National Tsing Hua University
Department of Electrical Engineering
EE429200 IC Design Laboratory, Fall 2022

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

Assigned on Nov 3, 2022
Due by **Nov 17, 2022**

## Assignment Description

### I. Introduction

**Fig. 1** Example of QR Code (14-byte "www.google.com")

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 to 40 (from 21x21 to 177x177 pixel), four different character sets, and different levels of error correction capability. In this assignment, you will implement a version-1 (21x21 pixels, total 26 codewords), 8-bit Byte mode (ISO 8859-1 code), and error correction level L (7% recovery capacity) QR Code decoder.

### II. Goal

The structure of QR Code is shown in Fig. 2. The version 1-L QR Code is a 21x21 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 26 codewords. 19 of them are data codewords, and the other 7 of them are error correction codewords. Since the QR Codes in this assignment are all error-free, error correction is not required to perform. You can directly decode the text from the data codewords. All the decoding steps will be explained in detail in the next section.

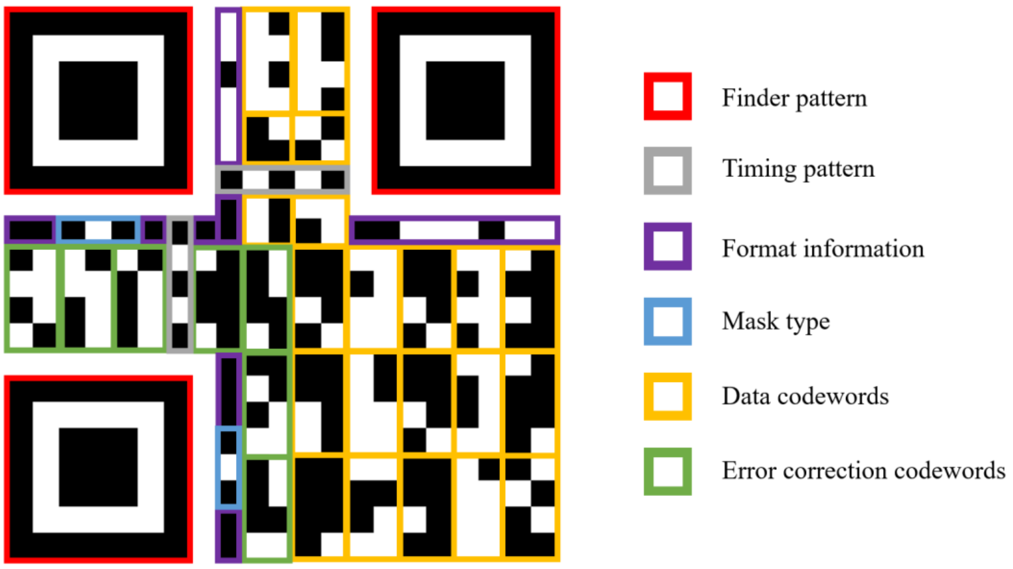


**Fig. 2** Structure of version 1-L QR Code.

In this assignment, four different types of test pattern, Rank A, Rank B, Rank C, and Rank D, will be used to test your design. Rank A stands for the most complex pattern, and Rank D stands for the simplest pattern. Table 1 describes the format of test pattern for each Rank, and Fig. 3 provides some examples of test pattern for each Rank.

| Pattern | # of QR Code in a pattern | Location | Rotation |
|---|---|---|---|
| Rank A | Arbitrary (1 to 4) | Arbitrary | Arbitrary |
| Rank B | 2 | Arbitrary | Arbitrary |
| Rank C | 1 | Arbitrary | Arbitrary |
| Rank D | 1 | Upper-Left | 0-degree |

**Table. 1** Format of test pattern for each Rank

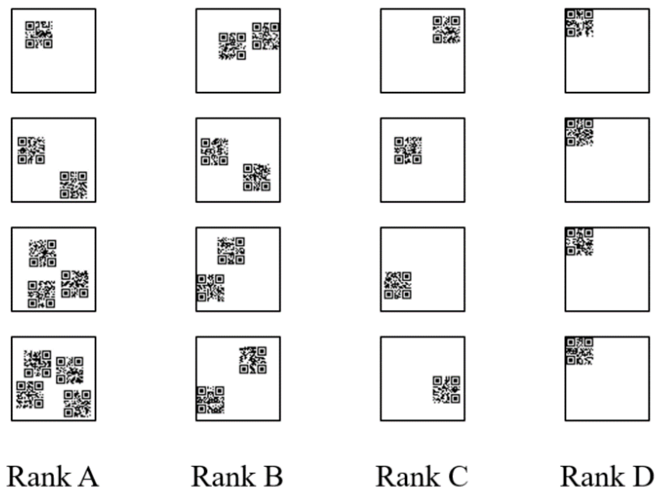


**Fig. 3** Examples of test pattern for each Rank

## III. Detailed description of decoding a QR Code

### 1. Data arrangement in SRAM

The 21x21 QR Code is put into a 64x64 image, and the whole 64x64 image is stored in SRAM. The SRAM has a capacity of 4096 words with 1-bit word size. Each address contains one pixel. "0" denotes that the pixel is white and "1" denotes that the pixel is black. The data arrangement is illustrated in Fig. 4.

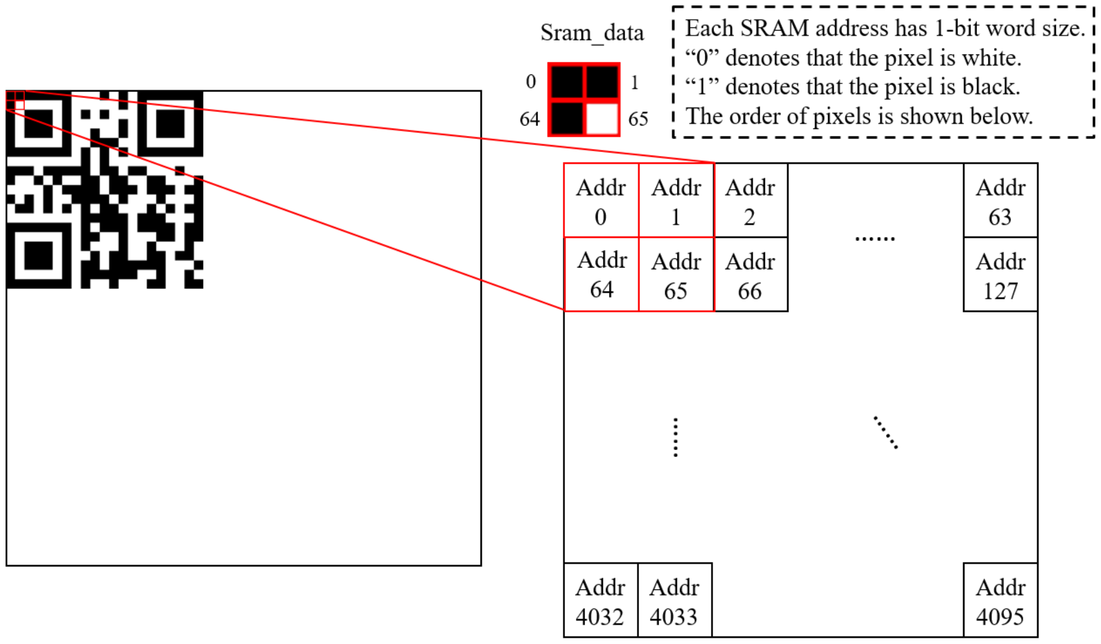

Sram_data
0
1
64
65
Each SRAM address has 1-bit word size.
"0" denotes that the pixel is white.
"1" denotes that the pixel is black.
The order of pixels is shown below.
Addr 0
Addr 1
Addr 2
Addr 63
Addr 64
Addr 65
Addr 66
Addr 127
Addr 4032
Addr 4033
Addr 4095

**Fig. 4** Data arrangement in SRAM.

### 2. Rotation of the QR Code

For Rank A, Rank B and Rank C, the QR Codes are at arbitrary location with arbitrary rotation. You need to locate the QR Codes and determine their rotation before decoding it. The rotation of the QR Codes depend on the location of the three finder patterns. Fig. 5 illustrates the definition of all four kinds of rotation.

### 3. Location of the QR Code

For Rank A and Rank B, the number of QR Code in a test pattern might be more than one. Therefore, we use two output ports: *[5:0] loc_y* and *[5:0] loc_x* to specify which QR Code in the test pattern does the output *[7:0] decode_text* belongs to. The definition of the QR Code location is illustrated in Fig. 5.

**Note:**

Although there is only one QR Code in each test pattern for Rank C and Rank D, you should still specify and output the location of the QR Code through *loc_y* and *loc_x*.

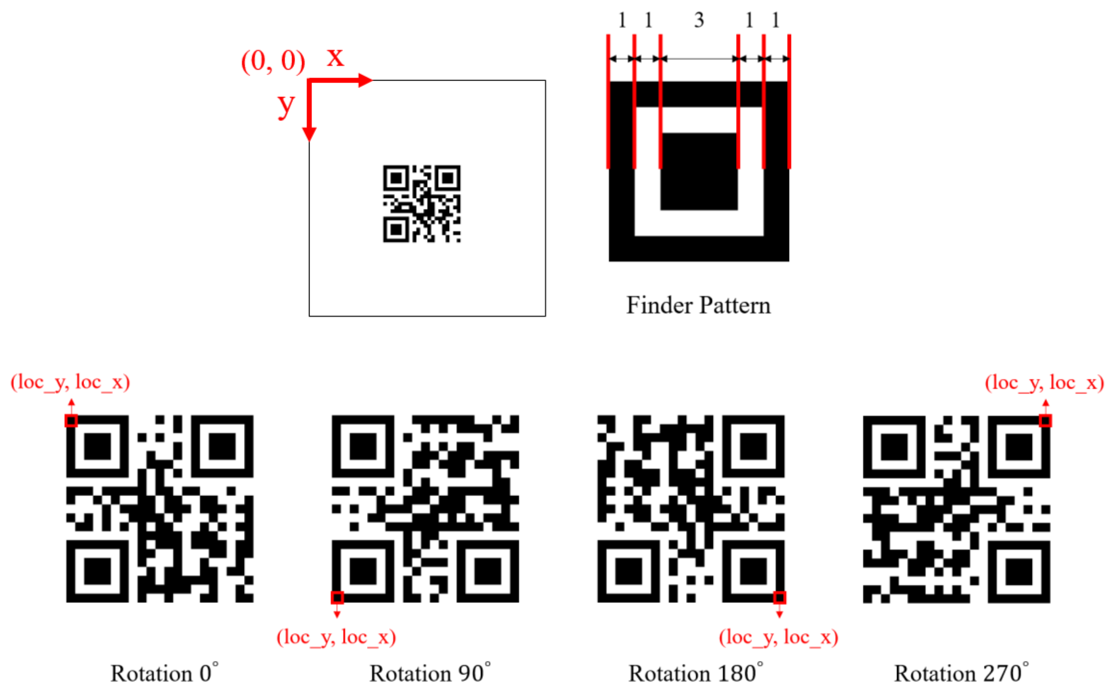


**Fig. 5** Definition of location and rotation of the QR Code

## 4. De-masking process

All the codewords in the QR Code are 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), (8, 3), (8, 4)} in the QR Code with rotation $0^{\circ}$. The 3-bit data should first be XORed with a fixed 3'b101 to derive the real mask ID. There are total 8 different mask patterns. The pixel of the mask pattern is equal to "1" if it satisfies the corresponding condition in Fig. 6. You can refer to Fig. 7 for what each mask pattern should look like. An example of the de-masking process is provided in Fig. 8.

| 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]

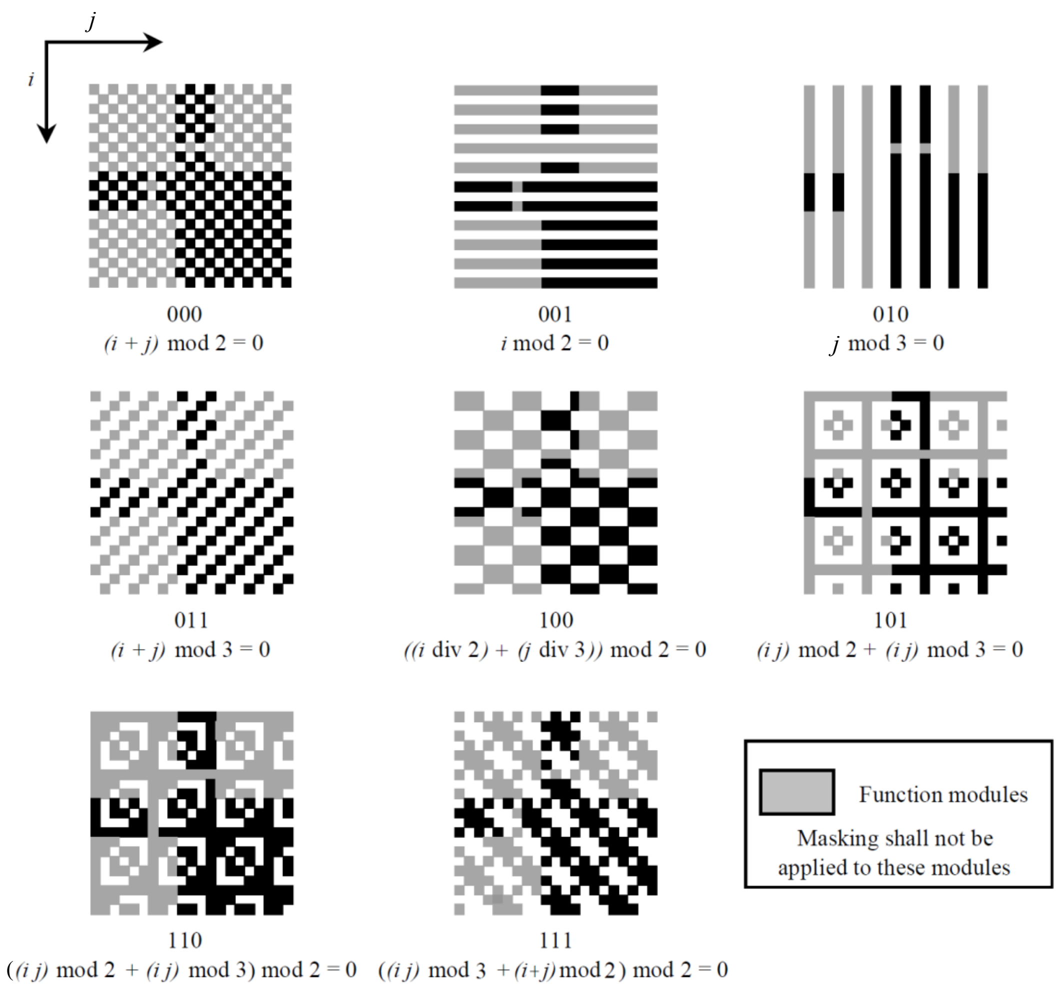

j
i
000
(i + j) mod 2 = 0
001
i mod 2 = 0
010
j mod 3 = 0
011
(i + j) mod 3 = 0
100
((i div 2) + (j div 3)) mod 2 = 0
101
(i j) mod 2 + (i j) mod 3 = 0
110
((i j) mod 2 + (i j) mod 3) mod 2 = 0
111
((i j) mod 3 + (i+j) mod 2 ) mod 2 = 0
Function modules
Masking shall not be applied to these modules

**Fig. 7** Eight kinds of mask pattern [1]

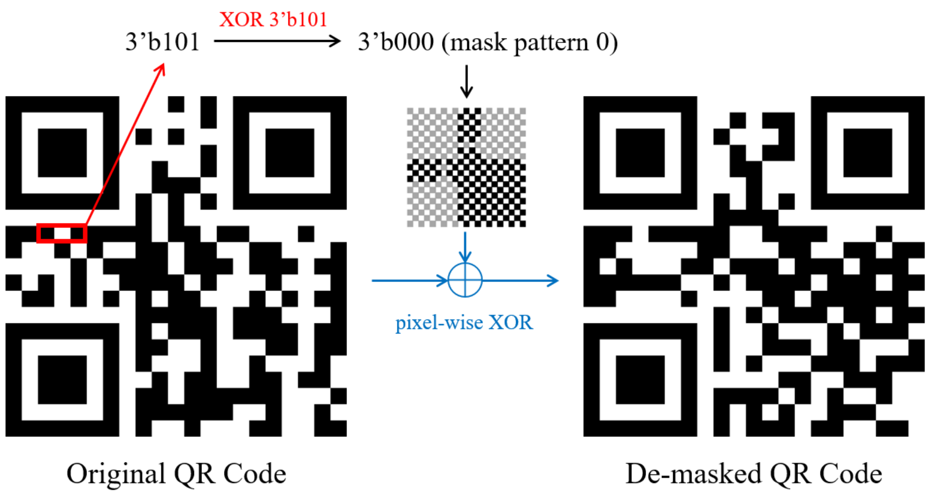


**Fig. 8** Example of de-masking process

## 5. Decode the data codewords

After demasking, you can read data codewords from the de-masked QR Code. The arrangement of each data codeword and its bit order is shown in Fig. 9. Decode the codewords by the order of number labelled in each block.

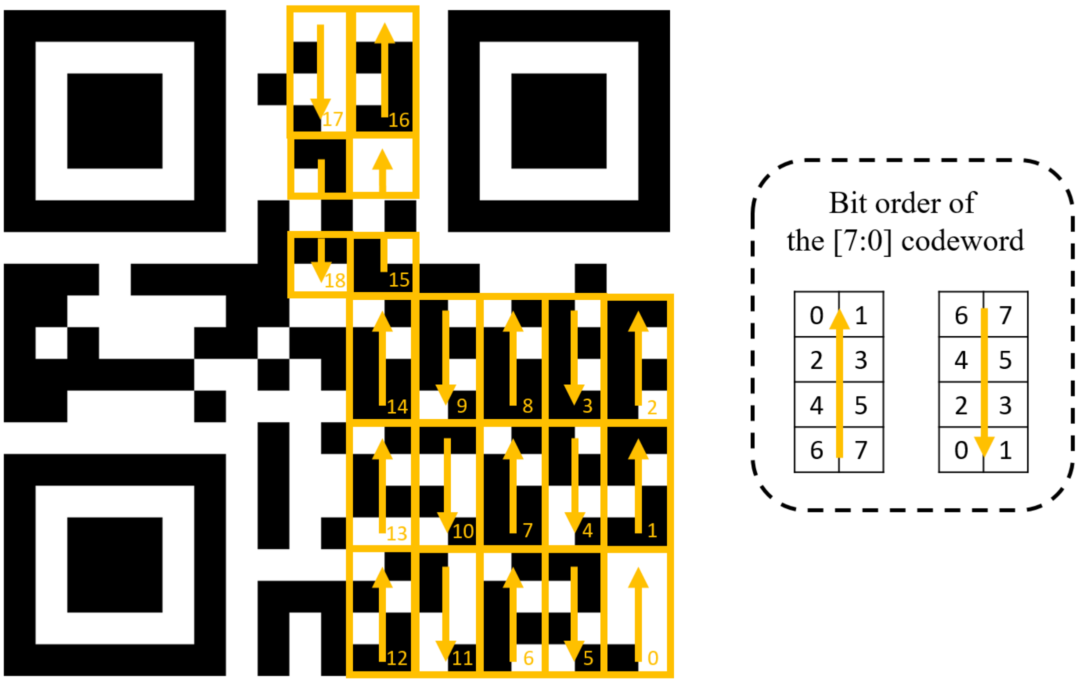


**Fig. 9** Arrangement of each data codeword and its bit order

## 6. Decode the text sequentially

The data in data codewords is arranged as shown in Fig. 10. 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 ISO 8859-1 code to see the value of each character. The last four bits of codeword 0 and the 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 should be decoded. For the rest of the data 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 *[7:0] decode_text* and set the output port *valid* to high. The testbench will check the output text when the output valid is high.

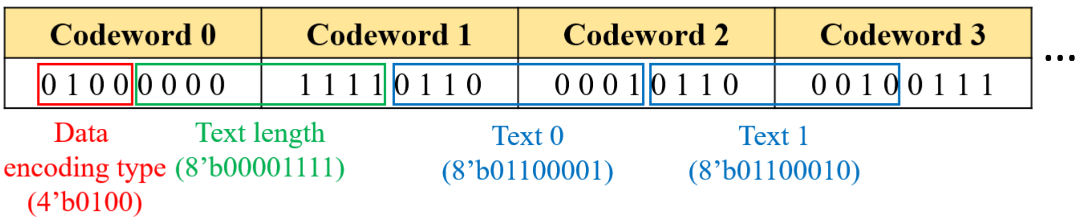


**Fig. 10** Data arrangement in data codewords

## I/O Definition

| Type | Name | Number of bits | Description |
|---|---|---|---|
| Input | clk | 1 | Clock |
| Input | srst_n | 1 | System reset (synchronous, active low) |
| Input | start | 1 | Indicate that you can start decoding (one-cycle pulse) |
| Input | sram_rdata | 1 | SRAM read data |
| Output | sram_raddr | 12 | SRAM read address |
| Output | loc_y | 6 | Row location of the QR Code |
| Output | loc_x | 6 | Column location of the QR Code |
| Output | decode_text | 8 | Decoded text |
| Output | valid | 1 | Indicate that the decode_text is valid when it is high |
| Output | finish | 1 | Set high after all QR Codes are decoded |

## Timing Diagram

Take a test pattern which contains two QR Codes as an example. One QR Code is located at (0, 0), and its decoded text is “abc” which the number is {8’h61, 8’h62, 8’h63} respectively in ISO 8859-1 code. The other QR Code is located at (30, 40), and its decoded text is “NTHU” which the number is {8’h4e, 8’h54, 8’h48, 8’h55} respectively in ISO 8859-1 code.

Fig. 11 shows the timing diagram for this example. Once the one-pulse *start* signal is detected, the circuit can start to operate. When one QR Code in the test pattern is decoded, the decoded text of the QR Code and its location is output sequentially, and the output *valid* is set high to indicate that the output *decode_text* is valid. After that, when the other QR Code is decoded, the decoded result is output in the same way. After all the QR Codes in the test pattern have been found and decoded, set the output *finish* high to tell that the work is done.

When there are more than one QR Code in the test pattern, your design should be able to find and decode all of them. But the output order of these QR Codes is not limited. In the above example, you can output the QR Code at (0, 0) first as shown in Fig. 11, or you can output the QR Code at (30, 40) first as shown in Fig. 12.

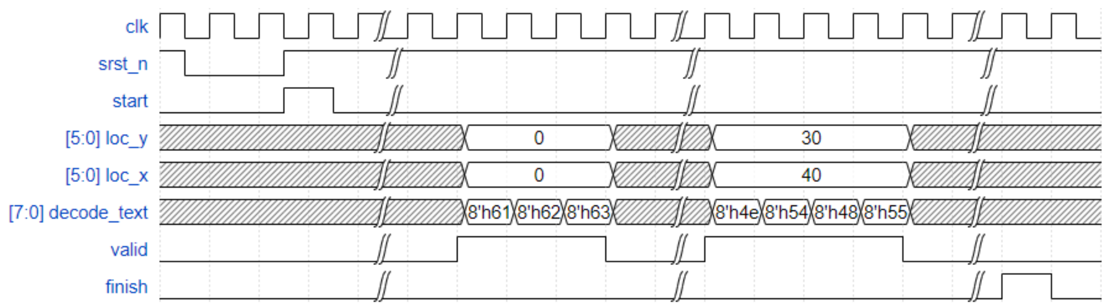


**Fig. 11** Timing diagram of the test pattern which contains two QR Codes (QR Code at (0, 0) is output first)

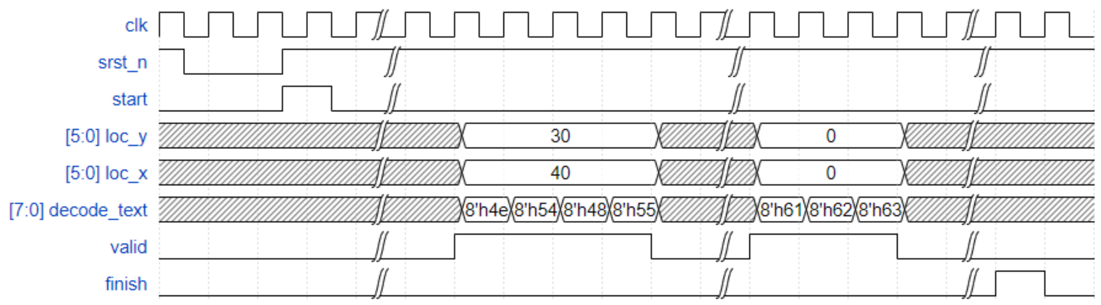


**Fig. 12** Timing diagram of the test pattern which contains two QR Codes (QR Code at (30, 40) is output first)

Besides, when output the decoded result of a QR Code, the output *decode_text* is not mandatory to be consecutive. Fig. 13 provides an example of nonconsecutive output behavior. However, the order of the output *decode_text* should still be correct.

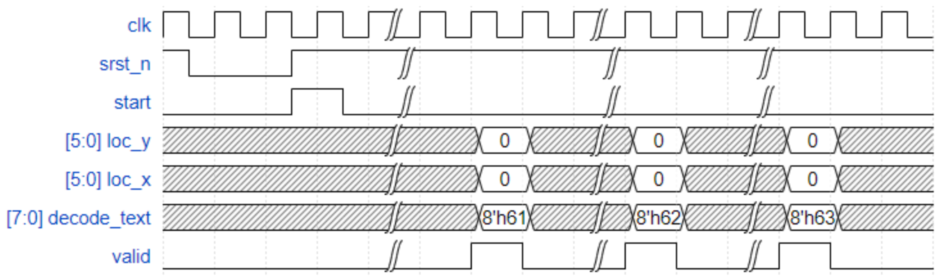


**Fig. 13** Another possible timing diagram when decoding a QR Code

Notice that although the output order of the QR Codes in a test pattern is not limited, the output behavior like Fig. 14 is not acceptable. You can only output the decoded result of the next QR Code after all the decoded result of the previous one have been output.

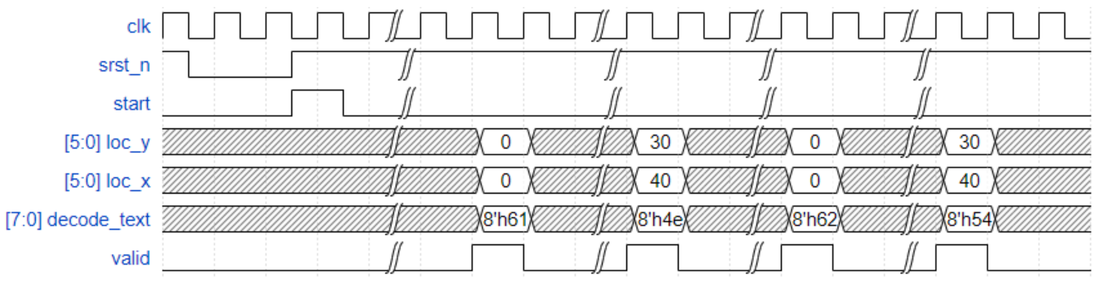


**Fig. 14** Not acceptable output behavior

## Grading Policy

The grading policy is divided into two part: functionality score and synthesis PI score. The functionality score depends on which types of test pattern (Rank A, Rank B, Rank C, Rank D) your design could pass during the pre-simulation. The synthesis PI score depends on the synthesis performance of your design, and the PI criteria is different for each grade. Table 2 shows detailed grading policy for each grade.

Before synthesis, you should run Spyglass examination to make sure your design is synthesizable. For fairness, please do logic synthesis with the provided scripts. Only two parts of the scripts can be modified. One is at **line 16** in **0_readfile.tcl**, you can add all your hdl files here. The other is at **line 5** in **synthesis.tcl**, you can modify the clock timing constraint "TEST_CYCLE" here. The timing slack of your synthesis result should not be negative.

Total Cycle Count C in PI depends on the simulation result of pattern

| **Passed Pattern** | **PI Grade** | **Synthesis PI ($A \times T \times C^2$)** | **Functionality Score (pass pre-sim)** | **Synthesis PI Score** | **Total Score** |
|---|---|---|---|---|---|
| Rank A<br>Rank B<br>Rank C<br>Rank D | A1 | $PI \leq 4.7 \times 10^{15}$ | 7 | 8 | 15 |
| | A2 | $4.7 \times 10^{15} < PI \leq 8.4 \times 10^{15}$ | 7 | 6 | 13 |
| | A3 | $PI > 8.4 \times 10^{15}$ | 7 | 4 | 11 |
| Rank B<br>Rank C<br>Rank D | B1 | $PI \leq 2.8 \times 10^{14}$ | 5 | 4 | 9 |
| | B2 | $2.8 \times 10^{14} < PI \leq 5 \times 10^{14}$ | 5 | 3 | 8 |
| | B3 | $PI > 5 \times 10^{14}$ | 5 | 2 | 7 |
| Rank C<br>Rank D | C1 | $PI \leq 8.6 \times 10^{13}$ | 3 | 3 | 6 |
| | C2 | $8.6 \times 10^{13} < PI \leq 1.5 \times 10^{14}$ | 3 | 2 | 5 |
| | C3 | $PI > 1.5 \times 10^{14}$ | 3 | 1 | 4 |
| Rank D | D1 | $PI \leq 5.4 \times 10^{12}$ | 1 | 2 | 3 |
| | D2 | $PI > 5.4 \times 10^{12}$ | 1 | 1 | 2 |

**Table 2.** Grading policy for each Rank

Note that for each grade, your design should pass all corresponding ranks of pattern listed in Table 2 to get the full functionality score. If your design fails pattern of Rank B or Rank C in the grade, you will get 2% points deduction for each Rank. If your design fails pattern of Rank D in the grade, you will get 1% point deduction.

For example, to get the full functionality score of grade A, your design should pass pattern of Rank A, Rank B, Rank C, and Rank D. If pattern of Rank A and Rank D are passed, but pattern of Rank B and Rank C are failed, you will get only $7 - 2 - 2 = 3$ points for the functionality score.

For another example, to get the full functionality score of grade B, your design should pass pattern of Rank B, Rank C, and Rank D. If pattern of Rank B and Rank C are passed, but pattern of Rank D is failed, you will get only $5 - 1 = 4$ points for the functionality score.

The performance index (PI) for this assignment is defined as:

$$\boldsymbol{PI} = \boldsymbol{A} \times \boldsymbol{T} \times \boldsymbol{C}^2$$

**A**: Total cell area which is shown in report_area_qrcode_decoder.out

**T**: TEST_CYCLE (your setting of clock timing constraint when synthesis)

**C**: Total cycle count to complete the whole simulation. It is shown on the terminal such as below when the whole simulation is passed and finished:

```
===================== Summary ======================
Congratulation! All patterns are successfully passed! \(O v O)/
Total cycle count C = xxxxxx
```

Note that we take the square of C when computing PI in this assignment, which means the effect of C on the PI is larger than A and T.

## Bonus

For works that pass the pre-simulation with test pattern of Rank A, 2% bonus score will be given to the best work of them in terms of the performance index (PI), and 1% bonus score will be given to the second-best work. (Only for those who submit the homework before the deadline.)

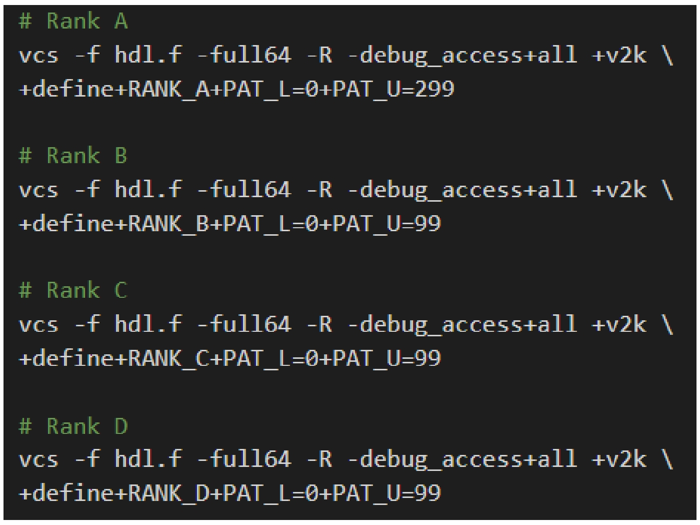
## Simulation

For Rank A, there are total 300 patterns.

For Rank B, Rank C, and Rank D, there are total 100 patterns for each Rank.

To pass each Rank, your design should pass all patterns in the Rank.

Simulation commands for each Rank are as below:



```
# Rank A
vcs -f hdl.f -full64 -R -debug_access+all +v2k \
+define+RANK_A+PAT_L=0+PAT_U=299

# Rank B
vcs -f hdl.f -full64 -R -debug_access+all +v2k \
+define+RANK_B+PAT_L=0+PAT_U=99

# Rank C
vcs -f hdl.f -full64 -R -debug_access+all +v2k \
+define+RANK_C+PAT_L=0+PAT_U=99

# Rank D
vcs -f hdl.f -full64 -R -debug_access+all +v2k \
+define+RANK_D+PAT_L=0+PAT_U=99
```

For debugging convenience, you can modify PAT_L and PAT_U to simulate with any specific range of patterns. For example, to simulate pattern No. 0 to No. 3, you can assign PAT_L to 0 and PAT_U to 3. If you just want to simulate with one certain pattern, you can assign PAT_L and PAT_U to the same number. For example, to only simulate pattern No. 77, you can assign both PAT_L and PAT_U to 77.

## Deliverable

1. **Synthesizable verilog**
   - qrcode_decoder.v
   - All other your own defined RTL files (*.v) (if any)
2. **Command-line arguments file**
   - hdl.f (should include all the RTL files used in your design)
3. **Spyglass report**
   - spyglass.rpt
4. **Synthesis scripts and results**
   - 0_readfile.tcl
   - synthesis.tcl
   - da.log
   - report_time_qrcode_decoder.out
   - report_area_qrcode_decoder.out
   - qrcode_decoder_syn.v
   - qrcode_decoder_syn.sdf
5. **Text file**
   - misc.txt

     Summarize your Grade and PI, including each value of A, T, and C, of your submitted design. A template is provided in the released package.

## File Organization

| Directory | Filename |
|---|---|
| HW4_{studentID}/ | misc.txt |
| | spyglass.rpt |
| HW4_{studentID}/hdl/ | qrcode_decoder.v |
| | All other your own defined RTL files (*.v) (if any) |
| HW4_{studentID}/sim/ | hdl.f |
| HW4_{studentID}/syn/ | 0_readfile.tcl |
| | synthesis.tcl |
| | da.log |
| | report_time_qrcode_decoder.out |
| | report_area_qrcode_decoder.out |
| | qrcode_decoder_syn.v |
| | qrcode_decoder_syn.sdf |

## Note:

If your studentID is 123456789, HW4_{studentID} denotes HW4_123456789.
Compress your whole folder HW4_{studentID} into HW4_{studentID}.zip and submit to eeclass.
Wrong file delivery or wrong file organization will get 1% punishment.

## Reference

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