
**Information technology — International
symbology specification — Data matrix**

*Technologies de l'information — Spécification internationale des
symboles — Matrice de données*

IECNORM.COM : Click to view the full PDF of ISO/IEC 16022:2000

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

IECNORM.COM : Click to view the full PDF of ISO/IEC 16022:2000

© ISO/IEC 2000

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 734 10 79
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Contents

Introduction	1
1 Scope	1
2 Normative References	1
3 Definitions and Mathematical Symbols	1
3.1 Definitions	1
3.1.1 Alignment Pattern	1
3.1.2 Codeword	1
3.1.3 Module	2
3.1.4 Convolutional Coding	2
3.1.5 Cyclic Redundancy Check (CRC)	2
3.1.6 Extended Channel Interpretation (ECI)	2
3.1.7 Pattern Randomizing	2
3.2 Mathematical Symbols and Operation	2
4 Symbol Description	2
4.1 Basic Characteristics	2
4.2 Summary of Additional Features	3
4.3 Symbol Structure	3
4.3.1 Finder Pattern	4
4.3.2 Symbol Sizes	4
5 ECC 000 - 140 Requirements	4
5.1 Encode Procedure Overview	4
5.2 Data Encodation	5
5.2.1 Base 11 - Numeric Encodation	6
5.2.2 Base 27 - Upper-case Alphabetic Encodation	6
5.2.3 Base 37 - Upper-case Alphanumeric Encodation	6
5.2.4 Base 41 - Upper-case Alphanumeric plus Punctuation Encodation	6
5.2.5 ASCII Encodation	7
5.2.6 8-bit Byte Encodation	7
5.3 User Selection of Error Correction Level	7
5.3.1 Selection of Error Correction Level	7
5.3.2 Additional Error Correction Levels Based on Convolutional Codes	7
5.4 Constructing the Unprotected Bit Stream	7
5.4.1 Format ID Bit Field	7
5.4.2 CRC Bit Field	7
5.4.3 Data Length Bit Field	7
5.4.4 Data Prefix Construction	7
5.4.5 Completing the Unprotected Bit Stream	7
5.5 Constructing the Unrandomized Bit Stream	7
5.5.1 Header Construction	8
5.5.2 Applying Convolutional Coding to Create the Protected Bit Stream	8
5.5.3 Trailer Construction	8
5.5.4 Completing the Unrandomized Bit Stream	8
5.6 Pattern Randomizing	8
5.7 Module Placement in Matrix	8
6 ECC 200 Requirements	8
6.1 Encode Procedure Overview	8
6.2 Data Encodation	9
6.2.1 Overview	9
6.2.2 Default Character Interpretation	9
6.2.3 ASCII Encodation	9
6.2.4 Symbol Control Characters	10
6.2.5 C40 Encodation	11
6.2.6 Text Encodation	12
6.2.7 ANSI X12 Encodation	12
6.2.8 EDIFACT Encodation	13

6.2.9	Base 256 Encodation	14
6.3	User Considerations	14
6.3.1	User Selection of Extended Channel Interpretation	14
6.3.2	User Selection of Symbol Size and Shape	14
6.4	Extended Channel Interpretation	14
6.4.1	Encoding ECIs	15
6.4.2	ECIs and Structured Append	15
6.4.3	Post-Decode Protocol	17
6.5	ECC 200 Symbol Attributes	17
6.5.1	Symbol Sizes and Capacity	17
6.5.2	Insertion of Alignment Patterns into Larger Symbols	17
6.6	Structured Append	17
6.6.1	Basic Principles	17
6.6.2	Symbol Sequence Indicator	17
6.6.3	File Identification	17
6.6.4	FNC1 and Structured Append	17
6.6.5	Buffered and Unbuffered Operation	18
6.7	Error Detection and Correction	18
6.7.1	Generating the Error Correction Codewords	18
6.7.2	Error Correction Capacity	18
6.8	Symbol Construction	19
6.8.1	Symbol Character Placement	19
6.8.2	Alignment Pattern Module Placement	19
6.8.3	Finder Pattern Module Placement	19
7	Symbol Dimensions	20
7.1	Dimensions	20
7.2	Quiet Zone	20
8	Symbol Quality	20
8.1	Obtaining the Test Image	20
8.2	Symbol Quality Parameters	20
8.2.1	Decode	20
8.2.2	Symbol Contrast	20
8.2.3	“Print” Growth	20
8.2.4	Axial Nonuniformity	20
8.2.5	Unused Error Correction	20
8.3	Overall Symbol Grade	21
8.4	Process Control Measurements	21
9	Reference Decode Algorithm for Data Matrix	21
10	User Guidelines	26
10.1	Human Readable Interpretation	26
10.2	Autodiscrimination Capability	26
10.3	System Consideration	27
11	Transmitted Data	27
11.1	Protocol for FNC1 (ECC 200 Only)	27
11.2	Protocol for FNC1 in the Second Position	27
11.3	Protocol for Macro Characters in the First Position (ECC 200 only)	27
11.4	Protocol for ECIs (ECC 200 Only)	27
11.5	Symbology Identifier	28
11.6	Transmitted Data Example	28
Annexe A (Normative)		29
	ECC 000 - 140 Symbol Attributes	29
A.1	ECC 000	29
A.2	ECC 050	29
A.3	ECC 080	30
A.4	ECC 100	30
A.5	ECC 140	31
Annexe B (Normative)		32

ECC 000 - 140 Data Module Placement Grids	32
Annexe C (Normative)	41
ECC 000 - 140 Character Encodation Schemes	41
C.1 Base 11 Encodation Scheme	44
C.1.1 First Stage Procedure	44
C.1.2 Second Stage Procedure	44
C.1.3 Example	44
C.2 Base 27 Encodation Scheme	45
C.2.1 First Stage Procedure	45
C.2.2 Second Stage Procedure	45
C.2.3 Example	45
C.3 Base 37 Encodation Scheme	46
C.3.1 First Stage Procedure	46
C.3.2 Second Stage Procedure	46
C.3.3 Example	46
C.4 Base 41 Encodation Scheme	47
C.4.1 First Stage Procedure	47
C.4.2 Second Stage Procedure	47
C.4.3 Example	47
Annexe D (Normative)	48
ECC 000 - 140 CRC Algorithm	48
D.1 CRC State Machine	48
D.2 CRC Polynomial	48
D.3 CRC 2-Byte Header	48
Annexe E (Normative)	49
ECC 000 - 140 Error Checking and Correcting Algorithms	49
E.1 ECC 000	49
E.2 ECC 050	49
E.3 ECC 080	49
E.4 ECC 100	49
E.5 ECC 140	49
E.6 Processing the Convolutional Code	49
E.7 Convolutional Codes Reference Decode Algorithm	49
Annexe F (Normative)	55
ECC 000 - 140 Master Random Bit Stream	55
Annexe G (Normative)	56
ECC 200 Interleaving Process	56
G.1 Schematic Illustration	56
G.2 Starting Sequence for Interleaving in Different Sized Symbols	57
Annexe H (Normative)	59
ECC 200 Pattern Randomizing	59
H.1 253-State Algorithm	59
H.1.1 253-State Randomizing Algorithm	59
H.1.2 253-State Un-Randomizing Algorithm	59
H.2 255-State Algorithm	59
H.2.1 255-State Randomizing Algorithm	59
H.2.2 255-State Un-Randomizing Algorithm	59
Annexe J (Normative)	60
ECC 200 Encodation Character Sets	60
J.1 C40 Encodation Character Set	60
J.2 Text Encodation Character Set	61
J.3 EDIFACT Encodation Character Set	62
Annexe K (Normative)	63
ECC 200 Alignment Patterns	63
Annexe L (Normative)	65
ECC 200 Reed-Solomon Error Detection and Correction	65
L.1 Error Correction Codeword Generator Polynomials	65

L.2 Error Correction Calculation	66
Annexe M (Normative)	68
ECC 200 Symbol Character Placement	68
M.1 Symbol Character Placement Program	68
M.2 Symbol Character Placement Rules	71
M.2.1 Non-standard Symbol Character Shapes	71
M.2.2 Symbol Character Arrangement	71
M.3 Symbol Character Placement Examples for ECC 200	75
Annexe N (Normative)	82
2D Matrix Bar Code Print Quality - Guideline	82
N.1 Obtaining the Test Image	82
N.2 Assessing Symbol Parameters	82
N.2.1 Decode	82
N.2.2 Symbol Contrast	82
N.2.3 "Print" Growth	83
N.2.4 Axial Nonuniformity	83
N.2.5 Unused Error Correction	83
N.3 Overall Symbol Grade	84
Annexe P (Normative)	84
Symbology Identifier	84
Annexe Q (Informative)	84
ECC 000-140 Encode Example Using ECC 050	84
Q.1 CRC Calculation for Example	89
Annexe R (Informative)	90
ECC 200 Encode Example	90
Annexe S (Informative)	91
Encoding Data Using the Minimum Symbol Data Characters for ECC 200	91
Annexe T (Informative)	92
Useful Process Control Techniques	92
T.1 Symbol Contrast	93
T.2 Special Reference Symbol	93
T.3 Assessing Axial Nonuniformity	93
T.4 Visual Inspection for Symbol Distortion and Defects	93
Annexe U (Informative)	94
Autodiscrimination Capabilities	94
Annexe V (Informative)	94
System Considerations	94

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 16022 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 31, *Automatic identification and data capture techniques*.

International Standard ISO/IEC 16022 was prepared by AIM International (as ANSI/AIM BC11) and was adopted, under a special "fast-track procedure", by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

Annexes A to P form a normative part of this International Standard. Annexes Q to V are for information only.

Information technology — International symbology specification — Data matrix

Introduction

Data Matrix is a two-dimensional matrix symbology which is made up of square modules arranged within a perimeter finder pattern. Though primarily shown and described in this document as a dark symbol on light background, Data Matrix symbols can also be printed to appear as light on dark.

Manufacturers of bar code equipment and users of the technology require publicly available standard symbology specifications to which they can refer when developing equipment and application standards. The publication of Symbology Specifications is designed to achieve this.

1 Scope

This specification defines the requirements for the symbology known as Data Matrix. It specifies the Data Matrix symbology characteristics, data character encodation, symbol formats, dimensions and print quality requirements, error correction rules, decoding algorithm, and user-selectable application parameters.

2 Normative References

This specification incorporates provisions from other publications. These normative references are cited at the appropriate places in the text and the publications are listed below. The latest edition of the publication referred to applies.

EN796	Bar Coding : Symbology Identifiers
EN1556	Bar Coding : Terminology
ANSI X3.182	Bar Code Print Quality - Guideline (Same as EN1635 - Bar Coding : Test Specifications for Bar Code Symbols)
ANSI X3.4	Coded Character Sets - 7-bit American National Standard Code for Information Interchange (7-bit ASCII) (equivalent to the US national version of ISO 646)

ISO/IEC
8859-1 Information Processing - 8-bit
Single-byte Coded Graphic
Character Sets - Part 1 (Latin
Alphabet Number 1)

ECI Assignments Document- AIM International

3 Definitions and Mathematical Symbols

3.1 Definitions

For the purposes of this specification, the following definitions in EN 1556 shall apply:

algorithm, application standard, ASCII, autodiscrimination, binary, bit, CCD, code page, code set, data character, data codeword, data region, data separator character, decode algorithm, decoder, error correction, finder pattern, human readable character, latch character, leading zeros, matrix symbology, modulo, numeric, omnidirectional, orientation pattern, overhead, pad character, pixel, quiet zone, reference decode algorithm, Reed-Solomon error correction, scanner, shift characters, structured append, symbol character, symbology, symbology identifier, X-dimension

The following definitions also apply to this specification. Although some of the terms below are defined in EN1556, the definitions which follow below are more appropriate for this specification.

3.1.1 Alignment Pattern

A unique pattern in larger ECC 200 Data Matrix symbols, made up of a solid line of contiguous dark cells abutting a line of alternating dark and light cells. The alignment patterns run horizontally and vertically within the symbols.

3.1.2 Codeword

A symbol character value. An intermediate level of coding between source data and the graphical encodation in the symbol.

3.1.3 Module

A single cell in a matrix symbology used to encode one bit of data. In Data Matrix the module is a square shape.

3.1.4 Convolutional Coding

An Error Checking and Correcting (ECC) algorithm that processes a set of input bits into a set of output bits that can recover from damage. The encoding process consists of breaking the input bits into blocks, then convolving each input block with the contents of a multi-stage shift register to produce protected output blocks. These encoders can be constructed in hardware using input and output switches, shift registers, and exclusive-or (XOR) gates.

3.1.5 Cyclic Redundancy Check (CRC)

An error detection algorithm which performs binary modulo 2 long division of a binary input stream by a fixed binary bit stream and produces as output the remainder of this division. The fixed binary bit stream is usually represented by a polynomial. The remainder of the division is the CRC value. The division can be implemented in hardware by shift registers and XOR gates. The purpose of a CRC is to provide error detection allowing validation of the user data after correction by the ECC process.

3.1.6 Extended Channel Interpretation (ECI)

A protocol used by some symbologies that allows the output data stream to have interpretations other than that of the default character set.

3.1.7 Pattern Randomizing

A procedure which converts an original bit pattern to another bit pattern by inverting selected bits. The resulting bit stream is less likely to have repeating patterns.

3.2 Mathematical Symbols and Operation

For the purposes of this specification, the mathematical symbols which follow shall apply globally unless defined locally:

- d number of error correction codewords
- e number of erasures
- k (for ECC 000 - 140) the number of bits in a complete segment input to the state machine to

generate the convolutional code

(for ECC 200) total number of error correction codewords

m the memory order of the convolutional code

n (for ECC 000 - 140) the number of bits in a complete segment generated by the state machine producing the convolutional code (for ECC 200) total number of data codewords

N the numerical base in an encoding scheme

p number of misdecode protection codewords

S symbol character

t number of errors

u the input bit segment to the state machine, taken k bits at a time

v the output bit segment from the state machine, generated n bits at a time

X horizontal and vertical width of a module

ε error correction codeword

For the purposes of this specification, the notations and mathematical operations which follow shall apply:

- div is the integer division operator
- mod is the integer remainder after division
- XOR is the exclusive-or logic function whose output is one only when its two inputs are not equivalent.
- LSB Least significant bit
- MSB Most significant bit

4 Symbol Description

4.1 Basic Characteristics

Data Matrix is a two-dimensional matrix symbology. There are two types: ECC 000 - 140 with several available levels of convolutional error correction and ECC 200 which uses Reed-Solomon error correction. For new applications ECC 200 is recommended. ECC 000 - 140 should only be used in closed applications where a single party controls both the production and reading of the symbols and is responsible for overall system performance. The characteristics of Data Matrix are:

a. Encodable character set:

1. values 0 - 127 in accordance with ANSI X3.4, i.e. all 128 ASCII characters

- (equivalent to the U.S. national version of ISO 646)
2. values 128 - 255 in accordance with ISO 8859-1; Latin Alphabet No. 1. These are referred to as extended ASCII
- b. Representation of data: A dark module is a binary one and a light module is a zero.
 - c. Symbol size in modules (not including quiet zone):

<u>ECC 000 - 140</u>	<u>ECC 200</u>
9 by 9	10 by 10
to 49 by 49	to 144 by 144
Odd Only	Even Only
 - d. Data characters per symbol (for maximum symbol size in ECC200):

1. Alphanumeric data:	up to 2335 characters
2. 8-bit byte data:	1556 characters
3. Numeric data:	3116 digits
 - e. Selectable error correction:

ECC 000 - 140:	Four levels of convolutional error correction, plus the option to apply only error detection
ECC 200:	Reed-Solomon error correction
 - f. Code type: Matrix
 - g. Orientation independence: Yes

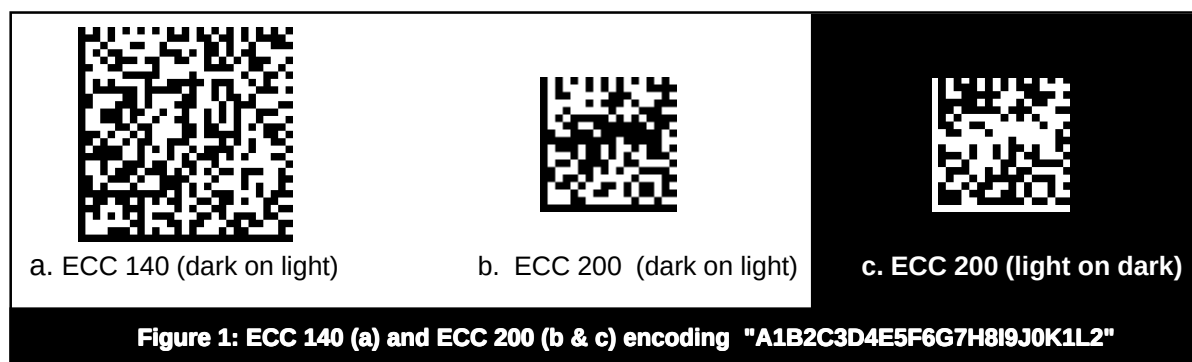
4.2 Summary of Additional Features

The following summarises additional features which are inherent or optional in Data Matrix:

- a. Reflectance reversal: (Inherent) Symbols are intended to be read when marked so that the image is either dark on light or light on dark (see Figure 1).
- b. Extended Channel Interpretations: (ECC 200 only, optional) This mechanism enables characters from other character sets (e.g. Arabic, Cyrillic, Greek, Hebrew) and other data interpretations or industry-specific requirements to be represented.
- c. Rectangular symbols: (ECC 200 only, optional) Six symbol formats are specified in a rectangular form.
- d. Structured append: (ECC 200 only, optional) This allows files of data to be represented in up to 16 Data Matrix symbols. The original data can be correctly reconstructed regardless of the order in which the symbols are scanned.

4.3 Symbol Structure

Each Data Matrix symbol consists of data regions which contain nominally square modules set out in a regular array. In larger ECC 200 symbols, data regions are separated by alignment patterns. The data region is surrounded by a finder pattern, and this shall be surrounded on all four sides by a quiet zone border. Figure 1 illustrates an ECC 140 and two representations of an ECC 200 symbol.



4.3.1 Finder Pattern

The finder pattern is a perimeter to the data region and is one module wide. Two adjacent sides, the left and lower sides, forming the L boundary, are solid dark lines; these are used primarily to determine physical size, orientation and symbol distortion. The two opposite sides are made up of alternating dark and light modules. These are used primarily to define the cell structure of the symbol, but also can assist in determining physical size and distortion.

4.3.2 Symbol Sizes and Capacities

ECC 000 - 140 symbols have an odd number of rows and an odd number of columns. Symbols are square with sizes from 9 x 9 to 49 x 49 (modules) not including quiet zones. These symbols can be recognized by the upper right corner module being dark. The complete attributes of ECC 000 - 140 symbols are given in Annexe A.

ECC 200 symbols have an even number of rows and an even number of columns. Some symbols are square with sizes from 10 x 10 to 144 x 144 not including quiet zone. Some symbols are rectangular with sizes from 8 x 18 to 16 x 48 not including quiet zone. All ECC 200 symbols can be recognized by the upper right corner module being light. The complete attributes of ECC 200 symbols are given in Table 11 in Section 6.5.

5 ECC 000 - 140 Requirements

For new applications or open systems ECC 200 is recommended (See Section 6).

5.1 Encode Procedure Overview

This section provides an overview of the encoding procedure. Following sections will provide more details. An example encode for ECC 050 is given in

Annexe Q. The following steps convert user data to an ECC 000 - 140 symbol:

Step 1: Data Encodation

The user data is analyzed to identify the variety of different characters to be encoded. For maximum compaction efficiency, the lowest level encodation scheme capable of encoding the data should be selected. If the user does not specify the matrix size, then choose the smallest size that accommodates the data. The result of this step is called the Encoded Data Bit Stream.

Step 2: Data Prefix Construction

A Data Prefix Bit Stream is constructed from the Format ID, CRC, and Data Length bit fields. This Data Prefix Bit Stream is prefixed to the Encoded Data Bit Stream to produce the Unprotected Bit Stream.

Step 3: Error Checking and Correcting

The Unprotected Bit Stream is processed by the user specified convolutional coding encode algorithm to produce the Protected Bit Stream. This step is omitted for ECC 000.

Step 4: Header and Trailer Construction

A header containing only the ECC bit field is prefixed to the Protected Bit Stream. A trailer containing pad bits (zeros) is appended to the Protected Bit Stream. The Protected Bit Stream with the header and trailer added is called the Unrandomized Bit Stream.

Step 5: Pattern Randomizing

The Unrandomized Bit Stream is processed by the pattern randomizing algorithm and produces the Randomized Bit Stream.

Encodation Scheme	Characters	Bits per Data Character
Base 11	Numeric data	3.5
Base 27	Upper-case alphabetic	4.8
Base 37	Upper-case alphanumeric	5.25
Base 41	Upper-case alphanumeric and punctuation	5.5
ASCII	Full 128 ASCII set	7
8-bit Byte	User defined	8

Table 1: Encodation Schemes

Step 6: Module Placement in Matrix

Modules are placed in a matrix to construct the finder pattern. The Randomized Bit Stream is placed into the matrix one module at a time according to the data module placement algorithm given in Annexe B.

Figure 2 shows the various bit streams during the encode process.

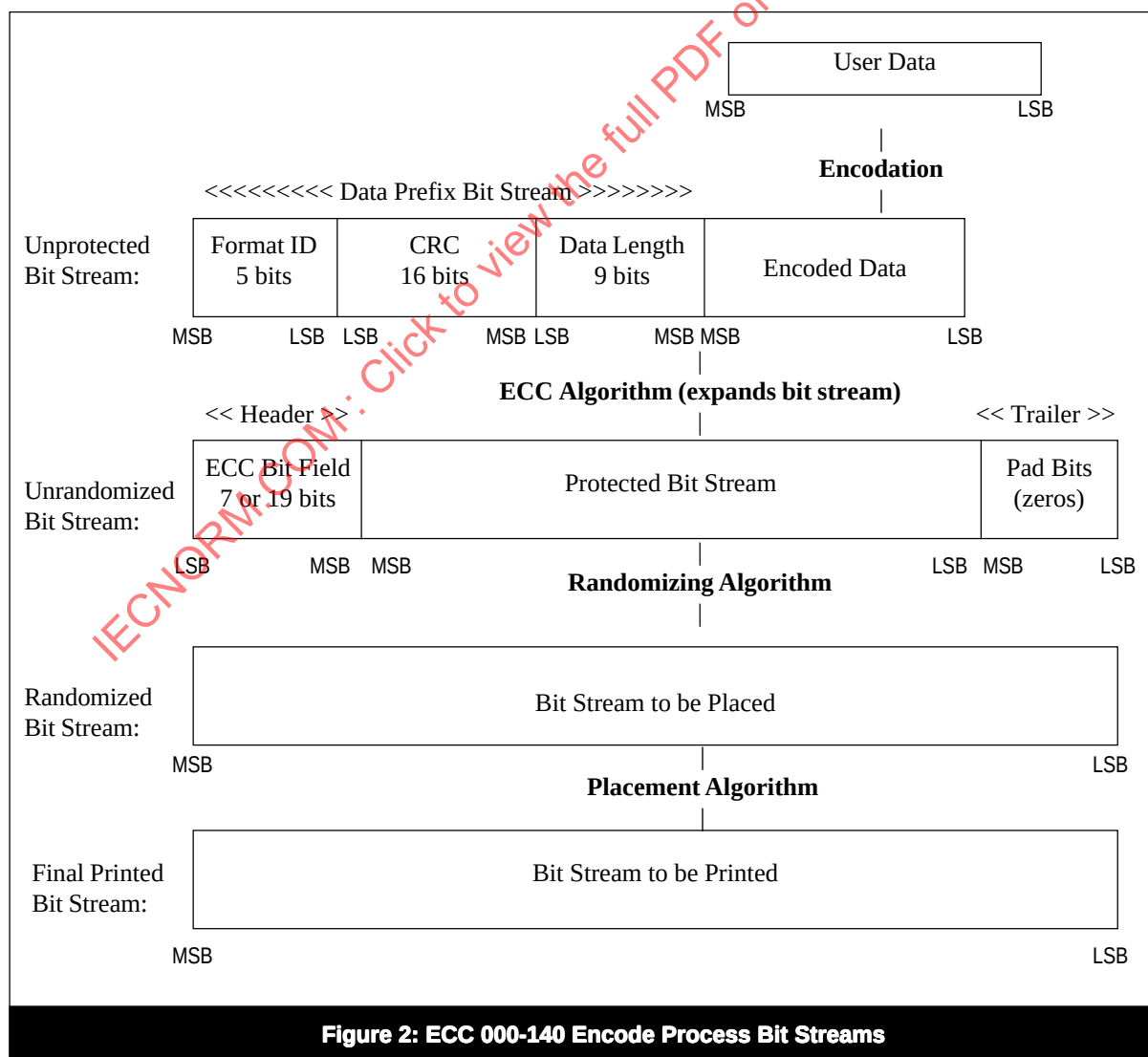
5.2 Data Encodation

The data shall be encoded using one of six encodation schemes (see Table 1).

The encodation scheme is fixed for the entire symbol, and thus the selection of the most appropriate encodation scheme can have a considerable effect on the number of bits required to encode any given data.

The same data may be represented in ECC 000 - 140 symbols in different ways through the use of the different encodation schemes. The character sets of all the encodation schemes, except the 8-bit byte scheme, are given in Annexe C. The 8-bit byte scheme is user definable.

The most efficient scheme to use is the lowest base number scheme which is capable of encoding all the characters in the message. Thus if all the characters can be encoded in Base 27, it is not efficient to use Base 37, Base 41 or ASCII.



To determine the appropriate encodation scheme, the data to be encoded should be analyzed. The character sets of each of the Base N encodation schemes should be compared with the data character set to be encoded starting with the Base 11 character set. If this is suitable then it should be used, if not the comparisons should continue with Base 27, Base 37 and Base 41 until the appropriate lowest level encodation scheme is found. If data characters beyond the capability of Base 41 need to be encoded, the ASCII set should be used, unless characters are beyond this; in which case the 8-bit byte set should be used.

For all encodation schemes, each compressed sequence of 4-24 bits is placed into the Encoded Bit Stream in reverse order (LSB first). This means that each individual compressed sequence is composed, then reversed, and output immediately to the Encoded Bit Stream. This does not mean that a complete compressed bit stream is formed, then reversed.

The details of each encodation scheme are given in the following clauses.

5.2.1 Base 11 - Numeric Encodation

The Base 11 (Numeric) Encodation scheme encodes 6 data characters as 21 bits, achieving an encodation of 3.5 bits per data character. The Base 11 code set enables the following 11 characters to be encoded:

0 to 9
space

The data is encoded in two stages. In the first stage, the actual data characters shall be replaced by their Base 11 code values, as given in Annexe C.

In the second phase, the Base 11 code values shall be compacted using a Base 11 to Base 2 conversion according to the procedures defined in Annexe C.1.

5.2.2 Base 27 - Upper-case Alphabetic Encodation

The Base 27 (Upper-case Alphabetic) Encodation scheme encodes 5 data characters as 24 bits, achieving an encodation of 4.8 bits per data character. The Base 27 code set enables the following 27 characters to be encoded:

A to Z
space

The data is encoded in two stages. In the first stage, the actual data characters shall be replaced by their

Base 27 code values, as given in Annexe C.

In the second stage, the Base 27 code values shall be compacted using a Base 27 to Base 2 conversion according to the procedures defined in Annexe C.2.

5.2.3 Base 37 - Upper-case Alphanumeric Encodation

The Base 37 (Upper-case Alphanumeric) Encodation scheme encodes 4 data characters as 21 bits, achieving an encodation of 5.25 bits per data character. The Base 37 code set enables the following 37 characters to be encoded:

A to Z
0 to 9
space

The data is encoded in two stages. In the first stage, the actual data characters shall be replaced by their Base 37 code values, as given in Annexe C.

In the second stage, the Base 37 code values shall be compacted using a Base 37 to Base 2 conversion according to the procedures defined in Annexe C.3.

5.2.4 Base 41 - Upper-case Alphanumeric plus Punctuation Encodation

The Base 41 (Upper-case Alphanumeric plus Punctuation) Encodation scheme encodes 4 data characters as 22 bits, achieving an encodation of 5.5 bits per data character. The Base 41 code set enables the following 41 characters to be encoded:

A to Z
0 to 9
space
, (period)
, (comma)
- (minus or hyphen)
/ (forward slash or solidus)

The data is encoded in two stages. In the first stage, the actual data characters shall be replaced by their Base 41 code values, as given in Annexe C.

In the second stage, the Base 41 code values shall be compacted using a Base 41 to Base 2 conversion according to the procedures defined in Annexe C.4.

5.2.5 ASCII Encodation

The ASCII Encodation scheme enables all 128 characters from ANSI X3.4 to be encoded. Each data character shall be encoded as a 7-bit byte equivalent to the decimal value shown in the ASCII column of Table C1 of Annexe C.

5.2.6 8-bit Byte Encodation

The 8-bit Byte Encodation scheme shall be used for closed applications, where the data interpretation shall be determined by the user. Each data character shall be encoded as an 8-bit byte.

5.3 User Selection of Error Correction Level

5.3.1 Selection of Error Correction Level

ECC 000 - 140 symbols offer five levels of error correction using convolutional code error correction, as set out in Table 2. In an application, it is important to understand that these error correction levels result in the generation of a proportional increase in the number of bits in the message (and hence increase in the size of the symbol), and offer different levels of error recovery.

Error Correction Code Level	Maximum % Damage	Increase in User Bits from ECC 0
000	none	none
050	28	33
080	5.5	50
100	12.6	100
140	25	300

Table 2: Error Correction Error Recovery and Overhead Percentages

5.3.2 Additional Error Correction Levels Based on Convolutional Codes

Additional levels of error correction, based on convolutional code algorithms, are possible in Data Matrix. Symbols conforming to these constructions may be found in systems implemented prior to the publication of this specification. Information on these additional Error Correction levels is available from AIM USA.

5.4 Constructing the Unprotected Bit Stream

Figure 2 illustrates that the Unprotected Bit Stream has the Data Prefix Bit Stream as a prefix to the encoded data bits. The component parts of the Data Prefix Bit Stream are defined below.

5.4.1 Format ID Bit Field

The format ID defines the data encodation scheme. The format ID has a decimal value for the purposes of definition and a 5-bit segment value for encoding as defined in Table 3.

Format ID	Encodation Scheme	Binary Segment Value	
		MSB	LSB
1	Base 11	00000	
2	Base 27	00001	
3	Base 41	00010	
4	Base 37	00011	
5	ASCII	00100	
6	8-bit Byte	00101	

Table 3: Encoding the Format ID

5.4.2 CRC Bit Field

The CRC Bit Field is generated by the CRC algorithm. The CRC Value is generated from the original user data as 8-bit bytes before encodation and so produces an independent error check on the user data. Annexe D describes the complete procedure for generating the CRC Value.

5.4.3 Data Length Bit Field

The Data Length Bit Field is 9 bits in length and represents, as a binary value, the number of user data characters being encoded.

5.4.4 Data Prefix Construction

The Data Prefix Bit Stream is constructed as 30 bits as illustrated in Figure 3.

Format ID 5 bits	CRC Value 16 bits	Data Length 9 bits
MSB LSB	MSB LSB	MSB

Note: Some bit fields start with the MSB others start with LSB

Figure 3: Structure of Data Prefix Bit Stream

5.4.5 Completing the Unprotected Bit Stream

The encoded data bits are added as a suffix to the Data Prefix Bit Stream to construct the Unprotected Bit Stream.

5.5 Constructing the Unrandomized Bit Stream

Figure 2 illustrates that the Unrandomized Bit

Stream has three constituent parts:

- a. Header
- b. Protected Bit Stream
- c. Trailer

The component parts shall be generated as defined below.

5.5.1 Header Construction

The header of the Unrandomized Bit Stream contains the ECC Bit Field, which identifies the convolutional code structure used to protect the data encoded in the symbol. The ECC Bit Field is 7 or 19 bits long and the values are shown in Table 4.

ECC Level	Binary Segment Identifier	
	MSB	LSB
000		1111110
050	0001110000000001	1110
080	1110001110000001	1110
100	1111111110000001	1110
140	1111110001110001	1110

Table 4: ECC Bit Field

5.5.2 Applying Convolutional Coding to Create the Protected Bit Stream

One of the five error correction levels shall be applied. The selection criteria are defined in Section 5.3. No error correction is applied for ECC 000, so the Unprotected Bit Stream becomes the Protected Bit Stream.

For the other four error correction levels, convolutional coding is applied. This expands the user data proportionally throughout its length. The encoded bit stream shall be created by processing the unprotected bit stream through the appropriate error correction state machine and reading the results. The circuit diagrams of the four state machines for ECC 050 to 140 are given in Annexe E.

5.5.3 Trailer Construction

A Trailer containing pad bits (zeros) is appended to the Protected Bit Stream. Pad bits shall be added at the end of the bit stream to ensure that the square root of the total number of bits in the Unrandomized Bit Stream shall be an odd integer between 7 and 47.

This ensures that the symbol is square.

5.5.4 Completing the Unrandomized Bit Stream

The Protected Bit Stream, with the header and trailer added, is called the Unrandomized Bit Stream and is shown in Figure 2.

5.6 Pattern Randomizing

The Unrandomized Bit Stream is processed by the pattern randomizing algorithm and produces the Randomized Bit Stream. The pattern randomizing algorithm consists of a bitwise XOR operation between the Unrandomized Bit Stream and the Master Random Bit Stream as given in Annexe F starting with the MSB position and continuing for the length of the Unrandomized Bit Stream.

5.7 Module Placement in Matrix

The size of the sides of the data module grid is given by the odd integer square root (between 7 and 47) calculated in the procedure defined in Section 5.5.3. The Randomized Bit Stream is placed into the matrix one module at a time according to the data module placement grids given in Annexe B.

The finder pattern (as defined in Section 4.3.1) shall be placed to produce an external border to the data module grid.

6 ECC 200 Requirements

6.1 Encode Procedure Overview

This section provides an overview of the encoding procedure. Following sections will provide more details. An encoding example for ECC 200 is given in Annexe R. The following steps convert user data to an ECC 200 symbol:

Step 1: Data Encodation

Analyze the data stream to identify the variety of different characters to be encoded. ECC 200 includes various encodation schemes which allow a defined set of characters to be converted into codewords more efficiently than the default scheme. Insert additional codewords to switch between the encodation schemes and to perform other functions. Add pad characters as needed to fill out the required number of codewords. If the user does not specify the matrix size, then

Encodation Scheme	Characters	Bits per Data Character
ASCII	double digit numerics	4
	ASCII values 0-127	8
	Extended ASCII values 128-255	16
C40	Primarily upper-case alphanumeric	5.33
Text	Primarily lower-case alphanumeric	5.33
X12	ANSI X12 EDI data set	5.33
EDIFACT	ASCII values 32-94	6
Base 256	All byte values 0-255	8

Table 5: Encodation Schemes for ECC 200

choose the smallest size that accommodates the data. A complete list of matrix sizes is shown in Section 6.5, Table 11.

Step 2: Error Checking and Correcting Generation

For symbols with more than 255 codewords, sub-divide the codeword stream into interleaved blocks to enable the error correction algorithms to be processed as shown in Annexe G. Generate the error correction codewords for each block. The result of this process expands the codeword stream by the number of error correction codewords. Place the error correction codewords after the data codewords.

Step 3: Module Placement in Matrix

Place the codeword modules in the matrix. Insert the alignment pattern modules, if any, in the matrix. Add the finder pattern modules around the matrix.

6.2 Data Encodation

6.2.1 Overview

The data may be encoded using any combination of six encodation schemes (see Table 5).

ASCII encodation is the basic scheme. All other encodation schemes are invoked from ASCII encodation and return to this scheme.

The compaction efficiencies given in Table 5 need to be interpreted carefully. The best scheme for a given set of data may not be the one with the fewest bits per data character. If the highest degree of compaction is required, account has to be taken of switching between code sets and of switching to and from encodation schemes (see Annexe S). It should also be noted that even if the number of codewords is minimized, the codeword stream may need to be expanded to fill a symbol. This fill process is done using pad characters.

6.2.2 Default Character Interpretation

The default character interpretation for character values 0 to 127 shall conform to ANSI X3.4. The default character interpretation for character values 128 to 255 shall conform to ISO 8859-1: Latin Alphabet No. 1. The graphical representation of data characters shown throughout this document complies with the default interpretation. This interpretation can be changed using Extended Channel Interpretation (ECI) escape sequences, see Section 6.4. The default interpretation corresponds to ECI 000003.

6.2.3 ASCII Encodation

ASCII encodation is the default set for the first symbol character in all symbol sizes. It encodes ASCII data, double density numeric data and symbology control characters.

Symbology control characters include function characters, the pad character and the switches to other code sets.

ASCII data is encoded as codewords 1 to 128 (ASCII value plus 1). Extended ASCII (data values 128 to 255) is encoded using the upper shift symbology control character (see Section 6.2.4.2). The digit pairs 00 to 99 are encoded with codewords 130 to 229 (numeric value plus 130). The ASCII code assignments are shown in Table 6.

Codeword	Data or Function
1-128	ASCII data (ASCII value + 1)
129	Pad
130-229	2-digit data 00-99 (Numeric Value + 130)
230	Latch to C40 Encodation
231	Latch to Base 256 Encodation
232	FNC1
233	Structured Append
234	Reader Programming
235	Upper Shift (shift to Extended ASCII)
236	05 Macro
237	06 Macro
238	Latch to ANSI X12 Encodation
239	Latch to Text Encodation
240	Latch to EDIFACT Encodation
241	ECI Character

Table 6: ASCII Encodation Values

6.2.4 Symbology Control Characters

ECC 200 symbols have several special symbology control characters, which have particular significance to the encodation scheme. These characters shall be used to instruct the decoder to perform certain functions or to send specific data to the host computer as described in Section 6.2.4.1 to 6.2.4.9. These symbology control characters are found predominantly in the ASCII encodation set.

6.2.4.1 Latch Characters

A Latch Character shall be used to switch from ASCII encodation to one of the other encodation schemes. All codewords which follow a Latch Character shall be compacted according to the new encodation scheme.

The encodation schemes have different methods for returning to the ASCII encodation set.

6.2.4.2 Upper Shift Character

The Upper Shift character is used to encode a single extended ASCII character (values 128-255). An extended ASCII character encoded in the ASCII, C40, or Text encodation scheme requires a preceding Upper Shift character and the extended ASCII character value decreased by 128 is then encoded according to the rules of the code set.

In ASCII encodation, the Upper Shift character is represented by codeword 235. The reduced data

value (i.e. ASCII value - 128) is transformed into its codeword value by adding 1.

For example, to encode ¥ (ASCII value 165), an Upper Shift character (Codeword 235) is followed by value 37 (165 - 128), which is encoded as codeword 38.

If there are long data strings of characters from the extended ASCII range, a Latch to Base 256 Encodation should be more efficient.

6.2.4.3 Pad Character

If the encoded data, irrespective of the encodation scheme in force, does not fill the data capacity of the symbol, pad characters shall be added (prefixed by codeword 254 or Unlatch if necessary) to fill out the remaining data capacity of the symbol. The pad characters shall only be used for this purpose.

The 253-State pattern randomizing algorithm is applied to the pad characters starting at the second pad character and continuing to the end of the symbol (see Annexe H.1).

6.2.4.4 Extended Channel Interpretation Character

An Extended Channel Interpretation (ECI) character is used to change from the default interpretation used to encode data. The Extended Channel Interpretation protocol is common across a number of symbologies and its application to ECC 200 is defined more fully in Section 6.4.

The ECI character shall be followed by one, two, or three codewords which identify the ECI being invoked. The new ECI remains in place until the end of the encoded data, or until another ECI character is used to invoke another interpretation.

6.2.4.5 Shift Characters in C40 and Text Encodation

The three shift characters in C40 and Text encodation are used to switch for a single data character to one of the subsets of these encodation schemes.

6.2.4.6 FNC1 Alternate Data Type Identifier

To encode data to conform to specific industry standards as authorized by AIM International, a FNC1 character shall appear in the first or second symbol character position (or in the fifth or sixth data positions of the first symbol of Structured Append). FNC1 encoded in any other position is used as a field separator and shall be transmitted as GS control character (ASCII value 29).

6.2.4.7 Macro Characters

Data Matrix provides a means of abbreviating an industry specific header and trailer in one symbol character. This feature exists to reduce the number of symbol characters needed to encode data in a symbol using certain structured formats.

The two Macro characters apply only when in the first symbol character position. They shall not be used in conjunction with Structured Append and their functions are defined in Table 7. The header will be transmitted as a prefix to the data stream and the trailer will be transmitted as a suffix to the data stream. The symbology identifier, if used, will precede the header.

Macro Codeword	Name	Interpretation	
		Header	Trailer
236	05 Macro	$\text{D} >^R_s 05^G_s$	$^R_s \text{E}_O_T$
237	06 Macro	$\text{D} >^R_s 06^G_s$	$^R_s \text{E}_O_T$

Table 7: Macro Functions

6.2.4.8 Structured Append Character

A Structured Append character is used to indicate that the symbol is part of a Structured Append according to the rules defined in Section 6.6.

6.2.4.9 Reader Programming Character

A Reader Programming character indicates that the symbol encodes a message used to program the reader system. The Reader Programming character should appear as the first codeword and Reader

Programming should not be used with Structured Append.

6.2.5 C40 Encodation

C40 encodation enables three alphanumeric data characters to be packed into two codewords. The upper-case alphabetic characters, numerics and space can each be encoded as a single C40 value. All other ASCII characters and special characters require two C40 values starting with one of three shift characters. A C40 shift character only affects the single C40 value following the shift character. The full C40 character set assignments are shown in Annexe J.1.

6.2.5.1 Switching to and from C40 Encodation

It is possible to switch to C40 encodation from ASCII encodation using the appropriate latch codeword (230).

Codeword 254 immediately following a pair of codewords in C40 encodation acts as an Unlatch codeword to switch back to ASCII encodation. Otherwise, the C40 encodation remains in effect to the end of the data encoded in the symbol.

6.2.5.2 C40 Encodation Rules

Each pair of codewords represents a 16-bit value where the first codeword represents the most significant 8 bits. Three C40 values (C_1 , C_2 , C_3) shall be encoded as:

$$(1600 * C_1) + (40 * C_2) + C_3 + 1$$

which produces a value 1 to 64000. Figure 4 illustrates three C40 values compacted into two codewords.

Characters in the Shift 1, Shift 2 and Shift 3 sets shall be encoded by first encoding the appropriate

Data Characters	AIM
C40 Values	14, 22, 26
Calculate 16-bit value	$(1600 * 14) + (40 * 22) + 26 + 1 = 23307$
1st Codeword: (16-bit value) div 256	$23307 \text{ div } 256 = 91$
2nd Codeword: (16-bit value) mod 256	$23307 \text{ mod } 256 = 11$
Codewords	91, 11

Figure 4: Example of C40 Encoding

shift character, and then the C40 value for the data.

C40 encodation may be in effect at the end of the symbol's codewords which encode data. The following rules apply when only one or two symbol characters remain in the symbol before the start of the error correction codewords:

- a. If two symbol characters remain and three C40 values remain to be encoded (which may include both data and shift characters) encode the three C40 values in the last two symbol characters. A final Unlatch codeword is not required.
- b. If two symbol characters remain and two C40 values remain to be encoded (the first C40 value may be a shift or data character but the second must represent a data character); encode the two remaining C40 values followed by a pad C40 value of 0 (Shift 1) in the last two symbol characters. A final Unlatch codeword again is not required.
- c. If two symbol characters remain and only one C40 value (data character) remains to be encoded, the first symbol character is encoded as an Unlatch character and the last symbol character is encoded with the data character using the ASCII encodation scheme.
- d. If one symbol character remains and one C40 value (data character) remains to be encoded, the last symbol character is encoded with the data character using the ASCII encodation scheme. The Unlatch character is not encoded, but is assumed, before the last symbol character whenever an odd number of symbol characters remain in the symbol in the C40 encodation scheme.

In all other cases, either an Unlatch character is used to exit the C40 encodation scheme before the end of the symbol, or a larger symbol size is required to encode the data.

6.2.5.3 Use of Upper Shift with C40

In C40 encodation the Upper Shift character is not a symbology function character but a shift within the encodation set. When a data character from the extended ASCII character range is encountered, three or four values in C40 encodation need to be encoded according to the following rule:

IF [ASCII value - 128] is in the Basic Set then:
[1(Shift 2)][30(Upper Shift)][V(ASCII value -

128)]

ELSE

[1(Shift 2)][30(Upper Shift)][0, 1, or 2(Shift 1, 2, or 3)][V(ASCII value - 128)]

In the rule the number in [] equates to the C40 values from Annexe J.1; V has been used to indicate the appropriate C40 value.

6.2.6 Text Encodation

Text encodation is designed to encode normal printed text, which is predominantly lowercase characters. It is similar in structure to the C40 encodation set, except that lowercase alphabetic characters are directly encoded (i.e. without using a shift). Upper-case alphabetic characters are preceded by a Shift 3. The full Text encodation character set assignments are shown in Annexe J.2.

6.2.6.1 Switching to and from Text Encodation

It is possible to switch to Text encodation from ASCII encodation using the appropriate latch codeword (239).

Codeword 254 immediately following a pair of codewords in text encodation acts as an Unlatch codeword to switch back to ASCII encodation. Otherwise, the Text encodation remains in effect to the end of the data encoded in the symbol.

6.2.6.2 Text Encodation Rules

The rules for C40 encodation apply.

6.2.7 ANSI X12 Encodation

ANSI X12 encodation is used to encode the standard ANSI X12 electronic data interchange characters, which are compacted three data characters to two codewords in a manner similar to C40 encodation. It encodes upper-case alphabetic characters, numerics, space and the three standard ANSI X12 terminator and separator characters. The ANSI X12 code assignments are shown in Table 8. There are no shift characters in the ANSI X12 encodation set.

X12 Value	Encoded Characters	ASCII Values
0	X12 segment terminator <CR>	13
1	X12 segment separator *	42
2	X12 subelement separator >	62
3	space	32
4-13	0-9	48-57
14-39	A-Z	65-90

Table 8: ANSI X12 Encodation Set

6.2.7.1 Switching to and from ANSI X12 Encodation

It is possible to switch to ANSI X12 encodation from ASCII encodation using the appropriate latch codeword (238).

Codeword 254 immediately following a pair of codewords in ANSI X12 encodation acts as an Unlatch codeword to switch back to ASCII encodation. Otherwise, the ANSI X12 encodation remains in effect to the end of the data encoded in the symbol.

6.2.7.2 ANSI X12 Encodation Rules

Data Character	ASCII		EDIFACT Value
	Decimal Value	8-bit Binary Value	
A	65	01000001	000001
9	57	00111001	111001

Note: During the decode process, if the leading (6th) bit is 1, the bits 00 are prefixed to create the 8-bit byte. If the leading (6th) bit is 0, the bits 01 are prefixed to create the 8-bit byte.

The exception to this is the EDIFACT value 011111 which is the symbology control Unlatch character to return to ASCII encodation.

Figure 5: The Relationship Between the EDIFACT Value and the 8-bit Byte Value

Data characters	D	A	T	A
Binary values (Annexe J.3)	00 01 00	00 00 01	01 01 00	00 00 01
Divide into 3 8-bit bytes	00 01 00 00	00 01 01 01	00 00 00 01	
Codeword values	16	21	1	

Figure 6: Example of EDIFACT Encodation

The rules of C40 encodation apply. The exception is at the end of encoding ANSI X12 data. If the data characters do not fully utilize pairs of codewords, then following the last complete pair of codewords switch to ASCII using codeword 254 and continue using ASCII encodation, except when a single symbol character is left at the end before the first error correction character. This single symbol character uses the ASCII encodation scheme without requiring an Unlatch codeword.

6.2.8 EDIFACT Encodation

The EDIFACT encodation scheme includes 63 ASCII values (values from 32 to 94) plus an Unlatch character (binary 011111) to return to ASCII encodation. EDIFACT encodation encodes four data characters in three codewords. It includes all the numeric, alphabetic and punctuation characters defined in the EDIFACT Level A character set without any of the shifts required in C40 encodation.

6.2.8.1 Switching to and from EDIFACT Encodation

It is possible to switch to EDIFACT encodation from

ASCII encodation using the appropriate latch codeword (240).

The Unlatch character in EDIFACT encodation shall be used as a terminator at the end of EDIFACT encodation, which reverts to ASCII encodation.

6.2.8.2 EDIFACT Encodation Rules

The EDIFACT encodation character set is defined in Annexe J.3. There is a simple relationship between the 6-bit EDIFACT value and the ASCII 8-bit byte. The leading two bits of the 8-bit byte are ignored to create the EDIFACT 6-bit value, as illustrated in Figure 5.

Strings of four EDIFACT characters are encoded in three codewords. For a simple encodation process, the leading two bits of the 8-bit byte are removed. The remaining 6-bit byte is the EDIFACT value and shall be directly encoded into the codeword as illustrated in Figure 6.

When EDIFACT Encodation is terminated with the Unlatch character, any remaining bits left in the single symbol character shall be filled with zeros. ASCII mode starts with the next symbol character. If EDIFACT Encodation is in effect at the end of the symbol before the first error correction character, and only one or two codewords remain after the last EDIFACT codeword triplet, these remaining codewords shall be encoded in ASCII Encodation without requiring an Unlatch character.

6.2.9 Base 256 Encodation

The Base 256 encodation scheme shall be used to encode any 8-bit byte data, including extended channel interpretations and binary data. The default interpretation is defined in 6.2.2

The 255-State pattern randomizing algorithm is applied to each Base 256 sequence within the encoded data (see Annexe H.2). It starts after the latch to Base 256 encodation and ends at the last character specified by the Base 256 field length.

6.2.9.1 Switching to and from Base 256 Encodation

It is possible to switch to Base 256 encodation from ASCII encodation using the appropriate latch codeword (231). At the end of Base 256 encodation, encodation automatically reverts to ASCII encodation.

The appropriate ECI, if other than the default, shall be invoked prior to switching. The ECI sequence need not occur immediately before switching to Base 256 encodation.

6.2.9.2 Base 256 Encodation Rules

After switching to Base 256 encodation, the first one (d1) or two (d1, d2) codewords define the data field length in bytes. Table 9 specifies how the field length is defined.

Thereafter, all encodation shall be of the byte values.

Field Length	Values of d1,d2	Permitted Values of d
Remainder of Symbol	d1 = 0	d1 = 0
1 to 249	d1 = length	d1 = 1 to 249
250 to 1555	d1 = (length DIV 250) + 249 d2 = length MOD 250	d1 = 250 to 255 d2 = 0 to 249

Table 9: Base 256 field Length

6.3 User Considerations

ECC 200 offers flexibility in the way data is encoded. Alternate character sets may be invoked using the ECI protocol. Data may be encoded in square or rectangular symbols. It is also possible to append up to 16 ECC 200 symbols.

6.3.1 User Selection of Extended Channel Interpretation

The use of an alternative Extended Channel Interpretation to identify a particular code page or more specific data interpretation requires additional codewords to invoke the feature. The use of the Extended Channel Interpretation protocol (see Section 6.4) provides the capability to encode data from alphabets other than the Latin alphabet (ISO 8859-1 Latin Alphabet No. 1) supported by the default interpretation (ECI 000003).

6.3.2 User Selection of Symbol Size and Shape

ECC 200 has twenty-four square and six rectangular symbol configurations. The size and shape may be selected to suit the requirement of the application. These configurations are technically specified in Section 6.5.

6.4 Extended Channel Interpretation

The Extended Channel Interpretation (ECI) protocol allows the output data stream to have interpretations

different from that of the default character set. The ECI protocol is defined consistently across a number of symbolologies. Four broad types of interpretations are supported in Data Matrix:

- international character sets (or code pages)
- general purpose interpretations such as encryption and compaction
- user defined interpretations for closed systems
- control information for structured append in unbuffered mode.

The Extended Channel Interpretation protocol is fully specified in the *Extended Channel Interpretation (ECI) Assignments* document. The protocol provides a consistent method to specify particular interpretations on byte values before printing and after decoding.

The Extended Channel Interpretation is identified by a 6-digit number which is encoded in the Data Matrix symbol by the ECI character followed by one to three codewords.

The Extended Channel Interpretation can only be used with readers enabled to transmit the symbology identifiers. Readers that are not enabled to transmit the symbology identifier shall not transmit the data from any symbol containing an ECI. An exception can be made if the ECI(s) can be handled entirely within the reader.

The Extended Channel Interpretation protocol shall only be applied to ECC 200 symbols. A specified Extended Channel Interpretation may be invoked anywhere in the encoded message.

6.4.1 Encoding ECIs

The various encodation schemes of Data Matrix for ECC 200 (defined in Table 5) may be applied under

any of the Extended Channel Interpretations. The ECI can only be invoked from ASCII encodation; once this has occurred, switching may take place between any of the encodation schemes. The encodation mode used is determined strictly by the 8-bit data values being encoded and does not depend on the Extended Channel Interpretation in force. For example, a sequence of values in the range 48 to 57 (decimal) would be most efficiently encoded in numeric mode even if they are not to be interpreted as numbers.

The ECI assignment is invoked using codeword 241 (ECI character) in ASCII encodation. One, two, or three additional codewords are used to encode the ECI Assignment number. The encodation rules are defined in Table 10.

The following examples illustrate the encodation:

ECI = 015000

Codewords: [241] [(15000 - 127) div 254 + 128] [(15000 - 127) mod 254 + 1]

= [241][58 + 128][141 + 1]

= [241][186][142]

ECI = 090000

Codewords: [241] [(90000 - 16383) div 64516 + 192] [(90000 - 16383) div 254 mod 254 + 1] [(90000 - 16383) mod 254 + 1]

= [241][1 + 192][289 mod 254 + 1][211 + 1]

= [241][193][36][212]

6.4.2 ECIs and Structured Append

ECIs may occur anywhere in the message encoded in a single or Structured Append (see Section 6.6) set

ECI Assignment Value	Codeword Sequence	Codeword Values	Ranges
000000 to 000126	C ₀ C ₁	241 ECI_no + 1	C ₁ = (1 to 127)
000127 to 016382	C ₀ C ₁ C ₂	241 (ECI_no - 127) div 254 + 128 (ECI_no - 127) mod 254 + 1	C ₁ = (128 to 191) C ₂ = (1 to 254)
0016383 to 999999	C ₀ C ₁ C ₂ C ₃	241 (ECI_no - 16383) div 64516 + 192 [(ECI_no - 16383) div 254] mod 254 + 1 (ECI_no - 16383) mod 254 + 1	C ₁ = (192 to 207) C ₂ = (1 to 254) C ₃ = (1 to 254)

Table 10: Encoding ECI Assignment Numbers in ECC 200

Symbol Size ¹		Data Region		Mapping Matrix	Total Codewords		Reed-Solomon Block		Inter-leaved	Data Capacity			Error Correction	Max. Correctable
Row	Col	Size	Number	Size	Data	Error	Data	Error	Blocks	Num. Cap.	Alphanum. Cap.	Byte Cap.	Overhead %	Codeword Error/Erasure
10	10	8x8	1	8x8	3	5	3	5	1	6	3	1	62.5	2/+
12	12	10x10	1	10x10	5	7	5	7	1	10	6	3	58.3	3/+
14	14	12x12	1	12x12	8	10	8	10	1	16	10	6	55.6	5/7
16	16	14x14	1	14x14	12	12	12	12	1	24	16	10	50	6/9
18	18	16x16	1	16x16	18	14	18	14	1	36	25	16	43.8	7/11
20	20	18x18	1	18x18	22	18	22	18	1	44	31	20	45	9/15
22	22	20x20	1	20x20	30	20	30	20	1	60	43	28	40	10/17
24	24	22x22	1	22x22	36	24	36	24	1	72	52	34	40	12/21
26	26	24x24	1	24x24	44	28	44	28	1	88	64	42	38.9	14/25
32	32	14x14	4	28x28	62	36	62	36	1	124	91	60	36.7	18/33
36	36	16x16	4	32x32	86	42	86	42	1	172	127	84	32.8	21/39
40	40	18x18	4	36x36	114	48	114	48	1	228	169	112	29.6	24/45
44	44	20x20	4	40x40	144	56	144	56	1	288	214	142	28	28/53
48	48	22x22	4	44x44	174	68	174	68	1	348	259	172	28.1	34/65
52	52	24x24	4	48x48	204	84	102	42	2	408	304	202	29.2	42/78
64	64	14x14	16	56x56	280	112	140	56	2	560	418	278	28.6	56/106
72	72	16x16	16	64x64	368	144	92	36	4	736	550	366	28.1	72/132
80	80	18x18	16	72x72	456	192	114	48	4	912	682	454	29.6	96/180
88	88	20x20	16	80x80	576	224	144	56	4	1152	862	574	28	112/212
96	96	22x22	16	88x88	696	272	174	68	4	1392	1042	694	28.1	136/260
104	104	24x24	16	96x96	816	336	136	56	6	1632	1222	814	29.2	168/318
120	120	18x18	36	108x108	1050	408	175	68	6	2100	1573	1048	28	204/390
132	132	20x20	36	120x120	1304	496	163	62	8	2608	1954	1302	27.6	248/472
144	144	22x22	36	132x132	1558	620	156	62	8*	3116	2335	1556	28.5	310/590
							155	62	2*					
Rectangular Symbols														
8	18	6x16	1	6x16	5	7	5	7	1	10	6	3	58.3	3/+
8	32	6x14	2	6x28	10	11	10	11	1	20	13	8	52.4	5/+
12	26	10x24	1	10x24	16	14	16	14	1	32	22	14	46.7	7/11
12	36	10x16	2	10x32	22	18	22	18	1	44	31	20	45.0	9/15
16	36	14x16	2	14x32	32	24	32	24	1	64	46	30	42.9	12/21
16	48	14x22	2	14x44	49	28	49	28	1	98	72	47	36.4	14/25
Note 1: Symbol size does not include quiet zone. *Note: In the largest symbol (144x144), the first eight Reed-Solomon blocks shall be 218 codewords long encoding 156 data codewords. The last two blocks shall encode 217 codewords (155 data codewords). All the blocks have 62 error correction codewords. +Note: Does not apply														
Table 11: ECC 200 Symbol Attributes														

of Data Matrix symbols. Any ECI invoked shall apply until the end of the encoded data, or until another ECI is encountered. Thus the interpretation of the ECI may straddle two or more symbols.

6.4.3 Post-Decode Protocol

The protocol for transmitting ECI data shall be as defined in Section 11.4. When using ECIs, symbology identifiers (see Section 11.5) shall be fully implemented and the appropriate symbology identifier transmitted as a preamble.

6.5 ECC 200 Symbol Attributes

6.5.1 Symbol Sizes and Capacity

There are 24 square symbols and 6 rectangular symbols available in ECC 200. These are as specified in Table 11.

6.5.2 Insertion of Alignment Patterns into Larger Symbols

As shown in Table 11, square symbols 32x32 and larger and four rectangular symbols (8 x 32, 12 x 36, 16 x 36, and 16 x 48) have two or more data regions. The square symbols are divided into 4, 16, or 36 data regions (as illustrated in Annexe K.1, K.2, and K.3). The rectangular symbols are divided into two data regions (as illustrated in Annexe K.4). These data regions are bounded by alignment patterns. The alternating dark modules of the alignment pattern shall be to the top and right of a data region and identify the even columns and rows.

6.6 Structured Append

6.6.1 Basic Principles

Up to 16 ECC 200 symbols may be appended in a structured format. If a symbol is part of a Structured Append, this is indicated by codeword 233 in the first symbol character position. This is immediately followed by three structured append codewords. The first codeword is the symbol sequence indicator. The second and third codewords are the file identification.

6.6.2 Symbol Sequence Indicator

This codeword indicates the position of the symbol within the set (up to 16) of ECC 200 symbols in the Structured Append format in the form m of n symbols.

The first 4 bits of this codeword identify the position of the particular symbol as the binary value of $(m-1)$. The last 4 bits identify the total number of

the symbols to be concatenated in the Structured Append format as the binary value of $(17-n)$. The 4-bit patterns shall conform to those defined in Table 12.

Example:

To indicate the 3rd symbol of a set of 7, this shall be encoded thus:

3rd position:	0010
Total 7 symbols:	1010
Bit pattern:	00101010
Codeword value:	42

6.6.3 File Identification

The file identification is defined by the value of its two codewords. Each file identification codeword may have a value 1 to 254, allowing 64516 different file identifications. The purpose of the file identification is to increase the probability that only logically linked symbols are processed as part of the same message.

Symbol Position	Bits 1234	Total Number of Symbols	Bits 5678
1	0000		
2	0001	2	1111
3	0010	3	1110
4	0011	4	1101
5	0100	5	1100
6	0101	6	1011
7	0110	7	1010
8	0111	8	1001
9	1000	9	1000
10	1001	10	0111
11	1010	11	0110
12	1011	12	0101
13	1100	13	0100
14	1101	14	0011
15	1110	15	0010
16	1111	16	0001

Table 12: Structured Append Symbol Position Bits

6.6.4 FNC1 and Structured Append

If Structured Append is used in conjunction with FNC1 (see Section 6.2.4.6), the first four codewords

shall be used for Structured Append and the fifth and sixth codewords are available for FNC1 usage. FNC1 shall not be repeated in these positions in the second and subsequent symbols, except when used as a field separator.

6.6.5 Buffered and Unbuffered Operation

The message within a Structured Append sequence can be buffered in the reader in its entirety and transmitted after all of the symbols have been read. Alternatively, the reader may transmit the decoded data in each symbol as it is read. In this unbuffered operation, the ECI protocol for structured append defines a control block that shall be prefixed to the beginning of each transmission.

6.7 Error Detection and Correction

ECC 200 symbols employ Reed-Solomon error correction.

For ECC 200 symbols with less than 255 total codewords, the error correction codewords are calculated from data codewords with no interleaving.

For ECC 200 symbols with more than 255 total codewords, the error correction codewords are calculated from data codewords with the interleaving procedure described in Annexe G.

Each ECC 200 symbol employs a specific combination of interleaving and number of error correction codewords. These combinations are given in Table 11.

The polynomial arithmetic for ECC 200 shall be calculated using bit-wise modulo 2 arithmetic and byte-wise modulo 100101101 (decimal 301) arithmetic. This is a Galois field of 2^8 with 100101101 representing the field's prime modulus polynomial: $x^8 + x^5 + x^3 + x^2 + 1$.

Sixteen different generator polynomials are used for generating the appropriate error correction codewords. These are given in Annexe L.1.

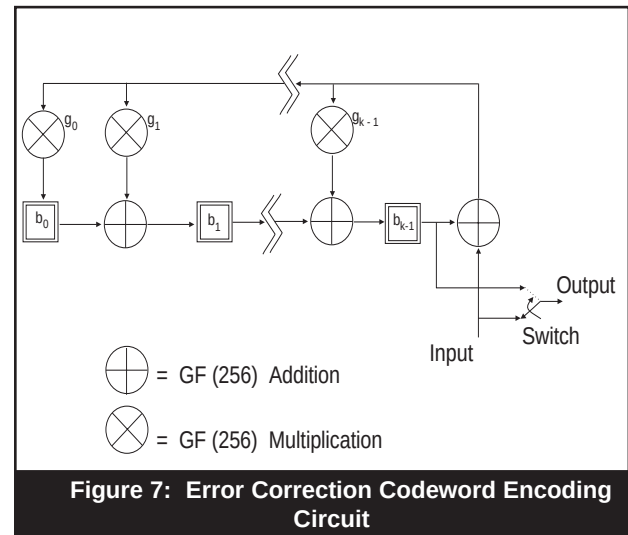
6.7.1 Generating the Error Correction Codewords

The error correction codewords are the remainder after dividing the data codewords by a polynomial $g(x)$ used for Reed-Solomon codes (see Annexe L.1).

The data codewords are the coefficients of the terms of a polynomial with the coefficient of the highest term being the first data codeword and the lowest power term being the last data codeword

before the first error correction codeword. The highest order coefficient of the remainder is the first error correction codeword and the zero power coefficient is the last error correction codeword and the last codeword.

This can be implemented by using the division circuit as shown in Figure 7. The registers b_0 through b_{k-1} are initialized as zeros. There are two phases to generate the encoding. In the first phase, with the switch in the down position the data codewords are passed both to the output and the circuit. The first phase is complete after n clock pulses. In the second phase ($n+1 \dots n+k$ clock pulses), with the switch in the up position, the error correction codewords $e_{k-1} \dots e_0$ are generated by flushing the registers in order and complementing the output while keeping the data input at 0. The codewords output from the shift register are in the order that they are to be placed in the symbol. If interleaving is used, the codewords will not be placed in consecutive symbol characters. (See Annexe G).



6.7.2 Error Correction Capacity

The error correction codewords can correct two types of erroneous codewords: erasures (erroneous codewords at known locations) and errors (erroneous

codewords at unknown locations). An erasure is an unscanned or undecodable symbol character. An error is a misdecoded symbol character. The number of erasures and errors correctable is given by the following formula:

$$e + 2t \leq d - p$$

where:

- e = number of erasures
- t = number of errors
- d = number of error correction codewords
- p = number of codewords reserved for error detection.

In the general case, $p = 0$. However, if most of the error correction capacity is used to correct erasures, then the possibility of an undetected error is increased. Whenever the number of erasures is more than half the number of error correction codewords, $p = 3$. For small symbols (10x10, 12x12, 8x18, and 8x32), erasure correction should not be used ($e = 0$ and $p = 1$).

6.8 Symbol Construction

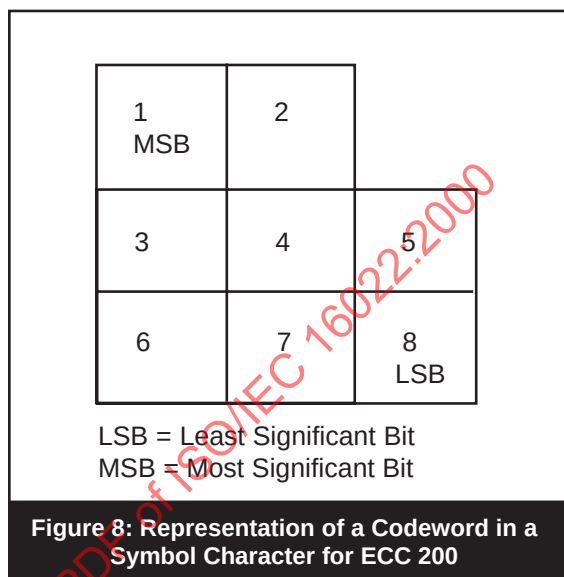
Given the codeword sequence obtained in the previous sections, an ECC 200 symbol is constructed using the following steps:

1. Place codeword modules in a mapping matrix
2. Insert alignment pattern modules, if any
3. Place finder modules along the perimeter

6.8.1 Symbol Character Placement

Each symbol character shall be represented by eight modules which are nominally square in shape; each module represents a binary bit. A dark module is a one and a light module is a zero. The eight modules are in order from left to right and top to bottom to form a symbol character as shown in Figure 8.

Because the symbol character shape defined in Figure 8 cannot be perfectly nested at the symbol boundary, some symbol characters are split into portions. Symbol character placement is defined in the C language program in Annexe M.1, described in Annexe M.2 and illustrated in Annexe M.3.



6.8.2 Alignment Pattern Module Placement

This step is only needed for larger matrices:

- square: 32x32 and larger
- rectangular: 8x32, 12x36 and larger

The mapping matrix is sub-divided into data regions, of the sizes defined in Table 11, for the chosen symbol format. The data regions are separated from each other by two-module-wide alignment patterns. This will result in some of the symbol characters being split between two adjacent data regions.

For square matrices, the alignment patterns are placed between the data regions horizontally and vertically in pairs with a total alignment pattern count of 2, 6, or 10 as shown in Annexe K.1 - K.3. For rectangular matrices, only a single vertical alignment pattern is placed between the data regions as shown in Annexe K.4.

6.8.3 Finder Pattern Module Placement

Modules are placed along the perimeter of the matrix to construct the finder pattern as described in Section 4.3.1.

7 Symbol Dimensions

7.1 Dimensions

Data Matrix symbols shall conform to the following dimensions:

X dimension:	the width of a module shall be specified by the application, taking into account the scanning technology to be used, and the technology to produce the symbol
finder pattern:	the width of the finder pattern shall equal X
alignment pattern:	the width of the alignment pattern shall equal 2X minimum

7.2 Quiet zone

The minimum quiet zone is equal to one module width on all four sides.

8 Symbol Quality

Data Matrix symbols shall be assessed for quality using the 2D matrix bar code symbol print quality guidelines presented in Annexe N, as augmented and modified below.

8.1 Obtaining the Test Image

A grey-scale image of the symbol being tested shall be obtained with a precision video camera-based setup as described in Annexe N.1, but with an illumination color and direction specified by the application.

8.2 Symbol Quality Parameters

8.2.1 Decode

The reference decode algorithm presented in Section 9 shall be applied to the test image. If it results in a successful decode of the entire data message, then Decode passes with a grade of "A" (4.0), otherwise it fails with a grade of "F" (0.0).

8.2.2 Symbol Contrast

Symbol contrast shall be graded using all of the grey-scale pixel values in the test image which fall within the symbol's boundary, including the 1X wide quiet zones, as determined by the reference decode.

The procedure is defined in Annexe N.2.2.

8.2.3 "Print" Growth

The reference decode begins by creating a high resolution binary digitized image of the test symbol, and at a later point establishes the position of the "alternating module centerlines" which bisect the alternating patterns on the perimeter of the symbol. "Print" Growth shall be assessed by checking that the "duty cycle" of the lines through the alternating patterns is around 50%.

Taking the top and right side of the symbol independently (since horizontal and vertical growth can differ substantially), proceed from the L-boundary along each perimeter peak line and sum the number of light (N_L) and dark (N_D) pixels encountered, traversing the full width or height of the symbol but excluding the final dark corner module on ECC 000 - 140 symbols which have an odd number of modules per side. The resulting measure of print growth in each direction is $D = N_D / (N_L + N_D)$, which shall be graded against $D_{NOM} = 0.50$ with $D_{MIN} = 0.35$ and $D_{MAX} = 0.65$ as prescribed in Annexe N.2.3. The Print Growth grade shall be the lower of those achieved along the top and right side of the symbol.

8.2.4 Axial Nonuniformity

The reference decode algorithm ultimately creates a grid of data module sampling points throughout the entire data region of the test image. The precise horizontal and vertical spacings of those sampling points are the basis for assessing axial nonuniformity.

Working independently with the horizontal and vertical spacings between adjacent data modules, compute their average values X_{AVG} and Y_{AVG} over the whole symbol. The Axial Nonuniformity grade is then based on how closely these two average spacings match each other, as computed by the procedure defined in Annexe N.2.4.

8.2.5 Unused Error Correction

For ECC 000 - ECC 140 versions of Data Matrix, the convolutional error encoding can correct for the following percentages of bit errors:

Grade	Reference Decode	Symbol Contrast	"Print" Growth	Axial Nonuniformity	Unused Error Correction
A(4.0)	Passes	$SC \geq 70\%$	$-0.50 \leq D' \leq 0.50$	$AN \leq 0.06$	$UEC \geq 0.62$
B(3.0)		$SC \geq 55\%$	$-0.70 \leq D' \leq 0.70$	$AN \leq 0.08$	$UEC \geq 0.50$
C(2.0)		$SC \geq 40\%$	$-0.85 \leq D' \leq 0.85$	$AN \leq 0.10$	$UEC \geq 0.37$
D(1.0)		$SC \geq 20\%$	$-1.00 \leq D' \leq 1.00$	$AN \leq 0.12$	$UEC \geq 0.25$
F(0.0)	Fails	$SC < 20\%$	$D' < -1.00$ or $D' > 1.00$	$AN > 0.12$	$UEC < 0.25$

Table 13: Summary of Data Matrix Symbol Print Quality Parameters

ECC 000: $E_{MAX} = 0.0\%$
 ECC 050: $E_{MAX} = 2.8\%$
 ECC 080: $E_{MAX} = 5.5\%$
 ECC 100: $E_{MAX} = 12.6\%$
 ECC 140: $E_{MAX} = 25.0\%$

Following a successful decode, the actual percentage of bit errors E_{ACT} is simply the number of bits that were corrected divided by the total number of bits in the symbol's data fields. The Unused Error Correction for these symbols is $UEC = 1.0 - (E_{ACT} / E_{MAX})$ which is then graded using the table in Annex N.2.5.

The ECC 200 version of Data Matrix employs Reed-Solomon error correction. The smaller symbols contain just a single error correcting block while the larger symbols are subdivided into two or more interleaved blocks. In all cases, each Reed-Solomon block shall be graded independently as specified in Annex N.2.5, then the Unused Error Correction grade shall be the lowest achieved by any of the blocks.

8.3 Overall Symbol Grade

The overall print quality grade for a Data Matrix symbol is the lowest of the five grades achieved above. Table 13 summarizes the test criteria.

8.4 Process Control Measurements

Several tools and methods are available which can perform useful measurements for monitoring and controlling the process of creating Data Matrix symbols. These include:

1. Symbol contrast readings from a linear bar code verifier.
2. Horizontal print growth from an ECC 200 reference symbol.
3. Determination of axial nonuniformity by

physical measurement

4. Visual inspection of the perimeter pattern for grid nonuniformities and defects.

These tools and methods are described in Annex T.

9 Reference Decode Algorithm for Data Matrix

This reference decode algorithm finds the symbols in an image and decodes them. This is the decode algorithm used in the determination of symbol quality. This decode algorithm refers to black and white states in the image. The algorithm is actually applied to the normal image and again with black and white states reversed.

- A. Find the candidate areas which may contain an "L" shaped finder pattern:
 1. Select a global threshold midway between the maximum reflectance and minimum reflectance in the image. Convert the image to black and white using the global threshold.
 2. Define a distance d_{min} which is the length of the shortest expected leg of an imaged symbol's "L" finder pattern (d_{min} will depend on the pixel density, camera focal length, maximum reading distance, and minimum X-dimension.)
 3. Find all the areas with closed continuous boundaries of white on the outside and black on the inside. Note that one candidate area may be totally contained within another candidate area.
 4. Disregard all areas which have a longest axis less than $0.5 d_{min}$.
 5. Form new merged candidate areas by merging any adjacent areas with a small

separation:

- a. If two areas are separated by less than $0.15 d_{\min}$, form a new area by joining the two areas across the minimum separation point.
 - b. Keep the two adjacent areas and the merged area as three candidate areas.
 - c. Repeat the merging process to form new unique merged areas from all original and merged area combinations.
6. Form new divided candidate areas from any candidate areas with a narrow neck separating large sub-areas, ignoring the merged areas from the previous step:
- a. If an area contains a neck less than $0.15 d_{\min}$ wide at its narrowest and separates two sub-areas each with a longest axis of at least $d_{\min}$, create two new candidate areas by breaking the original area into two new areas across the narrowest part of the neck.
 - b. Keep the original larger area and the two new divided areas as three candidate areas.
 - c. Repeat the dividing process to form new divided areas from all original and divided areas.
7. Ignore any area with a longest axis less than $d_{\min}$. The remaining candidate areas will be processed in the following steps to try to decode valid symbols.
- B. Find the “L” shaped finder pattern in a candidate area:
1. Sort the points in the area’s boundary into sets that fall along straight lines, within 3% of the longest axis of the candidate area. For each set of points, plot a line which minimizes the sum of distances between each of the points in the set and the closest point on the line.
 2. Select the line with the most points. Select a second line with the most number of points which is also perpendicular to the first line to within 10 degrees. If the second line does not contain at least 20% of the number of points in the first line, disregard the area.
 3. The two lines may form the outline of two sides of the “L” in a symbol’s finder pattern.
- C. Find two new lines which are a weighted best fit to the outside of the “L” in the area around the corner of the “L”:
1. Plot the intersection of the two lines. For each of the two sets of points, calculate the weights for each of the distances from the points to the line as follows.
 2. Divide the distance from the farthest point in the set and the intersection of the two “L” lines by $d_{\min}$. Discard the remainder. Call the quotient N.
 3. Divide the set of points into N subsets in sequence starting from the point closest to the intersection of the two lines and ending with the farthest point.
 4. Weight the point-to-line distances for the subset of points closest to the intersection by weight N, the next subset’s distances by N-1, ..., and the farthest subset’s distances by one.
 5. For each side, find a new line which minimizes the sum of the weighted distances between the new line and the side’s set of points.
- D. First assume that the candidate area contains a square symbol. If the area fails to decode as a square symbol, then try to find and decode a rectangular symbol starting from procedure J. For a square symbol, first plot a normalized graph of transitions for the equal sides of the candidate area in order to find the alternating module finder pattern:
1. Project a line through the candidate area bisecting the interior angle of the two sides of the “L” found in step C.5 as shown in Figure 9. Define the two equal areas formed by the bisecting line as the right side and left side as viewed from the corner of the “L”.
 2. For each side, form a line called a “search line” between a point $d_{\min}$ distance from the corner along the “L” line, parallel to the other “L” side line, and extending to the bisecting line as shown in Figure 9.
 3. Move each search line away from the corner of the “L” as shown in Figure 9, lengthening each line as it expands to span its two bounding lines, the “L” line and the bisecting line. Keep each search line parallel to the other “L” side line. As each side is moved by an image pixel, plot the sum of number of black/white and white/black transitions multiplied by the length of the longest “L”

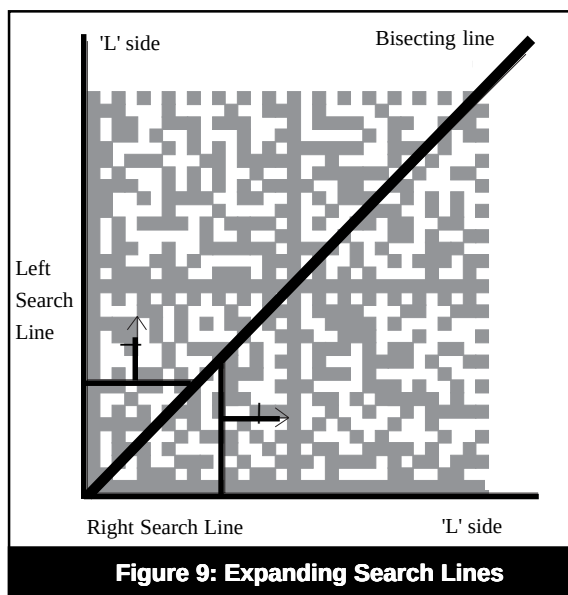


Figure 9: Expanding Search Lines

side divided by the current length of the search line measured between the two bounding lines:

$$T = (\text{number of transitions}) / (\text{"L" max line length} / (\text{search line length})).$$

This formula normalizes T to keep it from increasing because the line lengthens. Continue to calculate the T values until the search line is longer than the longest axis of the candidate area plus 50%.

4. Form a plot of the T values for each side, where the Y axis is the T value and the X axis is the search line's distance from the corner of the "L". A sample plot is shown in Figure 10.
5. Starting from the T value with the smallest X in the right side's plot and then increasing X, find the first instance of a descending line where the T value at the valley is less than 15% of the peak's value. If the peak or valley in the plot has a flat plateau or floor, select the peak or valley point closest to the descending line in the plot. The search line at the peak may correspond to an alternating finder pattern side. At the valley, the search line may correspond to the solid dark interior line or a light quiet zone.

6. Find a peak and valley in the left side's plot which most nearly matches the right peak and valley X values. If either of the selected left side peak or valley X values differ by more than 15% from the equivalent right side peak or valley X values, discard the right side peak and valley and continue searching from step D.5 for the next peak and valley. The 15% specifies the maximum allowed foreshortening.
7. The right side's valley search line, the left side's valley search line, and the two sides of the "L" outline a possible symbol's data region. Process the data region according to step E. If the decode fails, discard the right side peak and valley and continue searching from step D.5 for the next peak and valley.

- E. For each of the two sides of the alternating pattern, find the line passing through the center of the alternating light and dark modules:
 1. For each side, form a rectangular region bounded by the side's peak and valley search lines as the longer two sides of the rectangle, and the "L" side and the other side's valley search line as the shorter two sides, as shown in Figure 11.
 2. In each rectangular region, sort the black/white boundary points into sets which fit a straight line as in step B.1. Select the set with the most points and find the minimum distance line as in step B.1. This line should pass along the outside of the alternating pattern, shown as the "best fit line" in Figure 12.
 3. For each side, construct a line parallel to the step E.2 line which is offset toward the "L" corner by the length of the peak search line divided by twice the number of transitions in the peak search line:

$$\text{Offset} = \text{length of peak line} / (\text{number of transitions} * 2)$$

Each of the two constructed lines should correspond to the center-line or midline of the alternating module pattern on that side, see Figure 13.

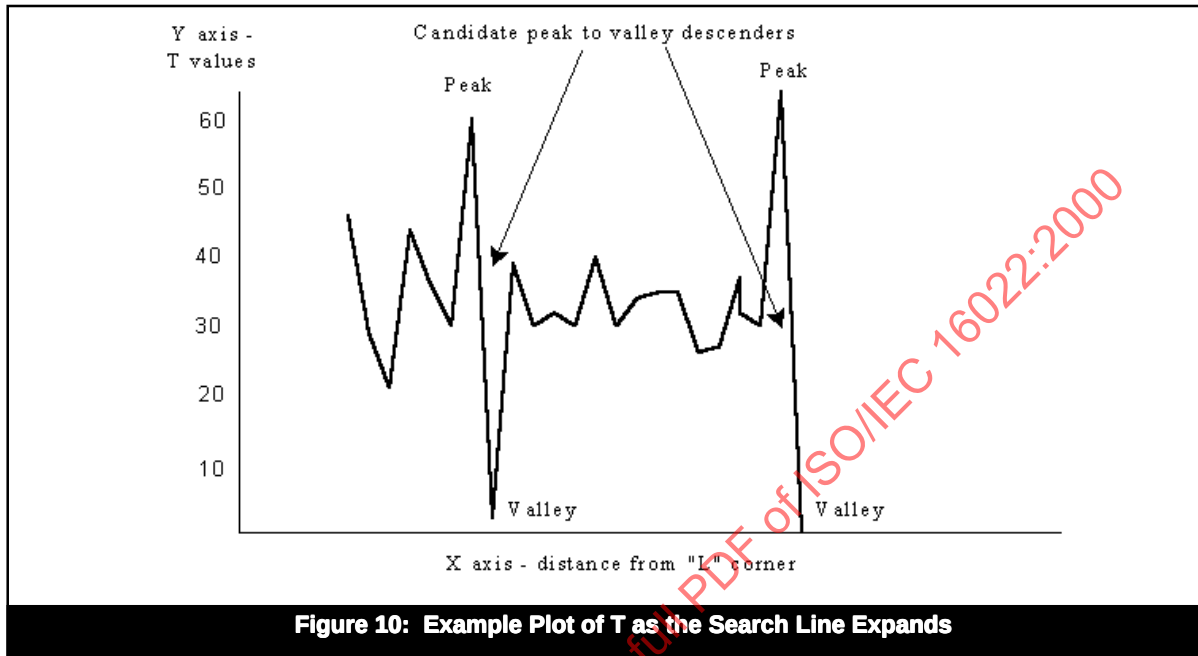


Figure 10: Example Plot of T as the Search Line Expands

- F. For each side, determine the number of data modules in the side of the square symbol or data region:
1. Bound the alternating pattern mid-line constructed in step E.3 by the adjacent "L" line and the other alternating pattern mid-line from step E.3. Call the length of this line M_d (see Figure 11).
 2. Along each bounded mid-line, measure the edge-to-edge distances between all the similar edges (note that this will include measurements across all adjacent sets of 2, 4, 6, ... modules).
 3. Sort the edge-to-edge measurements into sets where the edge-to-edge distances within each set differ by less than plus or minus 25% from the geometric mean of the set's two extremes.
 4. Calculate the average of the largest set of distances for the side. Call the average E_{avg} .
 5. The calculated number of data modules "dm" is defined by the formula:

$$dm = ((M_d * 2) / E_{avg}) - 1.5$$
 where "dm" is rounded to the nearest integer value.
 6. If the number of edge-to-edge measurements in the set is greater than "dm", delete the excess number from the edge-to-edge measurements found in step F.2 which differ most from E_{avg} . Recalculate E_{avg} and a new "dm" according to steps F.4 and F.5 using the remaining edge-to-edge measurements in the set. This should remove any spurious short and long edge-to-edge measurements caused by spots and voids.
 7. If "dm" differs for the two sides, discard the right side peak and valley and continue searching from step D.5 for the next peak and valley. Otherwise, dm is the size of the square data region.
- G. For each side, find the center points of the alternating pattern modules:
1. Divide the M_d by the sum of "dm" and 1.5 to form the module pitch:

$$\text{Module pitch} = M_d / (dm + 1.5).$$
 2. Calculate "dm" module centers along the step E.3 line with the first being 1.5 times the module pitch from the "L" side and each successive center spaced one module pitch further along the step E.3 line.
- H. Plot the data module sampling grid in the data region by extending the alternating pattern

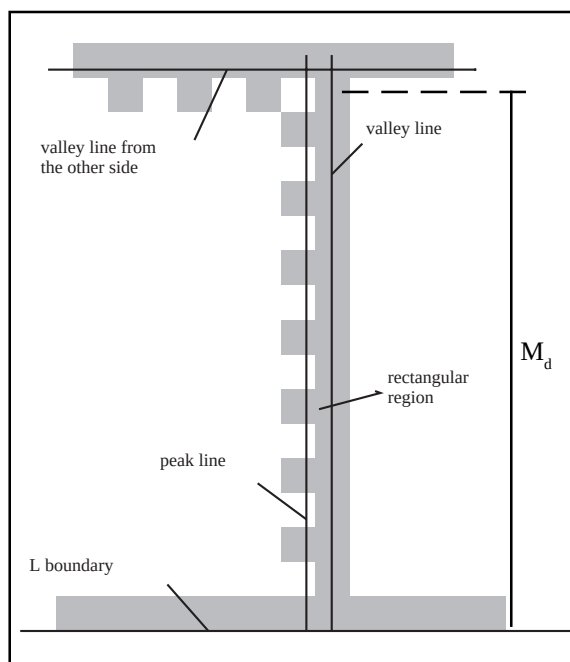


Figure 11: Rectangular Region Construction

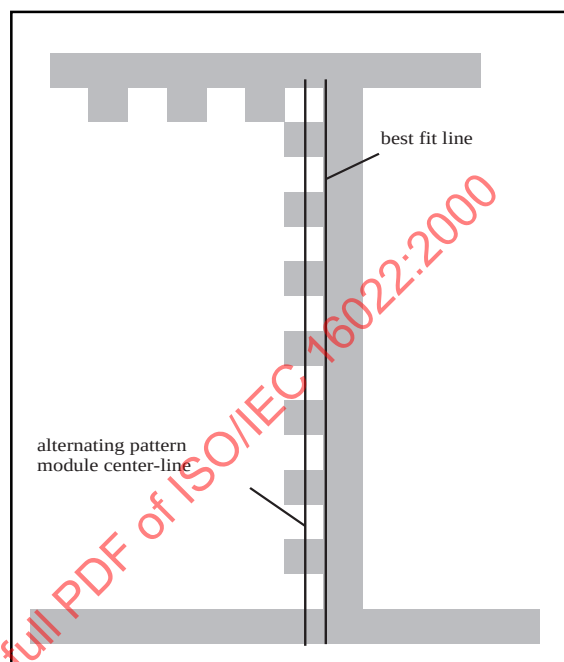


Figure 12: Alternating Module Center-line

module centers:

1. Extend each side's step E.3 midline and the opposite side's "L" line to form the vanishing point of the two nearly parallel or parallel extended lines.
 2. Extend rays from each vanishing point passing through the step G module centers of the nearly perpendicular step E.3 line.
 3. The intersection of the two sets of nearly perpendicular rays should correspond to the centers of the data modules in the data region, as shown in Figure 13.
- I. Continue to fill in the remaining data regions.
1. When a data region is processed, form a new "L" for the next data section to the "left" or "above" using one of two processes:
 - a. If the new data region is still bounded on one side by the original "L" from procedure B, repeat from procedure C to process the new data region using the selected set of points from step E.2 and the set of points on the "L" from step B.2 which lie beyond the step E.2 line.
 - b. If the new data region is bounded on two sides by data regions, repeat from

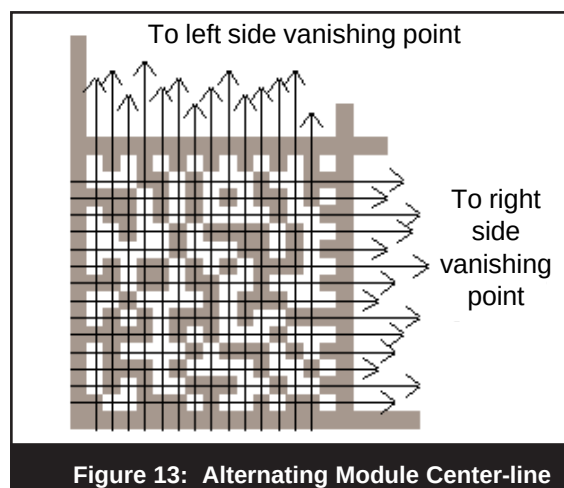


Figure 13: Alternating Module Center-line

procedure C to process the new data region using the selected set of points from step E.2 for each data region which are adjacent and bound the new region on two sides

2. If a data region does not match the number of modules in previously processed regions trim the symbol to the largest number of regions which correspond to a legal symbol.

3. Decode the symbol with its one or more data regions starting with procedure K.
- J. Find the data sections of a rectangular symbol.
1. For each side of the “L” move a line perpendicular to the side and scanning along the length of the other side of the “L”. As each side is moved by a pixel, plot the sum of number of black/white and white/black transitions:
$$T = (\text{number of transitions}) \times (\text{“L” max line length}) / (\text{search line length}).$$

Continue until the parallel line moves further than the perpendicular leg of the “L” plus 10%.

 - 2. Starting from the origin of the plot, for each direction, find the first instance of a descending line where the T value at the valley is less than 15% of the peak’s value. If the peak or valley in the plot has a flat plateau or floor, select the peak or valley point closest to the descending line in the plot. The valley line at this point may form a side of a symbol or data region.
 - 3. Find the alternating pattern lines for each side of the region similar to procedure E.
 - 4. Plot the module sample grid in the data region or symbol as in procedures F, G, and H. Skip step F.6 which requires that the region is square.
 - 5. If the data region defined is not a valid rectangular symbol, try to form a new data region using further valid peak to valley plot transitions.
 - 6. Process any additional regions as in procedure I.
 - 7. If a valid data region or two regions are detected, attempt to decode the symbol as in procedures K and L. If the region(s) were not valid or the decode fails, disregard the candidate area.
- K. If the number of data modules is even or the symbol forms a valid rectangular symbol, decode the symbol using Reed-Solomon error correction:
1. Sample the data modules at their predicted centers. Black at the center is a one and white is a zero.
 2. Convert the eight module samples in the defined codeword patterns into 8-bit symbol character values.
 3. Apply Reed-Solomon error correction to the symbol character values.
 4. Decode the symbol characters into data characters according to the specified encodation schemes.
- L. Otherwise the number of data modules is odd, so decode the symbol using convolution code error correction:
1. Sample the data modules at their predicted centers. Black at the center is a one and white is a zero.
 2. Apply the black/white balancing mask.
 3. Use the bit ordering table to convert the data into a bit stream.
 4. Apply the appropriate convolution code error correction.
 5. Convert the bit stream to data characters according to the encodation scheme specified.
 6. Verify that the CRC is correct.
- M. If a successful decode has not resulted, after all of the candidate regions have been processed, reverse the black and white states and reapply the entire algorithm to the reversed image.

10 User Guidelines

10.1 Human Readable Interpretation

Because Data Matrix symbols are capable of encoding thousands of characters, a human readable interpretation of the data characters may not be practical. As an alternative, descriptive text rather than the encoded text may accompany the symbol. The character size and font are not specified, and the message may be printed anywhere in the area surrounding the symbol. The human readable interpretation should not interfere with the symbol itself nor the quiet zones.

10.2 Autodiscrimination Capability

Data Matrix can be used in an autodiscrimination environment with a number of other symbologies. (See Annexe U).

10.3 System Consideration

Data Matrix application must be viewed as a total system solution (see Annexe V).

11 Transmitted Data

This section describes the standard transmission protocol for compliant readers. These readers may be programmable to support other transmission options.

All encoded data characters are included in the data transmission. The symbology control characters and error correction characters are not transmitted. More complex interpretations are addressed below.

11.1 Protocol for FNC1 (ECC 200 Only)

When FNC1 appears in the first symbol character position (or in the fifth symbol character position of the first symbol of a Structured Append), it shall signal that the data conforms to the UCC/EAN Application Identifier standard format. FNC1 in any other later position acts as a field separator. Transmission of symbology identifiers shall be enabled. The first FNC1 shall not be represented in the transmitted data, although its presence is indicated by the use of the appropriate option value (2) in the symbology identifier (see Section 11.5). When used as a field separator, FNC1 shall be represented in the transmitted message by the ASCII character <GS> (ASCII value 29).

11.2 Protocol for FNC1 in the Second Position

When FNC1 is in the second symbol character position (or in the sixth symbol character position of the first symbol of a Structured Append), it shall signal that the data conform to a particular industry standard format.

Transmission of symbology identifiers shall be enabled. The first FNC1 shall not be represented in the transmitted data, although its presence is indicated by the use of the appropriate option value (3) in the symbology identifier (see Section 11.5). The data encoded in the first symbol character shall be transmitted as normal at the beginning of the data. When used as a field separator, FNC1 shall be represented in the transmitted message by the ASCII character <GS> (ASCII value 29).

11.3 Protocol for Macro Characters in the First Position (ECC 200 only)

This protocol is used to encode two specific message headers and trailers in an abbreviated manner in ECC 200 symbols.

When a Macro character is in the first position a preamble and postamble shall be transmitted. If the first symbol character is 236 (i.e. encoding Macro 05), then the preamble $[>^R_5 05^G_S]$ shall precede the encoded data that follows it. If the first symbol character is 237 (i.e. encoding Macro 06), then the preamble $[>^R_5 06^G_S]$ shall precede the encoded data that follows it. The postamble $^R_5 E_{OT}$ shall be transmitted after the data in both cases.

11.4 Protocol for ECIs (ECC 200 Only)

In systems where ECIs are supported, the use of a symbology identifier prefix is required with every transmission. Whenever an ECI codeword is encountered, it shall be transmitted as the escape character 92_{DEC} (or $5C_{HEX}$), which represents the character “\” (backslash or reverse solidus) in the default interpretation. The next codeword(s) are converted into a 6-digit value, inverting the rules defined in Table 10. The 6-digit value is transmitted as the appropriate ASCII values (48-57).

Application software recognizing \nnnnnn should interpret all subsequent characters as being from the ECI defined by the 6-digit sequence. This interpretation remains in effect until the end of the encoded data or until another ECI sequence is encountered.

If the backslash (byte 92_{DEC}) needs to be used as encoded data, transmission shall be as follows. Whenever (ASCII 92_{DEC}) occurs as data, two bytes of that value shall be transmitted, thus a single occurrence is always an escape character and a double occurrence indicates true data.

Example:

Encoded data: A\\B\C

Transmission: A\\B\C

Use of the symbology identifier assures that the application can correctly interpret the escape character.

11.5 Symbology Identifier

EN796 provides a standard procedure for reporting the symbology which has been read, together with options set in the decoder and special features encountered in the symbol.

Once the structure of the data (including the use of any ECI) has been identified, the appropriate symbology identifier should be added by the decoder as a preamble to the transmitted data. The symbology identifier is required if ECIs appear anywhere in the symbol, or if FNC1 is used as defined in 11.1 or 11.2. See Annex P for the symbology identifier and option values which apply to Data Matrix.

11.6 Transmitted Data Example

In this example, the two-character message “¶ Ж” is to be encoded in ECC 200, using the ASCII encodation scheme. “¶” is represented by a byte value of 182 in Data Matrix's default character set (ECI 000003, which is equivalent to ISO 8859-1). “Ж” is a Cyrillic character not available in ECI 000003, but which can be represented in ISO 8859-5 (ECI 000007) by the same byte value of 182. The complete message can therefore be represented by inserting a switch to ECI 000007 after the first character, as follows:

The symbol encodes the message <¶> <Switch to ECI 000007> <Ж> using the following series of Data Matrix codewords (note that an Upper Shift character, followed by a codeword of value 55, encodes a byte value of 182; further note that ECIs are encoded in Data Matrix as the ECI number plus one):

[Upper Shift] [55] [ECI] [8] [Upper Shift] [55],
with decimal values of
[235], [55], [241], [8], [235], [55].

The decoder transmits the following bytes (including the symbology identifier prefix with an option value of 4, which indicates use of the ECI protocol):

93, 100, 52, 182, 92, 48, 48, 48, 48, 48, 55, 182
which, if viewed entirely in the default
interpretation, would appear graphically as:

]d4¶\000007¶

Note that the decoder is responsible for signalling the switch to ECI 000007, but not for interpreting

the result.

ECI-aware software in the receiving application would delete the ECI escape sequence \000007, and the Cyrillic character “Ж” would be represented in a system-dependent manner (eg, by changing the font in a desktop-publishing file). The final result would match the original message of “¶ Ж”.

Annexe A (Normative)

ECC 000 - 140 Symbol Attributes

A.1 ECC 000

Symbol Size*		Data Region Size		Numeric Capacity	Alphanum Capacity	8-bit byte Capacity	Error Correction Overhead %	% Correctable
Row	Col	Row	Col					
9	9	7	7	3	2	1	0.0	0.0
11	11	9	9	12	8	5	0.0	0.0
13	13	11	11	24	16	10	0.0	0.0
15	15	13	13	37	25	16	0.0	0.0
17	17	15	15	53	35	23	0.0	0.0
19	19	17	17	72	48	31	0.0	0.0
21	21	19	19	92	61	40	0.0	0.0
23	23	21	21	115	76	50	0.0	0.0
25	25	23	23	140	93	61	0.0	0.0
27	27	25	25	168	112	73	0.0	0.0
29	29	27	27	197	131	86	0.0	0.0
31	31	29	29	229	153	100	0.0	0.0
33	33	31	31	264	176	115	0.0	0.0
35	35	33	33	300	200	131	0.0	0.0
37	37	35	35	339	226	148	0.0	0.0
39	39	37	37	380	253	166	0.0	0.0
41	41	39	39	424	282	185	0.0	0.0
43	43	41	41	469	313	205	0.0	0.0
45	45	43	43	500	345	226	0.0	0.0
47	47	45	45	560	378	248	0.0	0.0
49	49	47	47	596	413	271	0.0	0.0

A.2 ECC 050

Symbol Size*		Data Region Size		Numeric Capacity	Alphanum Capacity	8-bit byte Capacity	Error Correction Overhead %	% Correctable
Row	Col	Row	Col					
11	11	9	9	1	1	**	25.0	2.8
13	13	11	11	10	6	4	25.0	2.8
15	15	13	13	20	13	9	25.0	2.8
17	17	15	15	32	21	14	25.0	2.8
19	19	17	17	46	30	20	25.0	2.8
21	21	19	19	61	41	27	25.0	2.8
23	23	21	21	78	52	34	25.0	2.8
25	25	23	23	97	65	42	25.0	2.8
27	27	25	25	118	78	51	25.0	2.8
29	29	27	27	140	93	61	25.0	2.8
31	31	29	29	164	109	72	25.0	2.8
33	33	31	31	190	126	83	25.0	2.8
35	35	33	33	217	145	95	25.0	2.8
37	37	35	35	246	164	108	25.0	2.8
39	39	37	37	277	185	121	25.0	2.8
41	41	39	39	310	206	135	25.0	2.8
43	43	41	41	344	229	150	25.0	2.8
45	45	43	43	380	253	166	25.0	2.8
47	47	45	45	418	278	183	25.0	2.8
49	49	47	47	457	305	200	25.0	2.8

* Excluding quiet zone

**Note: This combination is not possible

A.3 ECC 080

Symbol Row	Size* Col	Data Row	Region Col	Size	Numeric Capacity	Alphanum Capacity	8-bit byte Capacity	Error Correction Overhead %	% Correctable
13	13	11	11		4	3	2	33.3	5.5
15	15	13	13		13	9	6	33.3	5.5
17	17	15	15		24	16	10	33.3	5.5
19	19	17	17		36	24	16	33.3	5.5
21	21	19	19		50	33	22	33.3	5.5
23	23	21	21		65	43	28	33.3	5.5
25	25	23	23		82	54	36	33.3	5.5
27	27	25	25		100	67	44	33.3	5.5
29	29	27	27		120	80	52	33.3	5.5
31	31	29	29		141	94	62	33.3	5.5
33	33	31	31		164	109	72	33.3	5.5
35	35	33	33		188	125	82	33.3	5.5
37	37	35	35		214	143	94	33.3	5.5
39	39	37	37		242	161	106	33.3	5.5
41	41	39	39		270	180	118	33.3	5.5
43	43	41	41		301	201	132	33.3	5.5
45	45	43	43		333	222	146	33.3	5.5
47	47	45	45		366	244	160	33.3	5.5
49	49	47	47		402	268	176	33.3	5.5

A.4 ECC 100

Symbol Row	Size* Col	Data Row	Region Col	Size	Numeric Capacity	Alphanum Capacity	8-bit byte Capacity	Error Correction Overhead %	% Correctable
13	13	11	11		1	1	**	50.0	12.6
15	15	13	13		8	5	3	50.0	12.6
17	17	15	15		16	11	7	50.0	12.6
19	19	17	17		25	17	11	50.0	12.6
21	21	19	19		36	24	15	50.0	12.6
23	23	21	21		47	31	20	50.0	12.6
25	25	23	23		60	40	26	50.0	12.6
27	27	25	25		73	49	32	50.0	12.6
29	29	27	27		88	59	38	50.0	12.6
31	31	29	29		104	69	45	50.0	12.6
33	33	31	31		121	81	53	50.0	12.6
35	35	33	33		140	93	61	50.0	12.6
37	37	35	35		159	106	69	50.0	12.6
39	39	37	37		180	120	78	50.0	12.6
41	41	39	39		201	134	88	50.0	12.6
43	43	41	41		224	149	98	50.0	12.6
45	45	43	43		248	165	108	50.0	12.6
47	47	45	45		273	182	119	50.0	12.6
49	49	47	47		300	200	131	50.0	12.6

* Excluding quiet zone

**Note: This combination is not possible

A.5 ECC 140

Symbol Size*		Data Region Size		Numeric Capacity	Alphanum Capacity	8-bit byte Capacity	Error Correction Overhead %	% Correctable
Row	Col	Row	Col					
17	17	15	15	2	1	1	75.0	25.0
19	19	17	17	6	4	3	75.0	25.0
21	21	19	19	12	8	5	75.0	25.0
23	23	21	21	17	11	7	75.0	25.0
25	25	23	23	24	16	10	75.0	25.0
27	27	25	25	30	20	13	75.0	25.0
29	29	27	27	38	25	16	75.0	25.0
31	31	29	29	46	30	20	75.0	25.0
33	33	31	31	54	36	24	75.0	25.0
35	35	33	33	64	42	28	75.0	25.0
37	37	35	35	73	49	32	75.0	25.0
39	39	37	37	84	56	36	75.0	25.0
41	41	39	39	94	63	41	75.0	25.0
43	43	41	41	106	70	46	75.0	25.0
45	45	43	43	118	78	51	75.0	25.0
47	47	45	45	130	87	57	75.0	25.0
49	49	47	47	144	96	63	75.0	25.0

* Excluding quiet zone

*Annexe B (Normative) These tables are available in electronic format
ECC 000 - 140 Data Module Placement Grids*

NOTE: module position 0 is the MSB of the randomized bit stream

7x7 data:

```

2 45 10 38 24 21 1
12 40 26 5 33 19 47
22 31 29 15 43 8 36
34 20 48 13 41 27 6
44 9 37 23 17 30 16
39 25 4 32 18 46 11
0 28 14 42 7 35 3

```

9x9 data:

```

2 19 55 10 46 28 64 73 1
62 17 53 35 71 8 80 44 26
49 31 67 4 76 40 22 58 13
69 6 78 42 24 60 15 51 33
74 38 20 56 11 47 29 65 37
25 61 16 52 34 70 7 79 43
12 48 30 66 63 75 39 21 57
32 68 5 77 41 23 59 14 50
0 72 36 18 54 9 45 27 3

```

11x11 data:

```

2 26 114 70 15 103 59 37 81 4 1
117 73 18 106 62 40 84 7 95 51 29
12 100 56 34 78 92 89 45 23 111 67
65 43 87 10 98 54 32 120 76 21 109
82 5 93 49 27 115 71 16 104 60 38
96 52 30 118 74 19 107 63 41 85 8
24 112 68 13 101 57 35 79 48 90 46
75 20 108 64 42 86 9 97 53 31 119
102 58 36 80 77 91 47 25 113 69 14
39 83 6 94 50 28 116 72 17 105 61
0 88 44 22 110 66 11 99 55 33 3

```

13x13 data:

```

2 159 29 133 81 16 120 68 42 146 94 91 1
37 141 89 24 128 76 50 154 102 11 115 63 167
83 18 122 70 44 148 96 5 109 57 161 31 135
125 73 47 151 99 8 112 60 164 34 138 86 21
40 144 92 107 105 53 157 27 131 79 14 118 66
103 12 116 64 168 38 142 90 25 129 77 51 155
110 58 162 32 136 84 19 123 71 45 149 97 6
165 35 139 87 22 126 74 48 152 100 9 113 61
132 80 15 119 67 41 145 93 55 106 54 158 28
23 127 75 49 153 101 10 114 62 166 36 140 88
69 43 147 95 4 108 56 160 30 134 82 17 121
150 98 7 111 59 163 33 137 85 20 124 72 46
0 104 52 156 26 130 78 13 117 65 39 143 3

```

15x15 data:

```

2 187 37 157 97 217 22 142 82 202 52 172 112 7 1
41 161 101 221 26 146 86 206 56 176 116 11 131 71 191
93 213 18 138 78 198 48 168 108 105 123 63 183 33 153
28 148 88 208 58 178 118 13 133 73 193 43 163 103 223
80 200 50 170 110 5 125 65 185 35 155 95 215 20 140
54 174 114 9 129 69 189 39 159 99 219 24 144 84 204
106 127 121 61 181 31 151 91 211 16 136 76 196 46 166
134 74 194 44 164 104 224 29 149 89 209 59 179 119 14
186 36 156 96 216 21 141 81 201 51 171 111 6 126 66
160 100 220 25 145 85 205 55 175 115 10 130 70 190 40
212 17 137 77 197 47 167 107 67 122 62 182 32 152 92
147 87 207 57 177 117 12 132 72 192 42 162 102 222 27
199 49 169 109 4 124 64 184 34 154 94 214 19 139 79
173 113 8 128 68 188 38 158 98 218 23 143 83 203 53
0 120 60 180 30 150 90 210 15 135 75 195 45 165 3

```

17x17 data:

```

2 69 205 35 171 103 239 18 154 86 222 52 188 120 256 273 1
220 50 186 118 254 33 169 101 237 67 203 135 271 16 288 152 84
178 110 246 25 161 93 229 59 195 127 263 8 280 144 76 212 42
250 29 165 97 233 63 199 131 267 12 284 148 80 216 46 182 114
157 89 225 55 191 123 259 4 276 140 72 208 38 174 106 242 21
235 65 201 133 269 14 286 150 82 218 48 184 116 252 31 167 99
193 125 261 6 278 142 74 210 40 176 108 244 23 159 91 227 57
265 10 282 146 78 214 44 180 112 248 27 163 95 231 61 197 129
274 138 70 206 36 172 104 240 19 155 87 223 53 189 121 257 137
83 219 49 185 117 253 32 168 100 236 66 202 134 270 15 287 151
41 177 109 245 24 160 92 228 58 194 126 262 7 279 143 75 211
113 249 28 164 96 232 62 198 130 266 11 283 147 79 215 45 181
20 156 88 224 54 190 122 258 255 275 139 71 207 37 173 105 241
98 234 64 200 132 268 13 285 149 81 217 47 183 115 251 30 166
56 192 124 260 5 277 141 73 209 39 175 107 243 22 158 90 226
128 264 9 281 145 77 213 43 179 111 247 26 162 94 230 60 196
0 272 136 68 204 34 170 102 238 17 153 85 221 51 187 119 3

```

19x19 data:

```

2 82 234 44 348 196 120 272 25 329 177 101 253 63 215 139 291 6 1
239 49 353 201 125 277 30 334 182 106 258 68 220 144 296 11 315 163 87
343 191 115 267 20 324 172 96 248 58 210 134 286 310 305 153 77 229 39
132 284 37 341 189 113 265 75 227 151 303 18 322 170 94 246 56 360 208
28 332 180 104 256 66 218 142 294 9 313 161 85 237 47 351 199 123 275
185 109 261 71 223 147 299 14 318 166 90 242 52 356 204 128 280 33 337
251 61 213 137 289 4 308 156 80 232 42 346 194 118 270 23 327 175 99
225 149 301 16 320 168 92 244 54 358 206 130 282 35 339 187 111 263 73
292 7 311 159 83 235 45 349 197 121 273 26 330 178 102 254 64 216 140
316 164 88 240 50 354 202 126 278 31 335 183 107 259 69 221 145 297 12
78 230 40 344 192 116 268 21 325 173 97 249 59 211 135 287 158 306 154
55 359 207 131 283 36 340 188 112 264 74 226 150 302 17 321 169 93 245
198 122 274 27 331 179 103 255 65 217 141 293 8 312 160 84 236 46 350
279 32 336 184 108 260 70 222 146 298 13 317 165 89 241 53 355 203 127
326 174 98 250 60 212 136 288 285 307 155 79 231 41 345 193 117 269 22
110 262 72 224 148 300 15 319 167 91 243 53 357 205 129 281 34 338 186
62 214 138 290 5 309 157 81 233 43 347 195 119 271 24 328 176 100 252
143 295 10 314 162 86 238 48 352 200 124 276 29 333 181 105 257 67 219
0 304 152 76 228 38 342 190 114 266 19 323 171 95 247 57 209 133 3

```

21x21 data:

```

2 88 424 256 46 382 214 130 298 25 361 193 109 277 67 403 235 151 319 4 1
437 269 59 395 227 143 311 38 374 206 122 290 80 416 248 164 332 17 353 185 101
49 385 217 133 301 28 364 196 112 280 70 406 238 154 322 7 343 175 91 427 259
222 138 306 33 369 201 117 285 75 411 243 159 327 12 348 180 96 432 264 54 390
295 22 358 190 106 274 64 400 232 148 316 340 337 169 85 421 253 43 379 211 127
377 209 125 293 83 419 251 167 335 20 356 188 104 440 272 62 398 230 146 314 41
115 283 73 409 241 157 325 10 346 178 94 430 262 52 388 220 136 304 31 367 199
78 414 246 162 330 15 351 183 99 435 267 57 393 225 141 309 36 372 204 120 288
236 152 320 5 341 173 89 425 257 47 383 215 131 299 26 362 194 110 278 68 404
333 18 354 186 102 438 276 60 396 228 144 312 39 375 207 123 291 81 417 249 165
344 176 92 428 260 50 386 218 134 302 29 365 197 113 281 71 407 239 155 323 8
97 433 265 55 391 223 139 307 34 370 202 118 286 76 412 244 160 328 13 349 181
254 44 380 212 128 296 23 359 191 107 275 65 401 233 149 317 172 338 170 86 422
397 229 145 313 40 376 208 124 292 82 418 250 166 334 19 355 187 103 439 271 61
135 303 30 366 198 114 282 72 408 240 156 324 9 345 177 93 429 261 51 387 219
35 371 203 119 287 77 413 245 161 329 14 350 182 98 434 266 56 392 224 140 308
192 108 276 66 402 234 150 318 315 339 171 87 423 255 45 381 213 129 297 24 360
289 79 415 247 163 331 16 352 184 100 436 268 58 394 226 142 310 37 373 205 121
405 237 153 321 6 342 174 90 426 258 48 384 216 132 300 27 363 195 111 279 69
158 326 11 347 179 95 431 263 53 389 221 137 305 32 368 200 116 284 74 410 242
0 336 168 84 420 252 42 378 210 126 294 21 357 189 105 273 63 399 231 147 3

```

23x23 data:

2 102 470 286 56 424 240 148 516 332 33 401 217 125 493 309 79 447 263 171 355 10 1
 476 292 62 430 246 154 522 338 39 407 223 131 499 315 85 453 269 177 361 16 384 200 108
 50 418 234 142 510 326 27 395 211 119 487 303 73 441 257 165 349 4 372 188 96 464 280
 249 157 525 341 42 410 226 134 502 318 88 456 272 180 364 19 387 203 111 479 295 65 433
 513 329 30 398 214 122 490 306 76 444 260 168 352 7 375 191 99 467 283 53 421 237 145
 36 404 220 128 496 312 82 450 266 174 358 13 381 197 105 473 289 59 427 243 151 519 335
 208 116 484 300 70 438 254 162 346 378 369 185 93 461 277 47 415 231 139 507 323 24 392
 505 321 91 459 275 183 367 22 390 206 114 482 298 68 436 252 160 528 344 45 413 229 137
 80 448 264 172 356 11 379 195 103 471 287 57 425 241 149 517 333 34 402 218 126 494 310
 270 178 362 17 385 201 109 477 293 63 431 247 155 523 339 40 408 224 132 500 316 86 454
 350 5 373 189 97 465 281 51 419 235 143 511 327 28 396 212 120 488 304 74 442 258 166
 388 204 112 480 296 66 434 250 158 526 342 43 411 227 135 503 319 89 457 273 181 365 20
 100 468 284 54 422 238 146 514 330 31 399 215 123 491 307 77 445 261 169 353 8 376 192
 290 60 428 244 152 520 336 37 405 221 129 497 313 83 451 267 175 359 14 382 198 106 474
 416 232 140 508 324 25 393 209 117 485 301 71 439 255 163 347 194 370 186 94 462 278 48
 159 527 343 44 412 228 136 504 320 90 458 274 182 366 21 389 205 113 481 297 67 435 251
 331 32 400 216 124 492 308 78 446 262 170 354 9 377 193 101 469 285 55 423 239 147 515
 406 222 130 498 314 84 452 268 176 360 15 383 199 107 475 291 61 429 245 153 521 337 38
 118 486 302 72 440 256 164 348 345 371 187 95 463 279 49 417 233 141 509 325 26 394 210
 317 87 455 271 179 363 18 386 202 110 478 294 64 432 248 156 524 340 41 409 225 133 501
 443 259 167 351 6 374 190 98 466 282 52 420 236 144 512 328 29 397 213 121 489 305 75
 173 357 12 380 196 104 472 288 58 426 242 150 518 334 35 403 219 127 495 311 81 449 265
 0 368 184 92 460 276 46 414 230 138 506 322 23 391 207 115 483 299 69 437 253 161 3

25x25 data:

2 603 103 503 303 53 453 253 153 553 353 28 428 228 128 528 328 78 478 278 178 578 378 375 1
 123 523 323 73 473 273 173 573 373 48 448 248 148 548 348 98 498 298 198 598 398 23 423 223 623
 311 61 461 261 161 561 361 36 436 236 136 536 336 86 486 286 186 586 386 11 411 211 611 111 511
 467 267 167 567 367 42 442 242 142 542 342 92 492 292 192 592 392 17 417 217 617 117 517 317 67
 155 555 355 30 430 230 130 530 330 80 480 280 180 580 380 5 405 205 605 105 505 305 55 455 255
 370 45 445 245 145 545 345 95 495 295 195 595 395 20 420 220 620 120 520 320 70 470 270 170 570
 433 233 133 533 333 83 483 283 183 583 383 8 408 208 608 108 508 308 58 458 258 158 558 358 33
 139 539 339 89 489 289 189 589 389 14 414 214 614 114 514 314 64 464 264 164 564 364 39 439 239
 326 76 476 276 176 576 376 403 401 201 601 101 501 301 51 451 251 151 551 351 26 426 226 126 526
 499 299 199 599 399 24 424 224 624 124 524 324 74 474 274 174 574 374 49 449 249 149 549 349 99
 187 587 387 12 412 212 612 112 512 312 62 462 262 162 562 362 37 437 237 137 537 337 87 487 287
 393 18 418 218 618 118 518 318 68 468 268 168 568 368 43 443 243 143 543 343 93 493 293 193 593
 406 206 606 106 506 306 56 456 256 156 556 356 31 431 231 131 531 331 81 481 281 181 581 381 6
 621 121 521 321 71 471 271 171 571 371 46 446 246 146 546 346 96 496 296 196 596 396 21 421 221
 509 309 59 459 259 159 559 359 34 434 234 134 534 334 84 484 284 184 584 384 9 409 209 609 109
 65 465 265 165 565 365 40 440 240 140 540 340 90 490 290 190 590 390 15 415 215 615 115 515 315
 252 152 552 352 27 427 227 127 527 327 77 477 277 177 577 377 203 402 202 602 102 502 302 52 452
 572 372 47 447 247 147 547 347 97 497 297 197 597 397 22 422 222 622 122 522 322 72 472 272 172
 35 435 235 135 535 335 85 485 285 185 585 385 10 410 210 610 110 510 310 60 460 260 160 560 360
 241 141 541 341 91 491 291 191 591 391 16 416 216 616 116 516 316 66 466 266 166 566 366 41 441
 529 329 79 479 279 179 579 379 4 404 204 604 104 504 304 54 454 254 154 554 354 29 429 229 129
 94 494 294 194 594 394 19 419 219 619 119 519 319 69 469 269 169 569 369 44 444 244 144 544 344
 282 182 582 382 7 407 207 607 107 507 307 57 457 257 157 557 357 32 432 232 132 532 332 82 482
 588 388 13 413 213 613 113 513 313 63 463 263 163 563 363 38 438 238 138 538 338 88 488 288 188
 0 400 200 600 100 500 300 50 450 250 150 550 350 25 425 225 125 525 325 75 475 275 175 575 3

27x27 data:

2 658 118 550 334 64 496 280 712 172 604 388 37 469 253 685 145 577 361 91 523 307 199 631 415 10 1
 125 557 341 71 503 287 719 179 611 395 44 476 260 692 152 584 368 98 530 314 206 638 422 17 449 233 665
 327 57 489 273 705 165 597 381 30 462 246 678 138 570 354 84 516 300 192 624 408 405 435 219 651 111 543
 511 295 727 187 619 403 52 484 268 700 160 592 376 106 538 322 214 646 430 25 457 241 673 133 565 349 79
 714 174 606 330 39 471 255 687 147 579 363 93 525 309 201 633 417 12 444 228 660 120 552 336 66 498 282
 613 397 46 478 262 694 154 586 370 100 532 316 208 640 424 19 451 235 667 127 559 343 73 505 289 721 181
 32 464 248 680 140 572 356 86 518 302 194 626 410 5 437 221 653 113 545 329 59 491 275 707 167 599 383
 265 697 157 589 373 103 535 319 211 643 427 22 454 238 670 130 562 346 76 508 292 724 184 616 400 49 481
 143 575 359 89 521 305 197 629 413 8 440 224 656 116 548 332 62 494 278 710 170 602 386 35 467 251 683
 366 96 528 312 204 636 420 15 447 231 663 123 555 339 69 501 285 717 177 609 393 42 474 258 690 150 582
 514 298 190 622 406 442 433 217 649 109 541 325 55 487 271 703 163 595 379 28 460 244 676 136 568 352 82
 215 647 431 26 458 242 674 134 566 350 80 512 296 728 188 620 404 53 485 269 701 161 593 377 107 539 323
 418 13 445 229 661 121 553 337 67 499 283 715 175 607 391 40 472 256 688 148 580 364 94 526 310 202 634
 452 236 668 128 560 344 74 506 290 722 182 614 398 47 479 263 695 155 587 371 101 533 317 209 641 425 20
 654 114 546 330 60 492 276 708 168 600 384 33 465 249 681 141 573 357 87 519 303 195 627 411 6 438 222
 563 347 77 509 293 725 185 617 401 50 482 266 698 158 590 374 104 536 320 212 644 428 23 455 239 671 131
 63 495 279 711 171 603 387 36 468 252 684 144 576 360 90 522 306 198 630 414 9 441 225 657 117 549 333
 286 718 178 610 394 43 475 259 691 151 583 367 97 529 313 205 637 421 16 448 232 664 124 556 340 70 502
 164 596 380 29 461 245 677 137 569 353 83 515 299 191 623 407 226 434 218 650 110 542 326 56 488 272 704
 402 51 483 267 699 159 591 375 105 537 321 213 645 429 24 456 240 672 132 564 348 78 510 294 726 186 618
 770 254 686 146 578 362 92 524 308 200 632 416 11 443 227 659 119 551 335 65 497 281 713 173 605 389 38
 693 153 585 369 99 531 315 207 639 423 18 450 234 666 126 558 342 72 504 288 720 180 612 396 45 477 261
 571 355 85 517 301 193 625 409 4 436 220 652 112 544 328 58 490 274 706 166 598 382 31 463 247 679 139
 102 534 318 210 642 426 21 453 237 669 129 561 345 75 507 291 723 183 615 399 48 480 264 696 156 588 372
 304 196 628 412 7 439 223 655 115 547 331 61 493 277 709 169 601 385 34 466 250 682 142 574 358 88 520
 635 419 14 446 230 662 122 554 338 68 500 284 716 176 608 392 41 473 257 689 149 581 365 95 527 311 203
 0 432 216 648 108 540 324 54 486 270 702 162 594 378 27 459 243 675 135 567 351 81 513 297 189 621 3

29x29 data:

2 703 123 587 355 819 65 529 297 761 181 645 413 36 500 268 732 152 616 384 94 558 326 790 210 674 442 7 1
 141 605 373 837 83 547 315 779 199 663 431 54 518 286 750 170 634 402 112 576 344 808 228 692 460 25 489 257 721
 359 823 69 533 301 765 185 649 417 40 504 272 736 156 620 388 98 562 330 794 214 678 446 11 475 243 707 127 591
 76 540 308 772 192 656 424 47 511 279 743 163 627 395 105 569 337 801 221 685 453 18 482 250 714 134 598 366 830
 293 757 177 641 409 32 496 264 728 148 612 380 90 554 322 786 206 670 438 435 467 235 699 119 583 351 815 61 525
 201 665 433 56 520 288 752 172 636 404 114 578 346 810 230 694 462 27 491 259 723 143 607 375 839 85 549 317 781
 419 42 506 274 738 158 622 390 100 564 332 796 216 680 448 13 477 245 709 129 593 361 825 71 535 303 767 187 651
 513 281 745 165 629 397 107 571 339 803 223 687 455 20 484 252 716 136 600 368 832 78 542 310 774 194 658 426 49
 730 150 614 382 92 556 324 788 208 672 440 5 469 237 701 121 585 353 817 63 527 295 759 179 643 411 34 498 266
 632 400 110 574 342 806 226 690 458 23 487 255 719 139 603 371 835 81 545 313 777 197 661 429 52 516 284 748 168
 96 560 328 792 212 676 444 9 473 241 705 125 589 357 821 67 531 299 763 183 647 415 38 502 270 734 154 618 386
 335 799 219 683 451 16 480 248 712 132 596 364 828 74 538 306 770 190 654 422 45 509 277 741 161 625 393 103 567
 204 668 436 471 465 233 697 117 581 349 813 59 523 291 755 175 639 407 30 494 262 726 146 610 378 88 552 320 784
 463 28 492 260 724 144 608 376 840 86 550 318 782 202 666 434 57 521 289 753 173 637 405 115 579 347 811 231 695
 478 246 710 130 594 362 826 72 536 304 768 188 652 420 43 507 275 739 159 623 391 101 565 333 797 217 681 449 14
 717 137 601 369 833 79 543 311 775 195 659 427 50 514 282 746 166 630 398 108 572 340 804 224 688 456 21 485 253
 586 354 818 64 528 296 760 180 644 412 35 499 267 731 151 615 383 93 557 325 789 209 673 441 6 470 238 702 122
 836 82 546 314 778 198 662 430 53 517 285 749 169 633 401 111 575 343 807 227 691 459 24 488 256 720 140 604 372
 532 300 764 184 648 416 39 503 271 735 155 619 387 97 561 329 793 213 677 445 10 474 242 706 126 590 358 822 68
 771 191 655 423 46 510 278 742 162 626 394 104 568 336 800 220 684 452 17 481 249 713 153 597 365 829 75 539 307
 640 408 31 495 263 727 147 611 379 89 553 321 785 205 669 437 239 466 234 698 118 582 350 814 60 524 292 756 176
 55 519 287 751 171 635 403 113 577 345 809 229 693 461 26 490 258 722 142 606 374 838 84 548 316 780 200 664 432
 273 737 157 621 389 99 563 331 795 215 679 447 12 476 244 708 128 592 360 824 70 534 302 766 186 650 418 41 505
 164 628 396 106 570 338 802 222 686 454 19 483 251 715 135 599 367 831 77 541 309 773 193 657 425 48 512 280 744
 381 91 555 323 787 207 671 439 4 468 236 700 120 584 352 816 62 526 294 758 178 642 410 33 497 265 729 149 613
 573 341 805 225 689 457 22 486 254 718 138 602 370 834 80 544 312 776 196 660 428 51 515 283 747 167 631 399 109
 791 211 675 443 8 472 240 704 124 588 356 820 66 530 298 762 182 646 414 37 501 269 733 153 617 385 95 559 327
 682 450 15 479 247 711 131 595 363 827 73 537 305 769 189 653 421 44 508 276 740 160 624 392 102 566 334 798 218
 0 464 232 696 116 580 348 812 58 522 290 754 174 638 406 29 493 261 725 145 609 377 87 551 319 783 203 667 3

31x31 data:

2 759 139 635 387 883 77 573 325 821 201 697 449 945 46 542 294 790 170 666 418 914 108 604 356 852 232 728 480 15 1
 147 643 395 891 85 581 333 829 203 705 457 953 54 550 302 798 178 674 426 922 116 612 364 860 240 736 488 23 519 271 767
 379 875 69 565 317 813 193 689 441 937 38 534 286 782 162 658 410 906 100 596 348 844 224 720 472 7 503 255 751 131 627
 89 585 337 833 213 709 461 957 58 554 306 802 182 678 420 926 120 616 368 864 244 740 492 27 523 275 751 151 647 399 895
 321 817 197 693 445 941 42 538 290 786 166 662 414 910 104 600 352 848 228 724 476 11 507 259 755 135 631 383 879 73 569
 205 701 453 949 50 546 298 794 174 670 422 918 112 608 360 856 236 732 484 19 515 267 763 143 639 391 887 81 577 329 825
 437 933 34 530 282 778 158 654 406 902 96 592 344 840 220 716 468 465 499 251 747 127 623 375 871 65 561 313 809 189 685
 60 556 308 804 184 680 432 928 122 618 370 866 246 742 494 29 525 277 773 153 649 401 897 91 587 339 835 215 711 463 959
 292 788 168 664 416 912 106 602 354 850 230 726 478 13 509 261 757 137 633 385 881 75 571 323 819 199 695 447 943 44 540
 176 672 424 920 114 610 362 858 238 734 486 21 517 269 765 145 641 393 889 83 579 331 827 207 703 455 951 52 548 300 796
 408 904 98 594 346 842 222 718 470 5 501 253 749 129 625 377 873 67 563 315 811 191 687 439 935 36 532 284 780 160 656
 118 614 366 862 242 738 490 25 521 273 769 149 645 397 893 87 583 335 831 211 707 459 955 56 552 304 800 180 676 428 924
 350 846 226 722 474 9 505 257 753 133 629 381 877 71 567 319 815 195 691 443 939 40 536 288 784 164 660 412 908 102 598
 234 730 482 17 513 265 761 141 637 389 885 79 575 327 823 203 699 451 947 48 544 296 792 172 668 420 916 110 606 358 854
 466 511 497 249 745 125 621 373 869 63 559 311 807 187 683 435 931 32 528 280 776 156 652 404 900 94 590 342 838 218 714
 526 278 774 154 650 402 898 32 588 340 836 216 712 464 960 61 557 309 805 185 681 433 929 123 619 371 867 247 743 495 30
 758 138 634 386 882 76 572 324 820 200 696 448 944 45 541 293 789 169 665 417 913 107 603 355 851 231 727 479 14 510 262
 642 394 890 84 580 332 828 208 704 456 952 53 549 301 797 177 673 425 921 115 611 363 859 239 735 487 22 518 270 766 146
 874 68 564 316 812 192 688 440 936 37 533 285 781 161 657 409 905 99 595 347 843 223 719 471 6 502 254 750 130 626 378
 584 336 832 212 708 460 956 57 553 305 801 181 677 429 925 119 615 367 863 243 739 491 26 522 274 770 150 646 398 894 88
 816 196 692 444 940 41 537 289 785 165 661 413 909 103 647 227 723 475 10 506 258 754 134 630 382 878 72 568 320
 700 452 948 49 545 297 793 173 669 421 917 111 607 359 855 235 731 483 18 514 266 762 142 638 390 886 80 576 328 824 204
 932 33 529 281 777 157 653 405 901 95 591 343 839 219 715 467 263 498 250 746 126 622 374 870 64 560 312 808 188 684 436
 555 307 803 183 679 431 927 121 617 369 865 245 741 493 28 524 276 772 152 648 400 896 90 586 338 834 214 710 462 958 59
 787 167 663 415 911 105 601 353 849 229 725 477 12 508 260 756 136 632 384 880 74 570 322 818 198 694 446 942 43 539 291
 671 423 919 113 609 361 857 237 733 485 20 516 268 764 144 640 392 888 82 578 330 826 206 702 454 950 51 547 299 795 175
 903 97 593 345 841 221 717 469 4 500 252 748 128 624 376 872 66 562 314 810 190 686 438 934 35 531 283 779 159 655 407
 613 365 861 241 737 489 24 520 272 768 148 644 396 892 86 582 334 830 210 706 458 954 55 551 303 799 179 675 427 923 117
 845 225 721 473 8 504 256 752 132 628 380 876 70 566 318 814 194 690 442 938 39 535 287 783 163 659 411 907 101 597 349
 729 481 16 512 264 760 140 636 388 884 78 574 326 822 202 698 450 946 47 543 295 791 171 667 419 915 109 605 357 853 233
 0 496 248 744 124 620 372 868 62 558 310 806 186 682 434 930 31 527 279 775 155 651 403 899 93 589 341 837 217 713 3

33x33 data:

2	265	793	133	661	397	925	67	595	331	859	199	727	463	991	34	562	298	826	166	694	430	958	100	628	364	892	232	760	496	1024	1057	1
824	164	692	428	956	98	626	362	890	230	758	494	1022	65	593	329	857	197	725	461	989	131	659	395	923	263	791	527	1055	32	1088	560	296
676	412	940	82	610	346	874	214	742	478	1006	49	577	313	841	181	709	445	973	115	643	379	907	247	775	511	1039	16	1072	544	280	808	148
948	90	618	354	882	222	750	486	1014	57	585	321	849	189	717	453	981	123	651	387	915	255	783	519	1047	24	1080	552	288	816	156	684	420
602	338	866	206	734	470	998	41	569	305	833	173	701	437	965	107	635	371	899	239	767	503	1031	8	1064	536	272	800	140	668	404	932	74
886	226	754	430	1018	61	589	325	853	193	721	457	985	127	655	391	919	259	787	523	1051	28	1084	556	222	820	160	688	424	952	94	622	358
738	474	1002	45	573	309	837	177	705	441	569	111	639	375	903	243	771	507	1035	12	1068	540	276	804	144	672	408	936	78	606	342	870	210
1010	53	581	317	845	185	713	449	977	119	647	383	911	251	779	515	1043	20	1076	548	284	812	152	680	416	944	86	614	350	878	218	746	482
555	301	829	169	697	433	961	103	631	367	895	235	763	499	1027	4	1060	532	268	796	136	664	400	928	70	598	334	862	202	730	466	994	37
855	195	723	459	987	129	657	393	921	261	789	525	1053	30	1086	558	294	822	162	690	426	954	96	624	360	888	228	756	492	1020	63	591	327
707	443	971	113	641	377	905	245	773	509	1037	14	1070	542	278	806	146	674	410	938	80	608	344	872	212	740	476	1004	47	575	311	839	179
979	121	649	385	913	253	781	517	1045	22	1078	550	286	814	154	682	418	946	88	616	352	880	220	748	484	1012	55	583	319	847	187	715	451
633	369	897	237	765	501	1029	6	1062	534	270	798	138	666	402	930	72	600	336	864	204	732	468	996	39	567	303	831	171	699	435	963	105
917	257	785	521	1049	26	1082	554	290	818	158	686	422	950	92	620	356	884	224	752	488	1016	59	587	323	851	191	719	455	983	125	653	389
769	505	1033	10	1066	538	274	802	142	670	406	934	76	604	340	868	208	736	472	1000	43	571	307	835	175	703	439	967	109	637	373	901	241
1041	18	1074	546	282	810	150	678	414	942	84	612	348	876	216	744	480	1008	51	579	315	843	183	711	447	975	117	645	381	909	249	777	513
1058	530	266	794	134	662	398	926	68	596	332	860	200	728	464	992	35	563	299	827	167	695	431	959	101	629	365	893	233	761	497	1025	529
295	823	163	691	427	955	97	625	361	889	229	757	493	1021	64	592	328	856	196	724	460	988	130	658	394	922	262	790	526	1054	31	1087	559
147	675	411	939	81	609	345	873	213	741	477	1005	48	576	312	840	180	708	444	972	114	642	378	906	246	774	510	1038	15	1071	543	279	807
419	947	89	617	353	881	221	749	485	1013	56	584	320	848	188	716	452	980	122	650	386	914	254	782	518	1046	23	1079	551	287	815	155	683
73	601	337	865	205	733	469	997	40	568	304	832	172	700	436	964	106	634	370	898	238	766	502	1030	7	1063	535	271	799	139	667	403	931
357	885	225	753	489	1017	60	588	324	852	192	720	456	984	126	654	390	918	258	786	522	1050	27	1083	555	291	819	159	687	423	951	93	621
209	737	473	1001	44	572	308	836	176	704	440	968	110	638	374	902	242	770	506	1034	11	1067	539	275	803	143	671	407	935	77	605	341	869
481	1009	52	580	316	844	184	712	448	976	118	646	382	910	250	778	514	1042	19	1075	547	283	811	151	679	415	943	85	613	349	877	217	745
36	564	300	828	168	696	432	960	102	630	366	894	234	762	498	1026	1023	1059	531	267	795	135	663	399	927	69	597	333	861	201	729	465	993
326	854	194	722	458	986	128	656	392	920	260	788	524	1052	29	1085	557	293	821	161	689	425	953	95	623	359	887	227	755	491	1019	62	590
178	706	442	970	112	640	376	904	244	772	508	1036	13	1069	541	277	805	145	673	409	937	79	607	343	871	211	739	475	1003	46	574	310	838
450	978	120	648	384	912	252	780	516	1044	21	1077	549	285	813	153	681	417	945	87	615	351	879	219	747	483	1011	54	582	318	846	186	714
104	632	368	896	236	764	500	1028	5	1061	533	269	797	137	665	401	929	71	599	355	863	203	731	467	995	38	566	302	830	170	698	434	962
388	916	256	784	520	1048	25	1081	553	289	817	157	685	421	949	91	619	355	883	223	751	487	1015	58	586	322	850	190	718	454	982	124	652
240	768	504	1032	9	1065	537	273	801	141	669	405	933	75	603	339	867	207	735	471	999	42	570	306	834	174	702	438	966	108	636	372	900
512	1040	17	1073	545	281	809	149	677	413	83	611	347	875	215	743	479	1007	50	578	314	842	182	710	446	974	116	644	380	908	248	776	
0	1056	528	264	792	132	609	396	924	66	594	330	858	198	726	462	990	33	561	297	825	165	693	429	957	99	627	363	891	231	759	495	3

35x35 data:

2	290	850	150	710	430	990	80	1200	640	360	920	220	780	500	1060	45	1165	605	325	885	185	745	465	1025	115	675	395	955	255	815	535	1095	10	1
859	159	739	459	919	8209	649	369	929	229	789	509	1069	54	1174	614	334	894	194	754	474	1034	124	684	404	264	824	544	1084	264	824	544	1084	264	824
701	421	981	71	1191	631	351	911	271	831	551	1111	321	881	181	741	461	1021	115	671	391	951	251	811	531	1091	10	1131	571	291	851	151	711	431	991
1014	104	1224	664	384	244	804	524	1084	69	1189	629	439	859	1049	139	699	419	979	279	839	559	1119	34	1154	594	314	874	174	734	454				
1207	647	927	227	787	507	1067	52	1172	612	332	892	192	752	472	1032	122	682	402	262	822	542	1002	17	1137	577	297	857	157	717	437	977	87		
376	936	236	796	516	1076	61	1161	621	341	901	201	761	481	1041	131	691	411	971	271	831	551	1111	26	1146	586	306	866	166	726	446	1006	96	1216	
218	778	498	1058	43	1163	623	883	183	743	463	1023	113	673	393	953	253	813	533	1093	8	1128	568	288	848	148	708	428	988	78	1198	638	358	918	
520	1060	608	328	888	188	748	468	1028	118	678	398	958	258	818	538	1098	13	1133	573	293	853	153	713	433	993	83	993	403	1203	660	380	940	240	
617	377	897	197	757	107	657	407	967	267	827	547	1027	22	1142	582	302	862	162	722	442	1002	62	1122	652	372	932	223	783	503	1063	72	1177		
767	487	1047	137	697	417	977	277	837	557	1117	32	1152	592	312	872	172	732	452	1012	102	1222	662	382	942	202	862	522	1082	67	1187	627	347	907	
1030	120	680	400	960	260	820	540	1100	15	1135	575	295	855	155	715	435	995	85	355	925	225	785	505	1065	50	1170	610	330	890	190	750	470		
689	409	969	269	829	549	1109	24	1144	584	304	864	164	724	444	1004	64	1124	654	374	934	234	794	514	1074	59	1179	639	359	919	759	479	1039	129	
951	251	811	531	1091	6	1126	566	286	846	146	706	426	986	76	1196	636	356	916	216	776	496	1056	41	1161	601	321	881	141	701	421	1111	671	391	
833	553	1113	28	1148	588	308	868	168	728	448	1008	98	1218	658	378	938	238	798	518	1078	63	1183	623	343	903	203	763	483	1043	133	693	413	973	
1096	11	1131	571	21	851	151	711	431	991	81	1201	41	361	921	221	781	501	1061	46	1166	606	326	886	146	706	426	986	76	1196	636	356	916	216	
282	842	142	702	422	982	72	1192	632	352	912	212	772	492	1052	37	1157	637	357	917	417	1077	57	1137	637	357	917	417	1077	57	1137	637	357	917	
173	738	438	1038	103	1238	663	383	943	243	803	523	1083	68	1188	628	348	908	208	768	488	1048	138	698	418	978	278	838	558	1118	33	1153	593	313	
436	996	86	1206	646	366	926	226	786	506	1066	51	1171	611	331	891	191	751	471	1031	121	681	401	261	821	541	1101	16	1136	576	296	856	156	716	
95	1215	655	375	935	235	795	515	1075	60	1180	620	340	900	200	760	480	1040	130	690	410	970	270	830	550	1110	25	1145	585	305	865	165	725	445	
637	357	917	217	777	497	1057	42	1162	602	322	882	182	742	462	1022	112	672	392	952	252	812	532	1092	7	1132	572	292	852	152	712	432	992		
939	239	799	519	1079	64	1184	624	344	904	204	764	484	1044	134	694	414	974	274	834	554	1114	29	1149	589	309	869	169	729	449	1009	99	1219		
782	502	1062	47	1167	607	327	887	187	747	467	1027	117	677	397	957	257	817	537	1097	12	1132	572	292	852	152	712	432	992	62	1182	642	362	922	
1071	56	1176	616	336	896	196	756	476	1036	126	686	406	966	266	826	546	1106	21	1141	581	301	861	161	721	441	1001	91	1211	651	371	931	231		
1158	598	318	878	178	738	458	1018	108	668	388	948	248	808	528	1088	688	408	968	268	828	548	1108	33	1153	593	313	873	513	1073	213	773	493	1053	
346	906	206	766	486	1046	136	696	416	976	276	836	556	1116	31	1151	591	311	871	491	1011	101	1221	661	381	941	241	801	521	1081	66	1186	626		
189	749	469	1029	119	679	399	959	259	819	539	1099	14	1134	574	294	854	154	714	434	994	84	1124	654	374	934	234	794	514	1074	60	1174	574		
478	1038	128	688	408	968	268	828	548	1108	23	1143	583	303	863	163	723	443	1003	93	1213	653	373	933	233	793	513	1073	58	1178	618	338	898	198	
110	670	390	950	250	810	530	1090	5	1125	565	285	845	145	705	425	985	75	1195	635	355	915	215	775	495	1055	40	1160	620	380	940	240	800		
412	972	272	832	552	1112	27	1147	587	307	867	167	727	447	1007	97	1217	657	377	937	237	797	517	1077	62	1182	642	362	902	262	762	482	1042		
254	814	534	1094	9	1129	569	289	849	149	709	429	989	79	1199	639	359	919	219	779	499	1059	44	1164	604	324	884	144	704	424	984	244	804		
543	1103	18	1138	578	298	858	158	718	438	998	88	1208	648	368	928	228	788	508	1068	53	1173	613	333	893	193	753	473	1033	123	683	403	963	263	
0	1120	560	280	840	140	700	420	980	70	1130	630	350	910	210	770	490	1050	35	1155	595	315	875	175	735	455	1015	105	665	385	945	245	805	525	

37x37 data:

2	302	894	154	1338	746	450	1042	80	1264	672	376	968	228	820	524	1116	43	1227	635	369	931	191	783	487	1079	117	1301	709	413	1005	265	857	561	1153	6																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
917	177	1361	769	473	1065	103	1287	695	399	991	251	843	527	1129	66	1250	658	362	954	214	806	510	1102	147	1079	117	1301	709	413	1005	265	857	561	1153	6																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
1343	751	455	1047	485	1269	677	381	973	233	825	529	1121	64	1232	649	356	196	788	492	1064	122	636	714	418	1010	270	862	566	1158	11	1195	603	307	899	159																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
1464	1056	94	1278	686	390	982	244	804	538	1130	57	1241	649	353	945	205	797	501	1093	131	1315	773	419	1019	279	871	575	1167	20	1204	612	316	908	168	1352	760																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
75	1259	667	373	853	223	815	519	1111	38	1222	630	334	926	186	778	482	1074	112	1296	704	408	1000	260	852	556	1148	1190	1185	593	297	889	149	1333	741	445	1037																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
702	406	998	258	850	554	1146	73	1257	665	369	961	221	813	517	1109	147	1331	739	443	1035	295	867	591	1183	36	1220	628	324	184	1368	776	480	1072	110	1294																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
980	240	832	536	1128	55	1239	647	351	943	203	795	499	1091	129	1313	721	425	1017	277	869	573	1155	18	1202	610	314	906	166	750	462	1054	92	1276	684	388																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
841	545	1137	64	1248	656	360	952	212	804	508	1100	138	1322	730	434	1026	286	878	582	1174	27	1211	613	315	175	1359	767	471	1063	101	1285	693	397	989	249																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
1119	46	1230	638	342	934	194	786	490	1082	120	1304	712	416	1008	268	860	564	1156	9	1193	601	305	899	157	1341	749	453	1045	83	1267	675	379	971	231	823	527																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
1253	661	365	957	217	809	513	1105	143	1327	735	439	1031	291	883	587	1179	32	1216	624	302	162	1346	754	458	1050	88	1272	680	384	976	236	828	532	1124	51	1235	643																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
347	939	199	791	495	1087	125	1309	717	421	1013	273	865	569	1161	14	1198	606	310	902	162	1346	754	458	1050	88	1272	680	384	976	236	828	532	1124	51	1235	643																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
408	1007	115	1296	707	411	1003	262	852	559	1151	4	1188	596	300	892	152	1336	744	448	1040	78	1262	670	374	889	393	985	245	837	541	1133	630	323	929	189	781																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
145	1329	737	441	1033	293	885	589	1181	34	1218	626	330	922	182	1366	774	478	1070	108	1292	700	404	996	256	818	552	1144	71	1255	663	367	959	219	811	515	1107																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
719	423	1015	275	867	571	1163	16	1200	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300	608	312	1300</

39x39 data:									
2	328	952	172	1420	796	484	1108	94	1342
962	182	1430	806	494	1118	104	1352	728	416
1410	786	474	1098	84	1332	708	396	1020	240
499	1212	109	1357	733	421	1045	265	1513	889
893	1337	713	401	1025	245	1493	869	557	1181
723	411	1055	255	1503	879	567	1191	60	1308
1015	235	1484	859	547	1171	40	1288	664	352
1520	896	584	1208	77	1321	705	389	1013	233
565	1189	58	1306	662	370	994	214	1462	838
672	360	984	200	1004	224	1472	848	536	1160
1009	229	1477	853	541	1165	511	1399	775	463
1457	833	521	1445	313	1379	755	443	1067	287
531	1155	541	1389	765	453	1057	277	901	589
121	1369	748	403	1057	277	901	589	121	1369
1070	290	914	602	1225	67	1265	641	329	953
1216	7	1235	631	319	943	163	1411	787	475
324	948	168	1416	792	480	1104	90	1338	714
178	1426	802	490	1114	720	1368	744	432	1056
1129	115	1363	739	427	1051	277	901	589	121
1344	702	408	1032	256	1504	880	568	1192	61
892	1404	866	554	1178	47	1295	671	1359	883
241	579	1203	72	1320	696	384	1008	228	1476
1183	52	1300	676	364	988	208	1456	832	520
354	978	158	1446	822	510	1134	120	1368	744
834	522	1146	132	1380	756	444	1068	288	912
1156	1442	1390	766	454	1078	292	922	612	1234
1370	746	434	1058	278	902	590	1214	5	1253
459	1083	303	927	615	1239	30	1278	654	342
283	907	595	1219	10	1258	634	322	946	166
605	1229	20	1268	644	332	956	176	1424	800
0	1248	624	312	936	156	1404	780	1326	78

41x41 data:

2	332	1644	988	168	1480	824	496	1152	86
1677	1021	1513	857	1328	1185	1331	745	487	1103
181	1493	837	509	1165	99	1411	755	427	1083
847	513	1175	1020	1421	785	437	1093	235	1486
1426	70	1442	1098	278	1590	834	606	1262	71
422	1078	28	1570	914	586	1242	5	1365	709
268	1580	924	596	1252	63	1375	719	391	1047
903	575	1231	42	1354	698	370	1026	206	1518
1370	717	388	1046	225	1337	881	855	1209	43
398	1055	23	1547	931	563	1129	153	1465	809
213	1527	871	543	1199	133	1445	789	461	1137
896	568	1224	58	1470	814	486	1142	322	1634
1460	804	471	1131	1024	866	640	1296	25	1337
456	1113	293	1604	948	620	1276	5	1317	661
325	1637	981	653	1309	38	1350	694	366	1678
961	63	1289	38	1330	674	346	1658	1002	482
1239	64	1396	40	1394	684	358	1670	1012	488
361	1673	1017	1509	853	525	1181	115	1427	771
997	177	1489	833	505	1141	95	1407	751	463
1499	843	513	1171	1005	1417	761	1433	1089	269
434	1350	84	1396	40	1421	1068	448	1094	506
757	423	1085	265	1577	921	993	1249	60	1372
1095	275	1587	931	532	1062	706	1382	728	454
1368	124	73	1387	711	403	1059	239	1551	895
608	1264	75	1387	711	403	1059	239	1551	895
721	393	1049	231	1584	885	357	1213	1407	1459
1028	208	1580	924	596	1252	63	1375	719	391
1353	897	569	1225	556	1471	815	1487	1143	323
540	1205	139	1453	755	467	1123	303	1615	959
785	467	1113	293	1608	948	620	1276	5	1317
1138	138	1530	924	596	1252	63	1375	719	391
1610	954	1626	1282	11	1323	67	1343	867	551
636	1229	20	1268	644	332	956	176	1424	800
0	1312	656	328	1460	794	164	1065	188	497

43x43 data:

2	359	1735	1057	847	562	1230	51	1219	101	1477	789	445	1284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740	660	1348	15	1
1746	1058	1398	1574	886	542	1230	112	1488	80	456	1832	1144	284	1660	971	617	1305	58	1434	46	402	1778	1090	230	586	918	574	1262	144	1520	882	488	1176	320	1065	740				

45x45 data:

[illegible]

47x47 data:

1	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	2
2	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	3
3	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	4
4	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	5
5	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	6
6	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	7
7	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	8
8	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	9
9	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	10
10	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	11
11	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	12
12	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244	304.1808	1056	680.2184	1432	69.1573	821	445.1949	1197	257.1761	1009	63.2137	1385	163.1667	915	539.2043	1291	351.1885	1103	727.1479	13
13	398.1502	1150	210.1714	962	586.2090	1338	116.1620	868	492.1996	1244</																				

Annexe C (Normative)

ECC 000 - 140 Character Encodation Schemes

This Annexe provides details of the ASCII character set (ANSI X3.4) used for one of the ECC 000-140 encodation schemes, and the four encodation schemes showing the mapping of the data character to the encodation scheme code value.

ASCII SET		ENCODATION SCHEME			
Character	Decimal Value	Base 11 Code Value	Base 27 Code Value	Base 37 Code Value	Base 41 Code Value
NUL	0				
SOH	1				
STX	2				
ETX	3				
EOT	4				
ENQ	5				
ACK	6				
BEL	7				
BS	8				
HT	9				
LF	10				
VT	11				
FF	12				
CR	13				
SO	14				
SI	15				
DLE	16				
DC1	17				
DC2	18				
DC3	19				
DC4	20				
NAK	21				
SYN	22				
ETB	23				
CAN	24				
EM	25				
SUB	26				
ESC	27				
FS	28				
GS	29				
RS	30				
US	31				
space	32	0	0	0	0
!	33				
"	34				
#	35				
\$	36				
%	37				

Table C1: Mapping of Data Character Value to Encodation Scheme Value

ASCII SET		ENCODATION SCHEME			
Character	Decimal Value	Base 11 Code Value	Base 27 Code Value	Base 37 Code Value	Base 41 Code Value
&	38				
'	39				
(	40				
)	41				
*	42				
+	43				
,	44				38
-	45				39
.	46				37
/	47				40
0	48	1		27	27
1	49	2		28	28
2	50	3		29	29
3	51	4		30	30
4	52	5		31	31
5	53	6		32	32
6	54	7		33	33
7	55	8		34	34
8	56	9		35	35
9	57	10		36	36
:	58				
;	59				
<	60				
=	61				
>	62				
?	63				
@	64				
A	65		1	1	1
B	66		2	2	2
C	67		3	3	3
D	68		4	4	4
E	69		5	5	5
F	70		6	6	6
G	71		7	7	7
H	72		8	8	8
I	73		9	9	9
J	74		10	10	10
K	75		11	11	11
L	76		12	12	12
M	77		13	13	13
N	78		14	14	14
O	79		15	15	15
P	80		16	16	16
Q	81		17	17	17
R	82		18	18	18
S	83		19	19	19

Table C1: Mapping of Data Character Value to Encodation Scheme Value (Continued.)

ASCII SET		ENCODATION SCHEME			
Character	Decimal Value	Base 11 Code Value	Base 27 Code Value	Base 37 Code Value	Base 41 Code Value
T	84		20	20	20
U	85		21	21	21
V	86		22	22	22
W	87		23	23	23
X	88		24	24	24
Y	89		25	25	25
Z	90		26	26	26
[	91				
\	92				
]	93				
^	94				
-	95				
'	96				
a	97				
b	98				
c	99				
d	100				
e	101				
f	102				
g	103				
h	104				
i	105				
j	106				
k	107				
l	108				
m	109				
n	110				
o	111				
p	112				
q	113				
r	114				
s	115				
t	116				
u	117				
v	118				
w	119				
x	120				
y	121				
z	122				
{	123				
	124				
}	125				
~	126				
DEL	127				

Table C1: Mapping of Data Character Value to Encodation Scheme Value (Concluded)

C.1 Base 11 Encodation Scheme

C.1.1 First Stage Procedure

The data characters shall be converted to their Base 11 code values using Table C1 as the conversion table.

C.1.2 Second Stage Procedure

The following procedure shall be used to compact the Base 11 code values to a binary string.

1. Sub-divide the number of Base 11 characters into a sequence of six characters, from left to right. If less than six characters go to Step 5.
2. Assign the code values of the six Base 11 characters as C_1 to C_6 , where C_1 is the first character.
3. Carry out a Base 11 to Base 2 conversion to produce a sequence of 21 bits, using equation 6 of Table C2.

4. Repeat from Step 1 as necessary.

5. When there are less than six characters, carry out a Base 11 to Base 2 conversion using the appropriate equation of Table C2 which is equal to the number of remaining Base 11 characters.

C.1.3 Example

Using the data character string:

123 45678

the complete Base 11 encodation process is shown in Figure C1.

Number of Data Characters	Encodation Equation	Bit Length
1	C_1	4
2	$C_1 + C_2 * 11$	7
3	$C_1 + C_2 * 11 + C_3 * 11^2$	11
4	$C_1 + C_2 * 11 + C_3 * 11^2 + C_4 * 11^3$	14
5	$C_1 + C_2 * 11 + C_3 * 11^2 + C_4 * 11^3 + C_5 * 11^4$	18
6	$C_1 + C_2 * 11 + C_3 * 11^2 + C_4 * 11^3 + C_5 * 11^4 + C_6 * 11^5$	21

Table C2: Base 11 (Numeric) Encodation Equations

Data	1	2	3	4	5	6	7	8
Base 11 code value	2	3	4	0	5	6	7	8
Character position	C_1	C_2	C_3	C_4	C_5	C_6	C_1	C_2
Decimal value	1040030					1184		
Binary string	011111101111010011110					10010100000		

Figure C1: Base 11 Example

C.2 Base 27 Encodation Scheme

C.2.1 First Stage Procedure

The data characters shall be converted to their Base 27 code values using Table C1 as the conversion table.

C.2.2 Second Stage Procedure

The following procedure shall be used to compact the Base 27 code values to a binary string.

1. Sub-divide the number of Base 27 characters into a sequence of five characters, from left to right. If less than five characters go to Step 5.
2. Assign the code values of the five Base 27 characters as C_1 to C_5 , where C_1 is the first character.

3. Carry out a Base 27 to Base 2 conversion to produce a sequence of 24 bits, using equation 5 of Table C3.

4. Repeat from Step 1 as necessary.

5. When there are less than five characters, carry out a Base 27 to Base 2 conversion using the appropriate equation of Table C3 which is equal to the number of remaining Base 27 characters.

C.2.3 Example

Using the data character string:

DATA MATRIX

the complete Base 27 encodation process is shown in Figure C2.

Number of Data Characters	Encodation Equation	Bit Length
1	C_1	5
2	$C_1 + C_2 * 27$	10
3	$C_1 + C_2 * 27 + C_3 * 27^2$	15
4	$C_1 + C_2 * 27 + C_3 * 27^2 + C_4 * 27^3$	20
5	$C_1 + C_2 * 27 + C_3 * 27^2 + C_4 * 27^3 + C_5 * 27^4$	24

Table C3: Base 27 (Upper-case Alphabetic) Encodation Equations

Data	D	A	T	A		M	A	T	R	I	X
Base 27 code value	4	1	20	1	0	13	1	20	18	9	24
Character position	C_1	C_2	C_3	C_4	C_5	C_1	C_2	C_3	C_4	C_5	C_1
Decimal value	34294					5151883					24
Binary string	000000001000010111110110					010011101001110010001011					11000

Figure C2: Base 27 Example

C.3 Base 37 Encodation Scheme

C.3.1 First Stage Procedure

The data characters shall be converted to their Base 37 code values using Table C1 as the conversion table.

C.3.2 Second Stage Procedure

The following procedure shall be used to compact the Base 37 code values to a binary string.

1. Sub-divide the number of Base 37 characters into a sequence of four characters, from left to right. If less than four characters go to Step 5.
2. Assign the code values of the four Base 37 characters as C_1 to C_4 , where C_1 is the first character.
3. Carry out a Base 37 to Base 2 conversion to produce a sequence of 21 bits, using equation 4 of Table C4.

4. Repeat from Step 1 as necessary.

5. When there are less than four characters, carry out a Base 37 to Base 2 conversion using the equation (1 to 3) of Table C4 which is equal to the number of remaining Base 37 characters.

C.3.3 Example

Using the data character string:

123ABCD89

the complete Base 37 encodation process is shown in Figure C3.

Number of Data Characters	Encodation Equation	Bit Length
1	C_1	6
2	$C_1 + C_2 * 37$	11
3	$C_1 + C_2 * 37 + C_3 * 37^2$	16
4	$C_1 + C_2 * 37 + C_3 * 37^2 + C_4 * 37^3$	21

Table C4: Base 37 (Upper-case Alphanumeric) Encodation Equations

Data	1	2	3	A	B	C	D	8	9
Base 37 code value	28	29	30	1	2	3	4	35	36
Character position	C_1	C_2	C_3	C_4	C_1	C_2	C_3	C_4	C_1
Decimal value	92824			1778444				36	
Binary string	000010110101010011000				110110010001100001100				100100

Figure C3: Base 37 Example

C.4 Base 41 Encodation Scheme

C.4.1 First Stage Procedure

The data characters shall be converted to their Base 41 code values using Table C1 as the conversion table.

C.4.2 Second Stage Procedure

The following procedure shall be used to compact the Base 41 code values to a binary string.

1. Sub-divide the number of Base 41 characters into a sequence of four characters, from left to right. If less than four characters go to Step 5.
2. Assign the code values of the four Base 41 characters as C_1 to C_4 , where C_1 is the first character.
3. Carry out a Base 41 to Base 2 conversion to produce a sequence of 22 bits, using equation 4 of Table C5.

4. Repeat from Step 1 as necessary.

5. When there are less than four characters, carry out a Base 41 to Base 2 conversion using the appropriate equation of Table C5 which is equal to the number of remaining Base 41 characters.

C.4.3 Example

Using the data character string:

AB/C123-X

the complete Base 41 encodation process is shown in Figure C4.

Number of Data Characters	Encodation Equation	Bit Length
1	C_1	6
2	$C_1 + C_2 * 41$	11
3	$C_1 + C_2 * 41 + C_3 * 41^2$	17
4	$C_1 + C_2 * 41 + C_3 * 41^2 + C_4 * 41^3$	22

Table C5: Base 41 (Upper-case Alphanumeric + Punctuation) Encodation Equations

Data	A	B	/	C	1	2	3	-	X
Base 41 code value	1	2	40	3	28	29	30	39	24
Character position	C_1	C_2	C_3	C_4	C_1	C_2	C_3	C_4	C_1
Decimal value		274086			2739566				24
Binary string	0001000010111010100110				1010011100110101101110				011000

Figure C4: Base 41 Example

Annexe D (Normative)

ECC 000 - 140 CRC Algorithm

Following are two implementations for representing CRC.

D.1 CRC State Machine

The CRC may be represented as a schematic, as illustrated in Figure D1. After the data bits have been shifted through the state machine the resulting CRC is read out of the 16 memory registers (m) in the diagram (left most register is the MSB).

D.2 CRC Polynomial

The CRC algorithm shall be the CCITT standard polynomial:

$X^{16}+X^{12}+X^5+1$

With X equals 2, the value of the polynomial shown as a 17 bit value is:

$10001000000100001_{base\ 2}$

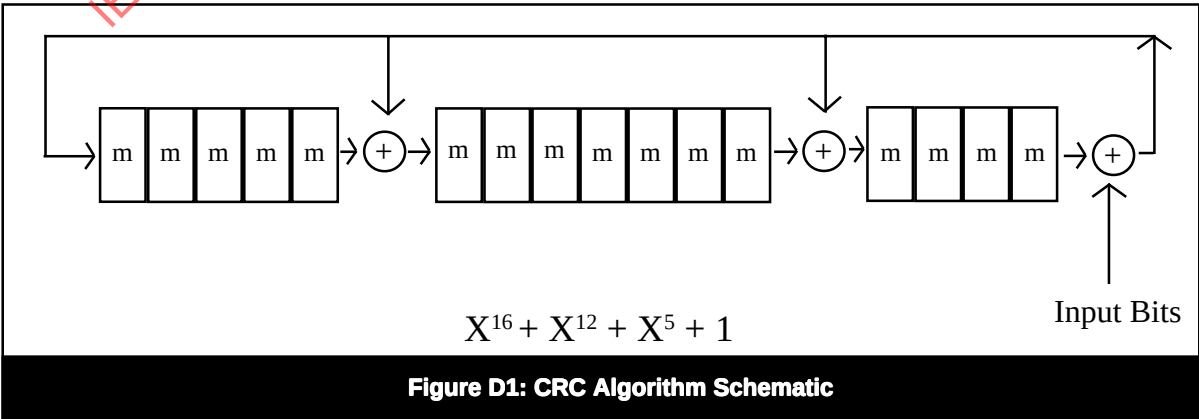
The CRC is the remainder after dividing the data string by this value.

D.3 CRC 2-Byte Header

The CRC Calculation Headers, as defined in Table D1, are used in the CRC operation as a prefix to the 8-bit byte values of the data characters. The CRC 2-byte header is shifted into the state machine prior to the calculation of the CRC.

Format ID	Encodation Scheme	CRC Calculation Header		
		MS Byte	LS Byte	Hex
1	Base 11	00000001	00000000	01 00
2	Base 27	00000010	00000000	02 00
3	Base 41	00000011	00000000	03 00
4	Base 37	00000100	00000000	04 00
5	ASCII	00000101	00000000	05 00
6	8-bit Byte	00000110	00000000	06 00

Table D1: CRC Calculation Header



Annexe E (Normative)

ECC 000 - 140 Error Checking and Correcting Algorithms

E.1 ECC 000

This provides no error correction.

E.2 ECC 050

The error correction bit stream 'v' for ECC 050 shall be created by processing the user data bit stream 'u' through a state machine suitable for a convolutional code of the structure 4-3-3, as illustrated in Figure E1.

E.3 ECC 080

The error correction bit stream 'v' for ECC 080 shall be created by processing the user bit stream 'u' through a state machine suitable for a convolutional code of the structure 3-2-11, as illustrated in Figure E2.

E.4 ECC 100

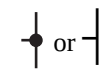
The error correction bit stream 'v' for ECC 100 shall be created by processing the user bit stream 'u' through a state machine suitable for a convolutional code of the structure 2-1-15, as illustrated in Figure E3.

E.5 ECC 140

The error correction bit stream 'v' for ECC 140 shall be created by processing the user bit stream 'u' through a state machine suitable for a convolutional code of the structure 4-1-13, as illustrated in Figure E4.

E.6 Processing the Convolutional Code

In the state machine circuit diagrams, the following notation is used:

-  represents a single bit storage register
-  represents a one bit binary adder which outputs the lowest bit. It is equivalent to an odd parity generator.
-  or  such adjoining lines are connected
-  such intersecting lines are not connected

The state machine is operated as follows:

1. The memory storage registers (m) are filled with a zero value before starting the process.
2. An input cycle is performed, consisting of passing a user data bit through the input switch to a memory storage register (m) for each possible input switch position, i.e. for k bits.
3. Once a complete set of k input bits has been entered, an output cycle is performed. An output cycle consists of reading out an error corrected bit for each possible output switch position, i.e. for n bits. At each position, the output bit is computed by performing an XOR operation on the connected memory storage register values.
4. After one input and output cycle, a shift operation is performed by shifting all memory storage register values to the right by one position.
5. Steps 2 through 4 are repeated until all raw data bits have been input. At the end:
 - a. Some zero bits may need to be added to the end of the last segment of input bits to ensure that k bits are input.
 - b. Sufficient additional zero bits shall be input to ensure that the m memory storage registers shall all return to zero values.

The output from Steps 5a and 5b is part of the encoded data. The process is complete when all true data bits have passed through the last (rightmost) memory storage register.

E.7 Convolutional Codes Reference Decode Algorithm

The Fano algorithm can be used for error correction of data protected by convolutional codes. A basic description of the operation of the Fano algorithm is given in Lin and Costello (see Reference below). The following guidelines should be used in constructing a convolutional coding decoder.

The startup variable values must be as follows:

Backward Metric = maximum negative number
 Current Metric = 0
 Forward Metric = 0
 Threshold = 0

The metric is computed by determining the number of bits that are different between the damaged block and the candidate match block:

$$\text{Metric} = (1 * \text{correct bits}) - (\text{penalty} * \text{incorrect})$$

Table E1 presents values for the Single Bit Penalty and Delta which should be used when decoding each of the ECC levels.

ECC Level	Single Bit Penalty	Delta
050	31	20
080	16	11
100	8	6
140	4	1

Table E1: Fano Algorithm Coefficients

Reference Text: Lin and Costello, "Error Control Coding: Fundamentals and Applications," Prentice Hall, 1983.

Reference Text: C. Britton Rorabaugh, "Error Coding Cookbook," McGraw Hill, 1996.

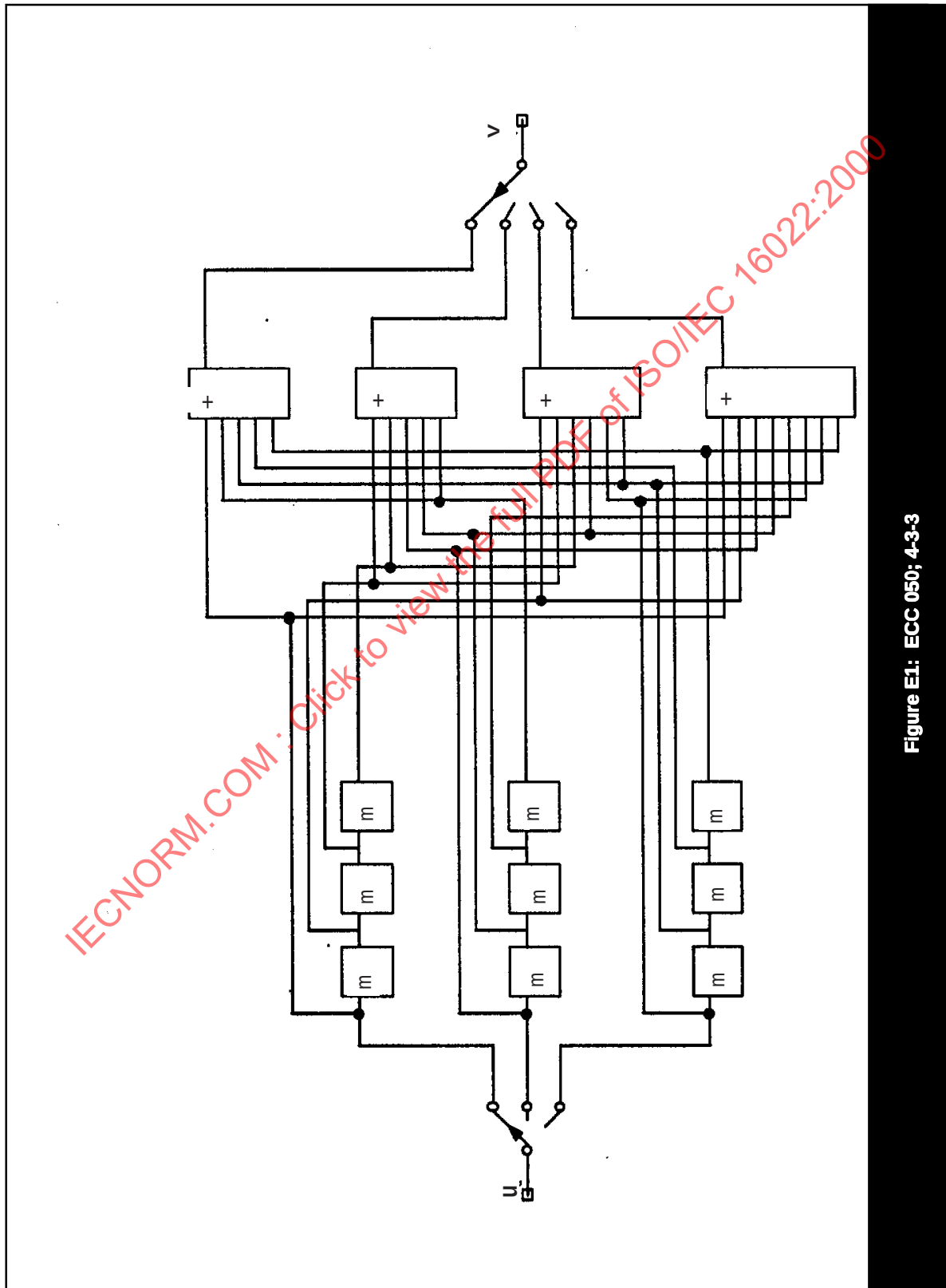


Figure E1: ECC 050; 4-3-3

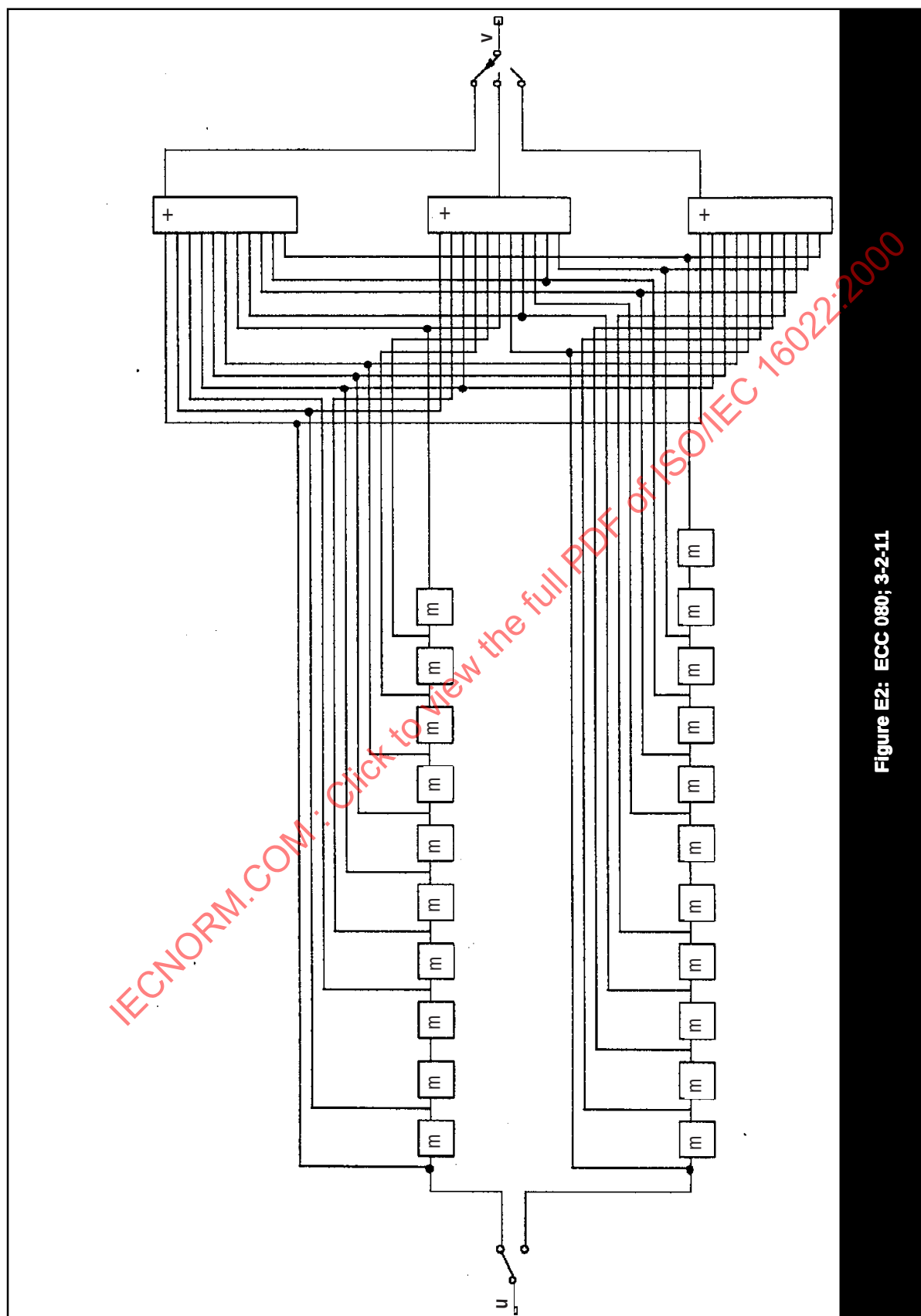
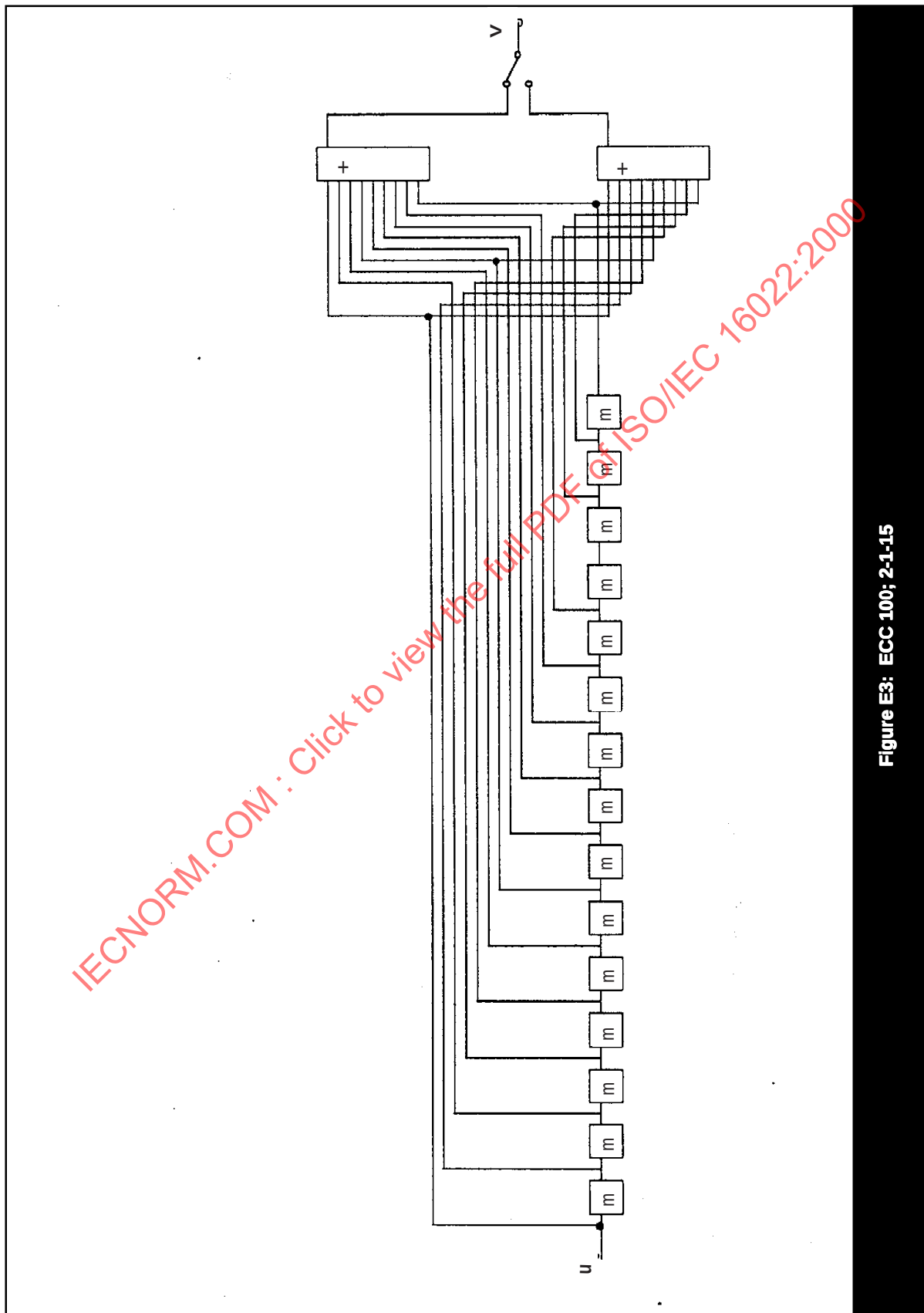


Figure E2: ECC 080; 3-2-11



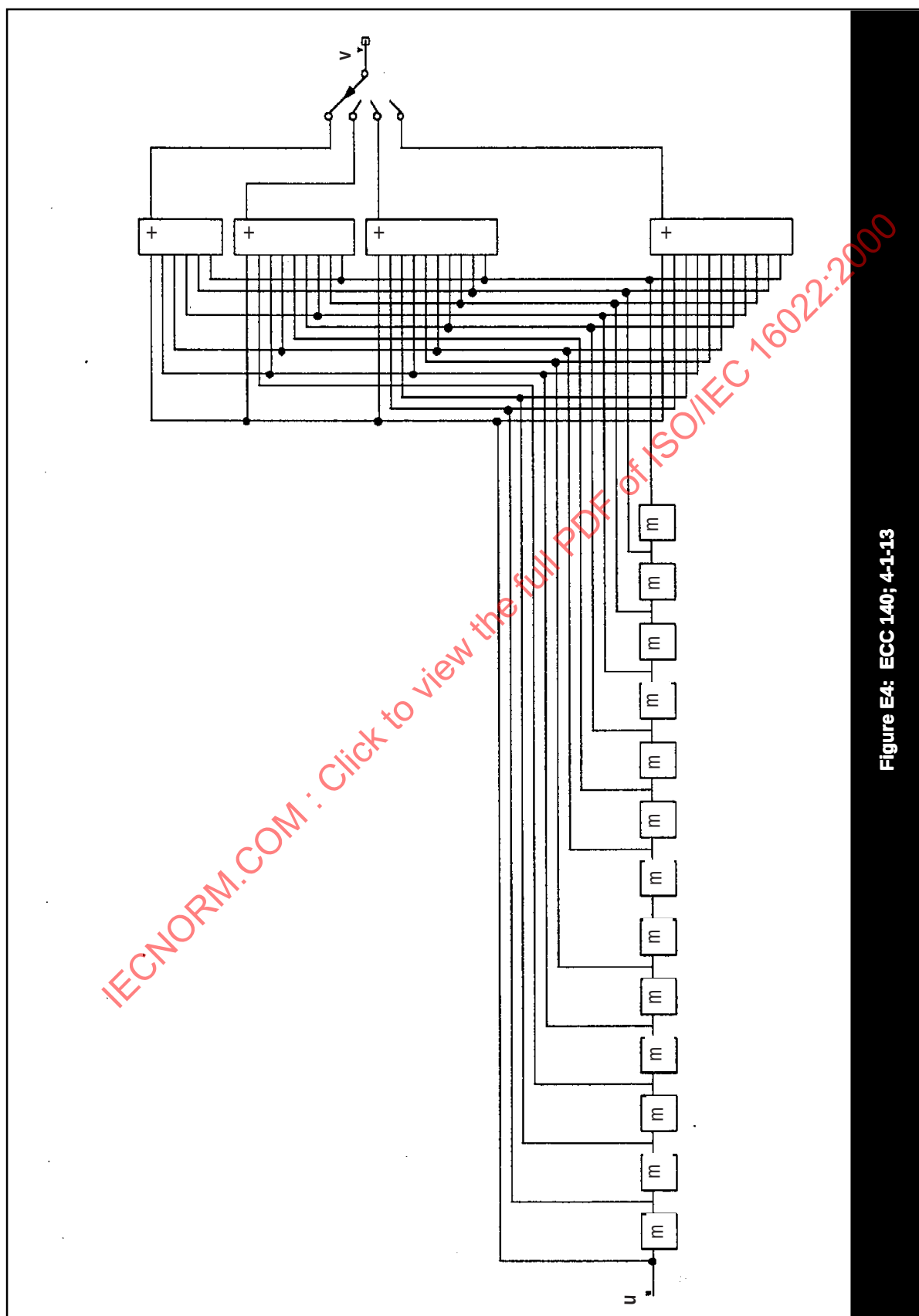


Figure E4: ECC 140; 4-1-13

*Annexe F (Normative)**ECC 000 - 140 Master Random Bit Stream (in hexadecimal)*

(MSB)

05	ff	c7	31	88	a8	83	9c	64	87	9f	64	b3	e0	4d	9c	80	29	3a	90
b3	8b	9e	90	45	bf	f5	68	4b	08	cf	44	b8	d4	4c	5b	a0	ab	72	52
1c	e4	d2	74	a4	da	8a	08	fa	a7	c7	dd	00	30	a9	e6	64	ab	d5	8b
ed	9c	79	f8	08	d1	8b	c6	22	64	0b	33	43	d0	80	d4	44	95	2e	6f
5e	13	8d	47	62	06	eb	80	82	c9	41	d5	73	8a	30	23	24	e3	7f	b2
a8	0b	ed	38	42	4c	d7	b0	ce	98	bd	e1	d5	e4	c3	1d	15	4a	cf	d1
1f	39	26	18	93	fc	19	b2	2d	ab	f2	6e	a1	9f	af	d0	8a	2b	a0	56
b0	41	6d	43	a4	63	f3	aa	7d	af	35	57	c2	94	4a	65	0b	41	de	b8
e2	30	12	27	9b	66	2b	34	5b	b8	99	e8	28	71	d0	95	6b	07	4d	3c
7a	b3	e5	29	b3	ba	8c	cc	2d	e0	c9	c0	22	ec	4c	de	f8	58	07	fc
19	f2	64	e2	c3	e2	d8	b9	fd	67	a0	bc	f5	2e	c9	49	75	62	82	27
10	f4	19	6f	49	f7	b3	84	14	ea	eb	e1	2a	31	ab	47	7d	08	29	ac
bb	72	fa	fa	62	b8	c8	d3	86	89	95	fd	df	cc	9c	ad	f1	d4	6c	64
23	24	2a	56	1f	36	eb	b7	d6	ff	da	57	f4	50	79	08	0			(LSB)

Annexe G (Normative)

ECC 200 Interleaving Process

G.1 Schematic Illustration

Using the example of the 72 x 72 symbol size, four levels of interleaving are required to encode a total of 368 data codewords and 144 error correction codewords. These are divided into four blocks of 92 data codewords and 36 error correction codewords, a total block length of 128 codewords.

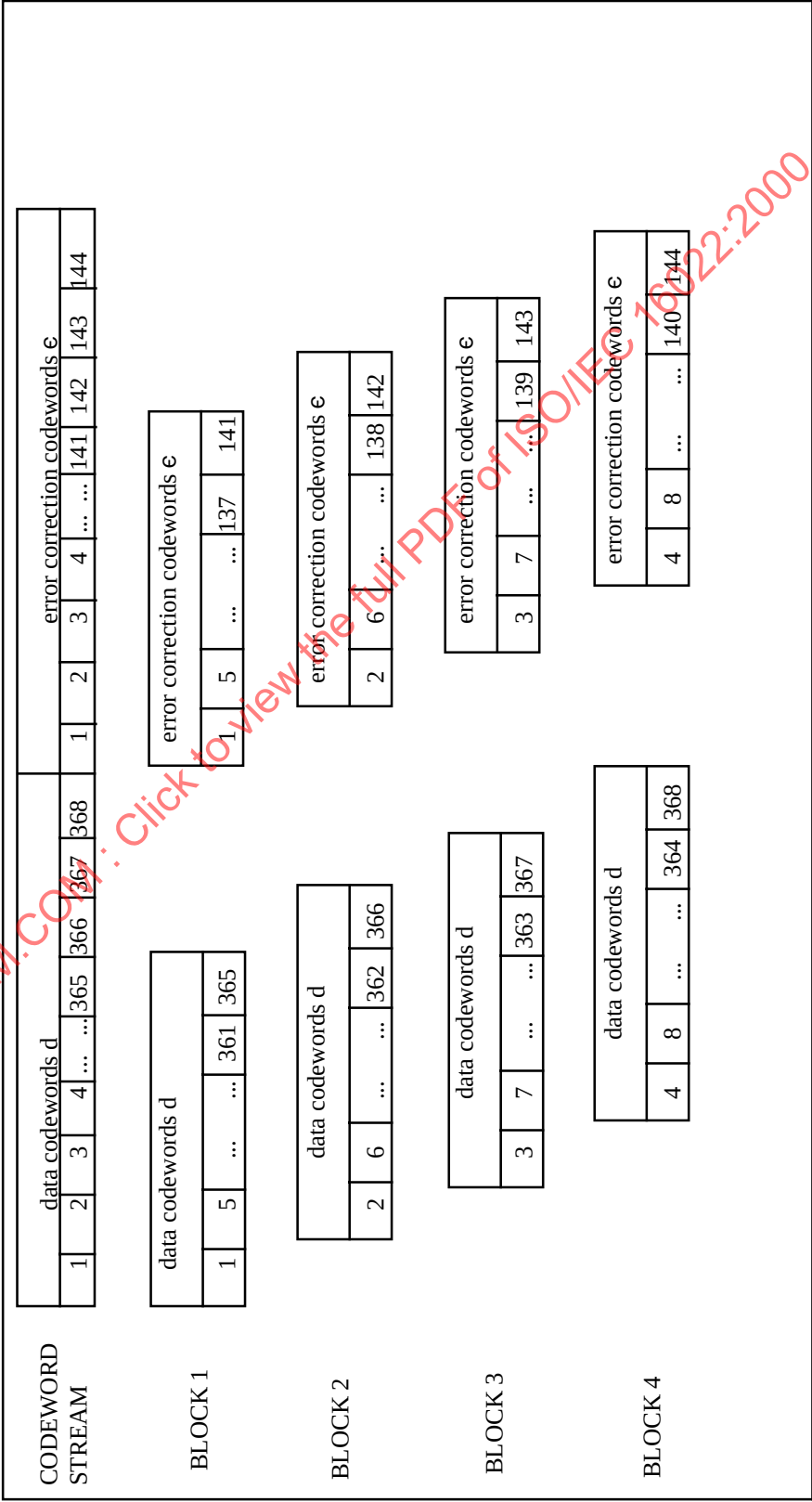


Figure G1: Illustration of Interleaving for 72 x 72 Symbol

G.2 Starting Sequence for Interleaving in Different Sized Symbols

The sequence of the interleaved data codewords and error correction codewords is given in Table G1.

Symbol Size	Reed-Solomon Block	Sequence of Data Codewords			Sequence of Error Correction Codewords		
52 x 52	1	1, 3, 5	...	201, 203	1, 3, 5	...	81, 83
	2	2, 4, 6	...	202, 204	2, 4, 6	...	82, 84
64 x 64	1	1, 3, 5	...	277, 279	1, 3, 5	...	109, 111
	2	2, 4, 6	...	278, 280	2, 4, 6	...	110, 112
72 x 72	1	1, 5, 9	...	361, 365	1, 5, 9	...	137, 141
	2	2, 6, 10	...	362, 366	2, 6, 10	...	138, 142
	3	3, 7, 11	...	363, 367	3, 7, 11	...	139, 143
	4	4, 8, 12	...	364, 368	4, 8, 12	...	140, 144
80 x 80	1	1, 5, 9	...	449, 453	1, 5, 9	...	185, 189
	2	2, 6, 10	...	450, 454	2, 6, 10	...	186, 190
	3	3, 7, 11	...	451, 455	3, 7, 11	...	187, 191
	4	4, 8, 12	...	452, 456	4, 8, 12	...	188, 192
88 x 88	1	1, 5, 9	...	569, 573	1, 5, 9	...	217, 221
	2	2, 6, 10	...	570, 574	2, 6, 10	...	218, 222
	3	3, 7, 11	...	571, 575	3, 7, 11	...	219, 223
	4	4, 8, 12	...	572, 576	4, 8, 12	...	220, 224
96 x 96	1	1, 5, 9	...	689, 693	1, 5, 9	...	265, 269
	2	2, 6, 10	...	690, 694	2, 6, 10	...	266, 270
	3	3, 7, 11	...	691, 695	3, 7, 11	...	267, 271
	4	4, 8, 12	...	692, 696	4, 8, 12	...	268, 272
104 x 104	1	1, 7, 13	...	805, 811	1, 7, 13	...	325, 331
	2	2, 8, 14	...	806, 812	2, 8, 14	...	326, 332
	3	3, 9, 15	...	807, 813	3, 9, 15	...	327, 333
	4	4, 10, 16	...	808, 814	4, 10, 16	...	328, 334
	5	5, 11, 17	...	809, 815	5, 11, 17	...	329, 335
	6	6, 12, 18	...	810, 816	6, 12, 18	...	330, 336
120 x 120	1	1, 7, 13	...	1039, 1045	1, 7, 13	...	397, 403
	2	2, 8, 14	...	1040, 1046	2, 8, 14	...	398, 404
	3	3, 9, 15	...	1041, 1047	3, 9, 15	...	399, 405
	4	4, 10, 16	...	1042, 1048	4, 10, 16	...	400, 406
	5	5, 11, 17	...	1043, 1049	5, 11, 17	...	401, 407
	6	6, 12, 18	...	1044, 1050	6, 12, 18	...	402, 408

Table G1: Sequence of Data and Error Correction Codewords for Different Symbol Sizes

Symbol Size	Reed-Solomon Block	Sequence of Data Codewords			Sequence of Error Correction Codewords		
132 x 132	1	1, 9, 17	...	1289, 1297	1, 9, 17	...	481, 489
	2	2, 10, 18	...	1290, 1298	2, 10, 18	...	482, 490
	3	3, 11, 19	...	1291, 1299	3, 11, 19	...	483, 491
	4	4, 12, 20	...	1292, 1300	4, 12, 20	...	484, 492
	5	5, 13, 21	...	1293, 1301	5, 13, 21	...	485, 493
	6	6, 14, 22	...	1294, 1302	6, 14, 22	...	486, 494
	7	7, 15, 23	...	1295, 1303	7, 15, 23	...	487, 495
	8	8, 16, 24	...	1296, 1304	8, 16, 24	...	488, 496
144 x 144	1	1, 11, 21	...	1541, 1551	1, 11, 21	...	601, 611
	2	2, 12, 22	...	1542, 1552	2, 12, 22	...	602, 612
	3	3, 13, 23	...	1543, 1553	3, 13, 23	...	603, 613
	4	4, 14, 24	...	1544, 1554	4, 14, 24	...	604, 614
	5	5, 15, 25	...	1545, 1555	5, 15, 25	...	605, 615
	6	6, 16, 26	...	1546, 1556	6, 16, 26	...	606, 616
	7	7, 17, 27	...	1547, 1557	7, 17, 27	...	607, 617
	8	8, 18, 28	...	1548, 1558	8, 18, 28	...	608, 618
	9	9, 19, 29	...	1549	9, 19, 29	...	609, 619
	10	10, 20, 30	...	1550	10, 20, 30	...	610, 620
Note: The 9th and 10th blocks of the 144 x 144 symbol have only 155 data codewords, whereas the other eight blocks have 156 data codewords							
Table G1: Sequence of Data and Error Correction Codewords for Different Symbol Sizes (concluded)							

Annexe H (Normative)

ECC 200 Pattern Randomizing

The pattern randomizing algorithms convert an input codeword at a given position to new randomized output codeword.

H.1 253-State Algorithm

This algorithm adds a pseudo-random number to the Pad codeword value. The pseudo-random number will always be in the range 1 to 253 and the randomized Pad codeword value be in the range 1 to 254.

The variable Pad_codeword_position is the number of data codewords from the beginning of encoded data.

H.1.1 253-State Randomizing Algorithm

```

INPUT ( Pad_codeword_value,
Pad_codeword_position )
pseudo_random_number = ( ( 149 *
    Pad_codeword_position ) mod 253 ) + 1
temp_variable = Pad_codeword_value +
    pseudo_random_number
IF ( temp_variable <= 254 )
    OUTPUT ( randomized_Pad_codeword_value =
        temp_variable )
ELSE
    OUTPUT ( randomized_Pad_codeword_value =
        temp_variable - 254 )

```

H.1.2 253-State Un-Randomizing Algorithm

```

INPUT ( randomized_Pad_codeword_value,
Pad_codeword_position )
pseudo_random_number = ( ( 149 *
    Pad_codeword_position ) mod 253 ) + 1
temp_variable = randomized_Pad_codeword_value -
    pseudo_random_number
IF ( temp_variable >= 1 )
    OUTPUT ( Pad_codeword_value =
        temp_variable )
ELSE
    OUTPUT ( Pad_codeword_value =
        temp_variable + 254 )

```

H.2 255-State Algorithm

This algorithm adds a pseudo-random number to the Base 256 Encodation codeword value. The pseudo-random number will always be in the range 1 to 255 and the randomized Base 256 codeword value be in the range 0 to 255.

The variable Base256_codeword_position is the number of data codewords from the beginning of encoded data.

H.2.1 255-State Randomizing Algorithm

```

INPUT ( Base256_codeword_value,
Base256_codeword_position )
pseudo_random_number = ( ( 149 *
    Base256_codeword_position ) mod 255 ) + 1
temp_variable = Base256_codeword_value +
    pseudo_random_number
IF ( temp_variable <= 255 )
    OUTPUT
        (randomized_Base256_codeword_value
        = temp_variable )
ELSE
    OUTPUT
        (randomized_Base256_codeword_value
        = temp_variable - 256 )

```

H.2.2 255-State Un-Randomizing Algorithm

```

INPUT ( randomized_Base256_codeword_value,
Base256_codeword_position )
pseudo_random_number = ( ( 149 *
    Base256_codeword_position ) mod 255 ) + 1
temp_variable = randomized_Base256_codeword_value -
    pseudo_random_number
IF ( temp_variable >= 0 )
    OUTPUT ( Base256_codeword_value =
        temp_variable )
ELSE
    OUTPUT ( Base256_codeword_value =
        temp_variable + 256 )

```

Annexe J (Normative)*ECC 200 Encodation Character Sets**J.1 C40 Encodation Character Set*

C40 Value	Basic Set		Shift 1 Set		Shift 2 Set		Shift 3 Set	
	Char	Decimal	Char	Decimal	Char	Decimal	Char	Decimal
0	Shift 1		NUL	0	!	33	‘	96
1	Shift 2		SOH	1	“	34	a	97
2	Shift 3		STX	2	#	35	b	98
3	space	32	ETX	3	\$	36	c	99
4	0	48	EOT	4	%	37	d	100
5	1	49	ENQ	5	&	38	e	101
6	2	50	ACK	6	‘	39	f	102
7	3	51	BEL	7	(	40	g	103
8	4	52	BS	8	)	41	h	104
9	5	53	HT	9	*	42	i	105
10	6	54	LF	10	+	43	j	106
11	7	55	VT	11	,	44	k	107
12	8	56	FF	12	-	45	l	108
13	9	57	CR	13	.	46	m	109
14	A	65	SO	14	/	47	n	110
15	B	66	SI	15	:	58	o	111
16	C	67	DLE	16	;	59	p	112
17	D	68	DC1	17	<	60	q	113
18	E	69	DC2	18	=	61	r	114
19	F	70	DC3	19	>	62	s	115
20	G	71	DC4	20	?	63	t	116
21	H	72	NAK	21	@	64	u	117
22	I	73	SYN	22	[	91	v	118
23	J	74	ETB	23	\	92	w	119
24	K	75	CAN	24	]	93	x	120
25	L	76	EM	25	^	94	y	121
26	M	77	SUB	26	_	95	z	122
27	N	78	ESC	27	FNC1		{	123
28	O	79	FS	28				124
29	P	80	GS	29			}	125
30	Q	81	RS	30	Upper Shift		~	126
31	R	82	US	31			DEL	127
32	S	83						
33	T	84						
34	U	85						
35	V	86						
36	W	87						
37	X	88						
38	Y	89						
39	Z	90						

Note: The relationship between the ASCII decimal value and the C40 value remain constant regardless of which ECI is in effect.

J.2 Text Encodation Character Set

Text Value	Basic Set Char	Decimal	Shift 1 Set Char	Decimal	Shift 2 Set Char	Decimal	Shift 3 Set Char	Decimal
0	Shift 1		NUL	0	!	33	'	96
1	Shift 2		SOH	1	"	34	A	65
2	Shift 3		STX	2	#	35	B	66
3	space	32	ETX	3	\$	36	C	67
4	0	48	EOT	4	%	37	D	68
5	1	49	ENQ	5	&	38	E	69
6	2	50	ACK	6	'	39	F	70
7	3	51	BEL	7	(	40	G	71
8	4	52	BS	8	)	41	H	72
9	5	53	HT	9	*	42	I	73
10	6	54	LF	10	+	43	J	74
11	7	55	VT	11	,	44	K	75
12	8	56	FF	12	-	45	L	76
13	9	57	CR	13	.	46	M	77
14	a	97	SO	14	/	47	N	78
15	b	98	SI	15	:	58	O	79
16	c	99	DLE	16	;	59	P	80
17	d	100	DC1	17	<	60	Q	81
18	e	101	DC2	18	=	61	R	82
19	f	102	DC3	19	>	62	S	83
20	g	103	DC4	20	?	63	T	84
21	h	104	NAK	21	@	64	U	85
22	i	105	SYN	22	[	91	V	86
23	j	106	ETB	23	\	92	W	87
24	k	107	CAN	24	]	93	X	88
25	l	108	EM	25	^	94	Y	89
26	m	109	SUB	26	_	95	Z	90
27	n	110	ESC	27	FNC1		{	123
28	o	111	FS	28				124
29	p	112	GS	29			}	125
30	q	113	RS	30	Upper Shift		~	126
31	r	114	US	31			DEL	127
32	s	115						
33	t	116						
34	u	117						
35	v	118						
36	w	119						
37	x	120						
38	y	121						
39	z	122						

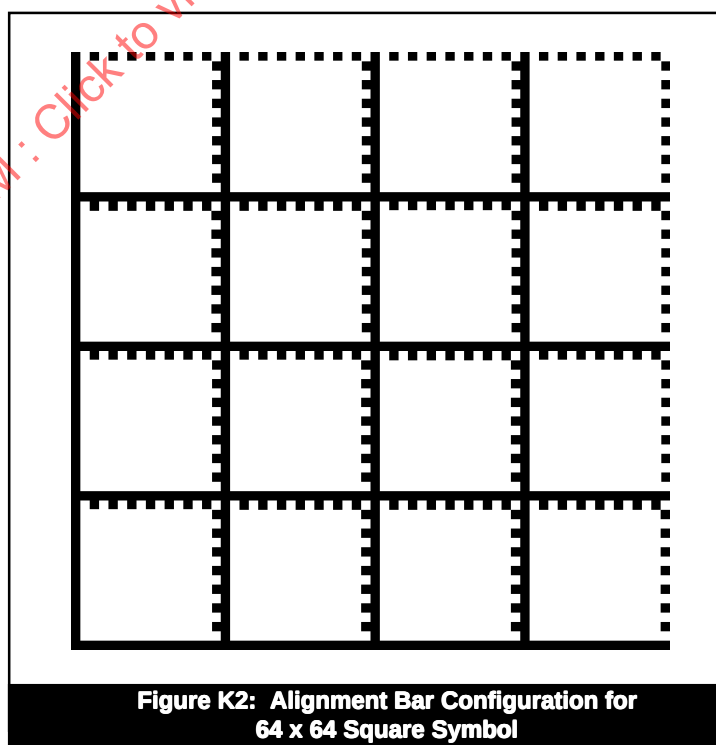
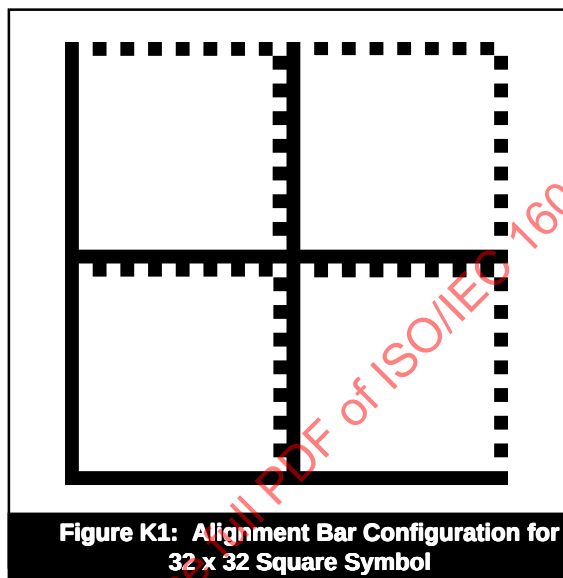
Note: The relationship between the ASCII decimal value and the Text value remain constant regardless of which ECI is in effect.

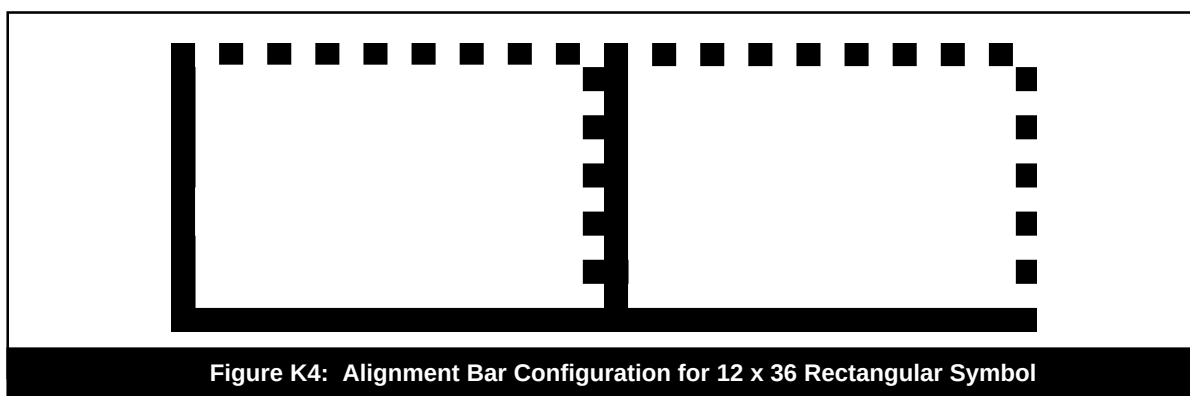
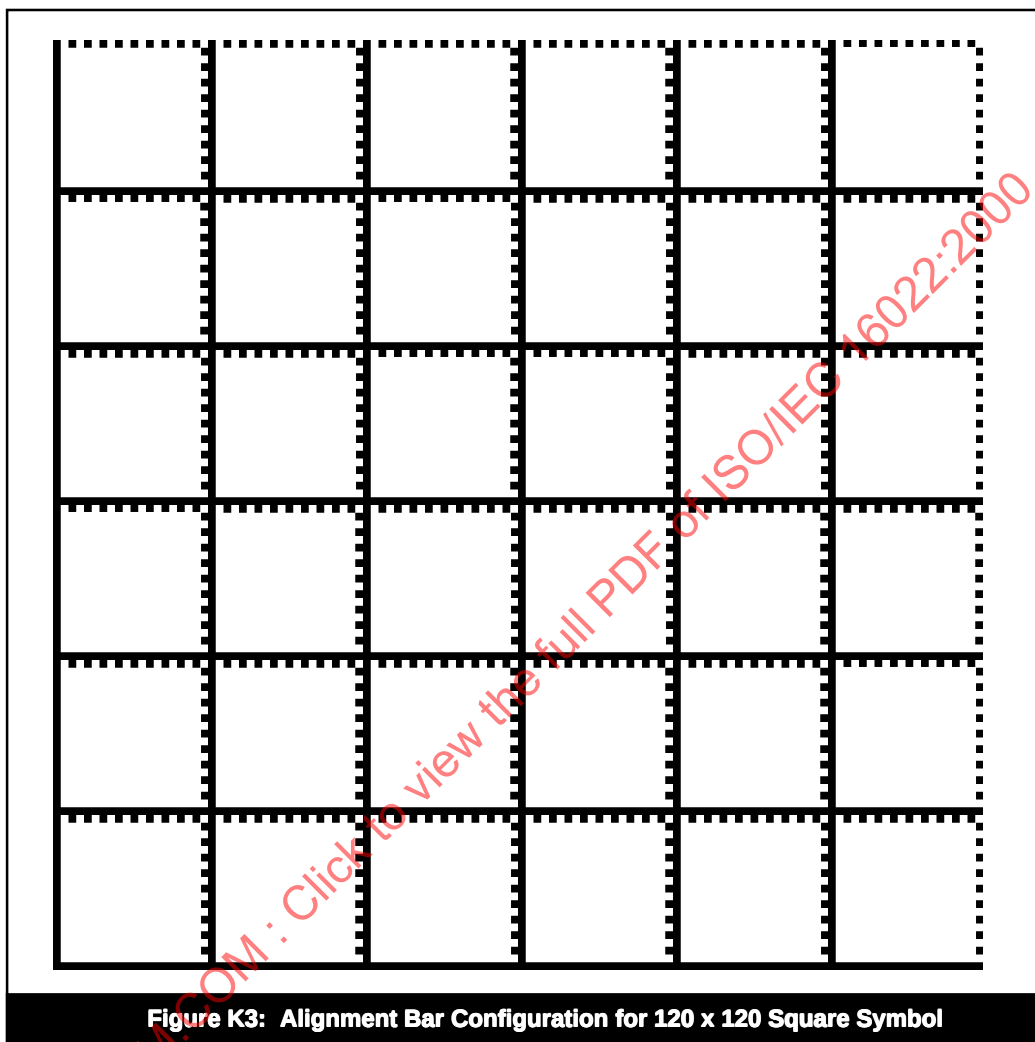
J.3 EDIFACT Encodation Character Set

Data Character				Data Character			
Char	Decimal Value	Binary Value	EDIFACT Binary Value	Char	Decimal Value	Binary Value	EDIFACT Binary Value
@	64	01000000	000000	space	32	00100000	100000
A	65	01000001	000001	!	33	00100001	100001
B	66	01000010	000010	"	34	00100010	100010
C	67	01000011	000011	#	35	00100011	100011
D	68	01000100	000100	\$	36	00100100	100100
E	69	01000101	000101	%	37	00100101	100101
F	70	01000110	000110	&	38	00100110	100110
G	71	01000111	000111	'	39	00100111	100111
H	72	01001000	001000	(	40	00101000	101000
I	73	01001001	001001	)	41	00101001	101001
J	74	01001010	001010	*	42	00101010	101010
K	75	01001011	001011	+	43	00101011	101011
L	76	01001100	001100	,	44	00101100	101100
M	77	01001101	001101	-	45	00101101	101101
N	78	01001110	001110	.	46	00101110	101110
O	79	01001111	001111	/	47	00101111	101111
P	80	01010000	010000	0	48	00110000	110000
Q	81	01010001	010001	1	49	00110001	110001
R	82	01010010	010010	2	50	00110010	110010
S	83	01010011	010011	3	51	00110011	110011
T	84	01010100	010100	4	52	00110100	110100
U	85	01010101	010101	5	53	00110101	110101
V	86	01010110	010110	6	54	00110110	110110
W	87	01010111	010111	7	55	00110111	110111
X	88	01011000	011000	8	56	00111000	111000
Y	89	01011001	011001	9	57	00111001	111001
Z	90	01011010	011010	:	58	00111010	111010
[	91	01011011	011011	;	59	00111011	111011
\	92	01011100	011100	<	60	00111100	111100
]	93	01011101	011101	=	61	00111101	111101
^	94	01011110	011110	>	62	00111110	111110
Unlatch		01011111	011111	?	63	00111111	111111

Note: The relationship between the ASCII decimal value and the EDIFACT value remain constant regardless of which ECI is in effect.

Annexe K (Normative)
ECC 200 Alignment Patterns





Annexe L (Normative)

ECC 200 Reed-Solomon Error Detection and Correction

L.1 Error Correction Codeword Generator Polynomials

The check character generation polynomial is used to divide the data symbol character polynomial where each symbol character is the coefficient of the dividend polynomial in descending power. The coefficients of the remainder of this division are the error correction symbol character values.

Each generator polynomial is the product of the first degree polynomials: $x - 2^1, x - 2^2, \dots, x - 2^n$; where n is the degree of the generator polynomial. For example the fifth degree generator polynomial is:

$$\begin{aligned} & (x + 2)(x + 4)(x + 8)(x + 16)(x + 32) \\ &= x^5 + (2 + 4 + 8 + 16 + 32)x^4 + ((2*4) + (2*8) + (2*16) + (2*32) + (4*8) + (4*16) + (4*32) + (8*16) + (8*32) + (16*32))x^3 + \\ & \quad ((2*4*8) + (2*4*16) + (2*4*32) + (2*8*16) + (2*8*32) + (2*16*32) + (4*8*16) + (4*8*32) + (4*16*32) + (8*16*32))x^2 + ((2*4*8*16) + (2*4*8*32) + (2*4*16*32) + (2*8*16*32) + (4*8*16*32))x + (2*4*8*16*32) \\ &= x^5 + 62x^4 + 111x^3 + 15x^2 + 48x + 228. \end{aligned}$$

Note that this Galois Field arithmetic is not normal integer arithmetic: $-$ is equivalent to $+$ which is an “exclusive-or” operation in this Field, and multiplication is bitwise modulo 100101101 for each binary polynomial term generated by bit-by-bit multiplication.

The polynomial divisor for generating 5 check characters is:

$$g(x) = x^5 + 62x^4 + 111x^3 + 15x^2 + 48x + 228.$$

The polynomial divisor for generating 7 check characters is:

$$g(x) = x^7 + 254x^6 + 92x^5 + 240x^4 + 134x^3 + 144x^2 + 68x + 23.$$

The polynomial divisor for generating 10 check characters is:

$$g(x) = x^{10} + 61x^9 + 110x^8 + 255x^7 + 116x^6 + 248x^5 + 223x^4 + 166x^3 + 185x^2 + 24x + 28.$$

The polynomial divisor for generating 11 check characters is:

$$g(x) = x^{11} + 120x^{10} + 97x^9 + 60x^8 + 245x^7 + 39x^6 + 168x^5 + 194x^4 + 12x^3 + 205x^2 + 138x + 175.$$

The polynomial divisor for generating 12 check characters is:

$$g(x) = x^{12} + 242x^{11} + 100x^{10} + 178x^9 + 97x^8 + 213x^7 + 142x^6 + 42x^5 + 61x^4 + 91x^3 + 158x^2 + 153x + 41.$$

The polynomial divisor for generating 14 check characters is:

$$g(x) = x^{14} + 185x^{13} + 83x^{12} + 186x^{11} + 18x^{10} + 45x^9 + 138x^8 + 119x^7 + 157x^6 + 9x^5 + 95x^4 + 252x^3 + 192x^2 + 97x + 156.$$

The polynomial divisor for generating 18 check characters is:

$$g(x) = x^{18} + 188x^{17} + 90x^{16} + 48x^{15} + 225x^{14} + 254x^{13} + 94x^{12} + 129x^{11} + 109x^{10} + 213x^9 + 241x^8 + 61x^7 + 66x^6 + 75x^5 + 188x^4 + 39x^3 + 100x^2 + 195x + 83.$$

The polynomial divisor for generating 20 check characters is:

$$g(x) = x^{20} + 172x^{19} + 186x^{18} + 174x^{17} + 27x^{16} + 82x^{15} + 108x^{14} + 79x^{13} + 253x^{12} + 145x^{11} + 153x^{10} + 160x^9 + 188x^8 + 2x^7 + 168x^6 + 71x^5 + 233x^4 + 9x^3 + 244x^2 + 195x + 15.$$

The polynomial divisor for generating 24 check characters is:

$$g(x) = x^{24} + 193x^{23} + 50x^{22} + 96x^{21} + 184x^{20} + 181x^{19} + 12x^{18} + 124x^{17} + 254x^{16} + 172x^{15} + 5x^{14} + 21x^{13} + 155x^{12} + 223x^{11} + 251x^{10} + 197x^9 + 155x^8 + 21x^7 + 176x^6 + 39x^5 + 109x^4 + 205x^3 + 88x^2 + 190x + 52.$$

The polynomial divisor for generating 28 check characters is:

$$g(x) = x^{28} + 255x^{27} + 93x^{26} + 168x^{25} + 233x^{24} + 151x^{23} + 120x^{22} + 136x^{21} + 141x^{20} + 213x^{19} + 110x^{18} + 138x^{17} + 17x^{16} + 121x^{15} + 249x^{14} + 34x^{13} + 75x^{12} + 53x^{11} + 170x^{10} + 151x^9 + 37x^8 + 174x^7 + 103x^6 + 96x^5 + 71x^4 + 97x^3 + 43x^2 + 231x + 211.$$

The polynomial divisor for generating 36 check characters is:

$$g(x) = x^{36} + 112x^{35} + 81x^{34} + 98x^{33} + 225x^{32} + 25x^{31} + 59x^{30} + 184x^{29} + 175x^{28} + 44x^{27} + 115x^{26} + 119x^{25} + 95x^{24} + 137x^{23} + 101x^{22} + 33x^{21} + 68x^{20} + 4x^{19} + 2x^{18} + 18x^{17} + 229x^{16} + 182x^{15} + 80x^{14} + 251x^{13} + 220x^{12} + 179x^{11} + 84x^{10} + 120x^9 + 102x^8 + 181x^7 + 162x^6 + 250x^5 + 130x^4 + 218x^3 + 242x^2 + 127x + 245.$$

The polynomial divisor for generating 42 check characters is:

$$g(x) = x^{42} + 5x^{41} + 9x^{40} + 5x^{39} + 226x^{38} + 177x^{37} + 150x^{36} + 50x^{35} + 69x^{34} + 202x^{33} + 248x^{32} + 101x^{31} + 54x^{30} + 57x^{29} + 253x^{28} + x^{27} + 21x^{26} + 121x^{25} + 57x^{24} + 111x^{23} + 214x^{22} + 105x^{21} + 167x^{20} + 9x^{19} + 100x^{18} + 95x^{17} + 175x^{16} + 8x^{15} + 242x^{14} + 133x^{13} + 245x^{12} + 2x^{11} + 122x^{10} + 105x^9 + 247x^8 + 153x^7 + 22x^6 + 38x^5 + 19x^4 + 31x^3 + 137x^2 + 193x + 77.$$

The polynomial divisor for generating 48 check characters is:

$$g(x) = x^{48} + 19x^{47} + 225x^{46} + 253x^{45} + 92x^{44} + 213x^{43} + 69x^{42} + 175x^{41} + 160x^{40} + 147x^{39} + 187x^{38} + 87x^{37} + 176x^{36} + 44x^{35} + 82x^{34} + 240x^{33} + 186x^{32} + 138x^{31} + 66x^{30} + 100x^{29} + 120x^{28} + 88x^{27} + 131x^{26} + 205x^{25} + 170x^{24} + 90x^{23} + 37x^{22} + 23x^{21} + 118x^{20} + 147x^{19} + 16x^{18} + 106x^{17} + 191x^{16} + 87x^{15} + 237x^{14} + 188x^{13} + 205x^{12} + 231x^{11} + 238x^{10} + 133x^9 + 238x^8 + 22x^7 + 117x^6 + 32x^5 + 96x^4 + 223x^3 + 172x^2 + 132x + 245.$$

The polynomial divisor for generating 56 check characters is:

$$g(x) = x^{56} + 46x^{55} + 143x^{54} + 53x^{53} + 233x^{52} + 107x^{51} + 203x^{50} + 43x^{49} + 155x^{48} + 28x^{47} + 247x^{46} + 67x^{45} + 127x^{44} + 245x^{43} + 137x^{42} + 13x^{41} + 164x^{40} + 207x^{39} + 62x^{38} + 117x^{37} + 201x^{36} + 150x^{35} + 22x^{34} + 238x^{33} + 144x^{32} + 232x^{31} + 29x^{30} + 203x^{29} + 117x^{28} + 234x^{27} + 218x^{26} + 146x^{25} + 228x^{24} + 54x^{23} + 132x^{22} + 200x^{21} + 38x^{20} + 223x^{19} + 36x^{18} + 159x^{17} + 150x^{16} + 235x^{15} + 215x^{14} + 192x^{13} + 230x^{12} + 170x^{11} + 175x^{10} + 29x^9 + 100x^8 + 208x^7 + 220x^6 + 17x^5 + 12x^4 + 238x^3 + 223x^2 + 9x + 175.$$

The polynomial divisor for generating 62 check

characters is:

$$g(x) = x^{62} + 204x^{61} + 11x^{60} + 47x^{59} + 86x^{58} + 124x^{57} + 224x^{56} + 166x^{55} + 94x^{54} + 7x^{53} + 232x^{52} + 107x^{51} + 4x^{50} + 170x^{49} + 176x^{48} + 31x^{47} + 163x^{46} + 17x^{45} + 188x^{44} + 130x^{43} + 40x^{42} + 10x^{41} + 87x^{40} + 63x^{39} + 51x^{38} + 218x^{37} + 27x^{36} + 6x^{35} + 147x^{34} + 44x^{33} + 161x^{32} + 71x^{31} + 114x^{30} + 64x^{29} + 175x^{28} + 221x^{27} + 185x^{26} + 106x^{25} + 250x^{24} + 190x^{23} + 197x^{22} + 63x^{21} + 245x^{20} + 230x^{19} + 134x^{18} + 112x^{17} + 185x^{16} + 37x^{15} + 196x^{14} + 108x^{13} + 143x^{12} + 189x^{11} + 201x^{10} + 188x^9 + 202x^8 + 118x^7 + 39x^6 + 210x^5 + 144x^4 + 50x^3 + 169x^2 + 93x + 242.$$

The polynomial divisor for generating 68 check characters is:

$$g(x) = x^{68} + 186x^{67} + 82x^{66} + 103x^{65} + 96x^{64} + 63x^{63} + 132x^{62} + 153x^{61} + 108x^{60} + 54x^{59} + 64x^{58} + 189x^{57} + 211x^{56} + 232x^{55} + 49x^{54} + 25x^{53} + 172x^{52} + 52x^{51} + 59x^{50} + 241x^{49} + 181x^{48} + 239x^{47} + 223x^{46} + 136x^{45} + 231x^{44} + 210x^{43} + 96x^{42} + 232x^{41} + 220x^{40} + 25x^{39} + 179x^{38} + 167x^{37} + 202x^{36} + 185x^{35} + 153x^{34} + 139x^{33} + 66x^{32} + 236x^{31} + 227x^{30} + 160x^{29} + 15x^{28} + 213x^{27} + 93x^{26} + 122x^{25} + 68x^{24} + 177x^{23} + 158x^{22} + 197x^{21} + 234x^{20} + 180x^{19} + 248x^{18} + 136x^{17} + 213x^{16} + 127x^{15} + 73x^{14} + 36x^{13} + 154x^{12} + 244x^{11} + 147x^{10} + 33x^9 + 89x^8 + 56x^7 + 159x^6 + 149x^5 + 251x^4 + 89x^3 + 173x^2 + 228x + 220.$$

L.2 Error Correction Calculation

The Peterson-Gorenstein-Zierler algorithm may be used to correct errors in decoded ECC 200 symbols. The calculation described below follows this error correcting algorithm, using the Reed-Solomon error correction codewords.

Erasures shall be corrected as errors by initially filling any erasure codeword positions with dummy values.

All calculations shall be done using GF(2⁸) arithmetic operations. Addition and subtraction are equivalent to the binary XOR operation. Multiplication and division can be performed using log and antilog tables.

Construct the symbol character polynomial C(x) = C_{n-1}xⁿ⁻¹ + C_{n-2}xⁿ⁻² + ... + C₁x¹ + C₀ where the n coefficients are the codewords read with C_{n-1} being the first symbol character and where n is the total number of symbol characters.

Calculate i syndrome values S₀ through S_{i-1} by

evaluating $C(x)$ at $x = 2^k$ for $k = 1$ through i and where i is the number of error correction codewords in the symbol.

Form and solve j simultaneous equations with j unknowns L_0 through L_{j-1} using the i syndromes:

$$\begin{aligned} S_0 L_0 + S_1 L_1 + \dots + S_{j-1} L_{j-1} &= S_j \\ S_1 L_0 + S_2 L_1 + \dots + S_j L_{j-1} &= S_{j+1} \\ &\vdots \\ S_{j-1} L_0 + S_j L_1 + \dots + S_{2j-2} L_{j-1} &= S_{2j-1} \end{aligned}$$

where j is $i/2$.

Construct the error locator polynomial:

$$L(x) = L_{j-1} x^j + L_{j-2} x^{j-1} + \dots + L_0 x + 1$$

from the j values of L obtained above. Evaluate $L(x)$ at $x = 2^k$ for $k = 0$ through $n-1$ where n is the total number of symbol characters in the symbol.

Whenever $L(2^k) = 0$, an error location is given by $n-1-k$. If more than j error locations are found, the symbol is not correctable.

Save the error locations in m error location variables E_0 through E_{m-1} where m is the number of error locations found. Form and solve m simultaneous equations with m unknowns X_0 through X_{m-1} (the error magnitudes) using the error location variables E and the first m syndromes S :

$$\begin{aligned} E_0 X_0 + E_1 X_1 + \dots + E_{m-1} X_{m-1} &= S_0 \\ E_0^2 X_0 + E_1^2 X_1 + \dots + E_{m-1}^2 X_{m-1} &= S_1 \\ E_0^3 X_0 + E_1^3 X_1 + \dots + E_{m-1}^3 X_{m-1} &= S_2 \\ &\vdots \\ E_0^m X_0 + E_1^m X_1 + \dots + E_{m-1}^m X_{m-1} &= S_{m-1} \end{aligned}$$

Add the error magnitudes X_0 through X_{m-1} to the symbol character values at the corresponding error locations E_0 through E_{m-1} to correct the errors.

This algorithm, written in C, is available from AIM^{USA} on the Data Matrix Developers Diskette, see Reference Document section.

*Annexe M (Normative)**ECC 200 Symbol Character Placement**M.1 Symbol Character Placement Program*

The following C language program generates symbol character placement diagrams:

```
#include <stdio.h>
#include <alloc.h>

int nrow, ncol, *array;

/* "module" places "chr+bit" with appropriate wrapping within array[] */
void module(int row, int col, int chr, int bit)
{ if (row < 0) { row += nrow; col += 4 - ((nrow+4)%8); }
  if (col < 0) { col += ncol; row += 4 - ((ncol+4)%8); }
  array[row*ncol+col] = 10*chr + bit;
}

/* "utah" places the 8 bits of a utah-shaped symbol character in ECC200 */
void utah(int row, int col, int chr)
{ module(row-2,col-2,chr,1);
  module(row-2,col-1,chr,2);
  module(row-1,col-2,chr,3);
  module(row-1,col-1,chr,4);
  module(row-1,col,chr,5);
  module(row,col-2,chr,6);
  module(row,col-1,chr,7);
  module(row,col,chr,8);
}

/* "cornerN" places 8 bits of the four special corner cases in ECC200 */
void corner1(int chr)
{ module(nrow-1,0,chr,1);
  module(nrow-1,1,chr,2);
  module(nrow-1,2,chr,3);
  module(0,ncol-2,chr,4);
  module(0,ncol-1,chr,5);
  module(1,ncol-1,chr,6);
  module(2,ncol-1,chr,7);
  module(3,ncol-1,chr,8);
}
void corner2(int chr)
{ module(nrow-3,0,chr,1);
  module(nrow-2,0,chr,2);
  module(nrow-1,0,chr,3);
  module(0,ncol-4,chr,4);
  module(0,ncol-3,chr,5);
  module(0,ncol-2,chr,6);
  module(0,ncol-1,chr,7);
  module(1,ncol-1,chr,8);
}
```

```

void corner3(int chr)
{ module(nrow-3,0,chr,1);
  module(nrow-2,0,chr,2);
  module(nrow-1,0,chr,3);
  module(0,ncol-2,chr,4);
  module(0,ncol-1,chr,5);
  module(1,ncol-1,chr,6);
  module(2,ncol-1,chr,7);
  module(3,ncol-1,chr,8);
}
void corner4(int chr)
{ module(nrow-1,0,chr,1);
  module(nrow-1,ncol-1,chr,2);
  module(0,ncol-3,chr,3);
  module(0,ncol-2,chr,4);
  module(0,ncol-1,chr,5);
  module(1,ncol-3,chr,6);
  module(1,ncol-2,chr,7);
  module(1,ncol-1,chr,8);
}

/* "ecc200" fills an nrow x ncol array with appropriate values for ECC200 */
void ecc200(void)
{ int row, col, chr;

/* First, fill the array[] with invalid entries */
  for (row=0; row<nrow; row++) {
    for (col=0; col<ncol; col++) {
      array[row*ncol+col] = 0;
    }
  }

/* Starting in the correct location for character #1, bit 8,... */
  chr = 1; row = 4; col = 0;

  do {
/* repeatedly first check for one of the special corner cases, then... */
    if ((row == nrow) && (col == 0)) corner1(chr++);
    if ((row == nrow-2) && (col == 0) && (ncol%4)) corner2(chr++);
    if ((row == nrow-2) && (col == 0) && (ncol%8 == 4)) corner3(chr++);
    if ((row == nrow+4) && (col == 2) && (!(ncol%8))) corner4(chr++);
/* sweep upward diagonally, inserting successive characters,... */
    do {
      if ((row < nrow) && (col >= 0) && (!array[row*ncol+col]))
        utah(row,col,chr++);
      row -= 2; col += 2;
    } while ((row >= 0) && (col < ncol));
    row += 1; col += 3;

/* & then sweep downward diagonally, inserting successive characters,... */
    do {

```

```

        if ((row >= 0) && (col < ncol) && (!array[row*ncol+col]))
            utah(row,col,chr++);
        row += 2; col -= 2;
    } while ((row < nrow) && (col >= 0));
    row += 3; col += 1;

/* ... until the entire array is scanned */
} while ((row < nrow) || (col < ncol));

/* Lastly, if the lower righthand corner is untouched, fill in fixed pattern */
if (!array[nrow*ncol-1]) {
    array[nrow*ncol-1] = array[nrow*ncol-ncol-2] = 1;
}
}

/* "main" checks for valid command line entries, then computes & displays array */
void main(int argc, char *argv[])
{ int x, y, z;

    if (argc <= 3) {
        printf("Command line: ECC200 #_of_Data_Rows #_of_Data_Columns\n");
    } else {
        nrow = ncol = 0;
        nrow = atoi(argv[1]); ncol = atoi(argv[2]);
        if ((nrow >= 6) && (~nrow&0x01) && (ncol >= 6) && (~ncol&0x01)) {
            array = malloc(sizeof(int) * nrow * ncol);

            ecc200();

            for (x=0; x<nrow; x++) {
                for (y=0; y<ncol; y++) {
                    z = array[x*ncol+y];
                    if (z == 0) printf(" WHI");
                    else if (z == 1) printf("BLK");
                    else printf("%3d.%d",z/10,z%10);
                }
                printf("\n");
            }
            free(array);
        }
    }
}

```

M.2 Symbol Character Placement Rules

M.2.1 Non-standard Symbol Character Shapes

Because the standard symbol character shape cannot always fit at the data module boundaries of the symbol and at some corners, a small set of non-standard symbol characters is required. There are six conditions: two boundary conditions which affect all symbol formats and four different corner conditions which apply to certain symbol formats:

- One portion of the symbol character shape is placed on one side and the other on the opposite side. This applies to two basic symbol character shapes (see Figure M1a). Variants of these arrangements concern the row to row relationship between the left and right hand boundary (see Table M1).
- One portion of the symbol character is placed on the top boundary and the other portion on the bottom boundary. This applies to two basic symbol character shapes (see Figure M1b). Variants of these arrangements concern the column to column relationship between the top and bottom boundary (see Table M1).
- Four symbol character shapes are split between two or three corners (see Figures M1c to M1f)

The non-standard symbol shapes are placed at opposite boundaries. The number of these pairings increases in general proportion to the size of the perimeter of the mapping matrix. The basic pattern

is as illustrated in Figures M1a and M1b. In Figure M1a, modules a8 and a7 are in the same row, as are modules b7 and b6. In Figure M1b module c6 and c3 are in the same column as are modules d3 and d1.

There are seven cases for boundary placement, which define the relative vertical position of the symbol characters illustrated in Figure M1a, the horizontal position of the symbol characters illustrated in M1b, and the corner conditions.

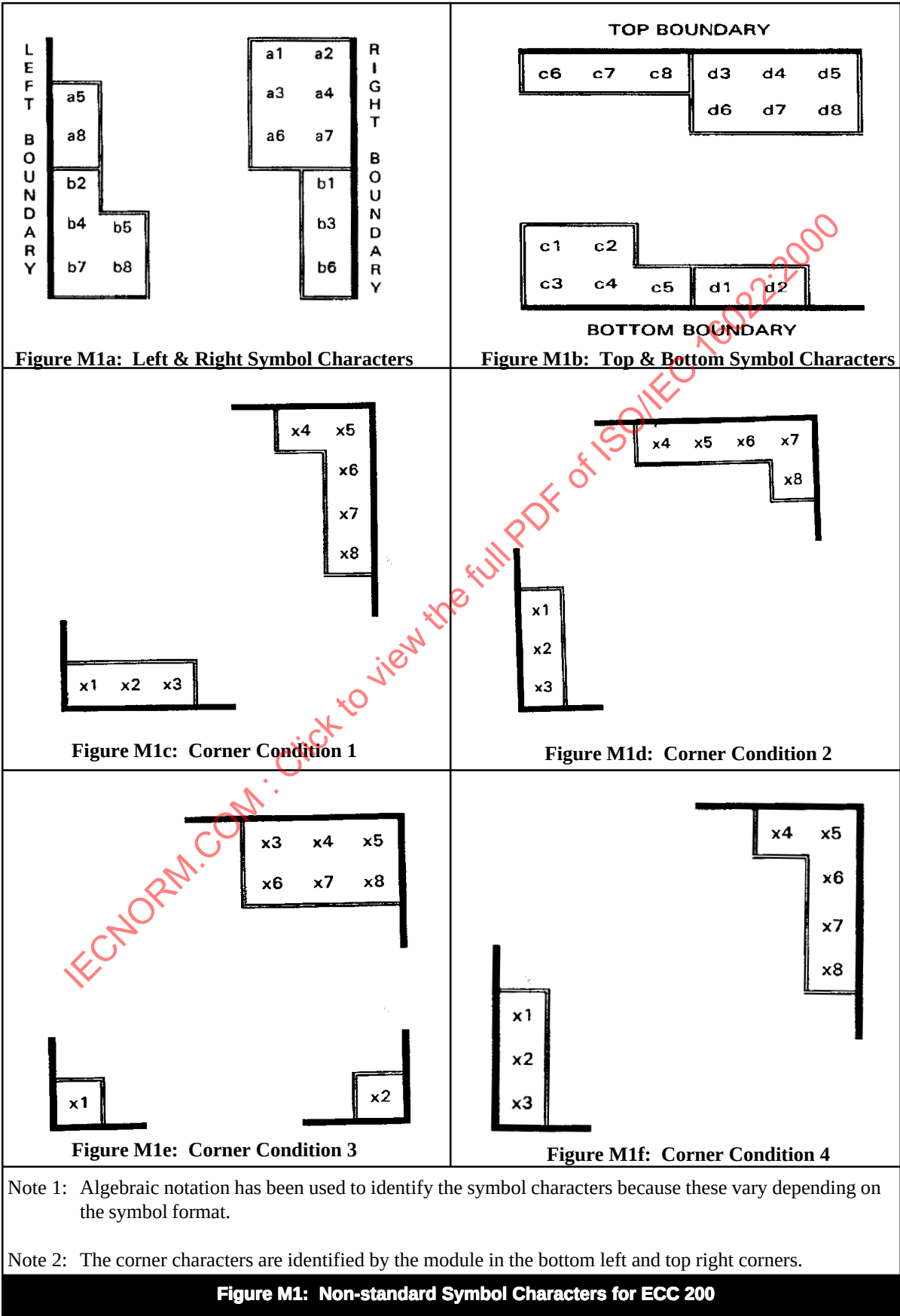
M.2.2 Symbol Character Arrangement

The symbol characters are placed in a matrix in the following manner:

- A mapping matrix is created.
 - For small symbols with only one data region, this equates to the mapping matrix.
 - For larger symbols with more than one data region, the mapping matrix equates to an area the size of the abutted data regions. In effect, the mapping matrix has no separating alignment patterns. For example, the 36 x 36 format symbol has four 16 x 16 data regions which abut to create a mapping matrix 32 x 32. The size of the mapping matrix for each symbol format is given in Table 11. The boundary placement case is given in Table M1.
- Symbol character 2 is placed in the uppermost left position, with its modules conforming to the bit (or module) sequence defined in Figure 8. Using the notation 2.1 to identify module 1 of symbol character 2, this module is in the top row and leftmost column of every mapping

Boundary Placement Case	Row Relationship of Module a8 and a7	Column Relationship of Module c6 and c3	Corner Condition Figure No.	Mapping Matrices Affected	Refer to Annex M Figure No. for Example
1	a7 Row = a8 Row	c3 Column = c6 Column	None	Square: $8^2, 16^2, 24^2, 32^2, 40^2, 48^2, 56^2, 64^2, 72^2, 80^2, 88^2, 96^2, \& 120^2$	Figure M4 & M11
2	a7 Row = a8 Row - 2	c3 Column = c6 Column - 2	None	Square: $10^2 \& 18^2$	Figure M5 & M12
3	a7 Row = a8 Row + 4	c3 Column = c6 Column + 4	M1c	Square: $12^2, 20^2, 28^2, 36^2, 44^2, 108^2, \& 132^2$	Figure M6 & M13
4	a7 Row = a8 Row + 2	c3 Column = c6 Column + 2	M1d	Square: $14^2 \& 22^2$	Figure M7 & M14
5	a7 Row = a8 Row	c3 Column = c6 Column + 2	M1e	Rectangular: 6x16 & 14x32	Figure M8
6	a7 Row = a8 Row	c3 Column = c6 Column - 2	None	Rectangular: 10x24 & 10x32	Figure M9
7	a7 Row = a8 Row + 4	c3 Column = c6 Column + 2	M1f	Rectangular: 6x28 & 14x44	Figure M10

Table M1: Factors which Determine the Boundary Placement Cases



2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8
2.6	2.7	2.8	5.3	5.4	5.5		
1.a	6.1	6.2	5.6	5.7	5.8		
1.b	6.3	6.4	6.5				
	6.6	6.7	6.8				

Note: The values a and b depend on the size of the mapping matrix

Figure M2: Starting Sequence for Module Placement

matrix. The module array sequence shown in Figure M2 is constant for all mapping matrices.

3. The corner shapes are positioned according to Table M1 and the appropriate Figures M1c to M1f. Plotting of the standard symbol character shapes continues, nesting the shapes as illustrated above for symbol characters 2, 5, and 6. The non-standard symbol characters are positioned as per Table M1. This process results in the mapping matrix being completely covered in symbol characters, most of which are un-numbered.
4. The sequence of symbol characters is determined as follows. Symbol characters are arranged on 45 degree parallel diagonal lines between the lower left and upper right, generally linking through the centers on module 8.
5. The first diagonal line starts with the line through module 8 of symbol character 1; this is module 8 except in the case of the 6 x 28 mapping matrix, where the corner condition, as defined in Figure M1f, determines the values of modules in symbol character 1 (i.e. making 1.b equal 1.2). The diagonal line continues through modules 2.8 and 3.8.
6. At this point, the diagonal line crosses the top row boundary. The next diagonal line is started 4 modules to the right in the top row, or

in the case of the 8 x 8 mapping matrix, 3 modules right and 1 module down; i.e. the diagonal line is always displaced by 4 modules. Symbol characters are numbered in order, based on the placement path crossing module 8. Thus the next characters are determined by the downward diagonal line crossing modules 4.8, 5.8, 6.8 and so on.

7. As shown in Figure M3, the placement path continues as diagonal lines four modules to the right (or four modules down, or combinations thereof) from the previous diagonal line. The first, and all odd numbered, diagonal lines map the symbol character sequence from bottom left to top right. The second, and all even numbered, diagonal lines map the symbol character sequence from the top right to the bottom left.
8. When the placement path encounters a non-standard symbol character shape, which is not completely contained within the boundaries of the mapping matrix, that symbol character is continued on the opposite side of the matrix. This has the effect of numbering the opposite portions of these symbol characters before the placement path crosses that position. For example, in the illustrated mapping matrix (see Figure M3) the other portions of symbol character 3 and 7 are pre-numbered before the placement path crosses them. Thus the placement path only numbers un-numbered symbol characters. These boundary and

corner conditions are specified in Table M1. This can be seen in Figure M3 for symbol characters 1, 3, 4, and 7.

The corner conditions also affect the numbering sequence. The bottom left corner as illustrated in:

Figure M1c is numbered immediately before the symbol character above it (see Figures M6 and M13 for examples).

Figure M1d is numbered immediately before the symbol character above it (see Figure M7 and M14 for examples).

Figure M1e is numbered immediately after the symbol character to its right (see Figure M8 for an example).

Figure M1f is numbered immediately before the symbol character above it (see Figure M10 for an example).

The remaining modules of the corner are numbered before the placement path crosses them.

9. The placement procedure continues until all symbol characters are placed, and it ends in the lower right of the mapping matrix. Four sizes of mapping matrix (10 x 10, 14 x 14, 18 x 18, and 22 x 22) have a 2 x 2 area remaining in the bottom right hand corner. The top left and bottom right modules of this area are dark (nominally encoding binary 1). This is illustrated in Figure M3.

Typical mapping matrices conforming to this procedure are illustrated in Annexe M.3. Figures M4 to M10 cover respective cases 1 to 7 for boundary placement. Figure M11 to M14 are another set of examples for cases 1 to 4. Annexe M.1 provides a C language program capable of mapping all encoded bits into the appropriate mapping matrix.

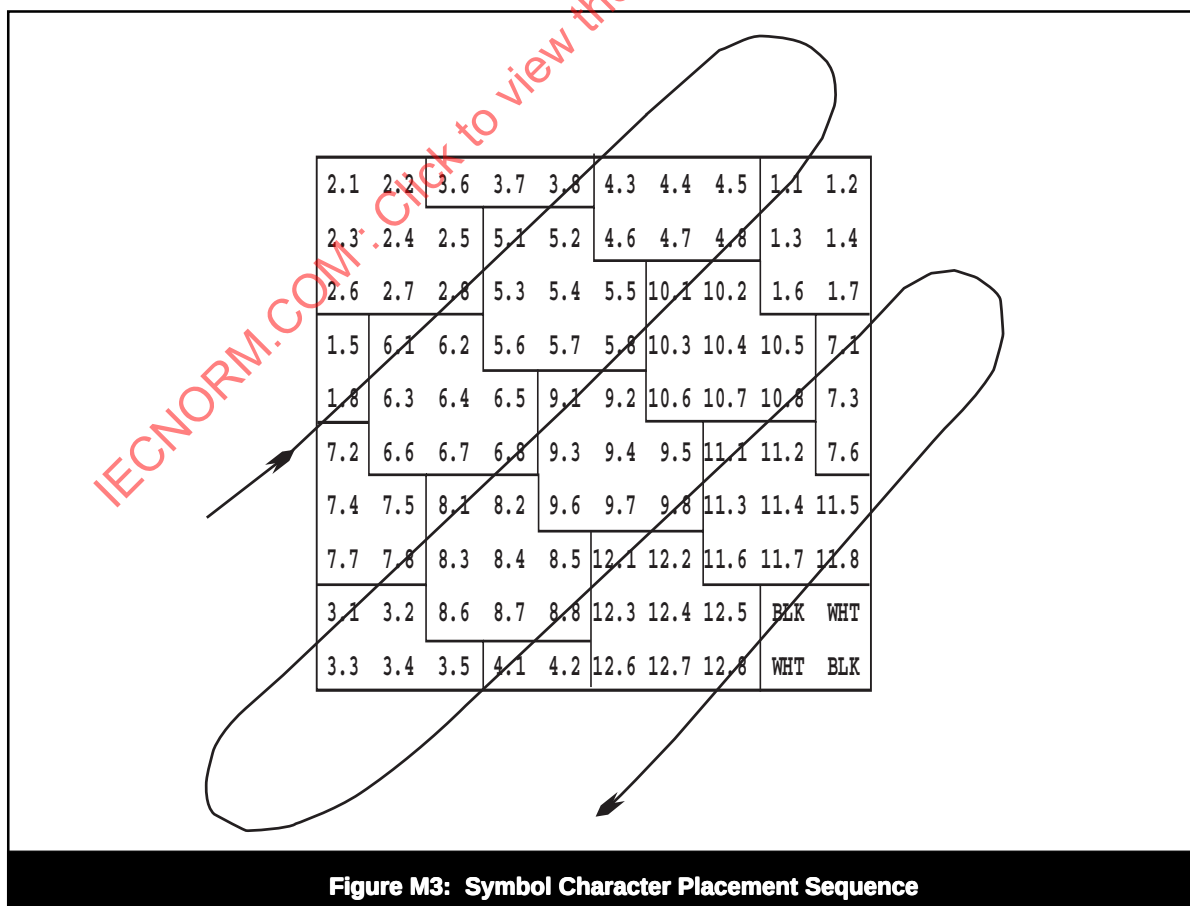


Figure M3: Symbol Character Placement Sequence

M.3 Symbol Character Placement Examples for ECC 200

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8
2.6	2.7	2.8	5.3	5.4	5.5	1.1	1.2
1.5	6.1	6.2	5.6	5.7	5.8	1.3	1.4
1.8	6.3	6.4	6.5	8.1	8.2	1.6	1.7
7.2	6.6	6.7	6.8	8.3	8.4	8.5	7.1
7.4	7.5	3.1	3.2	8.6	8.7	8.8	7.3
7.7	7.8	3.3	3.4	3.5	4.1	4.2	7.6

Figure M4: Codeword Placement for Square Mapping Matrix of size 8

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	1.1	1.2
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	1.3	1.4
2.6	2.7	2.8	5.3	5.4	5.5	10.1	10.2	1.6	1.7
1.5	6.1	6.2	5.6	5.7	5.8	10.3	10.4	10.5	7.1
1.8	6.3	6.4	6.5	9.1	9.2	10.6	10.7	10.8	7.3
7.2	6.6	6.7	6.8	9.3	9.4	9.5	11.1	11.2	7.6
7.4	7.5	8.1	8.2	9.6	9.7	9.8	11.3	11.4	11.5
7.7	7.8	8.3	8.4	8.5	12.1	12.2	11.6	11.7	11.8
3.1	3.2	8.6	8.7	8.8	12.3	12.4	12.5	BLK	WHT
3.3	3.4	3.5	4.1	4.2	12.6	12.7	12.8	WHT	BLK

Figure M5: Codeword Placement for Square Mapping Matrix of size 10

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	13.1	13.2	8.4	8.5
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	13.3	13.4	13.5	8.6
2.6	2.7	2.8	5.3	5.4	5.5	12.1	12.2	13.6	13.7	13.8	8.7
1.5	6.1	6.2	5.6	5.7	5.8	12.3	12.4	12.5	14.1	14.2	8.8
1.8	6.3	6.4	6.5	11.1	11.2	12.6	12.7	12.8	14.3	14.4	14.5
7.2	6.6	6.7	6.8	11.3	11.4	11.5	15.1	15.2	14.6	14.7	14.8
7.4	7.5	10.1	10.2	11.6	11.7	11.8	15.3	15.4	15.5	1.1	1.2
7.7	7.8	10.3	10.4	10.5	16.1	16.2	15.6	15.7	15.8	1.3	1.4
9.1	9.2	10.6	10.7	10.8	16.3	16.4	16.5	18.1	18.2	1.6	1.7
9.3	9.4	9.5	17.1	17.2	16.6	16.7	16.8	18.3	18.4	18.5	7.1
9.6	9.7	9.8	17.3	17.4	17.5	3.1	3.2	18.6	18.7	18.8	7.3
8.1	8.2	8.3	17.6	17.7	17.8	3.3	3.4	3.5	4.1	4.2	7.6

Figure M6: Codeword Placement for Square Mapping Matrix of size 12