National Tsing Hua University
Department of Electrical Engineering
EE429200 IC Design Laboratory, Fall 2021

Homework Assignment #4 **(15%)**
QR Code Decoder with Error Correction

Assigned on Nov 4, 2021
Due by **Nov 18, 2021**

Assignment Description

I. Introduction

Fig. 1 Example of QR code (17-byte "www.google.com.tw", test pattern no. 02)

QR code (Quick Response Code) [1] is a type of matrix barcode used for storing text information in a robust way. It is widely used for URL instant access. Its specification varies a lot, including version 1-40 (from 21x21 to 177x177 pixel), four different character sets, and different levels for error correcting capability. In this assignment, you will implement a version-2 (25x25 pixels, total 44 codewords), 8-bit Byte mode (JIS8 code), and error correction level M (15% recovery capacity, which means 8 codewords can be recovered at most) QR code decoder.

II. Goal

The structure of QR code is shown in Fig. 2. The version 2-M QR code is a 25x25 binary matrix, in which the black pixel stands for "1", and white for "0". The codewords of a QR code are "masked" (XORed) with one pre-defined pattern for balancing the number of black and white pixels. As a result, you need to find the mask pattern ID for de-masking the codewords. After de-masking, the codewords can be decoded sequentially. There are total 44 codewords, but some of them might be decoded wrong (due to noise, defect in QR code, etc.). Therefore, the next step is to apply the error correction algorithm to get the correct codewords. Finally, you can decode the text from your corrected codewords. All the decoding steps will be explained in detail in the next section.

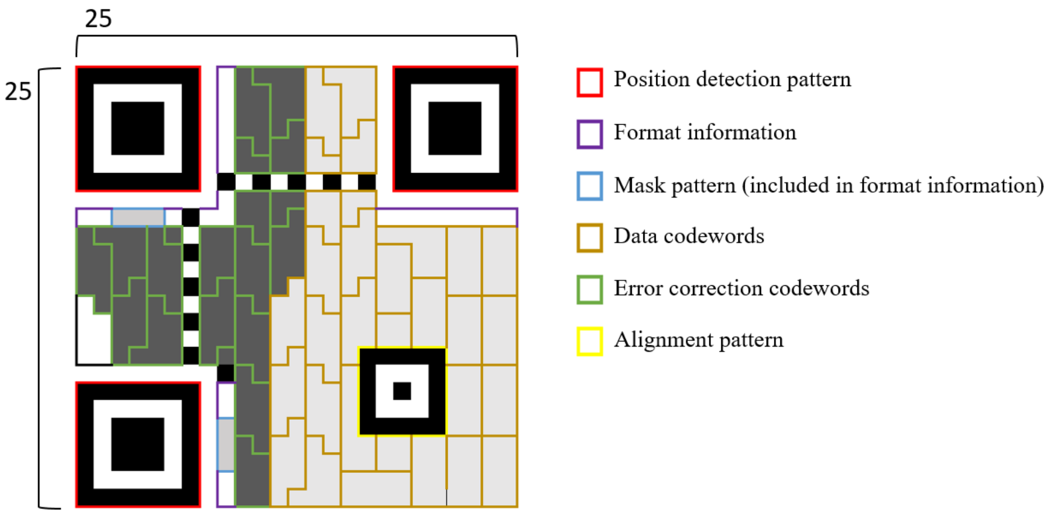

25
25
Position detection pattern
Format information
Mask pattern (included in format information)
Data codewords
Error correction codewords
Alignment pattern

Fig. 2 The structure of a version 2-M QR code.

In this assignment, 4 different types of QR code pattern will be used to test your function, please refer to Fig. 3 for what the test patterns look like:

- **Rank A**:
  Arbitrary rotation (0°, 90°, 180°, 270°) and position (placed in a 64x64 empty image), with error text in the QR code.
- **Rank B**:
  Fixed rotation and position(at the upper-left corner), with error text in the QR code.
- **Rank B1**:
  Arbitrary rotation (0°, 90°, 180°, 270°) and position (placed in a 64x64 empty image), without error text in the QR code.
- **Rank C**:
  Fixed rotation and position(at the upper-left corner), no error text in the QR code.

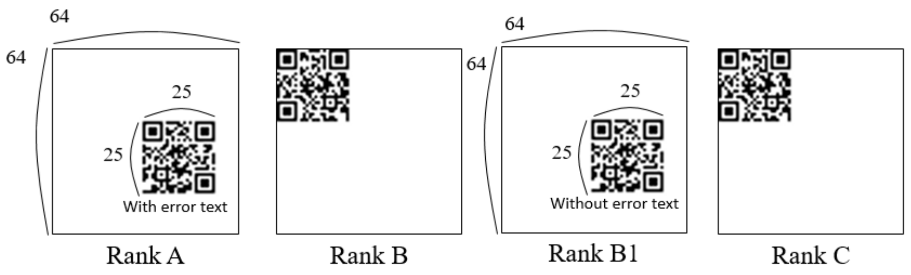


Fig. 3 Test patterns for Rank A, B, B1, C

The following steps might be useful for designing your finite state machine:

1. Determine the position and rotation of QR code from image.
2. Find the mask pattern ID, then do the de-masking.
3. Decode all the 26 codewords.
4. Apply error correction and correct the codewords.
5. Decode all the text from corrected codewords.

III. Detailed step of decoding

A. Data arrangement in SRAM

The 25x25 QR code patterns is putting into a 64x64 images, and the whole 64x64 images are stored in SRAM. The data arrangement is illustrated in Fig. 4. The SRAM has a capacity of 4096 words and 1-bit word size. Each address contains 1 pixel, arranged in the order shown in Fig. 4.

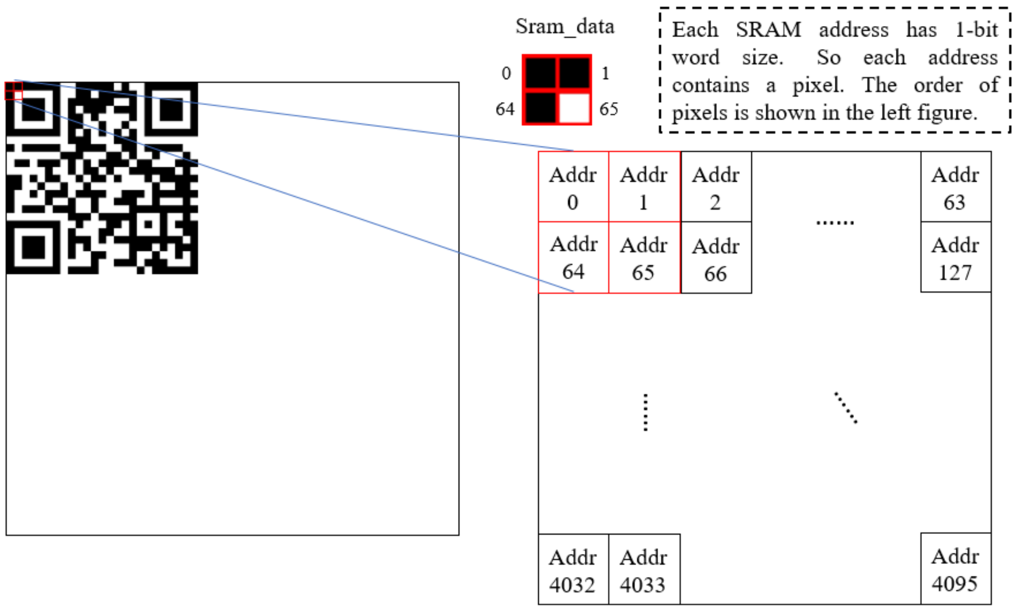

Sram_data
0
1
64
65
Each SRAM address has 1-bit word size. So each address contains a pixel. The order of pixels is shown in the left figure.
Addr 0
Addr 1
Addr 2
......
Addr 63
Addr 64
Addr 65
Addr 66
Addr 127
Addr 4032
Addr 4033
Addr 4095

Fig. 4 Data arrangement in SRAM.

B. Determine the position and rotation of the QR code

Since the QR code patterns are arbitrary rotation and position in Rank A, you need to locate the QR code before decoding it. Use the position detection pattern to locate the three corners, then determine the rotation and position by them.

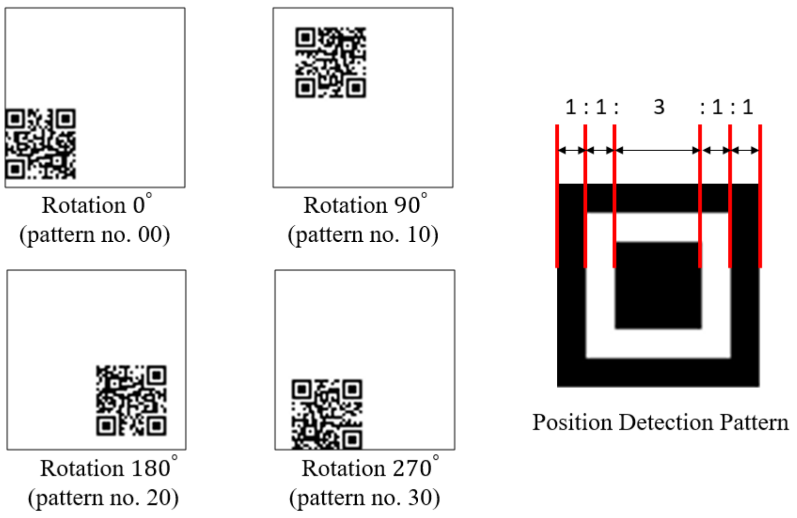


Fig. 5 Four different rotation pattern and position detection pattern.

C. De-masking process

All the codewords have been masked (XORed) with a specific mask pattern, and the mask pattern is decided by the mask pattern ID. The ID is a 3-bit data, which locates in position = (8, 2:4) in rotation 0° QR code. The 3-bit data should first be XORed with a fixed 3’b101 to derive the real mask ID. There are 8 different mask patterns as listed in Fig. 6, and the pixel of mask is equal to 1 if it satisfies the following condition. You can refer to Fig. 7 for what each mask should look like, and Fig. 8 for the whole de-masking process.

Mask Pattern Reference	Condition
000	$(i + j) \bmod 2 = 0$
001	$i \bmod 2 = 0$
010	$j \bmod 3 = 0$
011	$(i + j) \bmod 3 = 0$
100	$((i \text{ div } 2) + (j \text{ div } 3)) \bmod 2 = 0$
101	$(i j) \bmod 2 + (i j) \bmod 3 = 0$
110	$((i j) \bmod 2 + (i j) \bmod 3) \bmod 2 = 0$
111	$((i j) \bmod 3 + (i+j) \bmod 2) \bmod 2 = 0$

Fig. 6 Mask generation condition [1]

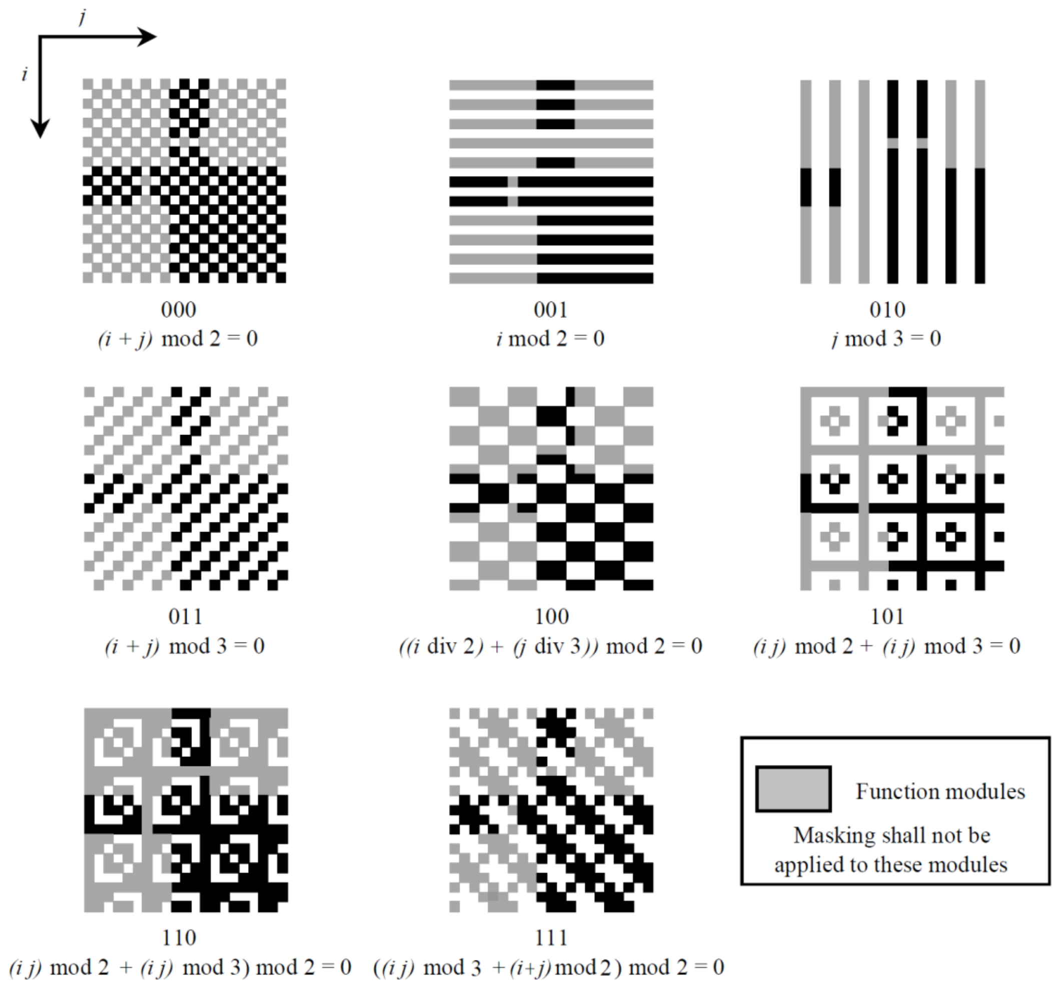

j
i
000
(i + j) mod 2 = 0
001
i mod 2 = 0
010
j mod 3 = 0
011
(i + j) mod 3 = 0
100
((i div 2) + (j div 3)) mod 2 = 0
101
(i j) mod 2 + (i j) mod 3 = 0
110
((i j) mod 2 + (i j) mod 3) mod 2 = 0
111
((i j) mod 3 + (i+j) mod 2) mod 2 = 0
Function modules
Masking shall not be applied to these modules

Fig. 7 Eight different mask patterns [1]

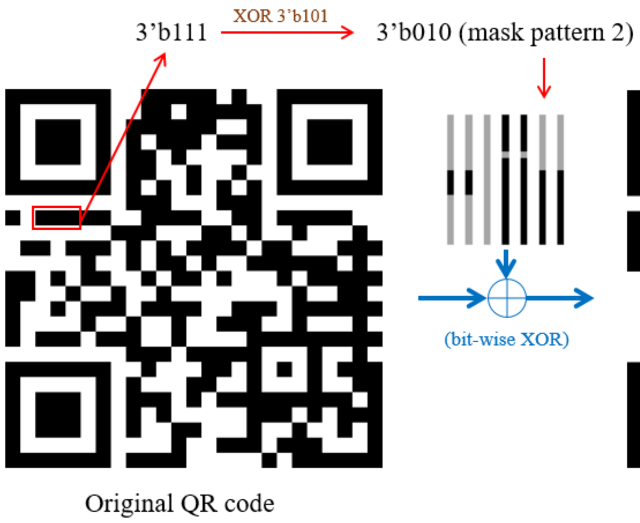


Fig. 8 De-masking process.

D. Decode the codewords

Now, you can read the codewords from the de-masked QR code. The arrangement of codewords is shown in Fig. 9. Please decode the codewords by the order of number labeled in each block. The bit order is also shown in Fig. 9, and you should follow the directions of arrow in each block to determine which order to be used. Note that the pixels in the orange boxes are timing patterns, so skip this area when you decode the codeword blocks.

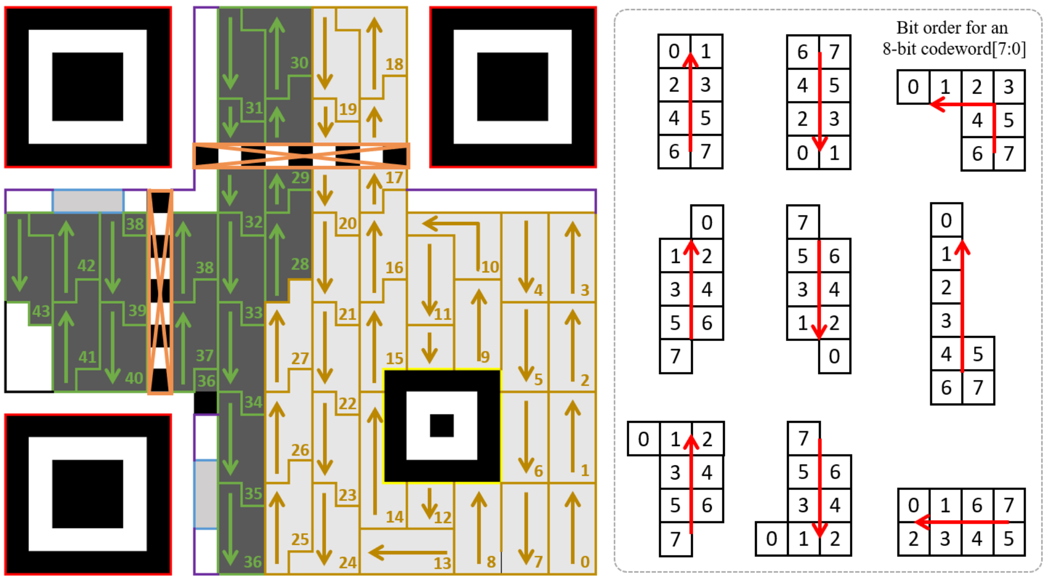


Fig. 9 The codeword arrangement and bit order of an 8-bit codeword.

E. Error correction procedure

(1) Understanding the basic of Galois Field [2][3]

The error correction of QR code is based on "**Reed-Solomon error correction**" which operates on a set of finite field (or **Galois Field**) elements. So first, you need to understand the basic concept and operation of Galois Field.

The Galois Field is a field that contains a finite number of elements. By definition, Galois field is a set on which addition, subtraction, multiplication, and division are defined, and the arithmetic result should also be in the same field. Because the codewords is 8-bit, we are using GF(2^8) to do error correction, which means that all the elements are lying in the interval of [0, 255].

The arithmetic of Galois Field is actually the polynomial arithmetic. The binary representation of a number also represents a polynomial. For example, the number 13 (8'b00001101) stands for the polynomial $\alpha^3 + \alpha^2 + 1$, 250 (8'b11111010) for $\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^1$. When we add the two numbers, we are adding the two polynomials, then apply modulo-2 to each coefficient of the polynomial. Because $(0 + 0)\%2 = 0$, $(0 + 1)\%2 = 1$, $(1 + 0)\%2 = 1$, $(1 + 1)\%2 = 0$, the addition in GF(2^8) is logically equivalent to **XOR** operation. Please see fig 10. for the detailed example. (Note: Subtraction in GF(2^8) is equal to addition. Ex: $(1 + 1)\%2 = (1 - 1)\%2 = 0$, $(0 + 1)\%2 = (0 - 1)\%2 = 1$)

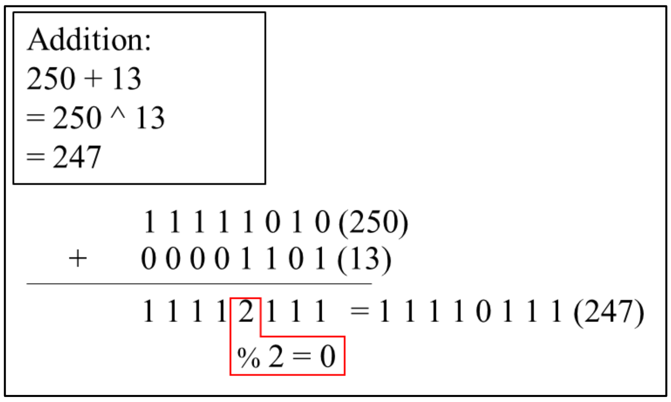


Fig. 10 Addition in GF(2^8)

For multiplication, the first step is to do the polynomial multiplication. Don't forget to apply modulo-2 to each coefficient of the product polynomial. If the power of product polynomial exceeds 8 (i.e. the number is larger than 255), you should apply modulo $P(\alpha)$, where $P(\alpha) = \alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1$(representing the decimal number 285) is the primitive polynomial of GF(2^8). Please see Fig 11. for a complete example. Since this process is quite complex, another method is to use lookup-table (see Appendix A). We use the notation $\alpha^n (0 \leq n \leq 255)$ to represent a number in GF(2^8). The multiplication can be simply calculated as the summation of power of the two number in α notation. Ex: The same example in Fig. 11, $133 = \alpha^{128}$, then $133 \times 133 = \alpha^{128} \times \alpha^{128} = \alpha^{128+128} = \alpha^{256}$. Since the power of α should be in the interval of [0,255], we will apply modulo 255 to the power that exceeds 255. Thus, $\alpha^{256} = \alpha^{256\%255} = \alpha^1 = 2$, which is the same result as in Fig. 11.

Multiplication:
133 x 133
= 2

(1)
$$133 = 8'b10000101 = \alpha^7 + \alpha^2 + 1$$

(2)
$$133 \times 133 = (\alpha^7 + \alpha^2 + 1)(\alpha^7 + \alpha^2 + 1)$$
$$= \alpha^{14} + 2\alpha^9 + 2\alpha^7 + \alpha^4 + 2\alpha^2 + 1 \longleftarrow$$ Apply modulo-2 to each coefficients
$$= \alpha^{14} + \alpha^4 + 1$$

(3)
Apply long division by dividing P(α):

$$\begin{array}{r|l} & \alpha^6 - \alpha^2 - \alpha - 1 \\ \hline P(\alpha) & \alpha^{14} + \alpha^4 + 1 \\ & \vdots \\ \hline & 2\alpha^5 + 4\alpha^4 + 2\alpha^3 + 2\alpha^2 + \alpha + 2 \longleftarrow \\ & = \alpha \\ & = 2 \end{array}$$

Apply modulo-2 to each coefficients

Fig. 11 Multiplication in GF(2^8)

Please make sure that you can fully understand the basic operation before going to the next section.

(2) <u>Error correction steps</u> [1]:

In this part, the detailed error correction steps are shown, and an example is provided in Appendix B for you to understand and debug. Please note that all the arithmetic operation should follow the GF(256) rules.

i. Preparing the codeword sequence $(r_0, r_1, \ldots, r_{43})$ after de-masking, and constructing the polynomial $R(x)$:

$$R(x) = r_0x^{43} + r_1x^{42} + \cdots + r_{42}x^1 + r_{43}$$

, where r_0 stands for the value of codeword 0, r_1 for codeword 1, …, r_{43} for codeword 43.

ii. Calculate the syndrome:

$$S_0 = R(1) = r_0 + r_1 + \cdots + r_{43}$$
$$S_1 = R(\alpha) = r_0\alpha^{43} + r_1\alpha^{42} + \cdots + r_{43}$$
$$S_2 = R(\alpha^2) = r_0\alpha^{86} + r_1\alpha^{84} + \cdots + r_{43}$$
$$S_3 = R(\alpha^3) = r_0\alpha^{129} + r_1\alpha^{126} + \cdots + r_{43}$$
$$S_4 = R(\alpha^4) = r_0\alpha^{172} + r_1\alpha^{168} + \cdots + r_{43}$$
$$S_5 = R(\alpha^5) = r_0\alpha^{215} + r_1\alpha^{210} + \cdots + r_{43}$$
$$S_6 = R(\alpha^6) = r_0\alpha^{3} + r_1\alpha^{252} + \cdots + r_{43}$$
$$S_7 = R(\alpha^7) = r_0\alpha^{46} + r_1\alpha^{39} + \cdots + r_{43}$$

If there is no error, all the syndrome values will be 0.

iii. (Find the error position)

First solve $\sigma_4,\ \sigma_3,\ \sigma_2,\ \sigma_1$ for

$$S_0\sigma_4 - S_1\sigma_3 + S_2\sigma_2 - S_3\sigma_1 + S_4 = 0$$
$$S_1\sigma_4 - S_2\sigma_3 + S_3\sigma_2 - S_4\sigma_1 + S_5 = 0$$
$$S_2\sigma_4 - S_3\sigma_3 + S_4\sigma_2 - S_5\sigma_1 + S_6 = 0$$
$$S_3\sigma_4 - S_4\sigma_3 + S_5\sigma_2 - S_6\sigma_1 + S_7 = 0$$

Then find $\alpha^i(\mathrm{i} = 0\sim 43)$ which makes $\sigma(\alpha^i) = 0$

$$,\text{where } \sigma(x) = \sigma_4 + \sigma_3 x + \sigma_2 x^2 + \sigma_1 x^3 + x^4$$

The solution i is the position of error. Please note that this position means the error occurs at which power term of x. Ex: If the solution $i = 42$, the error occurs at the x^{42} term, which means the error codeword is actually r_1 (codeword 1).

In this assignment, the number of error ≤ 4. So if the number $= 4$, then there will be four solution of i. If the number $= 3$, there will be only three solution, and so on.

iv. (Find error offset)

Solve $Y_1,\ Y_2,\ Y_3,\ Y_4$ for

$$Y_1(\alpha^{i_1})^1 + Y_2(\alpha^{i_2})^1 + Y_3(\alpha^{i_3})^1 + Y_4(\alpha^{i_4})^1 = S_0$$
$$Y_1(\alpha^{i_1})^2 + Y_2(\alpha^{i_2})^2 + Y_3(\alpha^{i_3})^2 + Y_4(\alpha^{i_4})^2 = S_1$$
$$Y_1(\alpha^{i_1})^3 + Y_2(\alpha^{i_2})^3 + Y_3(\alpha^{i_3})^3 + Y_4(\alpha^{i_4})^3 = S_2$$
$$Y_1(\alpha^{i_1})^4 + Y_2(\alpha^{i_2})^4 + Y_3(\alpha^{i_3})^4 + Y_4(\alpha^{i_4})^4 = S_3$$

Then the error offset is $Y_1\alpha^{i_1},\ Y_2\alpha^{i_2},\ Y_3\alpha^{i_3}$ and $Y_4\alpha^{i_4}$.

v. Correct the error with the position and offset value by adding (XOR) them together.

You can refer to the example of Appendix B to further understand the above steps.

F. Decode the text sequentially

After correcting the codewords, you can start to decode the text. The data in the codewords is arranged as shown in Fig. 12. The first four bits in codeword 0 is data encoding type. We only use 8-bit Byte mode(4'b0100) in this assignment, you can refer to JIS8 code in Appendix C to see the value of each character. The last four bits of codeword 0 and first four bits of codeword 1 indicates the text length encoded in this QR code. You should first determine the text length to decide how many characters to decode. To the rest of the codewords, the last 4 bits of the current codeword and the first four bits of the next codeword will form a text data. Once decoding a text data, you can put it to the output port "*decode_jis8_code [7:0]*". Don't forget to set "*decode_valid*" to 1 when the output text is valid. The testbench will check the output text when it detects "*decode_valid*" equals to 1 (testbench will only check one character when detecting one cycle valid). After all the text are decoded, set "*qr_decode_finish*" to 1 to finish the simulation.

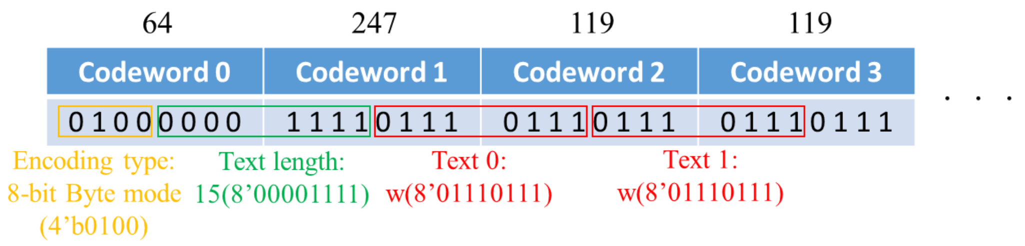


Fig. 12 The data arrangement in codewords.

I/O name definition

I/O name	Type	Number of bits	Description
clk	I	1	Clock input
srstn	I	1	Synchronous reset (active low)
qr_decode_start	I	1	Start decoding for one QR code. 1: start (one-cycle pulse)
sram_rdata	I	1	Read data from SRAM
sram_raddr	O	12	Read address to SRAM
decode_valid	O	1	Decoded code is valid
decode_jis8_code	O	8	Decoded JIS8 code
qr_decode_finish	O	1	1: decoding one QR code is finished

Grading policy

There are three grading policies provided. You can choose any of the three policy for your final score of this homework.

Policy 1

1. RTL coding and pass pre-simulation (Rank A:6%; Rank B:6%; Rank C:3%).
2. Use Spyglass to show the synthesizability (Rank A:2%; Rank B:2%; Rank C:2%).
3. Logic synthesis with the provided scripts. Only two script files can be modified: **0_readfile.tcl** for adding your self-defined RTL files, and **synthesis.tcl** for changing the clock timing constraint TEST_CYCLE (e.g. 2.3 for 2.3 ns). Your timing slack should not be negative. The grading will be based on your performance index:

$$\text{Rank A: } \begin{cases} \text{ClassC: 3\%} & if\ \boldsymbol{PI} > 1.4 \times 10^{10} \\ \text{ClassB: 5\%} & if\ 1.4 \times 10^{10} \geq \boldsymbol{PI} > 8 \times 10^{9} \\ \text{ClassA: 7\%} & if\ \boldsymbol{PI} \leq 8 \times 10^{9} \end{cases}$$

Policy 2

1. RTL coding and pass pre-simulation (Rank B:6%; Rank C:3%).
2. Use Spyglass to show the synthesizability (Rank B:2%; Rank C:2%).
3. Logic synthesis with the provided scripts. Only two script files can be modified: **0_readfile.tcl** for adding your self-defined RTL files, and **synthesis.tcl** for changing the clock timing constraint TEST_CYCLE (e.g. 2.3 for 2.3 ns). Your timing slack should not be negative. The grading will be based on your performance index:

$$\text{Rank B: } \begin{cases} \text{ClassB: 2\%} & if\ \boldsymbol{PI} > 2.5 \times 10^{9} \\ \text{ClassA: 3\%} & if\ \boldsymbol{PI} \leq 2.5 \times 10^{9} \end{cases}$$

Policy 3

1. RTL coding and pass pre-simulation (Rank B1:5%; Rank C:3%).
2. Use Spyglass to show the synthesizability (Rank B1:2%; Rank C:2%).
3. Logic synthesis with the provided scripts. Only two script files can be modified: **0_readfile.tcl** for adding your self-defined RTL files, and **synthesis.tcl** for changing the clock timing constraint TEST_CYCLE (e.g. 2.3 for 2.3 ns). Your timing slack should not be negative. The grading will be based on your performance index:

$$\text{Rank B1: } \begin{cases} \text{ClassB: 2\%} & if\ \boldsymbol{PI} > 1.2 \times 10^{9} \\ \text{ClassA: 3\%} & if\ \boldsymbol{PI} \leq 1.2 \times 10^{9} \end{cases}$$

$$\boxed{\mathbf{PI = A \times T \times C}}$$

A: Total area (shown in report/report_area_qr_decode.out)
T: TEST_CYCLE (your clock period to pass pre-sim, defined in test_qr_decode.v)
C: The cycle count you need to complete the simulation of test_qr_decode.v, which is shown on the monitor when the simulation is finished.
For example:

```
Congratulations! Simulation from pattern no 00 to 39 is successfully passed! Total
cycle count =          xxxxx!
```

4. **Bonus**. For Policy1 Rank A works, 3% will be given to the best work in terms of the performance index PI, and 1% given to the second best. (Only for those who submit the homework before 2021.11.18 23:59.)

Deliverable

1. Synthesizable Verilog functional module qr_decode.v and all other RTL codes for your own defined modules (if any).
2. Create a file named hdl.f, which contains all the RTL files (*.v) you use in this assignment
3. Spyglass report file
4. The two modified synthesis script files: 0_readfile.tcl and synthesis.tcl. The report files for timing and area: report_area_qr_decode.out and report_time_qr_decode.out. The synthesis log file da.log. The netlist file: qr_decode_syn.v and qr_decode_syn.sdf.
5. A text file misc.txt summarizing the grading policy you choose, your rank and class, performance index and your A, T, and C as well.

Submit File Structure (Note: 1% punishment for wrong file delivery.)

- HW4/
  - hdl/
    - ✓ hdl.f
    - ✓ qr_decode.v
    - ✓ (other Verilog files)

  - syn/
    - ✓ 0_readfile.tcl
    - ✓ synthesis.tcl
    - ✓ report_area_qr_decode.out

  - ✓ report_time_ qr_decode.out
  - ✓ da.log
  - ✓ qr_decode_syn.v
  - ✓ qr_decode_syn.sdf
- ■ Spyglass/
  - ✓ Your report file
- ■ misc.txt

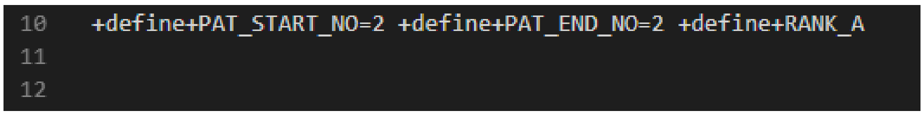
Remember to compress files as HW4_10XXXXXXX.zip ! Don't use .rar or other format. Or you will lose 1%.

File Organization

Directory	Filename	Description
hdl/	qr_decode.v	Main function
sim/	test_qr_decode.v	Testbench for QR code decoder
	sim.f	File list for simulation
	sram_model/ sram_4096x1b.v	SRAM behavior model
	bmp/*	QR code image for test pattern
	golden/*.dat	Test pattern information
syn/	*.tcl	.tcl file for DC scripts
	run_dc	Shell script for running synthesize

Note

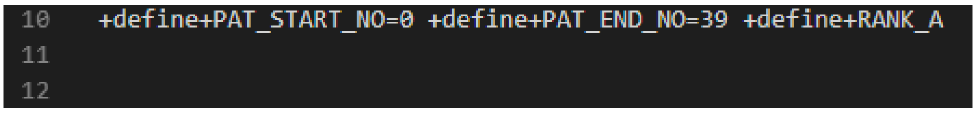
1. For debugging, you can run simulation for selected test patterns.
   For example, you can test the pattern no. 2, Rank A by modifying the definition in sim.f:



```
+define+PAT_START_NO=2 +define+PAT_END_NO=2 +define+RANK_A


```

or all the patterns of Rank A by



```
+define+PAT_START_NO=0 +define+PAT_END_NO=39 +define+RANK_A


```

2. You can uncomment line 104 in test_qr_decode.v to display the content of SRAM in your terminal.

Reference

[1] ISO/IEC 18004:2000(E), *Information technology — Automatic identification and data capture techniques — Bar code symbology — QR Code.*

[2] Error Correction Coding, https://www.thonky.com/qr-code-tutorial/error-correction-coding

[3] Wikipedia – Finite Field, https://en.wikipedia.org/wiki/Finite_field