# Final Project – N-storey Elevator Controller

104021219 鄭余玄

## A. 設計概念

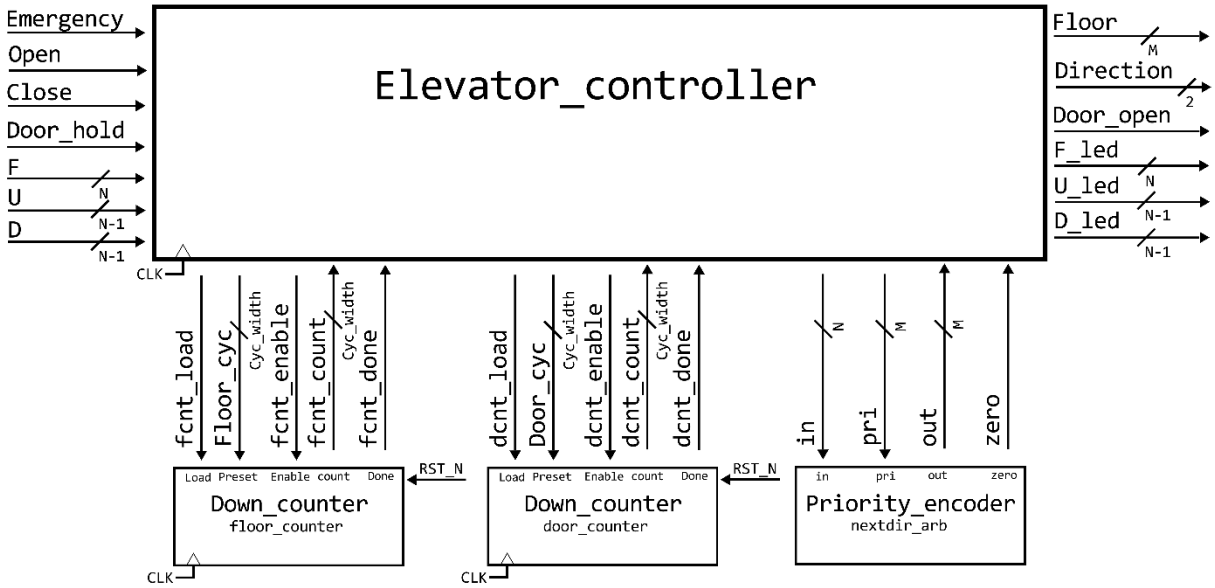
在設計電梯控制器狀態時，我發現其實電梯只有分三種狀態，分別是向上向下和靜止，這三個狀態是與樓層無關的。因此我在設計整個控制器時，電梯樓層是可以被參數化的，所以我的 Final Project 標題是 N-層樓的電梯控制器，也就是可以做成 101 大樓的電梯控制器。

不過，在 Project 的規格上需要和原本訂的 I/O 有修改。F = {F4, F3, F2, F1} 等等與樓層有關的 I/O，需要在輸入或輸出時就已經 aggregation。主要模組中，有搭配兩個 Down Counter 分別作為電梯和樓層的計數器，以及 Priority Encoder 作為很多樓層間選擇的仲裁器以及 Encoder。

## B. 設計規格

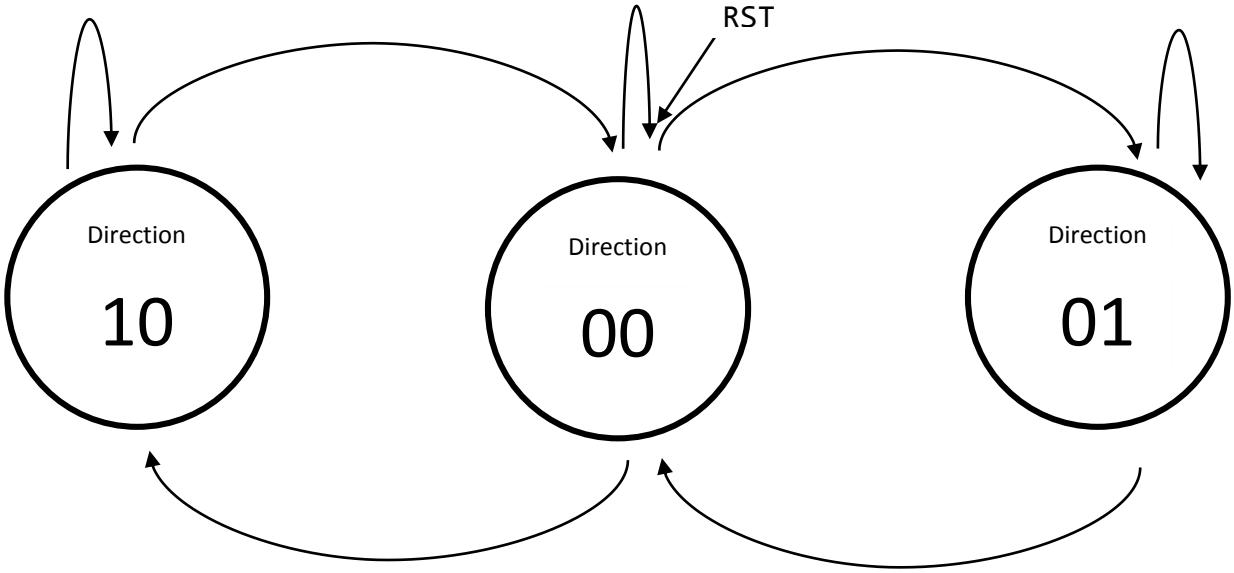
1. 電梯 I/O 在 Block diagram 中都有仔細說明。
2. 可以同時有多層樓一按電梯。
3. 當電梯在移動或靜止時，輸入的訊號都會相對應地反映在 LED 輸出上。
4. 可以按 Open、Close 和 Door_hold 來開門、關門以及延長關門。
5. 當按下 Emergency 時，移動的電梯會到最近的樓層開門；靜止的電梯會開門。
6. 樓層間 cycle 數和開門 cycle 數均參數化，可以任意調整。
7. 可以任意延伸樓層。

C. Block diagram




| **Parameter** | **說明** |
|---|---|
| N = 4 | N 層樓 |
| M = 2 | M = [lg N] 做 encoding |
| Floor_cyc = 12 | 樓層間 cycle 數 |
| Door_cyc = 5 | 開門到關門的 cycle 數 |
| Cyc_width = 4 | max{[lg Floor_cyc], [lg Door_cyc]} 最長資料寬度 |
| **Input** | **說明** |
| CLK | Clock |
| RST_N | Negative-edge reset |
| Emergency / Open / Close / Door_hold | 如同講義上的功能 |
| F [N - 1:0] | 電梯內，第 1 樓到第 N 樓的按鈕 |
| U [N - 2:0] | 樓層中，第 1 樓到第 N - 1 樓的向上按鈕 |
| D [N - 2 :0] | 樓層中，第 2 樓到第 N 樓的向下按鈕 |
| **Output** | **說明** |
| Floor [M - 1:0] | 1 到 N 樓 (用 0 ~ N - 1 來表示) |
| Direction [1:0] | 方向（00 停止；10 向上；01 向下） |
| Door_open | 指示電梯門是否是開的 |
| F_led [N - 1:0] | 電梯內，第 1 樓到第 N 樓的按鈕 LED |
| U_led [N - 2:0] | 樓層中，第 1 樓到第 N - 1 樓的向上按鈕 LED |
| D_led [N - 2:0] | 樓層中，第 2 樓到第 N 樓的向下按鈕 LED |

D. FSM




E. Test bench 環境與使用

針對 elevator_controller 這個模組，我有寫兩個 test bench，分別是 elevator4.v 和 elevator128.v。在資料夾中下指令：

- 「make」或是「make 4」，就會執行 elevator4.v 的測試模組。
- 「make 128」，就會執行 elevator128.v 的測試模組。
- 「make Allpass」，就會先執行 elevator4.v 再執行 elevator128.v 的測試模組。
- 「make clean」，清除 elevator4.fsdb 和 elevator128.fsdb 等 Verilog 相關 log 檔案。

F. Test bench 說明

附註：test bench 中，為了方便閱讀，以及更接近真實電梯，所以輸出的 Floor 信號有額外加一個 DFF 和 floor = Floor + 1。

1. Day 1: a 4-storey building with elevator

a. Scenario #1

i. A 和 B 在二樓搭電梯，分別想要到三樓和四樓。

ii. 電梯先載 A 到三樓，再載 B 到四樓。

iii. 當 B 從四樓電梯出來得時候，CD 在三樓按電梯，分別想要到二樓和一樓。

iv. 電梯到三樓後，接著載 C 到二樓，D 到一樓。

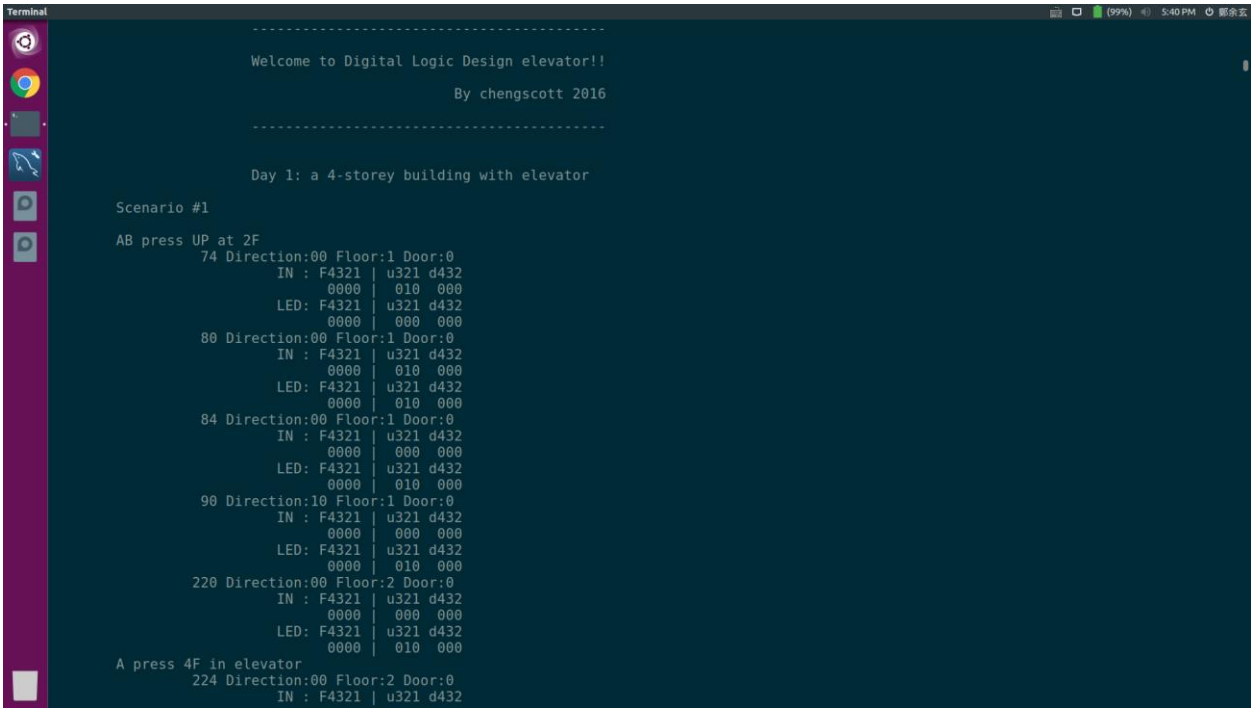
b. Scenario #2

i. A 在三樓搭電梯

ii. A 持續按了開門，電梯門一直開著

iii. A 按了關門，門就關了

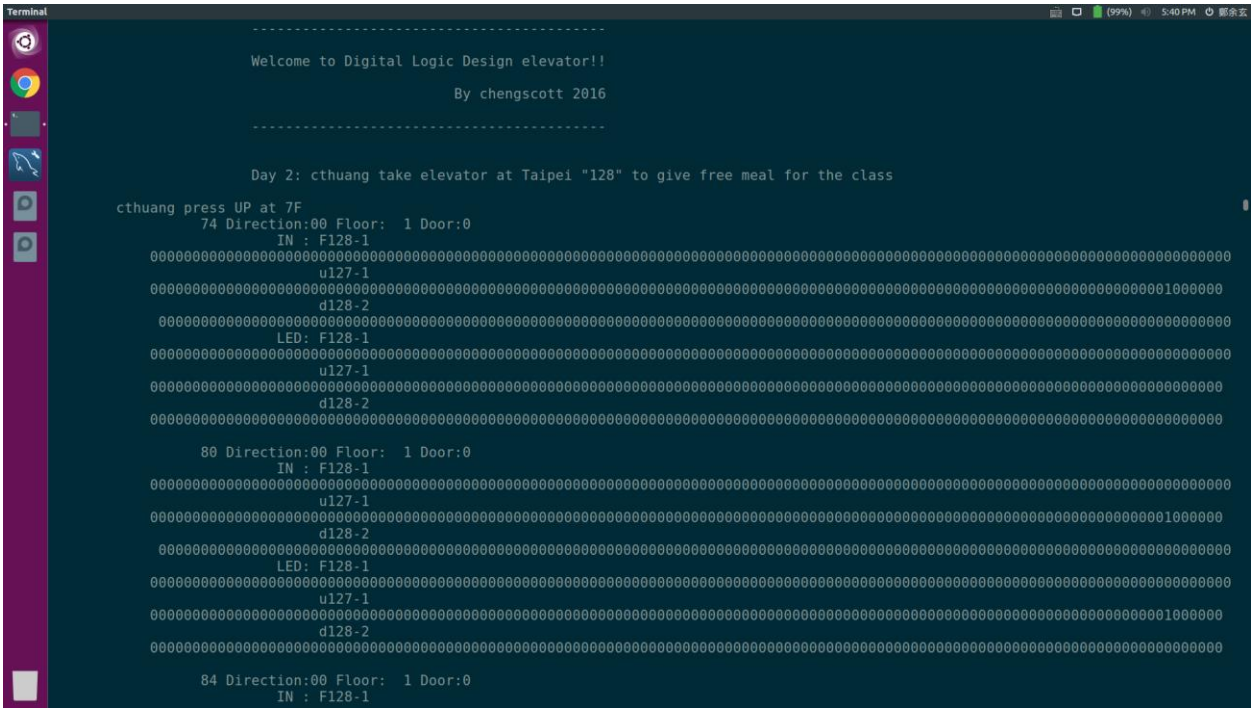
iv. 突然 final project（電梯）著火了，電梯馬上到最近的樓層開門




2. Day 2: cthuang take elevator at Taipei "128" to give free meal for the class

i. cthuang 教授從七樓辦公室搭電梯

ii. 搭到第 85 層樓




## G. 心得和說明

平常我在搭電梯時，其實很少注意電梯是怎麼跑的，因此在做這個 project 之前，我就實地的去搭了好幾趟電梯。結果在經過某樓層的時候，聽到外面的人在抱怨：奇怪，今天電梯怎麼等那麼久；還有：X 的，是誰在按電梯啦。於是我就默默地結束了實地訪查電梯。

priority encoder 模組主要是用來當作仲裁器，而它的優先序有分兩種，是由行進方向來決定。電梯如果正在向上，高樓層優先序會較高，反之亦然。我認為這是比較合理的設計，因為在構想的時候，有想過使用 round robin 做 load balance，不過仔細想想這部大符合現實情況。因為這樣會讓使用者在操作這台電梯時，沒有一個統一的原則，會讓使用者感到很困惑，所以做後還是用一般電梯決策方式。

這次程式寫起來有些粗心大意，priority encoder 部分不小心把 N 打成 M，結果就 de 了一個下午的 bug，再加上 Verilog debug 要盯著 nWave 的波形一個個檢查，而且很難想到居然會是 combinational block 寫錯，找到 bug 的時候簡直快要崩潰了。

此外，強烈建議教授在工作站上面安裝 git 版本管理套件，不然常常要改程式碼都會心驚膽跳，可能原本會動改一改就掛了，然後又不知從何復原，最後只能變成用資料夾做版本管理。

我覺得「System Design and Interfaces」這份講義的內容很重要，因為我在開始打程式碼以前，就先畫了好幾張 Block diagram 的草稿，先確定整個設計的規格。因此我在打程式碼的時候，除了不小心打 N 打成 M 這種錯誤以外，基本上寫起來就很順暢。如果有遇到問題，也可以在先畫好的圖上做確認，就可以判斷是邏輯上的問題，還是程式碼的問題。