中華民國第 63 屆中小學科學展覽會

作品說明書

高中組　電腦與資訊學科

佳作

052504

細胞運算－於康威生命遊戲的自動化數位電路設計方法探究

學校名稱：桃園市立南崁高級中學

作者：	指導老師：
高二 鍾佳龍	蔡明勳

關鍵詞：康威生命遊戲、數位電路、電子設計自動化

摘要

「康威生命遊戲」是一款在 1970 年由劍橋大學數學家約翰·康威（John Conway）發明的單人遊戲，由棋盤構成的細胞組成，可以模擬邏輯閘元件，並且可以構建簡易計算器。研究者在建構加法器和減法器的過程中，發覺手動放置電路元件非常耗時費力且容易出錯。為了解決這個問題，研究者決定開發一個能自動擺放元件的程式，以節省布局和放置元件所需的時間。參考了電子設計自動化(Electronic design automation)的概念，使用 Python 語言結合康威生命遊戲的應用接口模組，開發了一款自動電路布局程式。只要輸入邏輯電路的布林函式，就能自動設計、布局和放置元件，大大縮短了製作邏輯電路上的時間成本。因可直觀地展示訊號的傳遞和邏輯閘的運作，也有助於學生理解邏輯電路的運作原理，應用於教學。

壹、前言

一、研究動機

在一次的研究邏輯閘的主題中我看到一則影片標題叫做「Life in life」，這是 Phillip Bradbury 所製作的專案，他利用生命遊戲來模擬生命遊戲，這激起了我的好奇心：什麼是康威生命遊戲？這一格一格的格子卻可以做出這麼酷的東西。

研究後發現康威生命遊戲雖然規則簡單，卻可以做出邏輯閘等邏輯運算的元件，而電腦就是依靠這些邏輯元件運作的。

在深入探討後我發現有不少人已經嘗試利用康威生命遊戲製作「電腦」了，於是我也想自己嘗試製作類似的東西，所以選擇模擬數位電路中的加法器以及減法器。

不過在後來的研究中，因為手動放置電路元件，還要注意很多細節，所以使用了電子設計自動化的概念，來幫我完成自動化電路布局設計。

二、文獻探討

（一）細胞自動機──康威生命遊戲　（圖一至圖四）

「Life」是一款在 1970 年由劍橋大學數學家約翰·康威（John Conway）發明的單人遊戲，又稱為「The Game of Life」或「康威生命遊戲」。

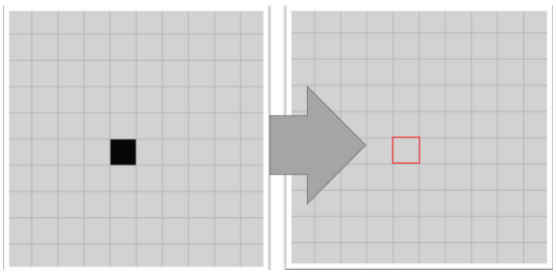
康威生命遊戲的棋盤是由一格格的小格子所組成，稱每個格子叫做細胞，細胞會有兩個狀態：存與亡。每個細胞會與周圍的八個細胞互動，有四個規則：

1.以該細胞為中心周圍少於兩個（不包含兩個）細胞時，該細胞會死亡（模擬生命數量太少，不足生存）

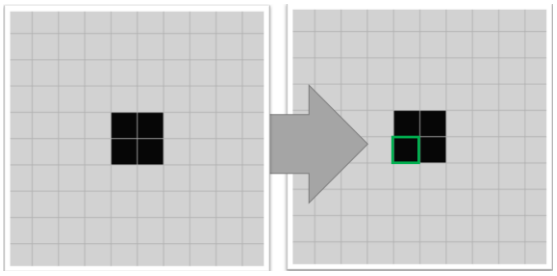
2.當周圍有二到三個細胞時，細胞保持不變

3.當周圍細胞超過三個時，該中心細胞死亡（模擬族群過大，資源無法負擔）

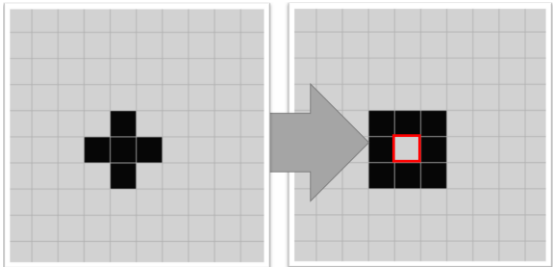
4.當一個死亡的細胞周圍有三個細胞存活時，該細胞變成存活（模擬繁殖）



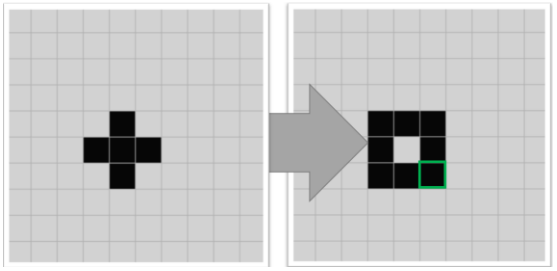
圖一　群體太小



圖二　維持不變



圖三　群體太大



圖四　誕生

圖片來源：研究者自行製作

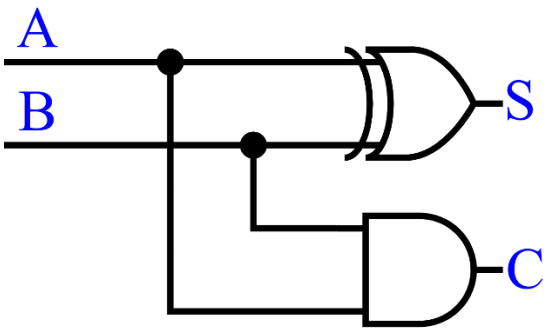
（二）電腦計算加法

1.半加器

在二進制的規則下可以利用相加的規則轉為邏輯閘實現。先令兩個一位數 A 和 B，輸出的結果本位叫做變量位，進位輸出的叫做進位位，以下以 S（sum）和 Cout（carry）代稱。

如果 A 或 B 其中為 1，另一個為 0 則相加為 01，S 為 1，C 為 0；如果 A 和 B 都是 1 則 S 為 0，C 為 1；再來 A 和 B 都是 0 的情況下，S 和 C 都為 0。

S 的真值表剛好對應到了互斥或閘（XOR gate），而 C 的真值表對應的是及閘（AND gate），所以電路圖可以設計成下圖。

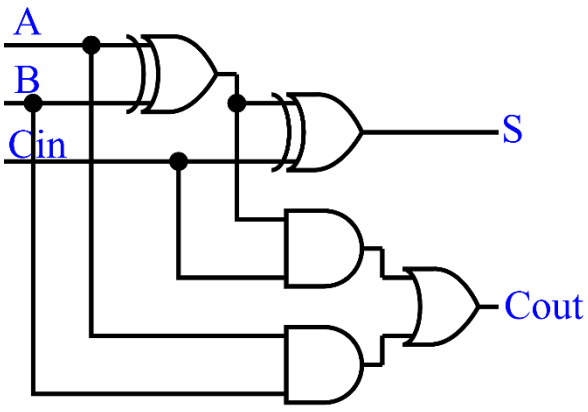


圖五　半加器電路圖（圖片來源：研究者自行繪製）

2.全加器

一個半加器只能計算單位數，且不能計算進位。為了解決進位問題需要再設計一種加法器能計算進位，稱為全加器。

其實全加器也就是比半加器多加一個數字，所以可以先給 A、B 做一次互斥或，再和進位的 Cin 加一次，即可獲得 S；不過在 Cout 這部分就比較複雜了，在三個數相加時，只有大於（含）兩個數才會進位，所以可以讓 A、B 以及 B、Cin 做一次及閘，再讓兩者的輸出做一次或閘，就可得到下圖。



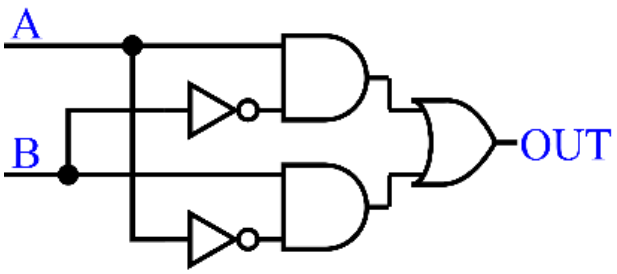
圖六　全加器電路圖（圖片來源：研究者自行繪製）

（三）互斥或閘

因為在邏輯閘的設計中並沒有互斥或閘的設計，所以如果要使用僅有的三種邏輯閘（及閘、或閘和反向器）來組成互斥或閘。

藉由觀察互斥或閘的真值表，互斥或閘輸出 1 的情況為 A!=B 的時候，也就式 A 和 B 分別為 10 或 01 的時候會輸出 1。所以如果式在 10 的情況下要輸出 1 但是 11 或 00 輸出 0 的話，只需要將反向後的 B 和 A 作和即可，01 的情況相反。而 10 和 01 只

要其中一個成立，輸出 1，所以最後再做一個或閘。電路圖如圖七。



圖七　互斥或閘的電路圖（圖片來源：研究者自行繪製）

（四）電腦計算減法

1.減法的轉換

減法其實可以看成正數加負數，例如 2-3 其實可以看成 2+(-3)，以此方式就可以用加法器來計算減法。

2.讓電腦表達負數

為了讓電腦表示的負數可以參與計算，所以就需要用到補碼表式法。補碼表式法的運作原理就是利用相加後溢位，回到 0，例如 1010 加上 0110 會等於 10000 但第五位的 1 溢位了，所以實際的數為 0000。則 0110 為-5 的補碼表式法。補碼表式法的好處就是可以實際參與運算。

3.把正數轉換成負數

而要取得一個數的補碼也很簡單，通過觀察後可以發現補碼就是原數取反再加一（1010 位元分別取反成 0101 再加 0001 成 0110），在電路上的設計就是位元分別先取反後再放入加法器中加上 1 即可。

4.減法器的設計

取得負數後只要與正數相加即可完成加減。所以在電路上需要先將 B 的每個數取反加 1 之後再與 A 數相加，整個過程可以在加法器的基礎上完成。

（五）訊號的來源

在生命遊戲中主要有兩個結構應用在邏輯電路中，分別是作為訊號的滑翔機和製造訊號的高斯帕機槍。

1.滑翔機

滑翔機是一個具有週期震盪的元件，他會以四週期為循環，每一週期結束後會往 45 度角方向前進一格（圖八），因為它會週期性的向固定的方向移動，所以被用於模擬電子訊號。

2.高斯帕機槍

高斯帕機槍也屬於週期震盪的元件，他會以每三十週期生成一個滑翔機（圖九）。因他以週期性的生成滑翔機，所以被用來做為訊號源，用來輸出高電位訊號。

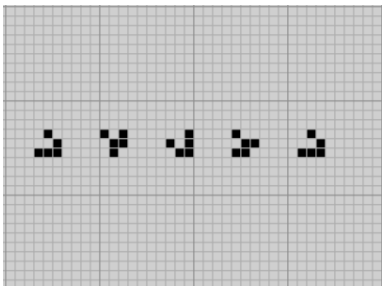
（六）訊號相交

在電路中如果有線路交匯的部分，在現實中會以多層 PCB 或是以絕緣的方式避免兩條電路干擾，不過在這個二維的世界中就要用其他方式處理了。

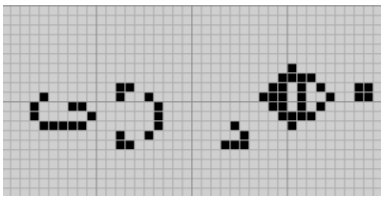
原本高斯帕機槍會以三十週期製造滑翔機，不過這樣造成的結果就是訊號太密，無法相交。而如果滑翔機以錯誤的方式相撞會發生不可控的情況，導致電路損壞。所以將原本的三十週期一個訊號變成六十週期，使其可以讓訊號交錯（圖十）。

（七）六十週期的訊號源

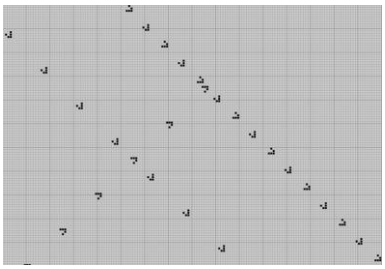
可以藉由正確的方式讓兩個高斯帕機槍的滑翔機互撞，並在原地留下一個穩定狀態的結構，讓下一個三十週期後的滑翔機撞到，從而達到將訊號密度減半的效果。（如圖十一、十二、十三）



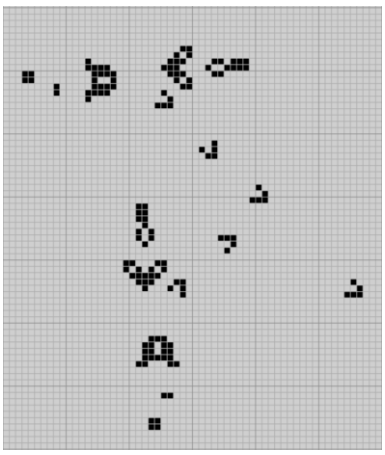
圖八　4 週期的滑翔機



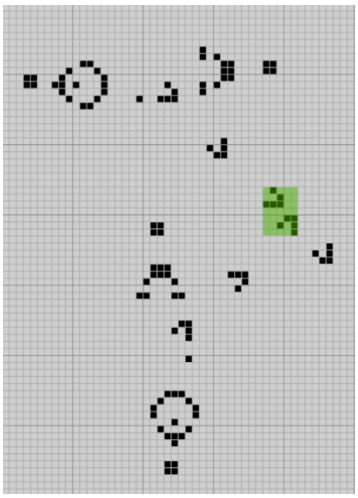
圖九　高斯帕機槍



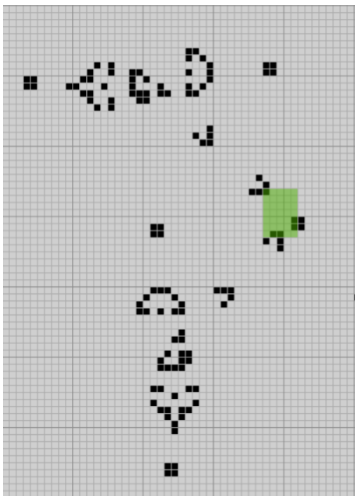
圖十　相交的訊號



圖十一　週期的機槍



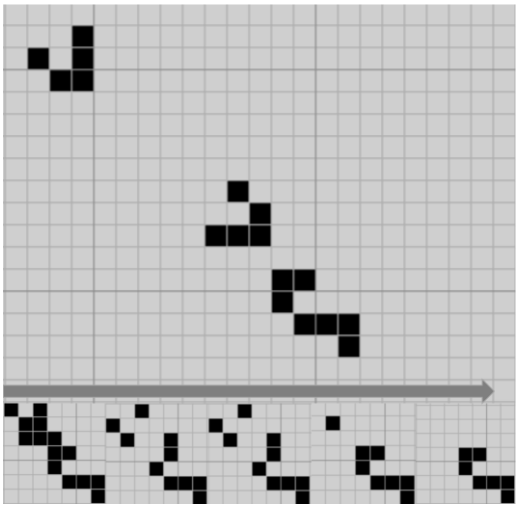
圖十二　正確的撞擊（前）



圖十三　正確的撞擊（後）

（八）訊號的終點

因為滑翔機會無限的循環移動，如果沒有適當的中止這些訊號，導致滑翔機撞擊到其他結構，一樣會使整個電路損壞，所以要用到叫做 eater 的結構（圖十四。下方為撞擊後的每週期變化）。



圖十四　eater

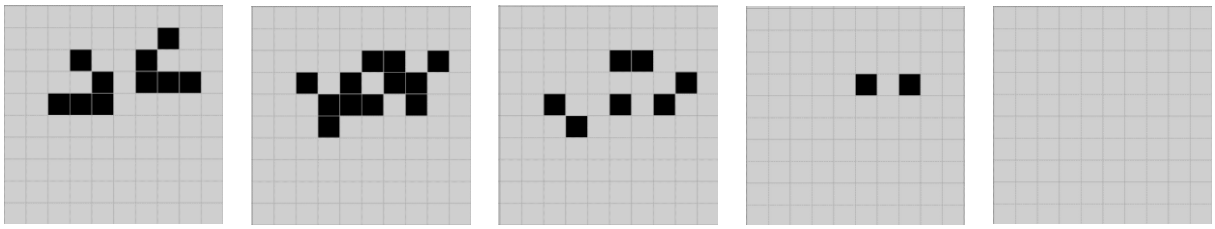
（九）模擬電路中需要注意的事項

若滑翔機以錯誤的方式撞擊會導致電路元件損壞，所以統一整個系統的滑翔機為同狀態（週期）是有必要的。而六十週期的訊號源剛好符合這個要素，因為滑翔機的震盪週期為四單位時間，而六十為四的倍數。

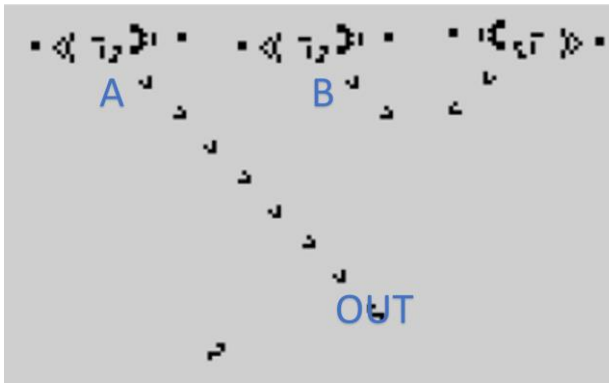
統一週期後滑翔機的之間的間距也會是相同的倍數，倍數就是每六十週期滑翔機的移動距離。這個現象可以在幾乎所有成功模擬電路的案例中發現。

（十）如何在康威生命遊戲中模擬邏輯閘

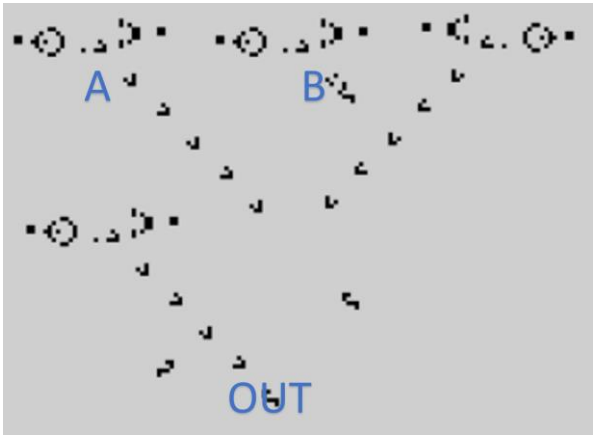
可以利用正確的撞擊（圖十五）消除或是改變電路的週期，以消除為例，可以以下面這些邏輯閘為例（圖十六、十七、十八）：



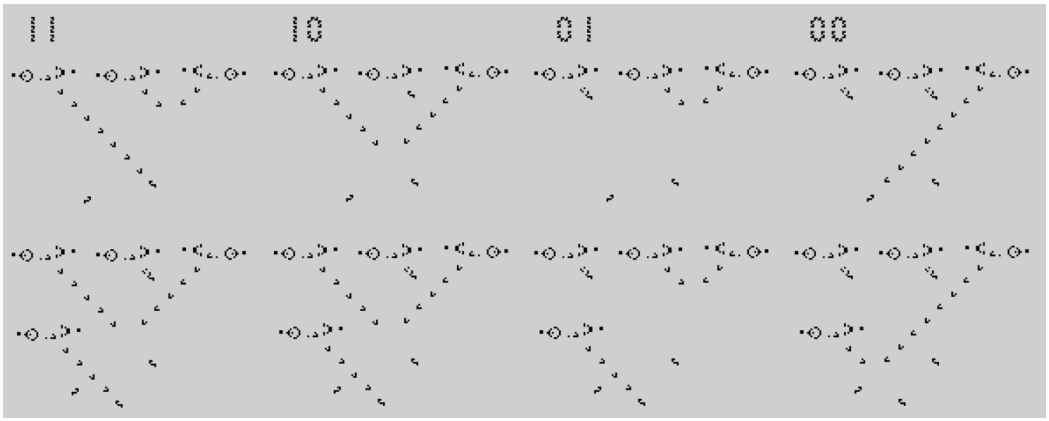
圖十五　正確的撞擊方式後的變化



圖十六　及閘



圖十七　或閘



圖十八　不同輸入下的及閘以及或閘

3.及閘

可以從上圖觀察到當 B 先攔截的右側的滑翔機 A 才能輸出，而如果 A 或 B 其中之一沒有輸出，則整個元件就不會輸出。

4.或閘

可從上圖觀察到只要 A 或 B 其中一個有輸入就可以攔截右側的滑翔機，讓下方的滑翔機可以輸出，所以當 A 和 B 都沒有輸入時，整個元件就不會輸出。

（十一）電路的其他元件

5.訊號複製器

在電路圖中也可以發現有些地方訊號是相連的，會將電流訊號分到另一端的地方，在現實中可以直接讓線路相連即可，可是在生命遊戲中，沒有特殊觸發是不會讓一個滑翔機一分為二的，這時候就需要「訊號複製器」這個元件。

6.轉向器

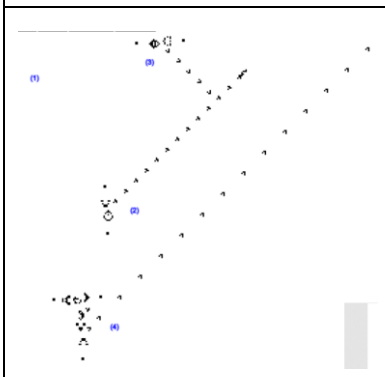
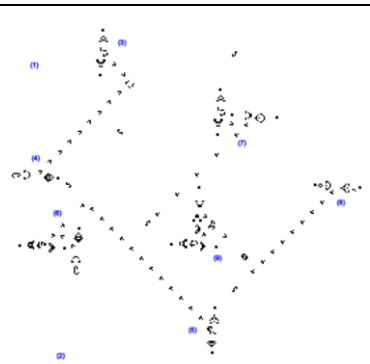
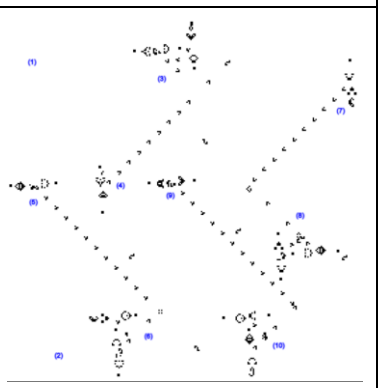
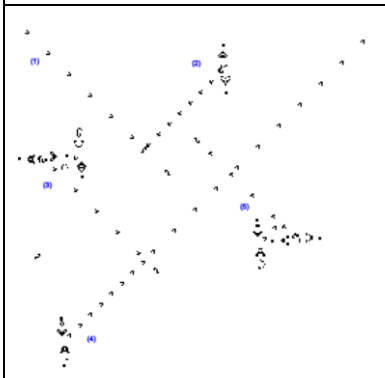
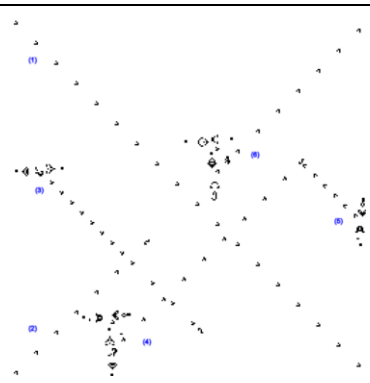
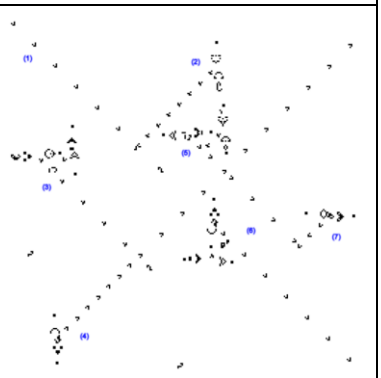
在電路中如果需要連接訊號，現實中會使用電線，如果需要轉向直接彎折線路即可，可是在生命遊戲中需要其他的元件輔助才能使滑翔機轉向。在部分實作中會使用到 queen bee 這個結構它可以使滑翔機轉向，但是有一個缺點就是它會使滑翔機的週期延遲，輸入和輸出後的週期不一。這就不符合統一週期這個要素，所以需要另外尋找轉向器的設計來達成可以轉向同時又能維持週期。

7.相交器

雖然使用六十週期的訊號密度可以讓兩條訊號相交，可是相交的前提是兩條訊號分別錯位，但是這樣就不符合距離為相同單位，所以需要用到一個元件可以暫時的讓訊號錯位，通過相交的點後再回復成正常的位置。

（十二）元件的選用

綜合要素後，我使用了 Nicholas Carlini 在 2020 年的設計作為我這次研究邏輯電路中元件。他的設計解決了上面提到的兩點，並且也有訊號複製器、轉向器和相交器。(表一)

反向器	及閘	或閘
		
轉向器	相交器	訊號複製器
		

表一 電路元件　來自 Nicholas Carlini 部落格文章

（十三）模擬康威生命遊戲

Golly 是一個用來模擬細胞自動機的開源軟體，主要作者是 Andrew Trevorrow 和 Tom Rokicki，其餘貢獻者為：Chris Rowett、Tim Hutton、Dave Greene、Jason Summers、Maks Verver、Robert Munafo、Brenton Bostick 和 Dongook Lee。在 Golly 中提供了可以使用 Python 和 Lua 作為腳本來完成一些動作。

（十四）在 Golly 上設定細胞狀態

因為要實現自動放置，所以需要類似腳本的功能。在 Golly 這個模擬器中，他支援使用 Python 和 Lua 這兩種程式語言作為腳本運行。在這次實作中會用到下面功能。

Python 使用前會先導入官方函式庫，以訪問函式庫接口的方式操作 Golly。以下將 golly 函式庫簡化為 g（import golly as g）。

1.Golly 中紀錄細胞狀態的資料類型

在 Golly 中，如果函式會返回一個區域的細胞狀態，則他會返回一個一維列，陣列中的每兩個數為一個座標，代表這個座標中存在活的細胞。

2.設置細胞狀態

可以使用 g.setcell(x, y, state)來設定畫布中(x, y)位置的狀態，如果 state 為 1 則該格設定為活細胞，反之亦然。

3.取得指定圍內的細胞結構

使用 cellList = g.getcells(x, y, dx dy])返回的值會賦予到變數 cellList，是第一點提到的細胞列表；其中 x 和 y 為左上角的座標，dx 和 dy 分別代表選取範圍的寬和高。

4.放置結構

放置結構的方式是呼叫 g.putcells(cell_list, x, y)函式，cell_list 為細胞列表資料，x 和 y 為放置的左上角。

（十五）電子設計自動化

電子設計自動化（Electronic Design Automation，EDA）是一項專門用於協助設計和製造電子系統的技術和工具集合。它涵蓋了從電子電路的設計、驗證、模擬到佈局和佈線等各個設計階段的自動化過程。

EDA 在現代電子設計中扮演著重要的角色，使得設計師能夠更加高效地開發和製造各種複雜的電子產品。

貳、研究設備與器材

一、研究設備

（一）作業系統：Windows 10 (64 位元)

（二）CPU：1 lth Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz

（三）記憶體：16 GB

二、使用軟體

（一）模擬康威生命遊戲：Golly

（二）文字編輯器：Visual Studio Code

三、程式語言

（一）Python 3.11 (64bit)

參、研究目的

一、在康威生命遊戲中模擬計算機中的加法器和減法器

二、製作自動電路布局設計程式

（一）將布林函數轉成電路設計圖

（二）從電路設計圖設計電路布局

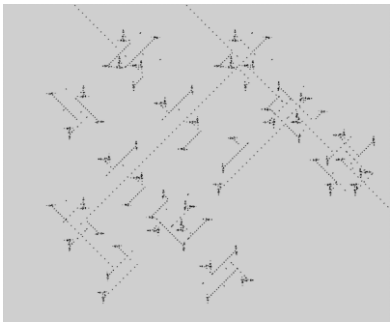
（三）依照電路布局將元件排在康威生命遊戲模擬器中

肆、研究過程及方法

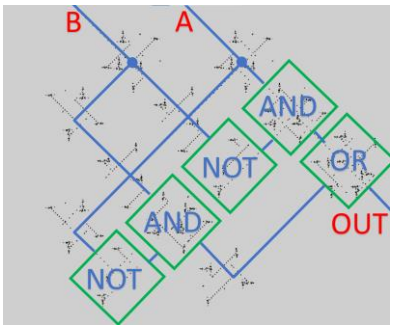
一、手動放置

（一）製作互斥或閘

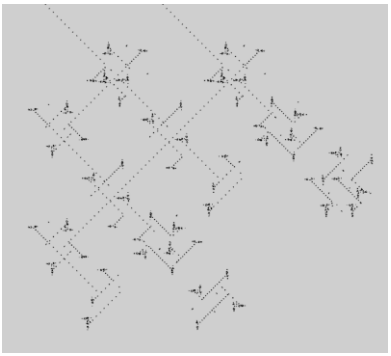
因為加法器的電路設計圖中有使用到互斥或閘，但是 Nicholas Carlini 在 2020 年的設計並沒有設計互斥或閘，所以要使用僅有的反向器、及閘以及或閘來做出互斥或閘。依照電路設計圖按圖施工即可。



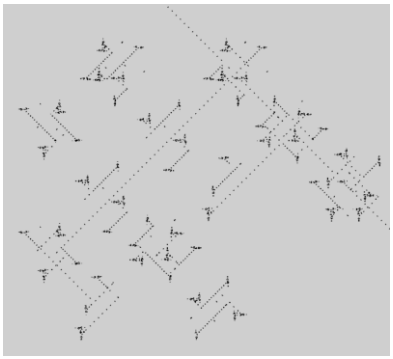
圖十九。完成的成果



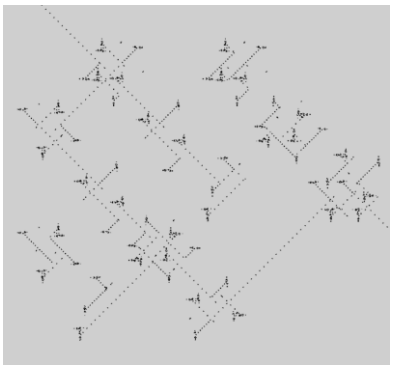
圖二十。標示邏輯閘



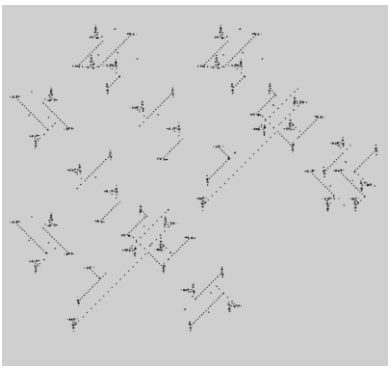
圖二十一。A1 B1



圖二十二。A1 B0



圖二十三。A0 B1



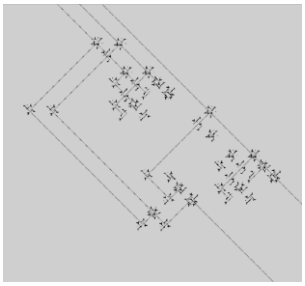
圖二十四。A0 B0

（二）製作加法器

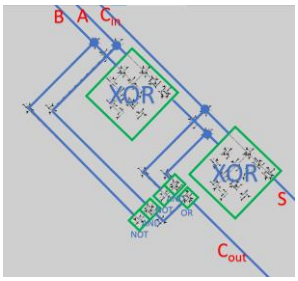
有了互斥或閘之後就可以來做加法器了，不過因為之後要串接成更高位的加法

器，所以直接做全加器，依照全加器的電路設計圖，在模擬器中放置電路元件。

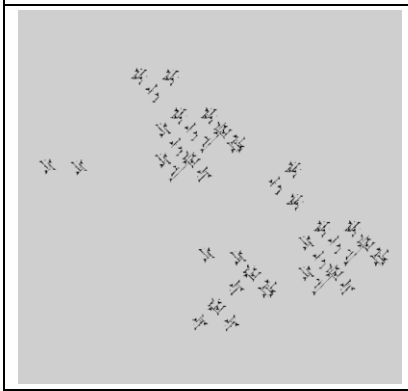
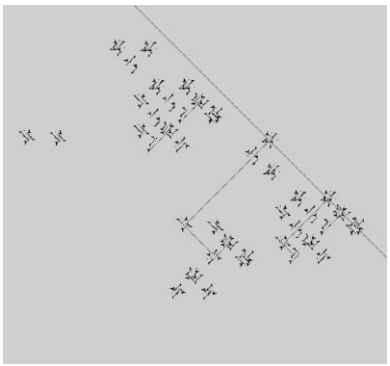
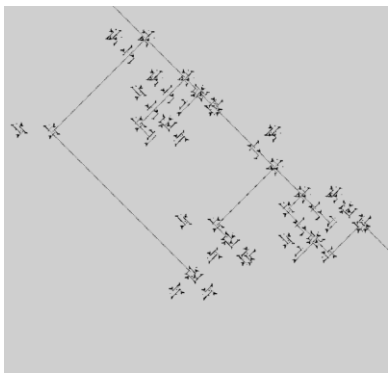
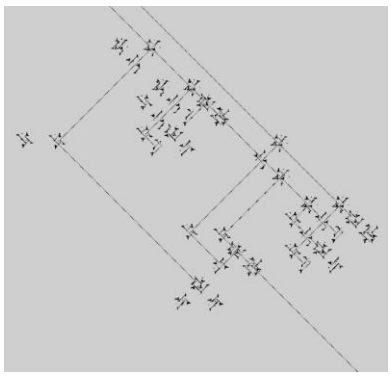
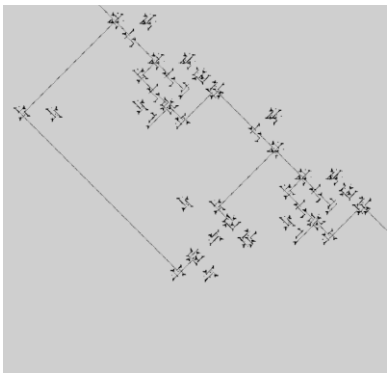
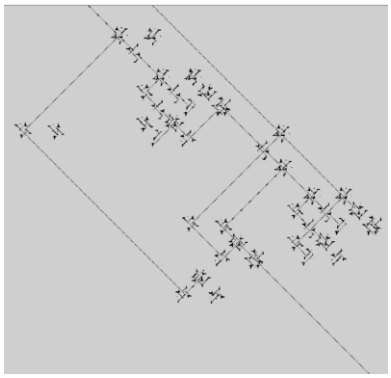
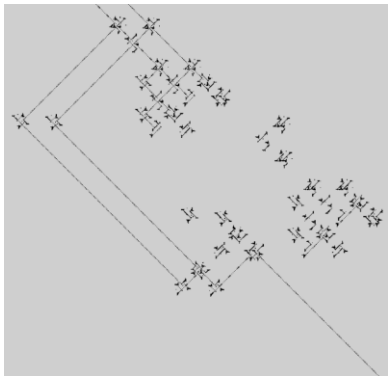
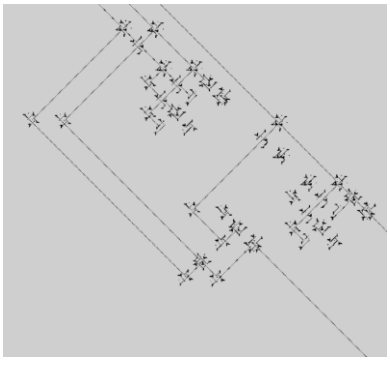
我將 C_{in} 坐在右側，這樣比較方面後面的串接。完成圖如圖二十五，標示邏輯閘如圖二十六。做完之後對這個加法器進行測試，驗證真值表是否符合預期(表二)。



圖二十五　完成圖

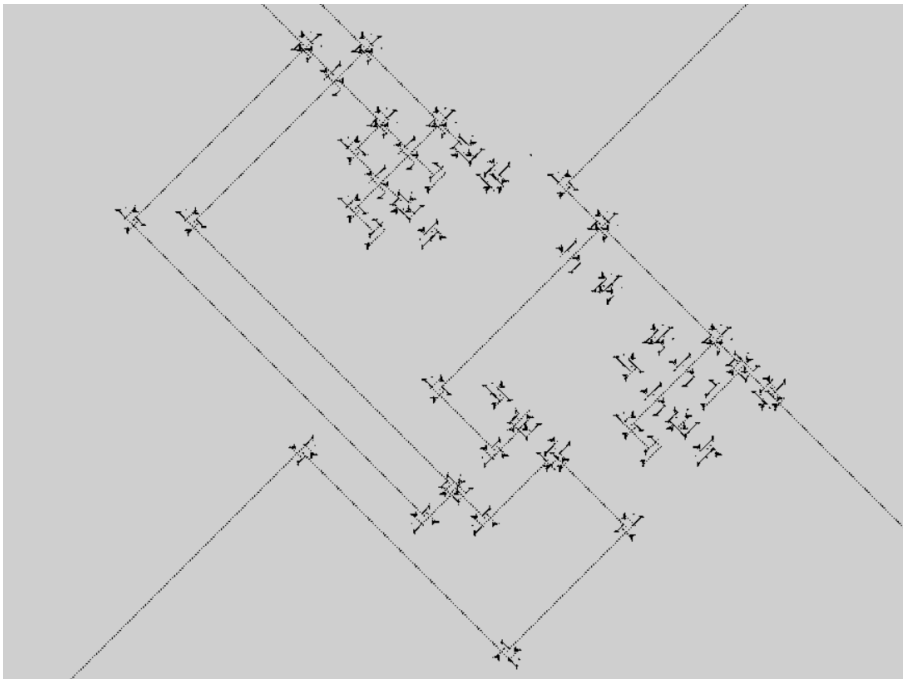


圖二十六　標上元件

0+0+0=00	0+0+1=01	0+1+0=01
		
0+1+1=10	1+0+0=01	1+0+1=10
		
1+1+0=10	1+1+1=11	
		

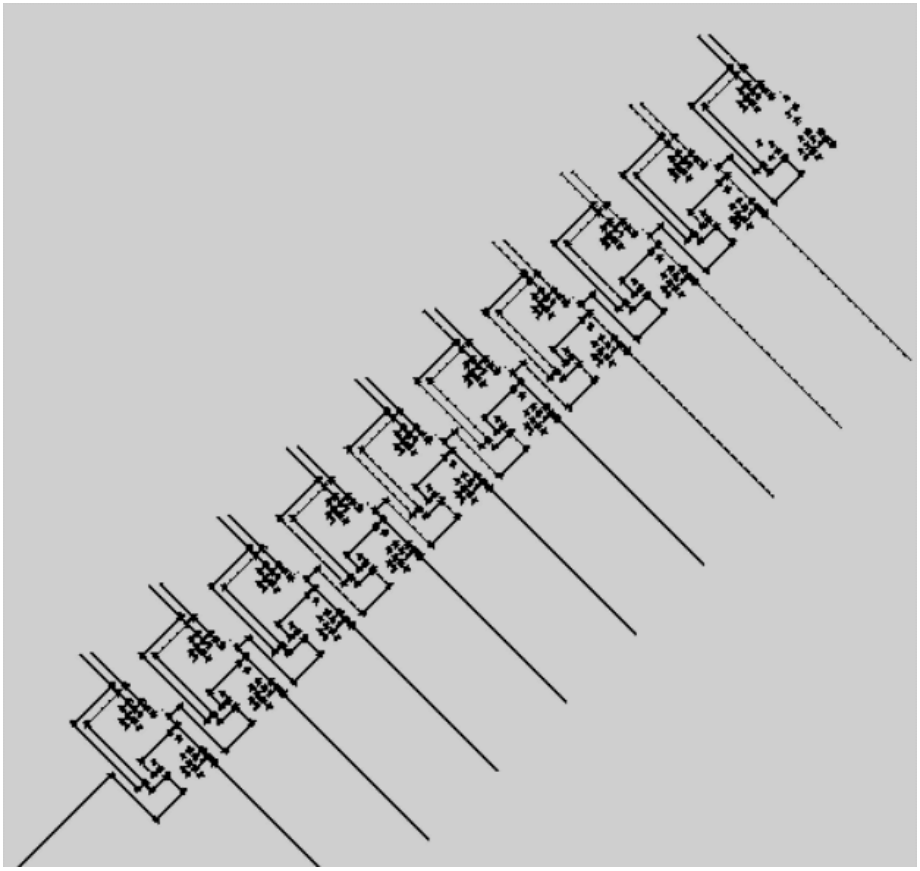
表二　所有輸入下的加法器

為了方便之後串接所以我將 C_{in} 和 C_{ou} 轉到側邊（圖二十七）



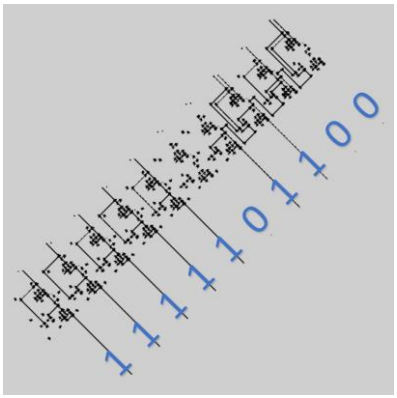
圖二十七　模組化的加法器

有了模組化的加法器之後就可以來串接多個加法器運算更多位的數字了，我以十位元為例，所以我串十個加法器如圖二十八。



圖二十八　十位元加法器

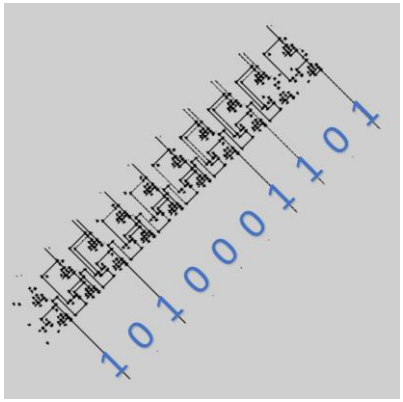
做完之後算看看計算結果是否正確（圖二十九和圖三十）。



圖二十九

679 + 325 = 1004

1010100111+010100 0101 = 1111101100



圖三十

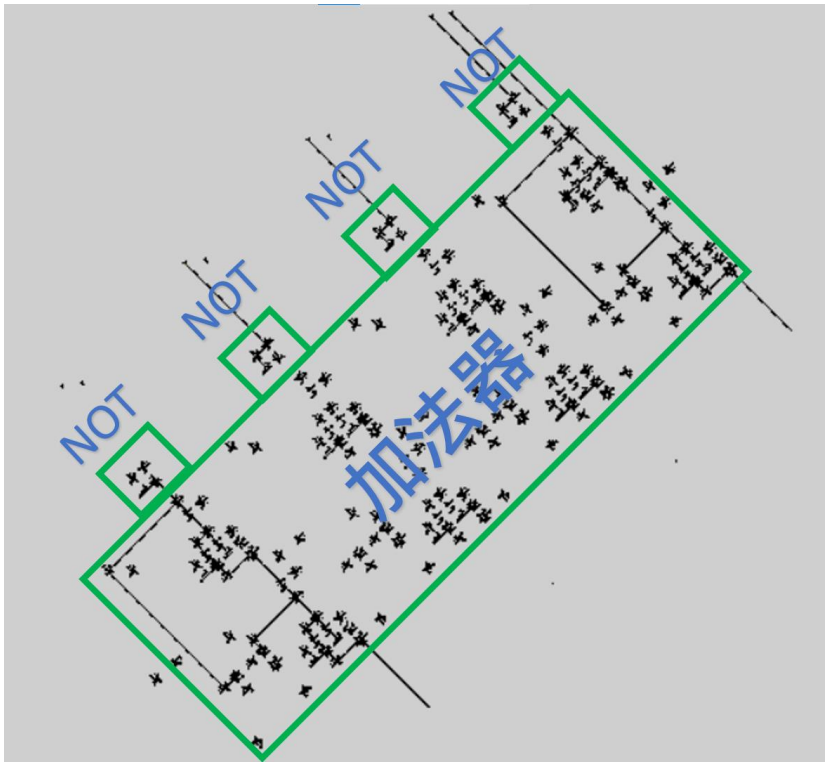
238 + 415 = 890

0011101110+ 0110011111= 1010001101

可以看出加法器運算是正確的，到此為止加法器就做完了。

（三）製作減法器

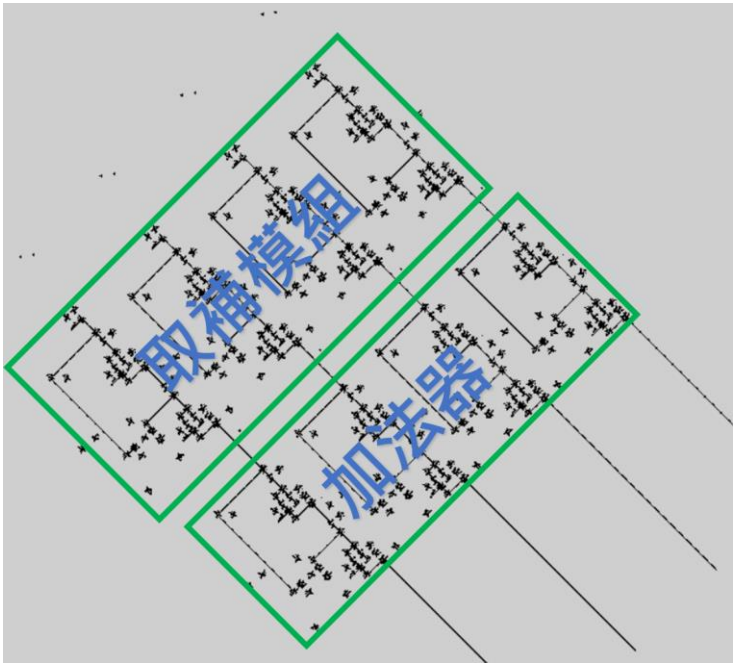
計算減法的方式是先取得補數，再與另一數相加。而取得補數的方式就是取反再加一。依照這個原理我們就可以先做出求補數的模組（圖三十一），再將模組與加法器相連即可。因為減法器至少需要兩位才看的出來效果，所以以下以四位試做。



圖三十一　取補數的模組

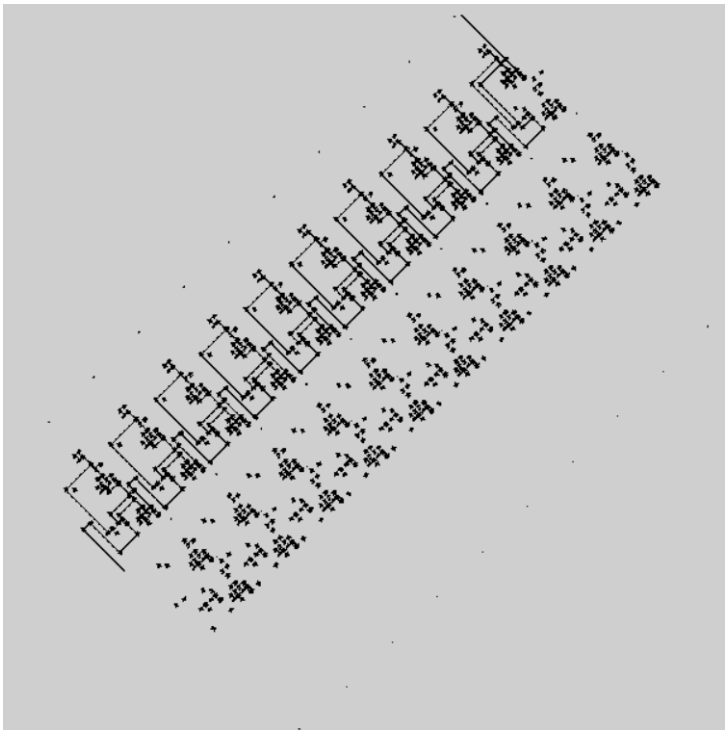
有了取補數的模組之後，直接接上加法器即可，也因為取補數的模組是加法器輸

出，所以訊號流的間距剛好可以接上新的加法器。在取補數的模組下再加上加法器就做出減法器了（圖三十二）。



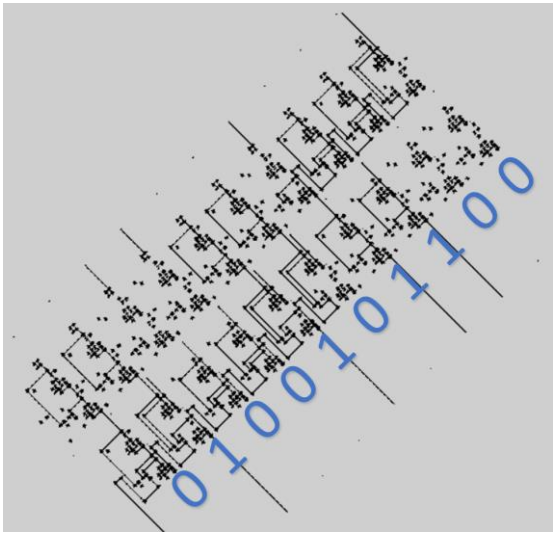
圖三十二　四位元減法器

最後如果要做出十位元的減法器只要串十個即可，在這個設計中是有預留進位輸入跟輸出的，所以直接接上即可。完成如圖三十三。



圖三十三　十位元減法器

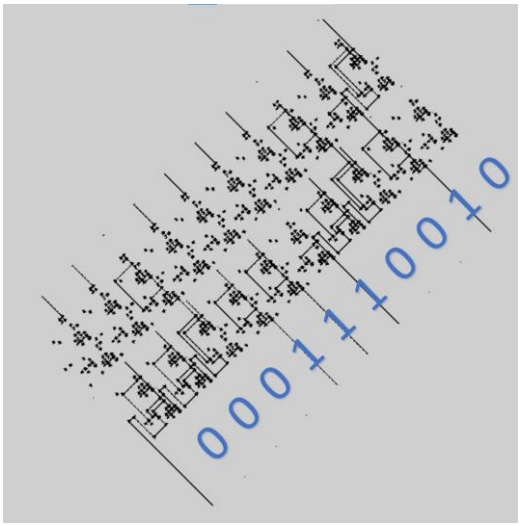
做完之後算看看計算是否正確（圖三十四和圖三十五）。



圖三十四

500 - 200 = 300

0111110100+0011001000 = 0100101100



圖三十五

1004 - 890 = 114

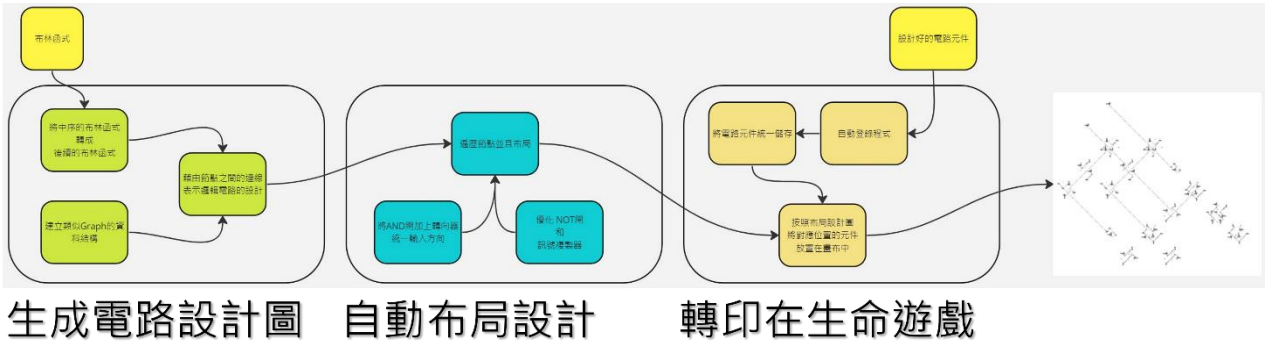
1111101100+0110011111= 0001110010

可以看出減法器運算是正確的，到此為止減法器就做完了。

二、自動電路布局設計

前面雖然成功做出了加法器和減法器，但是其電路元件全部都是採取手動放置來進行。由於系統對於放置的位置非常要求，一定要放在正確的位置，就算只是偏差一小格，也會讓整個系統報銷，這點使得放置電路元件的過程變得非常麻煩，所以我開始思考並著手研究：如何可以使系統直接生成我要的電路圖，並且直接完成電路元件的擺放設定。

而指定電路圖的方式我使用布林函式的方式，只要輸入布林函式就可以生成相對應的邏輯電路。為了實現這個功能我分三部份製作。



（一）使用布林函式生成電路設計圖

1.電路設計圖我使用 Python 的 class 製作了一個類似 graph 的資料類型，在這個 circuit_diagram 中，有一個 self.nodes 的變數，用來儲存每個節點的資料。而每個節點我分別又為了他們各做出新的 class 用來儲存每個節點的資料，class 如下：

```
class circuit_input:
    def __init__(self, serial_number, name=None):
        self.type = 'INPUT'
        self.O = []
        self.serial_number = serial_number
        self.name = name
```

圖三十六　circuit_input

```
class circuit_output:
    def __init__(self, serial_number, name=None):
        self.type = 'OUTPUT'
        self.A = None
        self.serial_number = serial_number
        self.name = name
```

圖三十七　circuit_output

```
class AND:
    def __init__(self, serial_number):
        self.type = 'AND'
        self.B = None
        self.A = None
        self.O = []
        self.serial_number = serial_number
```

圖三十八　AND

```
class OR:
    def __init__(self, serial_number):
        self.type = 'OR'
        self.A = None
        self.B = None
        self.O = []
        self.serial_number = serial_number
```

圖三十九　OR

```
class NOT:
    def __init__(self, serial_number):
        self.type = 'NOT'
        self.A = None
        self.O = []
        self.serial_number = serial_number
```

圖四十　NOT

各電路元件的 class 程式碼

在想怎麼從布林函式然後連接元件的時候，我想到可以把邏輯運算子看成是一般的運算符號，而如果電腦要計算一個算式的話就會先把中序運算式（infix）轉成後續運算式（postfix），再用後續運算式進行計算。所以我也先將布林函式轉成後續運算式，再用跟四則運算相似的方式去連接元件節點跟節點之間的邊。

```
def infix2postfix(infix):
    weight = {"(": 0, "&": 1, "|": 1, "!": 2}
    stack = []
    postfix = ""
    temp = ""

    for i in infix:
        if i != " ":
            if (i not in weight) and (i != ")"):
                temp += i
            else:
                if i in "&|!":
                    while stack and weight[stack[-1]] >= weight[i]:
                        postfix += " " + stack.pop()
                    stack.append(i)
                elif i == "(":
                    stack.append(i)
                elif i == ")":
                    while stack[-1] != "(":
                        postfix += " " + stack.pop()
                    stack.pop()

        elif i == " " and temp != "":
            postfix += " " + temp
            temp = ""

    return (postfix + " " + temp + " " + " ".join(stack[::-1]))
```

圖四十一　中序運算的布林函式轉後續運算式的程式碼

```
def postfix2circuit(postfix: str, circuit: circuit_diagram):
    stack = []
    for i in postfix.split():
        if i == "&":
            B = stack.pop()
            A = stack.pop()
            ser = circuit.addNode('AND')
            circuit.addEdge(A, ser, 'AND-A')
            circuit.addEdge(B, ser, 'AND-B')
            stack.append(ser)
        elif i == "|":
            B = stack.pop()
            A = stack.pop()
            ser = circuit.addNode('OR')
            circuit.addEdge(A, ser, 'OR-A')
            circuit.addEdge(B, ser, 'OR-B')
            stack.append(ser)
        elif i == "!":
            A = stack.pop()
            ser = circuit.addNode('NOT')
            circuit.addEdge(A, ser, 'NOT')
            stack.append(ser)
        else:
            ser = circuit.getNodeSerNoByName(i)
            if (ser):
                stack.append(ser)
            else:
                ser = circuit.addNode('INPUT', i)
                stack.append(ser)

    return [circuit, stack.pop()]

    # 註
    # circuit.addNode(type: str, *name: str) 用來新增節點 如果是 INPUT 或 OUTPUT 要加上 name
    # circuit.addEdge(ser1: int, ser2: int, port: str) 用來新增邊 ser1 和 ser2 是節點的編號 port 是連結的端點
```

圖四十二　輸入後序運算式設計電路圖的程式碼

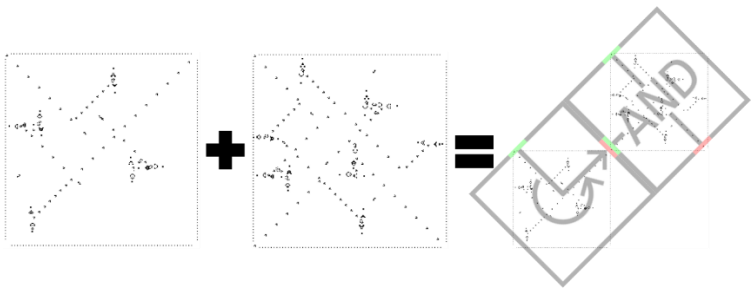
（二）規劃元件位置

目前已經生成了電路圖，接下來只要用深度優先搜索（Deep Fitst Search）的方式從輸出端往回遍歷整個 graph 找到輸入端，並且在遍歷的過程放置元件就可以完成布局。

我利用一個二維整數陣列來表示布局的結果，用代號在表示相對位置的元件，不過為了方便陣列的操作，陣列中表式的位置會和之後傳印的結果水平鏡像。另外原本電路走向看起來是斜的，為方便討論，以下將左上方定為上方，其他以此類推。

1.調整邏輯閘形狀

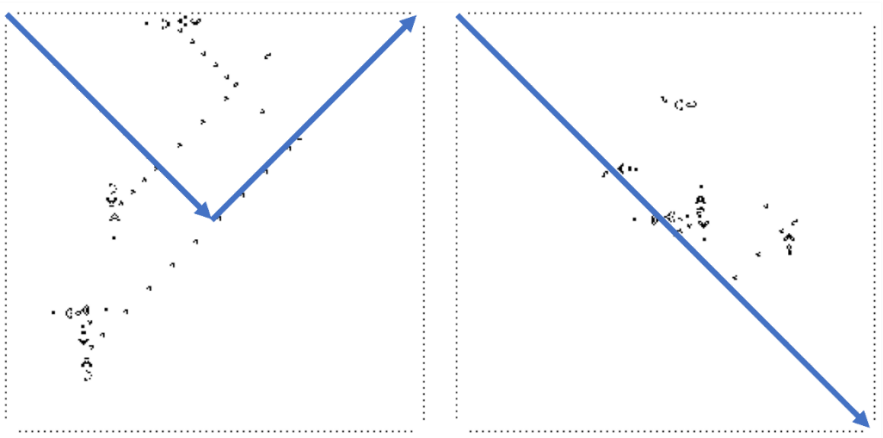
原本及閘和或閘的設計是由上方和左方輸、下方輸出，可是這個輸入的方向不利於自動化電路布局設計，所以將一個由上轉右的逆時針轉向器套在及閘和或閘的左側輸入端，讓兩個輸入並行且同方向輸入。



圖四十三　加上轉向器的及閘

而因為我希望反向器可以做到上進下出，可是如果要用轉向器完成這件式的話會變成一個 2×2 的單元，體積太大反而不利自動電路布局設計，所以我自行設計了新的反向器。新的反向器用到了兩個 queen bee 結構，讓輸入的訊號繞到高斯帕機槍前攔截訊號，達成反向器效果。

圖四十四（左：作者原設計。右：重新設計的反向器）



圖四十四

2.游標布局

在自動電路布局設計的時候會有一個游標代表這個元件的生成位置，藉由移動游標來生成元件。游標由兩個變數 x 和 y 組成，分別對應二維陣列中的列和行。游標的移動有幾個規則：

（1）往下遍歷的時候 x 座標加二

（2）當遞迴結束 x 座標回到上一層的位置

（3）當遍歷到輸入端時返回遍歷並且 y 加一

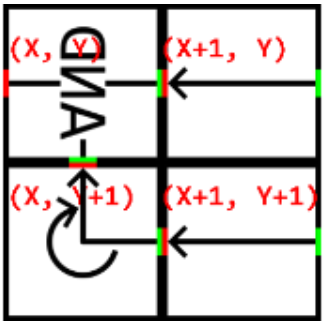
游標會移動後就是生成元件了，生成元件也有幾個規則（圖四十五）：

（1）元件本身生成在(x, y)的位置

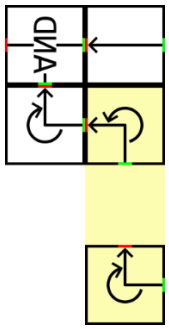
（2）在(x+1, y)和(x+1, y+1)的地方生成一條線（給訊號預留的位置）

（3）如果是及閘和或閘的話需要在(x, y+1)的地方生成轉向器

目前這樣還是有一點問題，因為 y 會變化，但是訊號只會往一個方向走，所以需要一個「橋」來幫忙串接垂直的訊號，橋是由兩個轉向器構成，類似淺望鏡的效果。



圖四十五



圖四十六

```
def DFS (CD: circuit_diagram, v:int, visited:list[int]):
    global scr, x, y
    visited[v] = True

    visitType = CD.getNodeBySerNo(v).type
    if visitType == 'INPUT':
        scr = addElement(scr, x, y, CD.getNodeBySerNo(v).name) # 放置輸入元件
        y += 1
        return
    elif visitType == 'OUTPUT':
        scr = addElement(scr, x, y, 5) # 放置輸出元件
        x += 1 # 往右移動
        scr = addElement(scr, x, y, 8) # 放置線
        x += 1 # 往右移動(下一個元件的位置)

        DFS(CD, CD.getNodeBySerNo(v).A, visited) # 繼續往下走
        return
    elif visitType in ["AND", "OR"]:
        onX = x # 紀錄這個元件的x座標
        onY = y+1 # 紀錄0的高度

        if visitType == "AND":
            scr = addElement(scr, x, y, 1) # 放置AND元件
        elif visitType == "OR":
            scr = addElement(scr, x, y, 2) # 放置OR元件
        scr = addElement(scr, x, y+1, 12) # 放置轉向器
        x += 1 # 往右移動
        scr = addElement(scr, x, y, 8) # 放置線
        scr = addElement(scr, x, y+1, 8) # 放置線

        x += 1 # 往右移動(下一個元件的位置)

        DFS(CD, CD.getNodeBySerNo(v).A, visited) # 繼續往下走


        if y > onY: # 如果row有變化
            scr = verticalLink(scr, onX+1, onY, y) # 垂直連接
        else:
            scr = addElement(scr, onX+1, y, 8)

        x = onX + 2 # 往右移動(下一個元件的位置)


        DFS(CD, CD.getNodeBySerNo(v).B, visited) # DFS的下一個點


        x = onX # 回到這個元件的x座標
        return
    elif visitType == "NOT":
        onX = x # 紀錄這個元件的x座標

        scr = addElement(scr, x, y, 20) # 放置NOT元件
        x += 1 # 往右移動
        scr = addElement(scr, x, y, 8) # 放置線
        x += 1 # 往右移動(下一個元件的位置)

        DFS(CD, CD.getNodeBySerNo(v).A, visited) # 繼續往下走

        x = onX # 回到這個元件的x座標
        return
```

圖四十七　搜尋並放置的主程式

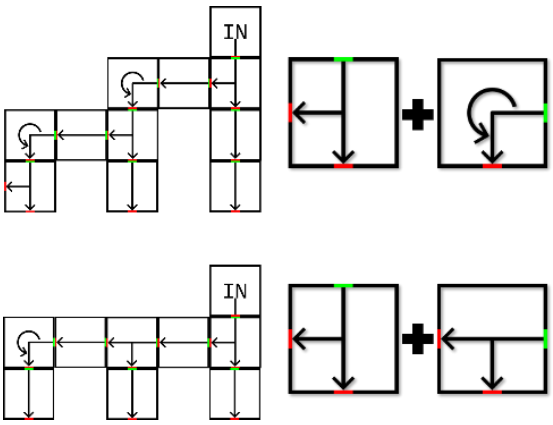
```
def verticalLink(list:list, x:int, highY:int, lowY:int):
    h = lowY - highY

    if h == 0:
        list = addElement(list, x, highY, 8)
        return list

    list = addElement(list, x, highY, 17)
    list = addElement(list, x, lowY, 12)

    for i in range(h-1):
        list = addElement(list, x, highY+i+1, 10)

    return list
```

圖四十八　建立「橋」的程式

3.連結輸入

從輸出端開始遍歷整個 graph 後會形成一個樹狀圖，在葉節點的不分式輸入端，不過這樣有可能造成同一個輸入端，可是卻不是同一個葉節點，所以要針對這點，統一輸入端，然後再利用訊號複製器分配給需要的葉節點。

為了方便自動電路布局設計（圖四十九），我修改了 Nicholas Carlini 原本的設計，做了一個鏡像並且旋轉版本的訊號複製器。



圖四十九　兩種布局方式

```
def setInput(input:list[str]):
    global scr

    scr = listInsertCol(scr, len(scr[0])+2)

    for i in range(len(input)):
        waitForShare = []
        for j in range(len(scr)):
            ipt = input[i]
            check = isInput(scr, j)
            if ipt == check[1]:
                waitForShare.append(check[2])
                # scr[check[2][1]][check[2][0]] = 0
        if len(waitForShare) == 1:
            scr = addElement(scr, len(scr[0])-1, waitForShare[0][1], 4)
            scr = addElement(scr, len(scr[0])-2, waitForShare[0][1], 8)
            scr = addElement(scr, len(scr[0])-3, waitForShare[0][1], 8)
            continue
        for j in range(len(waitForShare)):
            if j == 0:
                scr = addElement(scr, len(scr[0])-1, waitForShare[0][1], 4)
                scr = addElement(scr, len(scr[0])-2, waitForShare[0][1], 8)
                scr = addElement(scr, len(scr[0])-3, waitForShare[0][1], 6)
            else:
                scr = shareInput(scr, len(scr[0])-3, waitForShare[j-1][1], waitForShare[j][1], j == len(waitForShare)-1)

        scr = listInsertCol(scr, len(scr[0])-2)
```

圖五十　分享輸入訊號的主程式

```
def shareInput(list:list, x:int, topY:int, downY:int, end:bool):
    print("shareInput")
    h = downY - topY
    for i in range(h-1):
        list = addElement(list, x, topY+i+1, 7)

    if end:
        list = addElement(list, x, downY, 13)
    else:
        list = addElement(list, x, downY, 22)

    return list
```

圖五十一　shareInput 函式

4.優化體積

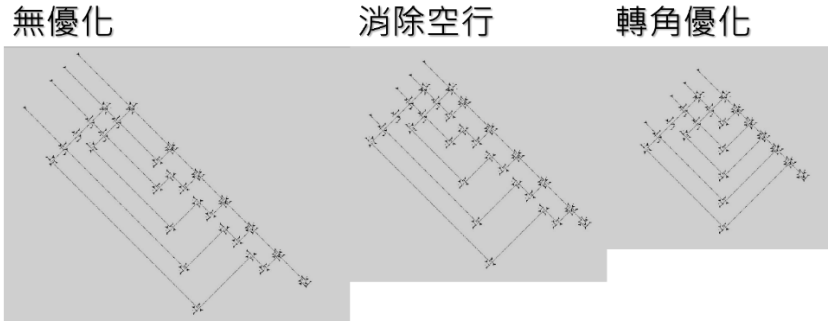
為了要簡化演算法，所以在自動電路布局設計的時候其實會產生很多不必要的空位，所以在最後再做兩個優化的功能，分別是消除空行、轉角優化。（圖五十二）

（1）消除空行

如果有一行是只存在線沒有其他元件的話，刪除那行也不會對=電路造成什麼影響。

（2）轉角優化

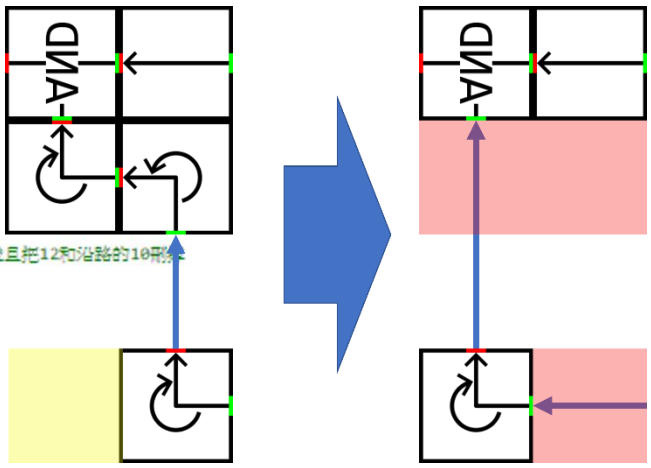
在橋的部分用了三個轉向器，形成一個 Z 字的結構，但其實可以只做對角的轉角，優化成 L 結構，然後再削去空行



圖五十二　經過優化的變化

```
def optimize():
    global scr

    # 搜尋陣列 找12
    for i in range(len(scr)):
        for j in range(len(scr[0])):
            if scr[i][j] != 17:
                continue
            # 找到12
            # 往X- 找12
            # 往y+ 找12
            # 如果都找到了 就在(第一個12的x座標, 第一個12的y座標)放置12 並且把12和沿路的10刪掉
            NW = False
            SE = False
            for x in range(j-1, -1, -1):
                if scr[i][x] == 12:
                    NW = True
                    break
            if not NW:
                continue
            for y in range(i+1, len(scr)):
                if scr[y][j] == 12:
                    SE = True
                    break
            if not SE:
                continue

            scr[i][j] = 0
            # 刪除NW
            NWx = 0
            SEy = 0
            for x in range(j-1, -1, -1):
                if scr[i][x] == 12:
                    scr[i][x] = 0
                    NWx = x
                    break
            # 刪除SE
            for y in range(i+1, len(scr)):
                if scr[y][j] == 12:
                    scr[y][j] = 0
                    SEy = y
                    break
                scr[y][j] = 0

            scr[SEy][NWx] = 12
```

轉角優化



圖五十四　轉角優化示意圖

```
# 如果遇到有一col只有0或8組成的話 就把那個col刪掉
i = 0
while i < len(scr[0]):
    flag = True
    for j in range(len(scr)):
        if scr[j][i] not in [0, 8]:
            flag = False
            break
    if flag:
        scr = listRemoveCol(scr, i)
        i -= 1

    i += 1
```

消除空行

圖五十三　優化的程式碼

5.翻轉陣列

前面為了生成布局設計的時候方便進行陣列操作，所以陣列的圖形會和到時候要轉印出來的成鏡像和 90 關係，在最終輸出前要先處理。

6.寫入檔案

最後在將生成好的設計圖寫到一個文字檔案中方便之後 golly 端可以讀取。

（三）轉印到 Golly

資料處理的部分做完了，再來是 Golly 端得部分，在這個部分中要先登錄元件的結構，之後要用的話只要讀取就能用了。登錄完後就依照先前生成好的自動電路布局設計圖，將元件放在正確的位置上。

1.登錄結構

getcells 函數可以讀取畫布上的細胞情況，而 putcells 可以將 getcells 讀到的內容再放到畫布中，所以如果用一個檔案來記錄 getcells 所記錄的元件結構，就可以藉由讀取檔案然後再用 putcells 放到特定位置。

不過在使用 getcells 時他紀錄的是活細胞的座標而不是選取範圍內的相對座標，但是讀取完的資料放到 putcells 就變成相對座標了，為了解決這個問題必須把結構的右上角對其畫布的原點，才能記錄他的結構。但是如果每一個都要手動放到原點再登入的話還是太麻煩了，所以我另外寫了一個用來登錄的程式，只要將所有元件統一排好，自動移到原點的位置並且登錄。（圖五十五）

```
import golly as g

# f = open("log.txt", "w+")
objs = open("objs.txt", "a+")

code2pos = {1: (1, 0), 4: (2, 0), 8: (3, 0), 12: (4, 0), 16: (5, 0), 20: (6, 0), # 元件編號對應的位置
            2: (1, 1), 5: (2, 1), 9: (3, 1), 13: (4, 1), 17: (5, 1), 21: (6, 1),
            3: (1, 2), 6: (2, 2), 10: (3, 2), 14: (4, 2), 18: (5, 2),
            0: (1, 3), 7: (2, 3), 11: (3, 3), 15: (4, 3), 19: (5, 3)}


for i in range(22):
    for j in range(270):
        for k in range(270):
            g.setcell(j, k, 0) # 清空(0, 0)~(269, 269)的格子

    copy = g.getcells([270*code2pos[i][0], 270*code2pos[i][1], 270, 270]) # 取得元件的格子
    g.putcells(copy, -270*code2pos[i][0], -270*code2pos[i][1]) # 貼上元件


    obj = g.getcells([0, 0, 270, 270]) # 取得元件的格子
    print(obj, file=objs) # 寫入檔案
```

圖五十五　登入元件結構的程式碼

2.放置元件

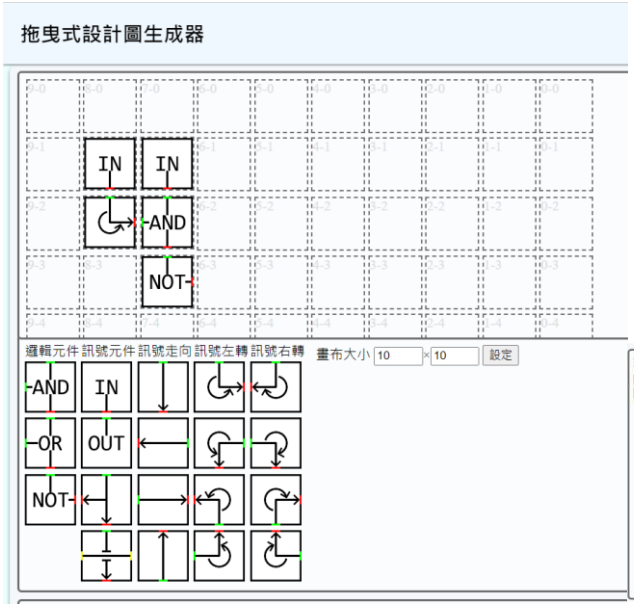
從先前寫好的自動電路布局設計圖的檔案中讀出二維陣列，依照相應座標放置相應的元件，就完成了。（圖五十六）

```
import golly as g

objs = open("objs.txt", "r")
log = open("log.txt", "w+")
draw = list(eval(open("draw.txt", "r").readline()))

objects = []

for i in range(22):
    objects.append(eval(objs.readline()))

for i in range(len(draw)):
    for j in range(len(draw[i])):
        posx = 270*i - 270*j
        posy = 270*j + 270*i

        g.putcells(objects[draw[i][j]], posx, posy)
```

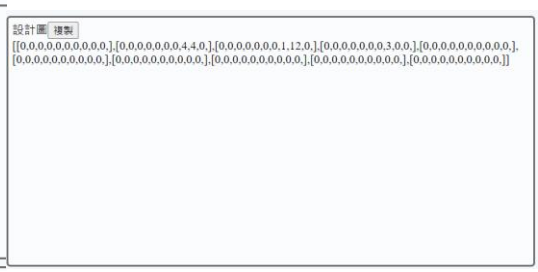
圖五十六　放置元件的程式碼

（四）補足自動設計的工具

在有些情況下有可能自動設計不一定會有滿意的設計（例如可能空間使用率不高），所以我又另外用 HTML、CSS、Javascript 簡單做了一個工具，可以透過拖曳的方式設計電路圖，並且生成電路圖的「自動電路布局設計圖」，可以搭配先前製作的放置腳本自動在模擬器中排好。



圖五十七　生成器畫面



圖五十八　生成好的設計圖

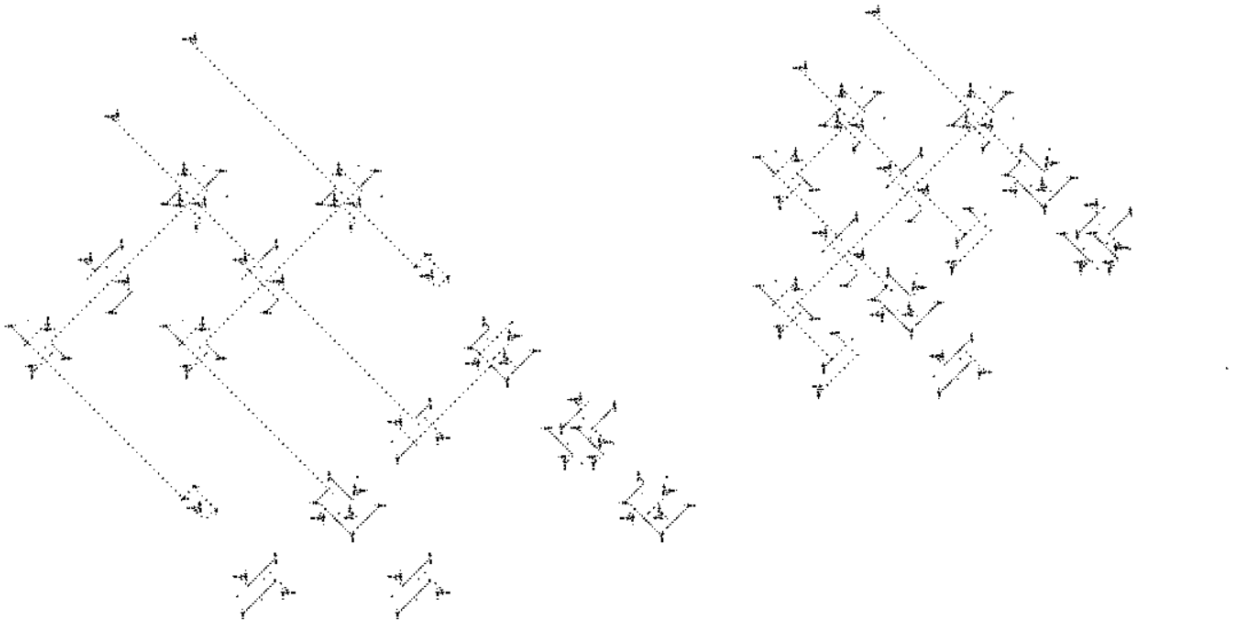
伍、研究結果

在這次的研究中我利用反向器、及閘和或閘做出了互斥或閘，再利用互斥或閘製作出一個加法器的模組，以此模組串接成十位元的加法器，可以計算兩個 1023 以內的加法。之後用補數以及加法器為基礎，做出可以計算兩個 1023 以內的減法。

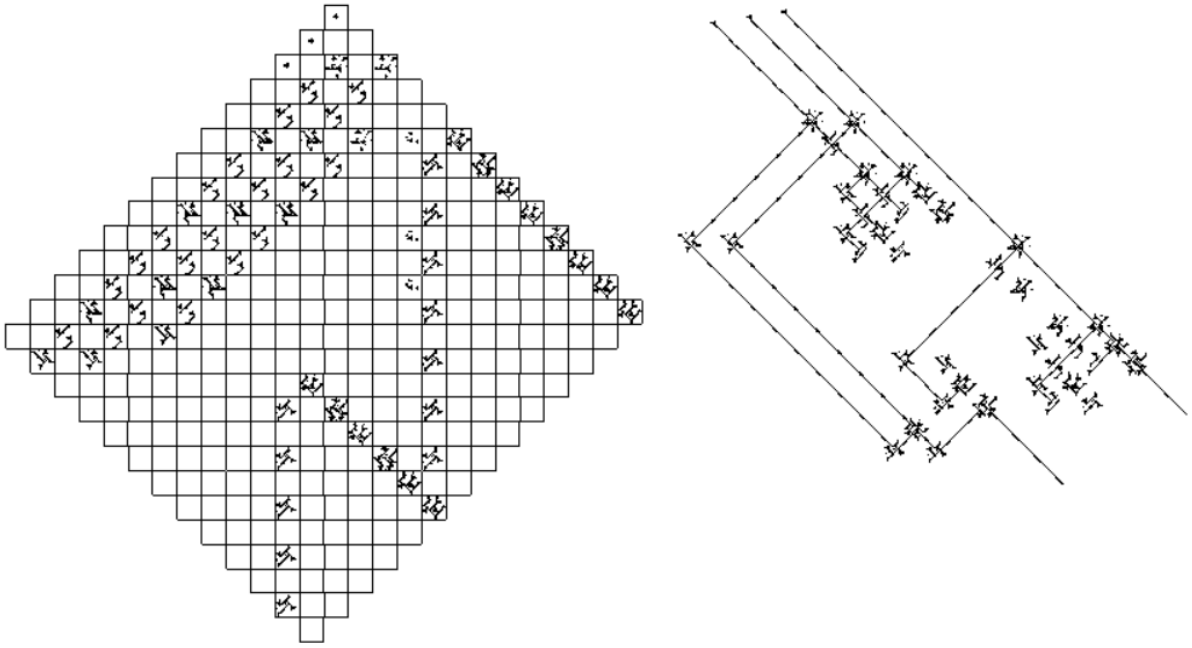
這次在康威生命遊戲中製作的加法器跟減法器都是模組化的，所以如果要計算更多位數，只要不斷疊加即可。假設串了n個模組，則能計算的上限為$2^n - 1$，例如串十個就能算 1023 以內的數。

但是在製作的過程中因放置元件非常麻煩，所以製作了可以自動完成電路的程式來解決擺放元件很麻煩的問題。雖然用自動完成的占用面積較手動排的體積較大（圖五十九），但僅需要輸入布林函式，就可以自己生成和自動電路布局設計圖。我在手動排互斥或閘的時候因為要對準座標，做這個互斥或閘就花了超過一小時，而自動生成的話我只需要不到一分鐘。

我也嘗試用我已經做過的全加器和用程式生成的全加器，雖然外型上體積大了不小，不過整個過程唯一花時間的只有在輸入布林函式時的打字時間。另外我也試做了一些我沒有做過的邏輯電路結構像是應用在記憶體的數據多工器、比較兩數大小的比較器。

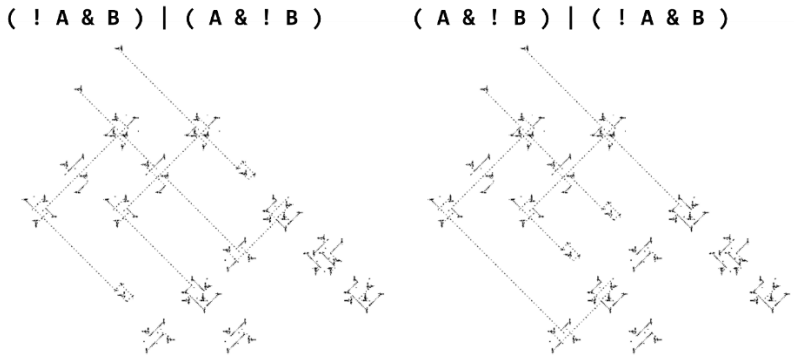


圖五十九　自動生成和手動排列的互斥或閘

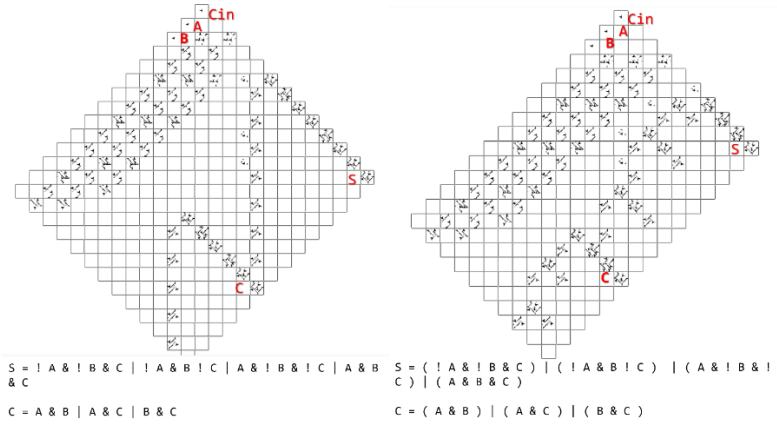


圖六十　自動生成和手動排列的加法器

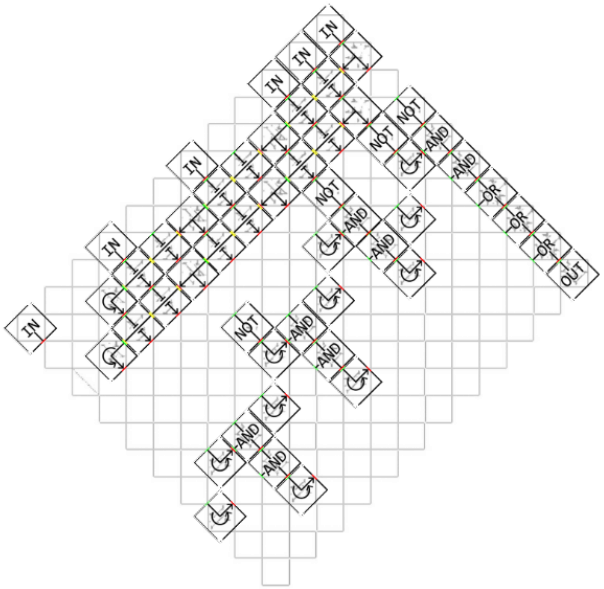
而如果不滿意自動電路布局設計的結果的話也可以更改布林函式的排序，雖然在邏輯上相同，但自動電路布局設計的結果是不同的（圖六十一、六十二）。



圖六十一　不同的布林函式生成的互斥或閘

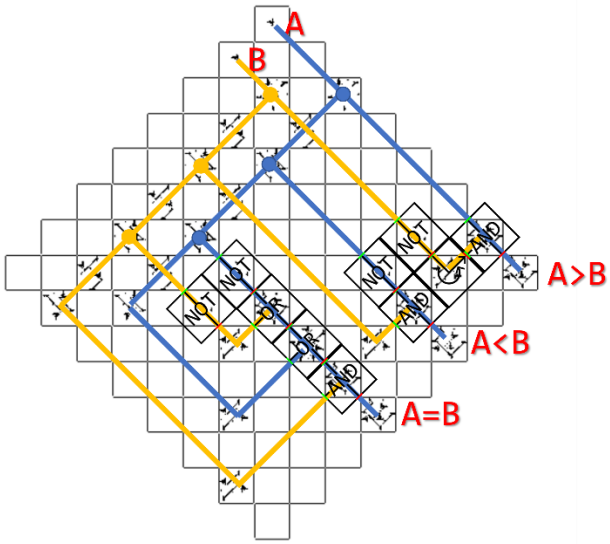


圖六十二　不同的布林函式生成的全加器



圖六十三　數據多工器

(! S1 & ! S0 & I0) | (! S1 & S0 & I1) | (S1 & ! S0 & I2) | (S1 & S0 & I3)



圖六十四　比較器

A>B : A & ! B

A<B : ! A & B

A=B : ! A & ! B | A & B

陸、討論

一、製作上的困難是什麼

一開始在拼接各元件過程中就經常遇到元件之間的週期對不上，這時候就要一直嘗試調整週期，或是週期對到了距離卻對不到，這也是要不斷重複嘗試，找對應的週期。

所以後來就往設計自動化的方向研究，來解決這個問題

二、這次研究的成果可以有什麼應用

在這次的研究中我利用了邏輯閘製作出了加法器和減法器來模擬電腦中的 CPU 運算，也模擬了現在很受關注的電子設計自動化（EDA）功能來幫我自動完成電路的設計，以及幫我放置在康威生命遊戲中。

我覺得這次的研究很適合做為教育用途，首先因為在康威生命遊戲中要實現邏輯電路的話會使用滑翔機來做為訊號，一串的滑翔機在縮小的狀態看起來就像一條線，可以很方便的看到訊號的走向、如何變化、經過邏輯閘在不同輸入下的不同輸出狀態，而也因為康威生命遊戲世界是由一個二維的平面所構成，所以也可以很直觀的看到邏輯閘內

部的工作。還有這次的自動電路布局設計也模擬的現在芯片產業中的電子設計自動化，以及在實現這個目標的過程中也應用了很多基礎程式設計的概念，像是：圖的深度優先搜索遍歷、中序以及後序運算式、堆疊（stack）、圖（Graph）等等。

柒、結論

這次的研究中我使用了 Nicholas Carlini 在 2020 年設計的邏輯閘和電路元件拼出了加法器以及減法器。經過串接後可以運算 1023 以內的數。但因為在實作將電路元件組裝成電路的過程中因為串接麻煩且容易失敗，所以著手研究如何讓製作電路這件事自動化。

在研究的過程中因為 Nicholas Carlini 的原設計不利於自動化布局的演算法設計，所以我幫及閘和或閘的一端輸入加上轉向器來統一輸入的方向，也重新設計的新的反向器，以及鏡像版本的訊號複製器。

最後我做出了只需要提供邏輯電路的布林函式就可以自動布局設計的程式，大大的節省了我在設計電路以及布局排版上所花的時間。

不過目前的設計仍然有需要改進的空間，像是因為演算法比較簡單，布局的結果會像一個樹狀圖，所以在很多情況布局的空間利用不是最佳化，未來可以朝優化這方面發展。

捌、參考文獻

Höltgen, S., Fecker, T., Pleikies, S., Wormsbecher, N., & Divani, S. (2020). A Case of Toy Computing Implementing Digital Logics with “Minecraft”. Humboldt University of Berlin (E-prints).

Omar Muñoz Urias (2022 年 10 月 27 日)。Full Adder Circuit – How it Works。https://www.build-electronic-circuits.com/full-adder/

Crowe J. & Hayes-Gill B. (1998). Introduction to digital electronics [recurso electrónico] $c. Arnold. Retrieved March 12 2023 from http://www.sciencedirect.com/science/book/9780340645703.

艾粒安鈉（2016 年 3 月 23 日）。電腦裡的生命遊戲，等你挑戰讓生命無限延續！。https://pansci.asia/archives/95148

Paul Rendell（2014）。Turing Machine Universality of the Game of Life。https://uwe-repository.worktribe.com/OutputFile/822581

Nicholas Carlini（2020 年 4 月 1 日）。Digital Logic Gates on Conway's Game of Life - Part 1。https://nicholas.carlini.com/writing/2020/digital-logic-game-of-life.html

Phillip Bradbury（2012 年 5 月 3 日）。Life in life〔影片〕。YouTube。https://youtu.be/xP5-iIeKXE8

Golly Help: Python Scripting. (n.d.). Golly. https://golly.sourceforge.net/Help/python.html

Chang, J. (2022, October 6). 【IC Design 豬屎屋 Marketing 20 天】IC 設計依賴工具，EDA 電子設計自動化. IT 邦幫忙. https://ithelp.ithome.com.tw/articles/10304898

Synopsys. (n.d.). Https://Www.Synopsys.Com/Glossary/What-Is-Electronic-Design-Automation.Html. Synopsys. https://www.synopsys.com/glossary/what-is-electronic-design-automation.html

【評語】052504

「康威生命遊戲」是一款在 1970 年由劍橋大學數學家約翰·康威（John Conway）發明的單人遊戲，由棋盤構成的細胞組成，可以模擬邏輯閘元件，並且可以構建簡易計算器。

此作品的作者在建構加法器和減法器的過程中，發覺手動放置電路元件非常耗時費力且容易出錯。為了解決這個問題，因此研究者決定開發一個能自動擺放元件的程式，以節省布局和放置元件所需的時間。作者使用 Python 語言結合康威生命遊戲的應用接口模組，開發了一款自動電路布局程式且將形成的電路用棋盤細胞的運作機制來實現。此作品具有創意但建議未來能比較與目前 EDA 方法的優缺點且凸顯此視覺化方法的教育價值。

作品海報

摘要

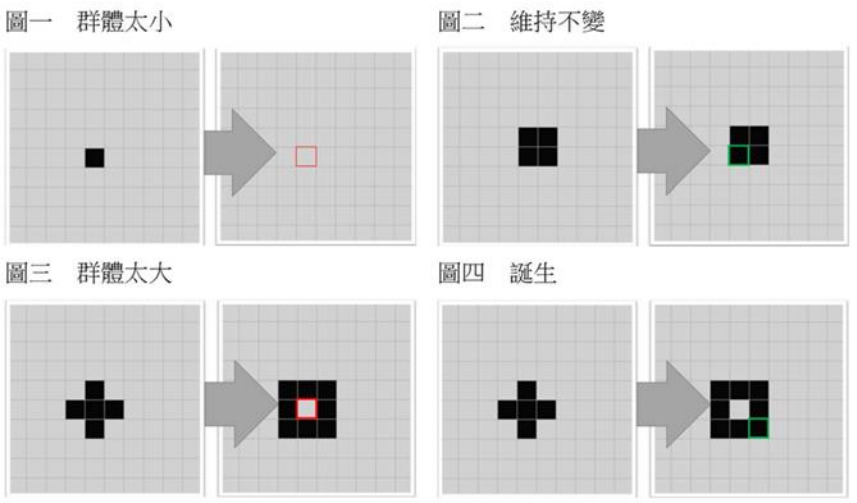
「康威生命遊戲」是一款在1970年由劍橋大學數學家約翰・康威（John Conway）發明的單人遊戲，由棋盤構成的細胞組成，可以模擬邏輯閘元件，並且可以構建簡易計算器。
在建構加法器和減法器的過程中，發覺手動放置電路元件非常耗時費力且容易出錯。為了解決這個問題，研究者決定開發一個能自動擺放元件的程式，節省布局和放置元件所需的時間。參考了電子設計自動化(Electronic design automation)的概念，使用Python語言結合康威生命遊戲的應用接口模組，開發了一款自動電路布局程式。只要輸入邏輯電路的布林函式，就能自動設計、布局和放置元件，大大縮短了製作邏輯電路上的時間成本。因可直觀地展示訊號的傳遞和邏輯閘的運作，也有助於學生理解邏輯電路的運作原理，應用於教學。

壹、前言

一、康威生命遊戲

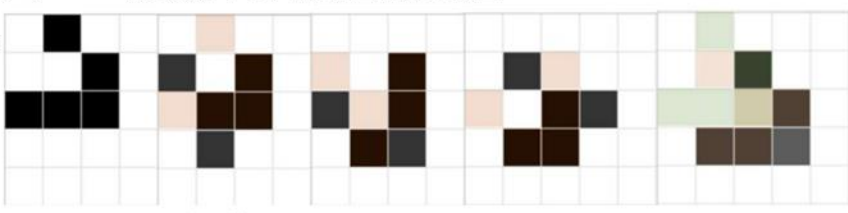
「康威生命遊戲」（Conway's Game of Life）是由約翰・康威在1970年發明的遊戲，由細胞組成。每個細胞會和周圍的八個細胞進行互動，有四個規則：

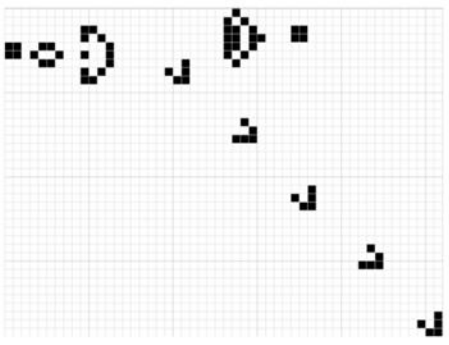
（一）該細胞為中心周圍少於兩個細胞時，細胞死亡。
（二）當周圍有二到三個細胞時，細胞保持不變。
（三）當周圍細胞超過三個時，該中心細胞死亡
（四）當一個空位周圍有三個細胞存活時，生成細胞。

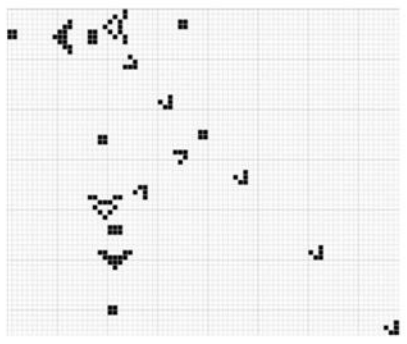


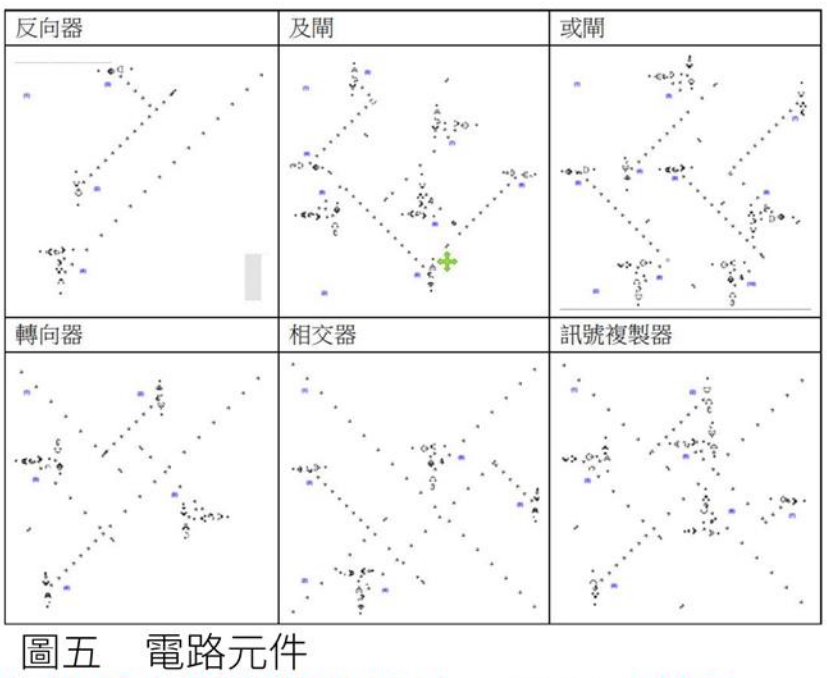
圖一　康威生命細胞的變化

要在康威生命遊戲中可以用滑翔機（Glider）這個結構，他會以四週期往斜向方向移動一格。
訊號源會用高斯帕機槍（Glider Gun）。以三十週期循環生成一個滑翔機，且可以使用兩個高斯帕機槍組成六十週期生成滑翔機的結構，來讓訊號可以相交。


圖二　滑翔機


圖三　高四怕機槍


圖四　六十周期的高斯帕機槍



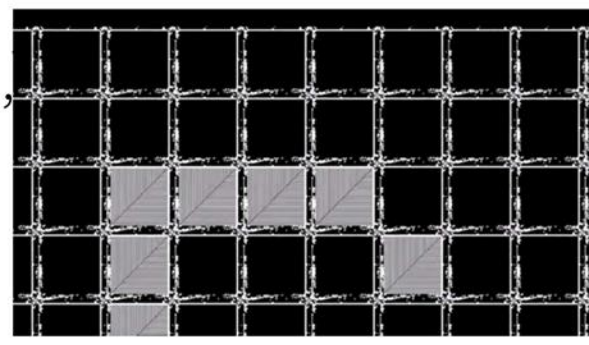
圖五　電路元件

二、電路元件

在這次的研究中我使用了Nicholas Carliniy在2020年發布在他個人部落格的設計。包含了AND、OR、NOT閘這三個邏輯處理元件，以及訊號複製器（Duplicator）、訊號轉向器（Rotator）、訊號相交器（Crossover）這三個用來控制訊號的元件。

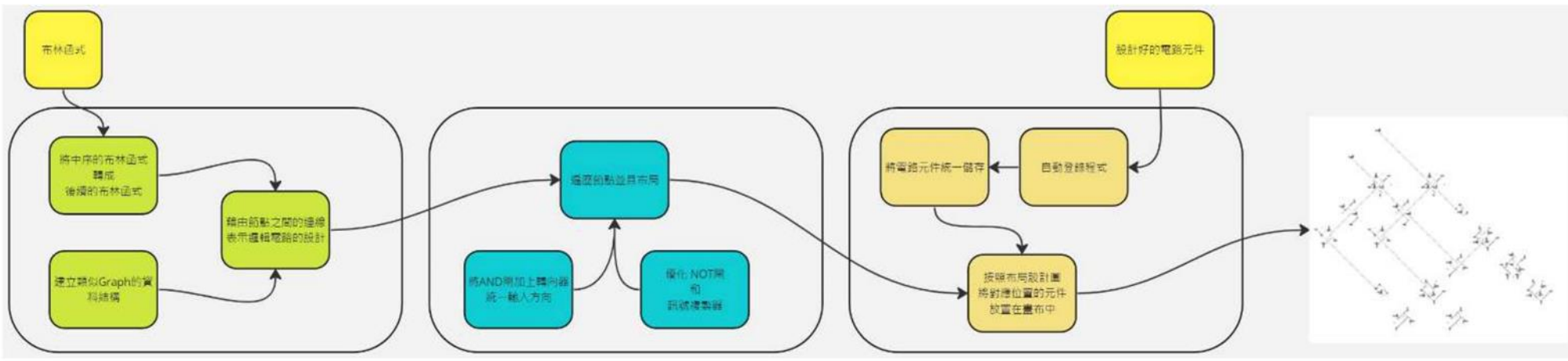
三、研究動機

在一次的研究邏輯閘的主題中我看到一則影片標題叫做「Life in life」這是Phillip Bradbury所製作的專案，他利用生命遊戲來模擬生命遊戲，這激起了我的好奇心：什麼是康威生命遊戲？這一格格的格子卻可以做出這麼酷的東西，還可以運算可視化。
深入探討後發現不少人利用康威生命遊戲做出「電腦」，於是我也想自己嘗試製作。但在後來的研究中，放置電路元件麻煩，所以使用了電子設計自動化的概念，來幫我自動完成電路布局設計。


圖六　Life in Life 影片截圖

貳、研究目的

一、在康威生命遊戲中模擬計算機中的加法器和減法器
二、製作自動電路布局設計程式
（一）將布林函數轉成電路設計圖
（二）從電路設計圖設計電路布局
（三）依照電路布局將元件排在康威生命遊戲模擬器中

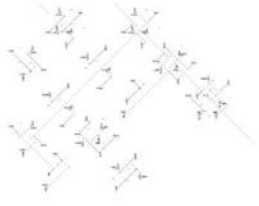


圖七　研究流程

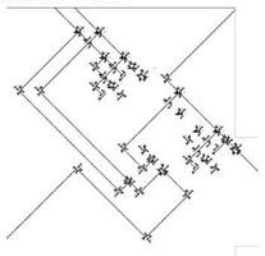
參、研究過程及方法

一、手動製作電路

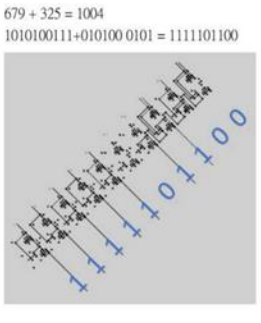
在一開始我手動擺放元件位置的時候，我先製作了互斥或閘，然後製作加法器才做減法器。



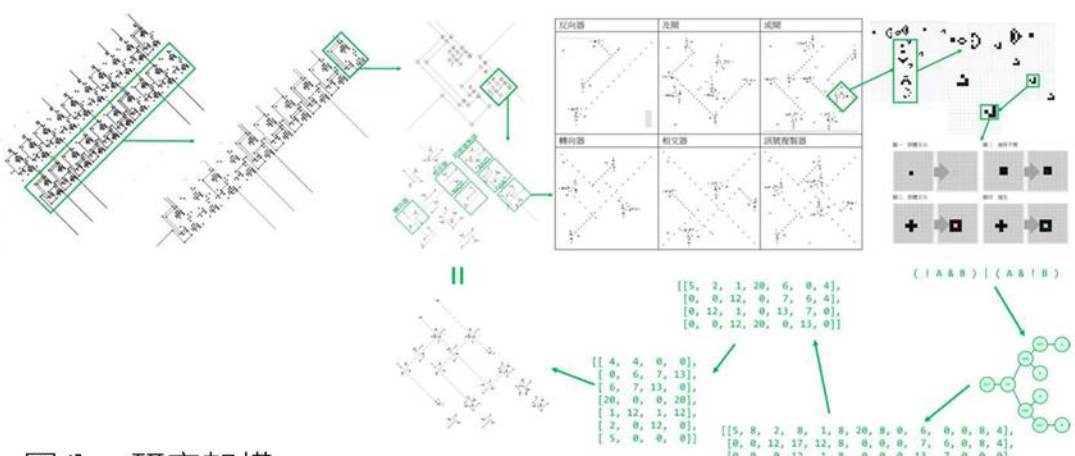
圖九　互斥或閘



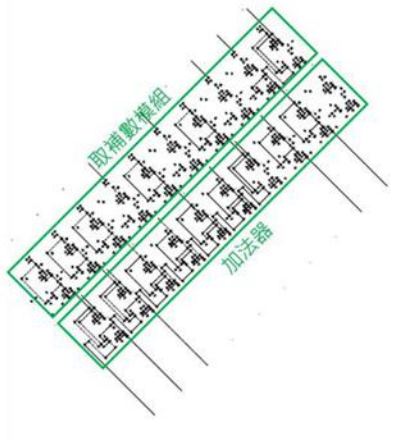
圖十　全加器



圖十一、圖十二　十位元加法器的運算結果



圖八　研究架構



圖十三、圖十四　十位元減法器的運算結果

二、手動製作電路的問題

在製作這些電路的時候，擺放的條件很嚴苛，一定要精準，就算差一格都會讓整個電路壞掉，為了解決手動放置困難的問題，我做了一個用來排版的網頁工具，可以藉由拉方塊的方式調整電路元件擺放的位置。

三、製作可以自動生成布局設計的程式

雖然有工具可以幫我跳過手動放置的步驟，但我還想再做的更好，所以想做一個只要我輸入布林函數，就可以自動生成整個電路，並且幫我排好位置。

（一）讓電腦表達電路圖

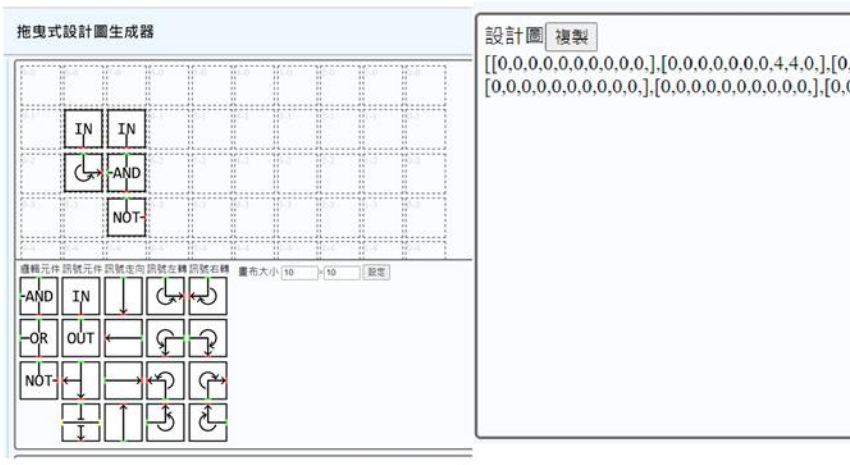
我在python中使用了class來做出graph，各節點就是電路中的元件，並且節點跟節點之間的邊就代表電路中各元件間的連接狀況。

（二）中序布林函數轉後序

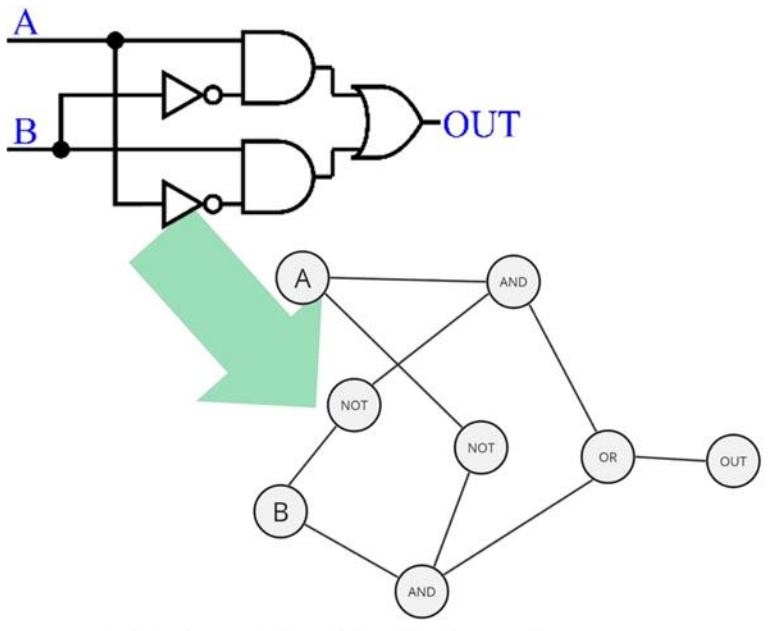
我參考一般處理算式的方式，將中序運算式的布林函數先轉成後序運算式的布林函數，再用處理好的布林函式進行節點的生成和節點之間的連結。

（三）改良邏輯閘

原本參考的設計，AND和OR閘的兩個輸入是互相垂直的，所以在邏輯閘的左邊再套一個轉向器，讓兩個輸入和輸出的訊號是相同方向的。我也重新設計NOT閘，讓輸入跟輸出也是相同方向的。



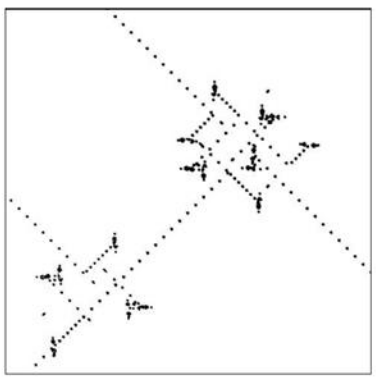
圖十五　拉方塊的編輯區　　圖十六　生成的結果

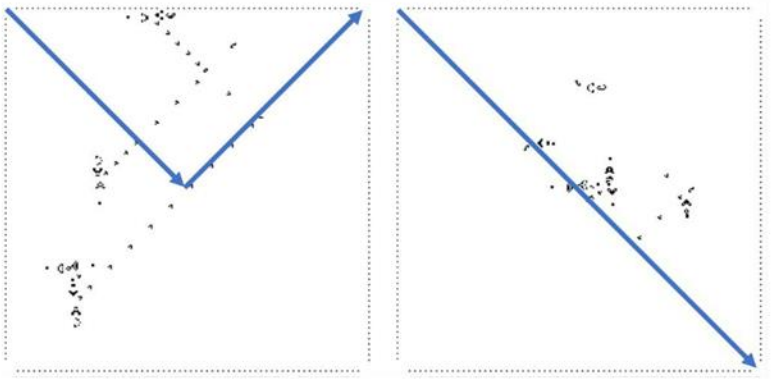


圖十七　把電路圖轉成graph

（四）生成布局

在剛剛生成好的graph中以輸出端往回遍歷的末端是輸入端，而每個輸入端都要佔一行，所以第一次先走到底，確認第一排後，再往下走，所以會用到深度優先搜索（DFS）。如果走到底了，也就是遇到輸入端的話就往下一行。

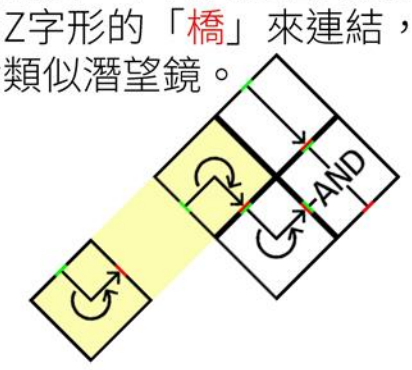




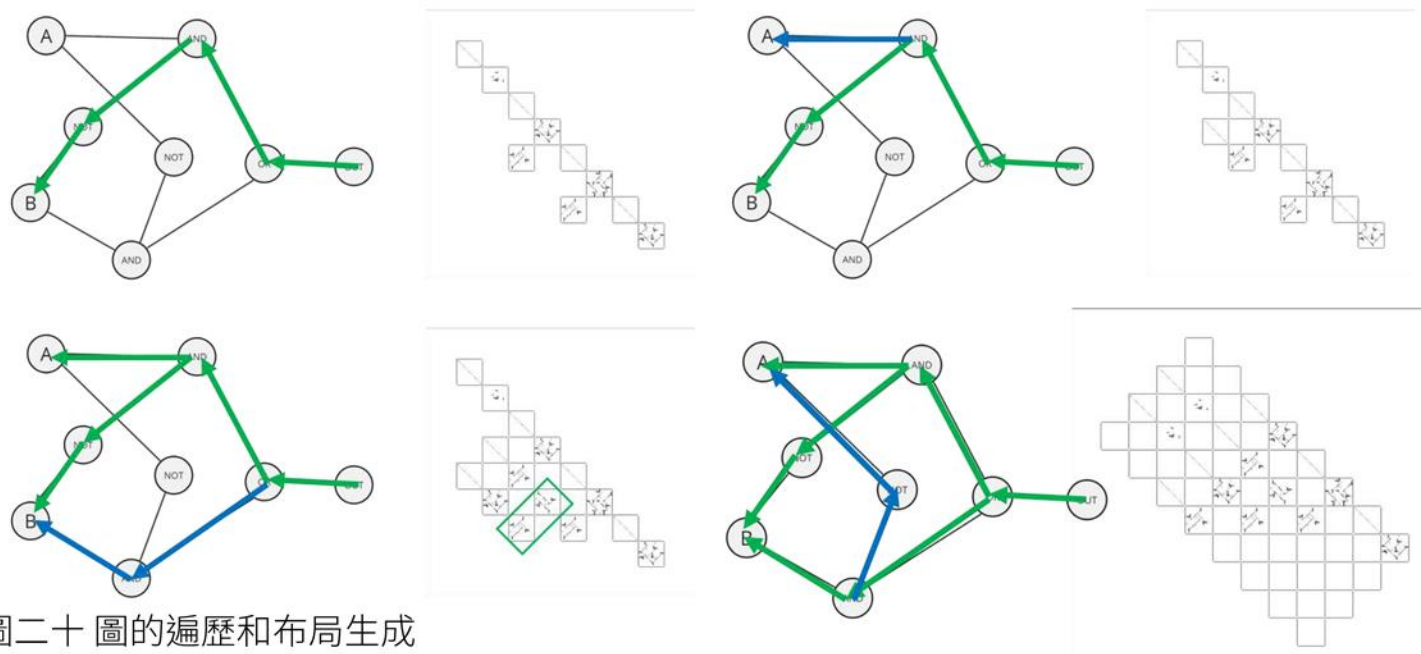
圖十八　改良的AND閘　圖十九　改良前漢改良後的NOT閘

（五）連接不同高度的訊號

因為當遍歷到葉節點的時候會換下一行，所以有可能兩個要連結的元件之間高度不同，為了解決這個問題，不同高度間的元件會使用Z字形的「橋」來連結，橋的概念類似潛望鏡。



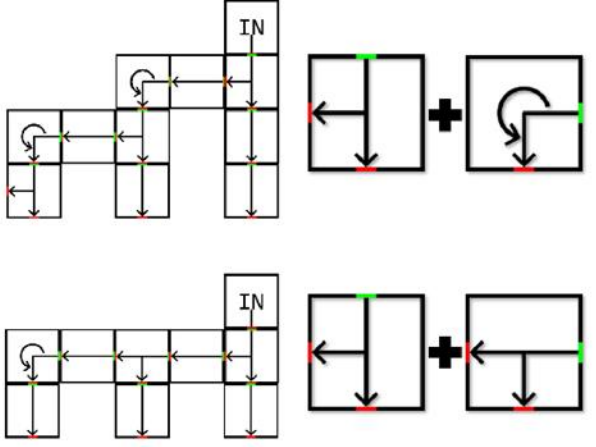
圖二十一 橋



圖二十 圖的遍歷和布局生成

（六）輸入分享

由圖遍歷後的樹狀圖的葉節點就是輸入端，但可能有相同的輸入端，但在不同高度，所以需要把輸入端複製到其他地方，若用原本的訊號複製器，面積會比較大，所以使用了鏡像版的訊號複製器來輸入分享。



圖二十二　輸入分享的比較

（七）優化面積

為了簡化演算法，減小布局面積，我使用兩個方法。分別是消除空行及轉角優化。

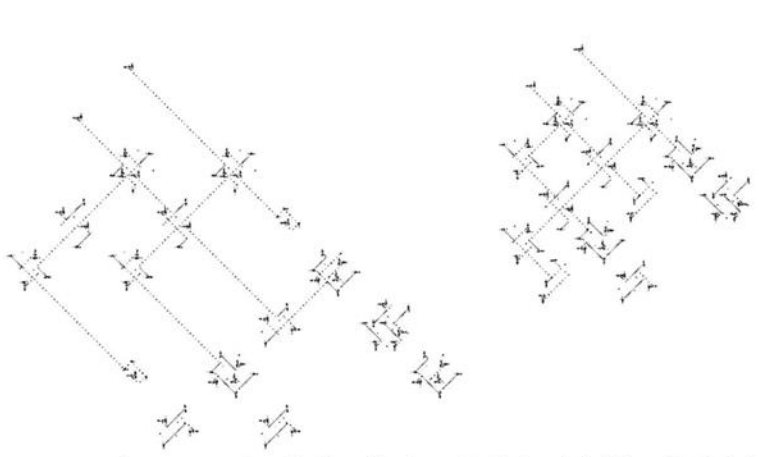
圖二十三　優化的變化

肆、研究結果

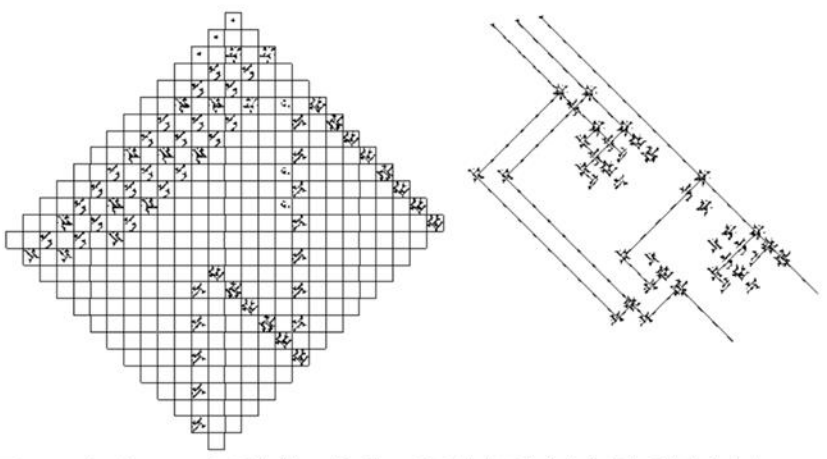
在這次的研究初期我手動做出了十位元的加法器和減法器，後來為了解決手動放置的嚴苛條件，所以做了一個可以自動布局設計的程式，以下是我生成的比較器和數據多工器。

圖二十四　由布林函式所生成的電路

雖然自動生成的布局方式不一定是最佳解，但可大幅減少設計時間。以我製作全加器的經驗來說，手動布局約花費了7個小時，但使用自動布局工具，一秒鐘即可生成！



圖二十五　自動生成和手動擺放的體積比較



圖二十六　自動生成和手動擺放的體積比較

比較項目	傳統設計	軟體布局設計
製作方式	人工	電腦自動化
花費時間	數小時	數秒鐘
電路面積	較小	較大

圖二十七　手動放置和自動布局設計差異比較

伍、應用

一、電子設計自動化（EDA）

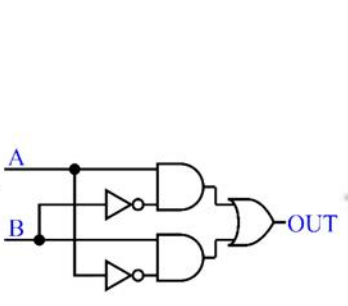
在這次的研究中我利用了邏輯閘製作出了加法器和減法器來模擬電腦中的CPU運算，也模擬了現在很受關注的電子設計自動化（EDA）功能，讓我只需要輸入布林函式，就可以幫我自動完成電路的設計，並放置於康威生命遊戲中。

在實現這個目標的過程中也應用了很多基礎程式設計的概念，像是：圖（Graph）與圖的深度優先搜索、中序以及後序運算式、堆疊（stack）……等。

二、運算可視化

布林運算與邏輯閘，是資訊教育的基礎。儘管使用模擬組合電路展示燈泡亮暗，但仍枯燥乏味。

如果使用本研究成果，數位電路變成一個個鮮活的生命，會移動、交戰、吞併。這些大大小小的生命，組成我們資訊的世界，對於學齡兒童的資訊教育，有很大的吸引力。



圖二十八　電路圖和康威生命遊戲