The G-CODE

Henry V. Pham

The G-CODE is designed to replace the existing dot-code or data-matrix labels like Barcode, Code-128, QR-CODE or any other Data-Matrix Code labels. The G-CODE can support any matrix data size from 8x8 to 64x64 with increment of 4 dots on each size, and 64x64 to 128x128 with increment of 8 dots on each size, and forever extendable size of 16 dots on each size from 128x128 and up until the computer cannot able to calculate the data checksum with current CPU supports of 64-bit. The G-CODE supports UTF-8, UTF-16 and UTF-32 as long as the user data matrix follows the G-CODE sizes described above. The G-CODE is designed with State Of The Art and is designed to last forever.

The existing Barcode, Code-128 and QR-Code contain low ratio of User Data verse Error Correction Codes. The existing codes cannot able to support dynamic extendable sizes, and do not provide the great Error Correction like the G-CODE does. The G-CODE provides up to 4 ways of line checksums, row-direction checksums, column-direction checksums, backward-diagonal-direction checksums and forward-diagonal-direction checksums. These 4 checksum methods will provide greatest Error Correction algorithm ever for the data matrix. With State Of The Art design, the G-CODE has the Great Eagle Symbol on the top part of the G-CODE frame, and the 4 identical square-corners with option of color or black-and-white mode. The 4 corners are painted with the U.S. flag in color mode, and painted withblack-white-black squares in black-and-white mode. Next to these 4 square-corners, there are 2-pairs of duplicated checksums for entire user data, one for entire user data of rows-checksum and the other for entire user dataof columns-checksum. These user data checksums are also calculated with 4-bit widthin row-direction for user data rows-checksum, with 4-bit widthin column-direction for user data columns-checksum. The G-CODE labels always come with 3-lines border, 2 solid lines and 1 white line. These border lines create a nice looking and great decoration for the G-CODE labels. The user data matrix is in the middle of the G-CODE labels, and the lines between the user data matrix and the border are the rows, columns, and the diagonals checksum lines.The G-CODE checksums are calculated by every 4-bit width of the data matrix for rows, columns and diagonals lines. The magic of 4-bit data checksum is to provide more

www.TheCodeOSCenter.com

condensed data checksum compare to 8-bit or higher, and this 4-bit data checksum method will provide more accurate and focus on scanning for error areas of 4x4 bits. The identity of G-CODE label is the Great Eagle Symbol and the 4square-corners with sizes depend on the size of the user data. The G-CODE label can be scanned in with or without the border lines depend on the user's scanner alignment or the scanner applications.

G-CODE Label:[16x16]:[32x32] Sample

The G-CODE sample matrix Label:[16x16]:[32x32]showing 16x16 user data with total of 32x32 included Frame and Border lines shown in Fig-1 below in color and black-and-white modes is coded for the text "**This is a G-CODE Sample."**. The G-CODE label below contains the data of 16x16 and the 5x lines frame for lines-checksum, plus the 3x border lines.

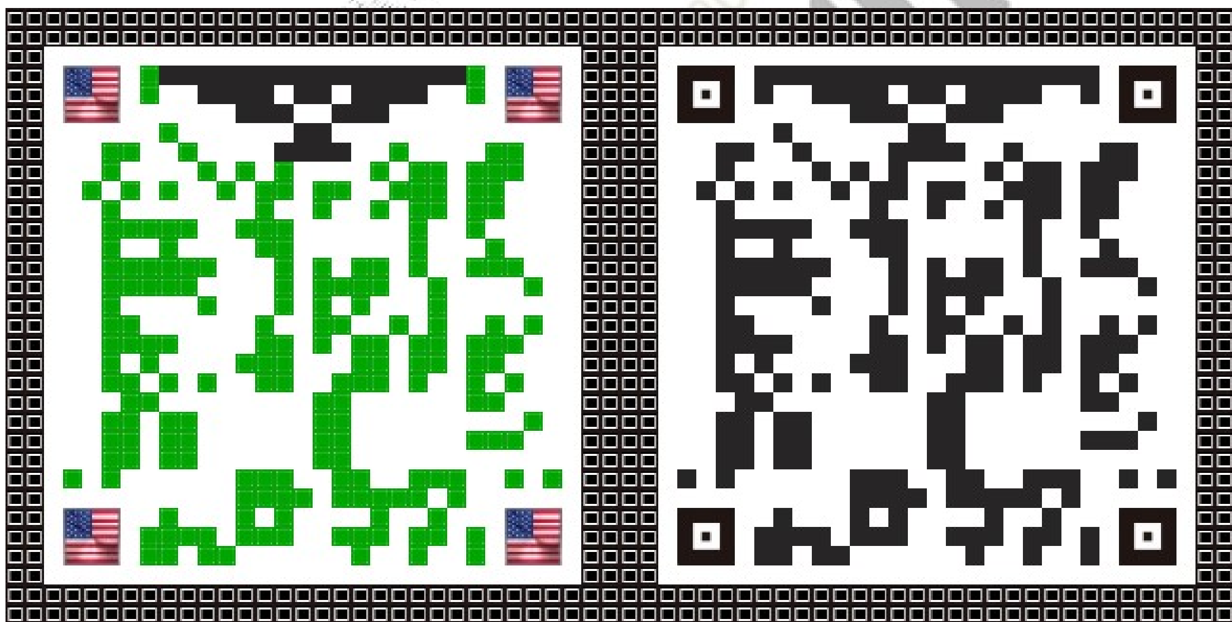


Fig-1: GCODE-Label:[16x16:Data]+[5x:LineChecksum+3x:Border]

This G-CODE label which contains the text "**This is a G-CODE Sample.**" will be used as an example for the Error Correction in this patent. The data in binary of this text is in the yellow square box, and the data checksum of rows, columns and the diagonals lines are on the sides of the G-CODE Label Frame with 5x checksum lines.

COPYRIGHT PROTECTED
www.TheCloudOSCenter.com

The G-CODE

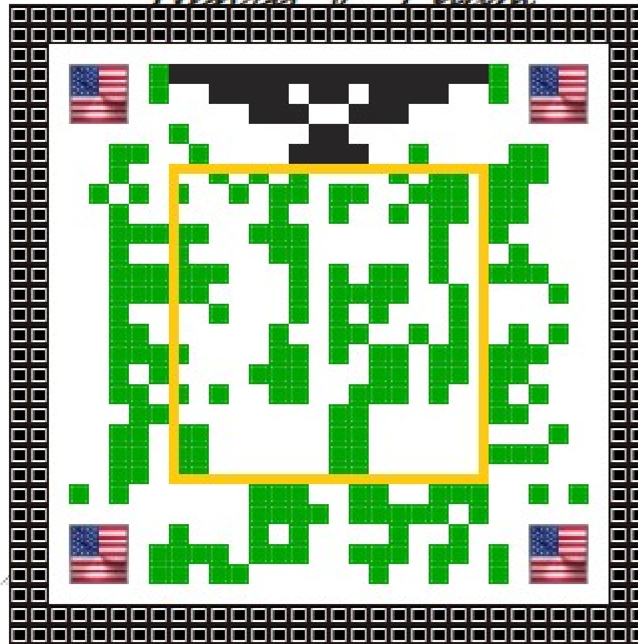


Fig-2: GCODE-Color-Label:[16x16:Data (In-Yellow-Square)]

The G-CODE data from the above label is scanned and transformed to binary matrix format. **Fig-3: GCODE-Data:[16x16]** shows the data in binary matrix of the text "**This is a G-CODE Sample.**" with the row (**r**) and column (**c**) mapping numbers. The checksum for each line of rows, columns and diagonals of this G-CODE is shown in hexadecimal values as below.

Rows 1-16:	13, 29, 13, 1E, 12, 1F, 1F, 10, 1A, 1F, 17, 1B, 0C, 18, 18, 18
Columns 1-16:	1F, 0E, 0E, 04, 0B, 1E, 23, 00, 28, 1E, 0E, 17, 0C, 22, 1F, 00
Backward-Diagonals 1-16:	0D, 14, 0A, 1A, 24, 2D, 24, 0D, 0A, 14, 0D, 0F, 20, 24, 22, 25
Forward-Diagonals 1-16:	1B, 1D, 06, 0A, 2E, 19, 25, 26, 0E, 18, 0E, 0B, 02, 25, 26, 26

COPYRIGHT © PROTECTED
www.TheCloudOSCenter.com

r\c	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	
1	0	0	1	0	1	0	1	0	0	0	0	1	0	1	1	0	1
2	1	0	0	1	0	1	1	0	1	1	0	0	1	1	1	0	2
3	0	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0	3
4	1	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0	4
5	1	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	5
6	1	1	1	0	0	0	1	0	1	0	1	1	0	1	0	0	6
7	1	1	0	0	0	0	1	0	1	1	1	1	0	0	1	0	7
8	0	0	1	0	0	0	1	0	1	0	1	0	0	0	1	0	8
9	0	0	0	0	0	1	0	0	1	1	0	0	1	0	1	0	9
0	1	0	0	0	0	1	1	0	1	0	1	1	0	1	1	0	0
1	0	0	0	0	1	1	1	0	0	0	1	1	0	1	1	0	1
2	1	0	1	0	0	1	1	0	0	1	1	1	0	1	0	0	2
3	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	3
4	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	4
5	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	5
6	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	6
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	c\r

Fig-3: GCODE-Data:[16x16]

The row R(n) and column C(n) checksums calculation for each 4-bit width data as followings from Fig-3:

R(1): Binary:[0010+1010+0001+0110] => Hex:[2+A+1+6] = **0x13**

R(2): Binary:[1001+0110+1100+1110] => Hex:[9+6+C+E] = **0x29**

..Etc…

C(1): Binary:[0101+1110+0101+0111] => Hex:[5+E+5+7] = **0x1F**

C(2): Binary:[0001+0110+0000+0111] => Hex:[1+6+0+7] = **0x0E**

..Etc…

COPYRIGHT © PROTECTED --HVPHAM
Henry V. Pham <HenryVPham@TheCloudOSCenter.com>
www.TheCloudOSCenter.com
Henry V. Pham

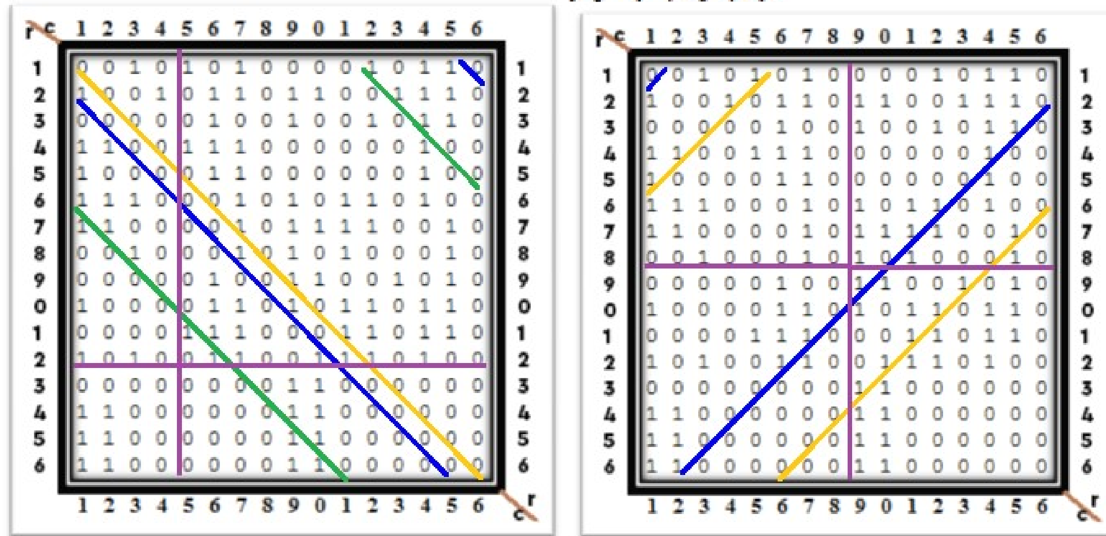


Fig-4:GCODE-Data:[16x16]

The (left) Backward Diagonal (Xb), and (right) Forward Diagonal (Xf) checksums calculation using "light-beam" reflection method for each 4-bit data as followings from Fig-4:

Xb(1)[r1c1,r16c16]:

Binary:[0000+0010+1011+0000] => Hex:[0+2+B+0] = **0x0D**

Xb(2)[r2c1,r16c15]+[r1c16,r1c16] – Reflect to r1c16 at point r16c15 :

Binary:[1000+0010+1010+0000] => Hex:[8+2+A+0] = **0x14**

..

Xb(6)[r6c1,r16c11]+[r1c12,r5c16] – Reflect to r1c12 at point r16c11:

Binary:[1110+0110+1101+1100] => Hex:[E+6+D+C] = **0x2D**

..Etc…

Xf(1)[r1c1,r1c1]+[r16c2,r2c16] – Reflect to r16c2 at point r1c1:

Binary:[0100+0110+1011+0110] => Hex:[4+6+B+6] = **0x1B**

..

Xf(5)[r5c1,r1c5]+[r16c6,r6c16] – Reflect to r16c6 at point r1c5:

Binary:[1101+1000+1111+1010] => Hex:[D+8+F+A] = **0x2E**

..Etc…

COPYRIGHT© PROTECTED -HVPHAM
Henry V. Pham@TheCloudOSCenter.com
www.TheCloudOSCenter.com

The R(n) rows and C(n) columns checksum lines are always printed on the label, but the Xb(n) and Xf(n) checksum lines are printed on the label only for some selected lines to minimize the G-CODE matrix size. However, the larger the G-CODE matrix size is, the more Xb(n) and Xf(n) checksum lines are printed on the label. For this sample of G-CODE label, only Xb(1):=0x0D, Xb(5):=0x24, Xb(9):=0x0A and Xb(13):=0x20 are printed on the label. These checksums are scanned back with the rows and columns line-checksums and use for the Error Correction. These Xb(n) and Xf(n) checksums are useful when the cross R(n)or C(n) checksum lines are not scanned in correctly or missing dots.

The G-CODE provides greatest Error Correction algorithm ever for the data-matrix. The algorithm is using the cross-checking methodology, each line of R(n) vs. C(n) can only be crossed at one point. If the crossing point (dot) is missing, the checksum of these crossed R(n) and C(n) are missing a '1'. The same method applies to R(n) vs. Xb(n), R(n) vs. Xf(n), C(n) vs. Xb(n), C(n) vs. Xf(n), and Xb(n) vs. Xf(n).

Row Error Correction

Let's start trying a scratched-row Error Correction in Fig-5 for the yellow scratched line R(6) for the missing dots. Assuming that we have all the R(n) and C(n) checksum lines are corrected after scanned in from the label. The algorithm will scan for errors using R(n) and C(n) cross-checking method.Fig-5 on the right hand side will mark '**E**' for entire R(6) because all rows have correct checksum except R(6), and of course C(n) calculated checksum lines are not correct. At this point, we know only R(6) has incorrect data.

COPYRIGHT PROTECTED
www.TheCloudOSCenter.com

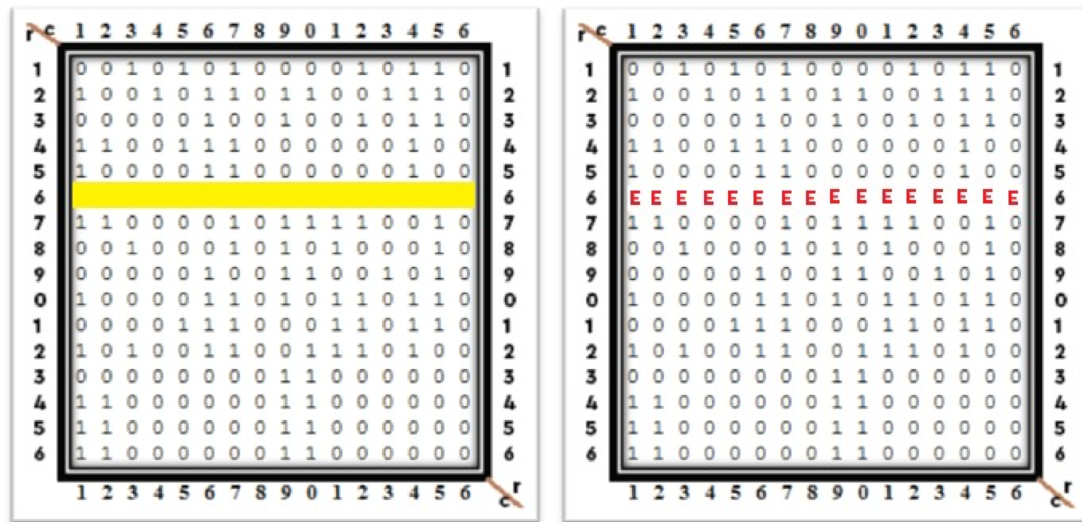


Fig-5: GCODE-Data:[16x16] with scratched row

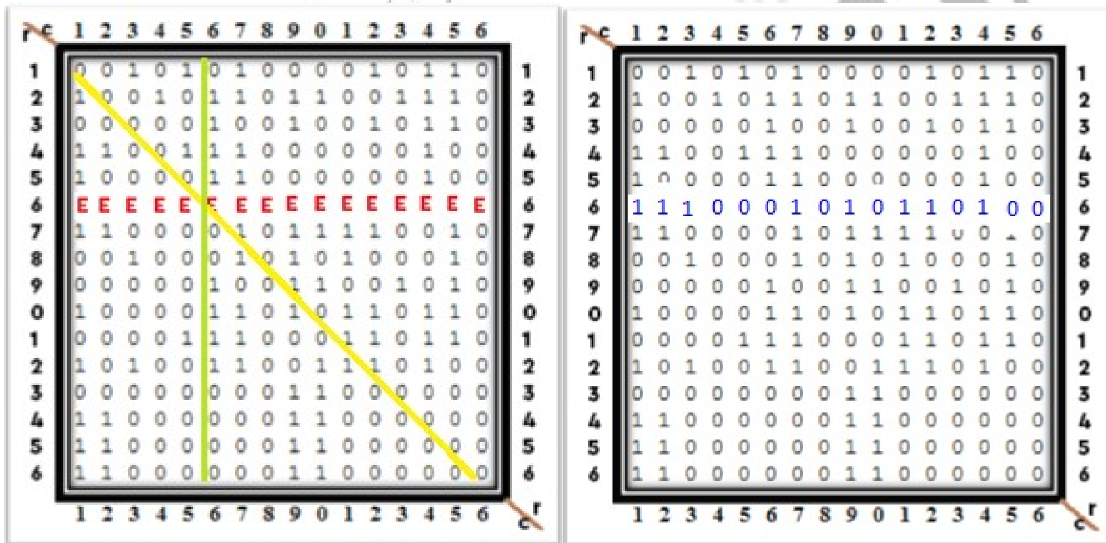


Fig-6: GCODE-Data:[16x16] with scratched row and corrected

First, assume R(6) has all '**0**', then the column calculated checksums C(1) to C(16) in **red** are missing '1' at row R(6) compare to the original checksums got from scanned label. From the columns calculated checksums below, we can conclude that the red bit in Binary should be '**1**' for the C(n) calculation below.

R(1)-R(16): **13, 29, 13, 1E, 12, 00, 1F, 10, 1A, 1F, 17, 1B, 0C, 18, 18, 18**

C(1)-C(16): **1B, 0A, 0A, 04, 0B, 1E, 1F, 00, 24, 1E, 0A, 13, 0C, 1E, 1F, 00**

www.TheCloudOSCenter.com

C(1):=Bin:[0101+1010+0101+0111] = Hex:[5+A+5+7] = 0x1B (Add 4 to be **0x1F**)

C(2):=Bin:[0001+0010+0000+0111] = Hex:[1+2+0+7] = 0x0A (Add 4 to be **0x0E**)

C(3):=Bin:[1000+0001+0001+0000] = Hex:[8+1+1+0] = 0x0A (Add 4 to be **0x0E**)

C(7) :=Bin:[1101+1011+0111+0000] = Hex:[D+B+7+0] = 0x1F (Add 4 to be **0x23**)

C(9) :=Bin:[0110+0011+1100+1111] = Hex:[6+3+C+F] = 0x24 (Add 4 to be **0x28**)

C(11):=Bin:[0000+0011+0111+0000] = Hex:[0+3+7+0] = 0x0A (Add 4 to be **0x0E**)

C(12):=Bin:[1010+0010+0111+0000] = Hex:[A+2+7+0] = 0x13 (Add 4 to be **0x17**)

C(14):=Bin:[1111+1000+0111+0000] = Hex:[F+8+7+0] = 0x1E (Add 4 to be **0x22**)

Second, assume R(6) has all '**1**', then the column calculated checksums C(1) to C(16) in **red** are having extra '1' at row R(6) compare to the original checksums got from the scanned label. From the columns calculated checksums below, we can conclude that the red bit in Binary should be '**0**' for the C(n) calculation below.

R(1)-R(16): **13, 29, 13, 1E, 12, 00, 1F, 10, 1A, 1F, 17, 1B, 0C, 18, 18, 18**

C(1)-C(16): **1F, 0E, 0E, 08, 0F, 22, 23, 04, 28, 22, 0E, 17, 10, 22, 23, 04**

C(4) :=Bin:[0100+0100+0000+0000] = Hex:[4+4+0+0] = 0x08 (Subtract 4 to be **0x04**)

C(5) :=Bin:[1001+0100+0010+0000] = Hex:[9+4+2+0] = 0x0F (Subtract 4 to be **0x0B**)

C(6) :=Bin:[0111+1100+1111+0000] = Hex:[7+C+F+0] = 0x22 (Subtract 4 to be **0x1E**)

C(8) :=Bin:[0000+0100+0000+0000] = Hex:[0+4+0+0] = 0x04 (Subtract 4 to be **0x00**)

C(10):=Bin:[0100+0110+1001+1111] = Hex:[4+6+9+F] = 0x22 (Subtract 4 to be **0x1E**)

C(13):=Bin:[0100+0100+1000+0000] = Hex:[4+4+8+0] = 0x10 (Subtract 4 to be **0x0C**)

C(15):=Bin:[1110+0111+1110+0000] = Hex:[E+7+E+0] = 0x23 (Subtract 4 to be **0x1F**)

C(16):=Bin:[0000+0100+0000+0000] = Hex:[0+4+0+0] = 0x04 (Subtract 4 to be **0x00**)

Third, using C(n) vs Xb(n) method we can correct and confirm C(6) and Xb(1) checksums. C(6) and Xb(1) cross at r6c6, and we have Xb(1) checksum equals 0x0D, and C(6) with checksum of 0x1E, therefore the cross-dot at R(6) and C(6) must be '**0**'. By using this Error Correction method recursively, we can correct all these missing dots faster and easier.

www.TheCloudOSCenter.com

Finally, we can fill back the data matrix with the correct data in Fig-6 on the right side by the below checksums. The G-CODE label also provides two ways of entire user data checksums which are scanned back from the corners of the G-CODE label. The row-direction user data checksum is '0x018C' and column-direction user data checksum is '0x0143', and we can confirm the calculated both directions user data checksums are matched with the user data checksum of both directions that scanned in from the label.

R(1)-R(16): **13, 29, 13, 1E, 12, 1F, 1F, 10, 1A, 1F, 17, 1B, 0C, 18, 18, 18**

C(1)-C(16): **1F, 0E, 0E, 04, 0B, 1E, 23, 00, 28, 1E, 0E, 17, 0C, 22, 1F, 00**

Column Error Correction

Now let's try a scratched-column Error Correction in Fig-7 for the yellow scratched column C(14) for the missing dots. Assuming that we have all the R(n) and C(n) checksum lines are corrected after scanned in from the label. The algorithm will scan for errors using R(n) and C(n) cross-checking method. Fig-7 on the right hand side will mark 'E' for entire C(14) because all columns have correct checksum except C(14), and of course R(n) calculated checksum lines are not correct. At this point, we know only C(14) has incorrect data.

Fig-7: GCODE-Data:[16x16] with scratched column

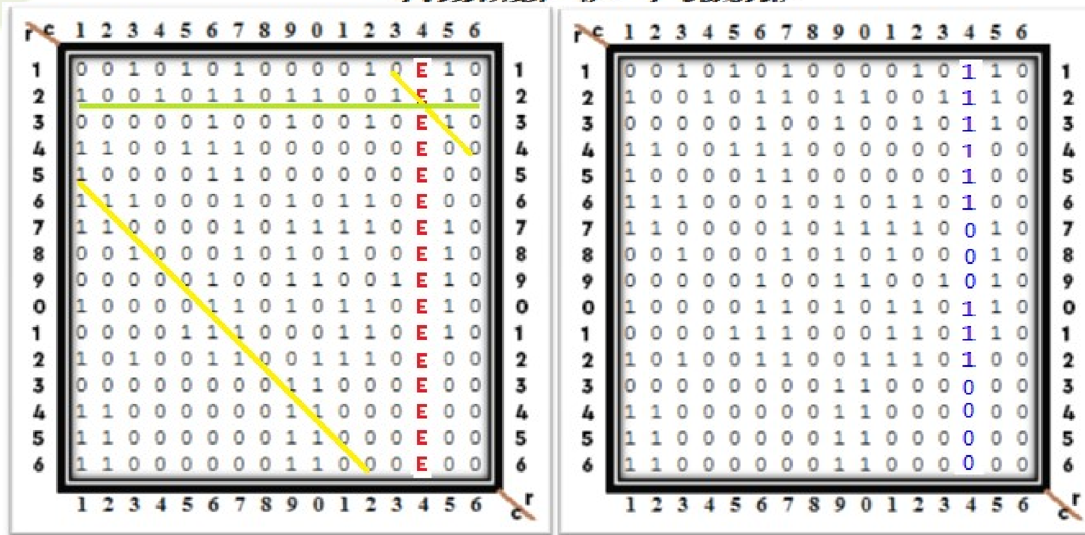


Fig-8: GCODE-Data:[16x16] with scratched column and corrected

First, assume C(14) has all '**0**', then the row checksums in **red** are missing '1' at column C(14) compare to the original checksums got from scanned label. From the rows checksums below, we can conclude that the red bit in Binary should be '**1**' for the R(n) calculation below.

R(1)-R(16): **0F, 25, 0F, 1A, 0E, 1B, 1F, 10, 1A, 1B, 13, 17, 0C, 18, 18, 18**

C(1)-C(16): **1F, 0E, 0E, 04, 0B, 1E, 23, 00, 28, 1E, 0E, 17, 0C, 00, 1F, 00**

R(1) :=Bin:[0010+1010+0001+0**0**10] = Hex:[2+A+1+2] = 0x0F (Add 4 to be **0x13**)

R(2) :=Bin:[1001+0110+1100+1**0**10] = Hex:[9+6+C+A] = 0x25 (Add 4 to be **0x29**)

R(3) :=Bin:[0000+0100+1001+0**0**10] = Hex:[0+4+9+2] = 0x0F (Add 4 to be **0x13**)

R(4) :=Bin:[1100+1110+0000+0**0**00] = Hex:[C+E+0+0] = 0x1A (Add 4 to be **0x1E**)

R(5) :=Bin:[1000+0110+0000+0**0**00] = Hex:[8+6+0+0] = 0x0E (Add 4 to be **0x12**)

R(6):=Bin:[1110+0010+1011+0**0**00] = Hex:[E+2+B+0] = 0x1B (Add 4 to be **0x1F**)

R(10):=Bin:[1000+0110+1011+0**0**10] = Hex:[8+6+B+2] = 0x1B (Add 4 to be **0x1F**)

R(11):=Bin:[0000+1110+0011+0**0**10] = Hex:[0+E+3+2] = 0x13 (Add 4 to be **0x17**)

R(12):=Bin:[1010+0110+0111+0**0**00] = Hex:[A+6+7+0] = 0x17 (Add 4 to be **0x1B**)

PROTECTED
www.TheCloudOSCenter.com

Second, assume C(14) has all '**1**', then the row checksums in **red** are having extra '1' at column C(14) compare to the original checksums got from scanned label. From the rows checksums below, we can conclude that the red bit in Binary should be '**0**' for the R(n) calculation below.

R(1)-R(16): **13, 29, 13, 1E, 12, 1F, 23, 14, 1E, 1F, 17, 1B, 10, 1C, 1C, 1C**

C(1)-C(16): **1F, 0E, 0E, 04, 0B, 1E, 23, 00, 28, 1E, 0E, 17, 0C, 00, 1F, 00**

R(7) :=Bin:[1100+0010+1111+0**1**10] = Hex:[C+2+F+6] = 0x23 (Subtract 4 to be **0x1F**)

R(8) :=Bin:[0010+0010+1010+0**1**10] = Hex:[2+2+A+6] = 0x14 (Subtract 4 to be **0x10**)

R(9) :=Bin:[0000+0100+1100+1**1**10] = Hex:[0+4+C+E] = 0x1E (Subtract 4 to be **0x1A**)

R(13):=Bin:[0000+0000+1100+0**1**00] = Hex:[0+0+C+4] = 0x10 (Subtract 4 to be **0x0C**)

R(14):=Bin:[1100+0000+1100+0**1**00] = Hex:[C+0+C+4] = 0x1C (Subtract 4 to be **0x18**)

R(15):=Bin:[1100+0000+1100+0**1**00] = Hex:[C+0+C+4] = 0x1C (Subtract 4 to be **0x18**)

R(16):=Bin:[1100+0000+1100+0**1**00] = Hex:[C+0+C+4] = 0x1C (Subtract 4 to be **0x18**)

Third, using C(n) vs Xb(n) cross-checking method we can correct and confirm R(2) and Xb(5) checksums. R(2) and Xb(5) cross at r2c14, and we have Xb(5) checksum equals 0x24, and R(2) with checksum of 0x29, therefore the dot at r2c14 must be '1'. R(2):=Bin:[1001+0110+1100+1**1**10] = Hex:[9+6+C+E] = **0x29** and Xb(5):=Bin:[1100+0110+1100+0**1**10] = Hex:[C+6+C+6] = **0x24**. By using this Error Correction method recursively, we can correct all these missing dots faster and easier.

Finally, we can fill back the data matrix with the correct data in Fig-8 on the right side by the below checksums. The G-CODE label also provides two ways of entire user data checksums which are scanned back from the corners of the G-CODE label. The row-direction user data checksum is '0x018C' and column-direction user data checksum is '0x0143', and we can confirm the user calculated data checksums are matched with the user data checksum of both directions that scanned in from the label.

R(1)-R(16): **13, 29, 13, 1E, 12, 1F, 1F, 10, 1A, 1F, 17, 1B, 0C, 18, 18, 18**

C(1)-C(16): **1F, 0E, 0E, 04, 0B, 1E, 23, 00, 28, 1E, 0E, 17, 0C, 22, 1F, 00**

www.TheCloudOSCenter.com

Scratched Area Error Correction

Let's start with more complex scratched data label, we can try the Error Correction for scratched area in Fig-9with yellow scratched area of 10 missing dots. Assuming that we have all the R(n) and C(n) checksum lines are corrected after scanned in from the label.The algorithm will scan for errors using R(n) and C(n) cross-checking method.Fig-10 on the left hand side will mark 'E' with value of '0' as an assumption for entire yellow area plus 2 dots at r7c9 and r8c9 because the R(6), R(7), R(8), C(9), C(10), C(11) and C(12) calculated checksum lines are not correct.

r \ c	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	
1	0	0	1	0	1	0	1	0	0	0	0	1	0	1	1	0	1
2	1	0	0	1	0	1	1	0	1	1	0	0	1	1	1	0	2
3	0	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0	3
4	1	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0	4
5	1	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	5
6	1	1	1	0	0	0	1	0					0	1	0	0	6
7	1	1	0	0	0	0	1	0	1				0	0	1	0	7
8	0	0	1	0	0	0	1	0	1				0	0	1	0	8
9	0	0	0	0	0	1	0	0	1	1	0	0	1	0	1	0	9
0	1	0	0	0	0	1	1	0	1	0	1	1	0	1	1	0	0
1	0	0	0	0	1	1	1	0	0	0	1	1	0	1	1	0	1
2	1	0	1	0	0	1	1	0	0	1	1	1	0	1	0	0	2
3	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	3
4	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	4
5	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	5
6	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	6
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	c \ r

Fig-9: GCODE-Data:[16x16] with scratched area

For this type of scratched errors, the algorithm will search for the checksum line with only one 'E' dot like Xb(11) at point r6c12, Xb(16) at point r8c9, Xf(14) at point r6c9, and Xf(3) at point r8c12. However, in this G-CODE sample label Data:[16x16]:[32x32] does not have these Xb(11), Xb(16), Xf(14) and Xf(3). We do have Xb(1), Xb(5), Xb(9) and Xb(13) checksum lines. In this case, the algorithm will chose the most crossing point available checksum lines, and this would be Xb(13), R(7) and C(11) at point r7c11 as shown in Fig-10 below.

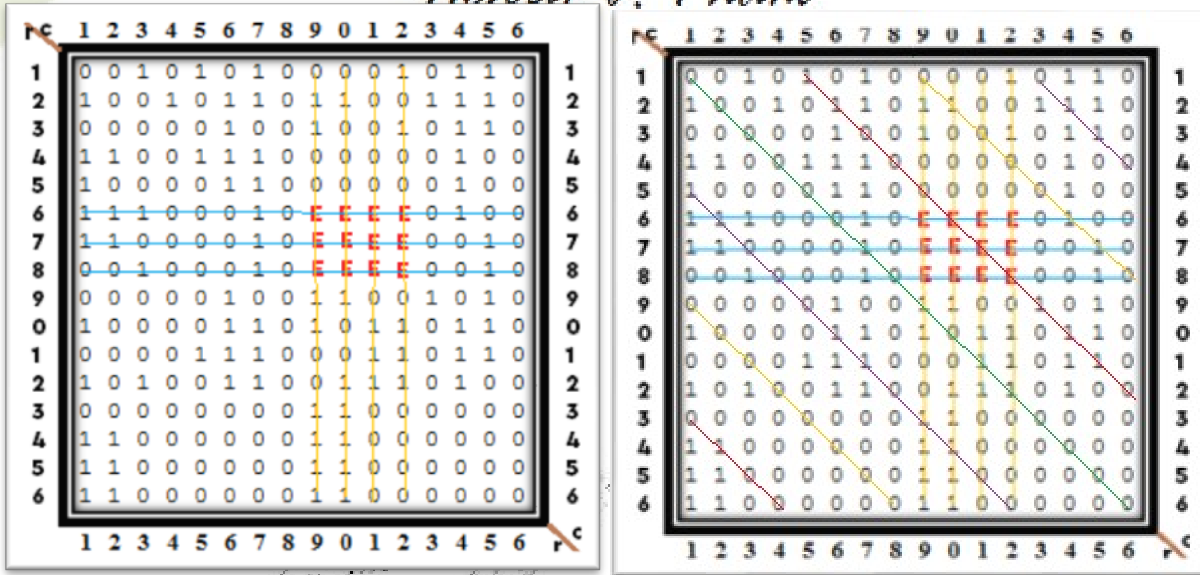


Fig-10: GCODE-Data:[16x16] with scanned forError dots by [rows/columns]

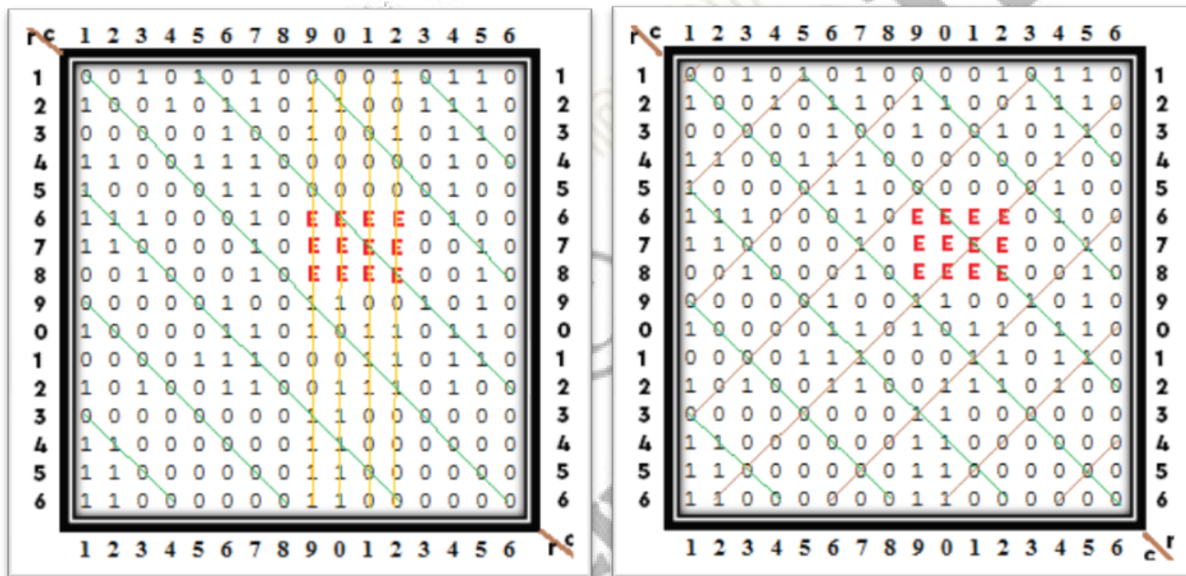


Fig-11: GCODE-Data:[16x16] with scanned for Error dots by [columns/backward-diagonals]

As described above, the algorithm will chose point r7c11 and do error correction at this point by play a series of data dots at points r7c9, r7c10, r7c11, r7c12, r6c11, r8c11, r6c10, and r8c12 with the values from '**0**' to '**1**' for each of these points until all 3 calculated checksum lines Xb(13), R(7) and C(11) are matched with the checksums scanned in from the label as shown in Fig-12 below. The algorithm will chose the shortest or the least crossing-points checksum line to play first, then pick the next shortest crossing-points checksum line to play for

data checksum matching. The algorithm also calculates for how many '**1s**' and how many '**0s**' for each checksum line based on their original checksums to play with for faster Error Correction.

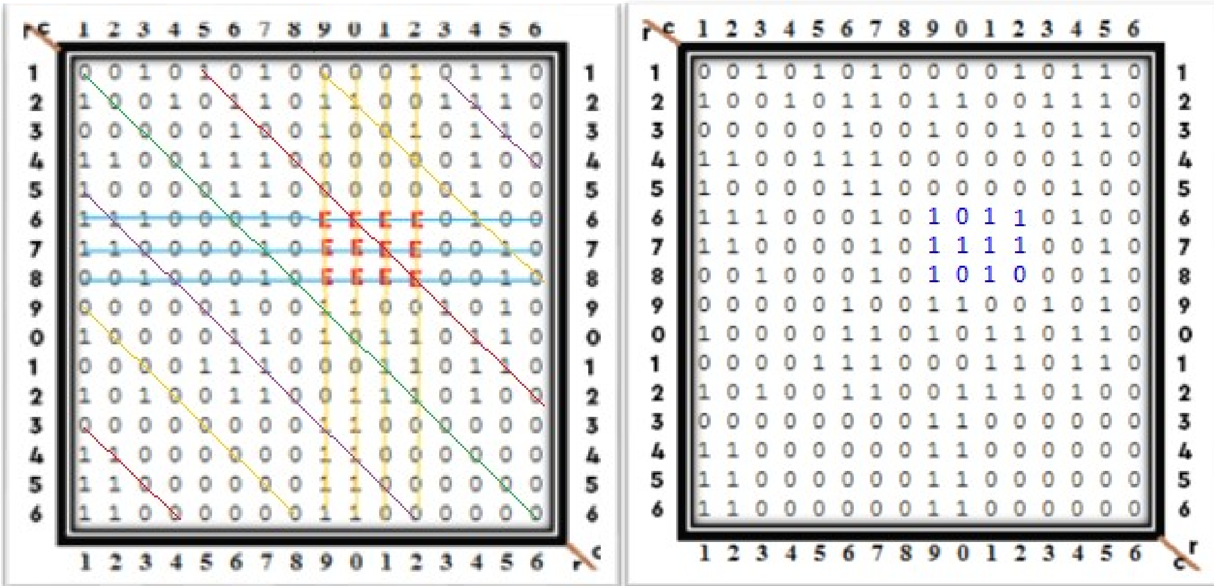


Fig-12: GCODE-Data:[16x16] Error dots Correction

Then the algorithm will use these corrected dots at r7c9:='**1**', r7c10:='**1**', r7c11:='**1**', r7c12:='**1**', r6c11:='**1**', r8c11:='**1**', r6c10:='**0**', and r8c12:='**0**' to correct other points by using the R(n) and C(n) cross-checking method. This would be easier after the 3Xs checksum lines correction. The end result shown in Fig-12 on the right side. By using this Error Correction method recursively, we can correct more of these missing dots faster and easier. This Error Correction algorithm can able to correct more than one scratched areas.

Finally, we can fill back the data matrix with the correct data in Fig-12 on the right side with the above Error Correction. The G-CODE label also provides two ways of entire user data checksums which are scanned back from the corners of the G-CODE label. The row-direction user data checksum is '0x018C' and column-direction user data checksum is '0x0143', and we can confirm the user data checksums are matched with the user data checksum of both directions that scanned in from the label.

www.TheCloudOSCenter.com

User Input ASCII-Text or Binary-Data Matrix

The above sample of G-CODE label can be input in 2 different forms. First, the GCODE Creator can take ASCII format “**This is a G-CODE Sample.**”, and the G-CODE Creator will pad EOT characters (End-Of-Text with value 0x03) if the text does not meet the requirement of G-CODE data matrix sizeof the matrix desired size. Note that the G-CODE labels showing in this patent have a ‘NULL’ character separator with value ‘0x00’ between the text and the padding EOT characters just to show the divider for easier data checking. Second, the GCODE Creator can take input data matrix in CSV format like **Fig-13: GCODE User Input Matrix Data:[16x16] in CSV format** below with pre-padding EOT characters or a user padding characters to fit the matrix desired size The G-CODE Creator also support comment lines starting with ‘#’ character for the CSV format. The comment lines must be before or after the data matrix lines. The user data matrix desired sizes can be multiple of 4x for data matrix from 8x8 to 64x64, multiple of 8x for data matrix from 64x64 to 128x128, and multiple of 16x for data matrix from 128x128 and above.

```
0,0,1,0,1,0,1,0,0,0,0,1,0,1,1,0
1,0,0,1,0,1,1,0,1,1,0,0,1,1,1,0
0,0,0,0,0,1,0,0,1,0,0,1,0,1,1,0
1,1,0,0,1,1,1,0,0,0,0,0,0,1,0,0
1,0,0,0,0,1,1,0,0,0,0,0,0,1,0,0
1,1,1,0,0,0,1,0,1,0,1,1,0,1,0,0
1,1,0,0,0,0,1,0,1,1,1,1,0,0,1,0
0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,0
0,0,0,0,0,1,0,0,1,1,0,0,1,0,1,0
1,0,0,0,0,1,1,0,1,0,1,1,0,1,1,0
0,0,0,0,1,1,1,0,0,0,1,1,0,1,1,0
1,0,1,0,0,1,1,0,0,1,1,1,0,1,0,0
0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0
1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0
1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0
1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0
```

Fig-13: GCODE-User-Input-Matrix:[16x16:Data] in CSV format

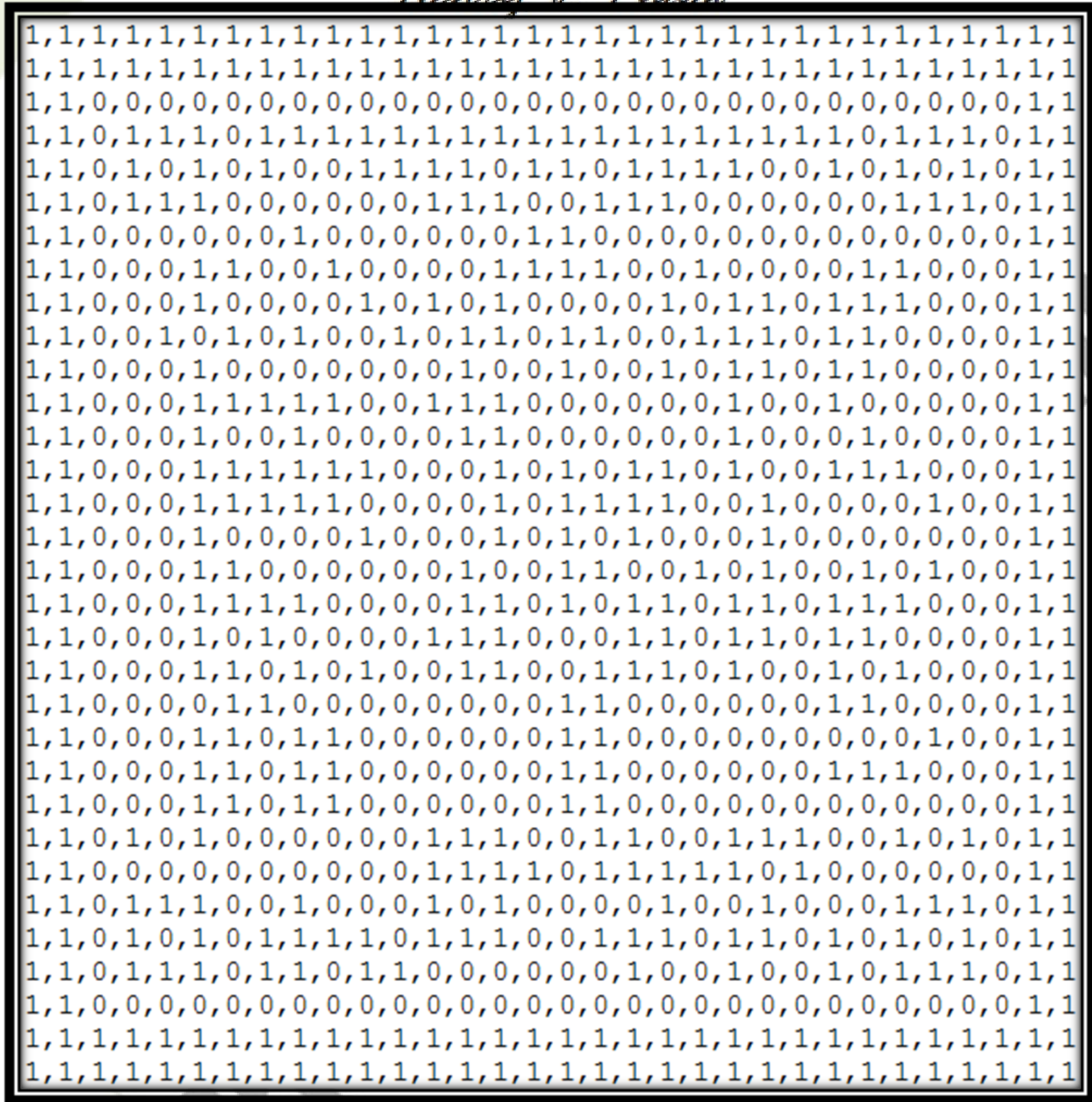
COPYRIGHT © PROTECTED
Henry V. Pham
www.TheCloudOSCenter.com

G-CODE Data Matrix Label Format

With the same above sample of G-CODE label, the G-CODE Creator will create the label matrix contains same user data size in the middle of the label and create the checksum lines with the checksum width from 7-bit and up to 64-bit checksum or the maximum that the current CPU can handle. The G-CODE Creator has an option to encode the user data, and the G-CODE Data-Reverter will decode user data when scan the G-CODE label to get back the user data for secured purposes. The G-CODE frame will hold the 4Xs checksum lines and fill in the R(n) and C(n) checksum lines, but only fill in the Xb(n) and Xf(n) when the G-CODE label has enough spaces available by the calculation of number of lines-checksum bits. The G-CODE Creator also calculates the minimum requirement to create number of frame lines for a given data matrix. In this sample of G-CODE label, the G-CODE frame only has 5x lines. There are 4 square-corners with identical dots or image with the same size depends on the data matrix size. The rears of these 4 square-corners are also printed with 2-pairs of entire user data checksums of both row-direction, and column-direction checksums. The G-CODE label always paint with 3x border lines, 2-solid lines and 1-inner-white line as shown in Fig-14. When scanned in the G-CODE label, the scanner application can take the data matrix with or without border lines. Fig-15 shows the G-CODE label scanned data matrix without the border lines.

The website http://gcode-creator.com will provide the users to create their G-CODE labels in image ".png" format. This website also provides the links to download the **GCODEDataReverter.jar** which allows the user to import or use this as a tool to scan and convert the G-CODE labels to the user data. This GCODEDataReverter.jar file is written in Java programming language to support any platforms. For others programming languages like C/C++ can use system calls with input/output files as the program parameters.

COPYRIGHT © PROTECTED
@HVPHAM
www.TheCloudOSCenter.com



```
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1
1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,1,1
1,1,0,1,0,1,0,1,0,0,1,1,1,1,0,1,1,0,1,1,1,1,0,0,1,0,1,0,1,0,1,1
1,1,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,1,1
1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1
1,1,0,0,0,1,1,0,0,1,0,0,0,0,1,1,1,1,0,0,1,0,0,0,0,1,1,0,0,0,1,1
1,1,0,0,0,1,0,0,0,0,1,0,1,0,1,0,0,0,0,1,0,1,1,0,1,1,1,0,0,0,1,1
1,1,0,0,1,0,1,0,1,0,0,1,0,1,1,0,1,1,0,0,1,1,1,0,1,1,0,0,0,0,1,1
1,1,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,1,1,0,1,1,0,0,0,0,1,1
1,1,0,0,0,1,1,1,1,1,0,0,1,1,1,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,1
1,1,0,0,0,1,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,1,1
1,1,0,0,0,1,1,1,1,1,1,0,0,0,1,0,1,0,1,1,0,1,0,0,1,1,1,0,0,0,1,1
1,1,0,0,0,1,1,1,1,1,0,0,0,0,1,0,1,1,1,1,0,0,1,0,0,0,0,1,0,0,1,1
1,1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,1,1
1,1,0,0,0,1,1,0,0,0,0,0,0,1,0,0,1,1,0,0,1,0,1,0,0,1,0,1,0,0,1,1
1,1,0,0,0,1,1,1,1,0,0,0,0,1,1,0,1,0,1,1,0,1,1,0,1,1,1,0,0,0,1,1
1,1,0,0,0,1,0,1,0,0,0,0,1,1,1,0,0,0,1,1,0,1,1,0,1,1,0,0,0,0,1,1
1,1,0,0,0,1,1,0,1,0,1,0,0,1,1,0,0,1,1,1,0,1,0,0,1,0,1,0,0,0,1,1
1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,1,1
1,1,0,0,0,1,1,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,1,1
1,1,0,0,0,1,1,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,1,0,0,0,1,1
1,1,0,0,0,1,1,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1
1,1,0,1,0,1,0,0,0,0,0,0,1,1,1,0,0,1,1,0,0,1,1,1,0,0,1,0,1,0,1,1
1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,1,1,1,1,0,1,0,0,0,0,0,0,1,1
1,1,0,1,1,1,0,0,1,0,0,0,1,0,1,0,0,0,0,1,0,0,1,0,0,0,1,1,1,0,1,1
1,1,0,1,0,1,0,1,1,1,1,0,1,1,1,0,0,1,1,1,0,1,1,0,1,0,1,0,1,0,1,1
1,1,0,1,1,1,0,1,1,0,1,1,0,0,0,0,0,0,1,0,0,1,0,0,1,0,1,1,1,0,1,1
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

Fig-14: Scanned-GCODE-Label: [16x16:Data]+[5x:LineChecksum+3x:Border] in CSV format

COPYRIGHT
www.TheCloudOSCenter.com

```
1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1
1,0,1,0,1,0,0,1,1,1,1,0,1,1,0,1,1,1,1,0,0,1,0,1,0,1
1,1,1,0,0,0,0,0,0,1,1,1,0,0,1,1,1,0,0,0,0,0,0,1,1,1
0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0
0,0,1,1,0,0,1,0,0,0,0,1,1,1,1,0,0,1,0,0,0,0,1,1,0,0
0,0,1,0,0,0,0,1,0,1,0,1,0,0,0,0,1,0,1,1,0,1,1,1,0,0
0,1,0,1,0,1,0,0,1,0,1,1,0,1,1,0,0,1,1,1,0,1,1,0,0,0
0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,1,1,0,1,1,0,0,0
0,0,1,1,1,1,1,0,0,1,1,1,0,0,0,0,0,0,1,0,0,1,0,0,0,0
0,0,1,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,1,0,0,0,1,0,0,0
0,0,1,1,1,1,1,1,0,0,0,1,0,1,0,1,1,0,1,0,0,1,1,1,0,0
0,0,1,1,1,1,1,0,0,0,0,1,0,1,1,1,1,0,0,1,0,0,0,0,1,0
0,0,1,0,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0,0,0
0,0,1,1,0,0,0,0,0,0,1,0,0,1,1,0,0,1,0,1,0,0,1,0,1,0
0,0,1,1,1,1,0,0,0,0,1,1,0,1,0,1,1,0,1,1,0,1,1,1,0,0
0,0,1,0,1,0,0,0,0,1,1,1,0,0,0,1,1,0,1,1,0,1,1,0,0,0
0,0,1,1,0,1,0,1,0,0,1,1,0,0,1,1,1,0,1,0,0,1,0,1,0,0
0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0
0,0,1,1,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0
0,0,1,1,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,1,0,0
0,0,1,1,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0
1,0,1,0,0,0,0,0,0,1,1,1,0,0,1,1,0,0,1,1,1,0,0,1,0,1
0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,1,1,1,1,0,1,0,0,0,0,0
1,1,1,0,0,1,0,0,0,1,0,1,0,0,0,0,1,0,0,1,0,0,0,1,1,1
1,0,1,0,1,1,1,1,0,1,1,1,0,0,1,1,1,0,1,1,0,1,0,1,0,1
1,1,1,0,1,1,0,1,1,0,0,0,0,0,0,1,0,0,1,0,0,1,0,1,1,1
```

Fig-15: Scanned-GCODE-Label: [16x16:Data]+[5x:LineChecksum] in CSV format

www.TheCloudOSCenter.com
Henry V. Pham

G-CODE Label:[32x32]:[50x50] Sample

The G-CODE sample 32x32 data labels below in color and black-and-white modes are coded for the text "**This is a G-CODE. Great CODE, Great CODE, Great CODE, Great CODE, and Great CODE. Greatest CODE Ever! Forever G-CODE!!!**". The G-CODE below contains the data of 32x32 with 6x lines frame for lines-checksum, plus the 3x border lines with total of 50x50 dots matrix for the entire label.

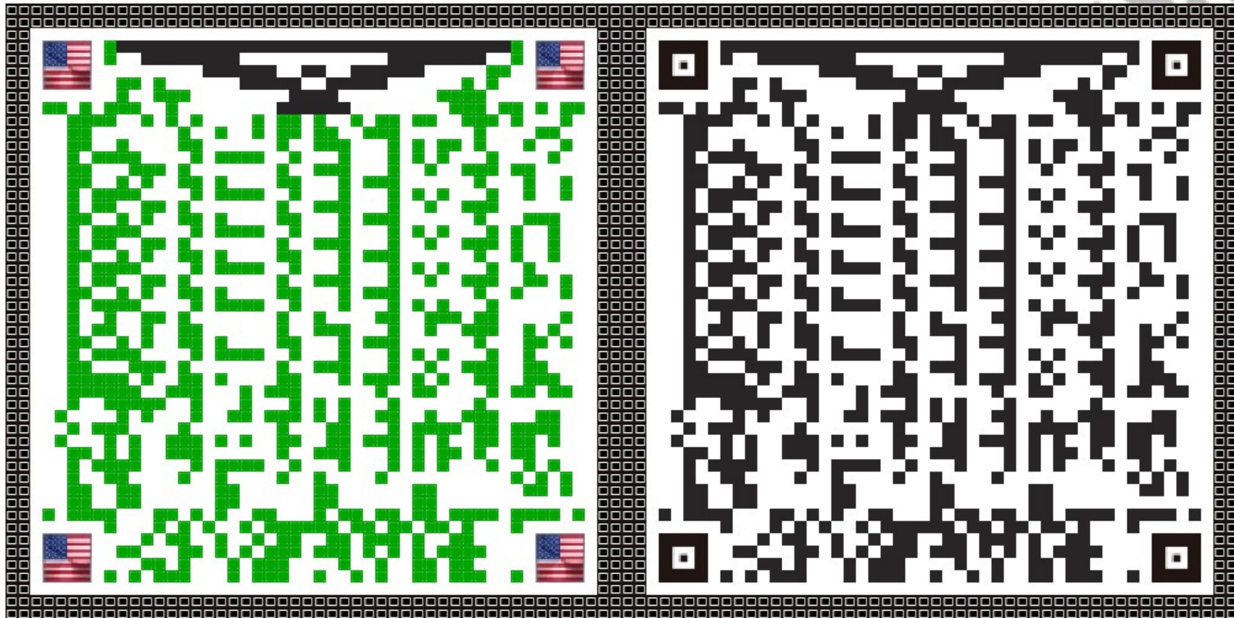


Fig-16: GCODE-Label: [32x32:Data]+[50x50 (6x:LineChecksum+3x:Border)]

G-CODE Label:[64x64]:[86x86] Sample

The G-CODE sample 64x64 data labels below in color and black-and-white modes are coded for the text "**This is a G-CODE. Great CODE, Great CODE, Great CODE, Great CODE, and Great CODE. Greatest CODE Ever! Forever G-CODE!!!This is a G-CODE. Great CODE, Great CODE, Great CODE, Great CODE, and Great CODE. Greatest CODE Ever! Forever G-CODE!!!This is a G-CODE. Great CODE, Great CODE, Great CODE, Great CODE, and Great CODE. Greatest CODE Ever! Forever G-CODE!!!This is a G-CODE. Great CODE, Great CODE, Great CODE, Great CODE, and Great CODE. Greatest CODE Ever! Forever G-CODE!!!**". The G-CODE

COPYRIGHT © HVPHAM Henry V. Pham www.TheCloudsCenter.com

below contains the data of 64x64 and the 8x lines frame for lines-checksum, plus the 3x border lines with total of 86x86 dots matrix for the entire label.

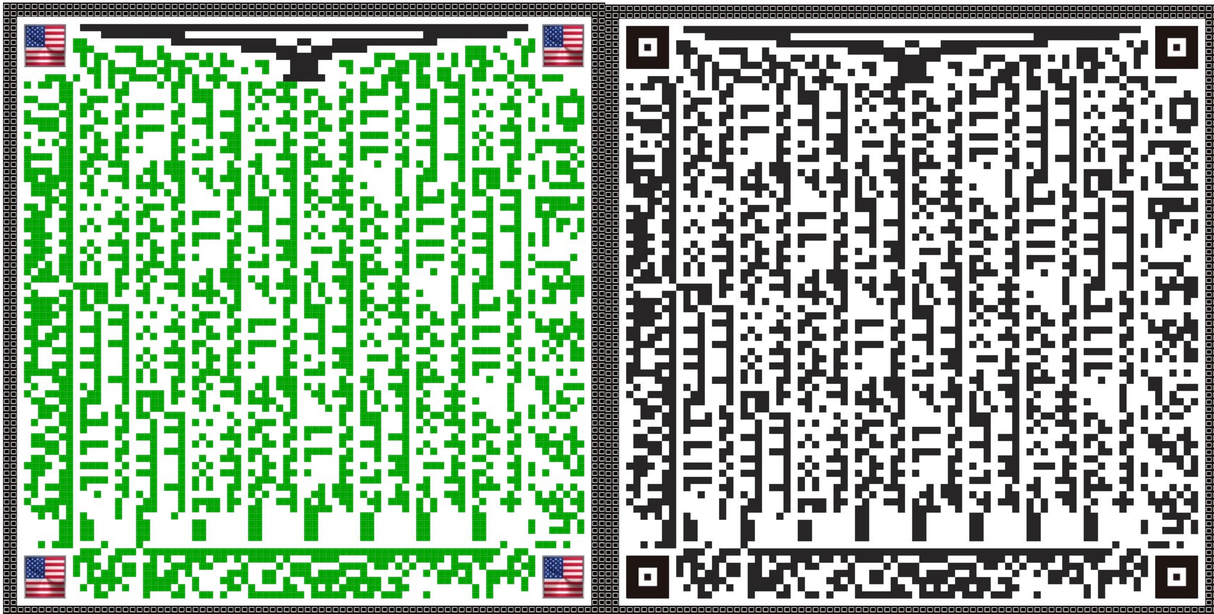


Fig-17: GCODE-Label: [64x64:Data]+[86x86(8x:LineChecksum+3x:Border)]

G-CODE Label:[1024x1024]:[1052x1052] Sample

The G-CODE sample data label in Fig-18below in color mode and coded for the text of 52 pages long from the link: http://uspto.gov/patents/basics/general-information-patents. The G-CODE below contains the data of 1024x1024 data matrix with the 11x lines frame for lines-checksum, plus the 3x border lines with total of 1052x1052 data matrix for the entire label.

COPYRIGHT © DRAFTED -HVPHAM
www.TheCloudOSCenter.com

The G-CODE

Fig-18: GCODE-Label: [1024x1024:Data]+[1052x1052 (Label-Dots)]

Conclusion

This G-CODE with State Of The Art design and dynamic grown in sizes, G-CODE is a Great CODE and designed to last ever. The G-CODE is designed to replace the existing dot-code or data-matrix labels like Barcode, Code-128, QR-CODE or any other Data-Matrix Code labels.

www.TheCloudOSCenter.com