpha^8 \cdot x - \alpha^4 \cdot x \cdot 0 = \alpha^8 \cdot x; \text{ go to Step 7.} \]

7  

\[ 2l \leq i \text{ is true; go to Step 9.} \]

9b  

\[ C(x) = E(x)/d = \alpha^8 \cdot x/\alpha^4 = \alpha^4 \cdot x; \text{ go to Step 10.} \]

10b  

\[ E(x) = T_2(x) = \alpha^8 \cdot x; \text{ go to Step 11.} \]

11  

\[ l = i + 1 - l = 2 + 1 - 1 = 2; \text{ go to Step 12.} \]

12  

\[ i = i + 1 = 2 + 1 = 3; \text{ go to Step 2.} \]

\[ i = 3, l = 2, E(x) = \alpha^8 \cdot x, C(x) = \alpha^4 \cdot x; \]

2  

\[ 3 \neq 4; \text{ go to Step 3.} \]

3  

\[ d = L_0 \cdot S_5 + L_1 \cdot S_3 + L_2 \cdot S_1 = 1 \cdot \alpha^6 + \alpha^0 \cdot \alpha^5 + \alpha^{11} \cdot \alpha^8 = \alpha^6 + \alpha^5 + \alpha^4; \quad d = \alpha^5 + \alpha^4 = \alpha^{14}; \text{ go to Step 4.} \]

4  

\[ d \neq 0; \text{ go to Step 6.} \]

6b  

\[ T_2(x) = E(x) - d \cdot x \cdot C(x) = \alpha^8 \cdot x - \alpha^{14} \cdot x \cdot \alpha^4 \cdot x = \alpha^8 + \alpha^3 \cdot x^2; \text{ go to Step 7.} \]

7  

Since \( 2l \leq i \) is false, go to Step 8.

8b  

\[ E(x) = T_2(x) = \alpha^8 \cdot x + \alpha^3 \cdot x^2; \text{ go to Step 5} \]

5b  

\[ C(x) = x \cdot C(x) = x \cdot \alpha^4 \cdot x = \alpha^4 \cdot x^2; \text{ go to Step 12.} \]

12  

\[ i = i + 1 = 4; \text{ go to Step 2.} \]

2  

Since \( i = 2l \); go to Step 13.

13  

Chien search; go to Step 14.

14  

Find \( P_1, P_2 \) by inverting the zeros of \( L(x) \). Stop.

We have found the error evaluator polynomial \( E(x) = \alpha^8 \cdot x + \alpha^3 \cdot x^2 \), which is the same as that in our previous example, and consequently results in the same error magnitudes. Having considered some of the algorithmic issues of RS and BCH codecs, in closing in the next chapter we provide a range of simulation results for a variety of practical BCH codecs, in order to be able to gauge the BER reductions achieved. A more comprehensive set of results for a range of convolutional, block and concatenated codes can be found, for example, in [109].
3.4 Summary and Conclusions

In this chapter, after a rudimentary introduction to finite fields, we have considered two decoding algorithms for RS and BCH codes. The PGZ decoder, which is based on matrix inversion techniques, and a combination of the BM and Forney algorithm was considered. The latter technique constitutes a computationally more effective iterative solution to the determination of the error positions and error magnitudes. The number of multiplications is proportional to $t^3$ in the case of the matrix inversion method, as opposed to $6t^2$ in the case of the BM algorithm, which means that whenever $t > 6$, the BM algorithm requires less computation than the matrix inversion-based PGZ decoder. Although for the sake of completeness we have provided a rudimentary introduction to RS and BCH codes, as well as to their algebraic decoding, for more detailed discussions on the topic the interested reader is referred to the literature [31, 32, 49, 85, 88, 90, 96].
Part II

Turbo Convolutional and Turbo Block Coding
Part III

Coded Modulation: TCM, TTCM, BICM, BICM-ID
Part IV

Space-Time Block and Space-Time Trellis Coding
Part V

Turbo Equalisation
Bibliography


[319] ETSI, *Digital Cellular Telecommunications System (Phase 2+); High Speed Circuit Switched Data (HSCSD) — Stage 1; (GSM 02.34 Version 5.2.1)*. European Telecommunications Standards Institute, Sophia Antipolis, Cedex, France, July 1997.


About the Authors

Lajos Hanzo (http://www-mobile.ecs.soton.ac.uk) received his degree in electronics in 1976 and his doctorate in 1983. During his 25-year career in telecommunications he has held various research and academic posts in Hungary, Germany and the UK. Since 1986 he has been with the Department of Electronics and Computer Science, University of Southampton, UK, where he holds the chair in telecommunications. He has co-authored eight books on mobile radio communications, published over 300 research papers, organised and chaired conference sessions, presented overview lectures and been awarded a number of distinctions. Currently he is managing an academic research team, working on a range of research projects in the field of wireless multimedia communications sponsored by industry, the Engineering and Physical Sciences Research Council (EPSRC) UK, the European IST Programme and the Mobile Virtual Centre of Excellence (VCE), UK. He is an enthusiastic supporter of industrial and academic liaison and he offers a range of industrial courses. He is also an IEEE Distinguished Lecturer. For further information on research in progress and associated publications please refer to http://www-mobile.ecs.soton.ac.uk

T.H. Liew received the B.Eng degree in Electronics Engineering from the University of Southampton, U.K. In 2001 he was awarded the degree of PhD by the same university and currently he is continuing his research as a postdoctoral research fellow. His current research interests are associated with coding and modulation for wireless channels, space-time coding, adaptive transceivers, etc. He published his research results widely.

Bee Leong Yap graduated in Electronics Engineering from the University of Southampton, UK, with a first class honours degree in 1996. In 2000, he was awarded a PhD and since then he has been continuing his research as a postdoctoral research fellow in Southampton. His research interests include turbo coding, turbo equalisation, adaptive modulation and space-time coding.
Other Related Wiley and IEEE Press Books


2For detailed contents please refer to http://www-mobile.ecs.soton.ac.uk
**Blurb**

For the sake of completeness and wide reader appeal, virtually no prior knowledge is assumed in the field of channel coding. In Chapter 1 we commence our discourse by introducing the family of convolutional codes and the hard- as well as soft-decision Viterbi algorithm in simple conceptual terms with the aid of worked examples.

Chapter 2 provides a rudimentary introduction to the most prominent classes of block codes, namely to Reed-Solomon (RS) and Bose-Chaudhuri-Hocquenghem (BCH) codes. A range of algebraic decoding techniques are reviewed and worked examples are provided.

Chapter 3 elaborates on the trellis-decoding of BCH codes using worked examples and characterises their performance. Furthermore, the classic Chase algorithm is introduced and its performance is investigated.

Chapter 4 introduces the concept of turbo convolutional codes and gives a detailed discourse on the Maximum A Posteriori (MAP) algorithm and its computationally less demanding counter-parts, namely the Log-MAP and Max-Log-MAP algorithms. The Soft Output Viterbi Algorithm (SOVA) is also highlighted and its concept is augmented with the aid of a detailed worked example. Then the effects of the various turbo codec parameters are investigated.

Chapter 5 comparatively studies the trellis structure of convolutional and turbo codes, while Chapter 6 characterises turbo BCH codes. Chapter 7 is a unique portrayal of the novel family of Redundant Residue Number System (RNS) based codes and their turbo decoding. Chapter 8 considers the family of joint coding and modulation based arrangements, which are often referred to as coded modulation schemes. Specifically, Trellis Coded Modulation (TCM), Turbo Trellis Coded Modulation (TTCM), Bit-Interleaved Coded Modulation (BICM) as well as iterative joint decoding and demodulation assisted BICM (BICM-ID) are studied and compared under various narrow-band and wide-band propagation conditions.

In Chapter 9 and 10 space-time block codes and space-time trellis codes are introduced. Their performance is studied comparative in conjunction with a whole host of channel codecs, providing guide-lines for system designers. As a lower-complexity design alternative to multiple-transmitter, multiple-receiver (MIMO) based schemes the concept of near-instantaneously Adaptive Quadrature Amplitude Modulation (AQAM), combined with near-instantaneously adaptive turbo channel coding is introduced in Chapter 11.

Based on the introductory concepts of Chapter 12, Chapter 13 is dedicated to the detailed principles of iterative joint channel equalisation and channel decoding techniques known as turbo equalisation. Chapter 14 provides theoretical performance bounds for turbo equalisers, while Chapter 15 offers a wide-ranging comparative study of various turbo equaliser arrangements. The problem of reduced implementational complexity is addressed in Chapter 16. Finally, turbo equalised space-time trellis codes are the subject of Chapter 17.