第一章
緒論

1.1 研究背景

目前高科技產業使用的模擬運動平台種類繁多，有三軸或六軸等應用。三軸有的是沿著 X、Y 和 Z 軸移動的三維運動平台，有的是可以繞著 X 軸、Y 軸以及 Z 軸旋轉的運動平台。六軸則將以上兩種融合，形成既可旋轉，亦可沿著軸移動的運動平台，較著名的就是史都華平台。

但不論是三軸或是六軸，其設備都龐大且操控介面複雜，在使用上受到諸多限制。有鑒於此，本論文鎖定的目標是為三維運動平台架構一個簡易的圖型控制的運動介面，並可應用於製圖機、自動塗膜機、電子束直寫機等。相異於傳統運動控制介面多是以文字，或硬體上固定的一組按鈕，本論文設計的圖形控制介面更加人性化，不但提供使用者預覽圖形的功能，並且提供直覺性地圖形化介面給使用者操作。

1.2 運動平台

三維運動平台於高科技產業中應用範圍廣泛，在各種電子產品中都會使用，因此為該運動控制平台設計一個良好的控制介面以提升工作效率顯得更加重要。

舉例來說：當建築工程師想繪製工程中的各種精確設計圖，所使用的繪圖

機，其列印方式與傳統照相利用感光的方式不同，其原理有點類似拼圖。首先電腦將打算要列印的圖檔切割成許多小方塊並標上編號，傳送給繪圖機列印。機器列印時將收到的小方塊都塗上類似原來照片的顏色，即使混在一起，因為有編號，所以能夠很容易再拼湊出原來的照片。如果印出來的效果並不好，可以增加圖片切成小方塊的數量，直到眼睛分辨不出原來的照片圖像與拼湊出的圖像的區別為止。[1]

從原理上分類，繪圖儀分為筆式、噴墨式、熱敏式、靜電式等。從結構上分，又可以分為平臺式和滾筒式兩種。平臺式繪圖儀的工作原理是，在電腦信號的控制下，筆或噴墨頭的Ｘ、Ｙ方向移動，而紙在平面上不動，從而繪出圖來。滾筒式繪圖儀的工作原理是，筆或噴墨頭沿Ｘ方向移動，紙沿Ｙ方向移動，這樣，可以繪出較長的圖樣。繪圖儀所繪圖也有單色和彩色兩種。目前，彩色噴墨繪圖儀繪圖線型多，速度快，解析度高，價格也不貴，很有發展前途。[2]

本論文所專注的主題：在三維運動平台上架構一個圖型控制的運動模式，也可以應用於繪圖機上。

1.2.1 運動平台應用於自動塗膜機

在國際競爭下，台灣如何降低產品成本，縮短供貨期，和提昇液晶顯示器的生產速度喊品質，已經成為各家顯示器廠商亟欲解決的問題。除了前段光學製程技術要領先外，後段模組的組裝流程也要加強改善。如果能利用自動塗膜機來解

決後半段人工塗膠過程的化，不但將低人工塗膠的人力成本以及人工搬運之費時費力，且能二十四小時無休的進行作業，經濟效益將能大幅度提升。

自動塗膜機的工作原理就是利用三維的運動平台上架設膠頭、LVDT 感測器、照明光源、鏡頭以及 CCD 感測器，透過壓力計對膠頭施壓以塗膠，並利用資料擷取裝置 (DAQ)來感測膠頭離物體表面的遠近。自動塗膜機會連上工業電腦，利用電腦所提供的塗膠系統來操作運動模式。

本論文所專注的主題：在三維運動平台上架構一個圖型控制運動介面，也可使用在自動塗膜機上，利用讀取寫好的圖形檔，讓運動平台依據畫好的圖形來進行塗佈工作。

1.2.2 運動平台應用於電子束直寫機

目前台灣大多數的晶圓廠在 157 奈米半導體製作方面，是採用 193 奈米搭配浸入式微影來實現，但濕浸式光學顯影仍然有多項關鍵因素待克服，如水中微泡的控制，當機台上的晶圓以每秒五十公分的速度移動時，其間形成的微氣泡可能損及晶圓上的成像，因此如何在事先去除氣體的純水可能預防氣泡生成是關鍵之一。其次水與光阻交互作用，因為水將會對不同光阻劑造成不同程度傷害，因此必須要作一些考量。

有人提出採用電子束微影來取代光學微影製程，電子束微影與光學微影製程的步驟類似，目的都是將所需的圖形縮小複製到晶片上，差別在於光微影術是利

用「光線」來刻劃圖形，電子束微影則是利用能量為數萬電子伏特(eV)的「電子束」作為曝光源。由於電子的波長比一般光微影製程所使用的光源波長更小，因此能提供更高的解析度。

圖 1-1：奈米科技研發中心所使用的電子束直寫機

電子束微影可輕易達成數百至數奈米尺寸的線寬，除了可以用來製作光微影所需的光罩(mask)外，它還有一獨特的優點：直寫(direct write)，即不需要光罩就能定義圖形，可輕易畫出不同的設計。

一般的電子束系統包括以下幾個部分：電子槍(Electron Gun，可用來提供電子源)，電磁透鏡(Condenser Lens，用來控制電子束的形狀及聚焦程度)、運動平台，以及電腦介面控制軟體。

本論文所專注的主題：在三維運動平台上架構一個圖型控制的運動模式，也適用於電子束微影系統，利用讀取圖形的功能，可完成一個接受 GDSII Layout 圖檔輸入之自動化運動系統，將三維的運動機台運動模式套用於二維的運動平台

上，移動電子槍讓電子束打在欲曝光之光阻上，完成不需光罩直接寫入的目的。[3-8]

1.3 研究項目

在前面舉的幾個例子：製圖機、自動塗膜機、電子束直寫機中可以看出來，三維運動平台應用之廣泛與在工業上的重要性。然而，目前的三維運動平台大多仍停留在傳統的文字為主的控制介面，而不能提供一個直覺化的使用者介面——也就是提供圖形，讓使用者和更為了解機器的運作，並強化彼此溝通、運行的介面。

由於運動平台可以應用於各種目的，本論文礙於時間與經濟的考量下，無法通通加以驗證，因此挑選幾項重要的功能來做設計並驗證：

1. 三維平台的運動：

將製作一個圖控而非文字的介面提供給使用者操作。

2. 圖形的輸入與輸出：

使用者可在直覺式的圖形上修改，再讓運動平台依據此圖形運動，提供彈性且方便的操作方式。由於圖形檔案種類繁多，本論文將專注於 Lay out 常用之圖形檔：GDS II 的支援開發。

針對以上的研究目標，本論文的研究項目如下：

1. 尋求一個快速且嚴謹的開發方式，以符合現今產業的需求。

2. 專注於電腦介面控制軟體，發展圖控之人機界面程式。

3. 可接受 GDSII 圖檔之自動化運動系統。

4. 挑選一種用途進行實機操作，以驗證本論文之設計概念。

第二章
分析與探討

本論文的目的是為了製作一個圖型化控制之運動平台，在前面我們界定了研究項目，為了達成前述目標，我們將由尋找適合開發圖控介面的工作環境開始，到實際操作所需用到的運動功能，依序介紹。

本章第一小節，將介紹開發圖形化介面的程式語言－－圖形化程式語言(Visual programming language，VPL)發展的原由以及 VPL 的優缺點；第二小節說明利用 VPL 所開發的圖形化介面之優點、特質以及本論文所選擇的 VPL；最後的部份則向大家說明本論文所使用的運動控制卡將會用到的相關運動控制功能，包含單軸直線運動、雙軸直線運動、三軸直線運動、兩軸圓弧運動。

2.1 圖形化程式語言發展原由

人類溝通的方式並不只是只有聲音或文字而已，當中有很重要的一部分是依靠著圖像來傳達資訊，例如在電腦的人機介面上，圖形語言扮演了關鍵的角色，而且其重要性日與劇增。[9]

將圖形語言應用於程式語言，即圖形化程式語言(Visual programming language，VPL)的誕生，VPL 意指任何將程式中的文字成分改用圖形來取代的程式語言。VPL 允許視覺化的編寫程式：在空間上任意安排文字以及圖形符號。大部分的 VPL 是基於「方塊和箭頭」這樣的點子，意思是將方塊、圓圈或是泡

泡，視為螢幕上的物件，利用箭頭、線條或是弧線來連結它們。非 VPL 的程式語言都是用文字來構成整個程式，除非你去執行它不然你絕對看不到任何圖像。這就是非 VPL 語言的致命缺點：使用者無法直觀的去修改執行後所產生的圖像或介面。

我們還可以依據你想要的用途和種類而使用不同類型的 VPL：可以使用由小圖塊拼構而成的程式、定義型式為主的程式或是以流程圖為主的程式。VPL 只要提供代表各種功能的圖塊，使用者就可依據他的需求或便利來組成她想要的程式類型。[10]

2.1.1 圖形化程式的優缺點

前面介紹了 VPL 的特色，在此小節我們要來分析 VPL 和一般使用文字的程式語言有何差異？VPL 和文字程式語言比較的優缺點又是什麼？為了方便明白，這邊將優缺點做成表格來說明。

VPL 的優缺點如下：

VPL 的優點	VPL 的缺點
花費較少的時間與精神 就可以得到好的結果	符號、標記和工具的圖像 表示缺乏統一的標準
更佳的表達能力	不易解讀和維護
圖像化的使用者介面	沒有互相轉換的平台
	高度的美學要求
	程式密度較少

表 2-1 VPL 的優缺點

在 VPL 優於傳統文字型程式的部分說明如下：

1. **花費較少的時間與精神就可以得到好的結果：**

VPL 利用圖像來表達意含，在表達點子、物件之間的關係以及人機介面的設計方面都比傳統的文字程式語言還要來的直覺。

2. **更佳的表達能力：**

利用小圖塊組成的程式，由於不同類型的資料型態或結構其圖塊都不同，讓使用者在編寫程式時很清楚的就可以掌握資料的流向和結構。

3. **圖像化的使用者介面：**

當你在設計 VPL 的程式時，就同時在設計使用者介面了。不像傳統文字

程式語言的介面大多是文字指令輸入輸出，VPL 以圖像化的介面提供給使用者，讓操作的便利性大大增加。

它也有些劣於傳統文字程式語言的地方，說明於下：

1. **符號、標記和工具的圖像表示缺乏統一的標準：**

造成這種現象除了因為各種 VPL 的平台差異以外，對同一個意涵難以訂定統一的圖像標準也是個原因，這造成使用者在 VPL 內看到一個新的符號時，必須去了解他背後所代表的意義。

2. **不易解讀和維護：**

雖然 VPL 撰寫時很容易，但是當欲解決的問題十分困難時，整個程式充滿了各種圖塊，反而讓其他人無法輕易地看懂該程式的原理。

3. **沒有互相轉換的平台：**

由於每個 VPL 平台所提供的圖像方塊和介面工具都不相同，因此不同平台所寫的 VPL 是無法互相轉譯的。

4. **高度的美學要求：**

VPL 雖然上手容易，但是能善用 VPL 的程式高手卻難以培養，一個會使用 VPL 但缺乏美術休養的程式設計師，可能把程式圖塊畫的一團糟，造成他人解讀時容易判斷錯誤。

5. **程式密度較少：**

同一個大小的顯示畫面，用文字程式語言的程式設計師可能已經完成了

程式的百分之八十，而使用 VPL 設計的程式可能還表達不到程式的百分之三十，這是由於圖塊設計所需的空間比文字大。這也造成 VPL 難以去設計龐大且複雜的程式。

雖然 VPL 有上述的缺點，但是我們在製作圖控介面時，VPL 仍然是我們的最佳選擇，因為它提供了直覺性的圖形介面支援，並讓程式設計師在編寫程式時就能同時繪製圖控人機介面。

本論文的主題為：圖控運動控制平台，為何要那麼強調圖控人機介面呢？和文字控制介面相比，圖控介面的好處又在哪裡呢？在下面的一小節將說明圖形化控制介面的優點。

2.2 成功的圖形控制介面特質

圖控介面的設計在原則上可以歸納為以下幾點：

1. 運用熟悉的事物做象徵。

2. 提供即時的回應。

3. 保持介面的一致與單純

4. 提供清單避免記憶。

5. 提供直接且適量的控制權給使用者。

運用熟悉的事物(metaphor)來做象徵的主要目的是，將使用者在生活中已熟

悉的景物設計於軟體的介面上，當使用者看到他熟悉的景物象徵時，便能較快的了解其功能而節省了學習介面的時間。

即時的回應是指當使用者以滑鼠或以鍵盤等裝置輸入了某些動作，電腦對其動作所做的回應。

另外，最好介面能提供清單(跳出式、隱藏式、或停留在螢幕邊緣)給使用者選擇而不需記憶指令。

2.2.1 選擇圖形控制語言

了解了介面設計的理念之後，接下來就是要挑選適合本論文求的 VPL，由於本論文的主題是：圖控運動控制平台。在介面設計上，要強調的是和儀器之間的互動和關聯，即所謂的虛擬儀器控制。

在此先介紹何謂虛擬儀控系統，有別於功能固定的傳統儀器限制，虛擬儀控系統讓使用者能夠利用電腦以及各種硬體，來建構屬於他們自己的儀控系統。這些以軟體為中心的系統，運用了電腦的運算、顯示以及連結能力，讓使用者能夠彈性的來建立所需要的各種儀控功能。

因此我們所要挑選的 VPL，當然是選擇支援製作虛擬儀控最佳的平台。現今有一套功能強大且設計完整的套裝軟體 LabVIEW，能幫助我們實現迅速建立自己測試系統這個目的。

LabVIEW (Laboratory Virtual Instrument Engineering Workbench)，是由 NI

(National Instrument，美國國家儀器)公司所製作研發的虛擬儀器圖控語言軟體。我們可以使用 LabVIEW 來控制所發展的系統，並且利用互動式的圖形人機介面來呈現。LabVIEW 也適用於各種平台，可以搭配 Windows NT/98/3.1、Mac OS、Sun、HP-UX 以及 Concurrent PowerMAX 來使用。它所支援的裝置包括 GPIB、VXI、PXI™ 、串列裝置、PLC，以及資料擷取(DAQ)介面卡。並且可以經由網際網路、ActiveX 等不同應用程式間的通訊協定、資料庫連結方式，連結其他的資料來源；亦可以在 LabVIEW 中嵌入 ActiveX 的物件，從其他環境中呼叫 LabVIEW 的編碼；並可使用 LabVIEW 來呼叫 Windows 底下既有的動態連結檔(DLL)或者其他平台下的共享程式庫。此外 LabVIEW 還可以製作出可獨立執行的程式，可在沒有 LabVIEW 發展系統的設備中執行。[13]

2.3 運動控制功能

當我們設置好 LabVIEW 程式設計平台後，可使用 LabVIEW 來呼叫運動控制卡既有的動態連結檔(DLL)，進而控制並執行三維運動平台的多組運動模式，至於呼叫動態連結檔的方法，在第三章會有詳細敘述。

在此我們將介紹驗證時所用到之相關運動控制功能及原理。本次實驗會用到的相關運動功能有：單軸直線運動、雙軸直線運動、三軸直線運動、兩軸圓弧運動。其中雙軸以上的多軸運動，則會進行動態插補，插補的動作由運動控制卡來完成。詳細介紹如下：

2.3.1 單軸直線運動模式

在單軸直線運動模式中，控制卡會對馬達輸出一串指令脈衝集，脈衝的個數對應到馬達所運行的「距離」，脈衝頻率則對應到馬達移動的「速率」，另外再輸出一個位元代表「方向」。

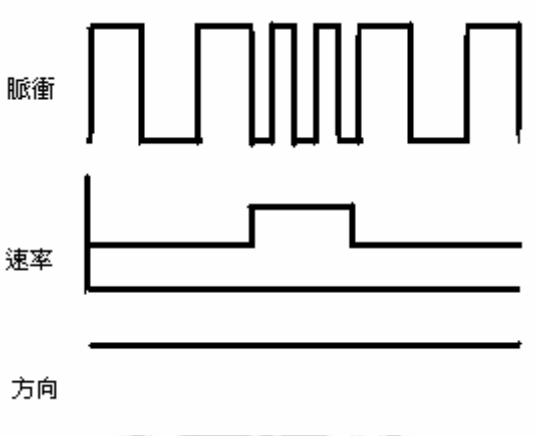

圖 2-1 單軸直線運動模式原理圖

2.3.2 雙軸直線運動模式

雙軸直線運動模式，其情況與單軸運動十分相似。在此模式之下，運動卡所進行的線性插補動作，其原理如下圖所示：

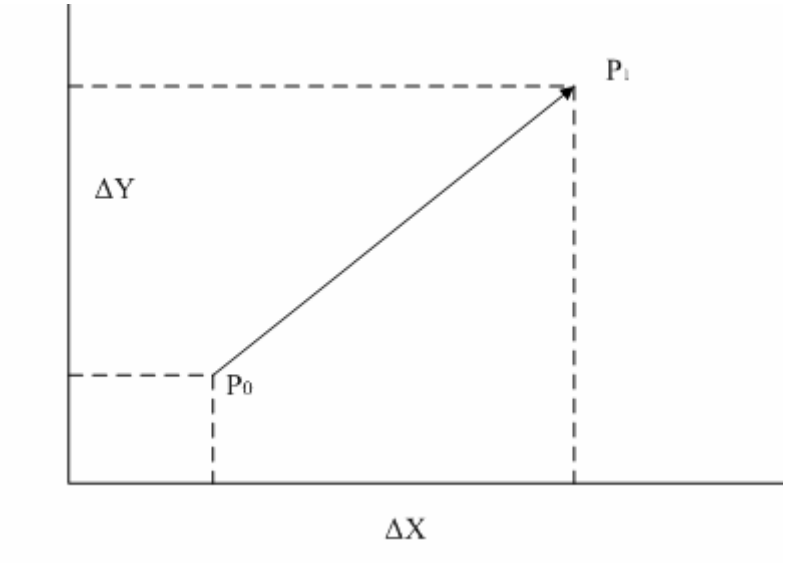

圖 2-2 雙軸運動模式原理圖

雙軸線性插補意味著由 P_0 移動到 P_1 的時候，兩軸同時開始運動且運動速度之比為 V_X: $V_Y = \Delta X$：ΔY 之定值，向量速度為 $\frac{\Delta P}{\Delta t} = \sqrt{(\frac{\Delta X}{\Delta t})^2 + (\frac{\Delta Y}{\Delta t})^2}$ 。當使用雙軸線性插補時，使用者必須提供向量速度，包含初始速度以及最大速度，運動控制卡會自動執行插補的動作。

插補的過程，定義最長的移動距離軸為長軸，其他軸為短軸，長軸輸出均勻的脈衝、維持恆定速度，而短軸移動則是在適當的時機插補入長軸的移動序列中。由上圖可以看出來，插補所造成的最大誤差會在 0.5 單位以內。

2.3.3 三軸直線運動模式

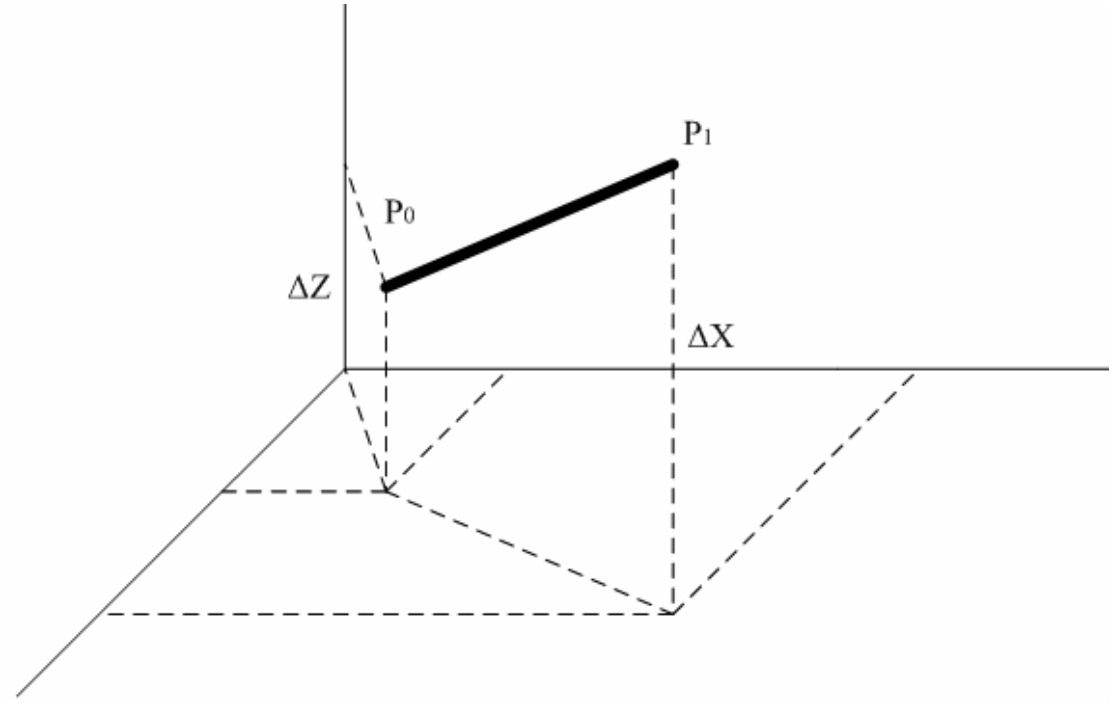

圖 2-4 三軸運動模式原理圖

三軸線性插補表示將三個軸，例如 XYZ，由 P_0 移動到 P_1，三軸同時開始運動且保持速度之比為 V_X: V_Y: V_Z = ΔX：ΔY：ΔZ 之定值，其運動向量速度為 $\frac{\Delta P}{\Delta t}=\sqrt{(\frac{\Delta X}{\Delta t})^2+(\frac{\Delta Y}{\Delta t})^2+(\frac{\Delta Z}{\Delta t})^2}$ 。

同樣的，三軸運動也是利用插補來完成，與二軸運動所產生的誤差原理相似，其最大插補誤差在 0.7 單位以內。

雙軸和三軸運動速度函數，使我們可以決定任意線段的切線速度，這在考慮要求相同曝光時間上的操作是非常重要的因素，只有維持相同切線速度，才能保證光阻所受到的照射時間是相等的。

2.3.4 雙軸圓弧運動模式

運動控制卡實現圓弧運動的原理有點類似實現直線插補的情況，如下圖所

示，利用 X 的絕對值大於還是小於 Y 的絕對值將整個圓分成八個象限，在|X|>|Y|的時候，對 X 軸輸出穩定脈衝，而 Y 軸在適當的時機進行插補；反之亦然。

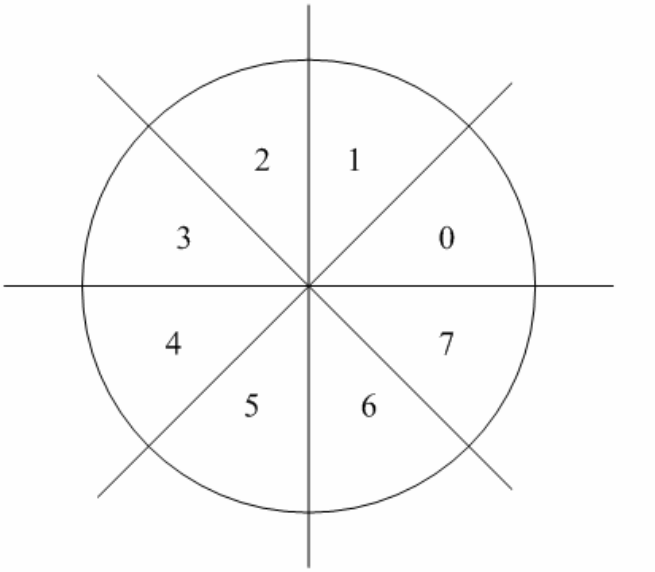

圖 2-5 雙軸圓弧運動模式原理圖

雖然運動控制卡提供的唯一非線性運動就是雙軸圓弧運動，但是我們可以利用侯杰利在《尋跡式創新塗膜》(Curve tracked novel coating)中提出的利用直線、圓弧來近似任意曲線之方法。[14]

第三章
系統設計

上一章我們介紹了本實驗所選取的相關工具，在本章裡，將從需求擷取、系統硬體架構設計到軟體規劃的實作路線，對圖控運動控制平台系統設計做個完整的規劃。需求擷取的部分我們將界定本論文要達成驗證的目標，透過系統架構設計將我們實驗所需的硬體設備做個規劃，最後軟體規劃的部份將詳細說明如何用 LabVIEW 實作介面程式。

3.1 系統需求擷取

我們的目標包含一個可移動之三維運動機台，以及讓機台運動之圖控介面程式。[13]另外為了模擬多種圖形複製之應用，該圖控介面必須包含可讀取圖形檔並執行運動之功能，本論文選取 GDSII 規格之圖形作為模擬之介面設計對象。另外就是為了模擬電子束直寫機曝光時各區域所受到的曝光量為均勻，我們必須讓機台之載具以等速率移動，該部分將由介面程式來完成。

總而言之，本論文所提出圖控運動平台，要驗證上述功能，茲將功能需求條列於下以供方便理解：

1. **可移動機台的圖控運動介面。**
2. **可支援 GDS II 規格的圖形並讓載具運動完成該圖形之功能。**

3.2 系統架構設計

下圖為整個系統之示意圖，包含了軟體以及硬體部分。硬體部分主要包括運動控制卡、馬達伺服器、運動平台，將在本小節做個詳細的說明與設定；圖控介面開發部分放在下一小節，屆時我將講解如何利用 LabVIEW 設計圖控介面。

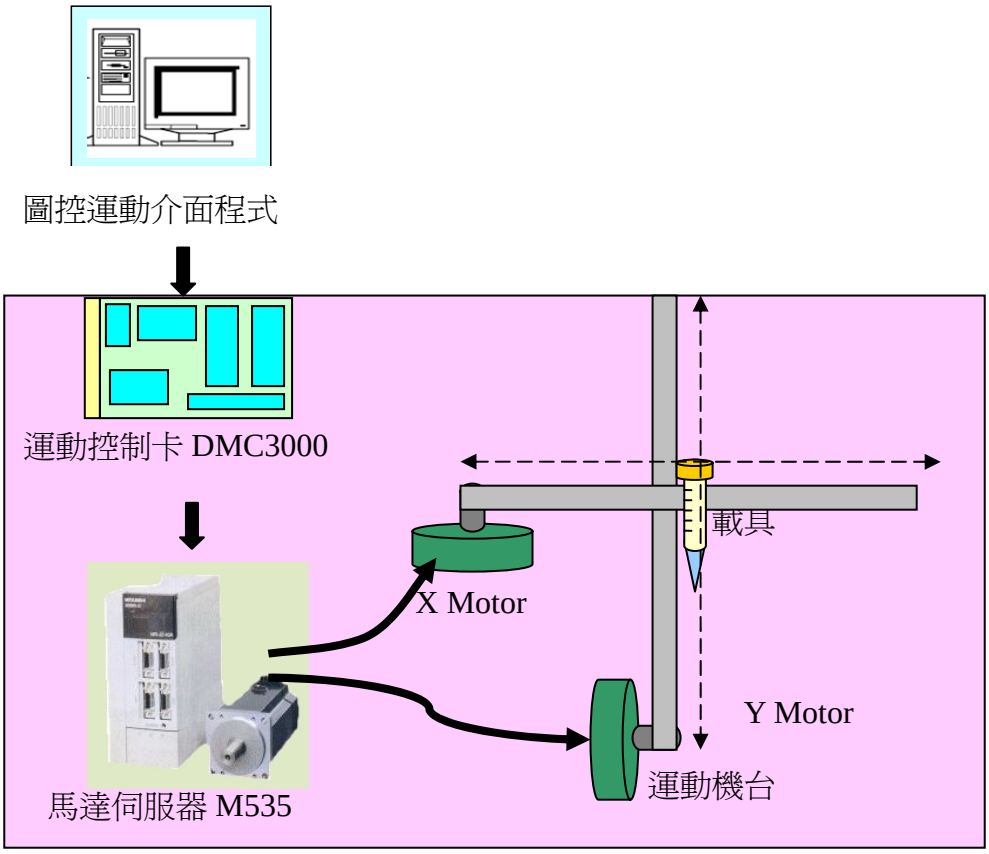

圖 3-1 系統示意圖

系統的硬體方塊圖如下頁圖所示：

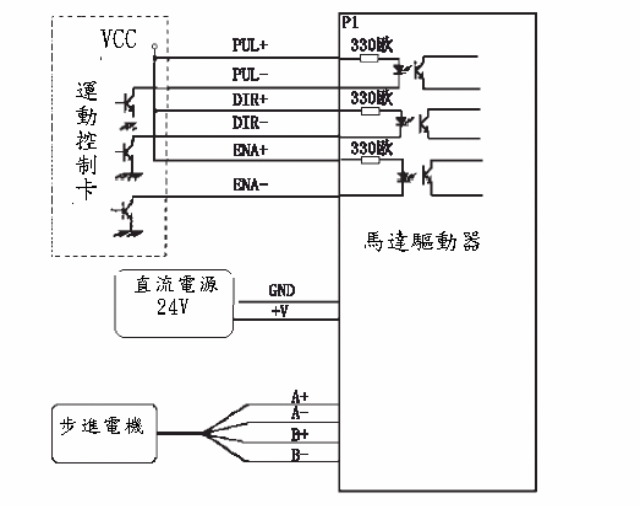

圖 3-2 硬體整合示意圖[14]

該系統包含的三個部份：運動控制卡、步進馬達與可動機台，運動控制卡將迅脈衝訊號、方向訊號以及使能訊號傳給馬達驅動器，馬達驅動器在提供電流驅動步進電機，底下將個別對各個部份做詳細的介紹。

3.2.1 運動控制卡

運動控制卡是將電腦的運動指令轉換成電子脈衝來驅動馬達，本次實驗我們選擇了 PCI 總線高性能的運動控制卡。

圖 3-3 運動控制卡 DMC3000[14]

他提供了使用者在 Windows 95/98/Me/NT/2000/XP 等操作系統環境下的驅

動程式以及運動函數動態聯結庫。運動函數庫是一個運動控制 API 的函數庫，使用者可以利用 C/C++或 Visual Basic 編寫、調用運動函數庫內的相關函數，就能隨意的對多軸步進馬達進行精確、迅速的控制。

利用控制卡來控制馬達的系統架構如下圖所示：

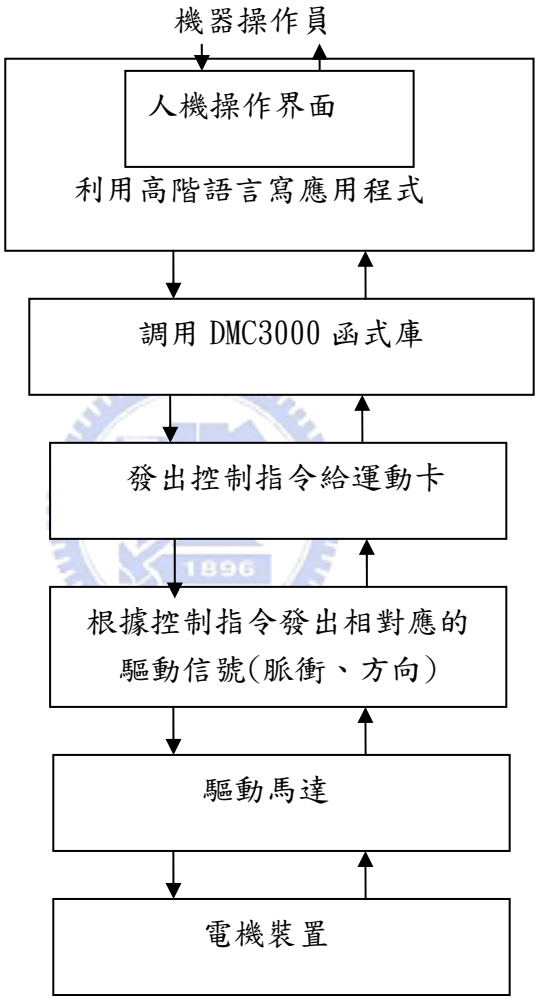

圖 3-4 運動控制卡系統架構

從上頁的示意圖可以看出來，整個系統的工作原理可以簡單分為：

1. 與使用者溝通的人機介面。(包含顯示器、滑鼠以及鍵盤。)

2. 透過介面程式，調用運動函數庫內的相關函數。

3. 運動函數再根據指令發出訊號給 PCI，控制驅動馬達。

4. 驅動馬達接受到指令，轉化為機械的動作。

3.2.2 馬達驅動器

我們採用可細分型步進驅動器 (Microstep)作為實驗的工具，如下圖

圖 3-5 實驗所選取的馬達驅動器 [14]

其工作原理乃是利用脈衝式的電流加於靜止之線圈，線圈纏繞於定子之上，一旦受到電流通過，則會產生磁場使轉子轉動。

一般來說，供電電壓越高電機高速時的力矩越大，越能避免高速時掉步，但電壓太高又會導致過壓，而可能損壞驅動器。

本次實驗所採用的細分型步進驅動器規格如下頁所示：

類型	1.8°的兩相混合式步進電機
驅動器電壓	24V
驅動器電流	3.5 安培以下
精確度	25 倍細分倍數
電機步數	5000 步/圈
最高工作轉速	20000（pluse/s）
加速到最高工作轉速所需時間	100 毫秒

表 3-1 硬體規格

3.2.3 三維運動控制平台

本實驗採用的三維運動控制平台如下圖所示：

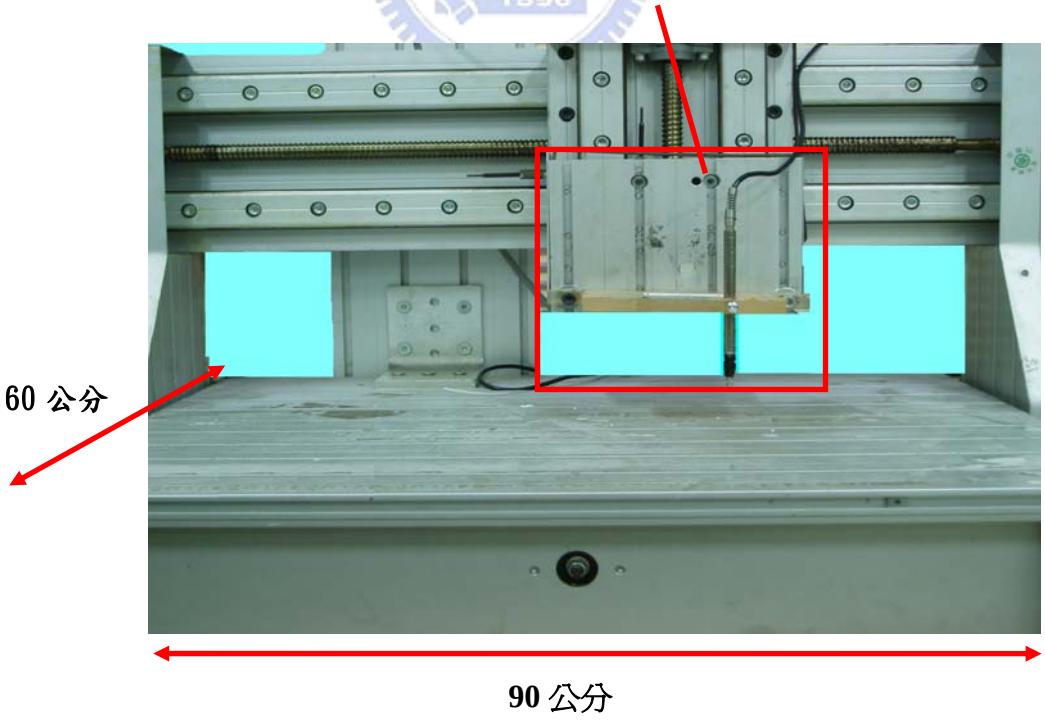

圖 3-6 三維運動控制平台圖［14］

三維運動控制平台長 90 公分，寬 60 公分，高 30 公分，由三組單軸滾珠螺桿組合而成。

當前級脈衝驅動馬達的齒輪轉動時，會帶動接於其上之單軸滾珠螺桿；螺桿利用機械原理，將旋轉運動轉換成直線運動，進而移動其上之模擬曝光工具。

本實驗中，前級脈衝驅動馬達的齒輪齒數為 200，且設定為 25 倍細分倍數。可得知一個脈衝將會讓轉子轉動 0.072 度，想要讓轉子轉動一周，需 5000 個脈衝(pulse)。轉子轉動一周可移動平台 4000um，因此，電腦發出的脈衝與平台移動的關係為 0.8(um/pluse)。[9]

$$\theta = \frac{360^\circ}{200 \times 25} = 0.072 \quad \cdots\cdots \text{(式 3-1)}$$

$$200 \times 25 = 5000 \quad \cdots\cdots \text{(式 3-2)}$$

$$\frac{4000\,(\mu\text{m})}{5000(\text{plus})} = 0.8(\text{um/pluse}) \quad \cdots\cdots \text{(式 3-3)}$$

3.3 軟體規劃與設計

軟體設計是可以使用各種不同技巧來達成目標，使用技巧的時機與選擇就端看系統分析者的構思及創意，但是有一點必須注意的就是一次只解決一個問題，不要同時將多個問題並在一起混著談。因此本小節前半部將解決「可移動機台的圖控運動介面」，後半部將驗證「可讀取 GDSII 規格的圖形並以讓載具以等速率運動完成該圖形之功能」。

為了控制運動卡，需要讓 LabVIEW 能夠呼叫該運動卡所提供的函數，其中

資訊交換的格式必須設定正確，否則將造成錯誤，在 3.3.1 將會提到該如何設定彼此間交換資訊的格式。

3.3.1 移動運動平台之介面及程式設計

圖 3.1 為此移動運動平台之介面，此介面具有最基本的和運動卡傳輸與擷取指令的功能，可讓電腦透過此介面操作運動平台動作，其功能可分為以下幾個部份。

1. 移動選擇區塊：用來設定移動軸，可自動執行勾選的移動軸進行插補。
2. 運動方式選擇區塊：可執行回原點、連續運動、定長運動、直線插補、圓弧插補等運動。
3. 運動參數設置區塊：包含了各軸運動參數設置、直線運動參數設置以及插補速度設置三大部分，依據選擇的運動方式，在相對應的部份輸入參數。
4. 模式設定區塊：包含位移方式選擇、停止方式選擇、加速曲線選擇、圓弧方向選擇，這些參數的設置影響運動的執行方式。
5. 狀態報告區塊：用來顯示目前載具的移動速率，座標之查詢回應。
6. 指令區塊：包含電源開關、運動控制這兩部分，使用者可以利用這區塊對載具下達命令，就算在移動中也可以執行指令，舉例來說，動態加減速就是讓載具在運動中加減速的按鈕。

圖 3-7 移動運動平台介面

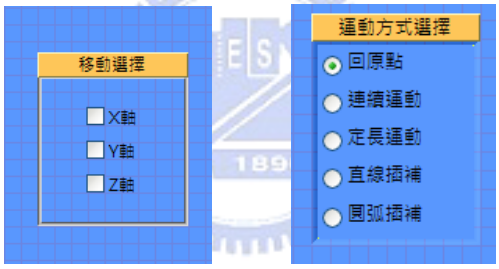

圖 3-8 移動選擇區塊　　運動方式選擇區塊

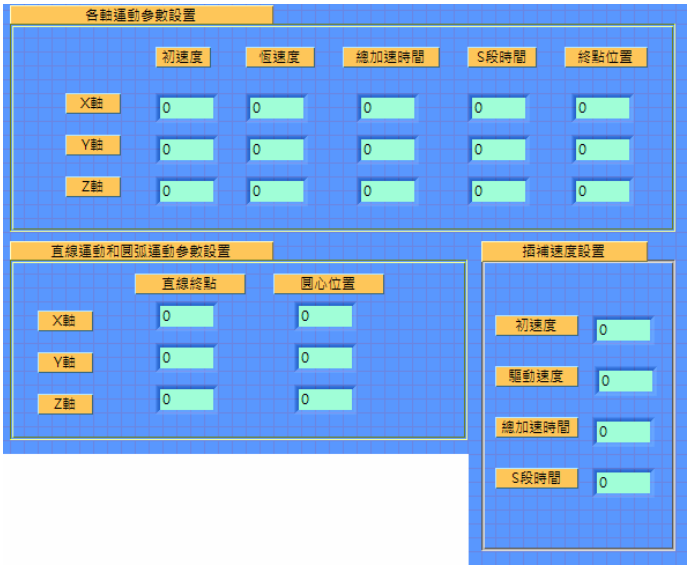

圖 3-9 運動參數設置區塊

圖 3-10 模式設定區塊

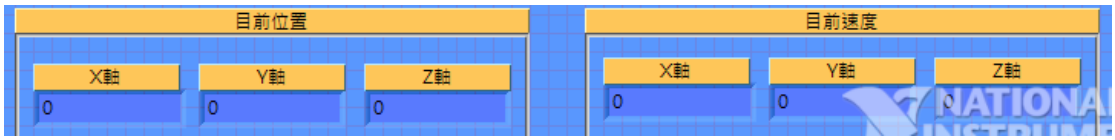

圖 3-11 狀態報告區塊

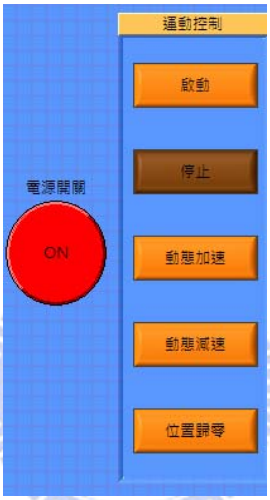

圖 3-12 指令區塊

此運動介面 LabVIEW 之設計流程，其中使用到了 Call function node 這個功能。

圖 3-13 Call function node 方塊圖

由於要透過 LabVIEW 來實現移動的目標，就必須能讓 LabVIEW 能夠呼叫該運動卡所提供的函數才行，LabVIEW 提供的一個功能讓使用者可使用 LabVIEW 來呼叫 Windows 底下既有的動態連結檔(DLL)或者其他平台下的共享程式庫，即是

Call function node，其設定如下圖。

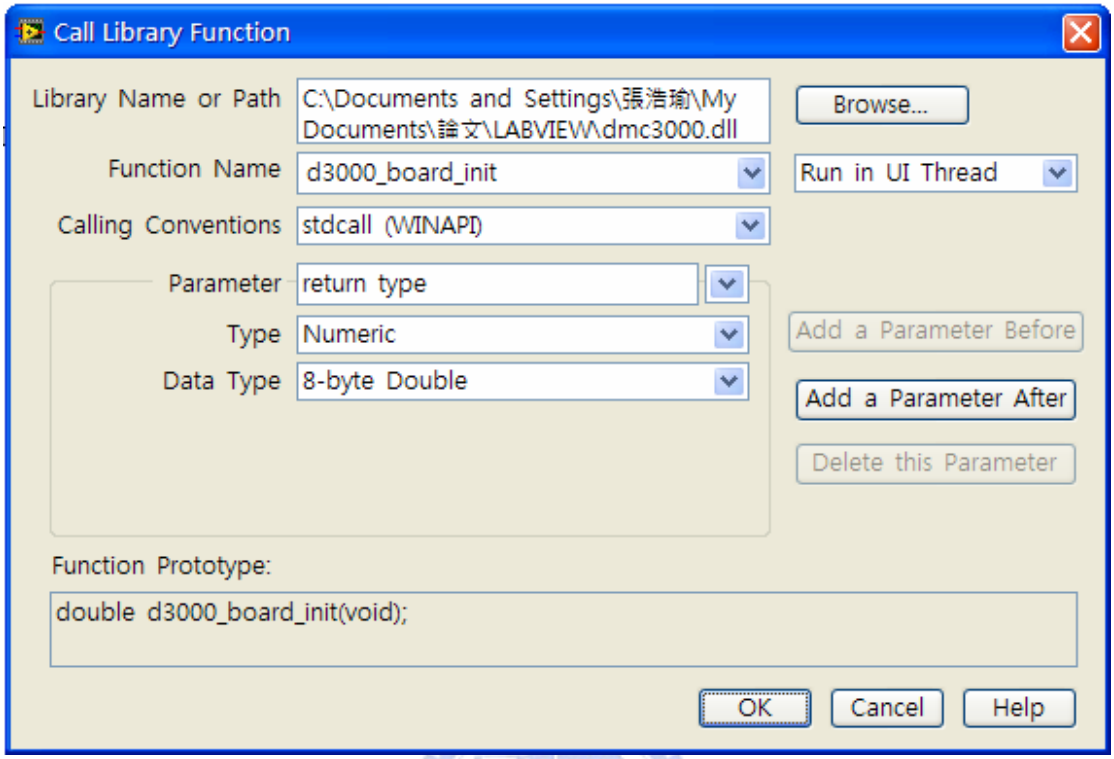

圖 3-14 Call function node 設定圖

首先將第一個欄位 Library Name or Path 指引到你欲使用的 DLL 檔案所在位置。第二欄位 Function Name 是一個下拉式選單，只要從選單中挑出你欲使用的功能即可。第三欄 Calling Conventions，由於我們是要和硬體做溝通連動，因此請選擇 stdcall (WINAPI)，若選擇 C 則會造成當機。右手邊請選擇 Run in UI Thread。Parameter 欄請先設定 Return type 傳回值為何種資料型態，利用底下 Type 以及 Data Type 來選擇回傳的資料型態。若是該功能需要輸入參數，則按右邊的 Add a parameter after，那他會在 Parameter 欄的下拉式清單新增一個參數，再如同之前進行參數的資料型態設定即可。完整設定好之後，該功能會以一個子程式的型態顯示於最下面那欄，讓你可以檢查參數的資料型態是否正確。

底下是運動控制介面程式設計流程圖：

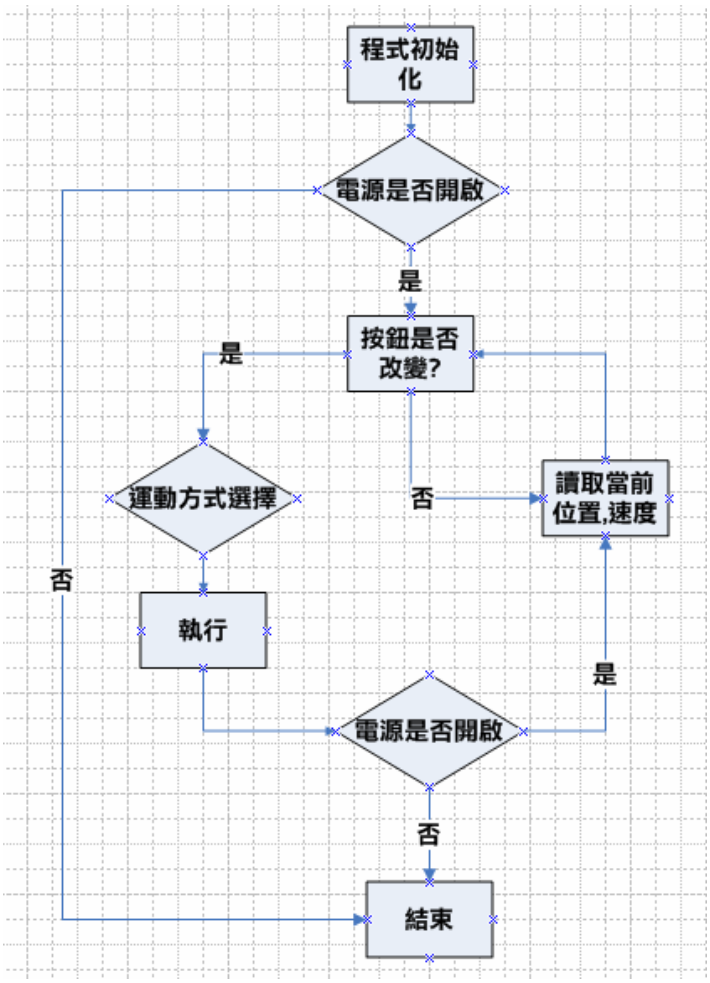

圖 3-15 介面程式設計流程圖

程式初始化：載入以及啟始介面卡。

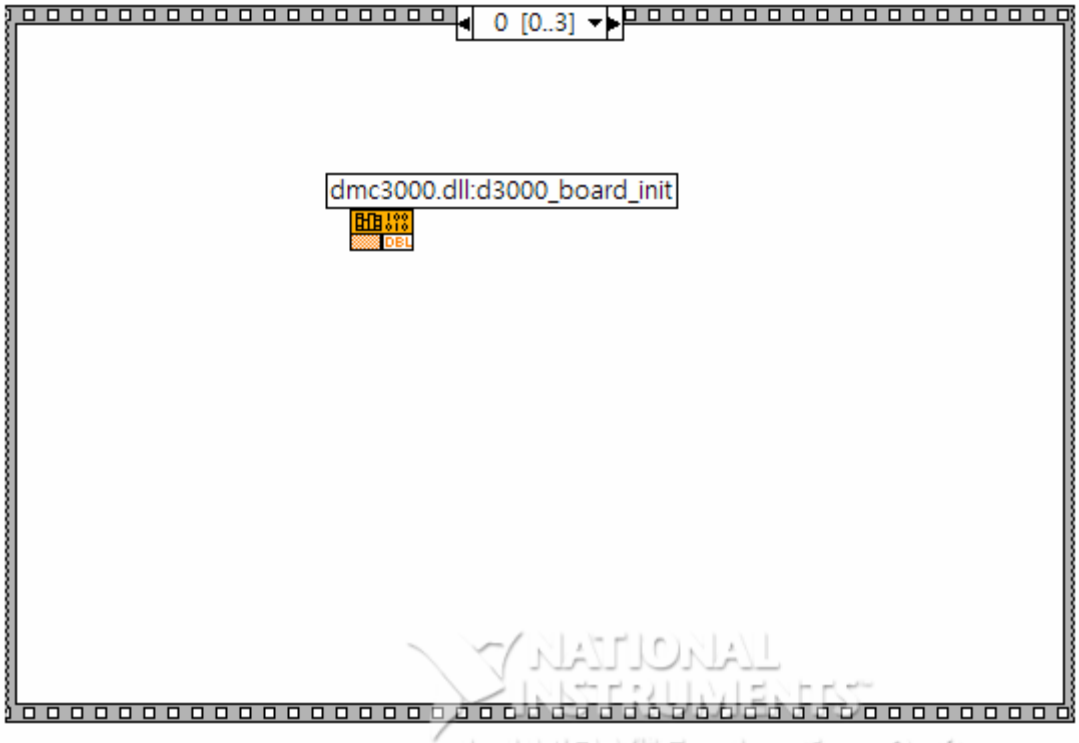

圖 3-16 LabVIEW 程式流程-啟始介面卡(1)

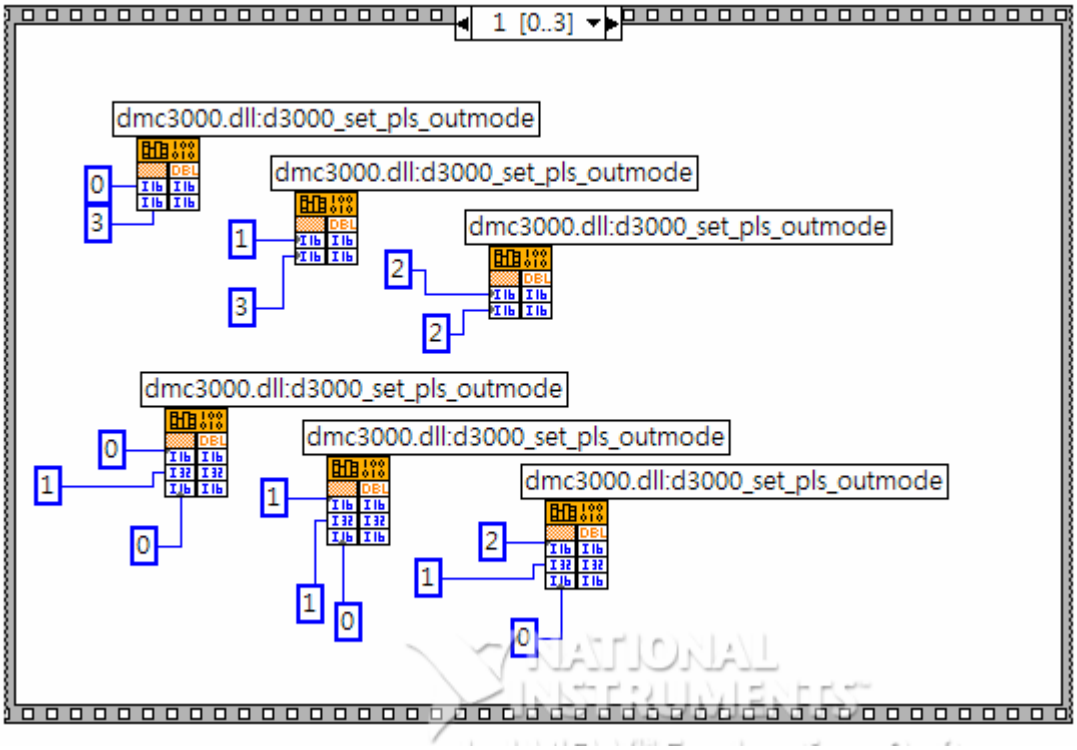

圖 3-17 LabVIEW 程式流程-設定參數(2)

這部份由於圖像龐大，故用下圖來表示解說區域：

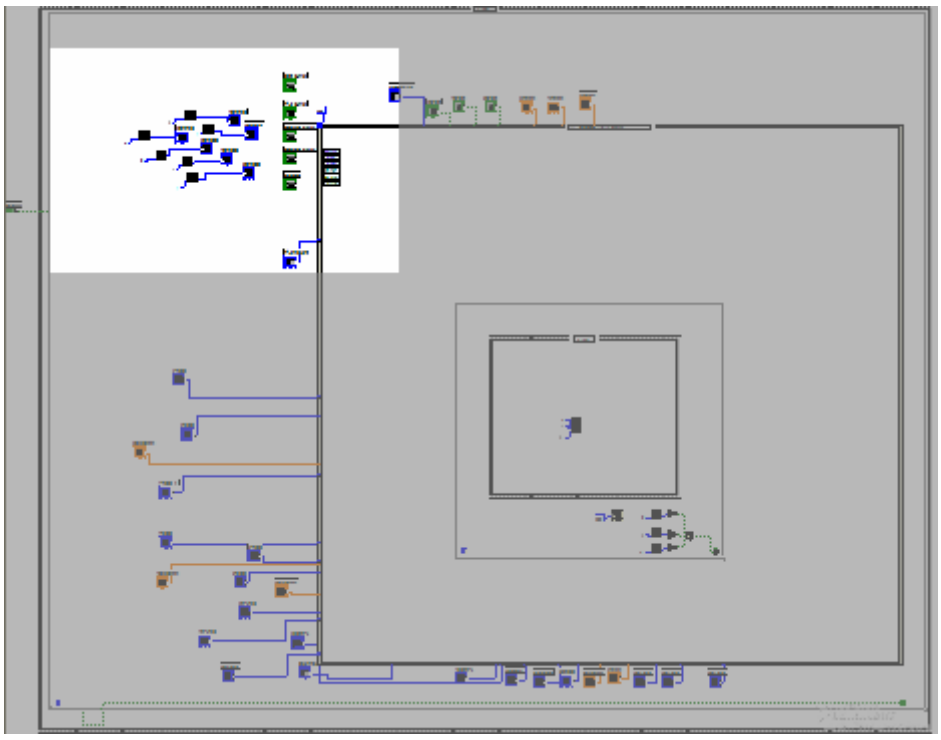

圖 3-18 LabVIEW 程式區域

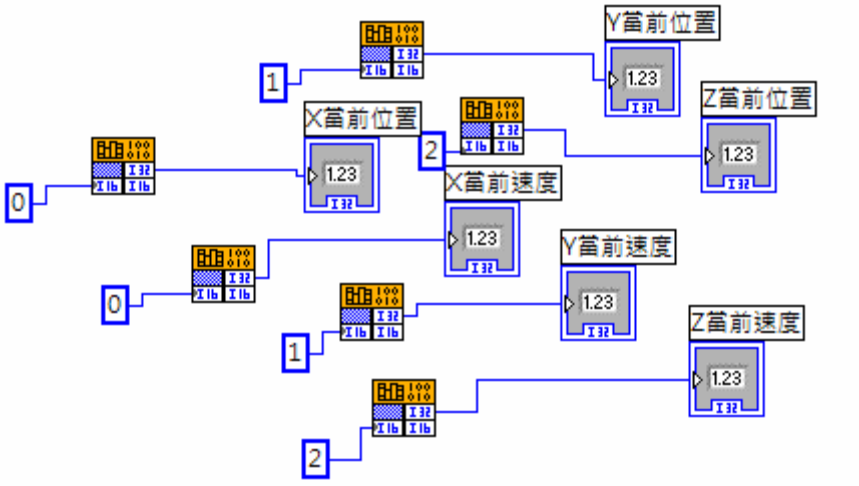

圖 3-19 LabVIEW 程式流程-讀取位置速度(3)

「讀取當前位置和速度」之部分放於 While 回圈內，除非電源切回 Off 否則將不斷進行讀取位置和速度的工作。

接下來我們將介紹中間區塊的 Event Structure 內包含的程式，Event

Structure 本身 Time out 設定為 0.5 秒其程式為空白，意即只要沒有輸入任何指令 0.5 秒，就會再回去做讀取位置和速度的動作。

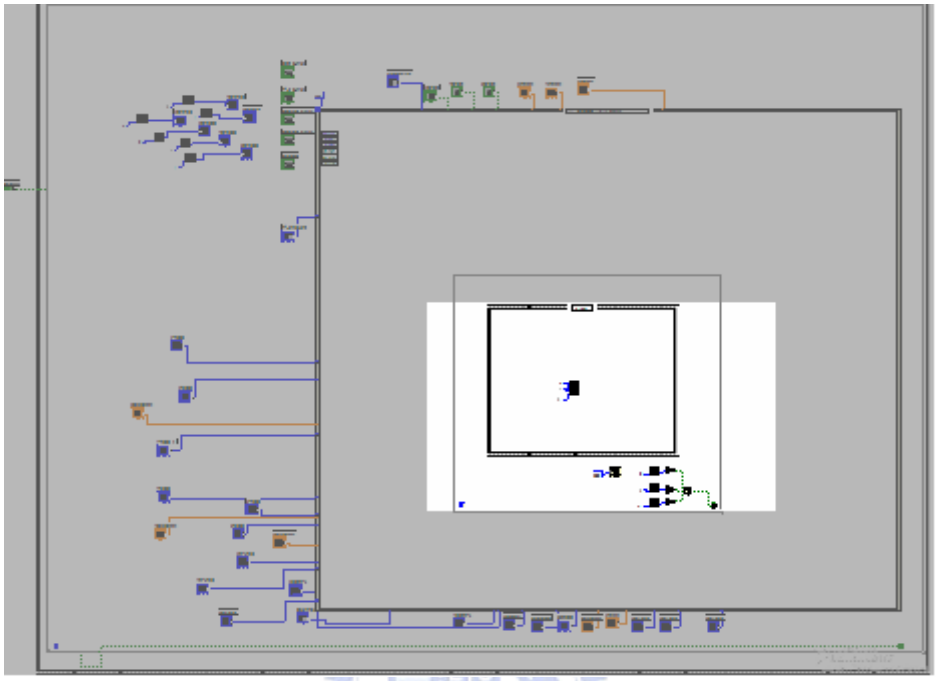

圖 3-20 LabVIEW 程式區域

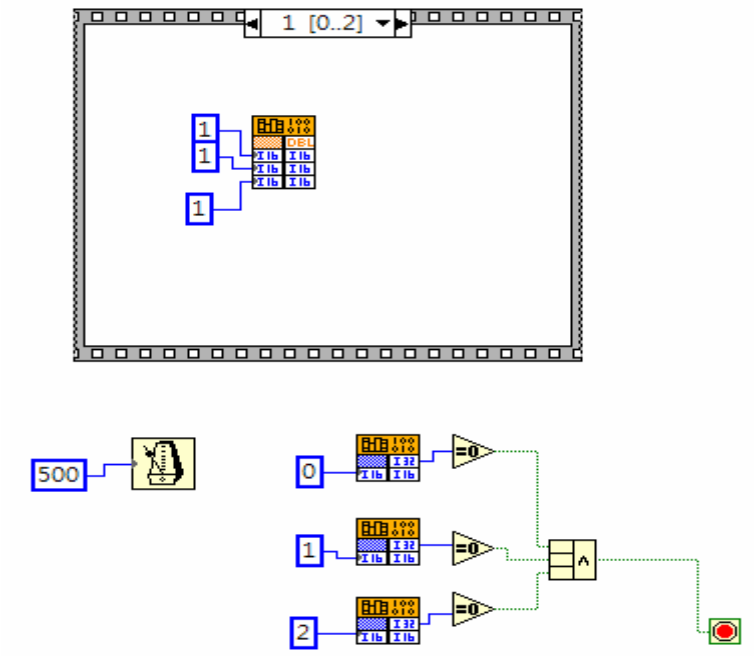

圖 3-21 LabVIEW 程式流程-位置歸零(4)

在下達當前所在位置歸零指令後，程式會確認三個軸都沒有在運動中，才

會對分別把三個軸所在位置設定為原點。

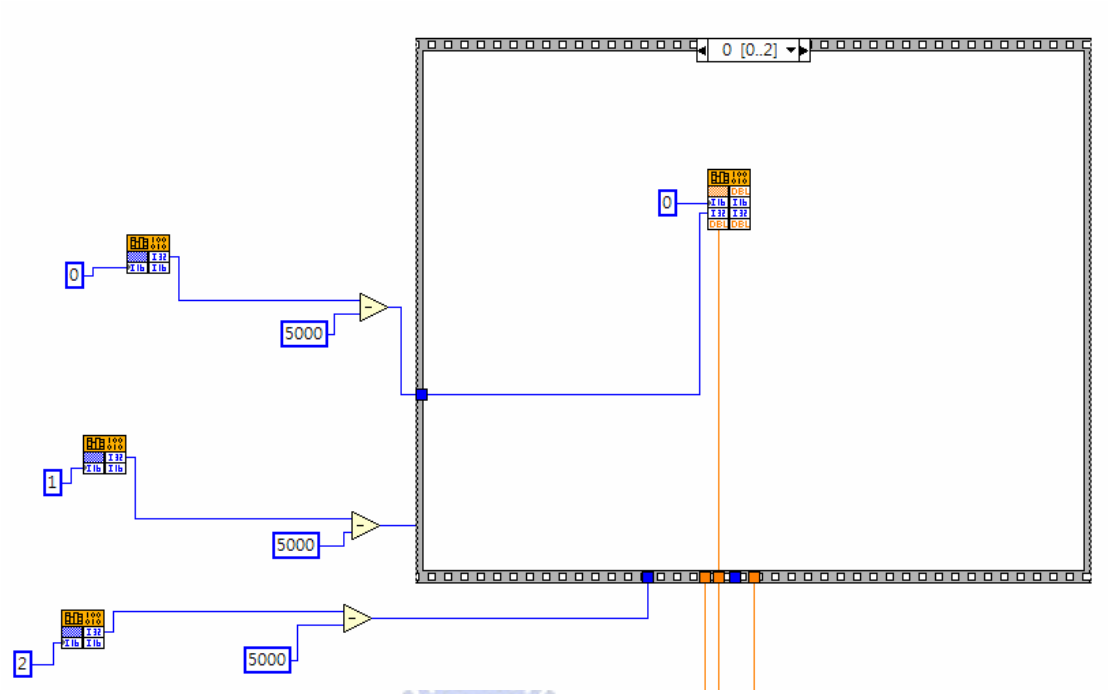

圖 3-22 LabVIEW 程式流程-動態減速(4)

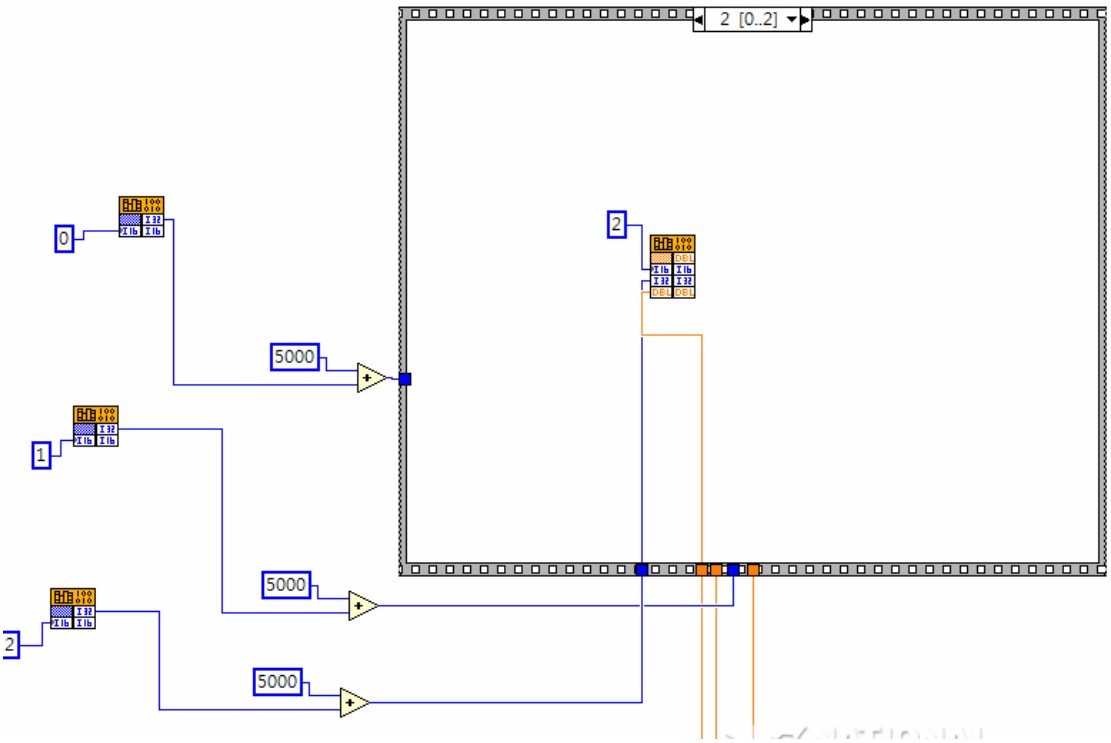

圖 3-23 LabVIEW 程式流程-動態加速(5)

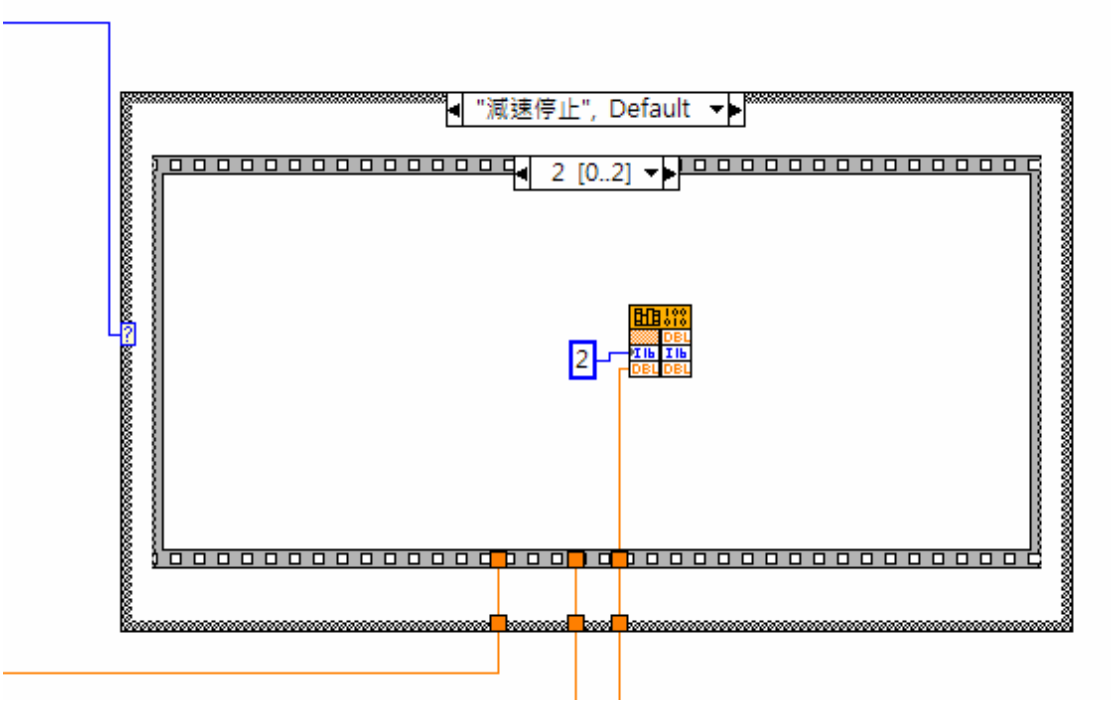

圖 3-24 LabVIEW 程式流程-停止(6)

當按下停止鈕時，系統會判讀停止的方式：是減速停止還是緊急停止？在選用對應的 Call function node。

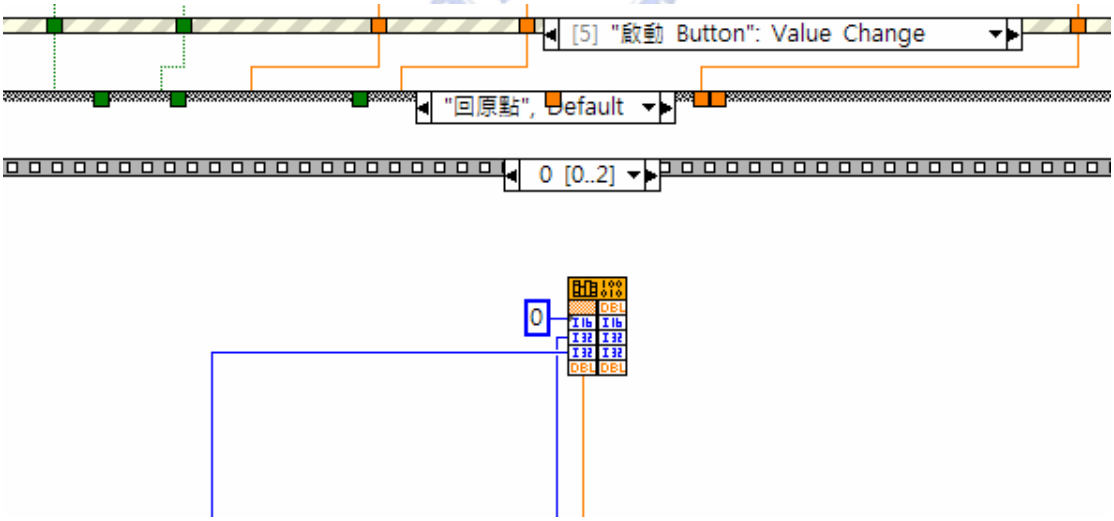

圖 3-25 LabVIEW 程式流程-啟動，回原點(7)

下達啟動指令後，程式會檢查你所選的運動模式，上圖為選擇回原點時之程式。

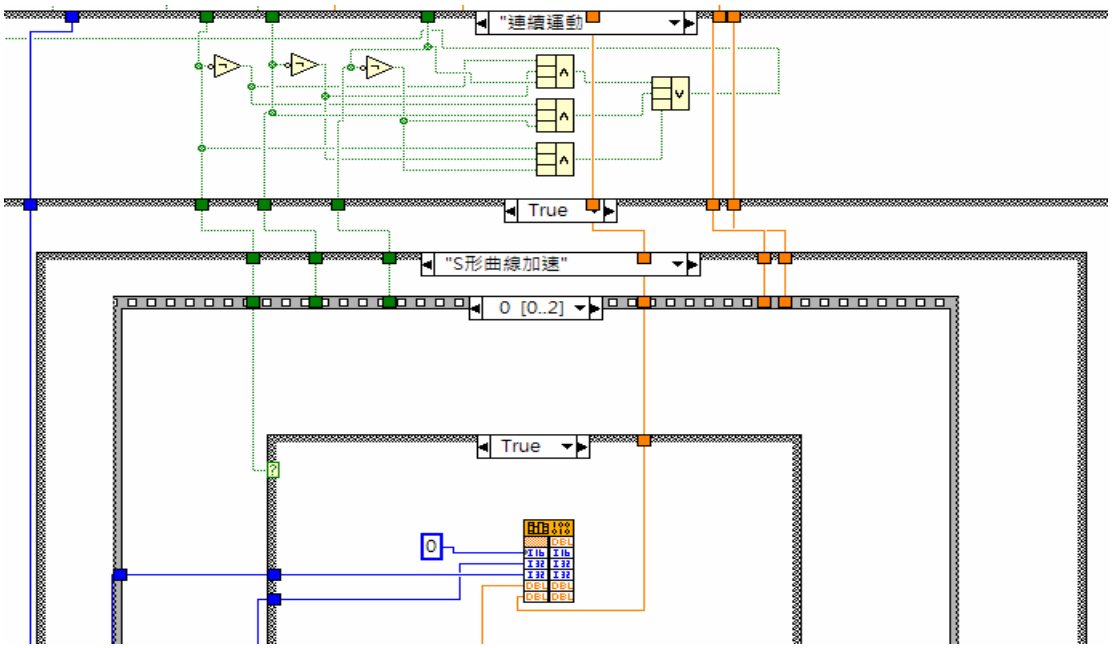

圖 3-26 LabVIEW 程式流程-啟動，連續運動(8)

當選擇了連續運動時，包含了 S 型、梯型曲線加速、X 軸、Y 軸或 Z 軸運動等等可能性。同理，在定長運動也是同連續運動一樣的考量，只是要再多絕對運動或是相對運動的判斷。

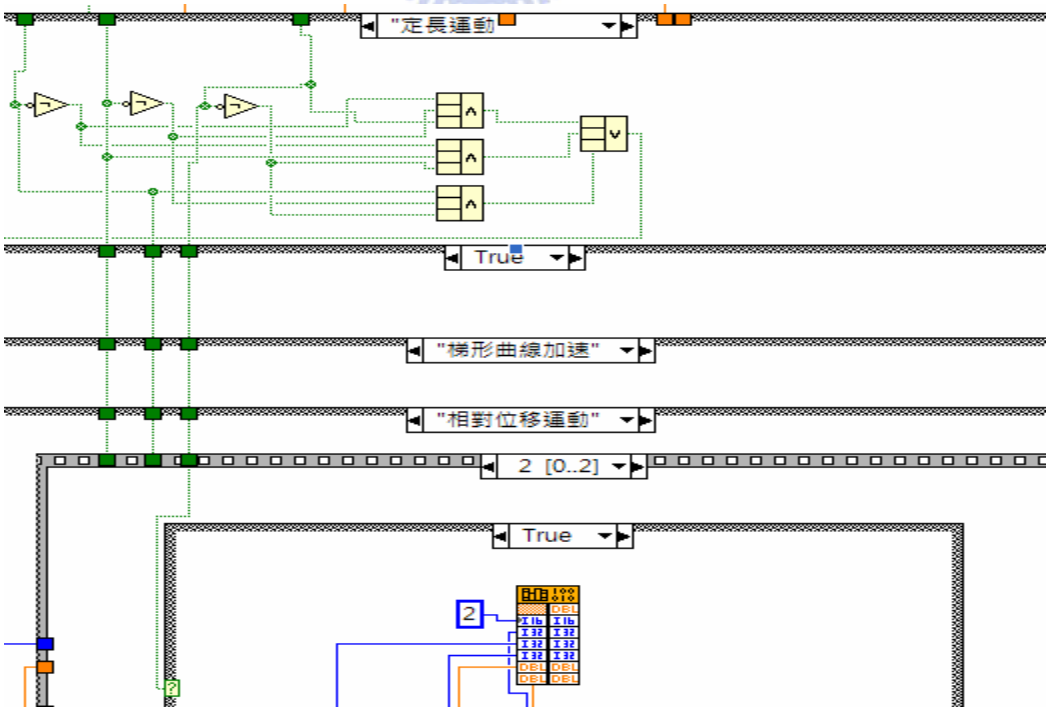

圖 3-27 LabVIEW 程式流程-啟動，定長運動(9)

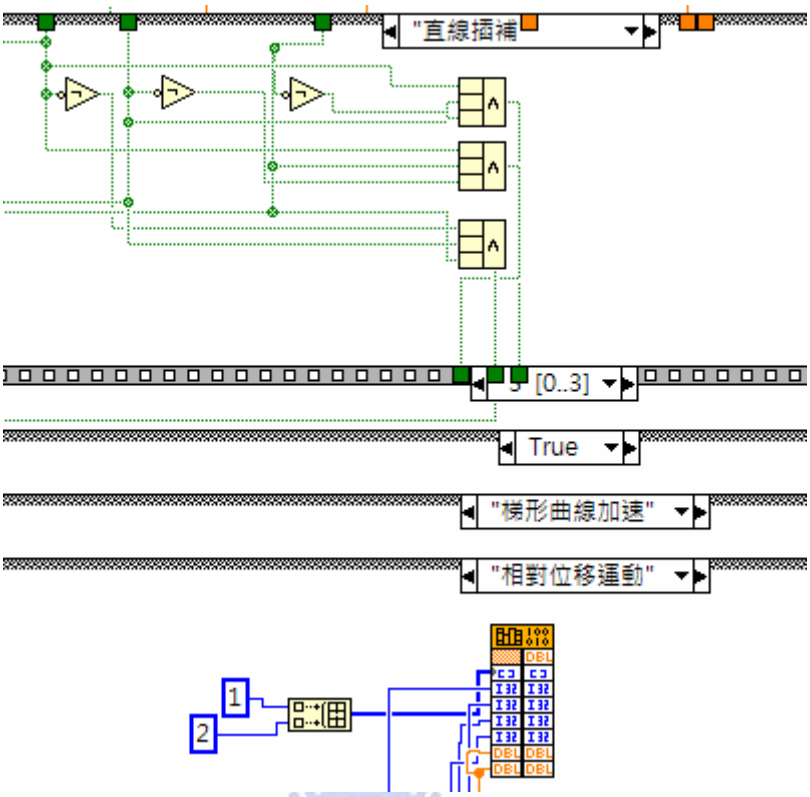

圖 3-28 LabVIEW 程式流程-啟動，直線插補(10)

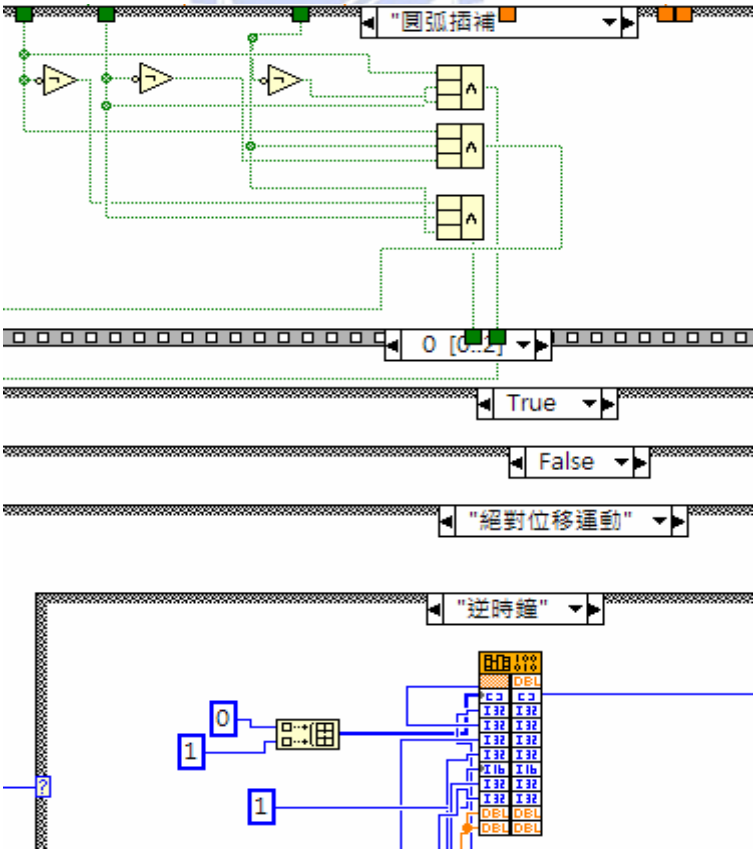

圖 3-29 LabVIEW 程式流程-啟動，圓弧插補(11)

當 Event Structure 執行完。(Time out 或是執行完任一個功能後) 系統將會檢查電源開關是否為 Off,若為 Off 則執行最後一個動作:關閉運動介面卡。

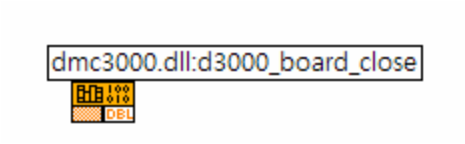

圖 3-30 LabVIEW 程式流程-關閉介面卡(12)

3.3.2 可接受圖型檔運動之介面及程式設計

下圖為可接受 GDSII 規格之圖控運動介面。

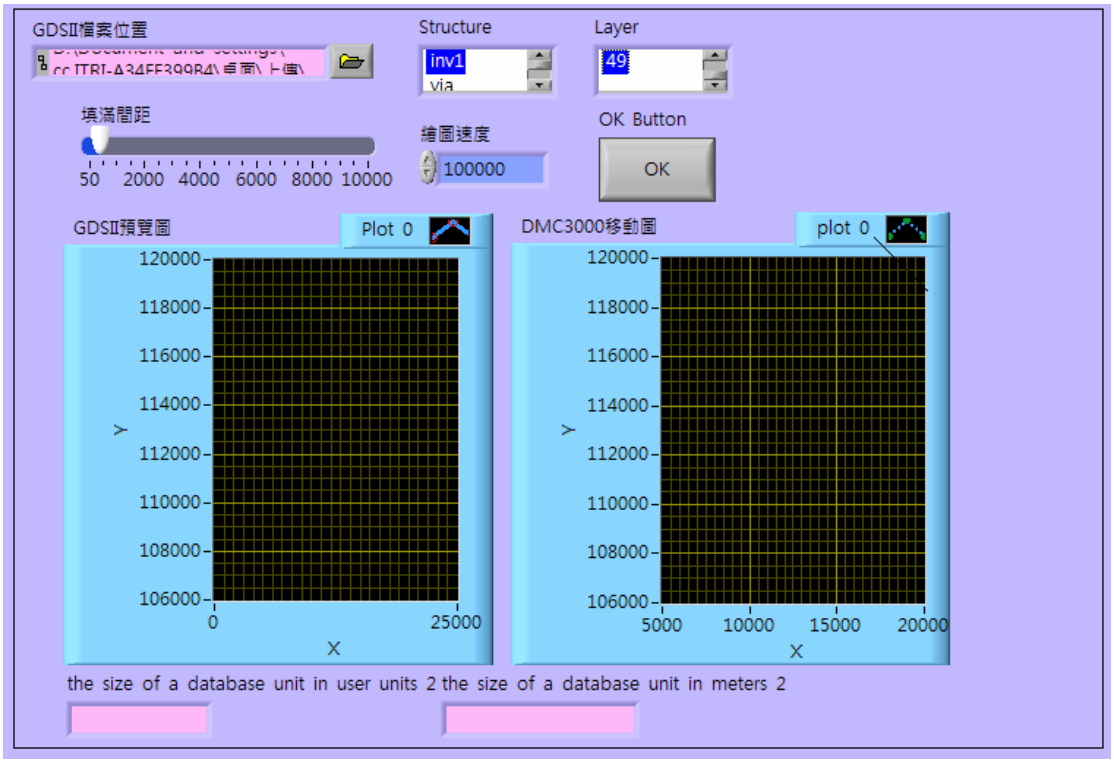

圖 3-31 可接受 GDSII 規格運動之介面設計

其功能可分為以下幾個部份:

1. 檔案輸入區塊:介面開始執行以後,點擊右方的 圖示會彈出一個路徑選擇視窗,在該視窗選擇欲執行的 GDS II 檔案,程式會自動將選擇的檔案

路徑顯示於  左側以供辨識。選擇檔案後，可以挑選欲繪製的結構(Structure)和圖層(Layer)。

2. 參數輸入區塊：當選擇了欲執行的 GDS II 檔後，要進行一些參數的設定，參數有兩部份：填滿密度以及繪圖速度。填滿間距意指當 GDS II 指定要繪製一個區域 (Boundary)的時候，載具會把區域內部畫上交錯的直線來填滿，當填滿間距愈大，則線條愈稀疏，使用者可以參考儀具上搭載的繪圖工具來選擇適當的填滿間距。繪圖速度意指該運動被執行的速度，數值愈大，則載具運動愈迅速。

3. 訊息回饋區塊：這部份包含了 GDS II 預覽圖，DMC 3000 移動圖以及該 GDS II 的真實比例資訊，讓使用者可以透過圖形顯示，明瞭目前繪製的圖形。

圖 3-32 檔案輸入區塊

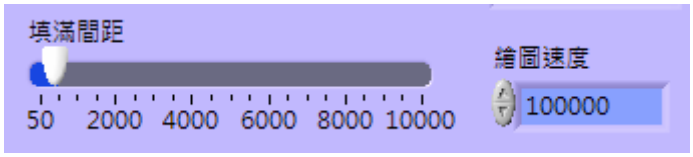

圖 3-33 參數輸入區塊

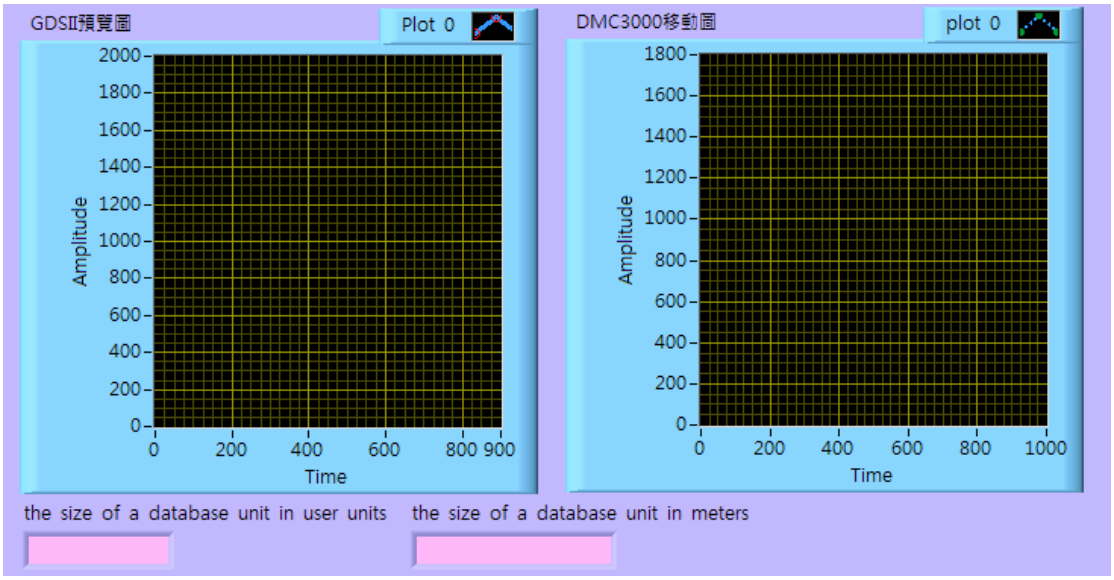

圖 3-34 訊息回饋區塊

在開始介紹程式的設計流程之前，要先和大家解釋一下 GDS II 的規則語法，透過免費軟體 Owlvision 我們可以將 GDS II 由十六進位編碼轉譯成可識別的 ASCII File，底下列舉一個範例 ASCII File 作為說明：

```
HEADER 5; # version
BGNLIB;
LASTMOD {98-8-25  15:53:12}; # last modification time
LASTACC {98-8-25  15:53:12}; # last access time
LIBNAME TEMPEGS.DB;
UNITS;
USERUNITS 0.01; PHYSUNITS 1e-08;

BGNSTR; # Begin of structure
CREATION {98-8-25  15:53:12}; # creation time
LASTMOD  {98-8-25  15:53:12}; # last modification time
STRNAME AAP;

BOUNDARY; LAYER 1;  DATATYPE 0;
   XY 5;
   X     -920000.000; Y      452000.000;
   X      656500.000; Y      765500.000;
   X      175000.000; Y     -174000.000;
```

```
   X     -756000.000; Y     -198000.000;
   X     -920000.000; Y      452000.000;
ENDEL;

ENDSTR AAP;

BGNSTR; # Begin of structure
CREATION {98-8-25  15:53:12}; # creation time
LASTMOD  {98-8-25  15:53:12}; # last modification time
STRNAME LAYOUT;

SREF;
SNAME AAP;
STRANS 0,0,0;
   XY 1;
   X    -1112500.000; Y    -1267000.000;
ENDEL;

PATH; LAYER 3;
   XY 4;
   X      891912.000; Y     2322024.000;
   X      966537.000; Y     1854278.000;
   X     2599515.000; Y     2311647.000;
   X     2626485.000; Y     2005353.000;
ENDEL;

TEXT; LAYER 3; TEXTTYPE 0; PRESENTATION 0,2,0; PATHTYPE 1;
STRANS 0,0,0; MAG 1875;
  XY 1;    X    -2256500.000; Y     1539500.000;
STRING "Boundary";
ENDEL;

ENDSTR LAYOUT;
ENDLIB;
```

[15]

其中我們會用到的幾組關鍵指令為BGLIB，ENDLIB；USERUNITS，PHYSUNITS；

BOUNDARY，PATH，ENDEL；XY。第一組指令宣告了整個程式的開始以及結束。第

二組告訴使用者該圖示的真實比例為多少。第三組宣告了繪圖的種類以及繪圖的開始與結束，最後一組是告訴我們圖形的組成座標。

利用以上幾組關鍵指令，我們開始設計該介面的運作流程，下圖為運動控制介面程式流程圖：

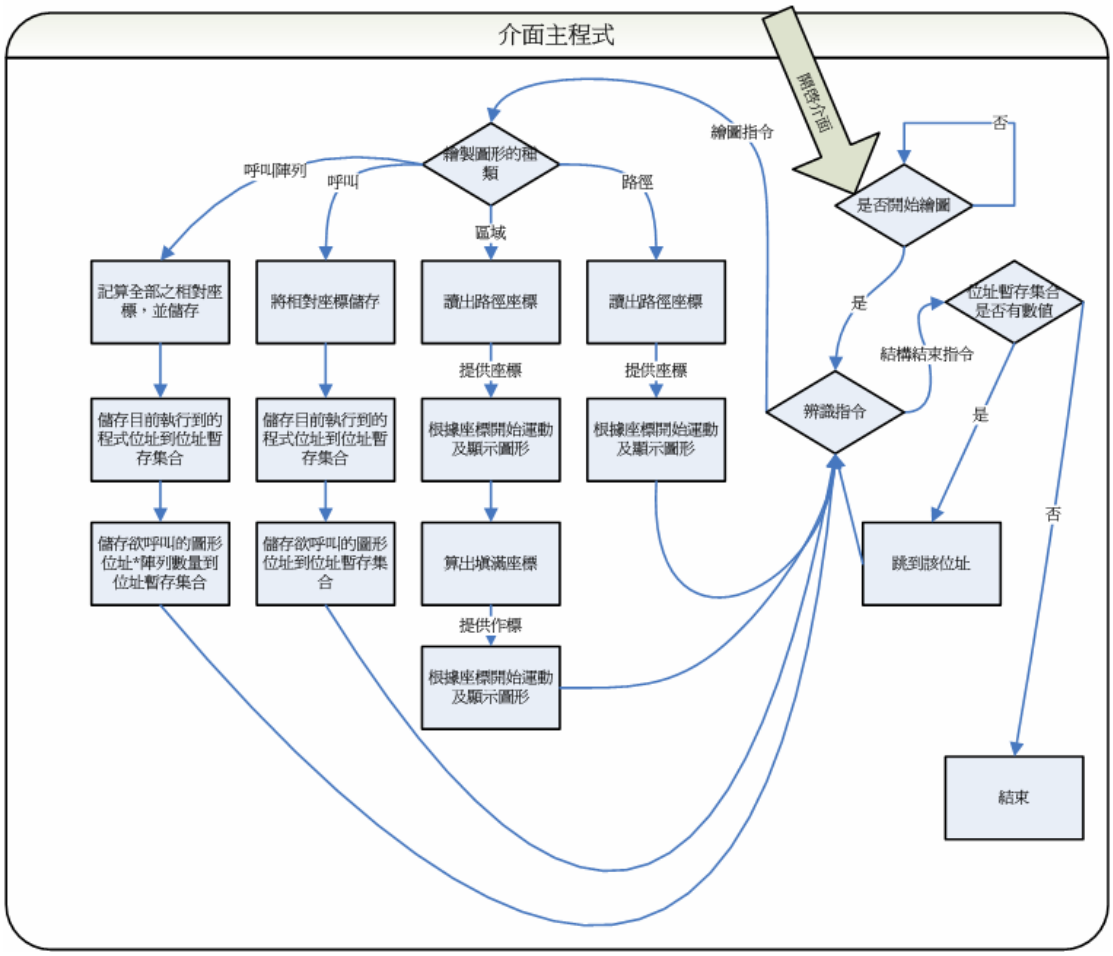

圖 3-35 GDSII 規格運動控制介面程式流程圖

由圖到圖顯示 LabVIEW 程式流程。

第一部分為初始化數值：

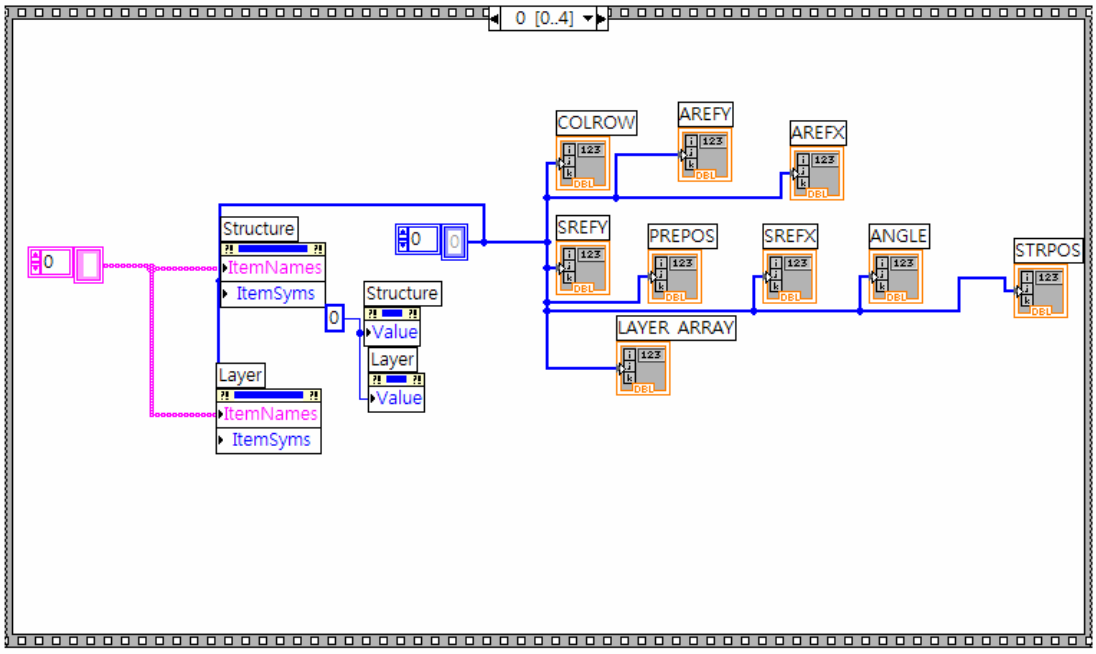

圖 3-36 GDSII 規格運動控制介面 LabVIEW 流程圖-初始化程式(1)

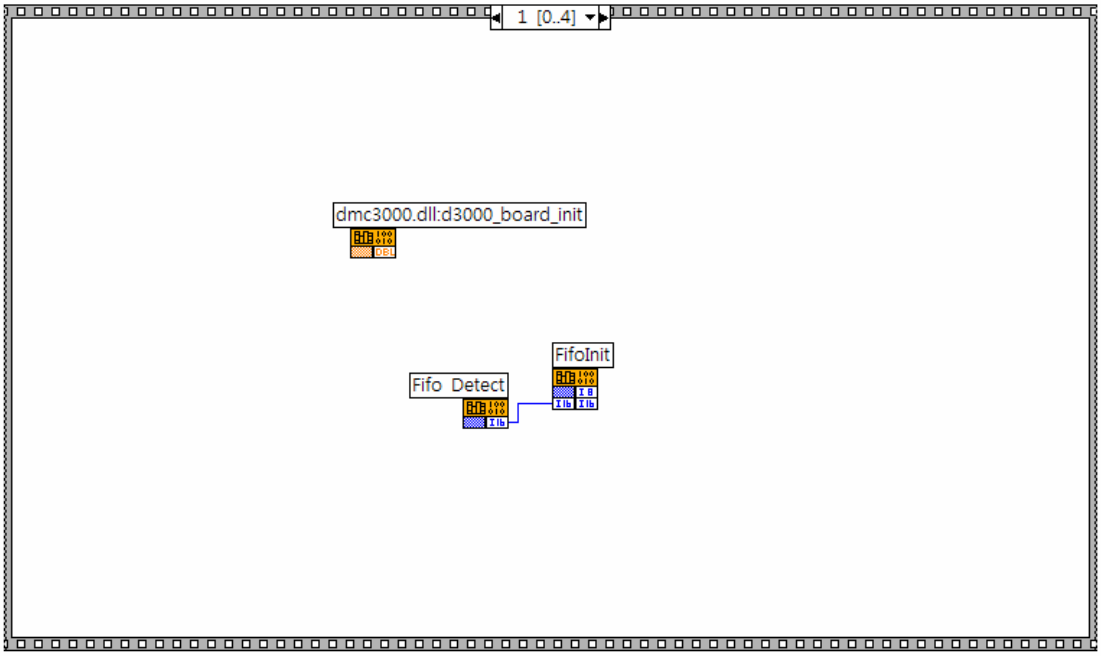

圖 3-37 GDSII 規格運動控制介面 LabVIEW 流程圖-初始化程式(2)

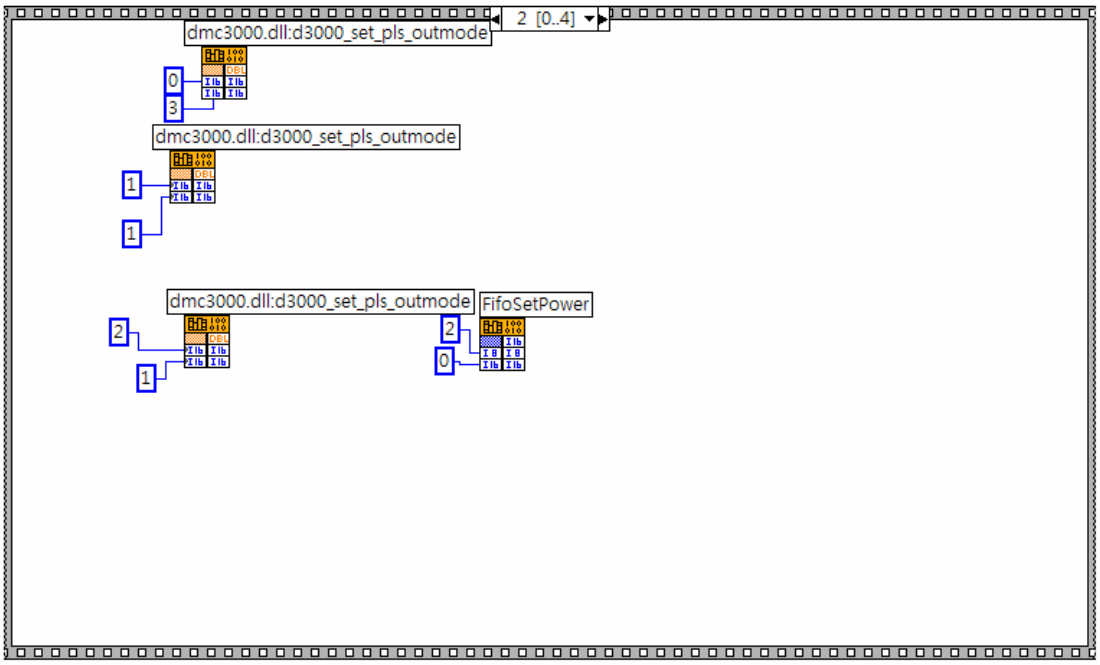

圖 3-38 GDSII 規格運動控制介面 LabVIEW 流程圖-初始化程式(3)

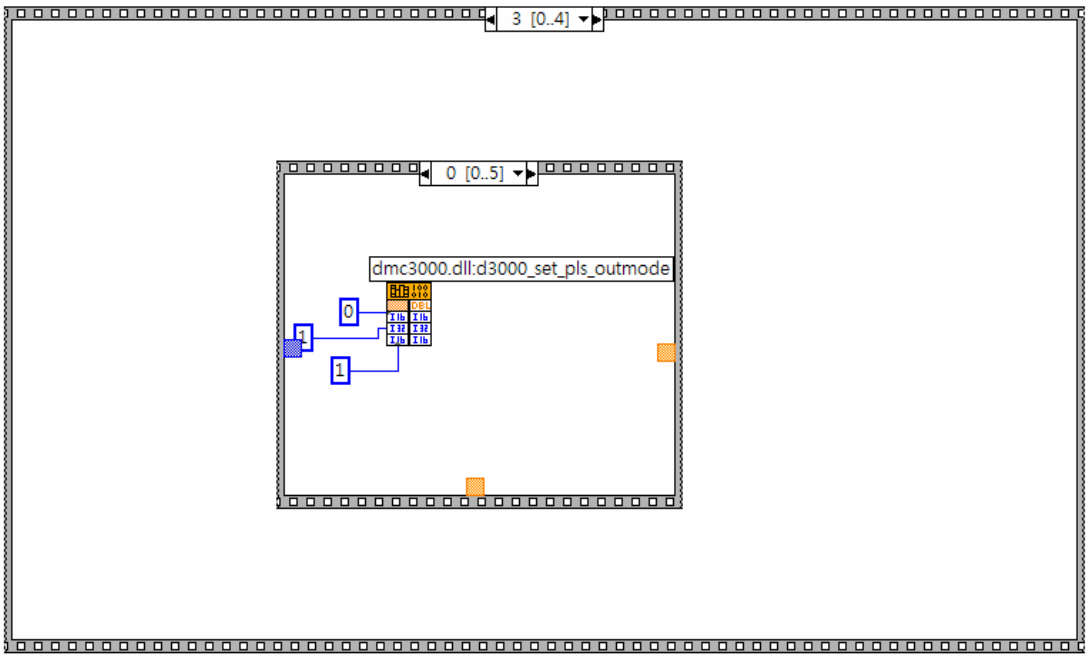

圖 3-39 GDSII 規格運動控制介面 LabVIEW 流程圖-歸原點(4)

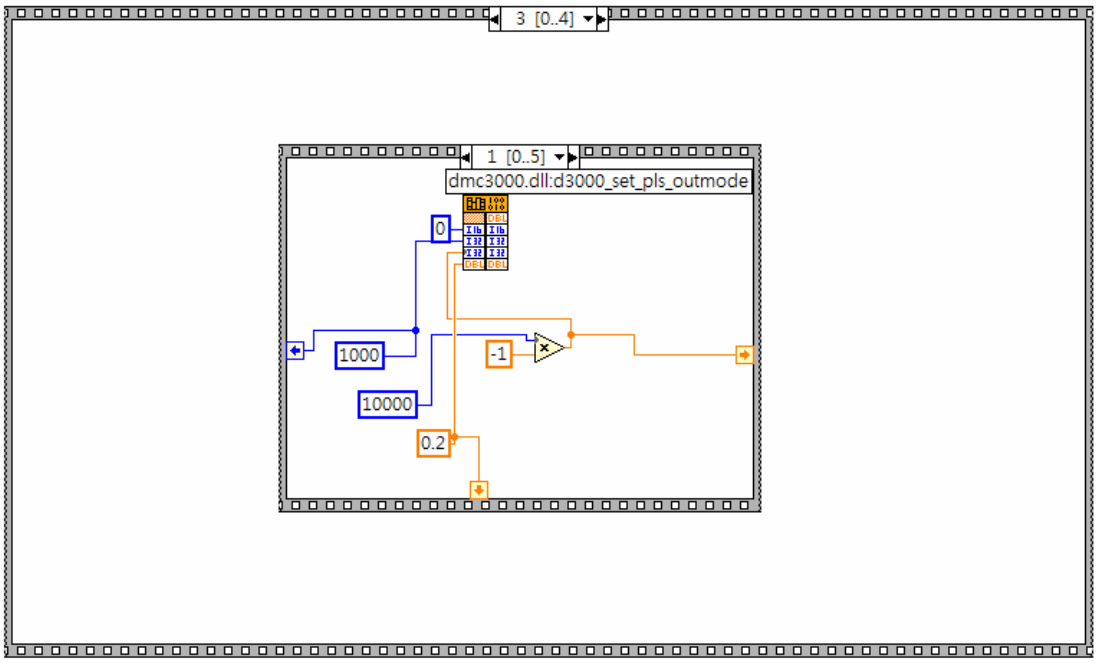

圖 3-40 GDSII 規格運動控制介面 LabVIEW 流程圖-歸原點(5)

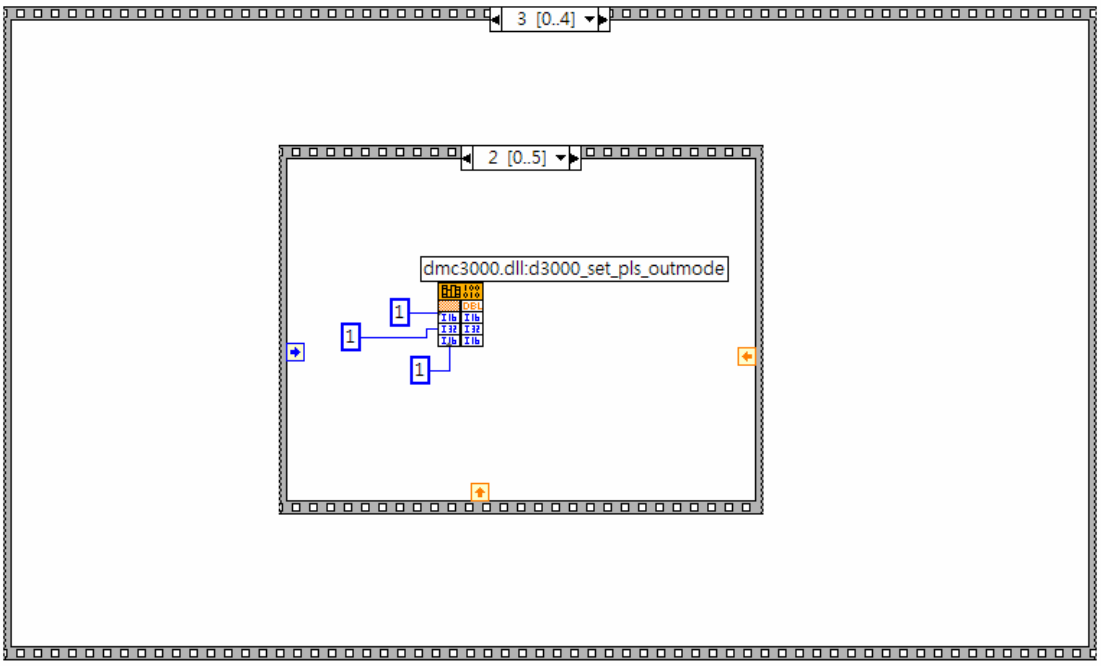

圖 3-41 GDSII 規格運動控制介面 LabVIEW 流程圖-歸原點(6)

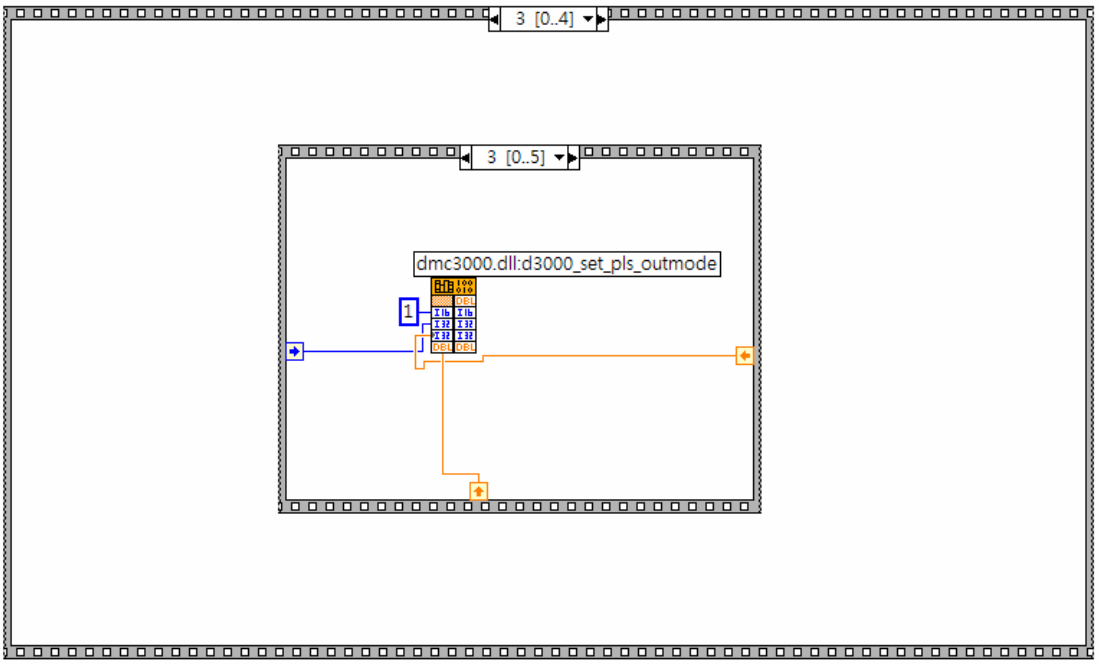

圖 3-42 GDSII 規格運動控制介面 LabVIEW 流程圖-歸原點(7)

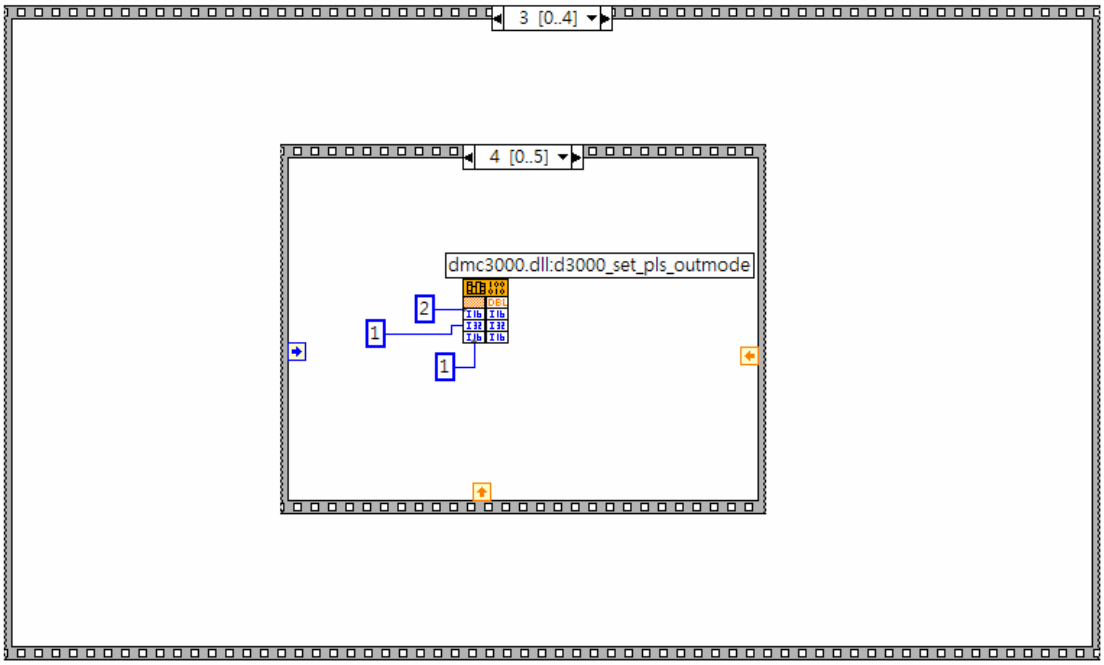

圖 3-43 GDSII 規格運動控制介面 LabVIEW 流程圖-歸原點(8)

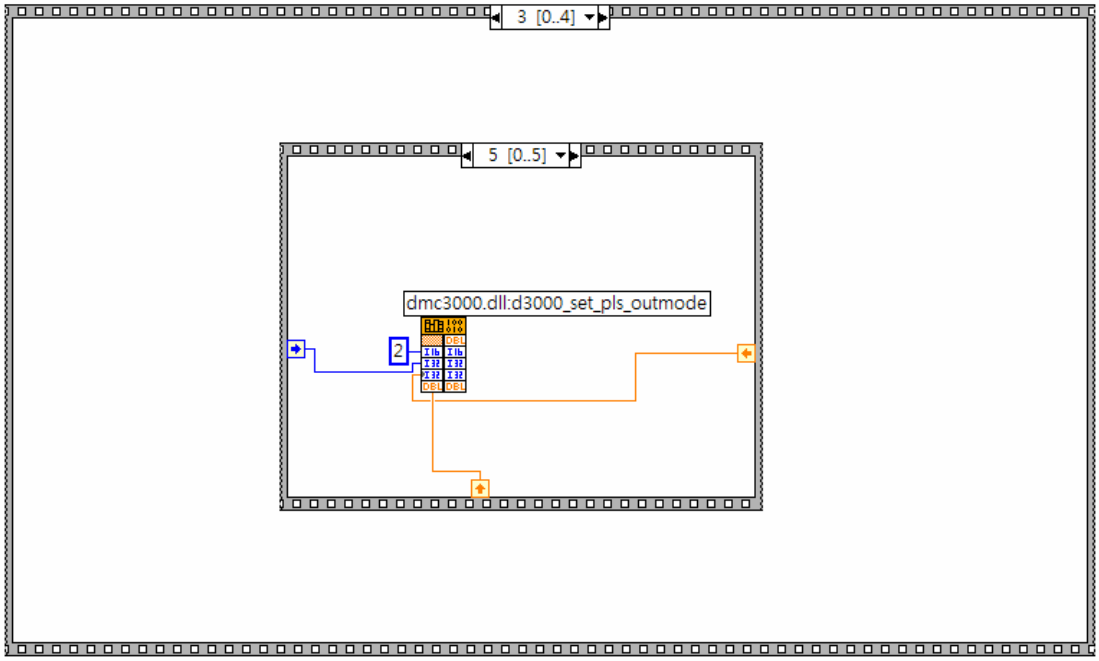

圖 3-49 GDSII 規格運動控制介面 LabVIEW 流程圖-歸原點(9)

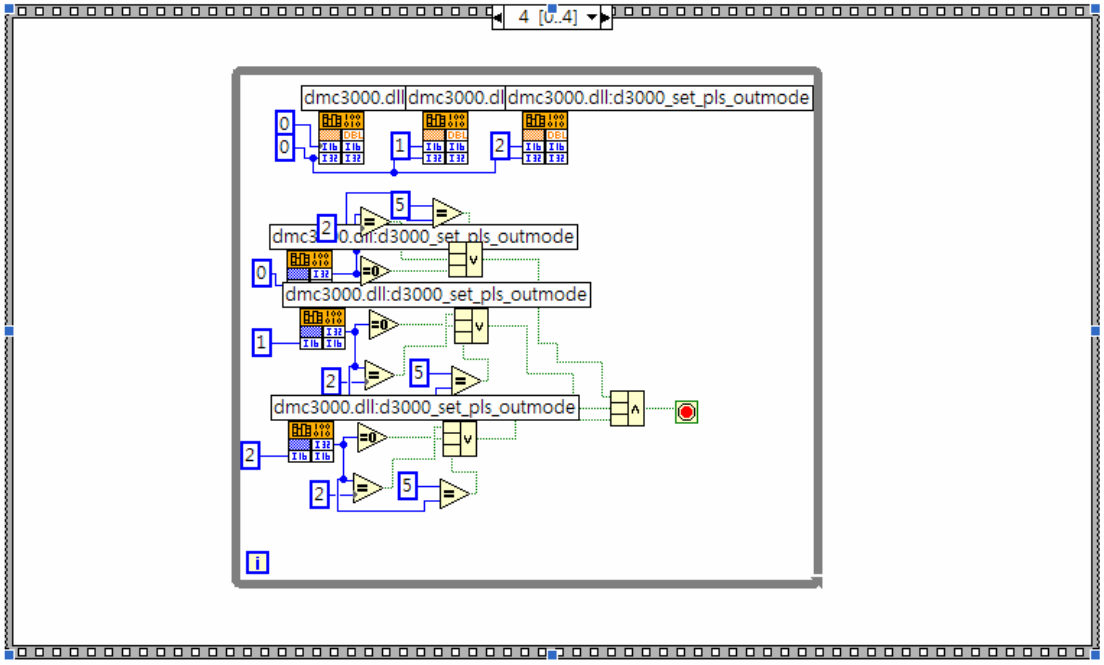

圖 3-50 GDSII 規格運動控制介面 LabVIEW 流程圖-歸原點(10)

接下來讀取檔案並擷取結構(Structure)和圖層(Layer)資料：

圖 3-51 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取檔案(11)

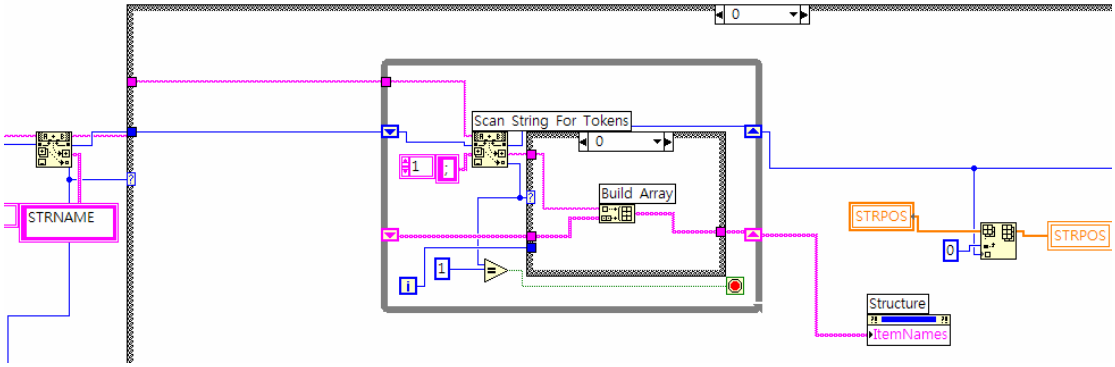

圖 3-52 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取結構(12)

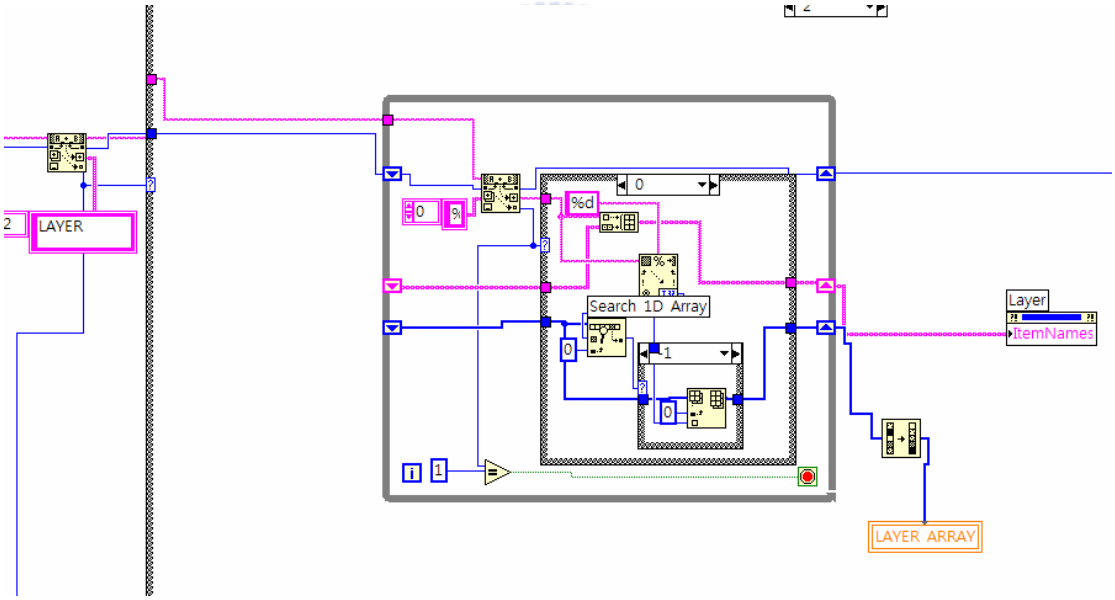

圖 3-53 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取圖層(13)

當檔案讀取完，按下「開始繪圖」按鈕，程式即會讀取使用者選擇要繪製的結構(Structure)和圖層(Layer)。

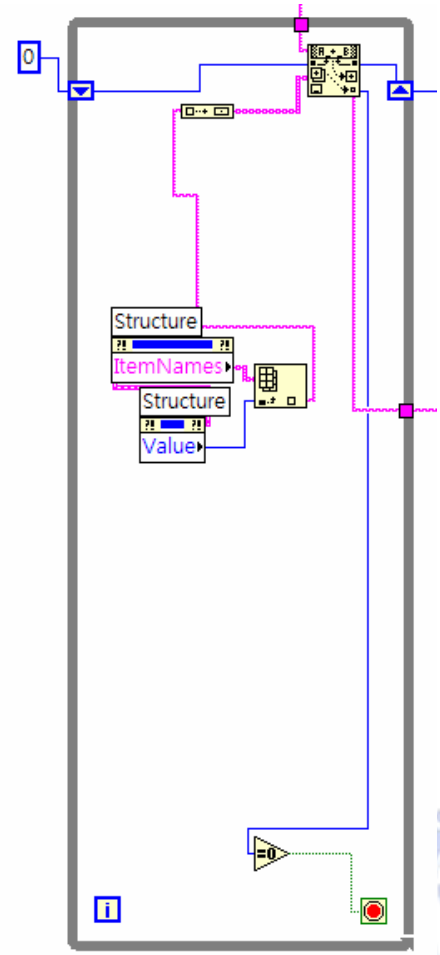

圖 3-54 GDSII 規格運動控制介面 LabVIEW 流程圖-輸入結構和圖層(14)

接下來解說程式中央區塊的部份，首先程式會讀取ASCII File並取出該圖的設計比例，顯示於介面上給使用者知曉。下頁圖為讀取USERUNITS，PHYSUNITS的LabVIEW流程圖。

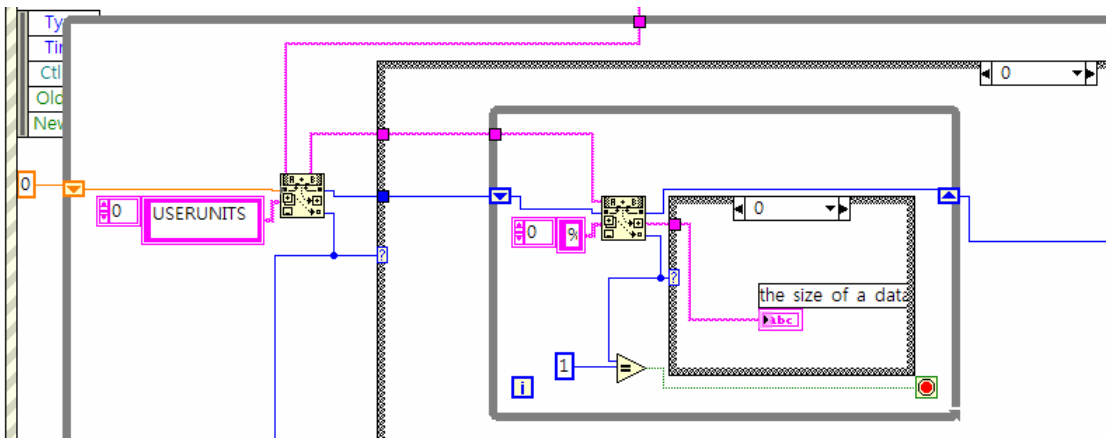

圖 3-55 GDSII規格運動控制介面LabVIEW流程圖-讀取圖形比例(15)

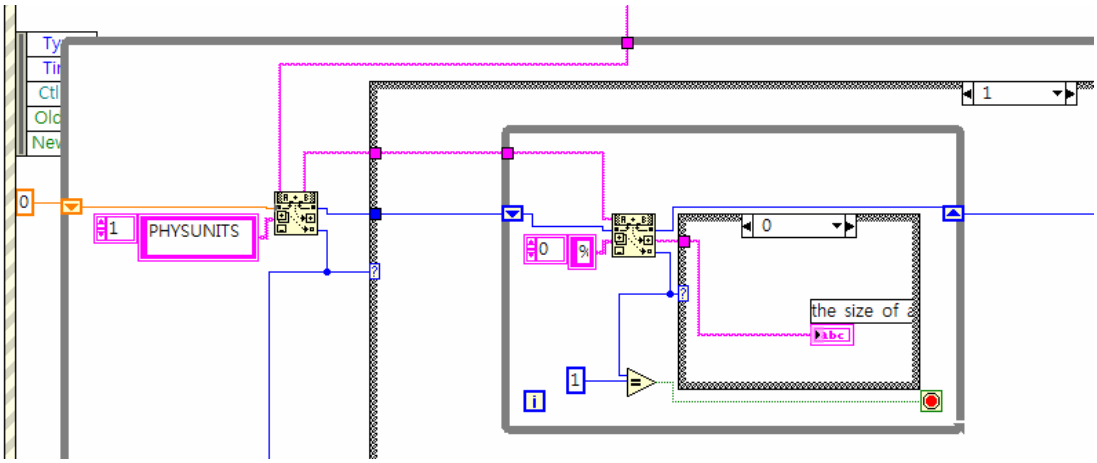

圖 3-56 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取圖形比例(16)

接下來將會進入繪圖的部分，由於圖形種類有兩種：BOUNDARY 和 PATH，因此設計兩個流程分別處理，在此先介紹 PATH 的部份，下圖為 PATH 處理流程圖：

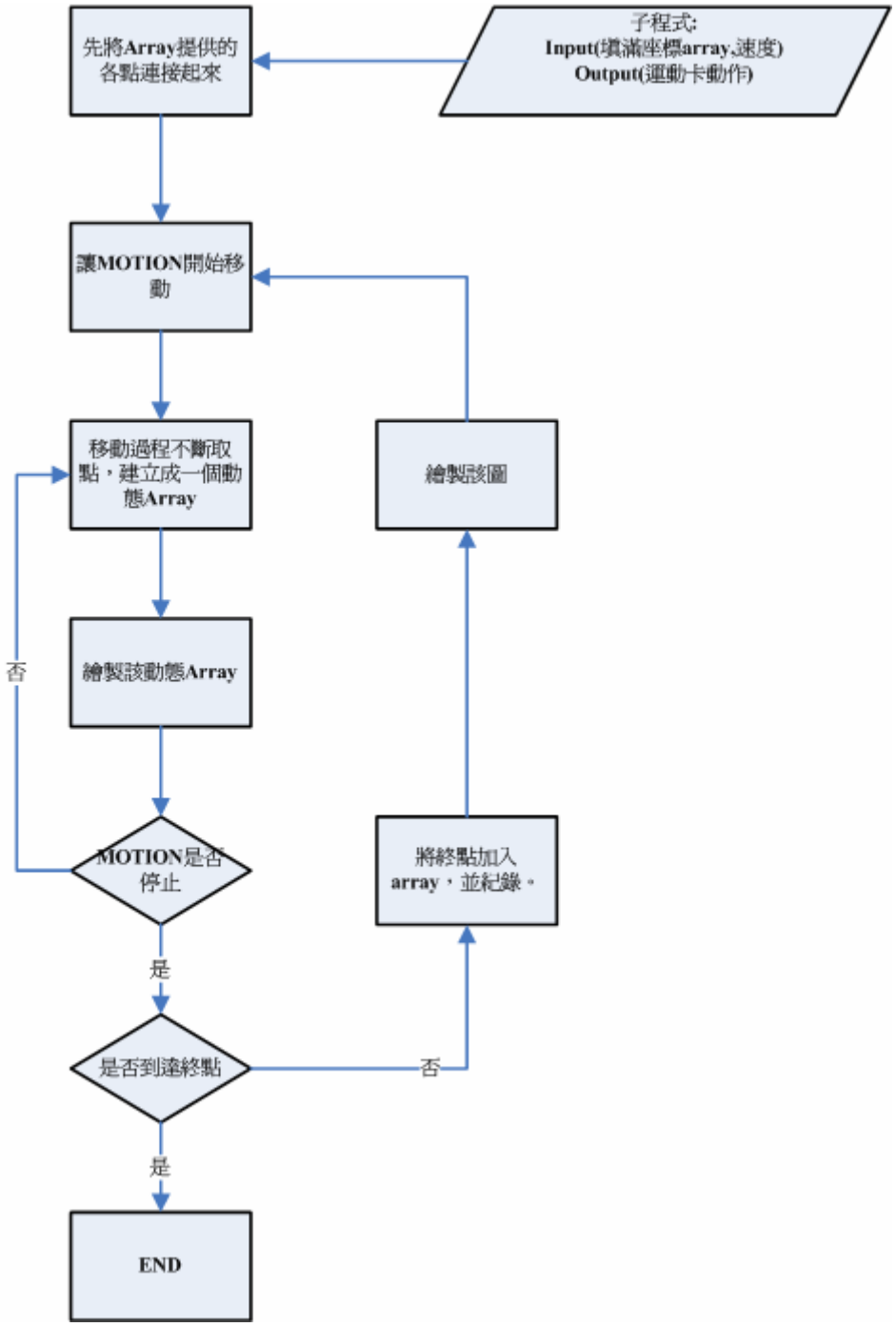

圖 3-57 PATH 處理流程圖

要執行第一步驟，得先讀取該 PATH 之座標 Array，我把此功能做成一個小圖示 

，利用 LabVIEW 來實現該程式的圖在下頁：

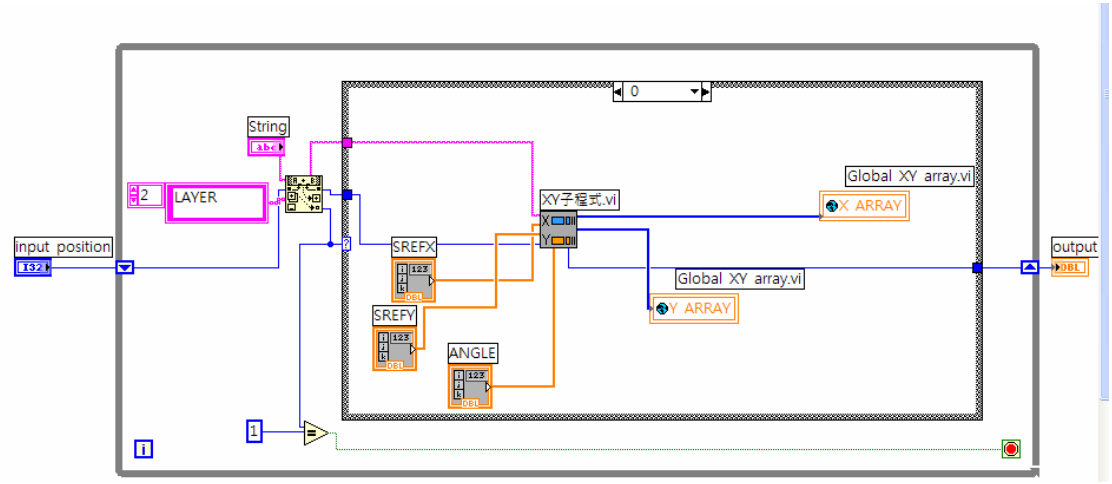

圖 3-58 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取 XY 數值 (17)

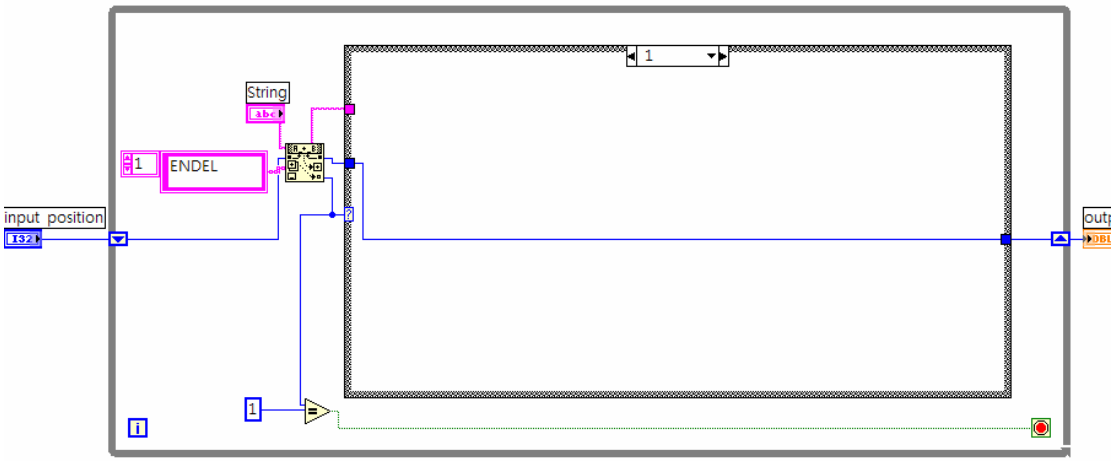

圖 3-59 GDSII 規格運動控制介面 LabVIEW 流程圖-結束跳出(18)

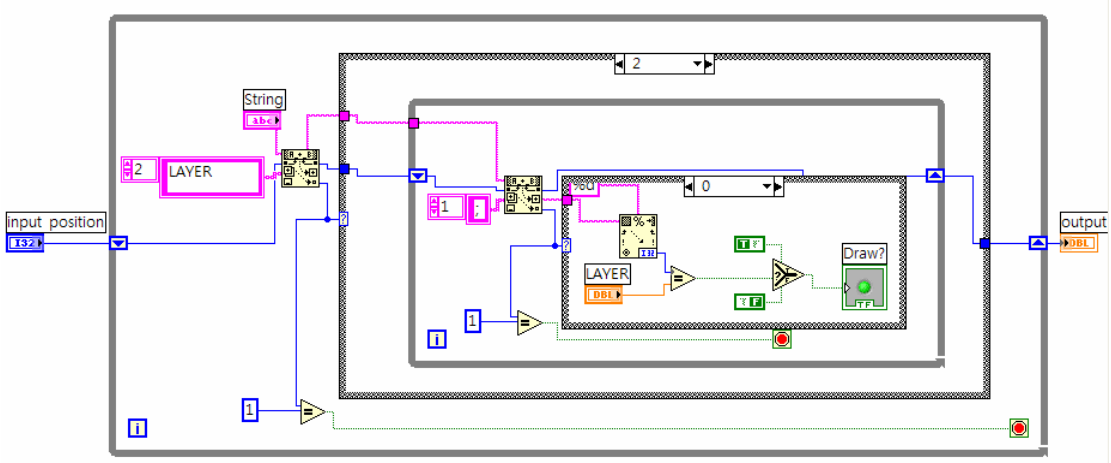

圖 3-60 GDSII 規格運動控制介面 LabVIEW 流程圖-是否繪圖(19)

上圖的 XY 子程式部分，主要功能為讀取 XY 位址並給其他程式使用，利用 LabVIEW 來實現該程式的圖在下頁。

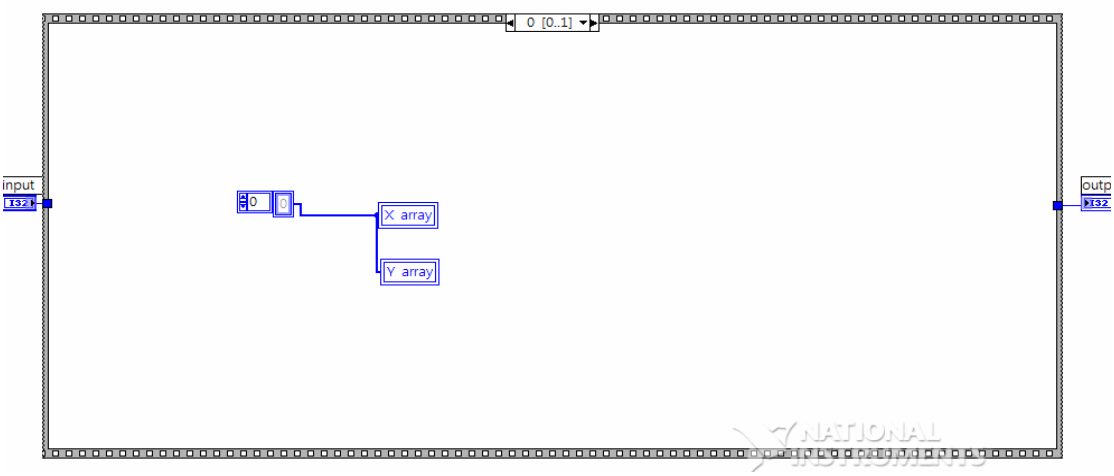

圖 3-61 GDSII 規格運動控制介面 LabVIEW 流程圖-初始化數值(20)

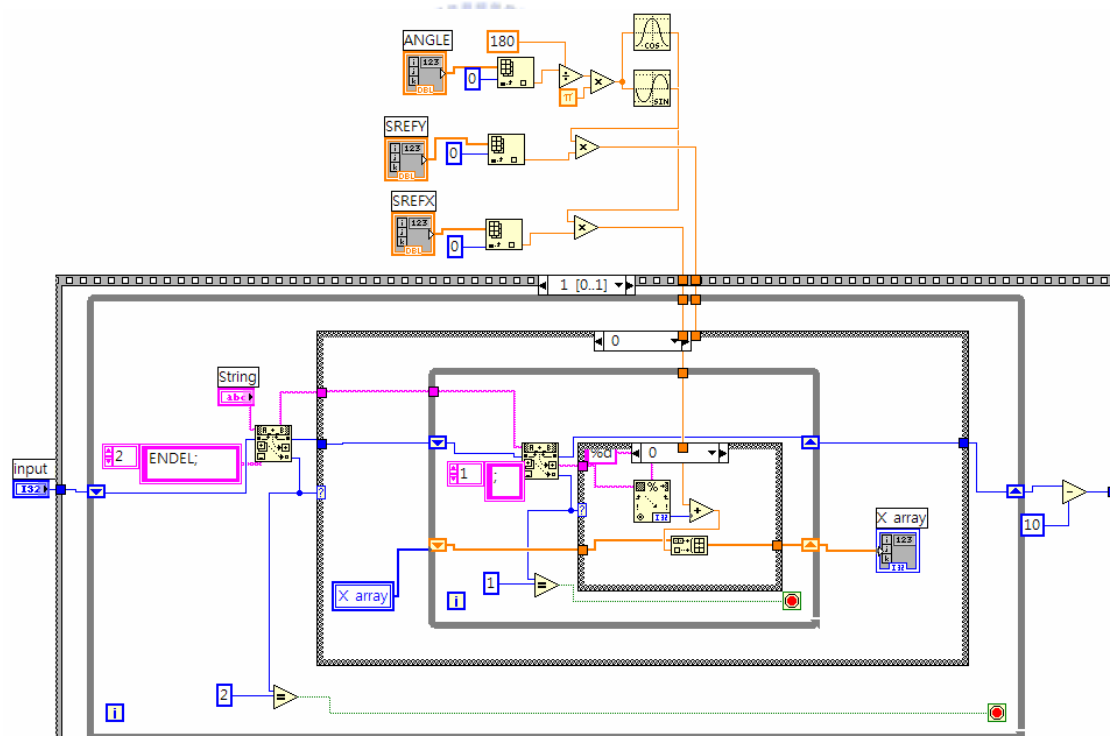

圖 3-62 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取 X 數值(21)

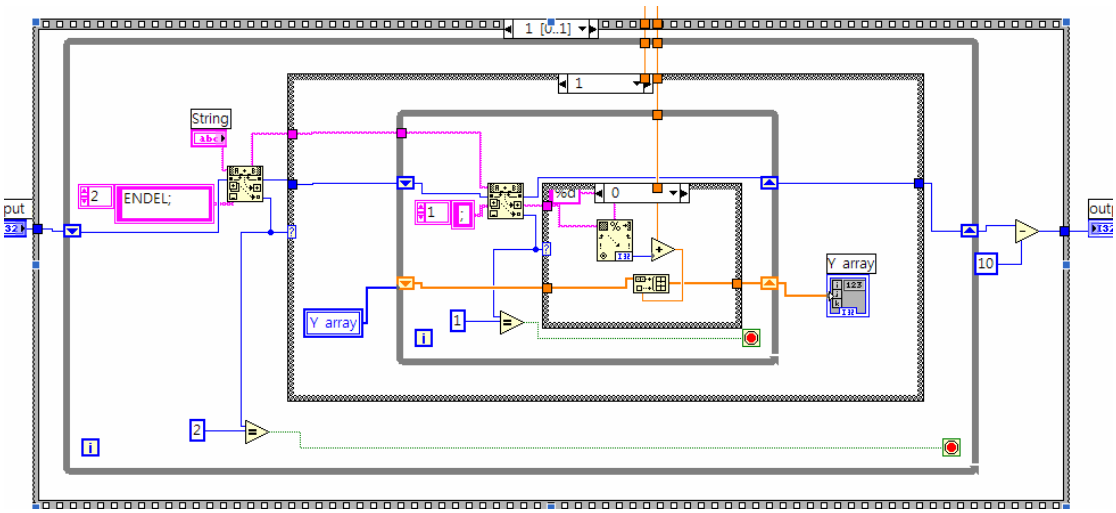

圖 3-63 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取 Y 數值(22)

接下來要來要把設計好的 PATH 子程式套入到主程式中運用，如下：

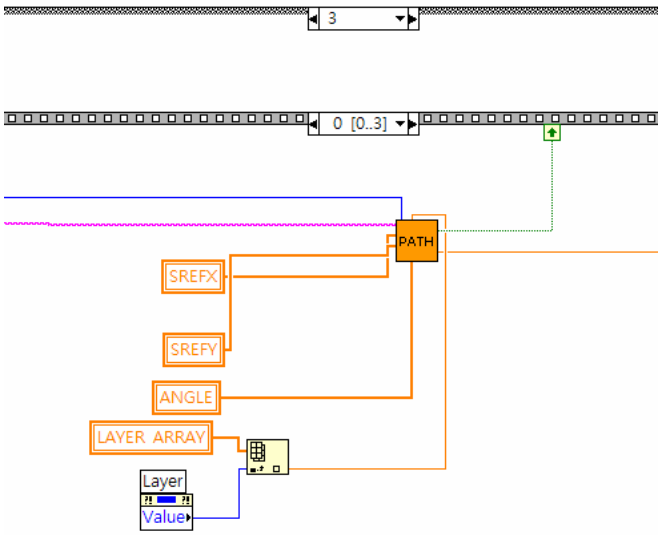

圖 3-64 GDSII 規格運動控制介面 LabVIEW 流程圖-輸出路徑座標(23)

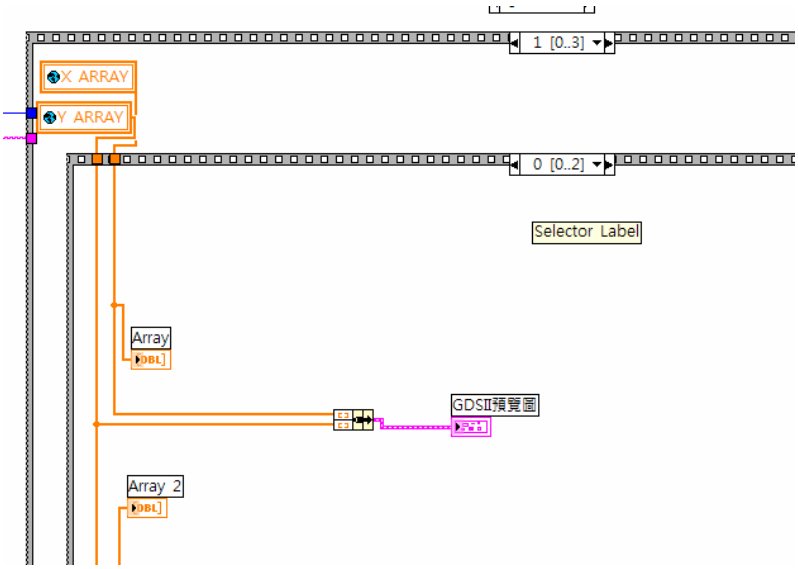

圖 3-65 GDSII 規格運動控制介面 LabVIEW 流程圖-繪製預覽圖(24)

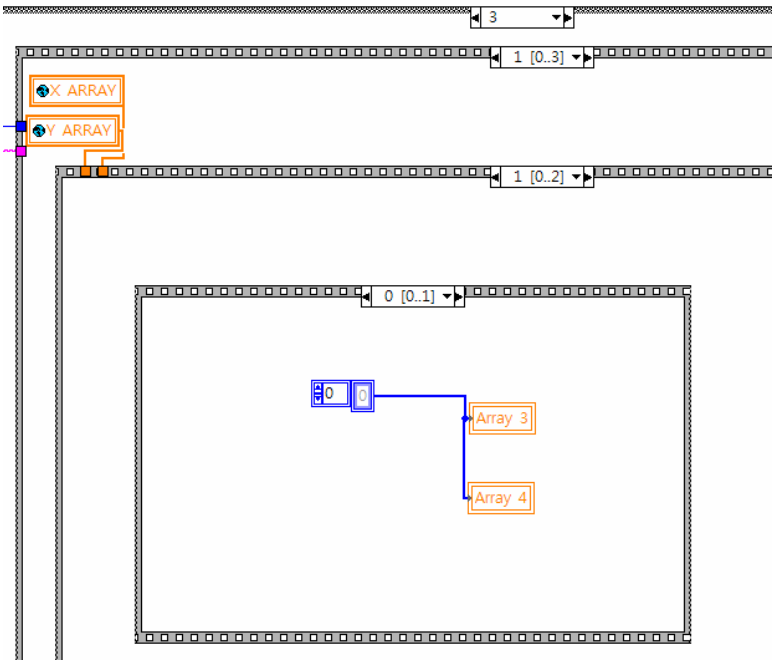

圖 3-66 GDSII 規格運動控制介面 LabVIEW 流程圖-初始化目前位置(25)

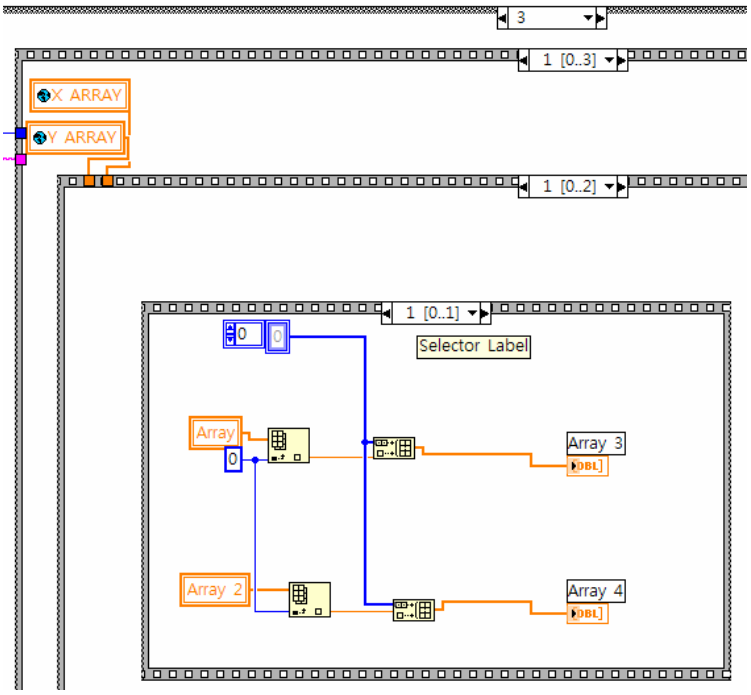

圖 3-67 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取起點(26)

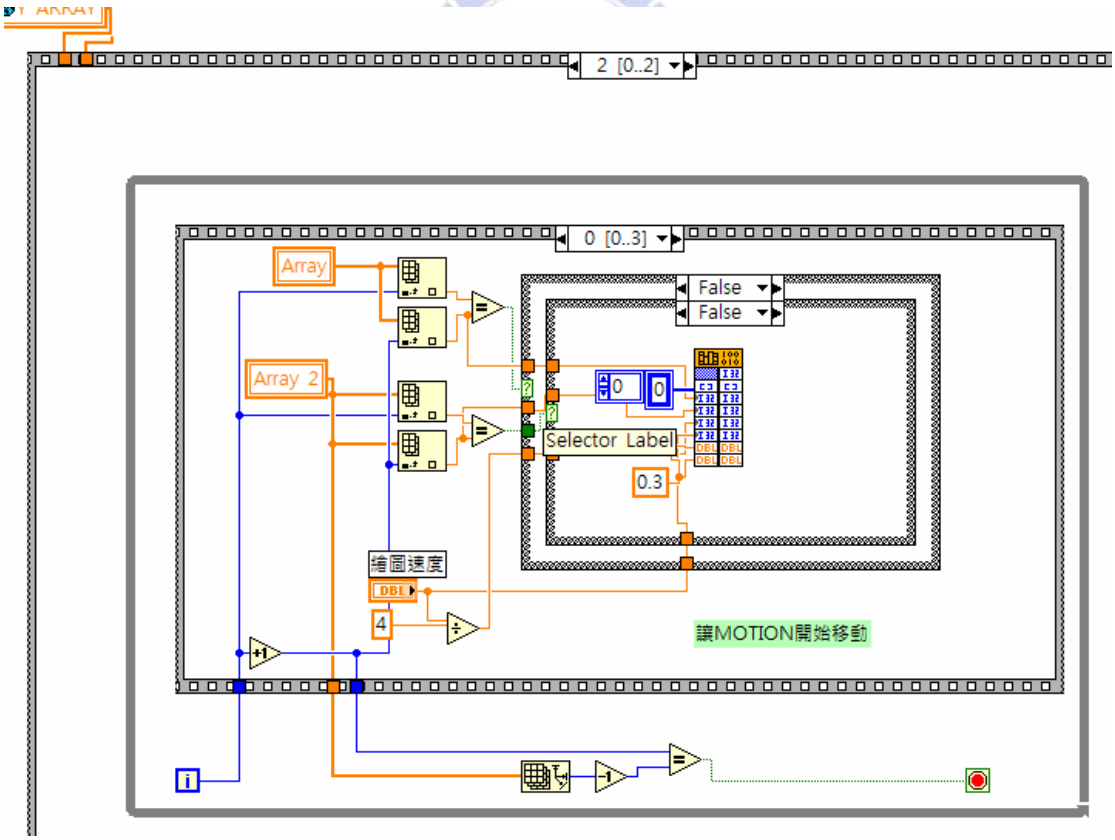

圖 3-68 GDSII 規格運動控制介面 LabVIEW 流程圖-開始移動(27)

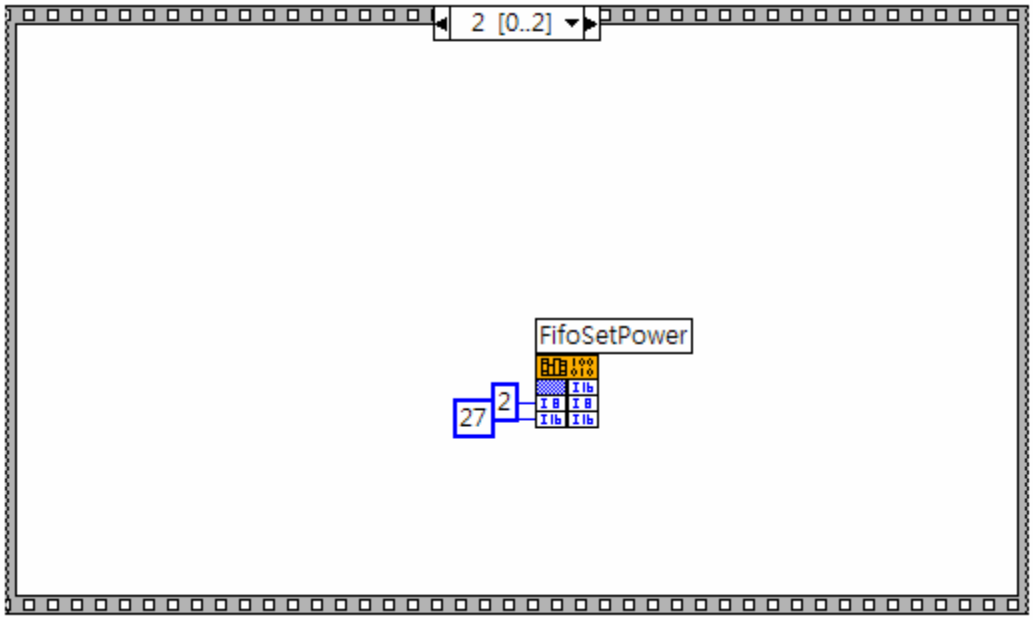

圖 3-69 GDSII 規格運動控制介面 LabVIEW 流程圖-啟動繪圖工具(28)

上圖對座標做個判斷，False 代表在該軸方向有移動量，True 則沒有，再對應情況使用直線插補或是單軸運動。

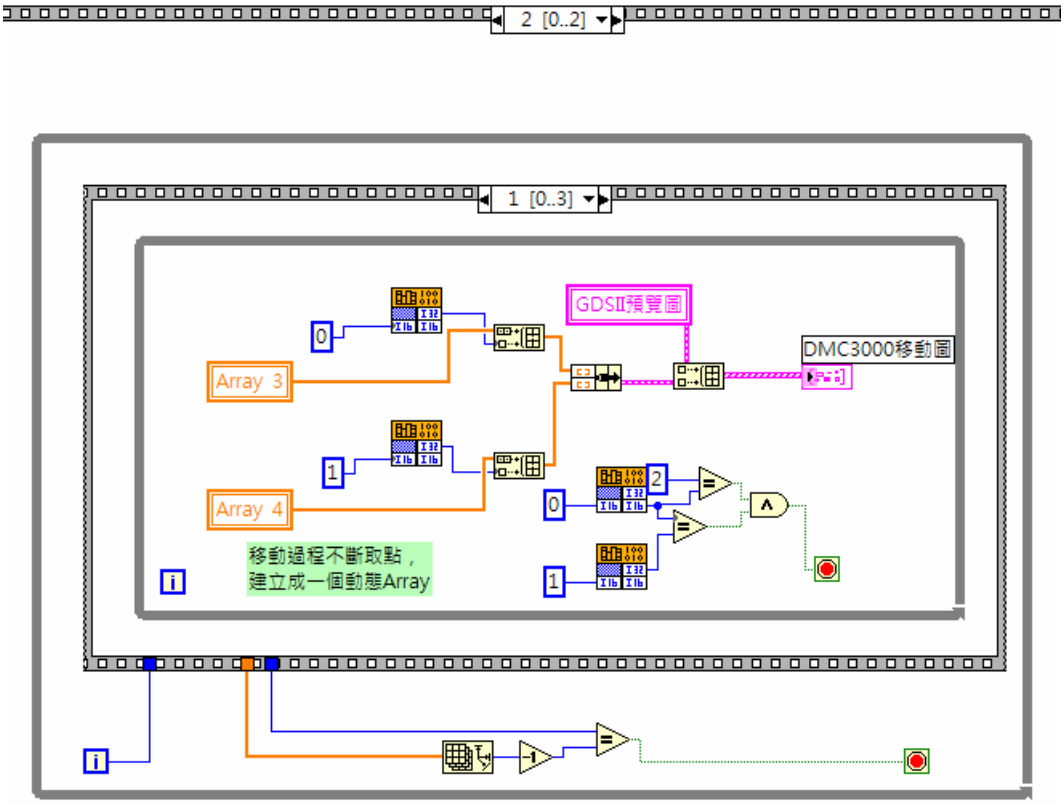

圖 3-70 GDSII 規格運動控制介面 LabVIEW 流程圖-繪製移動圖(29)

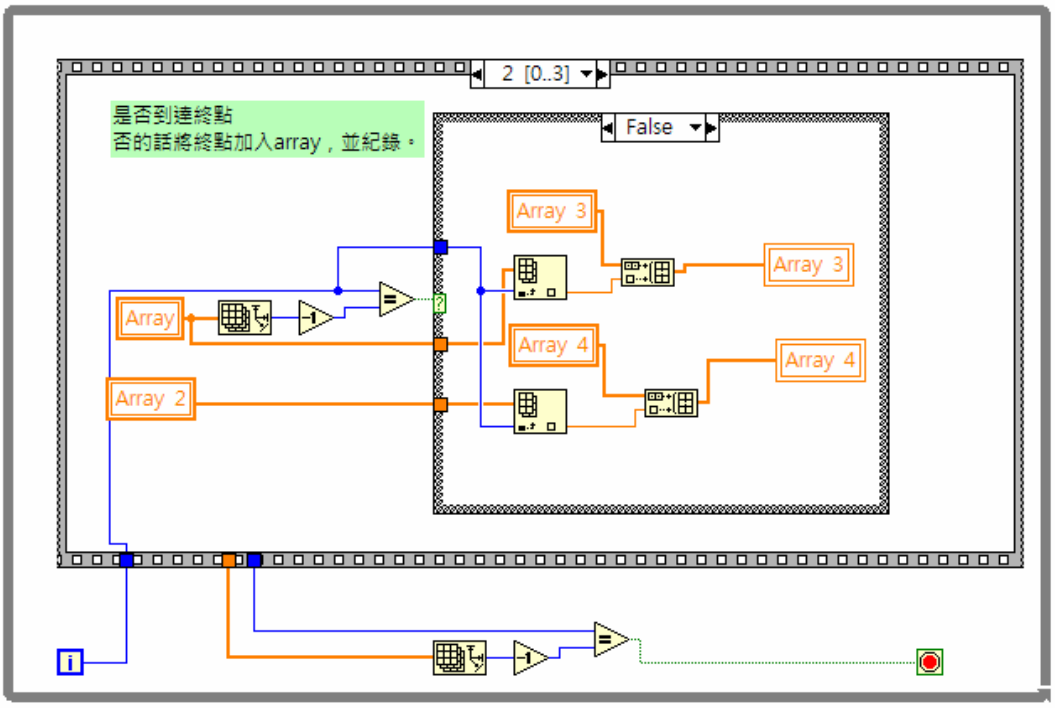

圖 3-71 GDSII 規格運動控制介面 LabVIEW 流程圖-轉折點(30)

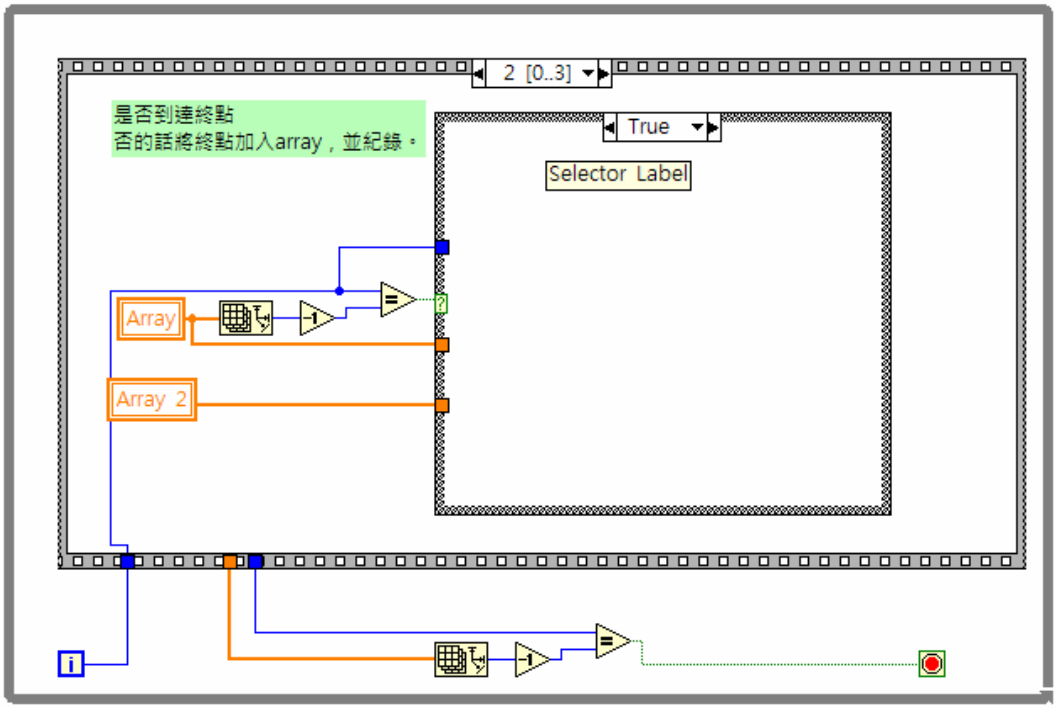

圖 3-72 GDSII 規格運動控制介面 LabVIEW 流程圖-判斷終點(31)

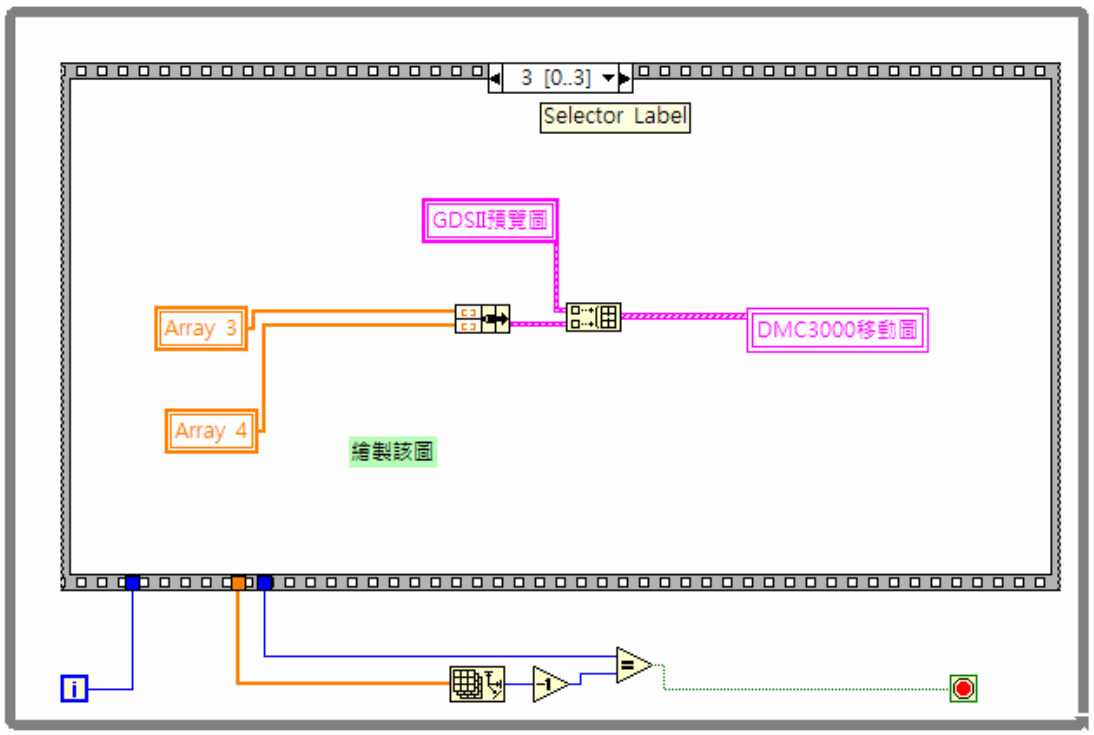

圖 3-73 GDSII 規格運動控制介面 LabVIEW 流程圖-繪製移動圖(32)

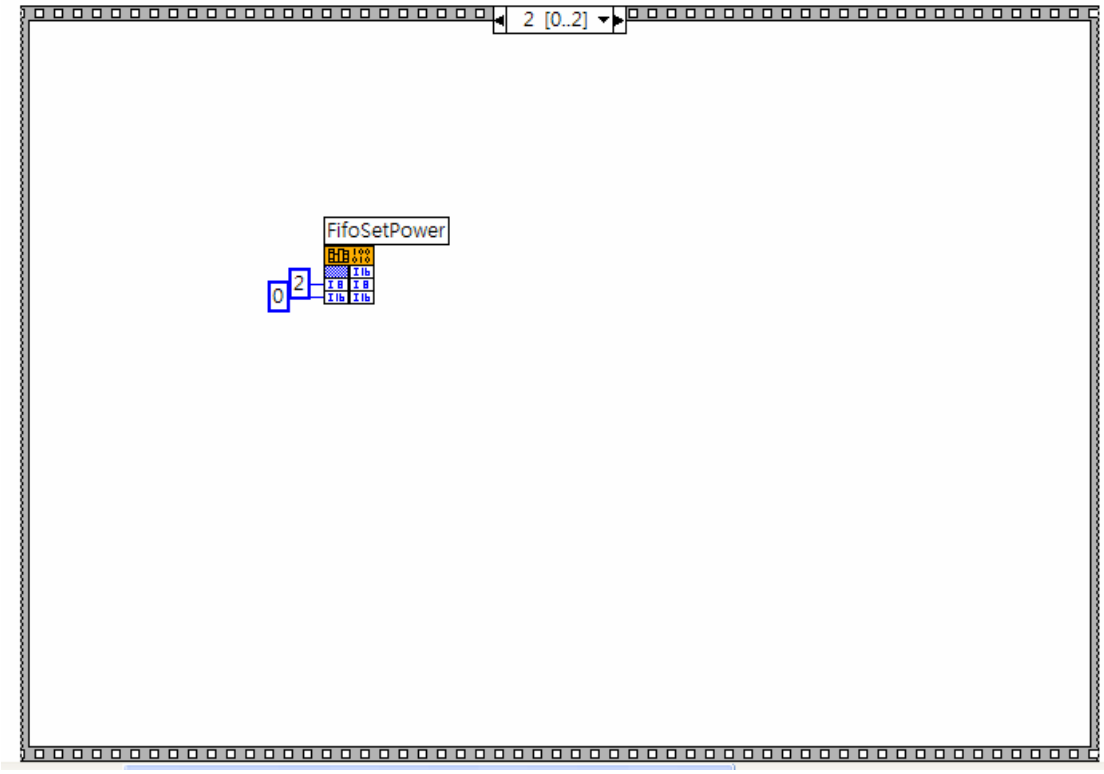

圖 3-74 GDSII 規格運動控制介面 LabVIEW 流程圖-關閉繪圖工具(33)

以上完成了執行 PATH 運動及繪圖的部份，接下來要製作 BOUNDARY 功能的程式，BOUNDARY 的流程圖如下：

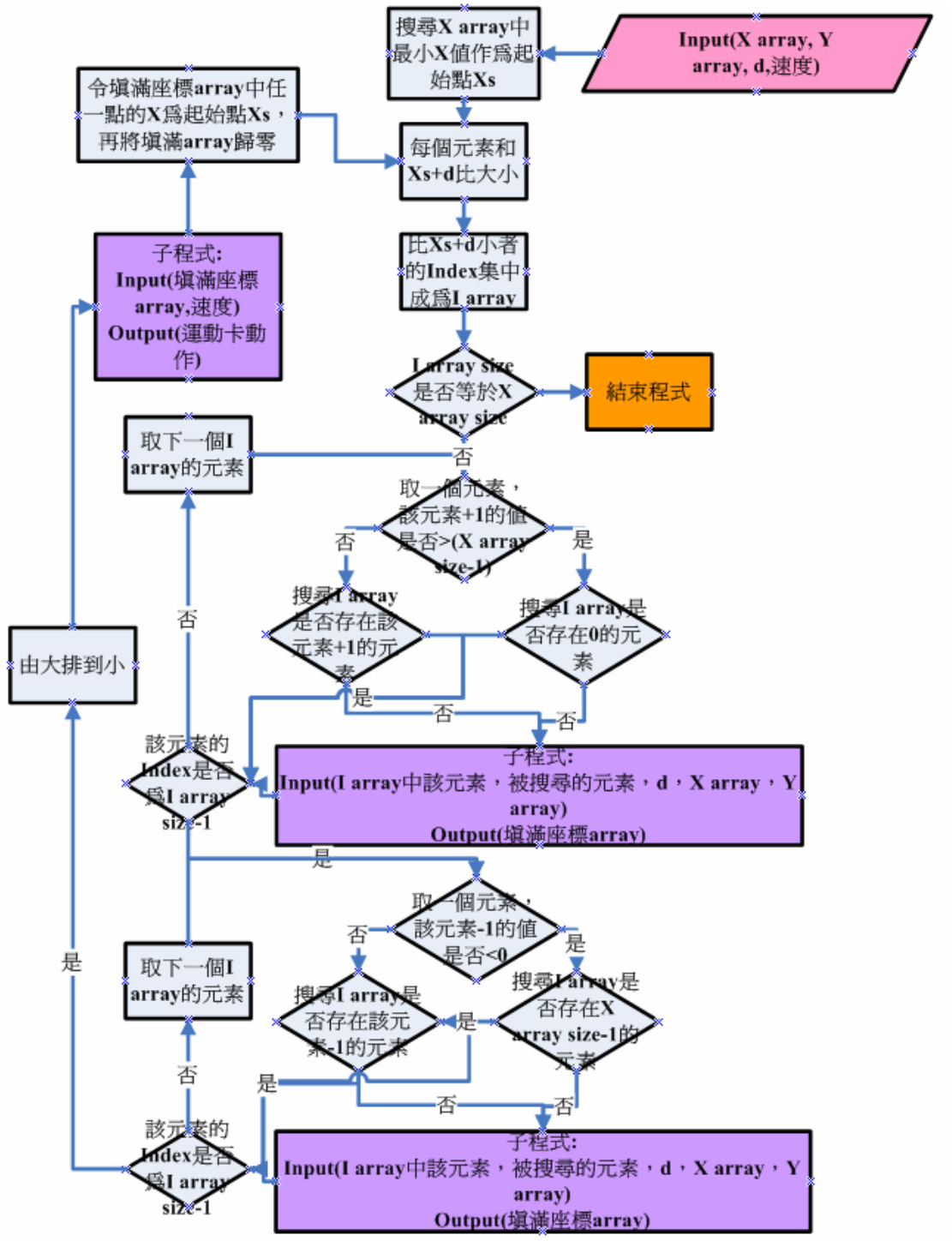

圖 3-75 BOUNDARY 處理流程圖

利用 LabVIEW 來完成上述程式的方法置於下頁，首先我們製作一個圖示

，其功能為輸入 BOUNDARY 座標，並輸出填滿該 BOUNDARY 所該移動的座標 Array。

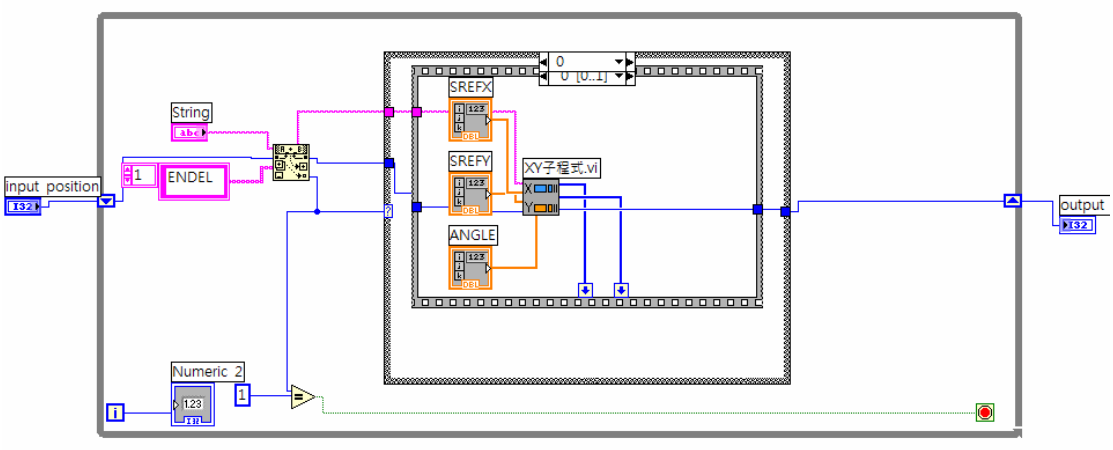

圖 3-76 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取 XY 座標(34)

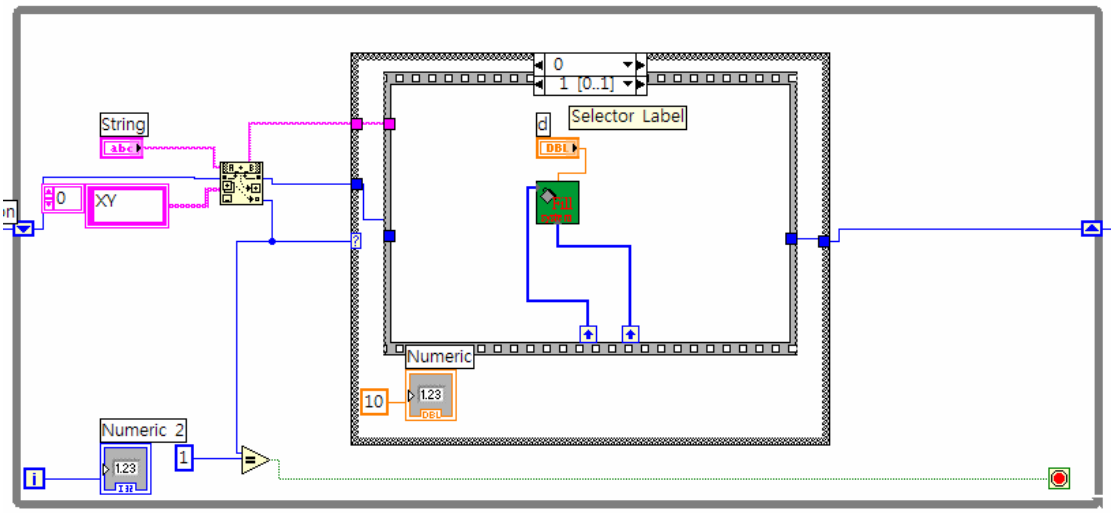

圖 3-77 GDSII 規格運動控制介面 LabVIEW 流程圖-計算填滿座標(35)

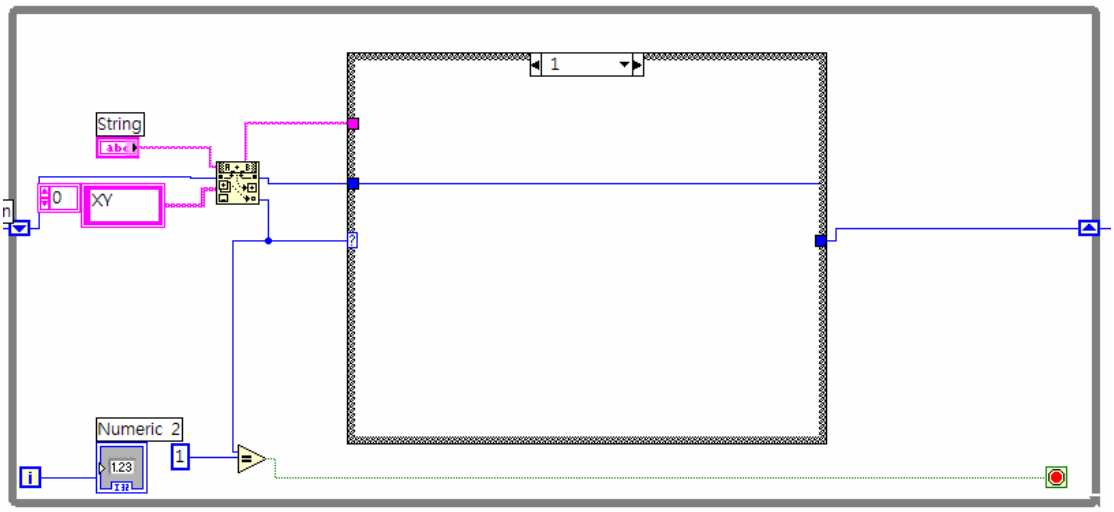

圖 3-78 GDSII 規格運動控制介面 LabVIEW 流程圖-結束子程式(36)

其中 之程式主要工作是計算要填滿該區域，所該輸出的座標為多少，程式內容如下：

圖 3-79 GDSII 規格運動控制介面 LabVIEW 流程圖-取 Xs 初始值(37)

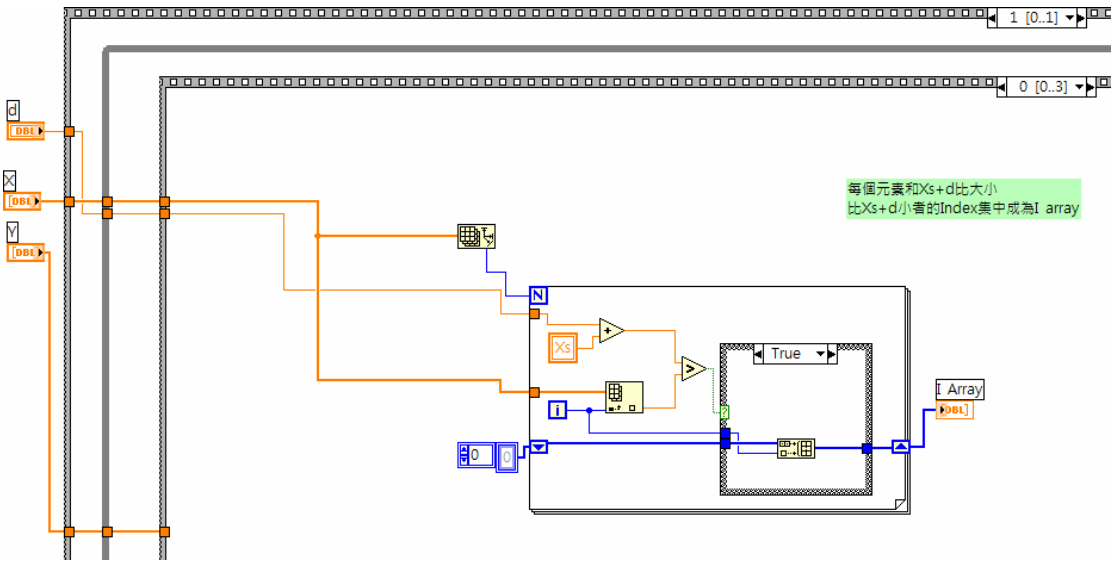

圖 3-80 GDSII 規格運動控制介面 LabVIEW 流程圖-計算 I 陣列(38)

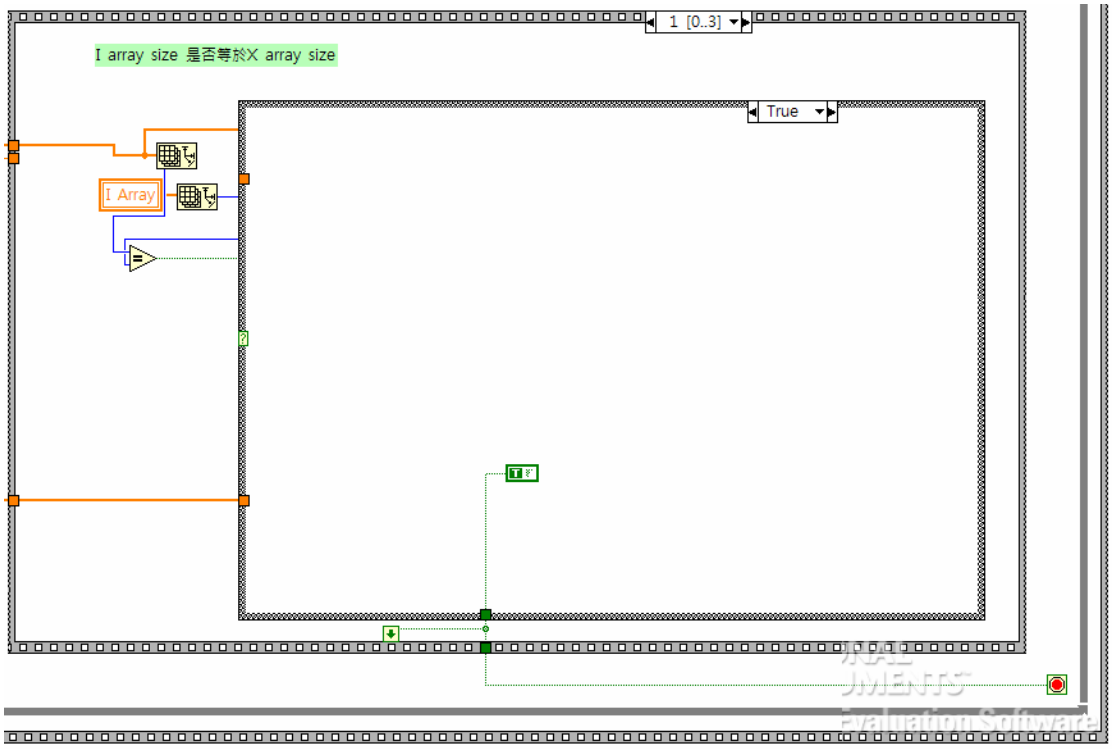

圖 3-81 GDSII 規格運動控制介面 LabVIEW 流程圖-判斷結束(39)

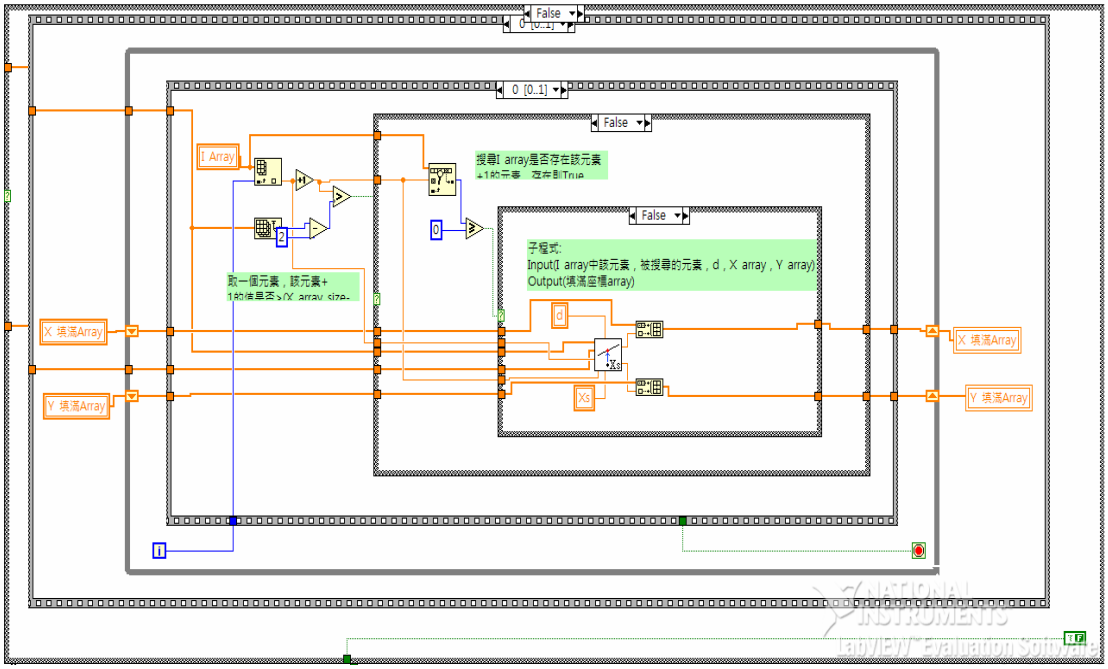

圖 3-82 GDSII 規格運動控制介面 LabVIEW 流程圖-尋找交錯端點(40)

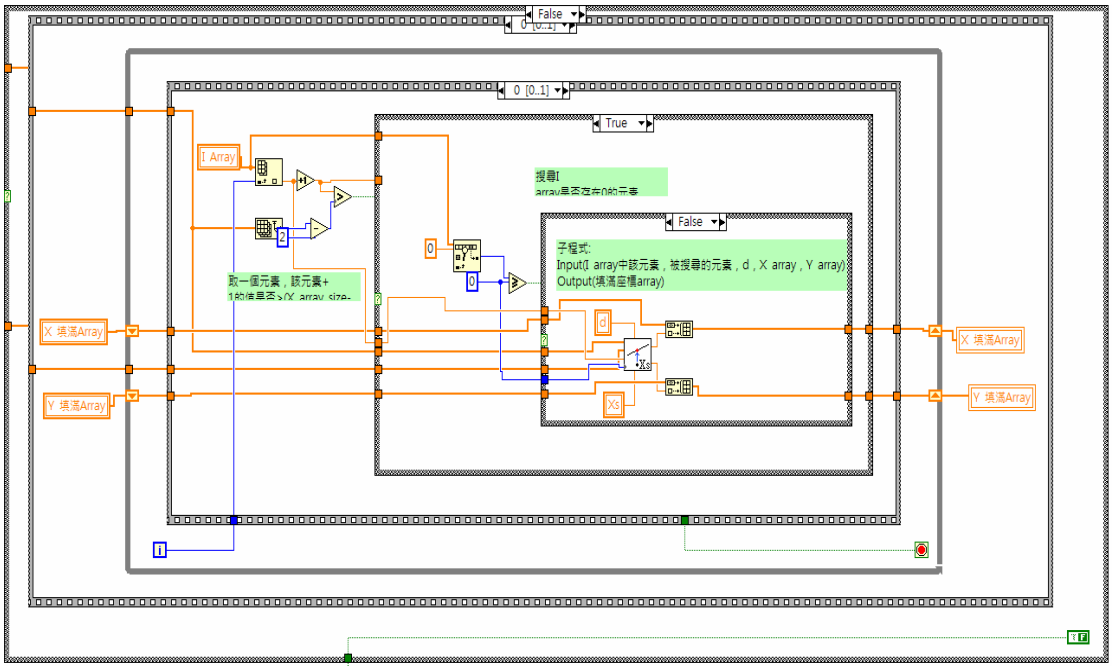

圖 3-83 GDSII 規格運動控制介面 LabVIEW 流程圖-尋找交錯端點(41)

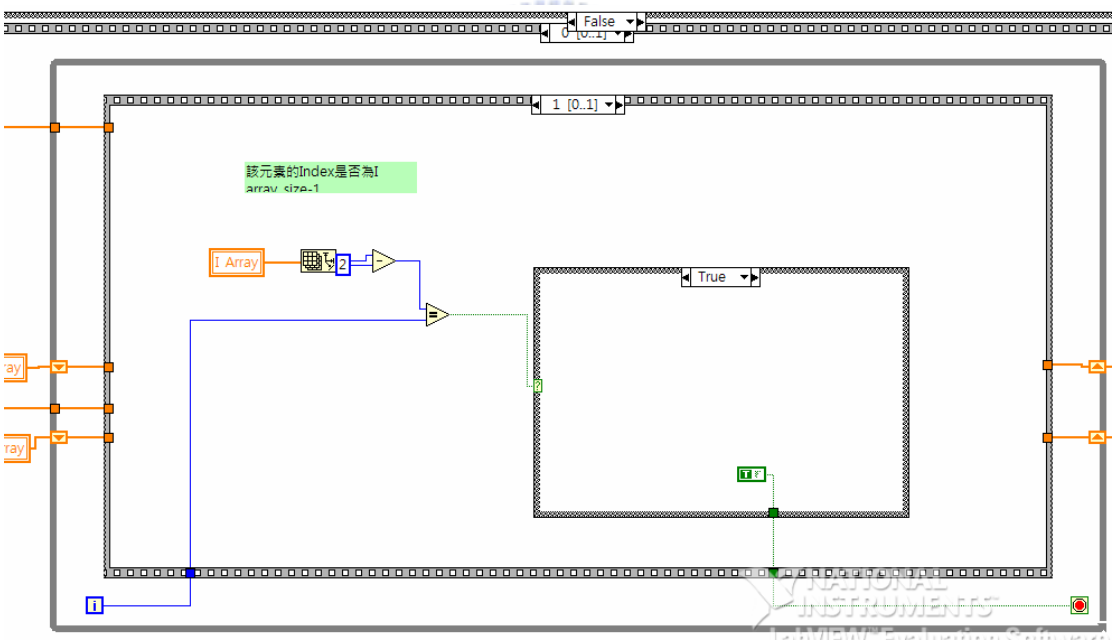

圖 3-84 GDSII 規格運動控制介面 LabVIEW 流程圖-判斷結束(42)

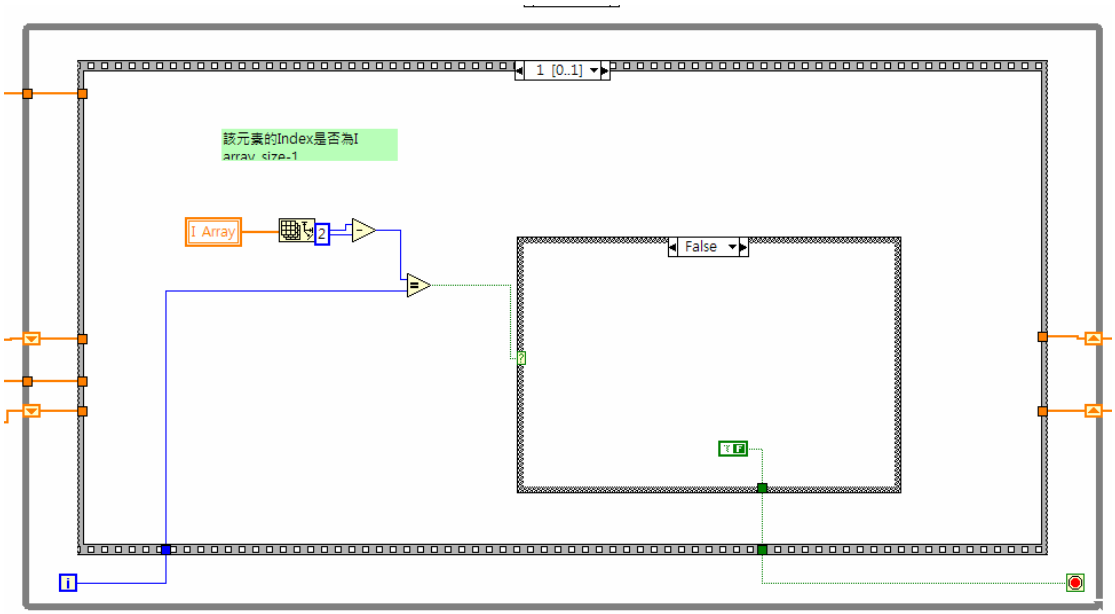

圖 3-85 GDSII 規格運動控制介面 LabVIEW 流程圖-判斷結束(43)

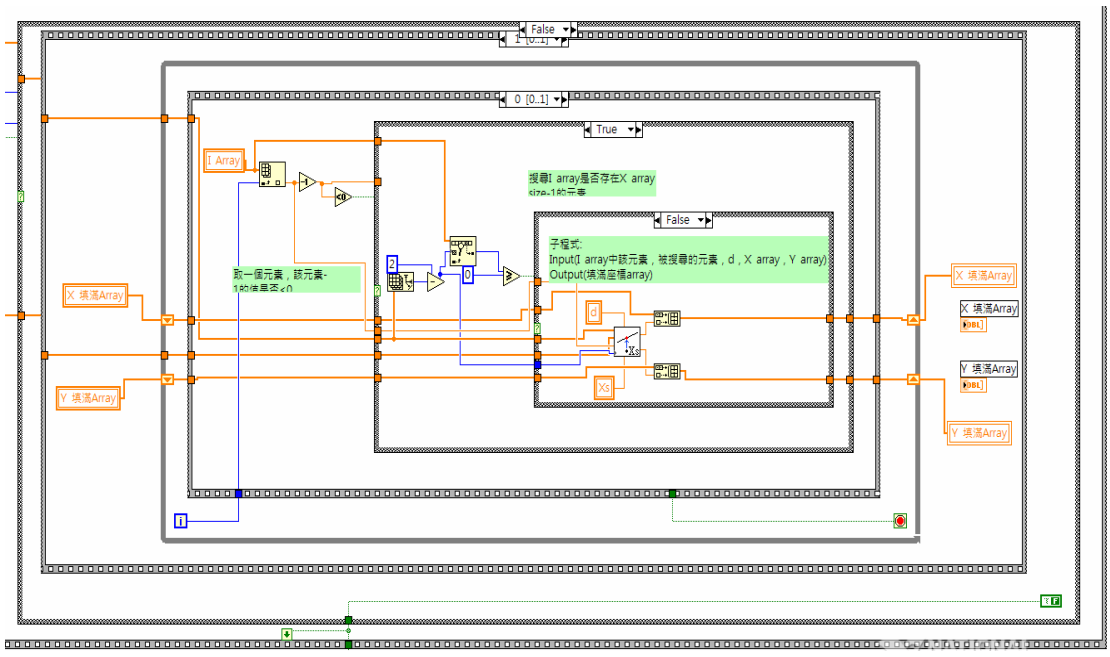

圖 3-86 GDSII 規格運動控制介面 LabVIEW 流程圖-尋找交錯端點(44)

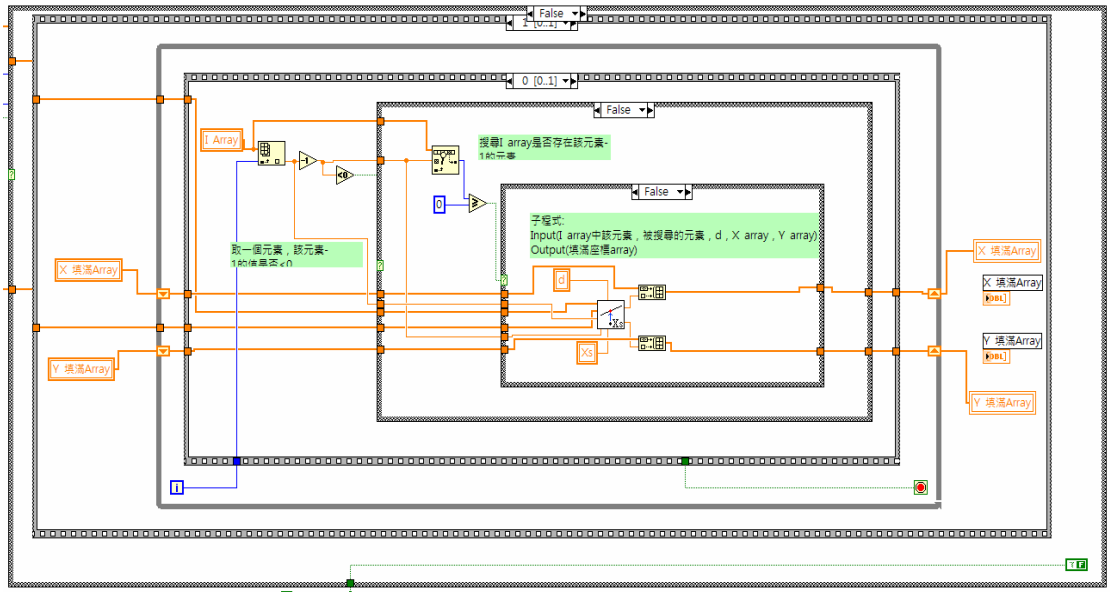

圖 3-87 GDSII 規格運動控制介面 LabVIEW 流程圖-尋找交錯端點(45)

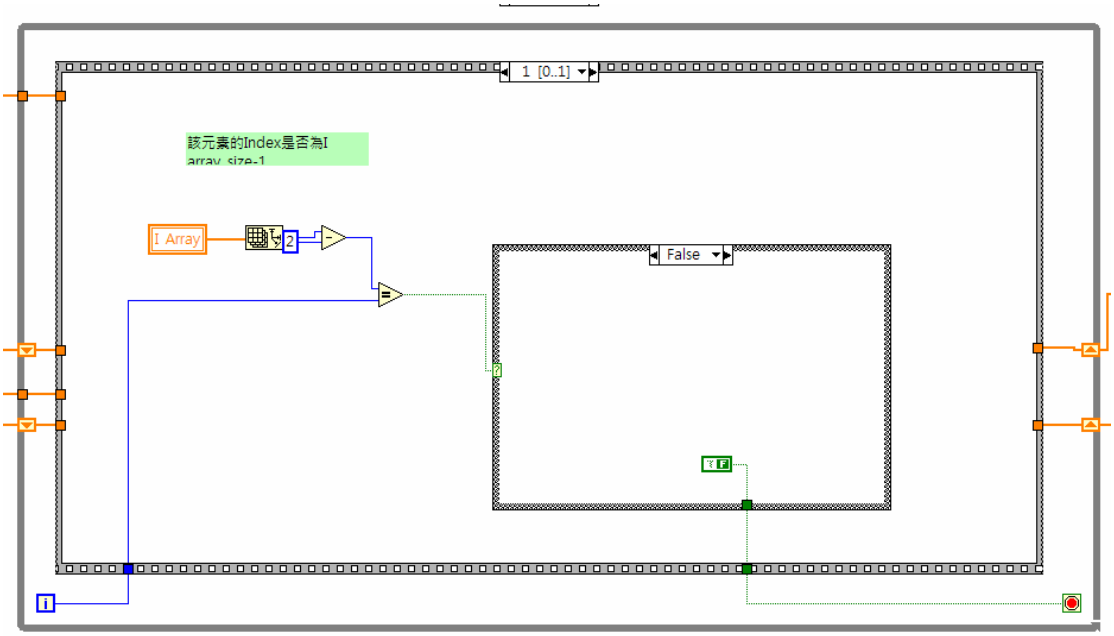

圖 3-88 GDSII 規格運動控制介面 LabVIEW 流程圖-判斷結束(46)

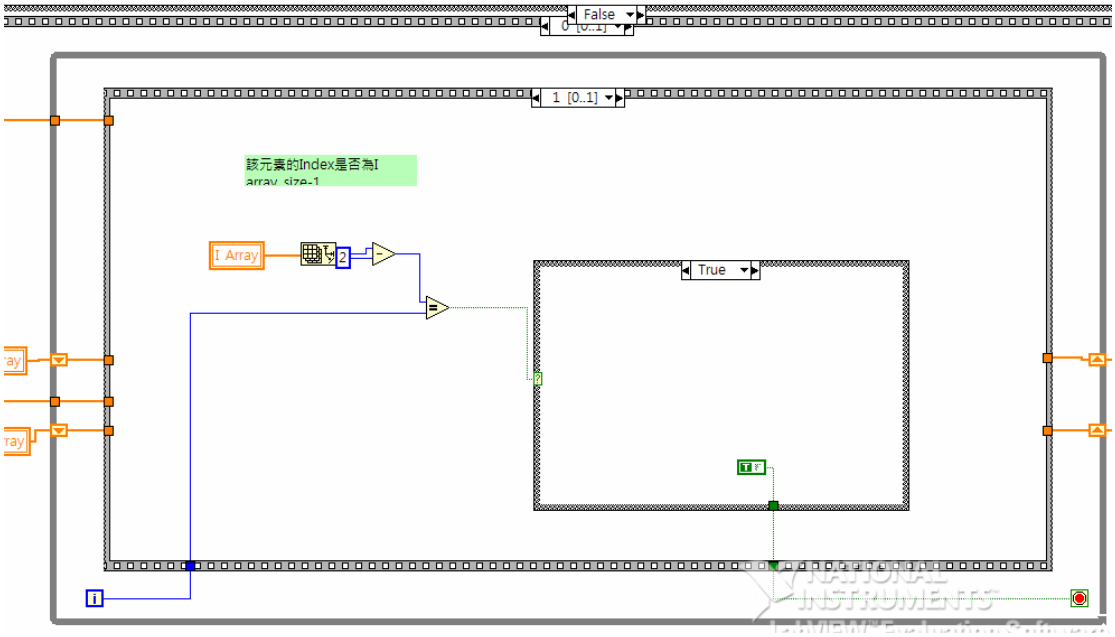

圖 3-89 GDSII 規格運動控制介面 LabVIEW 流程圖-判斷結束(47)

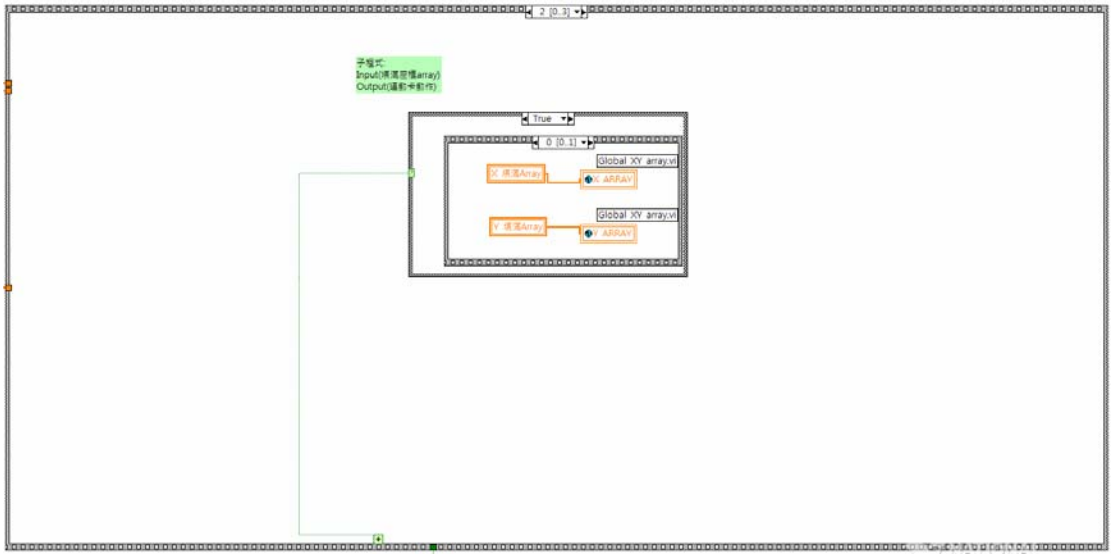

圖 3-90 GDSII 規格運動控制介面 LabVIEW 流程圖-輸出填滿座標(48)

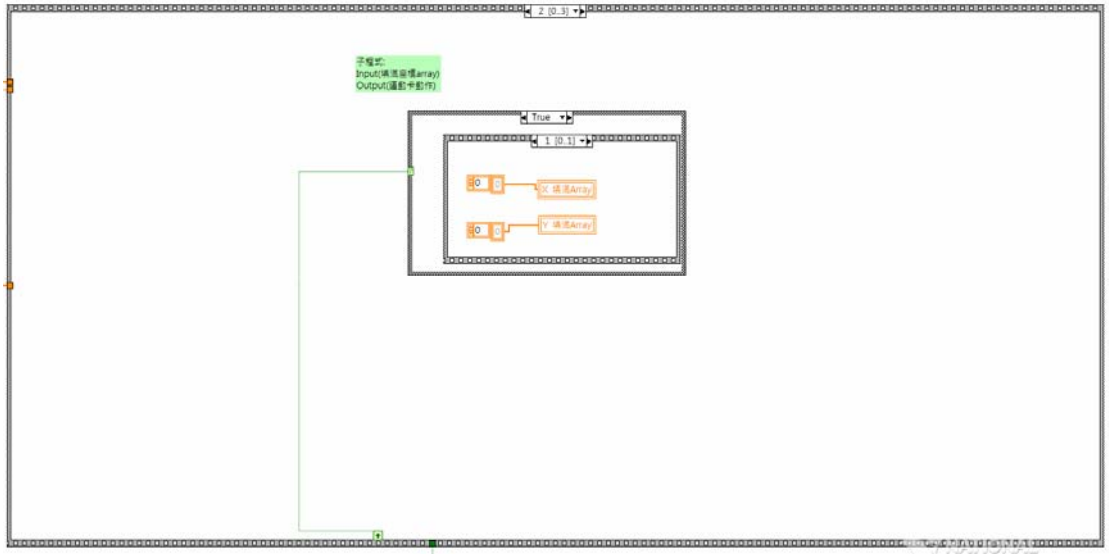

圖 3-91 GDSII 規格運動控制介面 LabVIEW 流程圖(49)

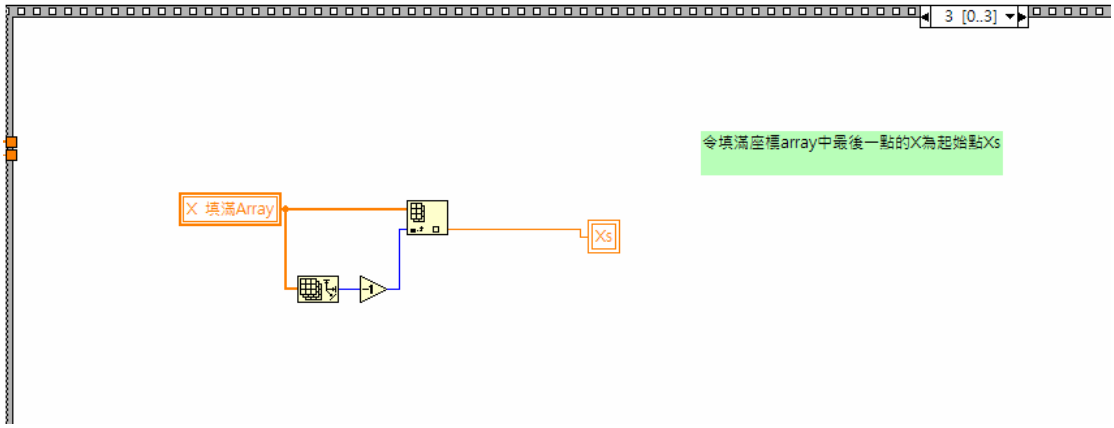

圖 3-92 GDSII 規格運動控制介面 LabVIEW 流程圖-設定下一個 Xs(50)

以上介紹完 

的功能，接下來要利用輸出的填滿座標 Array 來讓介面繪圖以及使機器移動。實現之 LabVIEW 程式圖如下：

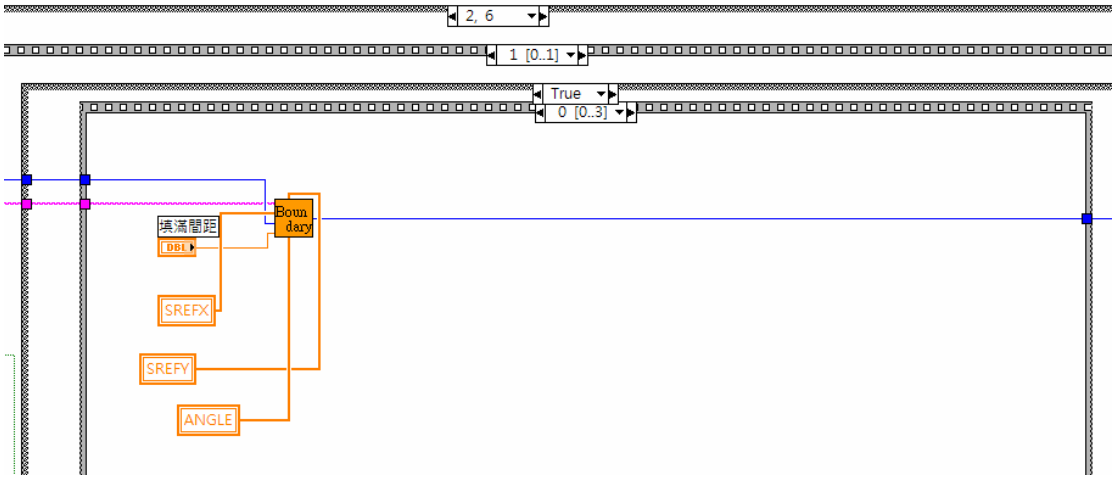

圖 3-93 GDSII 規格運動控制介面 LabVIEW 流程圖-輸出填滿座標(51)

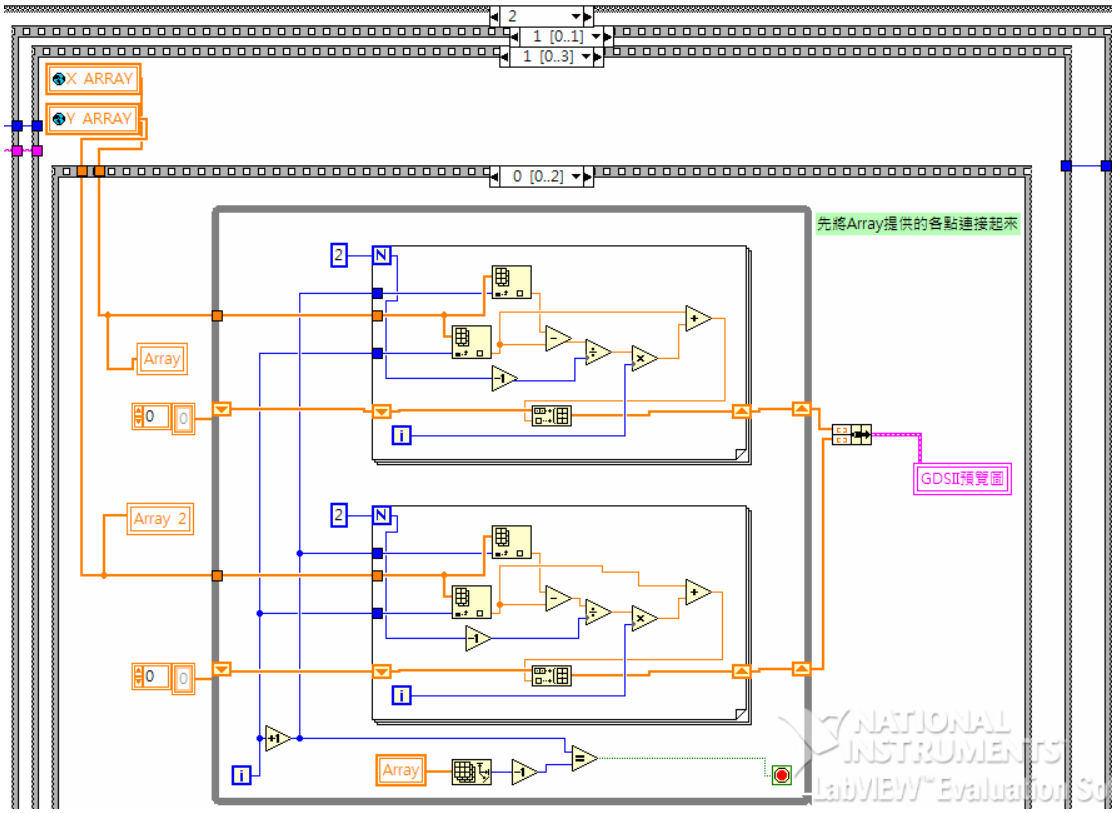

圖 3-94 GDSII 規格運動控制介面 LabVIEW 流程圖-繪製預覽圖(52)

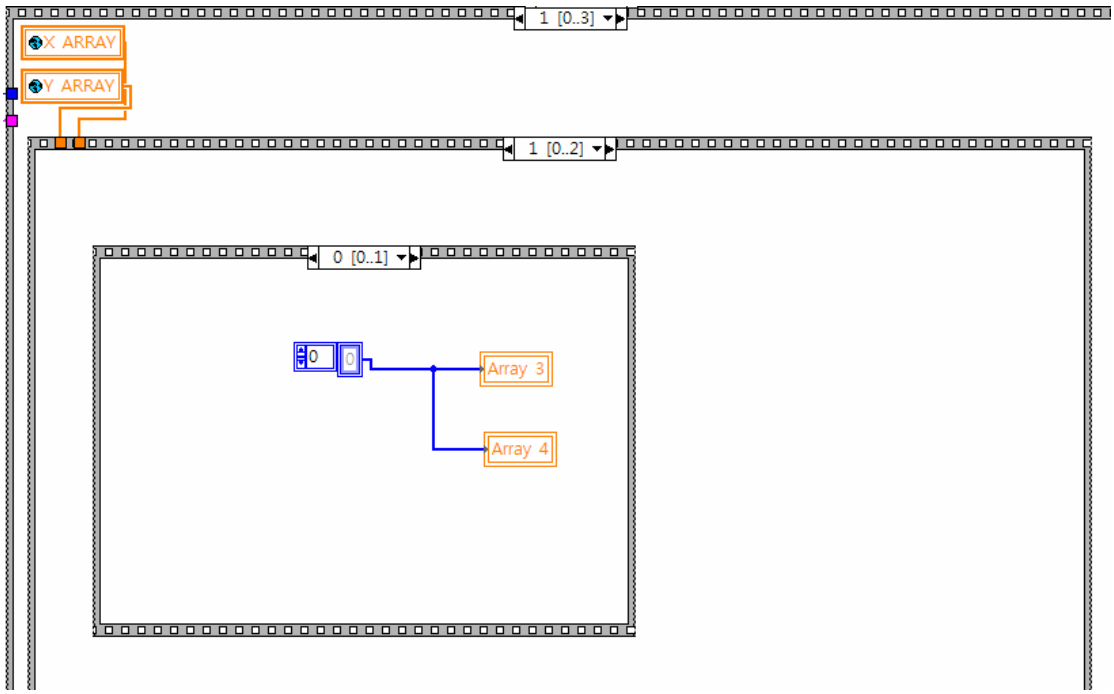

圖 3-95 GDSII 規格運動控制介面 LabVIEW 流程圖-初始化陣列(53)

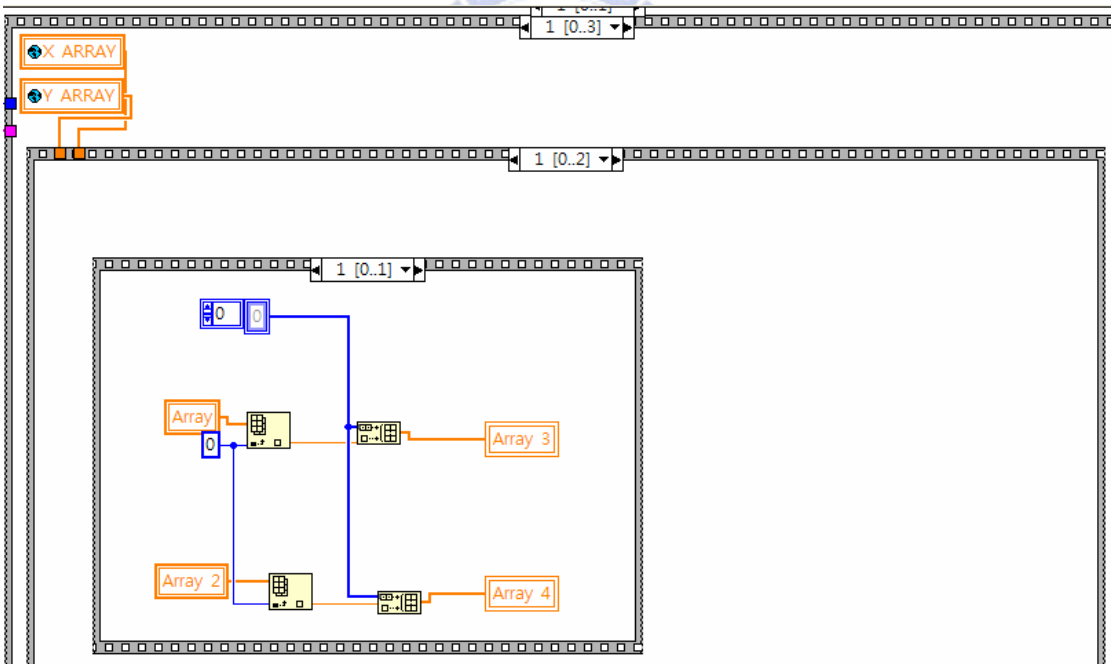

圖 3-96 GDSII 規格運動控制介面 LabVIEW 流程圖-初始化陣列(54)

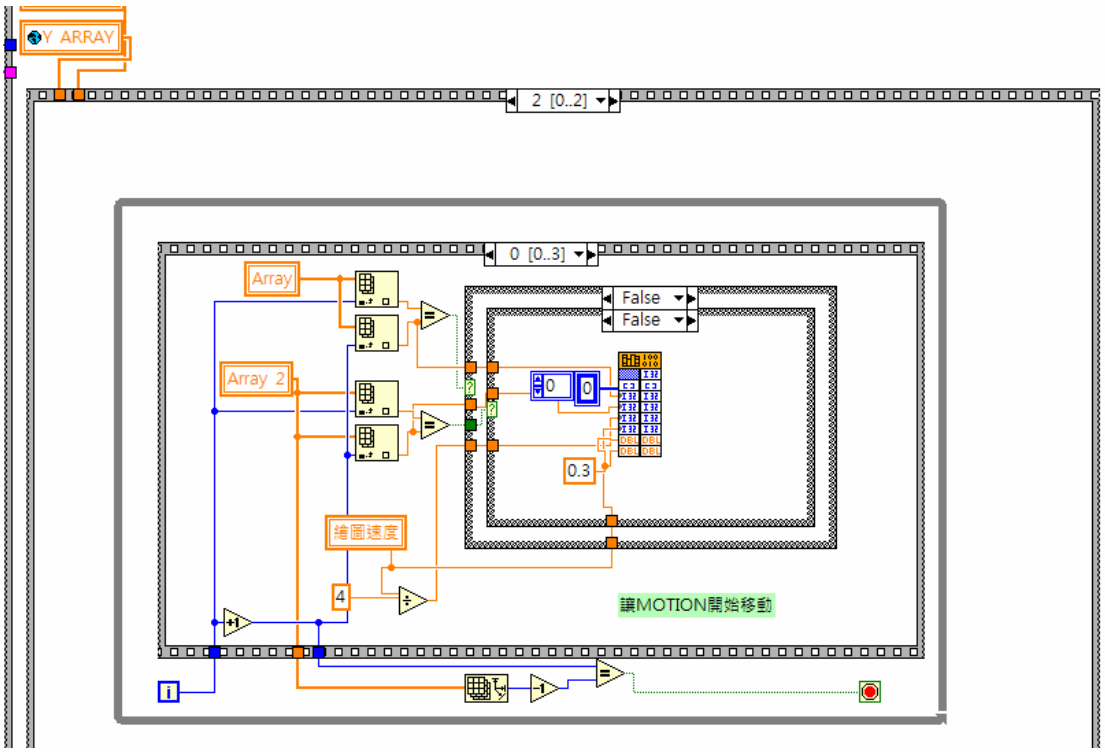

圖 3-97 GDSII 規格運動控制介面 LabVIEW 流程圖-開始移動(55)

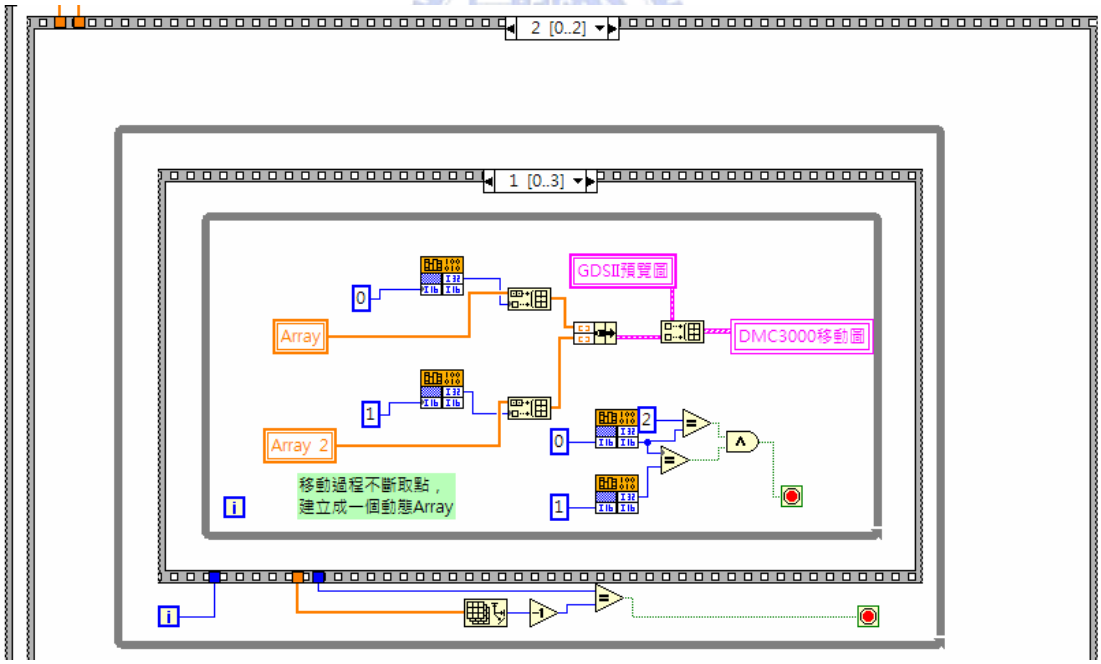

圖 3-98 GDSII 規格運動控制介面 LabVIEW 流程圖-繪製移動圖(56)

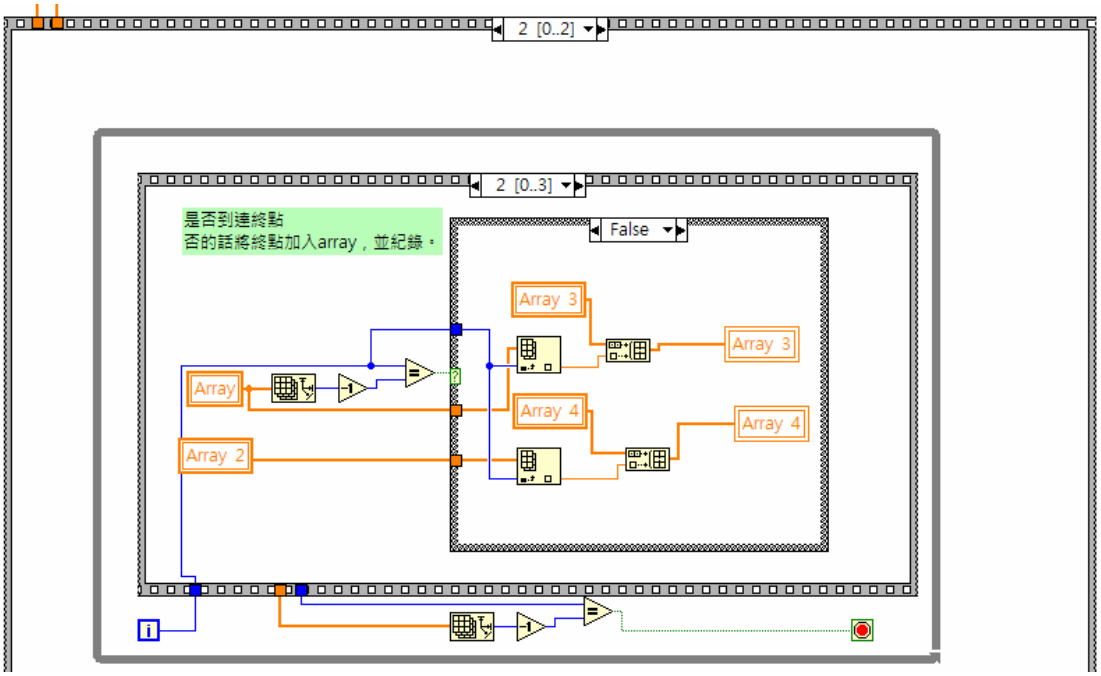

圖 3-99 GDSII 規格運動控制介面 LabVIEW 流程圖-判斷轉折點(57)

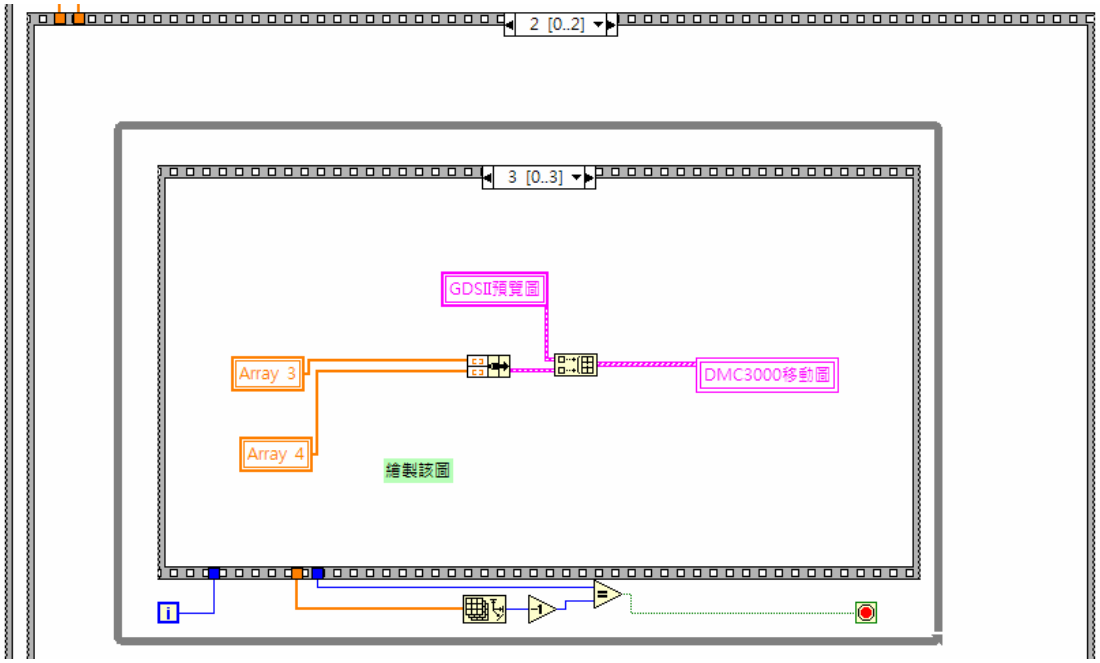

圖 3-100 GDSII 規格運動控制介面 LabVIEW 流程圖-繪製移動圖(58)

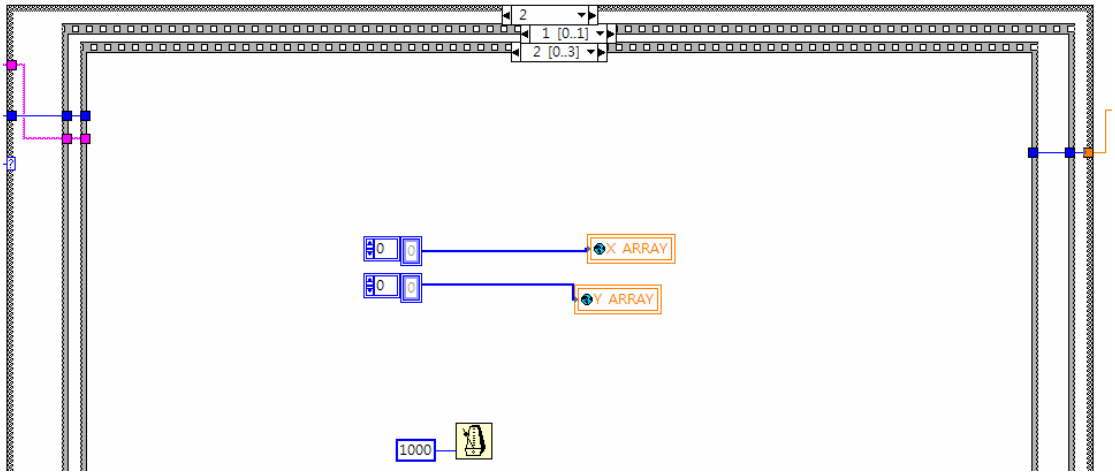

圖 3-101 GDSII 規格運動控制介面 LabVIEW 流程圖-初始化陣列(59)

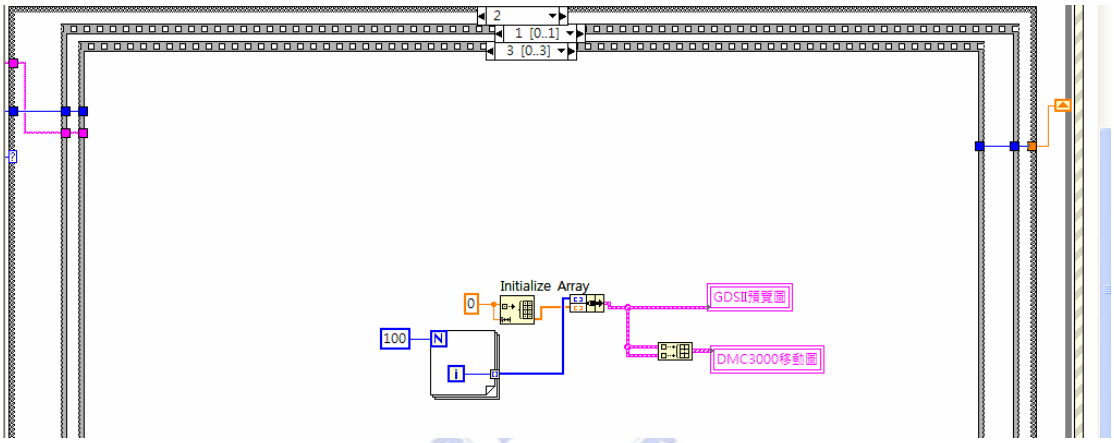

圖 3-102 GDSII 規格運動控制介面 LabVIEW 流程圖-清除圖形(60)

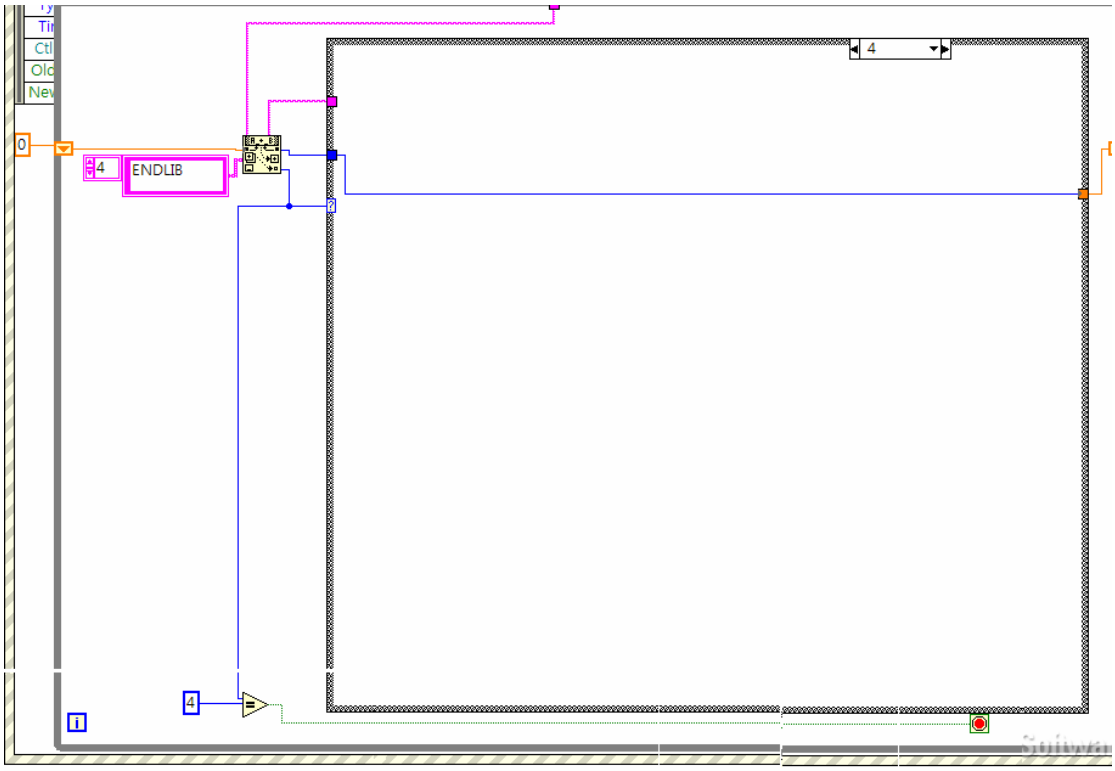

圖 3-103 GDSII 規格運動控制介面 LabVIEW 流程圖-結束繪圖(61)

接下來要製作呼叫圖形的部份，包含 SREF、AREF 指令。SREF 是只要指定一個結構(Structure)程式就會依照提供的座標，在該座標去繪製該 Structure 的圖形，讓使用者可以去呼叫已經繪好的圖形來節省繪製時間，在程式資料上也可以節省儲存空間。

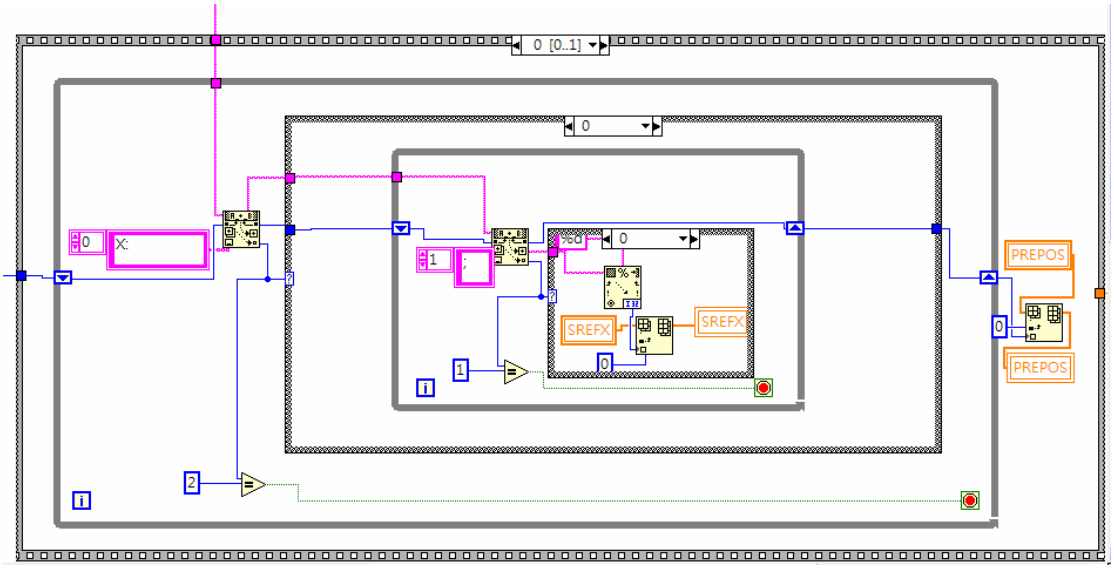

圖 3-104 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取相對座標 X(62)

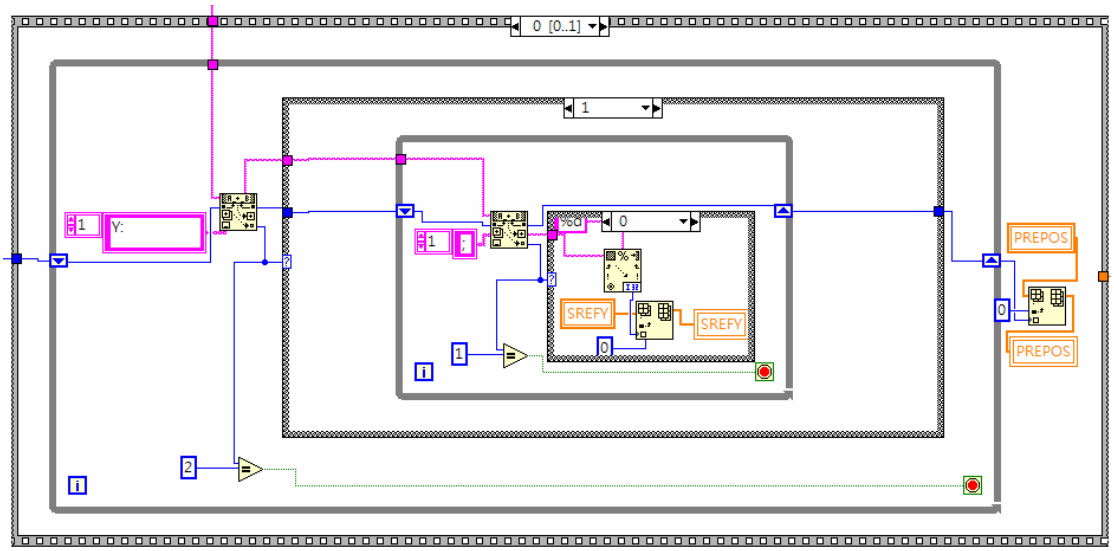

圖 3-105 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取相對座標 Y(63)

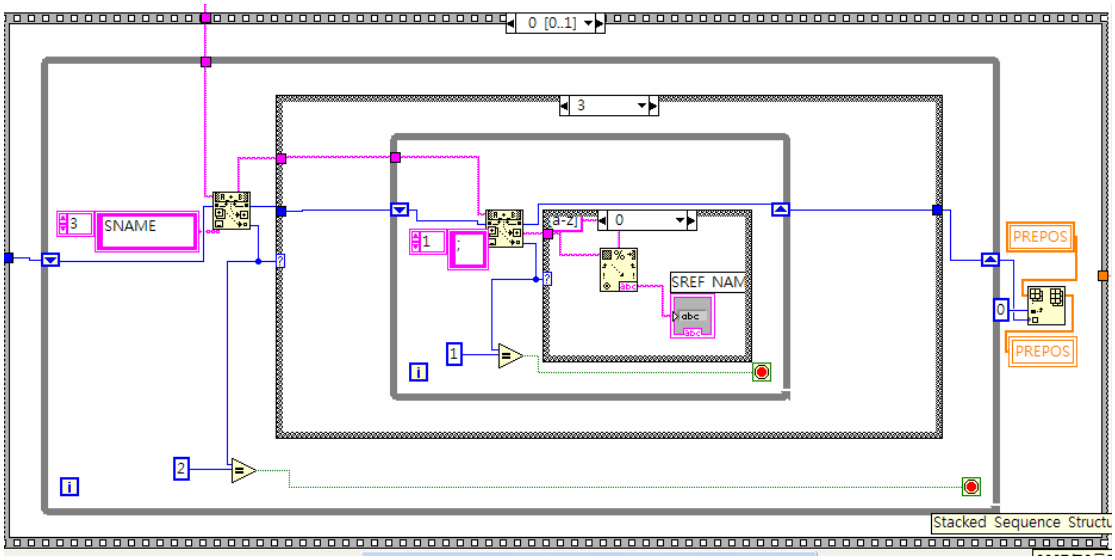

圖 3-106 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取呼叫結構名(64)

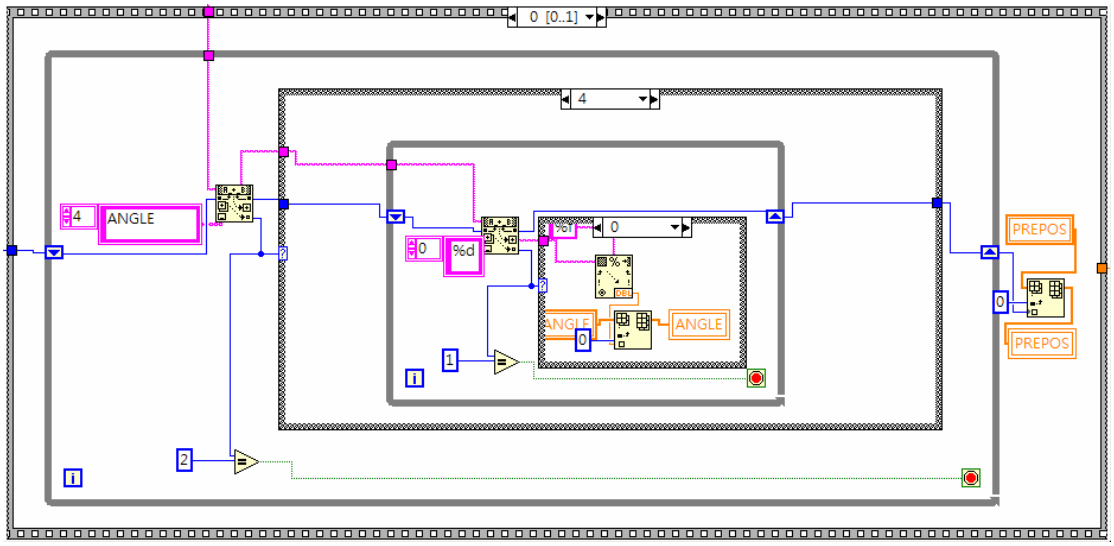

圖 3-107 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取旋轉角度(65)

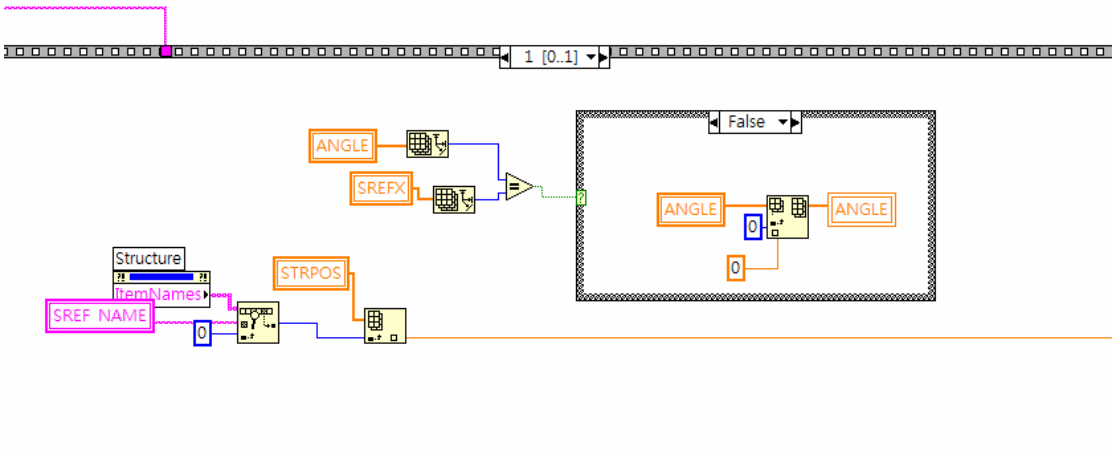

圖 3-108 GDSII 規格運動控制介面 LabVIEW 流程圖-呼叫結構位置(66)

AREF 類似於 SREF，不過他是呼叫結構(structure)並構成一個陣列。

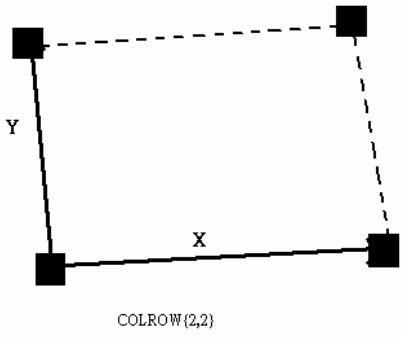

圖 3-109 陣列示意圖

如上圖，若是呼叫的結構為一個填滿的方格，並構成 2x2 的矩陣，繪製出來即會形成四個填滿的黑色方格。

程式如下:

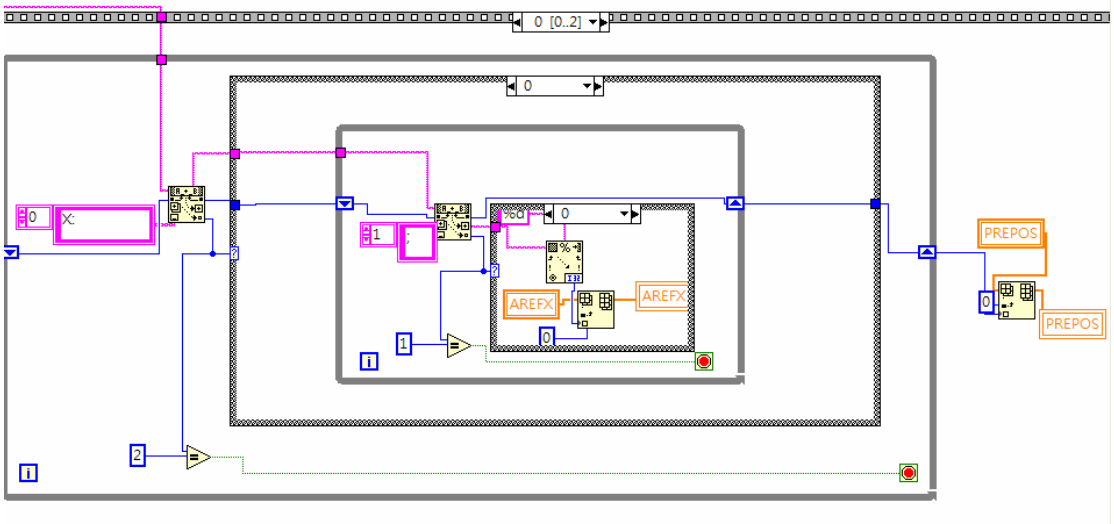

圖 3-110 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取陣列相對座標 X（67）

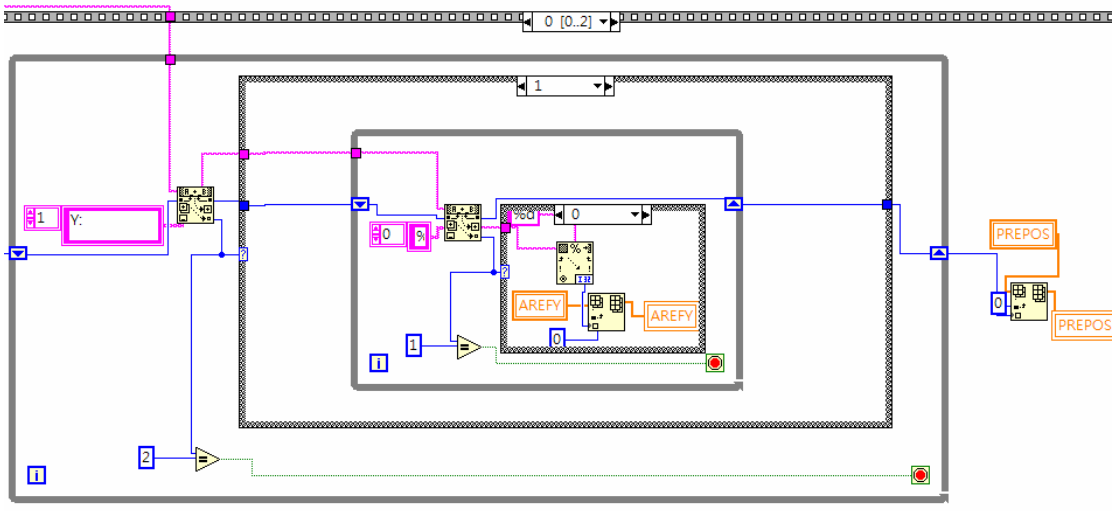

圖 3-111 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取陣列相對座標 Y(68)

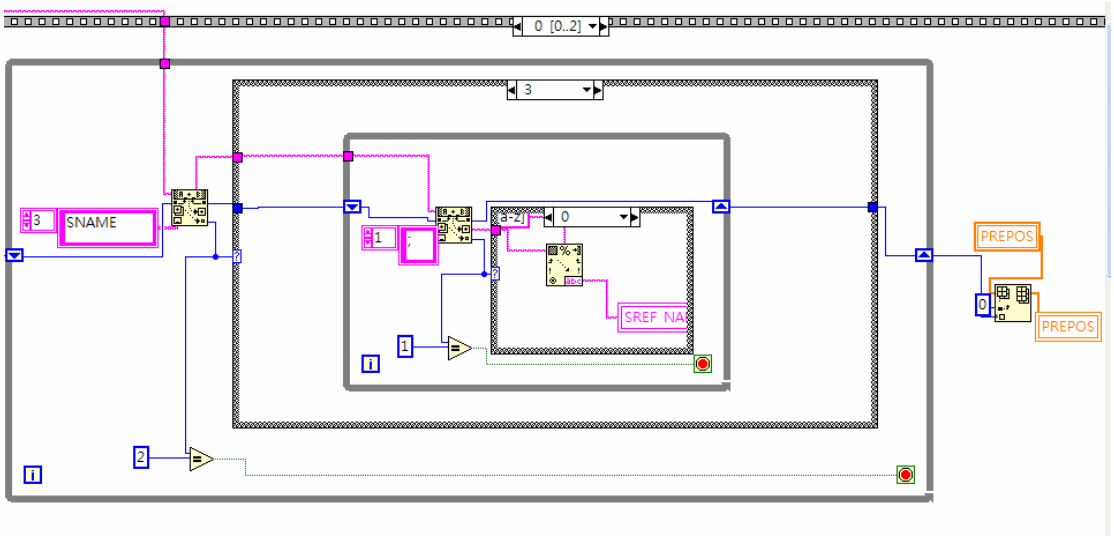

圖 3-112 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取呼叫結構名（69）

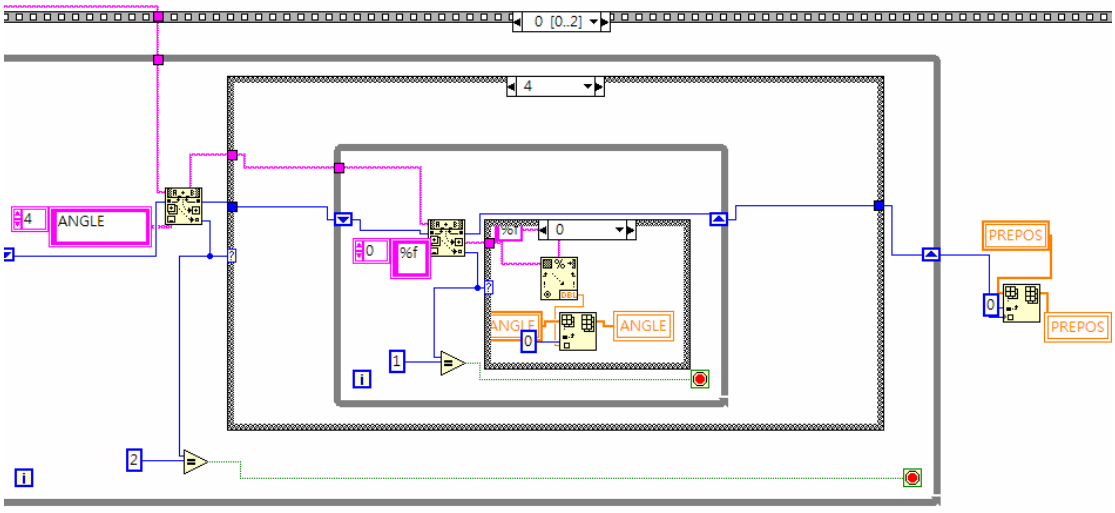

圖 3-113 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取旋轉角度(70)

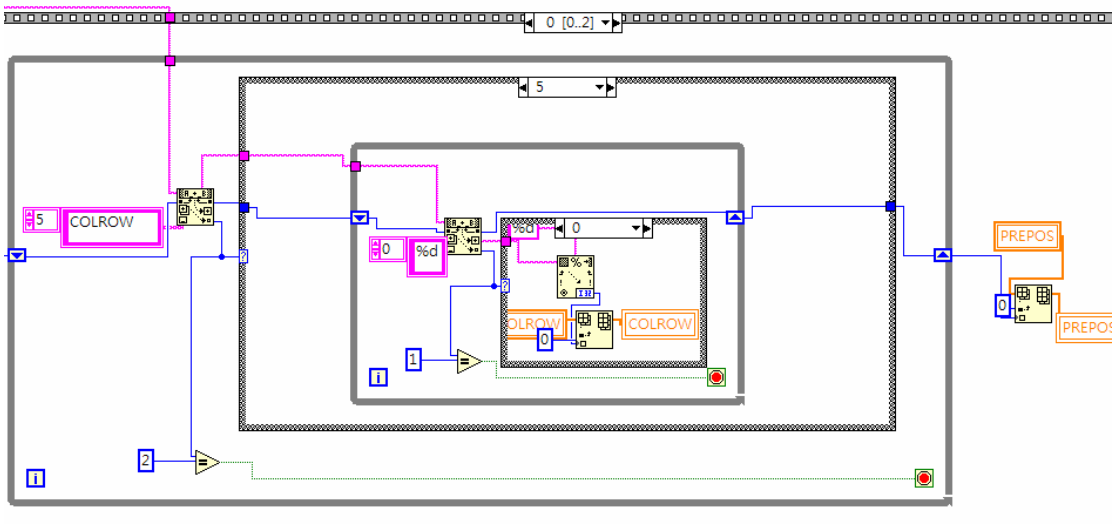

圖 3-114 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取行、列(71)

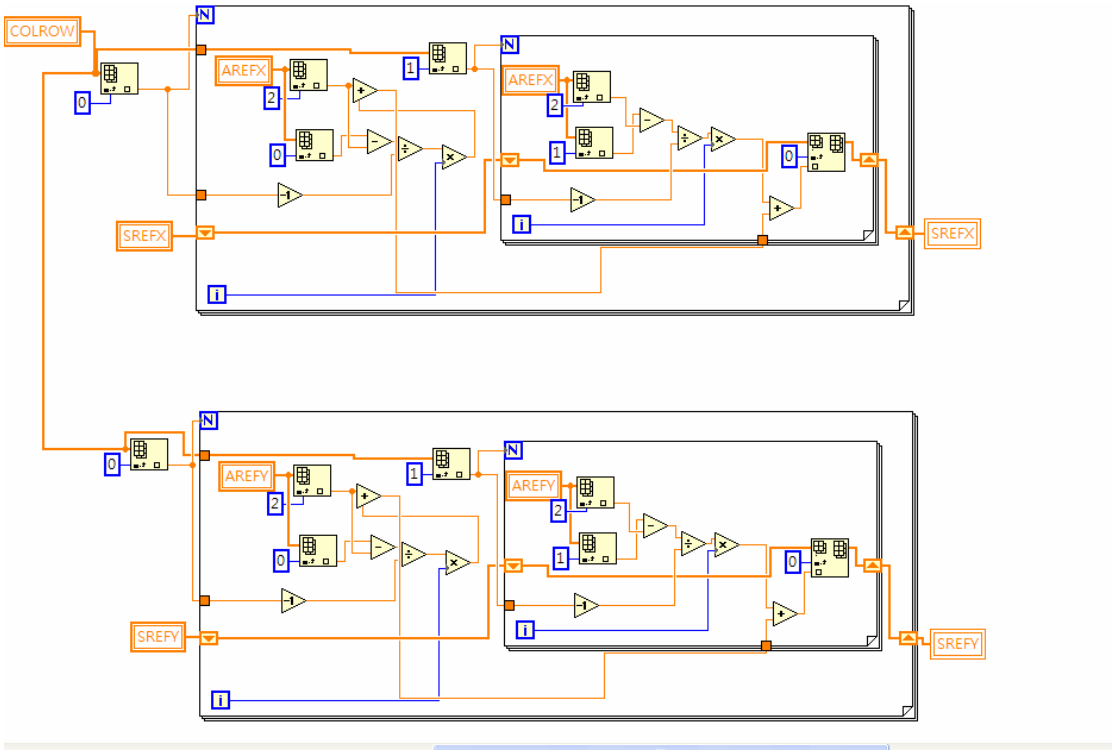

圖 3-115 GDSII 規格運動控制介面 LabVIEW 流程圖-計算各行列相對座標(72)

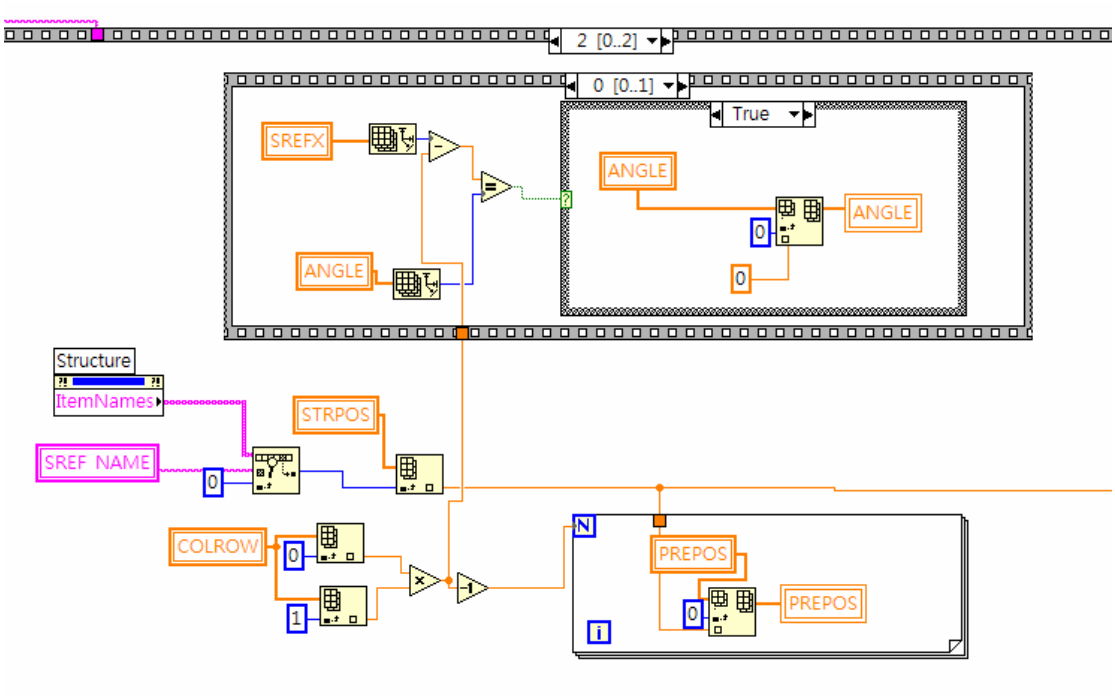

圖 3-116 GDSII 規格運動控制介面 LabVIEW 流程圖-輸出呼叫位址(73)

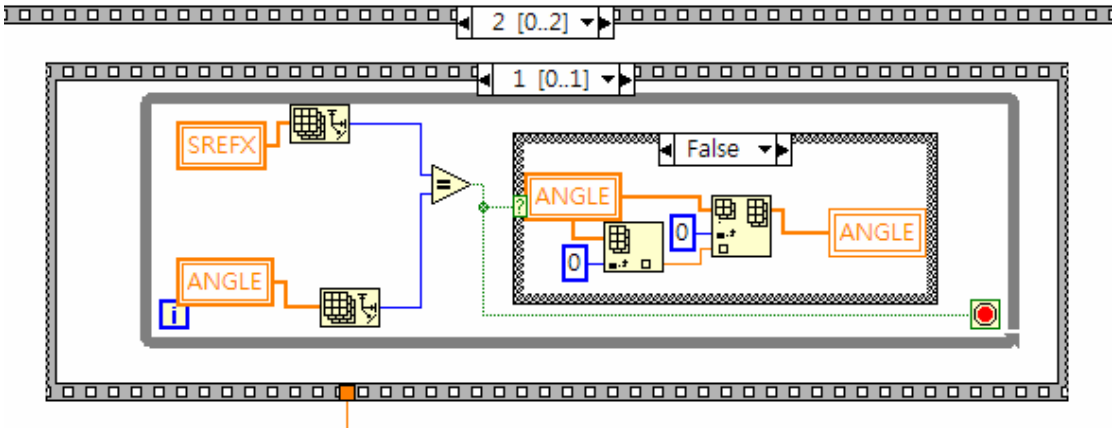

圖 3-117 GDSII 規格運動控制介面 LabVIEW 流程圖-輸出各點角度(74)

由於我們用到了讀取位置暫存器，當一個結構結束的時候，程式必須判斷是否為真正的結束議或只是完成一次呼叫的繪圖而已。判斷的程式如下:

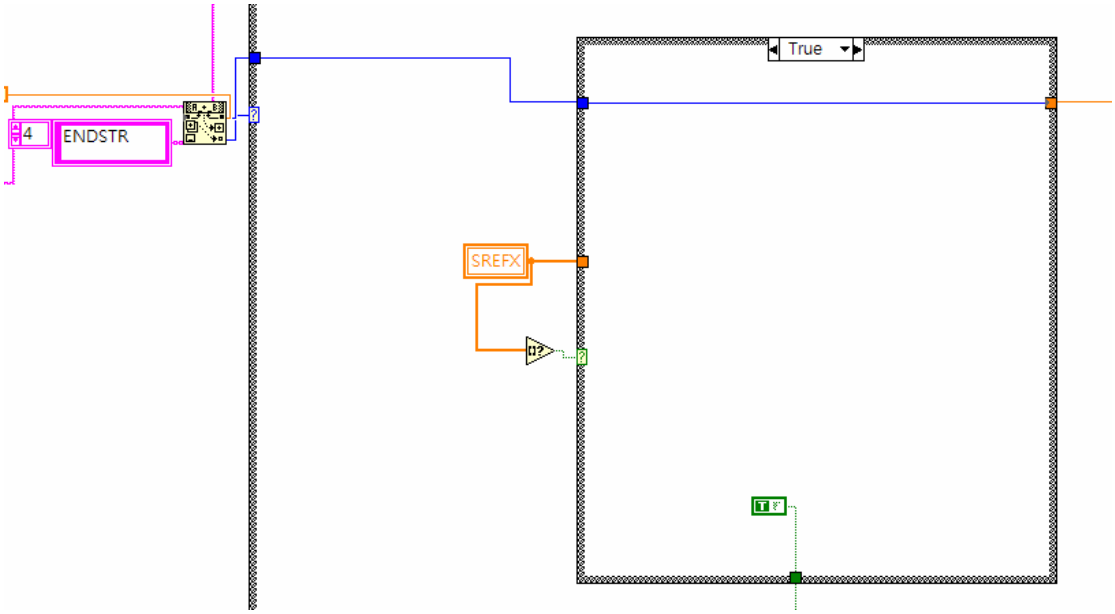

圖 3-118 GDSII 規格運動控制介面 LabVIEW 流程圖-無呼叫則結束繪圖 (75)

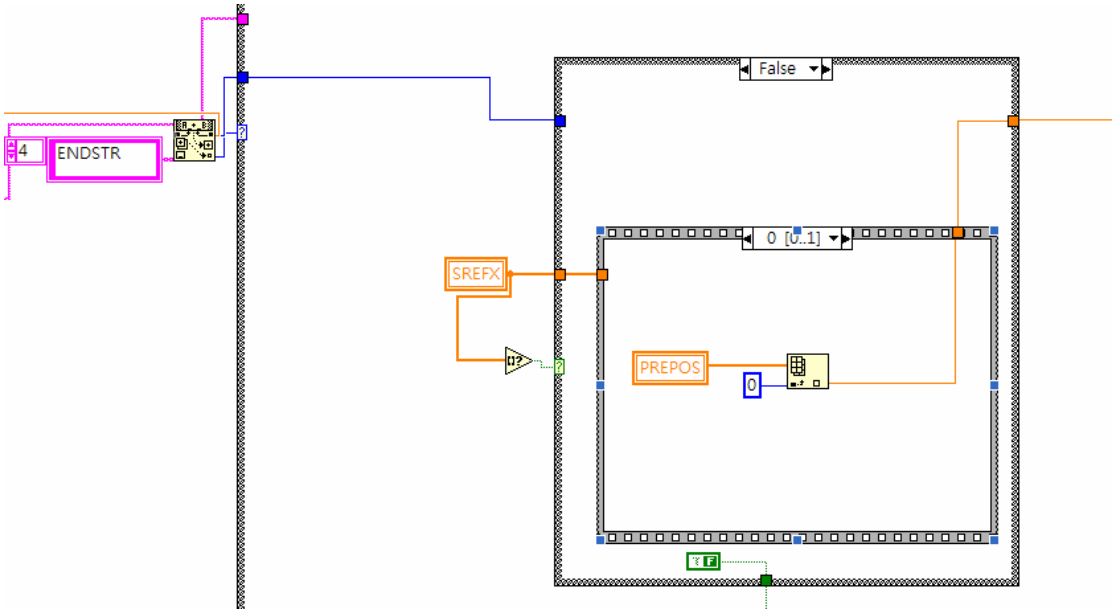

圖 3-119 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取呼叫位置(76)

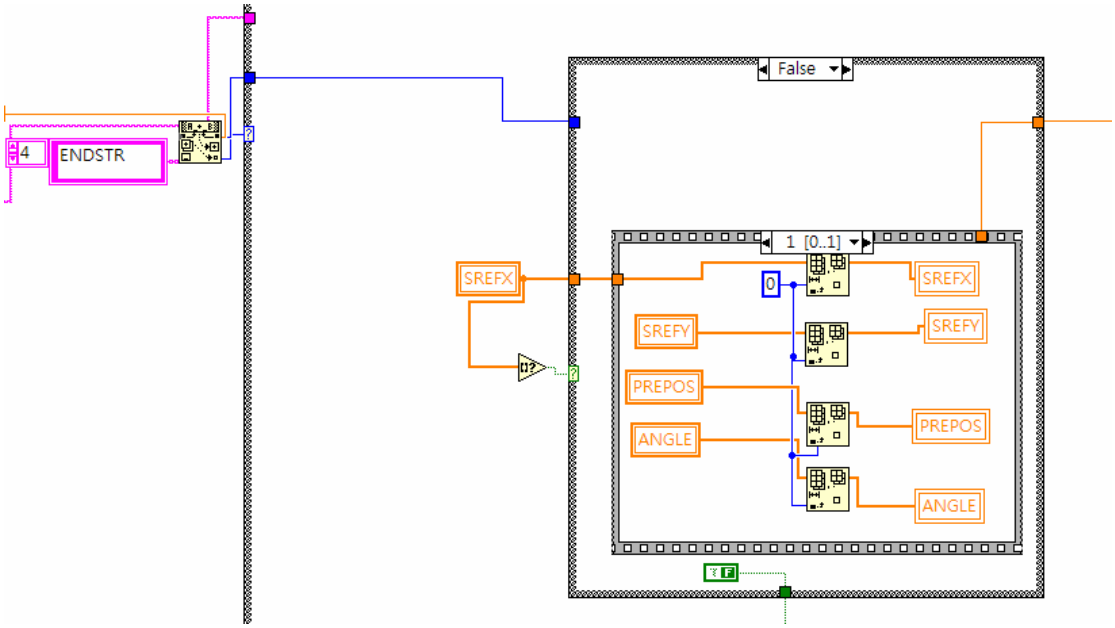

圖 3-120 GDSII 規格運動控制介面 LabVIEW 流程圖-讀取相對座標、角度(77)

到此我們終於完成了接受 GDSII 規格運動之介面。下一章將利用此次設計之介面進行實機驗證。

第四章

經過前面幾個章節的問題分析、系統設計後，在此章節將做驗證及討論。本章前半部份將實際挑選一個 GDSII 檔案，讓機台依照圖形運動並繪圖，後半部份將討論運動時所遇到的問題與改善方法。

4.1 實驗與結果

在模擬完各種 GDS II 程式指令無誤後，在本節將找一個電子元件 LAYOUT 圖檔，並依此圖檔讓機台繪製該圖。

首先將白紙置於機台上並固定，接著調整機台高度，如下圖。

圖 4-1 實驗圖(1)

接下來選擇各個不同圖層(Layer)，讓機器繪製圖形，這邊我們選擇接全部圖層皆繪在一起，實驗流程如下：

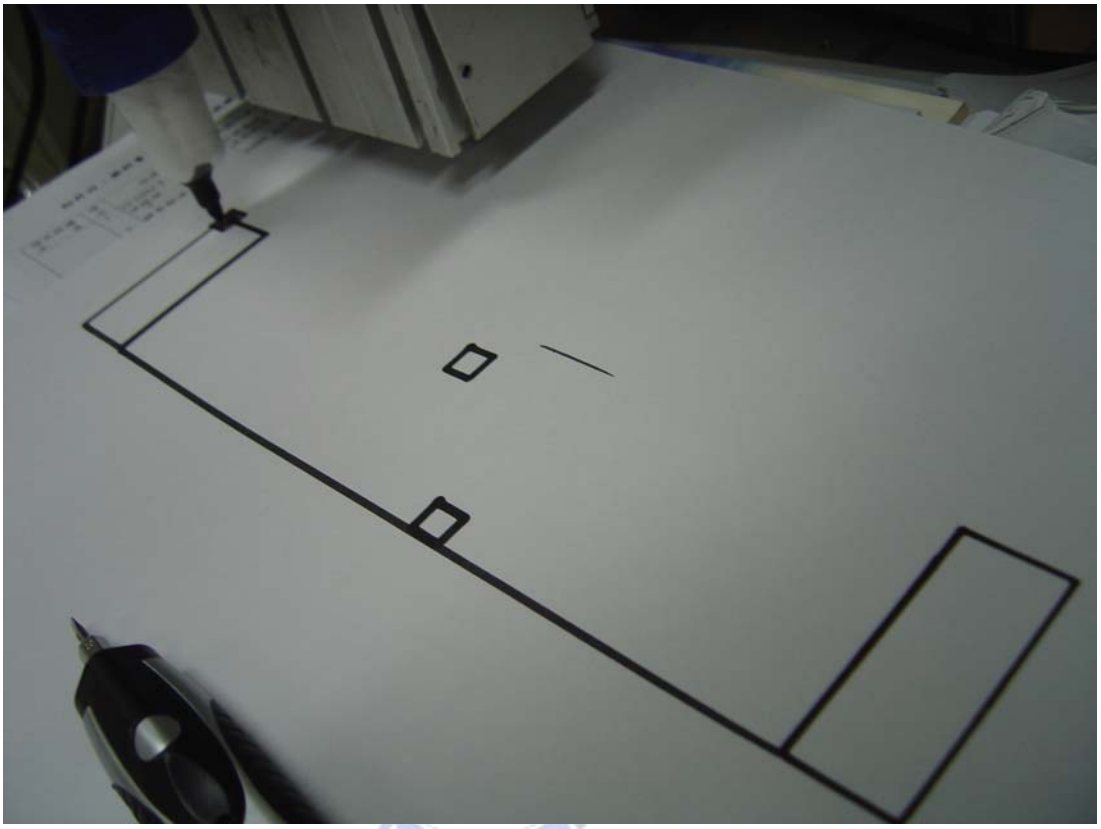

圖 4-2 實驗圖(2)

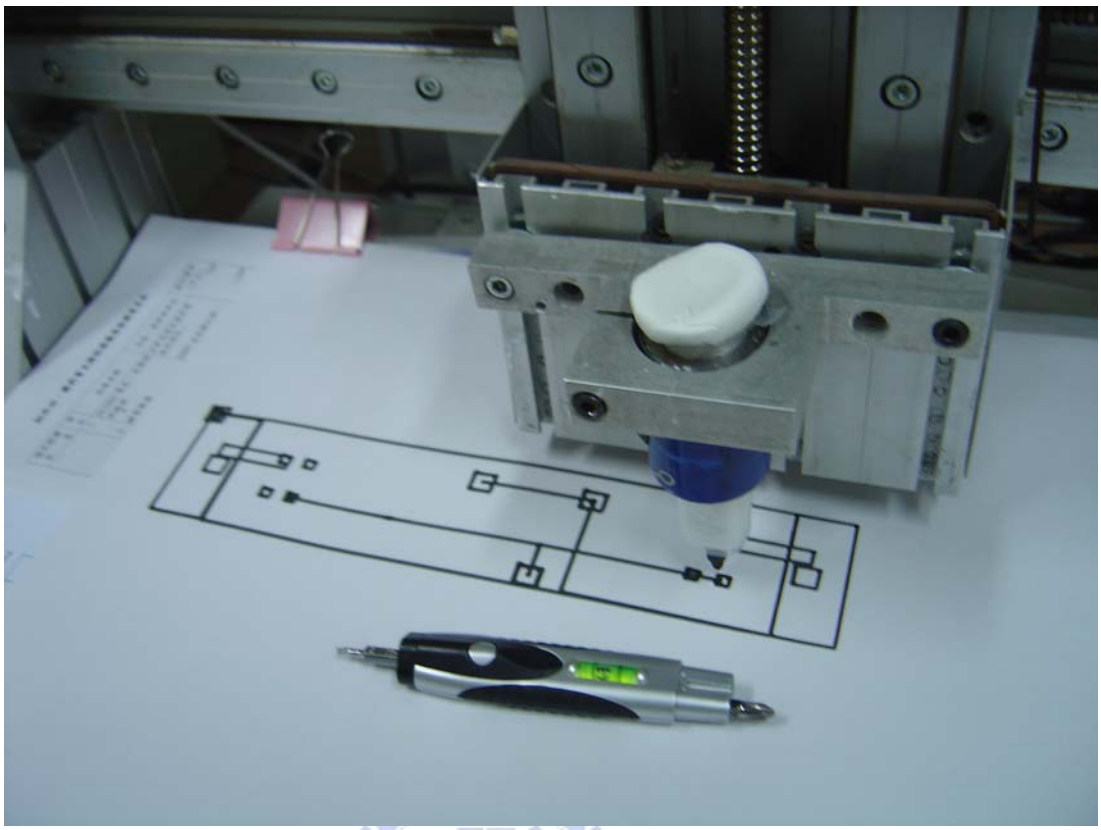

圖 4-2 實驗圖(3)

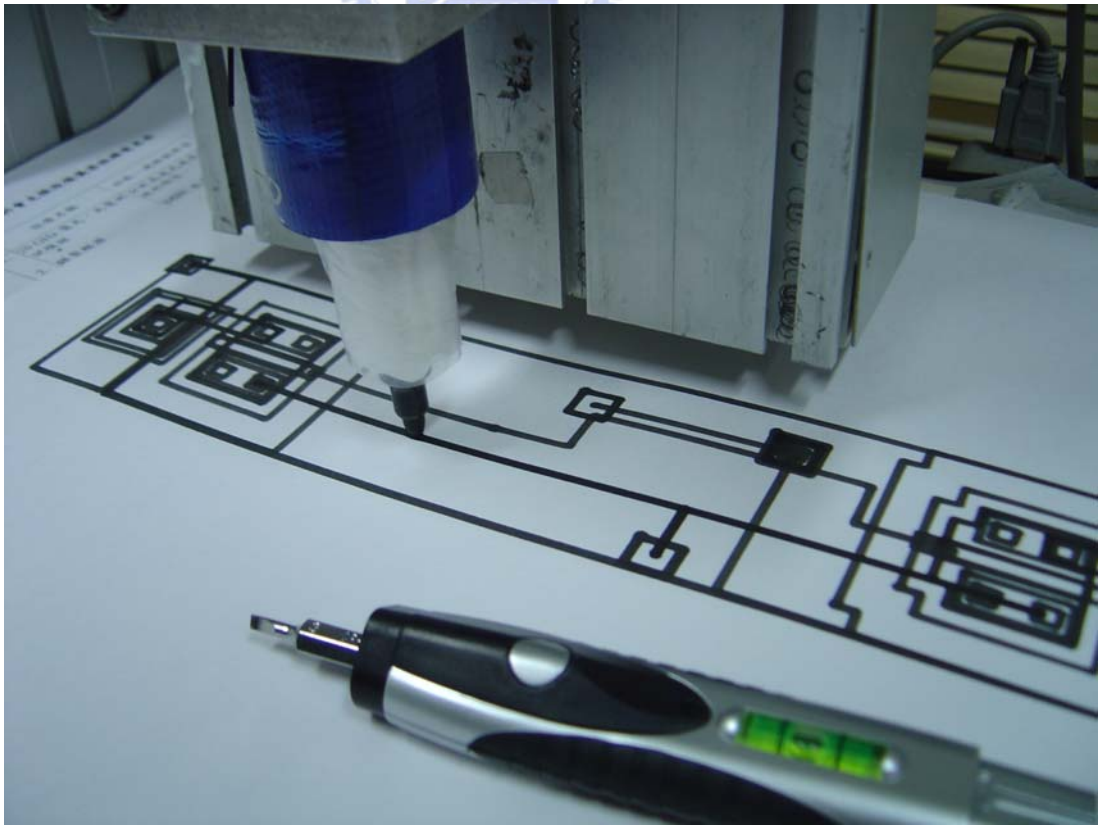

圖 4-4 實驗圖(4)

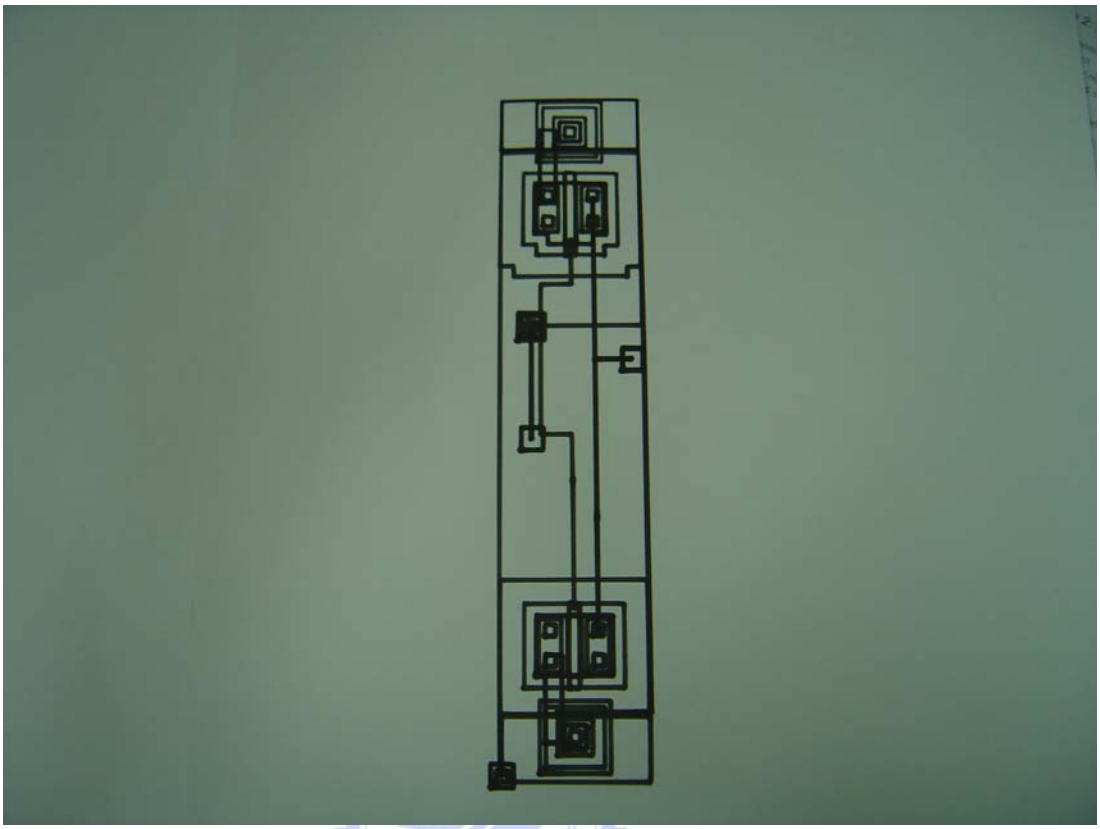

圖 4-5 實驗圖(5)

4.2 問題與改善

繪製完成後發現圖上的路徑，左下角都會有暈染開來的情形，一開始懷疑是在程式讀取和計算該通過的座標點時，因為等待而造成過度上墨的關係。

但是在仔細檢查後，發現該部分的計算並非十分複雜，所造成的延遲也很小，真正造成該部分有暈染的情況，是因為重複上墨所造成。

由於繪圖所使用的工具有一定的面積，這代表理論上應該是個細小的點，實際繪圖時卻是一個塗滿的小圓。這代表在線段的起始和結束處，將會重覆上墨。同樣的，在使用區域填滿功能時，由於周圍會重複繪製，因此也有如此的情

形。

要改善這樣的情況，得在繪製路徑的起點和終點時，預先留下繪圖工具最小點之半徑長度，同樣的，在繪製區域時，也得作此考量。

第五章
結論與未來研究方向

本研究對於運動平台的介面及功能做了圖控化的設計，在前面幾章作了完整的描述與討論。由運動平台的移動與操作、讀取 GDS II 檔案到最後驗證的 LAYOUT 模擬成果，相信在對於國內高科技產業可以略盡綿薄之力。此外，讀取 GDS II 檔案的演算法及其修正的方法，希望可以對以後的機台讀取其他種類的圖形檔運動的研究，提供一個方法及路徑。

5.1 貢獻

1. 本論文發展之圖控之人機界面程式，比傳統的文字介面程式更易於使用，即使是初學者，也可以在短短一兩次的操作後即可熟析。應用於高科技產業，更可縮短訓練操作人員的培訓時間，以求快速投入生產。
2. 操作機台運動不再只是輸入方向或是坐標，由於其具有可接受 GDSII 圖檔之自動化運動系統，作業人員可以輸入圖檔後再讓機器自動運作，不易發生人為操作失誤而導致失敗產品。
3. 本論文之設計概念，可做為未來接受其他規格圖檔設計之考量和指引，由於圖控語言程式十分容易修改，只要在關鍵幾處修改，即可

讀取其他規格之圖檔，且三維機台應用廣泛，本系統當可為各種應用之藍本，視需求而加以修改。

5.2 未來研究方向

1. 因為驗證時使用的繪圖裝置描繪線條粗寬，應用較為侷限;未來將使用較精密的繪圖裝置大，對更細微的物體進行精密繪製。
2. 實驗中使用的步進馬達，因為扭矩較小; 將來做高科技產業之應用，當改用大型馬達以提高繪圖速度。
3. 因應繪圖線寬而導致圖形重複部份，用軟體設計加以預留空間，以改善過度渲染或曝光的部份。

5.3 未來的應用發展

本研究討論了許多圖控運動平台設計上的方法，完成之介面加以特製化後當可應用於繪圖機、自動塗膜機以及電子束直寫機等等；亦可在其上加裝 LVDT 等偵測系統，按照特定軌跡來做電腦自動化模組檢測，應用於手機、CD 隨身聽外殼檢測等等。

參考文獻

[1]http://www.nsc.gov.tw/_newfiles/popular_science.asp?add_year=2006&popsc_aid=82

[2] http://www.allwiki.com/wiki/%E7%BB%98%E5%9B%BE%E6%9C%BA

[3] http://www.digit-life.com/articles2/intel-65nm/

[4] http://www.wooddoor.com.cn/company/DSC_0012.jpg

[5] http://nano.nchc.org.tw/dictionary/Optical_Lithography.html

[6] http://tw.myblog.yahoo.com/wei-tai/article?mid=11&prev=13&l=f&fid=5

[7]http://nano.nchc.org.tw/dictionary/ebl.html

[8] http://www.nanolab.itri.org.tw/Lab/EquCatalogdetail.aspx?nano_equno=8107740002 1

[9] Visual Language Theory by Kim Marriott Bernd Meyer

[10]http://en.wikipedia.org/wiki/Visual_programming_language

[11] http://home.rexsong.com/?p=153

[12] http://chinesetrad.joelonsoftware.com/uibook/chapters/1.html

[13] 曾嘉德 “A PCI Interfaced IEEE-488 Card”

[14] 侯杰利 “Curve tracked novel coating”

[15] http://www.xs4all.nl/~kholwerd/interface/bnf/example.html

http://163.23.210.65/w_1.htm

作者學經歷

建國高級中學

交通大學電子工程學系

交通大學光電顯示科技產業碩士專班