國立清華大學 電機工程學系

實作專題研究成果報告

Toward fast and scalable decoder hardware for tailored made quantum error correction codes

專題領域： 通訊領域

組　　別：B495

指導教授：張鑑元 助理教授

組員姓名：林峻逸

研究期間：113年1月1日 至 113年11月1日止，共9個月

Abstract

The main objective of this research report is to gain an in-depth understanding of the fundamental concepts of quantum information, and to design and implement a decoder for quantum error correction codes, ultimately deploying it on an FPGA board. Due to the unique properties of quantum information, such as quantum superposition and entanglement, quantum bits (qubits) are prone to errors during transmission and processing due to external interference. Unlike classical error correction codes, traditional methods cannot be directly applied to correct errors in quantum information, as quantum measurement alters the quantum state, causing information loss during the measurement process.

Therefore, in order to effectively perform quantum error correction, specialized quantum coding schemes must be designed to detect and correct quantum errors, using these coding schemes to assist in the decoding process. This research focuses on the design of decoding algorithms for quantum error correction codes and their implementation on FPGA hardware, exploring the feasibility and performance of such solutions in practical quantum computing applications.

摘要

本研究報告的主要目的是深入理解量子資訊的基本概念，並針對量子錯誤更正碼的解碼器進行設計與實作，最終將其實作於 FPGA 板上。由於量子資訊具有特殊的性質，如量子疊加與量子糾纏等，這些特性使得量子比特在傳輸與處理過程中容易受到外界干擾而發生錯誤。與傳統的經典錯誤更正碼不同，由於量子測量會改變量子態，造成測量過程中的資訊損失，傳統的錯誤更正碼無法直接應用於量子資訊的錯誤更正。

因此，為了有效地進行量子錯誤更正，必須設計特殊的量子編碼方案來對量子錯誤進行偵測與修正，並利用這些編碼方案來協助解碼。為此，本研究著重於量子錯誤更正碼的解碼演算法設計，並將其具體實現於 FPGA 硬體平台上，探索其在實際量子計算應用中的可行性與效能。

一、研究背景與動機

量子計算是當前計算領域中的前沿技術，具有處理複雜問題的潛力，但也面臨量子比特容易受到環境干擾和錯誤影響的挑戰。由於量子測量會改變量子態，傳統的經典錯誤更正碼無法直接應用於量子計算，這需要特殊的量子錯誤更正碼來保護量子資訊。

量子錯誤更正碼的設計旨在提高量子計算的穩定性，常見的編碼方案如 Surface code、Shor code 和 Steane code 等。這些編碼能有效糾正錯誤，現今多以軟體解碼，但其解碼速度遠不及量子電腦的速度，需要高效的演算法和硬體實現支持。因此，本研究採用 Surface code 的編碼方式，做出 Look-up Table，實現兩種規格(d=3 及 d=5)，並將其實作於 FPGA 平台，探索其在實際量子計算中的可行性與效能

二、研究流程圖

三、研究方法

（一）理解量子資訊

1.翻閱書籍

首先我們先讀了 Quantum Computation and Quantum Imformation (by Michael A.Nielsen and Isaac L.Chuang) [1]，了解到量子位元的邏輯閘、有可能出現哪些錯誤還有 surface code 的編碼原理及規則。

2.量子閘以及量子資訊錯誤

量子位元可以用線性代數表示，由兩個向量組合｜0＞和｜1＞為基底。

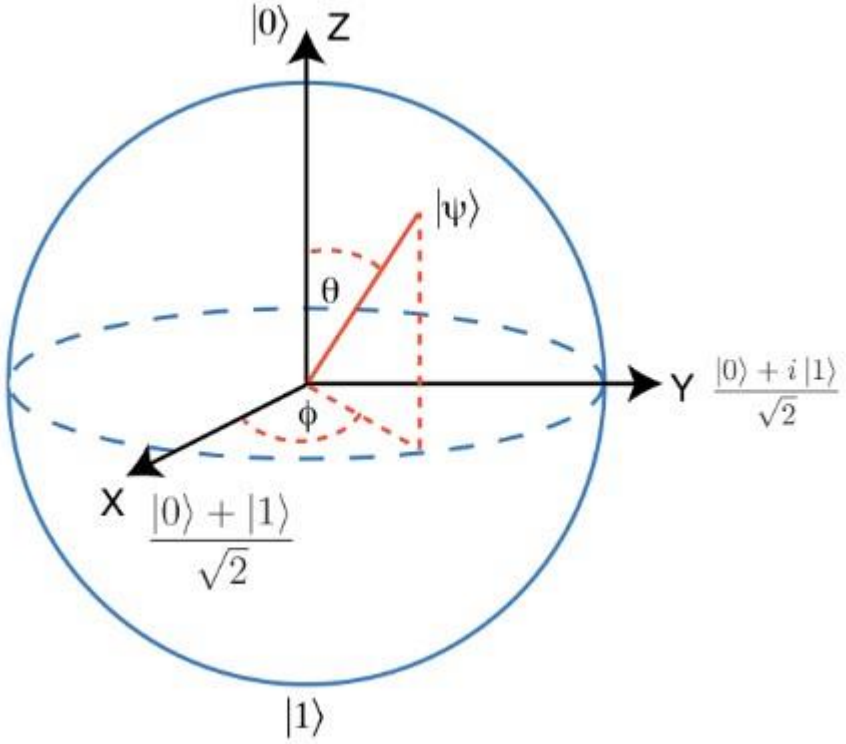
$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \quad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix}\alpha\\\beta\end{bmatrix}$$

量子位元有三個基本的邏輯閘分別是 X gate、Z gate、Hadamard gate(H gate)。

$$X = \begin{bmatrix}0 & 1\\1 & 1\end{bmatrix} \quad Z = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix} \quad H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$

每一個量子位元，可以用一個向量在 Bloch sphere 呈現其狀態，而會出現的錯誤分別有 X error 和 Z error，當一個位元同時出現 X error 和 Z error 時，將以 Y error 表示。X error 代表的是，該位元經過干擾後，位元向量以 X 軸旋轉 180 度；Z error 代表的是，該位元經過干擾後，位元向量以 Z 軸旋轉 180 度。



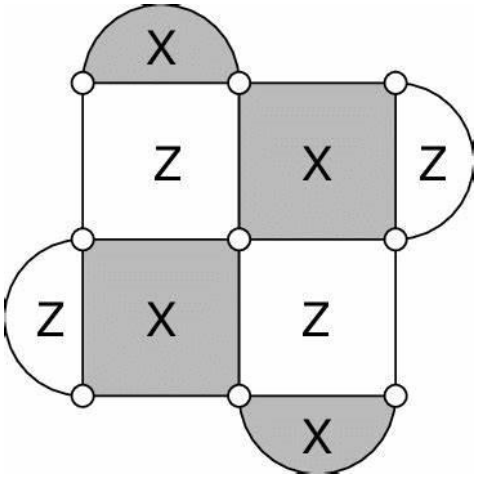
[2]

上圖為 Bloch sphere

對於發生 X error 的位元，我們可以使用 X gate 將其復原，同理發生 Z error 的位元，我們可以使用 Z gate 將其復原，而量子更正碼正是用此原理進行修復。

3.Surface code 的原理

本研究報告採用 Surface code 的編碼方式。其架構分別是以 data qubit 和 ancillary qubit 以距離 d（表示 Surface code 大小）按照一定的方式排列，一面 Surface code 只能表示實際的一個位元。在這次的研究，我選擇 d=3 和 d=5 的 Surface code 進行研究，接下來以 d=3 的 Surface code 說明原理。



上圖為 d=3 的 Surface code

參數 d 決定了 Surface code 的距離，d=3、5……必須為奇數，表示白色的點單一方向的距離。data qubit 代表儲存實際資料的位元，為上圖的白色圓圈，一共九個。而 ancillary qubit 分成兩種，X ancillary qubit 和 Z ancillary qubit，分別在上圖 X 字母與 Z 字母處，彼此相間，各四個。每個 ancillary qubit 皆會與身邊四個(或兩個) data qubit 形成一個 stablizer，當我們對其量測時，相當於對周圍的 data qubit 進行奇偶校正，當錯誤的 data qubit 為奇數個，ancillary qubit 的量測結果為 1，錯誤的 data qubit 為偶數個，ancillary qubit 的量測結果為 0，我們便能得知錯誤的 data qubit 位於什麼地方。再來 X stablizer 可以偵測出 Z error；Z stablizer 可以偵測出 X error，我們便能得知錯誤的 data qubit 是何種錯誤，來使用對應的邏輯閘修復。

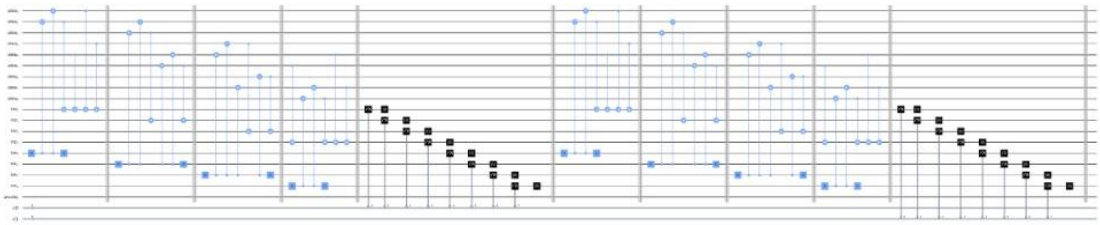
（二）生成量子電路

1.Syndrome

在上一個章節，我們知道對 ancillary qubit 量測會得到一個值，我們將這接數值收集起來，就會得到 syndrome。每一種錯誤的 pattern 會對應到一個 syndrome。我們藉由 syndrome 回推發生錯誤的地方。

2. Qiskit_surface_codes [3]的電路

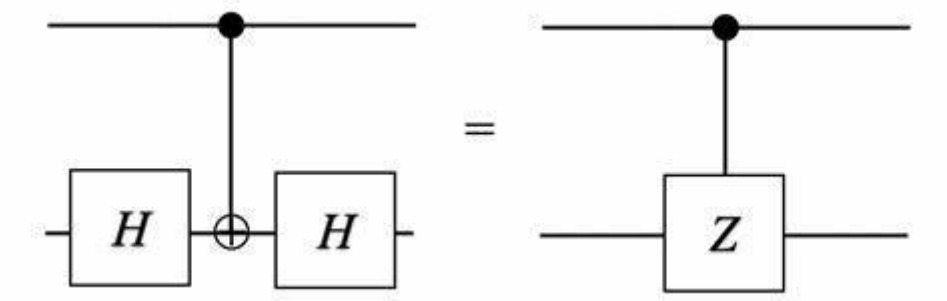
根據教授指示，我們使用了在 Github 上的量子電路模擬的程式碼，我們可以自由的選擇要用 d 為多少的 Surface code，並將錯誤放置在隨意的位置，並生成對應的 syndrome，在 d=3 的 Surface code，把一個 X error 和 Z error 放在所有可能的位置，並生成對應的 syndrome；d=5 的 Surface code，把一個或兩個的 X error 和 Z error 放在所有可能的，並生成對應的 syndrome



上圖是 Qiskit_surface_codes 的生成量子電路

3.量子邏輯閘和電路的實現

理想中，我們會用的是 X gate、Z gate、H gate，協助我們偵錯，但在實際的狀況下，並非每一種 gate 都能順利地被製造出來，但很幸運的可以藉由另外兩個 gate 組合，製造等效的 gate，在此模擬程式，就使用兩個 H gate 和一個 X gate，組成 Z gate。



上圖為 Z gate 等效 H gate 和一個 X gate

（三）生成 Look-up Table

1. Look-up Table 形成

我們要生成 Look-up Table 並轉成 verilog 可讀的形式，但我們必須留意，要刪掉重複的 syndrome，因為事實上，兩種（或以上）不同的 error 位置可能對產生相同的 syndrome，導致 syndrome –to-error 並非 1-to-1，在解碼上是致命的錯誤。最後我們寫了一段程式碼，協助我們完成。

```
[24]: from google.colab import files
      uploaded = files.upload()

      選擇檔案 沒有選擇檔案   Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
      Saving two_X_errors.txt to two_X_errors.txt

[47]: with open('two_X_errors.txt', 'r') as file:
        i=0
        j=1
        for line in file:
          index = format(pow(2, i)+pow(2, j), "b")
          print("%25s " % index, end="")
          print(line.strip())
          i=i+1
          if (i==25):
            i=0
            j+=j
```

上圖是轉成 verilog 可讀的輔助程式碼

2.如何決定刪除的 syndrome？

事實上刪除 syndrome 是有一定依據的，但我們在這裡是考慮理想的 Surface code，若對應到同一種 syndrome，這兩個 error 是等價的，只需要保留一個，所以我們只保留第一個出現的 error，刪除重複的 syndrome。

（四）實作在 FPGA 及驗證

1.Look-up Table based 的 decoder

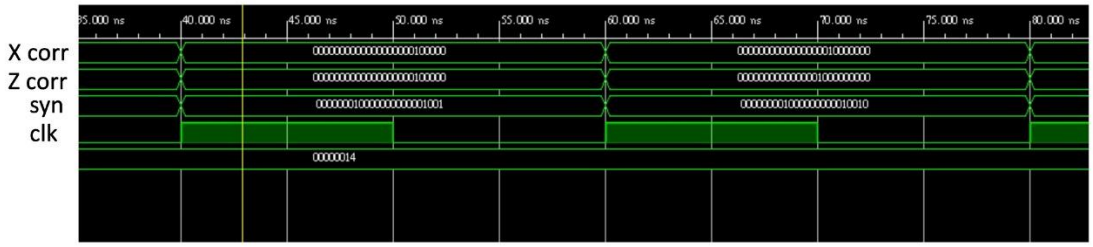
在這個 decoder，input 為 clk(clock)、syn(syndrome)，output 為 z_corr(correction for z error)、x_corr(correction for x error)，因為是 Look-up Table based 的 decoder 所以架構並不難，分別由一個 top module 和兩個 Look-up Table module 組成。

```
module top(clk,syn, x_corr, z_corr);
        output [24:0] z_corr;  // output correct code for z
        output [24:0] x_corr;  // output correct code for x
        input [23:0] syn;  // input syndrone x_z
        input clk;

  LUT ULUT(.syn(syn),.x_corr(x_corr),.z_corr(z_corr),.clk(clk));

endmodule
```

上圖為 top module 的架構

2. 驗證方法

我採取了兩種驗證方法，分別是自己寫 test bench 和使用 google 量子團隊的 open source data。



上圖為 test bench 的部分結果

我隨意寫不同的 syndrome，並檢查 waveview，結果和預期的一樣。至於 google 量子團隊的 open source data 的驗證，礙於時間問題，我在提交報告之前未能完成。

四、研究結果

(一) d=3 surface code

1. Look-up Table in verilog

```
module LUT_X(data_qubit, syn); // for X stabilizer
    output [8:0] data_qubit;
    input [3:0] syn;
    reg [8:0] data_qubit;

    always @*
    case(syn)
        4'd0: data_qubit = 9'b000000000;
        4'd1: data_qubit = 9'b000000001;
        4'd2: data_qubit = 9'b000000100; // Z2 or Z5
        4'd3: data_qubit = 9'b000000010;
        4'd4: data_qubit = 9'b000001000; // Z6 or Z3
        4'd5: data_qubit = 9'b001000001;
        4'd6: data_qubit = 9'b000010000;
        4'd7: data_qubit = 9'b000010001;
        4'd8: data_qubit = 9'b100000000;
        4'd9: data_qubit = 9'b100000001;
        4'd10: data_qubit = 9'b100000100;
        4'd11: data_qubit = 9'b100010010;
        4'd12: data_qubit = 9'b010000000;
        4'd13: data_qubit = 9'b010000001;
        4'd14: data_qubit = 9'b100010000;
        4'd15: data_qubit = 9'b010000010;
        default: data_qubit = 9'b000000000;
    endcase
endmodule
```

一個 z error 的 Look-up Table

```
module LUT_Z(data_qubit, syn); // for Z stabilizer
output [8:0] data_qubit;
input [3:0] syn;
reg[8:0] data_qubit;

always @*
    case(syn)
        4'd0: data_qubit = 9'b000000000;
        4'd1: data_qubit = 9'b000000010; // X1 or X0
        4'd2: data_qubit = 9'b000000100;
        4'd3: data_qubit = 9'b000000110;
        4'd4: data_qubit = 9'b001000000;
        4'd5: data_qubit = 9'b000001000;
        4'd6: data_qubit = 9'b001000100;
        4'd7: data_qubit = 9'b000001100;
        4'd8: data_qubit = 9'b100000000; //X8 or X7
        4'd9: data_qubit = 9'b000010000;
        4'd10: data_qubit = 9'b000100000;
        4'd11: data_qubit = 9'b000010100;
        4'd12: data_qubit = 9'b011000000;
        4'd13: data_qubit = 9'b001010000;
        4'd14: data_qubit = 9'b001100000;
        4'd15: data_qubit = 9'b000101000;
        default: data_qubit = 9'b000000000;
    endcase
endmodule
```

一個 x error 的 Look-up Table

2.waveview

上圖為 test bench 的部分結果

因為是自己寫的 test bench，所以原則上結果都會一致，之後若完成 open source data[4]的驗證，便能比較結果是否準確。

(二) d=5 surface code

1. Look-up Table in verilog

```
module LUT_X(syn,data_qubit,clk);
    output [24:0] data_qubit;
    input [11:0] syn;
    input clk;
    reg [24:0] data_qubit;

    always @(posedge clk)
     case(syn)
        12'b000000000001: data_qubit = 25'b0000000000000000000000001; // syn 1
        12'b000000000101: data_qubit = 25'b0000000000000000000000010; // syn 2
        12'b000000000110: data_qubit = 25'b0000000000000000000000100; // syn 3
        // 12'b000000001010: data_qubit = 25'b0000000000000000000001000; // syn 4
        // 12'b000000001000: data_qubit = 25'b0000000000000000000010000; // syn 5
        12'b000000010000: data_qubit = 25'b0000000000000000000100000; // syn 6
        12'b000000010100: data_qubit = 25'b0000000000000000001000000; // syn 7
        12'b000000100100: data_qubit = 25'b0000000000000000010000000; // syn 8
        12'b000000101000: data_qubit = 25'b0000000000000000100000000; // syn 9
        12'b000000001000: data_qubit = 25'b0000000000000001000000000; // syn 10
        12'b000000010000: data_qubit = 25'b0000000000000010000000000; // syn 11
        12'b000001010000: data_qubit = 25'b0000000000000100000000000; // syn 12
        12'b000001100000: data_qubit = 25'b0000000000001000000000000; // syn 13
        12'b000010100000: data_qubit = 25'b0000000000010000000000000; // syn 14
        12'b000010000000: data_qubit = 25'b0000000000100000000000000; // syn 15
        12'b000100000000: data_qubit = 25'b0000000001000000000000000; // syn 16
        12'b000101000000: data_qubit = 25'b0000000010000000000000000; // syn 17
        12'b001001000000: data_qubit = 25'b0000000100000000000000000; // syn 18
        // 12'b001010000000: data_qubit = 25'b0000001000000000000000000; // syn 19
        // 12'b000010000000: data_qubit = 25'b0000010000000000000000000; // syn 20
        12'b000100000000: data_qubit = 25'b0000100000000000000000000; // syn 21
        12'b010100000000: data_qubit = 25'b0001000000000000000000000; // syn 22
        12'b011000000000: data_qubit = 25'b0010000000000000000000000; // syn 23
        12'b101000000000: data_qubit = 25'b0100000000000000000000000; // syn 24
        12'b100000000000: data_qubit = 25'b1000000000000000000000000; // syn 25
```

一個 z error 的 Look-up Table

```
module LUT_Z(syn,data_qubit,clk);
    output [24:0] data_qubit;
    input [11:0] syn;
    input clk;
    reg [24:0] data_qubit;

    always @(posedge clk)
     case(syn)
        12'b000000000001: data_qubit = 25'b0000000000000000000000001; // syn 0
        // 12'b000000000001: data_qubit = 25'b0000000000000000000000010; // syn 1
        // 12'b000000000010: data_qubit = 25'b0000000000000000000000100; // syn 2
        12'b000000000010: data_qubit = 25'b0000000000000000000001000; // syn 3
        12'b000000000100: data_qubit = 25'b0000000000000000000010000; // syn 4
        12'b000000001001: data_qubit = 25'b0000000000000000000100000; // syn 5
        12'b000000010001: data_qubit = 25'b0000000000000000001000000; // syn 6
        12'b000000010010: data_qubit = 25'b0000000000000000010000000; // syn 7
        12'b000000100010: data_qubit = 25'b0000000000000000100000000; // syn 8
        12'b000000100100: data_qubit = 25'b0000000000000001000000000; // syn 9
        12'b000001001000: data_qubit = 25'b0000000000000010000000000; // syn 10
        12'b000001010000: data_qubit = 25'b0000000000000100000000000; // syn 11
        12'b000010010000: data_qubit = 25'b0000000000001000000000000; // syn 12
        12'b000010100000: data_qubit = 25'b0000000000010000000000000; // syn 13
        12'b000100100000: data_qubit = 25'b0000000000100000000000000; // syn 14
        12'b001001000000: data_qubit = 25'b0000000001000000000000000; // syn 15
        12'b010001000000: data_qubit = 25'b0000000010000000000000000; // syn 16
        12'b010010000000: data_qubit = 25'b0000000100000000000000000; // syn 17
        12'b100010000000: data_qubit = 25'b0000001000000000000000000; // syn 18
        12'b100100000000: data_qubit = 25'b0000010000000000000000000; // syn 19
        12'b001000000000: data_qubit = 25'b0000100000000000000000000; // syn 20
        // 12'b010000000000: data_qubit = 25'b0001000000000000000000000; // syn 21
        12'b010000000000: data_qubit = 25'b0010000000000000000000000; // syn 22
        // 12'b100000000000: data_qubit = 25'b0100000000000000000000000; // syn 23
        12'b100000000000: data_qubit = 25'b1000000000000000000000000; // syn 24
```

一個 x error 的 Look-up Table

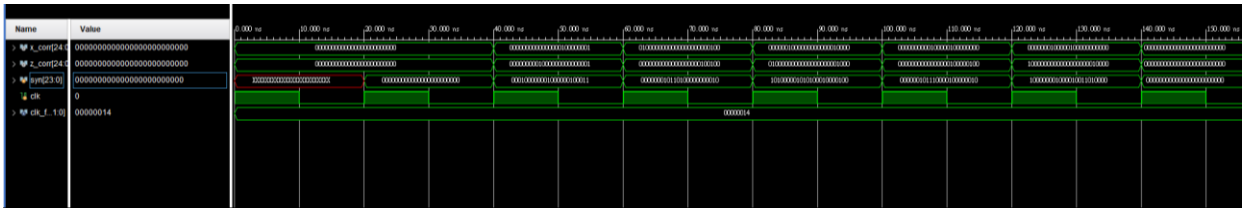
```
12'b000000000100: data_qubit = 25'b11;
12'b000000000111: data_qubit = 25'b101;
12'b000000001011: data_qubit = 25'b1001;
12'b000000001001: data_qubit = 25'b10001;
12'b000000010001: data_qubit = 25'b100001;
12'b000000010101: data_qubit = 25'b1000001;
12'b000000100101: data_qubit = 25'b10000001;
12'b000000101001: data_qubit = 25'b100000001;
// 12'b000000001001: data_qubit = 25'b1000000001;
// 12'b000000010001: data_qubit = 25'b10000000001;
12'b000001010001: data_qubit = 25'b100000000001;
12'b000001100001: data_qubit = 25'b1000000000001;
12'b000010100001: data_qubit = 25'b10000000000001;
12'b000010000001: data_qubit = 25'b100000000000001;
12'b000100000001: data_qubit = 25'b1000000000000001;
12'b000101000001: data_qubit = 25'b10000000000000001;
12'b001001000001: data_qubit = 25'b100000000000000001;
12'b001010000001: data_qubit = 25'b1000000000000000001;
// 12'b000010000001: data_qubit = 25'b100000000000000000001;
// 12'b000100000001: data_qubit = 25'b1000000000000000000001;
12'b010100000001: data_qubit = 25'b10000000000000000000001;
12'b011000000001: data_qubit = 25'b100000000000000000000001;
12'b101000000001: data_qubit = 25'b1000000000000000000000001;
12'b100000000001: data_qubit = 25'b1000000000000000000000001;
12'b000000000011: data_qubit = 25'b110;
12'b000000001111: data_qubit = 25'b1010;
12'b000000001101: data_qubit = 25'b10010;
// 12'b000000010101: data_qubit = 25'b100010;
// 12'b000000010001: data_qubit = 25'b1000010;
12'b000000100001: data_qubit = 25'b10000010;
12'b000000101101: data_qubit = 25'b100000010;
// 12'b000000001101: data_qubit = 25'b1000000010;
// 12'b000000010101: data_qubit = 25'b10000000010;
12'b000001010101: data_qubit = 25'b100000000010;
12'b000001100101: data_qubit = 25'b1000000000010;
12'b000010100101: data_qubit = 25'b10000000000010;
12'b000010000101: data_qubit = 25'b100000000000010;
12'b000100000101: data_qubit = 25'b1000000000000010;
12'b000101000101: data_qubit = 25'b10000000000000010;
12'b001001000101: data_qubit = 25'b100000000000000010;
12'b001010000101: data_qubit = 25'b1000000000000000010;
// 12'b000010000101: data_qubit = 25'b100000000000000000010;
// 12'b000100000101: data_qubit = 25'b1000000000000000000010;
```

兩個 z error 的部分 Look-up Table

```
//12'b000000000000: data_qubit = 25'b11;
12'b000000000011: data_qubit = 25'b101;
// 12'b000000000011: data_qubit = 25'b1001;
12'b000000000101: data_qubit = 25'b10001;
12'b000000001000: data_qubit = 25'b100001;
12'b000000010000: data_qubit = 25'b1000001;
12'b000000010011: data_qubit = 25'b10000001;
12'b000000100011: data_qubit = 25'b100000001;
12'b000000100101: data_qubit = 25'b1000000001;
12'b000001001001: data_qubit = 25'b10000000001;
12'b000001010001: data_qubit = 25'b100000000001;
12'b000010010001: data_qubit = 25'b1000000000001;
12'b000010100001: data_qubit = 25'b10000000000001;
12'b000100100001: data_qubit = 25'b100000000000001;
12'b001001000001: data_qubit = 25'b1000000000000001;
12'b010001000001: data_qubit = 25'b10000000000000001;
12'b010010000001: data_qubit = 25'b100000000000000001;
12'b100010000001: data_qubit = 25'b100000000000000001;
12'b100100000001: data_qubit = 25'b1000000000000000001;
12'b001000000001: data_qubit = 25'b1000000000000000001;
12'b010000000001: data_qubit = 25'b10000000000000000001;
// 12'b010000000001: data_qubit = 25'b100000000000000000000001;
12'b100000000001: data_qubit = 25'b1000000000000000000001;
// 12'b100000000001: data_qubit = 25'b10000000000000000000001;
// 12'b000000000011: data_qubit = 25'b110;
// 12'b000000000011: data_qubit = 25'b1010;
// 12'b000000000101: data_qubit = 25'b10010;
// 12'b000000001000: data_qubit = 25'b100010;
// 12'b000000010000: data_qubit = 25'b1000010;
// 12'b000000010011: data_qubit = 25'b10000010;
// 12'b000000100011: data_qubit = 25'b100000010;
// 12'b000000100101: data_qubit = 25'b1000000010;
// 12'b000001001001: data_qubit = 25'b10000000010;
// 12'b000001010001: data_qubit = 25'b100000000010;
// 12'b000010010001: data_qubit = 25'b1000000000010;
// 12'b000010100001: data_qubit = 25'b10000000000010;
// 12'b000100100001: data_qubit = 25'b100000000000010;
// 12'b001001000001: data_qubit = 25'b1000000000000010;
// 12'b010001000001: data_qubit = 25'b10000000000000010;
// 12'b010010000001: data_qubit = 25'b100000000000000010;
// 12'b100010000001: data_qubit = 25'b100000000000000010;
// 12'b100100000001: data_qubit = 25'b1000000000000000010;
// 12'b001000000001: data_qubit = 25'b1000000000000000010;
// 12'b010000000001: data_qubit = 25'b10000000000000000010;
// 12'b010000000001: data_qubit = 25'b100000000000000000010;
// 12'b100000000001: data_qubit = 25'b100000000000000000010;
```

兩個 X error 的部分 Look-up Table

2. waveview



因為是自己寫的 test bench，所以原則上結果都會一致，之後若完成 open source data 的驗證，便能比較結果是否準確。

(三)差異比較

1.d=3 v.s. d=5

d 代表 Surface code 的距離，決定了可以更正 error 的最大數量，由以下公式 number of errors=(d+1)/2，所以在 d=5 的 Surface code，我製造出兩個 errors 的 Look-up Table。若要比較容錯率，d=5 的 Surface code 肯定是勝於 d=3 的 Surface code。

2. d=3、d=5 v.s.Google open source data

Google 量子團隊使用的 decoder 是基於 sparse blossom matching [5] with a variant of the two-step re-weighting strategy[6]，其兩者最大的差別在於 Look-up Table 會隨著 d 參數增加，消耗更多的記憶體空間，Google 量子團隊使用的 decoder 並沒有這個問題。除此之外，Google 量子團隊使用的 decoder 因為使用了 sparse blossom matching，考量到實際量子電腦的情況，所以根據不同的 syndrome 會有最佳解，而我們 Look-up Table based 的 decoder 因為處於理想狀態，會發生有兩個(或以上)的等價解。

五、結論

本專題使用 Look-up Table 的方式製作量子錯誤更正碼專用的 decoder，並在理想的狀態下進行，成功設計並實現了一個基於 Surface code 的量子錯誤更正解碼器，並將其實作於 FPGA。研究過程中，我們深入理解了量子計算的基本概念，特別是量子錯誤更正碼的運作原理，並通過 Look-up Table 的設計有效解決了解碼過程中的複雜性問題。經過測試與驗證，我們的解碼器能夠準確地偵測並修正量子比特中的 X 錯誤和 Z 錯誤，證明了所設計的解碼器在實際量子計算中的可行性與效能，並提供軟體解碼以外的可能性。

六、參考資料

[1] Quantum Computation and Quantum Imformation (author: Michael A.Nielsen and Isaac L.Chuang)

[2] 台灣大學計資中心電子報
https://www.cc.ntu.edu.tw/chinese/epaper/home/20230320_006408.html

[3]qiskit_surface_codes
qiskit_surface_codes/tutorials/1_surface_code_encoding.ipynb at master · joon-huh/qiskit_surface_codes · GitHub

[4] Google open source data
https://zenodo.org/records/13273331

[5]Sparse blossom matching
O. Higgot *et al.*, Sparse Blossom: correcting a million errors per core second with minimum-weight matching arXiv:2303.15933 (2023)

[6] A variant of the two-step re-weighting strategy
A. Fowler, Optimal complexity correction of correlated errors in the surface code arXiv:1310.0863 (2013)

七、計劃管理及團隊分工

(一) 計劃管理

在第一學期，主要進行教授指定的論文進行 paper study，每兩周跟教授進行重點彙報，在這段時間內教授會問我問題，以知道我對量子資訊的熟悉度，同時進行教學。

在第二學期，便開始上手量子模擬的相關環境，並逐漸生成 Look-up Table，同時做簡單的驗證，一樣是每周報告進度。到最後一個月準備使用 Google open source data，並完成報告，但礙於 Google open source data 是 raw data 完全沒有註解，我花了很多時間在理解，最後只能呈現沒有完整驗證的結果。

(二) 團隊合作

在這個專題，由我和學弟黃柏霖一起完成，不過他還沒參加專題成果展。每個星期，我們會固定約時間討論，前一周的作業情形，並同時規劃這一周的進度，如果有什麼問題都盡量在這個時間提出。

在報告進度，我們會一起製作 ppt 確保彼此進度和理解都有跟上，每周輪流報告，訓練彼此的表達能力。