LabVIEW programming for beginners

fundamental.1

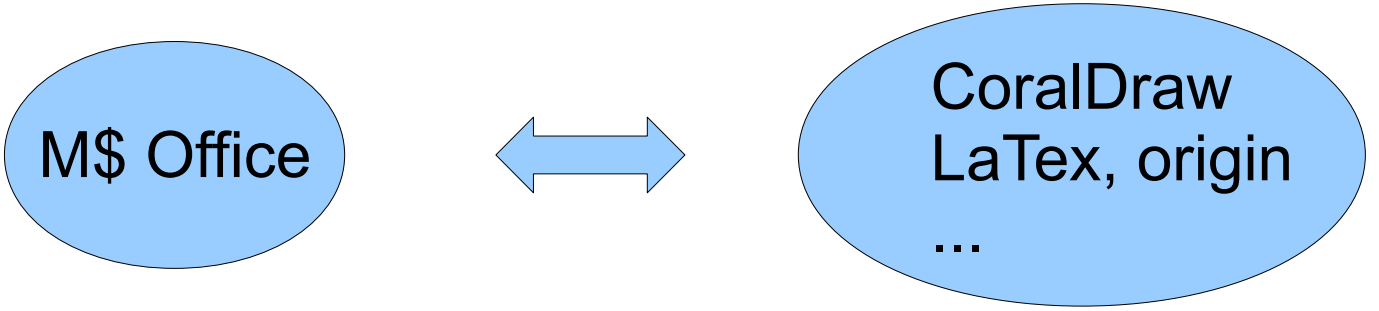
林致翰
b95901149@gmail.com
clin@hypro.com.tw

Why LabVIEW?

1. 強大的使用者介面
2. 用硬體綁架軟體
(Laboratory Virtual Instrument Engineering Workbench)
3. 快速 prototyping
4. 易學難精的學習曲線

LabVIEW ⟺

OpenCV (vision)
Matlab (data analysis& Visualization)
python/C
Verilog/VHDL (FPGA)
VisualBasic (GUI interface)



包山包海

很多人認為 LabVIEW 入門簡單沒有技術成分，其實只是沒有學到那麼深入（可以寫出一個不用按 Abort 就能正常結束的程式才算開始，很多人還不到這個程度就先畢業了，這又再一次證明 LabVIEW 受到各行各業 lab 青睞的原因）

官方認證牌位 (用途不大 , CLA 除外)

Architect

CLAD < CLD <<<<<<< CLA

40 道單選題 /2hr

2 周

完整功能程式 /4hr

三個月

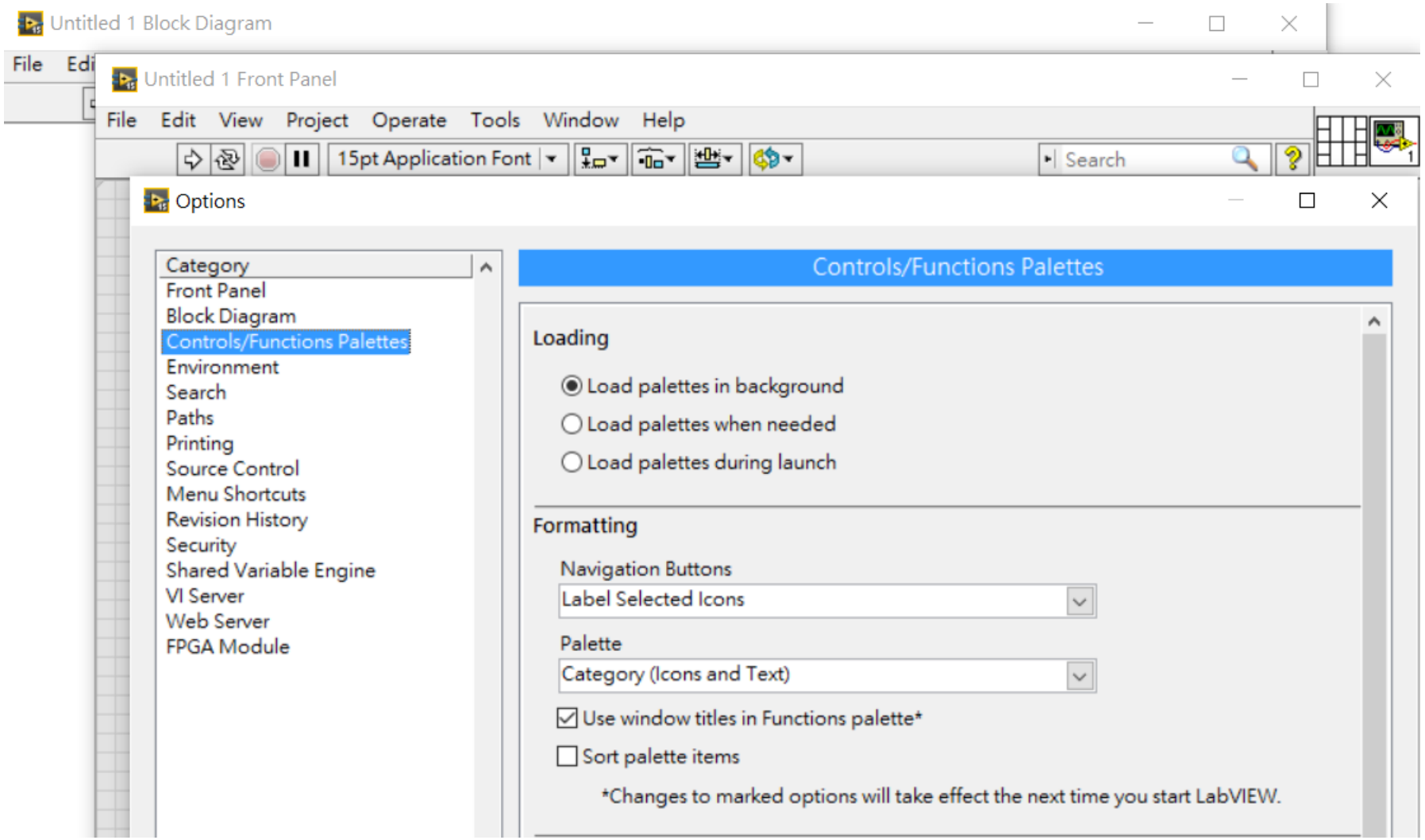
符合軟體規格書
數十個 tag /4hr

1yr ++

以開始工作才開始接觸 LabVIEW 來說
通常在你考到 CLA 之前就升到不需要 CLA 來證明
自身實力的職位

基本設定

Tool → option→ Controls/functions Palettes 照底下改



基本設定

Ctrl + B → 消除多餘接線
Ctrl + E → 切換 front panel 跟 block diagram

選取物件 + 方向鍵 → 移動物件
選取物件 +Shift (壓著)+ 方向鍵 → 移動物件 (移動距離較大)

選取物件 +Ctrl (壓著)+ 左鍵點擊移動 → 複製物件
空白處左鍵點擊移動 +Ctrl (壓著) → 增加空隙

Shift + 右鍵 → 格式選單 (tool palette, 改物件屬性 , 顏色等等)

Ctrl + space : quick drop , 超好用但可能跟 win 快捷鍵衝突 , 需要自訂
(比如說 Ctrl+A)

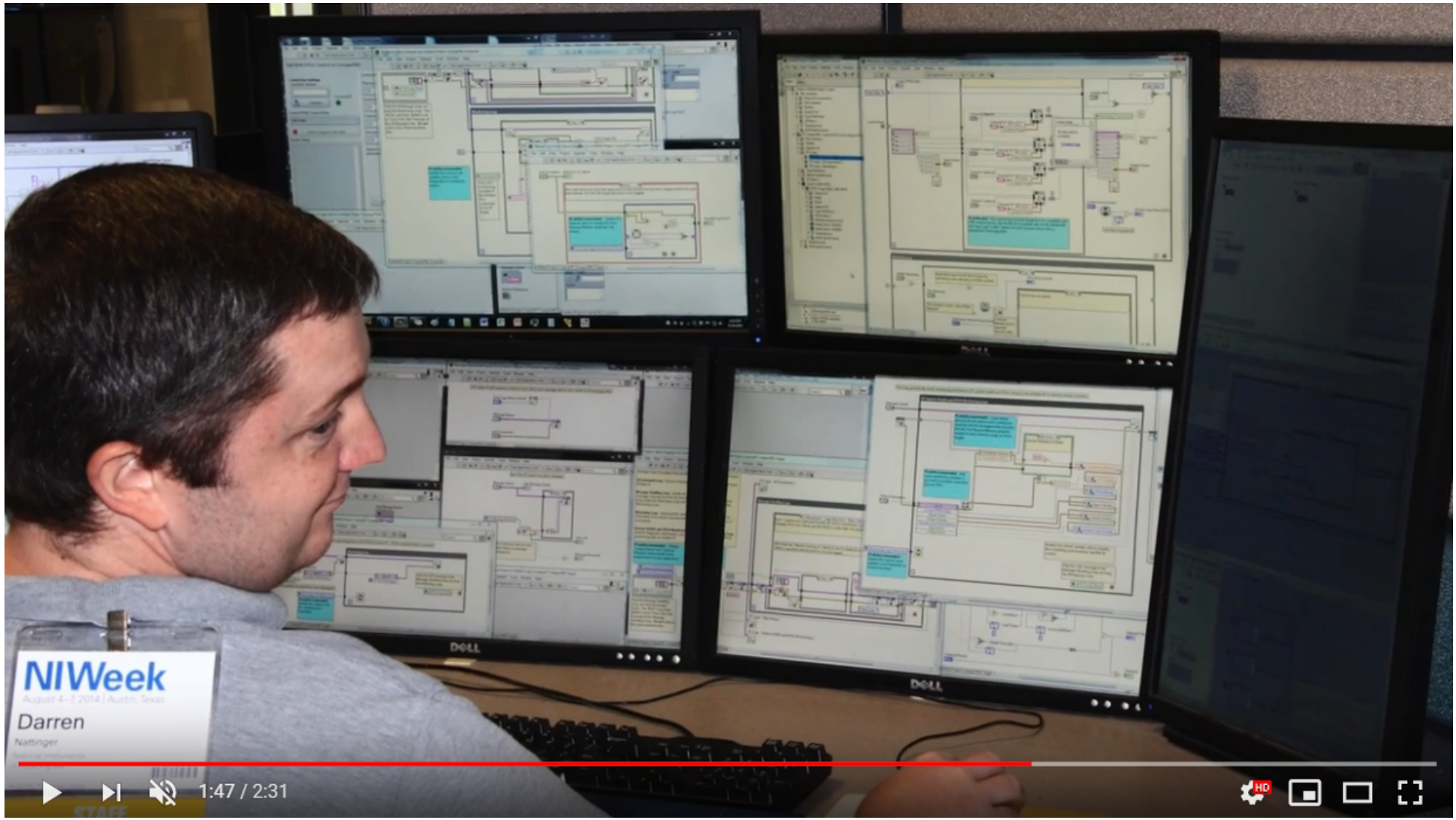
Quick drop (after LV2009)

神技，想要增加效率一定要背一些快速組合

Quick drop + <Ctrl +D> 指定 function 自動生成所有輸入輸出
Quick drop + <Ctrl +T> 指定 function 自動排列變數名稱
Quick drop + <Ctrl +W> 指定 function 自動接線

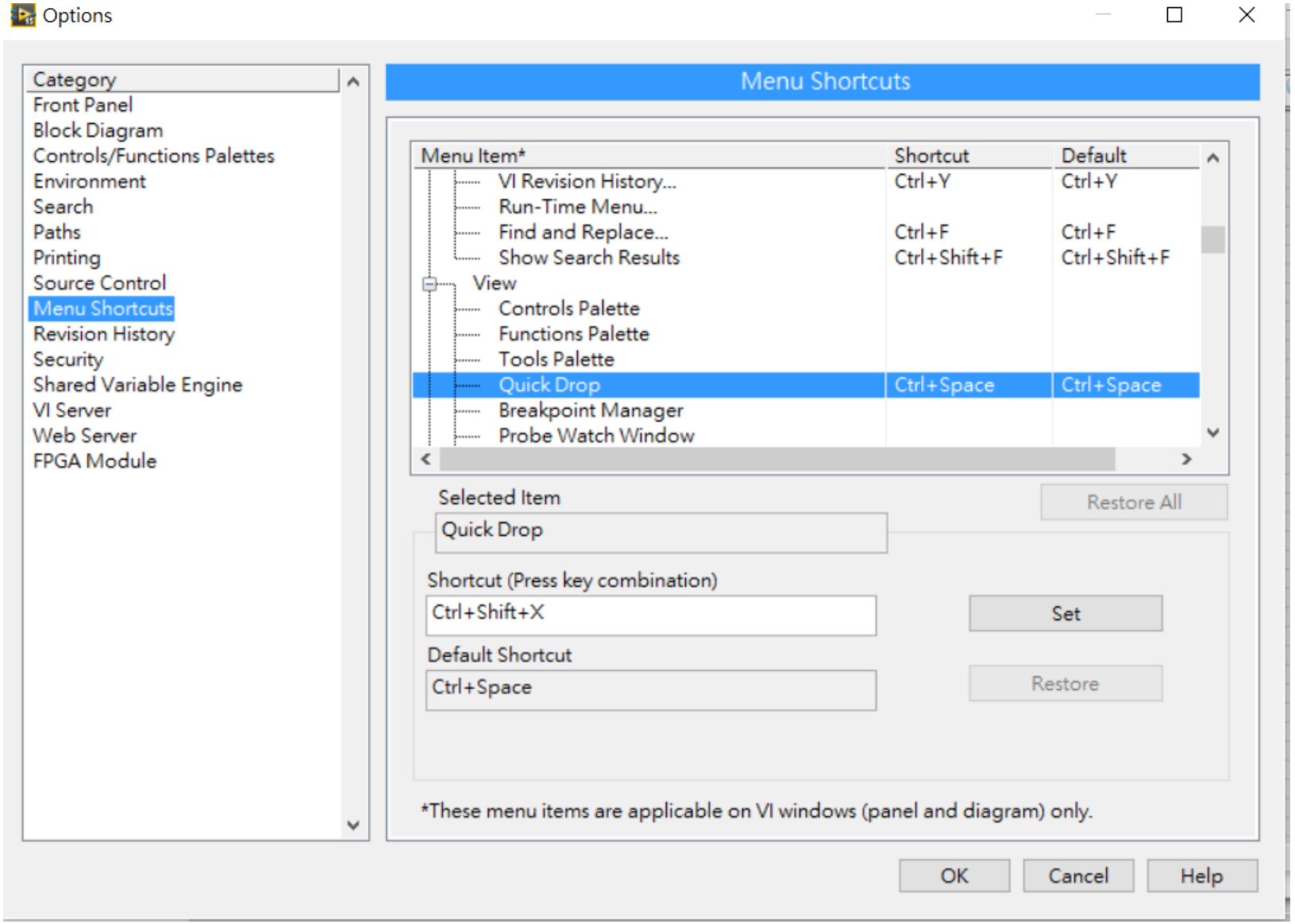
D → auto Drop & Input&Output
T → auto Text
W → auto Wire

World's Fastest LabVIEW Developer



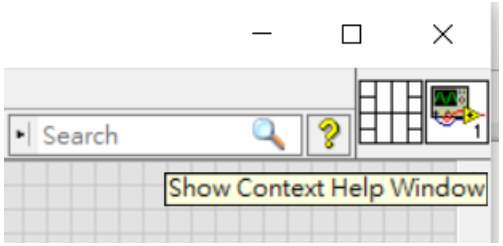
自訂快捷鍵

Tool → option→ Menu Shortcuts

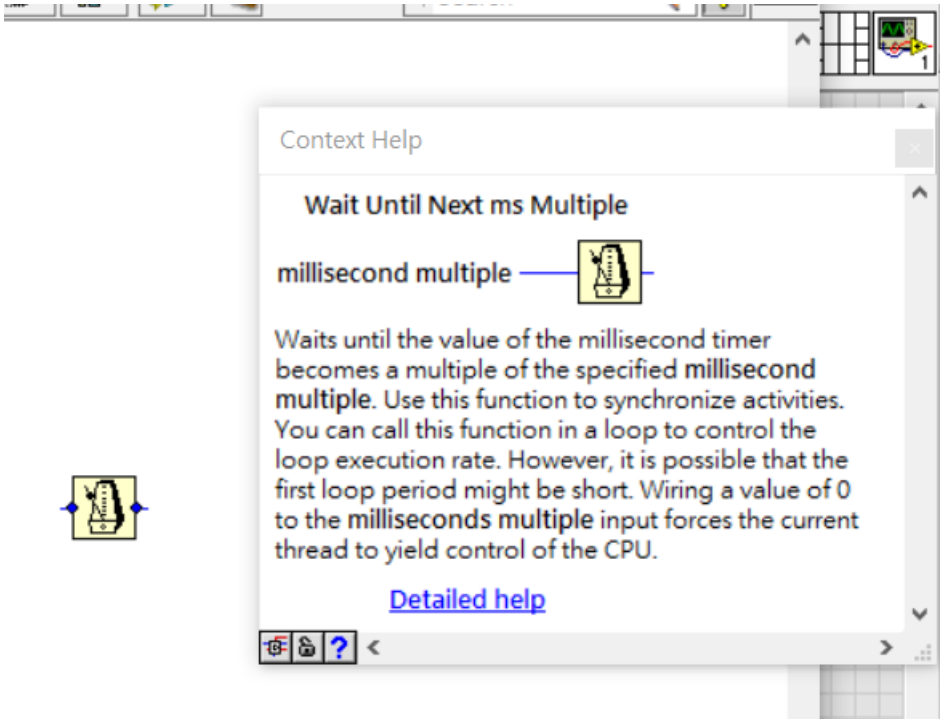


智慧接線切換 (綠色的 button)

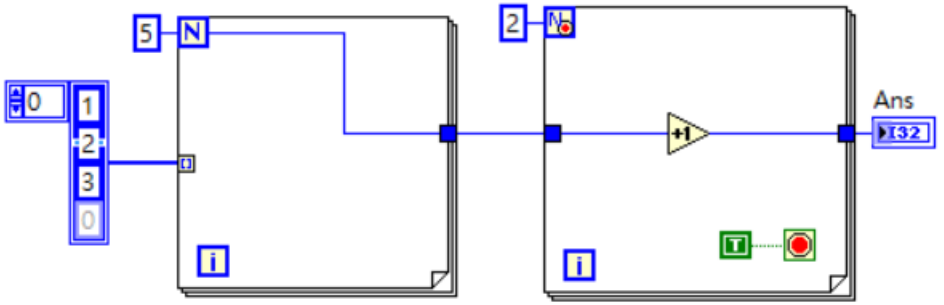
指令查詢



點擊 help button, (vi icon 旁的問號)
之後滑鼠移動到任意子 vi 後會顯示說明



迴圈

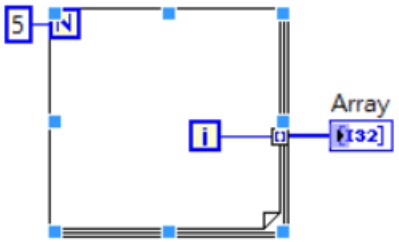


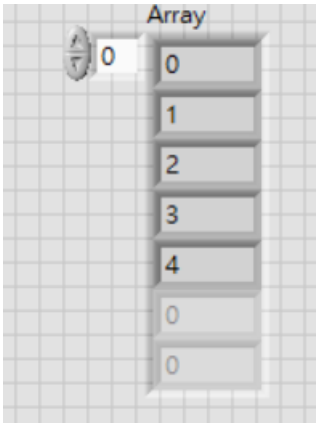
ANS 的值為？

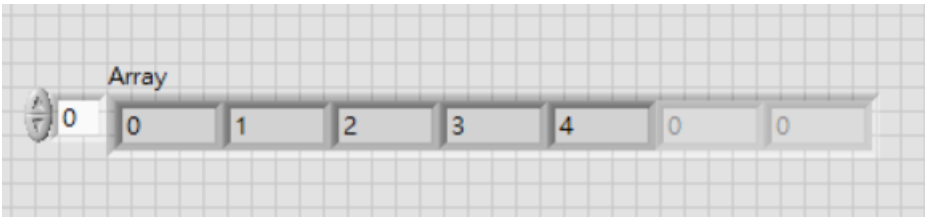
(a) 3
(b) 4
(c) 5
(d) 6
(e) 7

正確答案是 (b) 4， 如果你知道 4 是怎麼算出來的代表你對 LabVIEW 的 For 迴圈有一定程度的理解，可以跳過這個章節

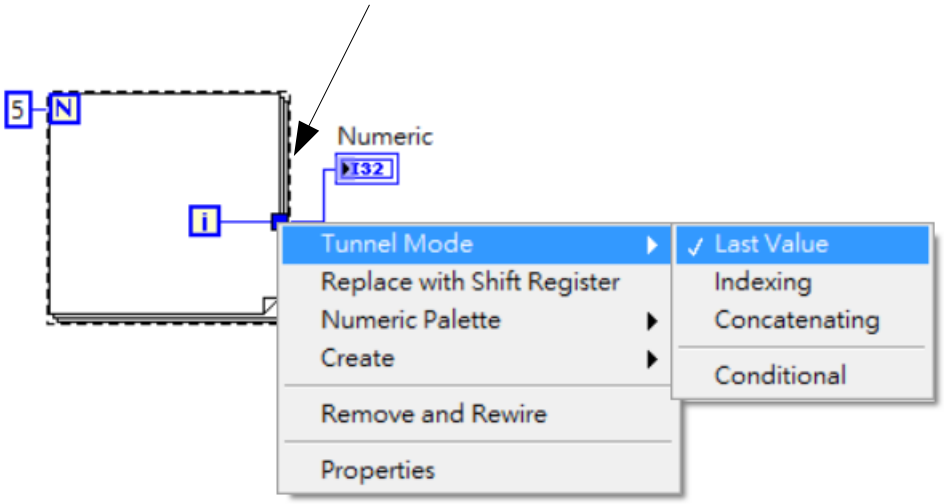
迴圈

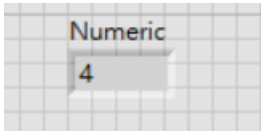

5
N
i
Array
I32


Array
0
0
1
2
3
4
0
0

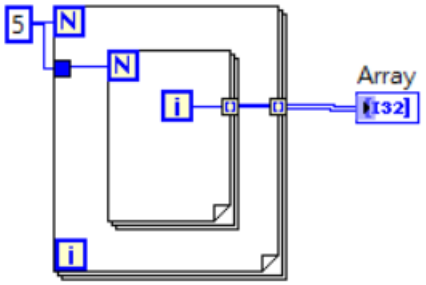

Array
0
0
1
2
3
4
0
0

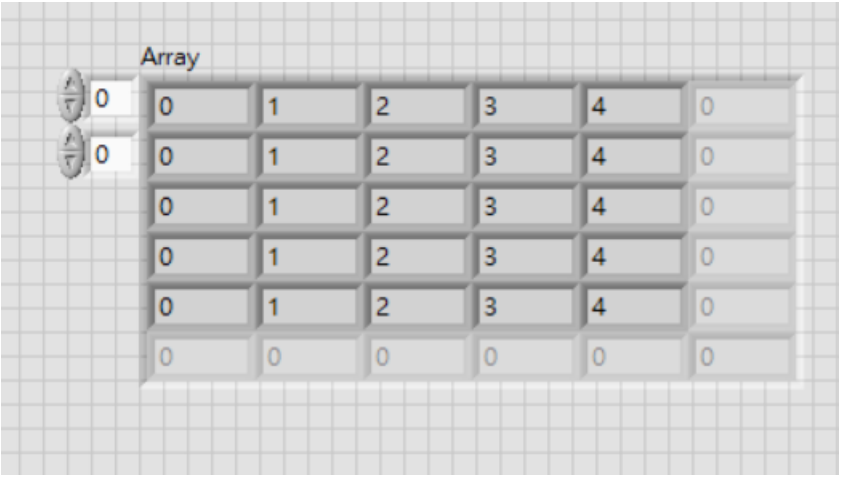
實心正方形


5
N
i
Numeric
I32
Tunnel Mode
Replace with Shift Register
Numeric Palette
Create
Remove and Rewire
Properties
Last Value
Indexing
Concatenating
Conditional

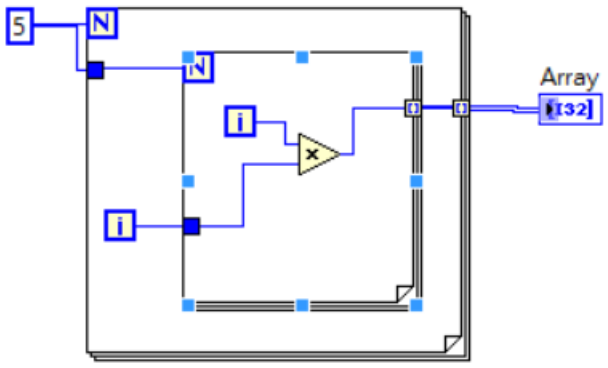

Numeric
4

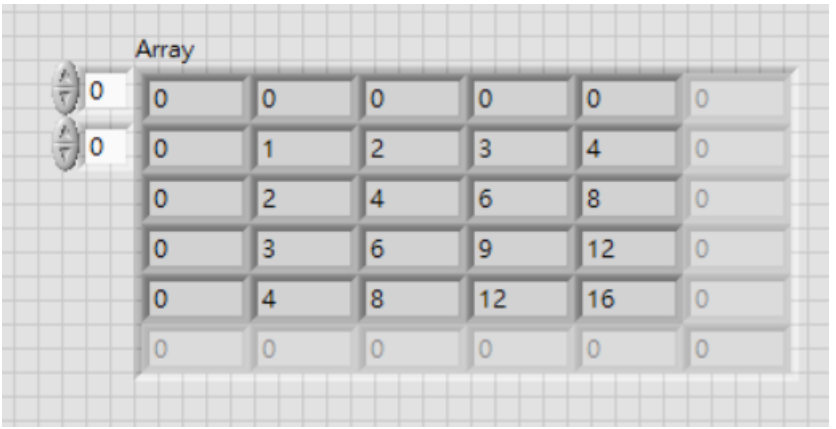
巢狀迴圈


5
N
N
i
i
Array
[32]


Array
0
0
0 1 2 3 4 0
0 1 2 3 4 0
0 1 2 3 4 0
0 1 2 3 4 0
0 1 2 3 4 0
0 0 0 0 0 0

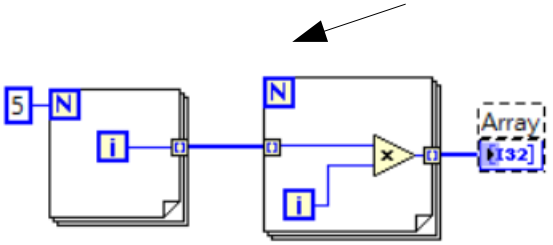
巢狀迴圈

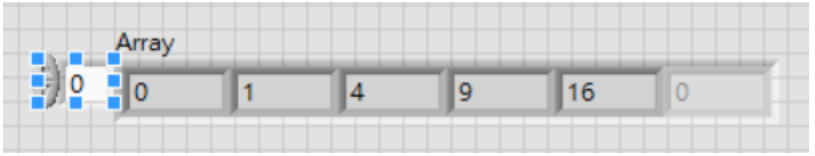

5
N
N
i
×
i
Array
[32]

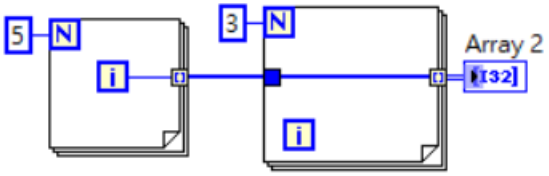

Array
0
0
0 0 0 0 0 0
0 1 2 3 4 0
0 2 4 6 8 0
0 3 6 9 12 0
0 4 8 12 16 0
0 0 0 0 0 0

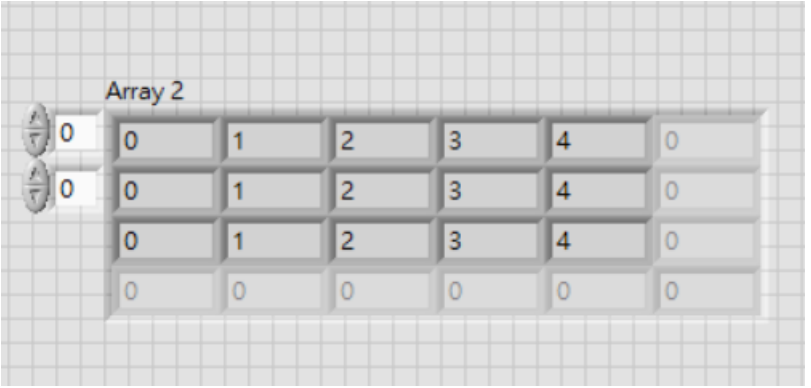
迴圈的其他功能

輸入端用 auto-index (雙正方形), 迴圈可以不指定圈數自動配

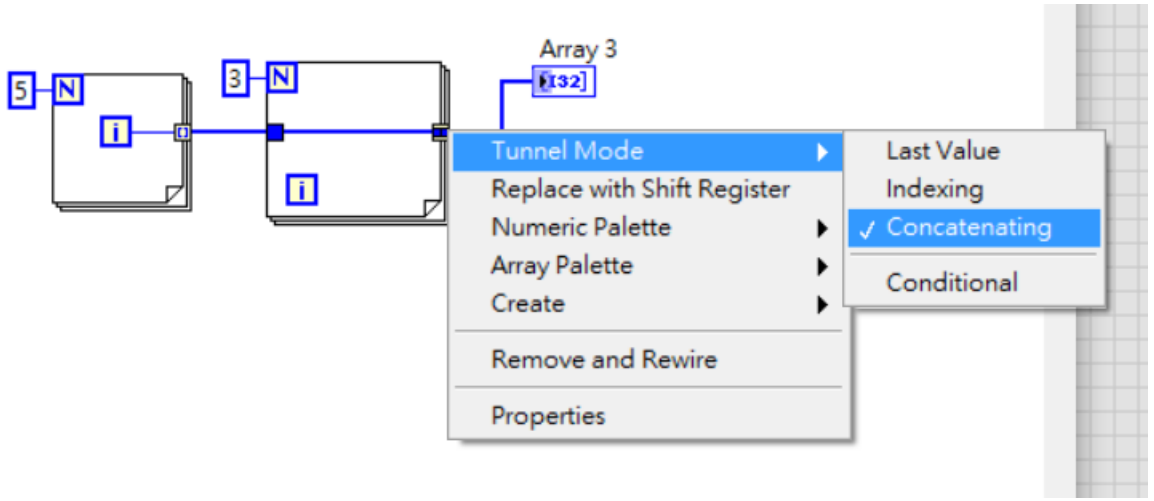


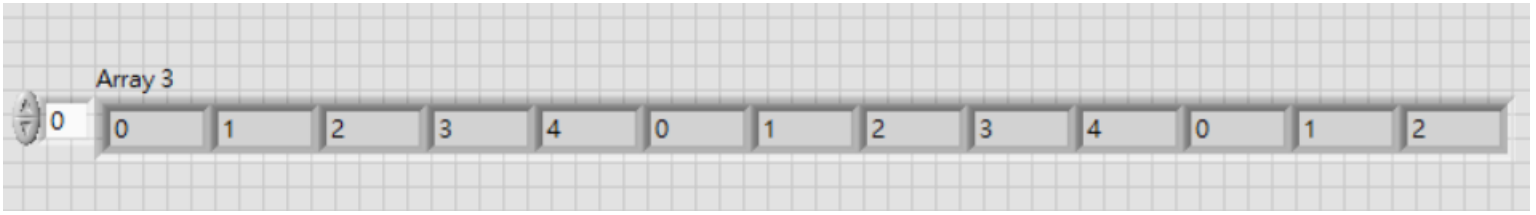


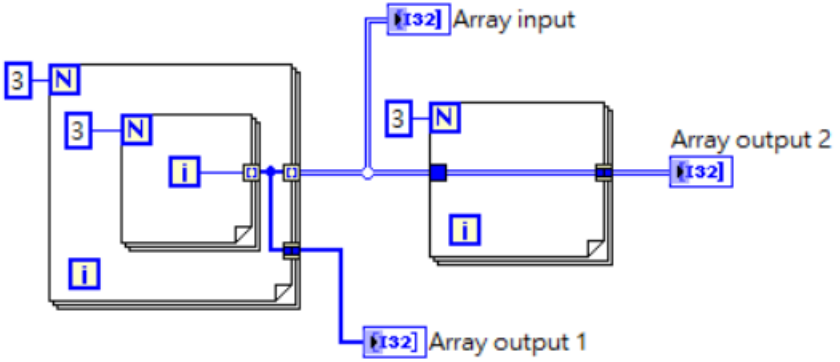


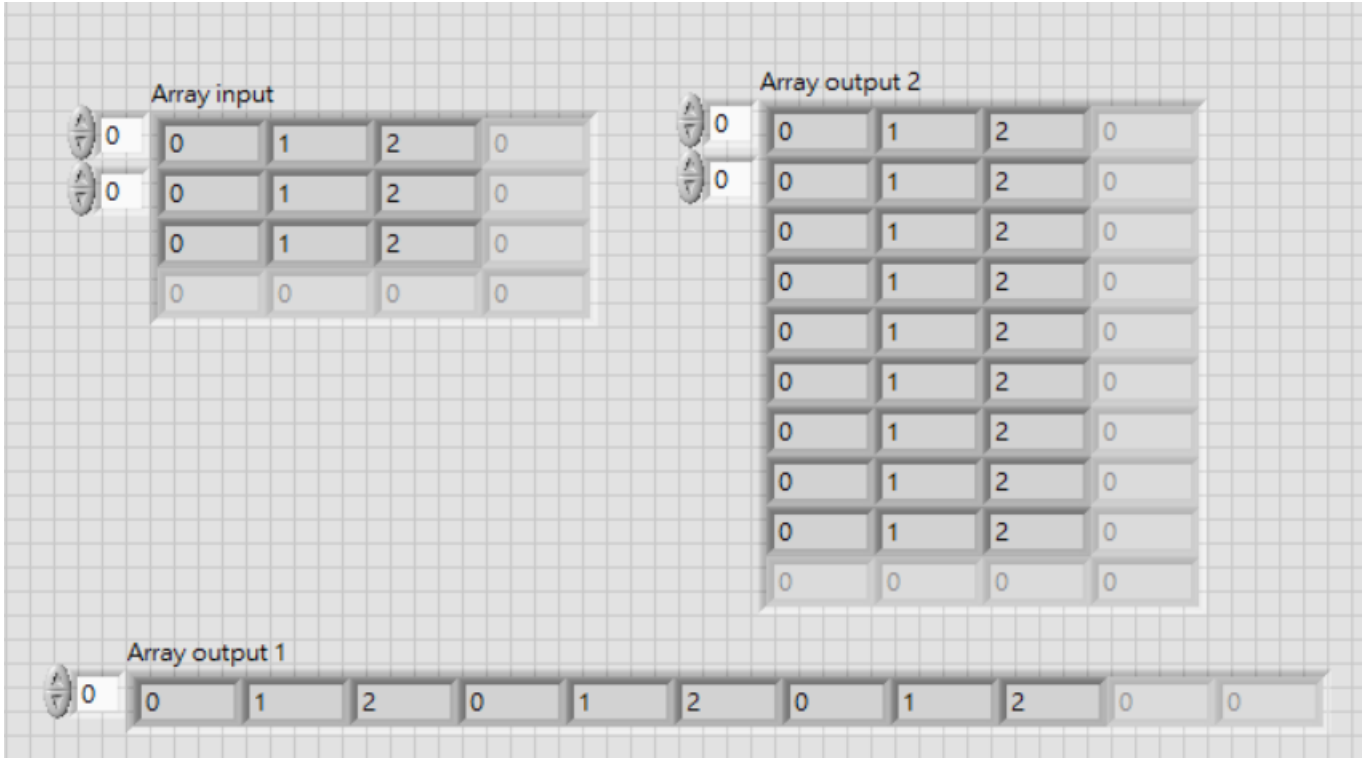


迴圈的 concatenating

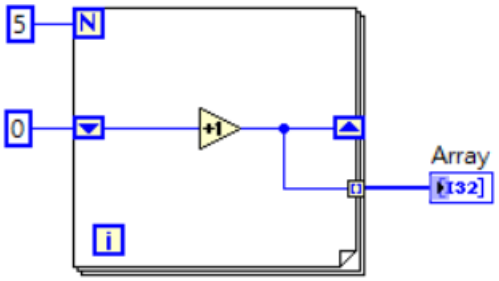


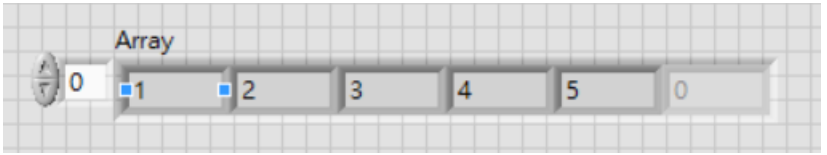


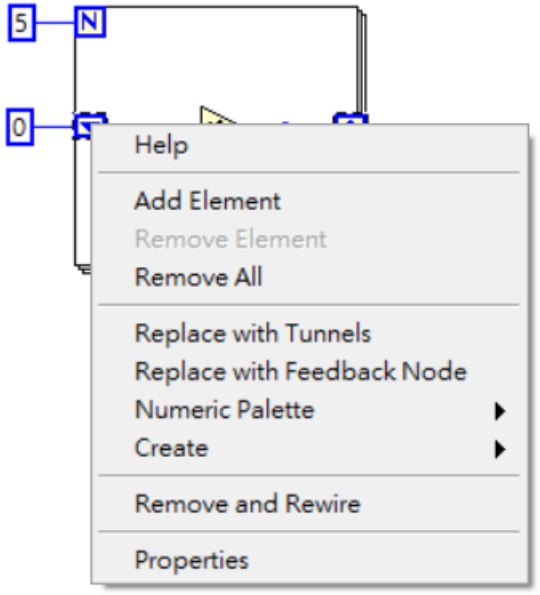

[32] Array input
3
N
3
N
i
i
3
N
i
Array output 2
[32]
[32] Array output 1


Array input
0
0
0 1 2 0
0 1 2 0
0 1 2 0
0 0 0 0
Array output 2
0
0
0 1 2 0
0 1 2 0
0 1 2 0
0 1 2 0
0 1 2 0
0 1 2 0
0 1 2 0
0 1 2 0
0 1 2 0
0 1 2 0
0 0 0 0
Array output 1
0
0 1 2 0 1 2 0 1 2 0 0

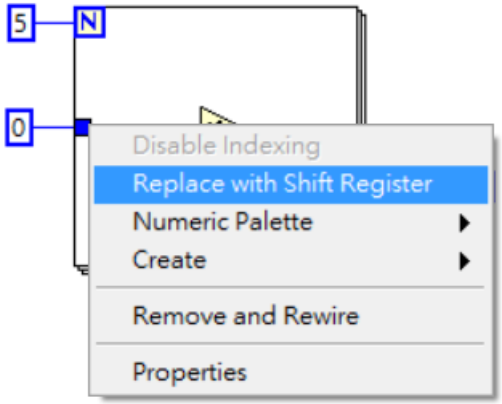
迴圈的 shift register



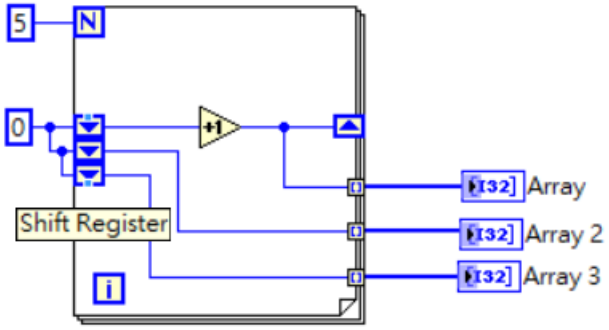


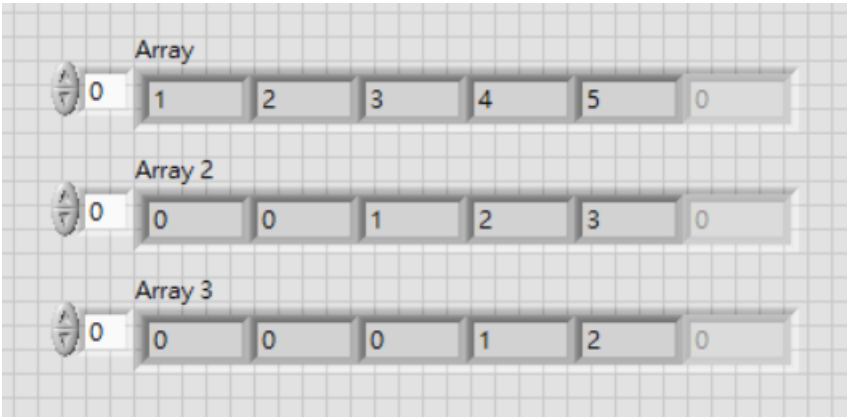


在節點上點擊右鍵可以修改節點屬性



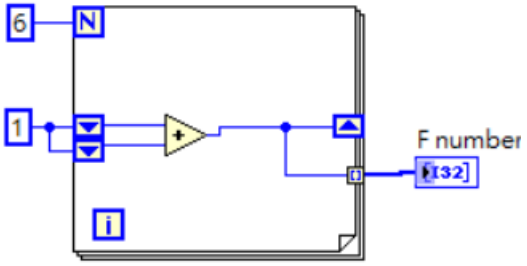
迴圈的 shift register

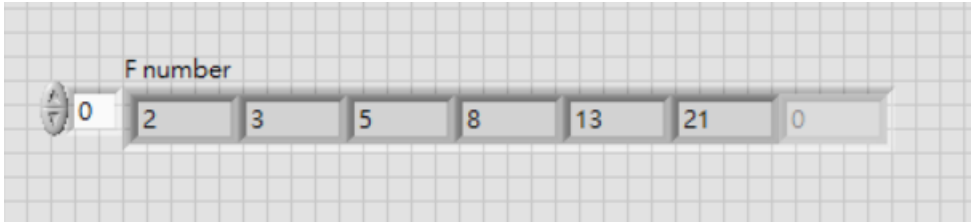




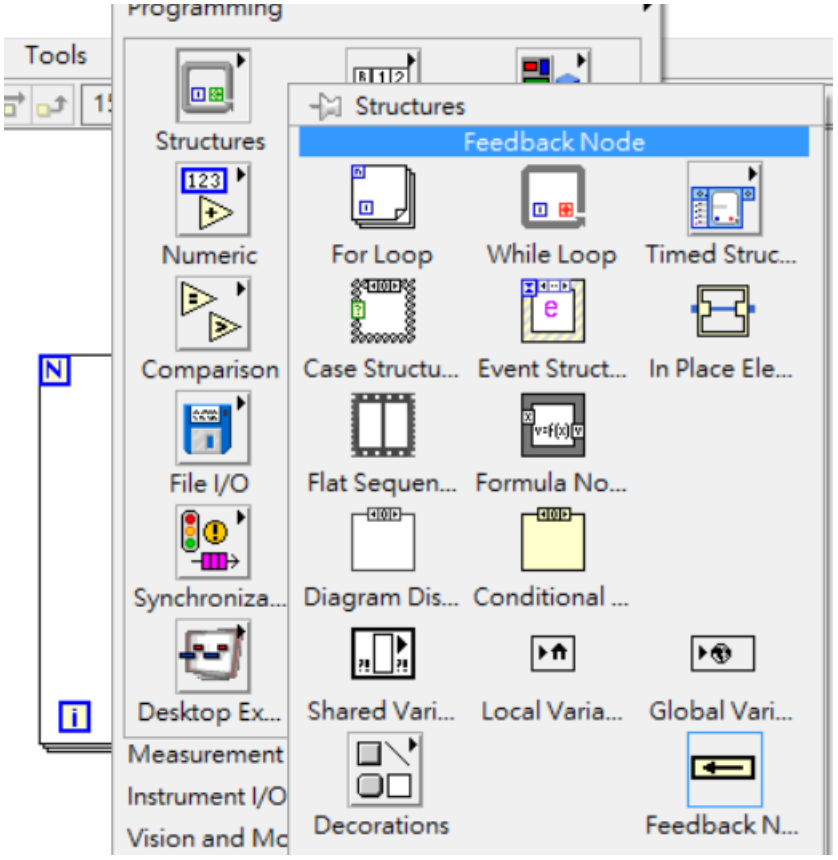
Shift register 可以拉更高階（兩次以前的值，三次以前的值）
Shift register 一定要給初始值

應用：菲波納契數列

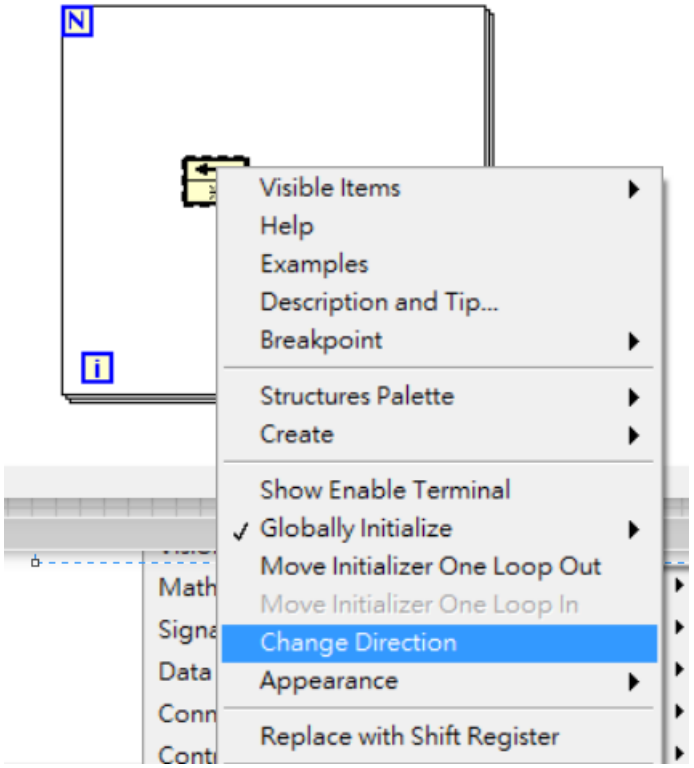




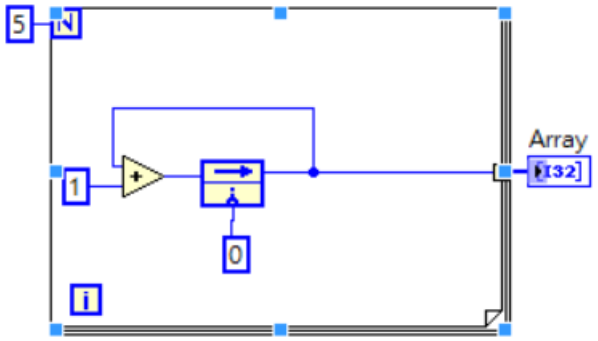
迴圈的 feedback node

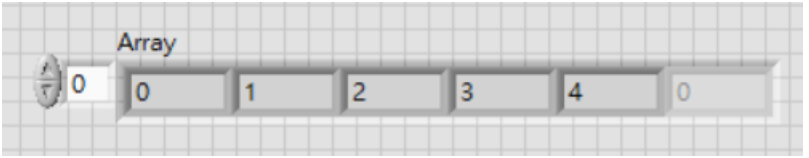


物件單擊右鍵選單可改方向



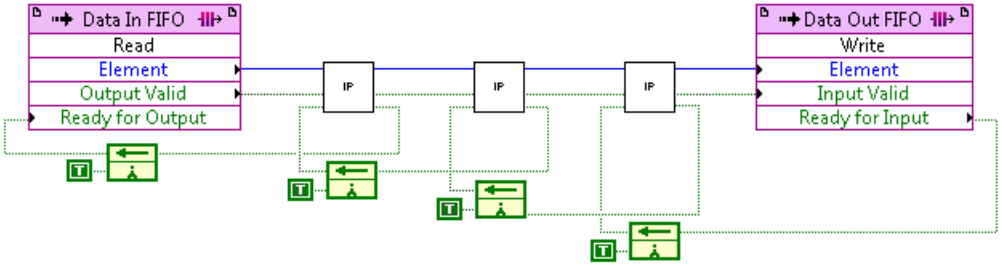
迴圈的 feedback node



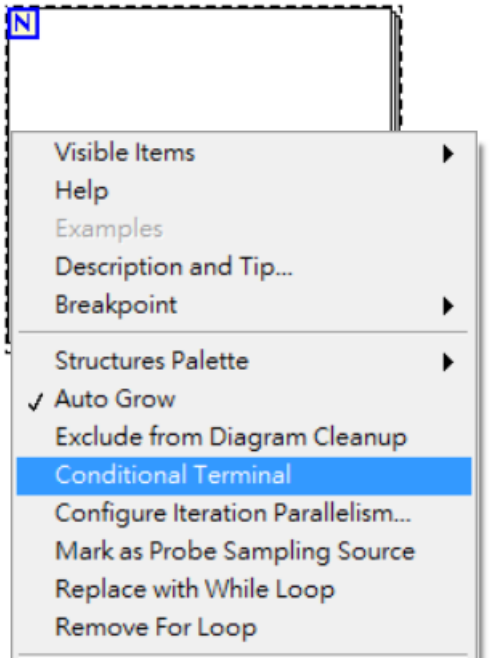


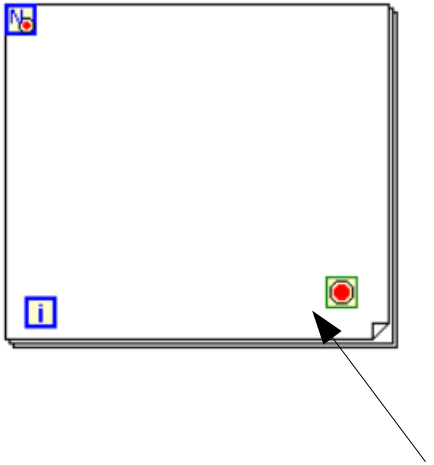
效果與 shift register 相同

使用時機： 拉線上需要空間 (ex. FPGA hand- shaking 寫法)



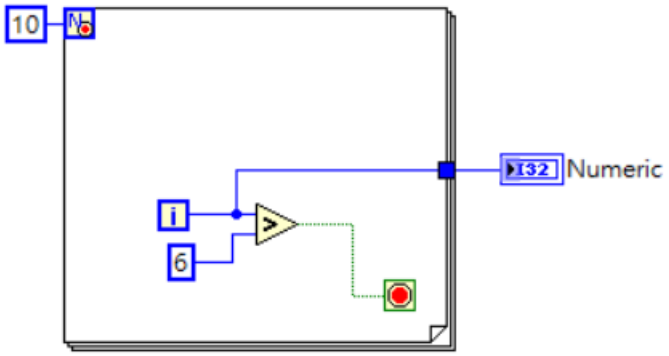
迴圈的 conditional stop (類似 while)

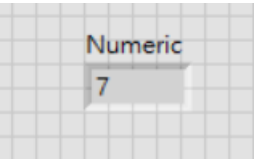




多一個 boolean indicator,
給它 True 執行完當前迴圈就會跳出

迴圈的 conditional stop (類似 while)

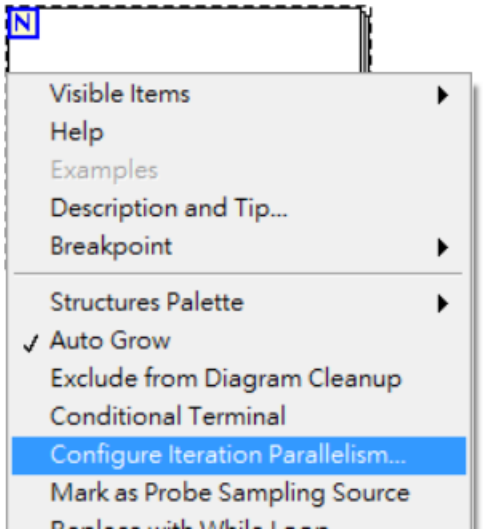


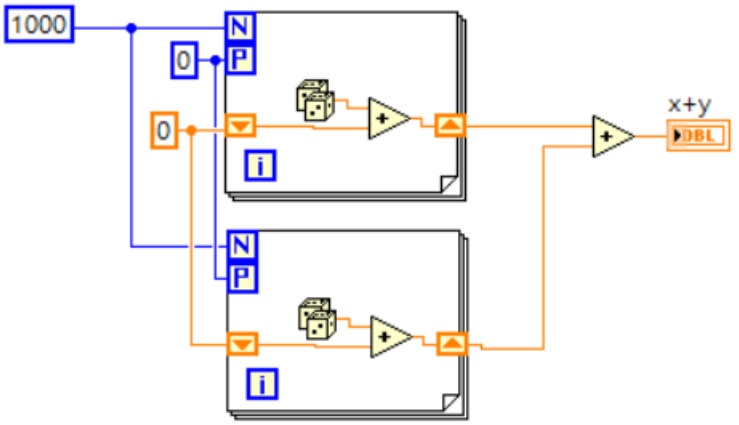


迴圈執行到第八圈 , index 等於 7 時 , 不再進行下一個迴圈
最終值存入 last value 節點 , 所以程式執行結果為 7
須注意的是 for 迴圈一定要指定執行圈數

應用 : 一個以 1D array 的 pulse 波型找半高寬

迴圈的平行運算 (after LV2009)

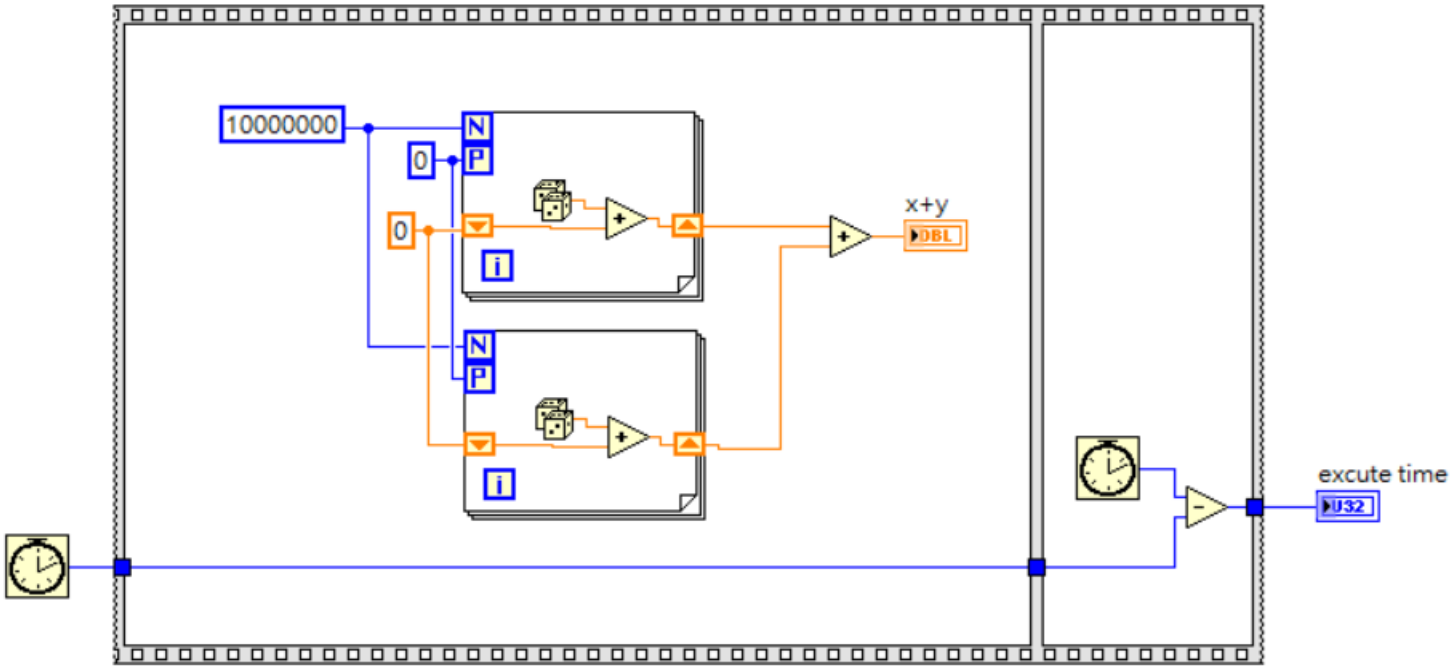


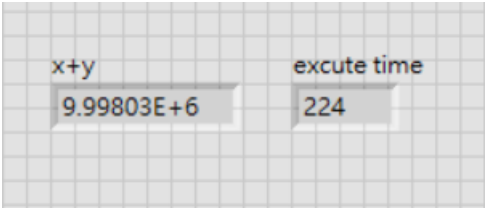


多了一個 P node 可以指定運算 CPU ID
平行迴圈要注意 racing condition

LV FPGA 中的迴圈自動就是平行運算迴圈

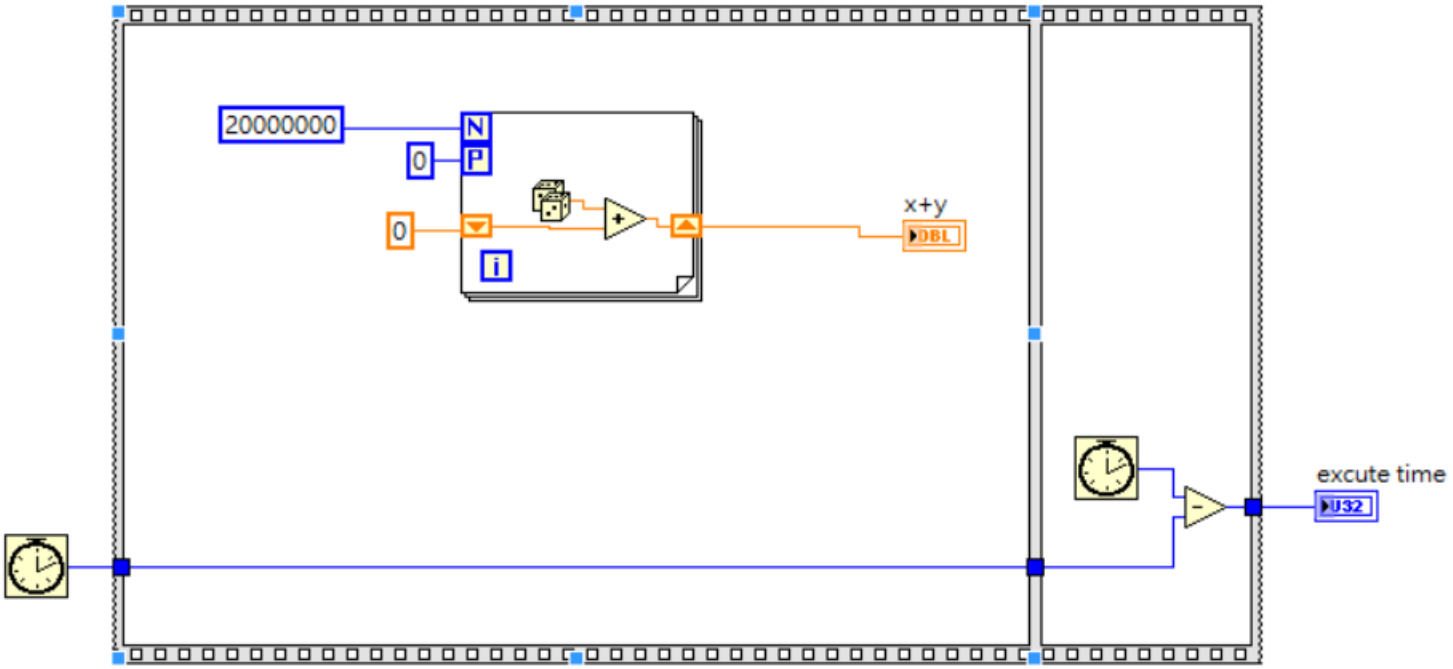
程式執行計時（使用 flat sequence)

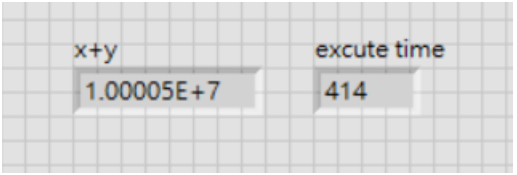




上一張範例計算 10000000 x 2 個亂數相加
約耗時 220 ms

程式執行計時（使用 flat sequence)





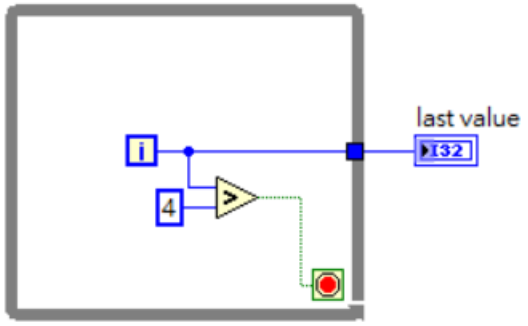
單一迴圈執行 20000000 個亂數相加
約耗時 414 ms

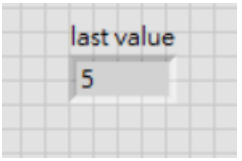
嚴格說起來測試時要關閉 LV 後重開再執行比較準
單一迴圈 enable P terminal 自動分配就可以看出效果

迴圈太複雜怎麼辦？

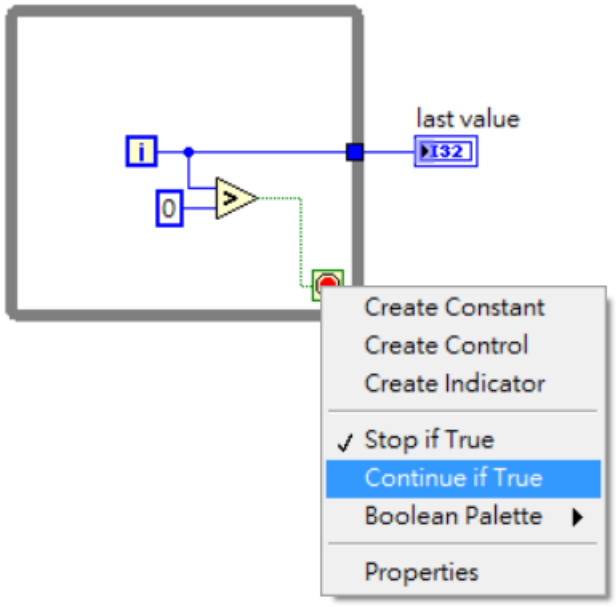
儘量不要使用超過兩層的巢狀迴圈
中間過程先拉出 indicator 觀察
使用 highlight execution & probe 觀察

while 迴圈

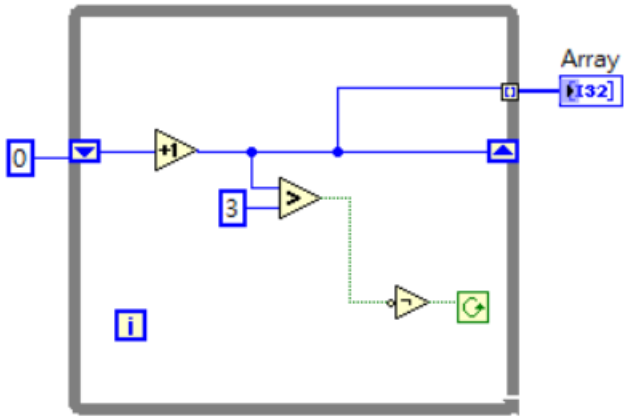


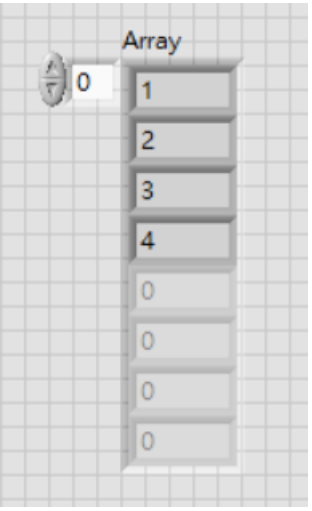


基本型相當於有 terminal stop 的 For loop，至少執行一次



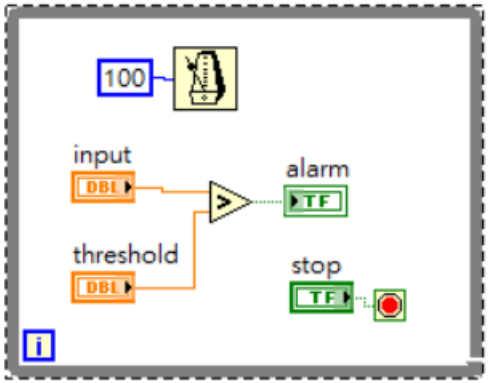
右鍵點擊 terminal indicator
可以將 while 迴圈改為 True 才執行

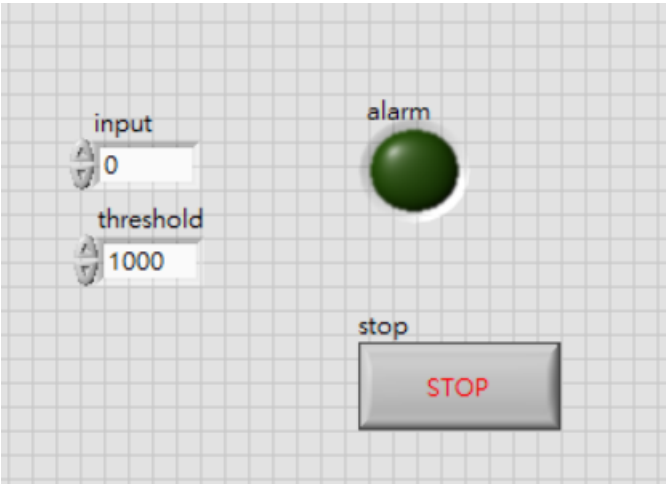




有些 DAQ 或 motion Control 的 subVI 輸出是 check valid, 所以會需要 continuous if true

簡單的迴圈控制



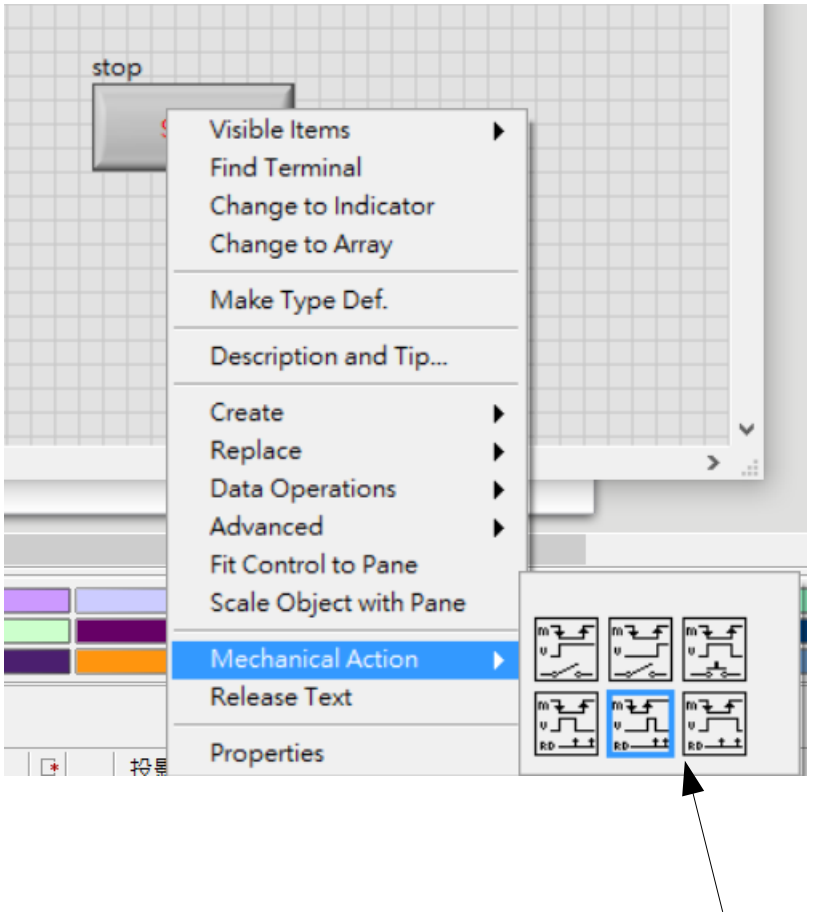


每 100ms 偵測一次 input 值與 threshold 比較
若 input 值大於 threshold 則 alarm 亮起

點擊 STOP 跳出程式

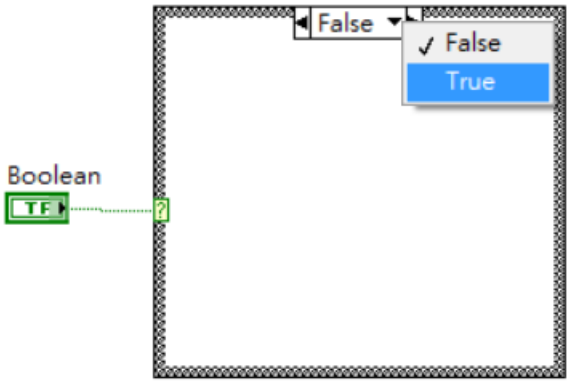
為何要設定 100ms delay? → 不嚴格同步 , HMI 互動通常設定 delay
避免迴圈大量執行占用記憶體資源

Button 的性質

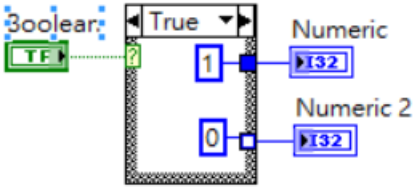


跟有沒有 Latch, 點擊與觸發 edge 有關
總共六種，自己嘗試即可知道差異

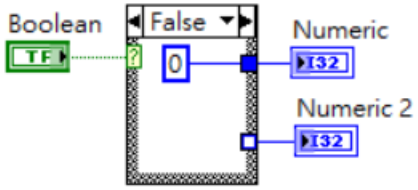
Case 的用法

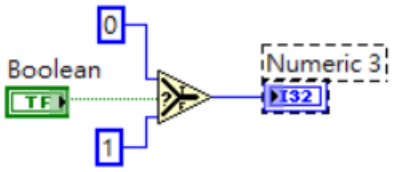


最簡單的 case, 輸入 boolean
→ 對應到 true 與 false 兩種情況

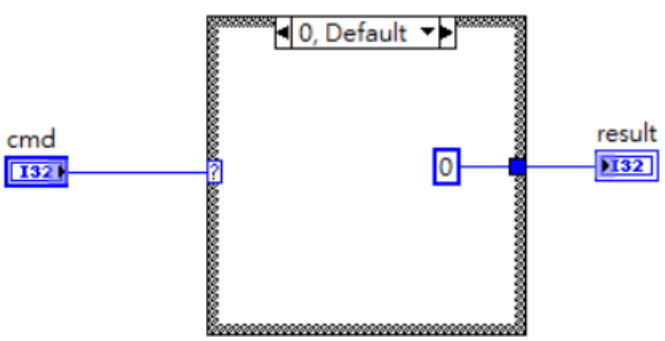


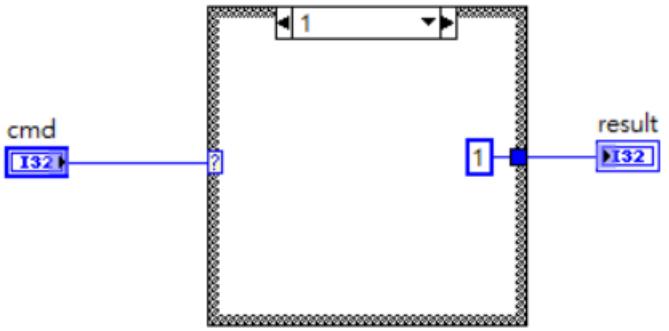
輸出接口若 case 未完成狀態會發現該接口會是空白的, 遇到多重 case 的時候可以用右鍵選單 Use default if unwired 避免斷線
簡單的狀況可以改用 Select 指令 (下圖)

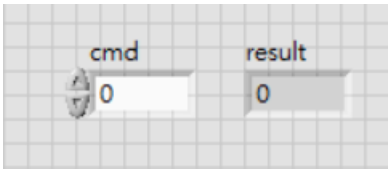


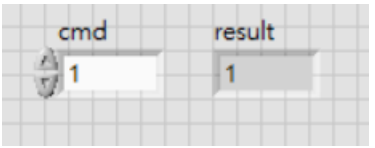


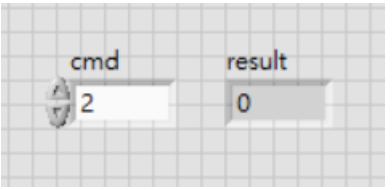
Case 的用法

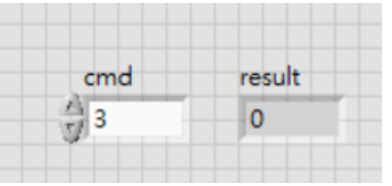




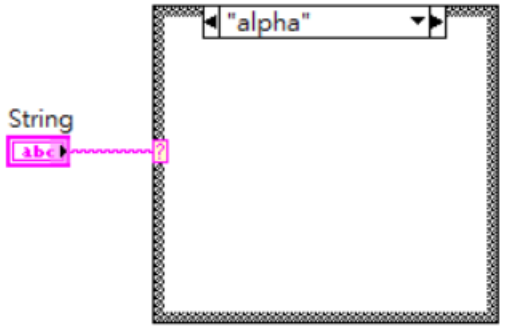




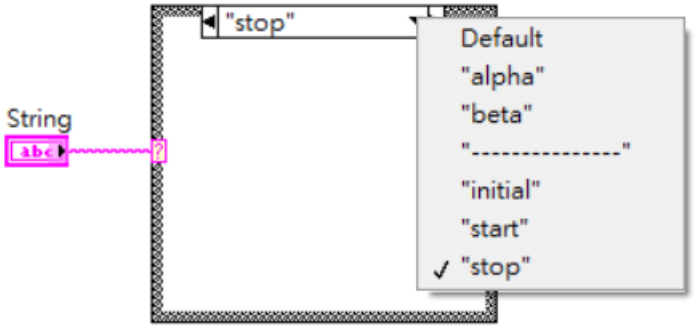




使用 integer 作為 case 的輸入
假如輸入的數值沒有對應的 case → 則自動取 default 的狀況

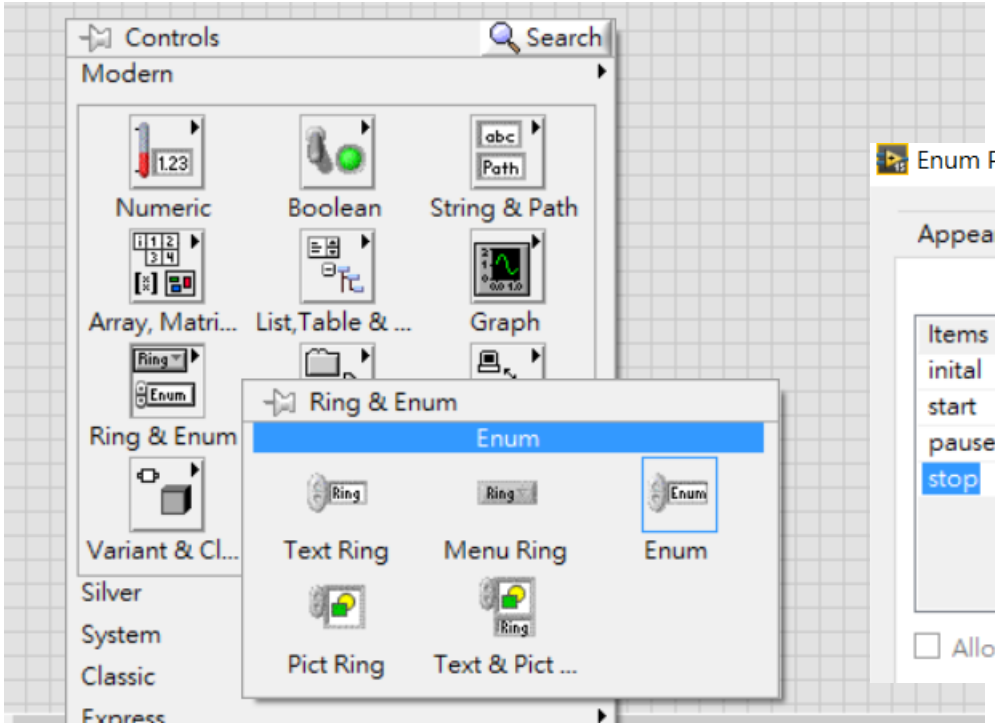


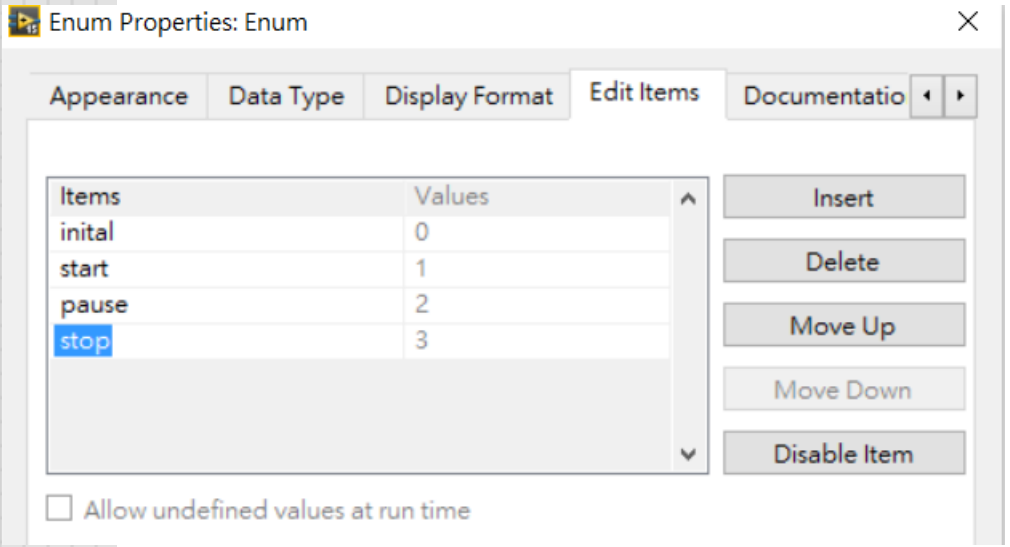
還可以用 string 輸入
在 queue 架構的 Producer-Consumer 架構下會使用到



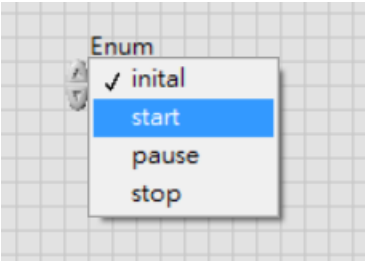
用 string 的好處是可以加入標示用的空 case

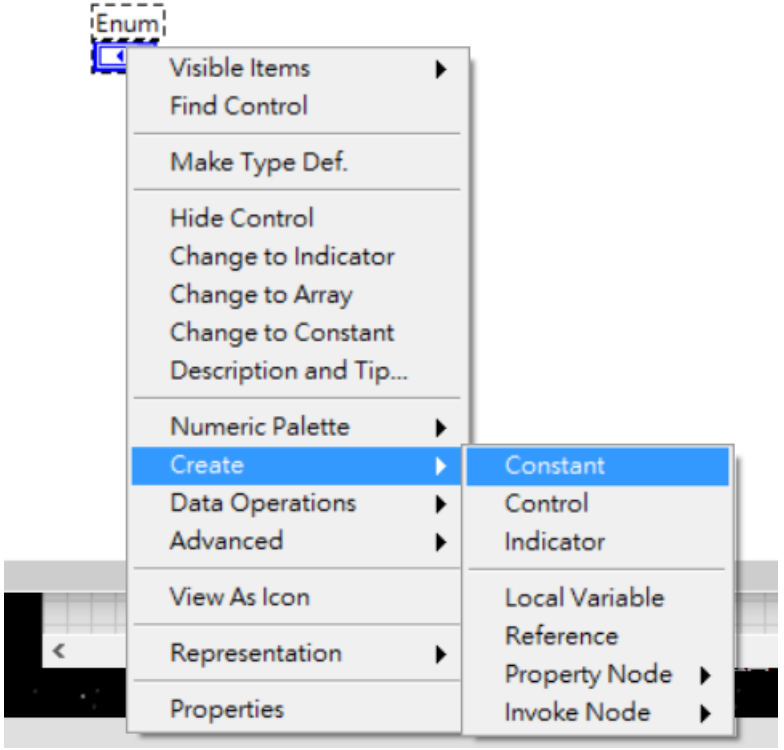
Enum 的用法





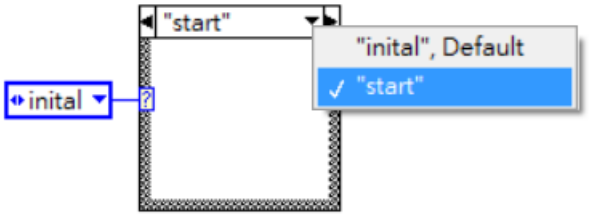
右鍵 → property → edit item
相當於製造出一個選單，每個 item 對應到一個 Integer，自然可以拿來做為 case 的輸入

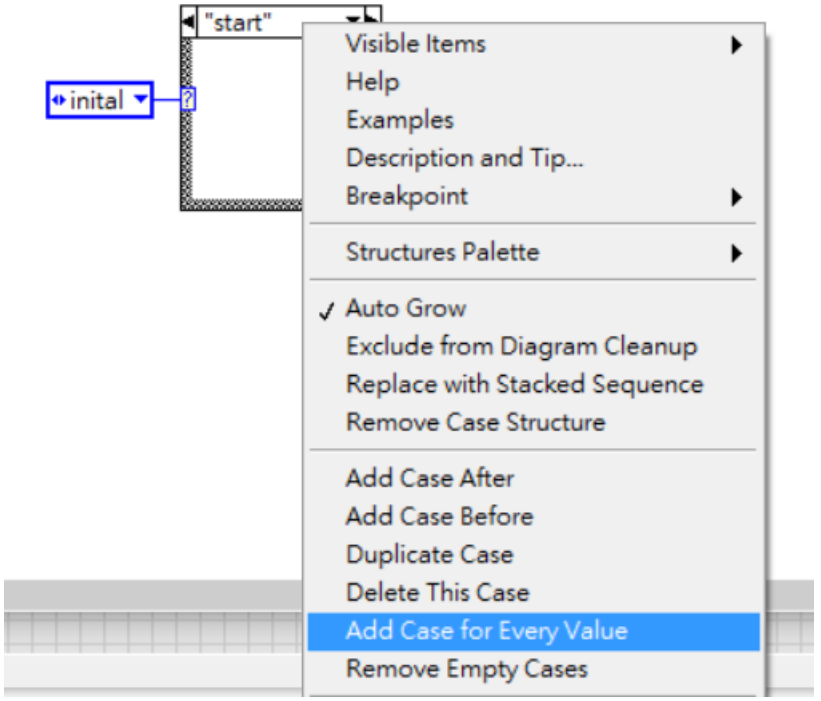




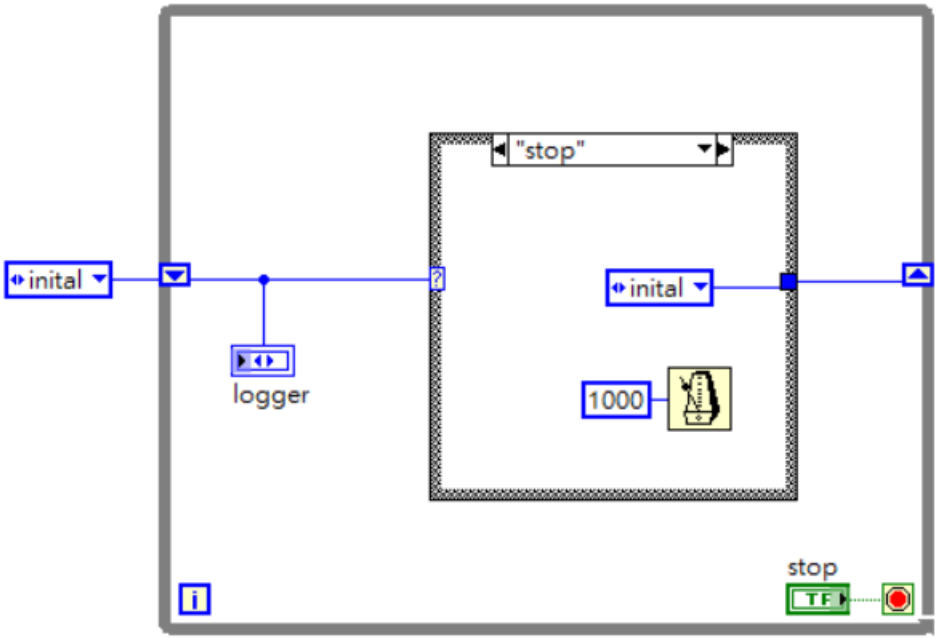
Block diagram 物件右鍵選單
可以製造出相同內容的 enum constant
(state machine 時會用到)

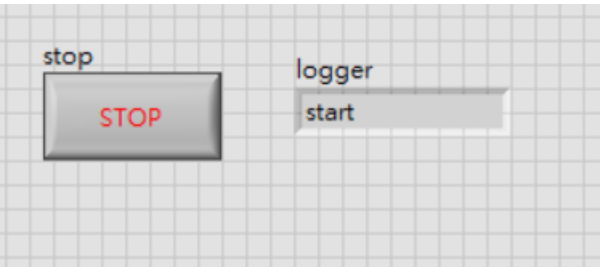
接到 case 上 ,case 右鍵選單 add case forevery value 可以自動生成對應的所有 case

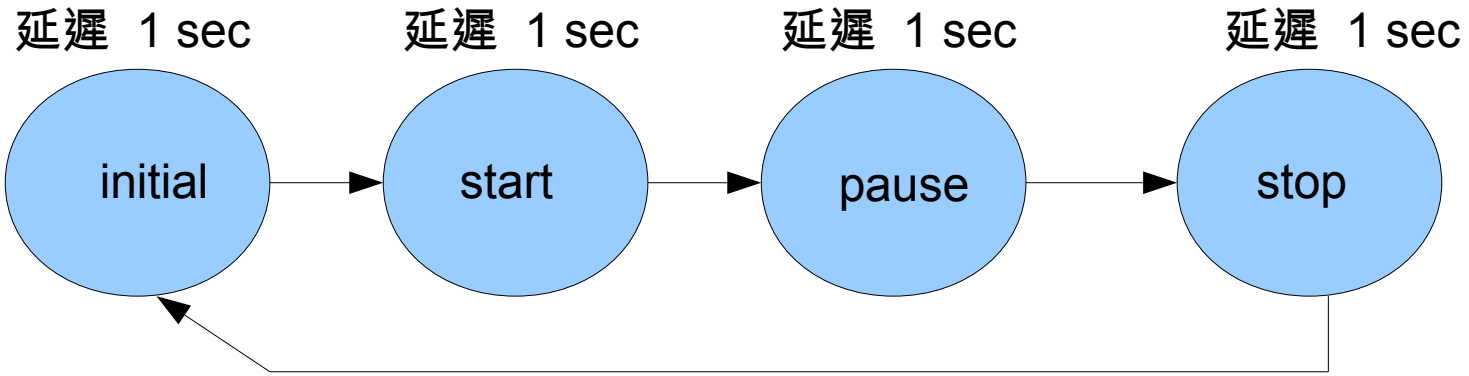




Enum + Case + 迴圈的應用 : state machine







畫的出流程圖與判定條件的通常都可以轉成 state machine
http://www.ni.com/tutorial/7595/en/

上一頁的範例有何缺點？

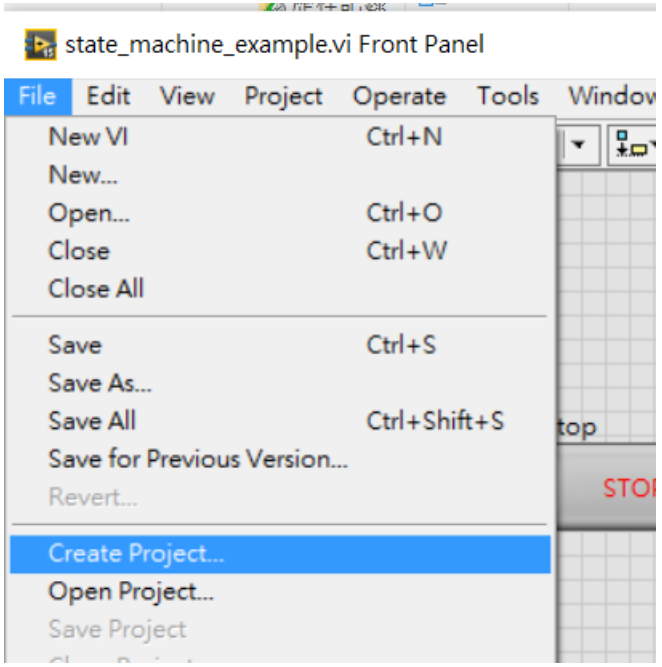
使用者按下 stop 時會直接中斷 → 應該有一個 stop 的 state, 完成中斷後
再送出 True 訊號到 while 迴圈的中斷控制

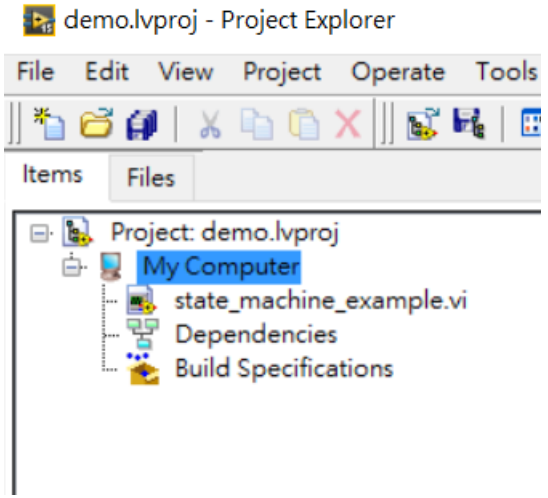
什麼時候判定按鈕是否有按下？
需要一個 idle 的 state

同時有多種程序需要觸發控制該怎麼辦？
別用 state Machine 了吧（加入 queue 的概念 , Producer-Consumer 架構）

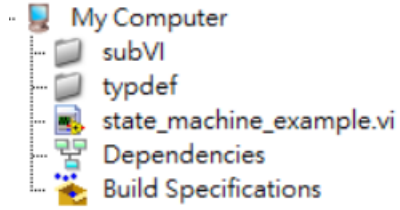
寫到一半發現要追加新 state, 難道要一個一個改 state 裡的 enum constant?
請愛用 type definition!!
開始用 project 來管理程式吧

使用 project

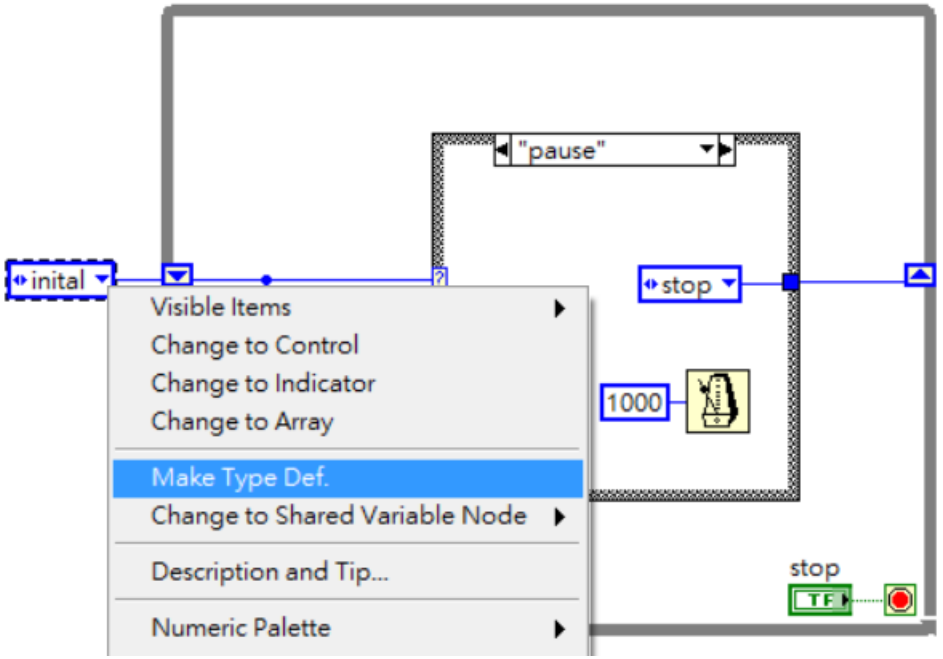




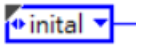
會多一些跟 labview project 相關的檔案，建議一個 project 用一個資料夾管理
Project explorer 的資料夾結構最好跟實體目錄一樣
夠專業還可以配合一些外掛做版本控制



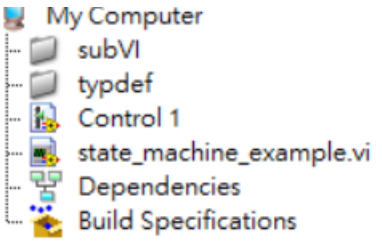
根據需求建一些 folder, subVI, API, typdef, gloable variable 之類的

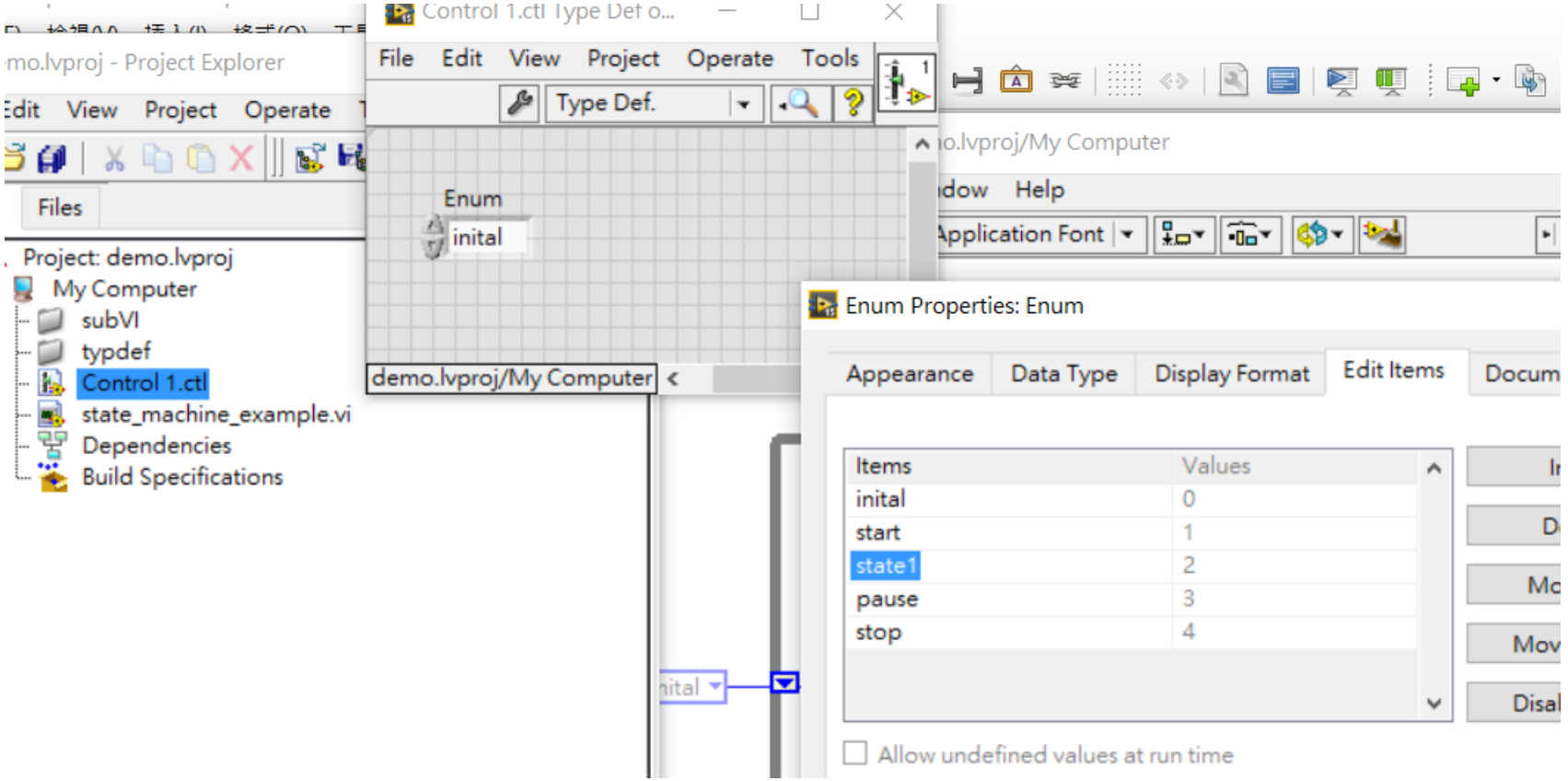


Make type def.



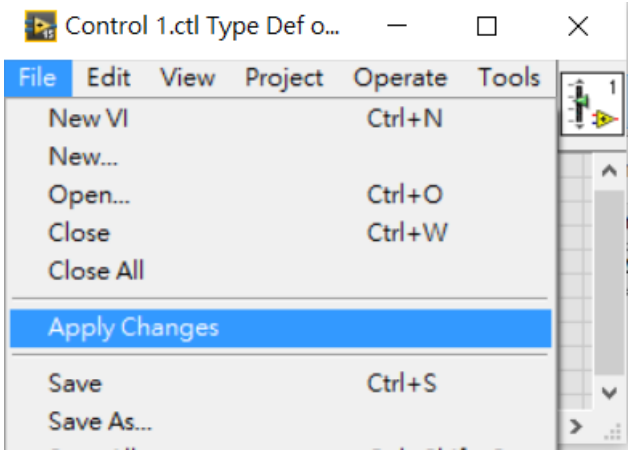
會多一個 control 的玩意
這就是 跟 type def





從 project 點開 control 編輯內容
File → apply changes

你會發現主程式裡用到的所有 type definition
Control 都會跟著改變



請記得的往後有需要變更的常數變數都先宣告成
type definition

如果先接好再 make type definition, 有可能會出現很難改的錯誤 (case 選項跑掉之類的)

另外 type definition 有兩種 , 一種是 strict type def.
自己 google 一下研究有何區別
在之前的例子一般用 strict type def.

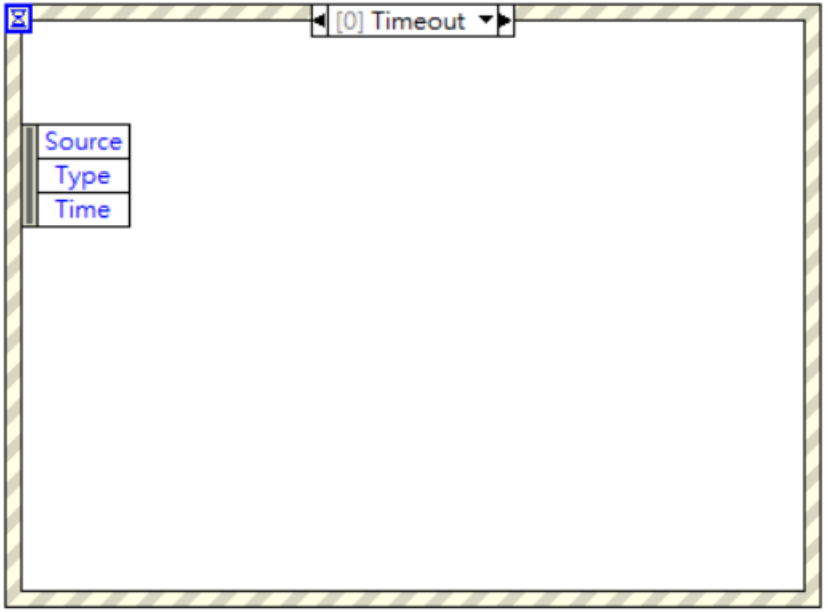
http://labview360.com/article/info.asp?TID=15804&FID=165

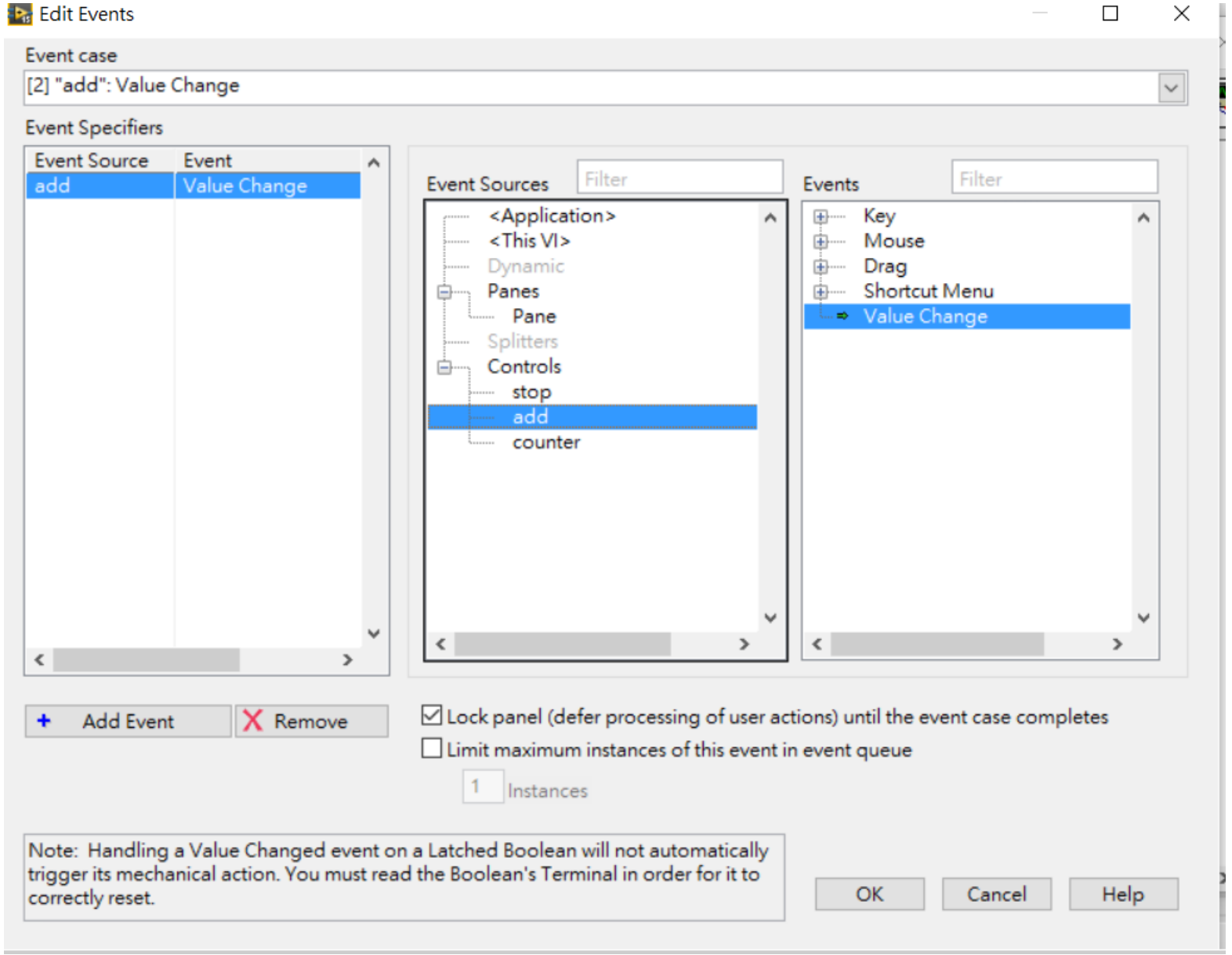
Event structure

如果需要複雜的 HMI 介面怎麼辦 (很多 button, 觸發各種不同事件) 怎麼辦？

連 event structure 都不會用 = 不會用 Labview
我的程式常常 miss 介面的按鈕觸發，設計的 stop 都關不掉
→ 你有用 event structure 去 pooling HMI 指令嗎？

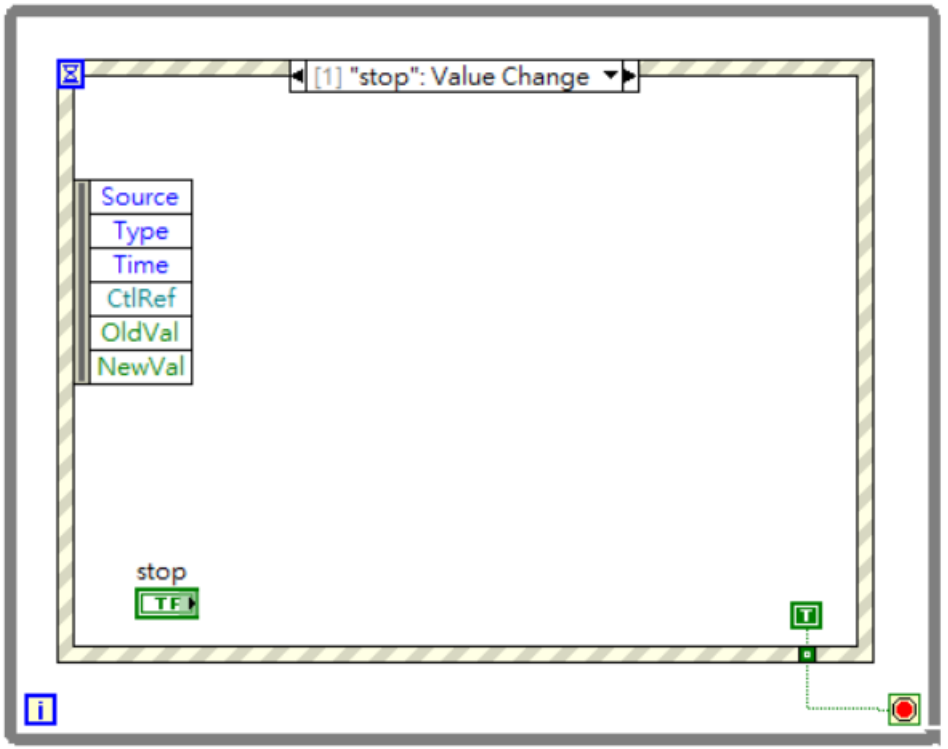
Event structure 相當於偵測面板 control 變化，有 queue 去存指令的 case





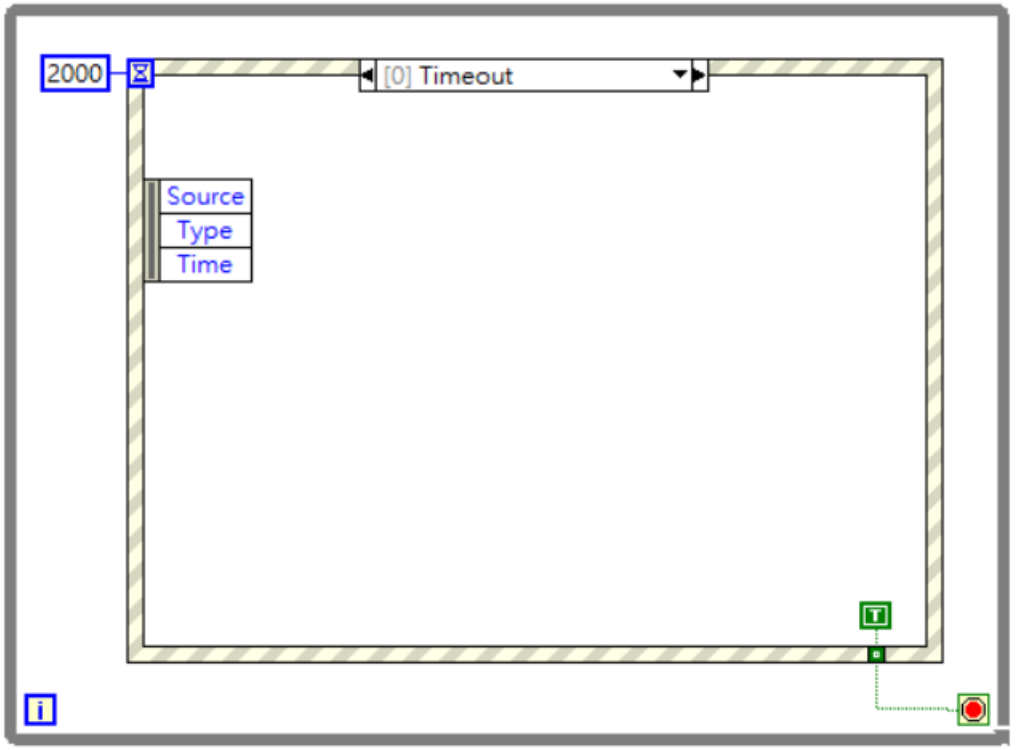
選取你要增加的 case，偵測哪個 control，哪種行為（被按下，滑鼠移進來等等）

Event structure



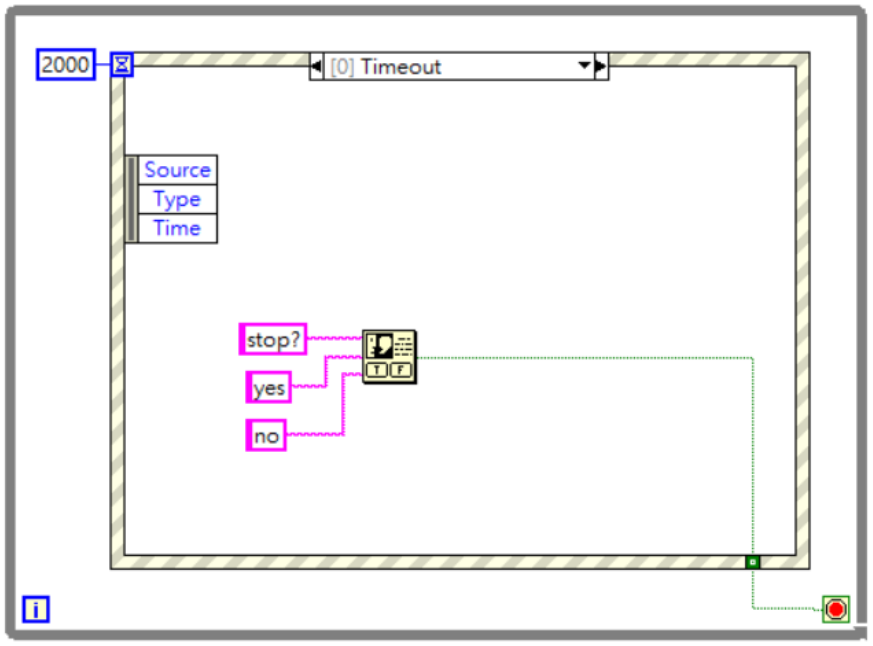
改完之後記得把偵測的 button 拉近對應的 case
最簡單的 event structure: 通常外面包 while
偵測一個 stop button, button 變化發送一個 True 終止迴圈
注意這裡的 node 都 use default when unwired (F), 這樣 event structure 其他 Case 都不會終止程式

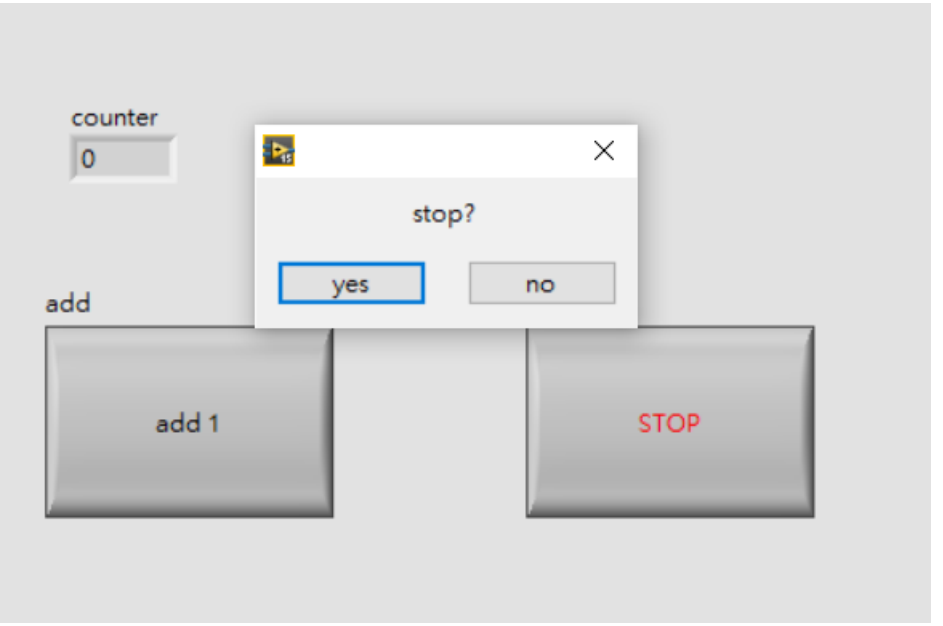
Event structure 的 timeout



漏斗的輸入預設為 -1 就是沒有 timeout
上圖表示 : 2000ms 面板無動作 , 送出一個 True 終止程式
Timeout 的原則也是不要設太短 (大於 50-100ms)

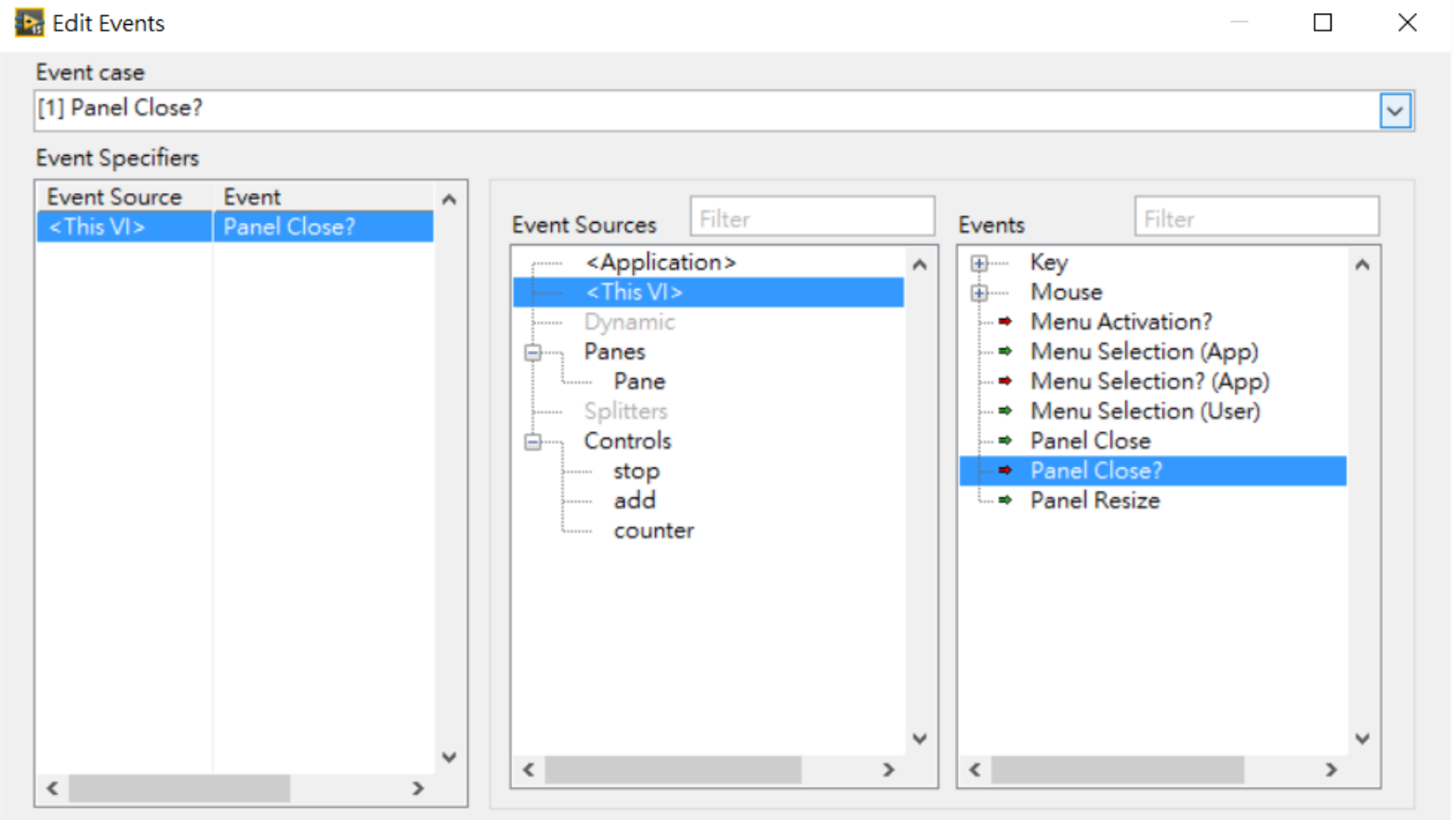
Event structure 的 timeout



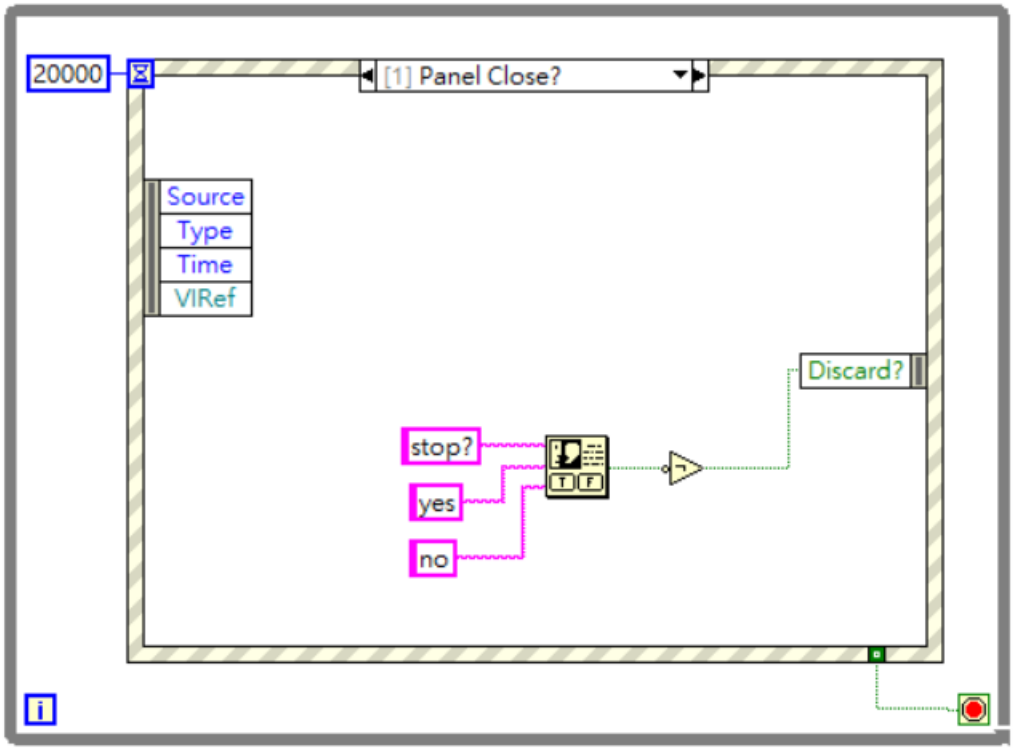


加點變化，增加互動感
每兩秒偵測一次使用者有動作
跳出提示視窗提示使用者關閉程式 (two button dialog)

更豐富的動作控制



當使用者按到程式關閉的叉叉要觸發的行為

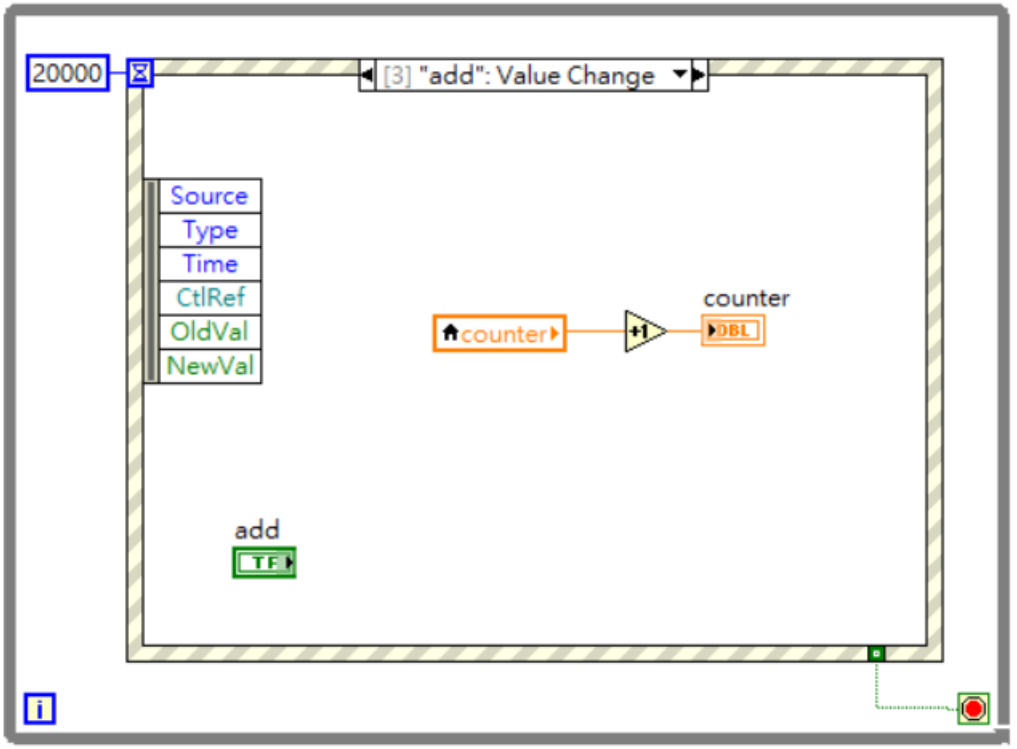


跳出選單要使用者選擇是否要關閉 panel

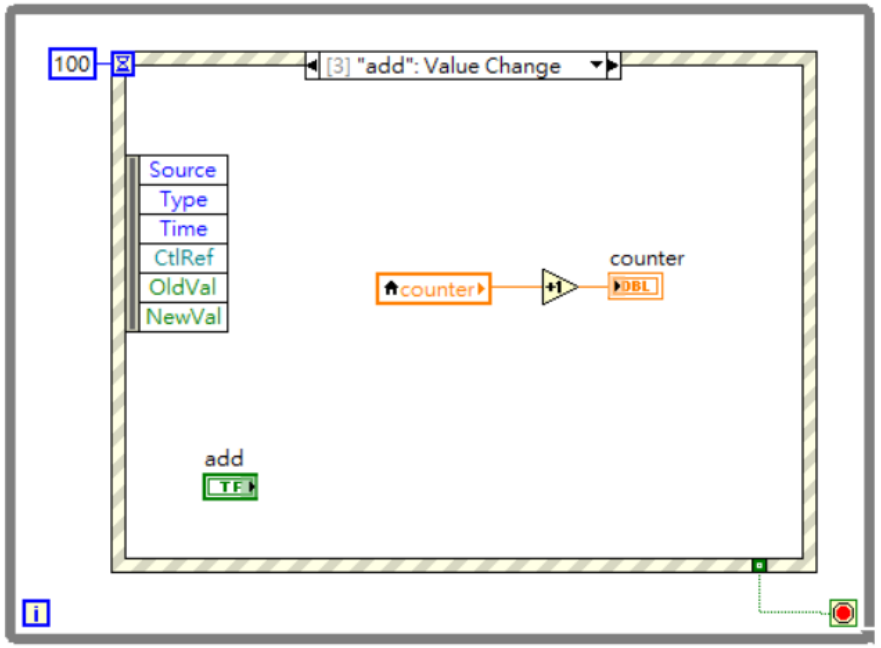
Yes → 送出 True → invert → False → discard

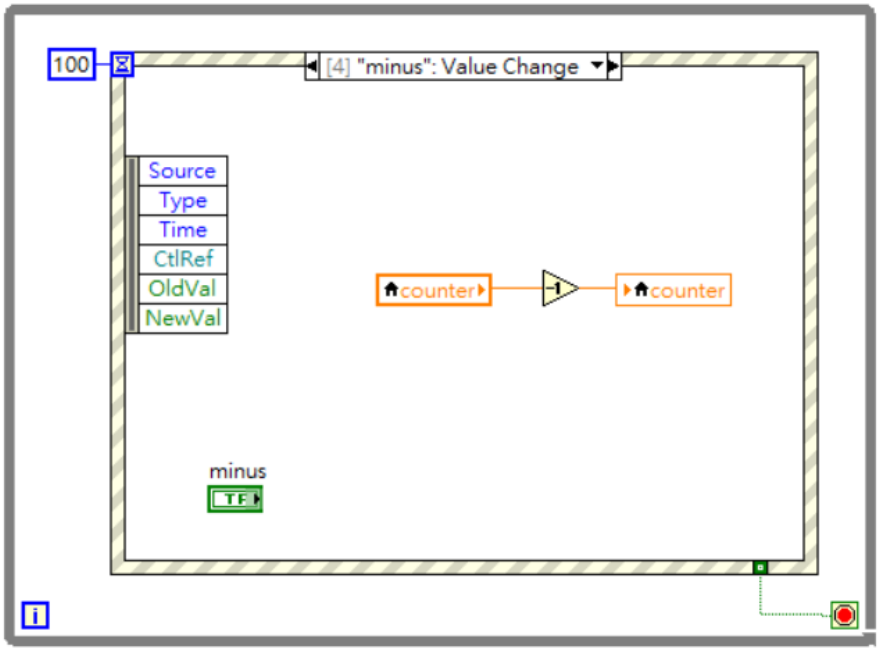
不要 discard, 繼續執行 panel close 的動作

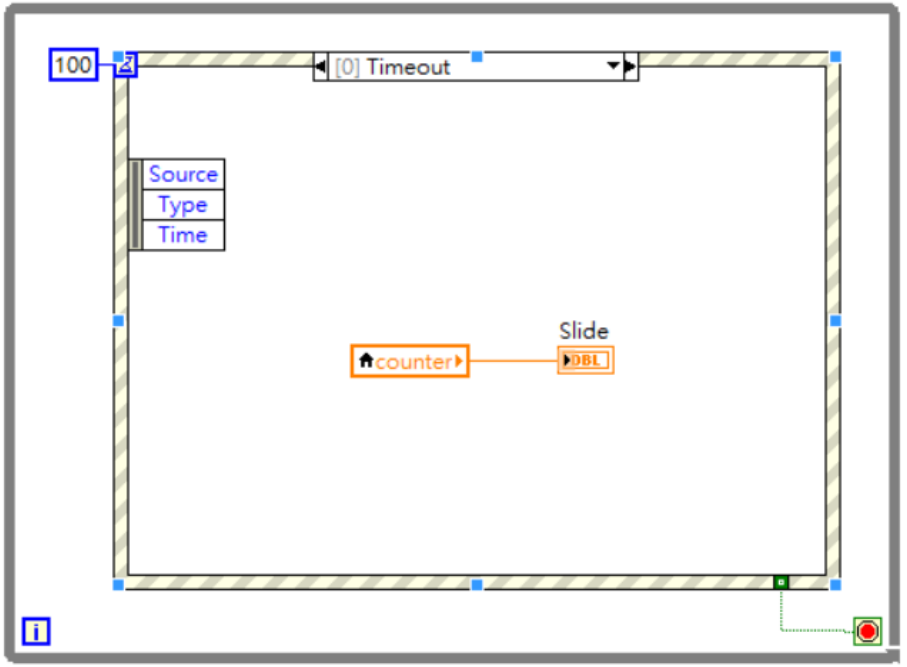
Event structure 的 timeout

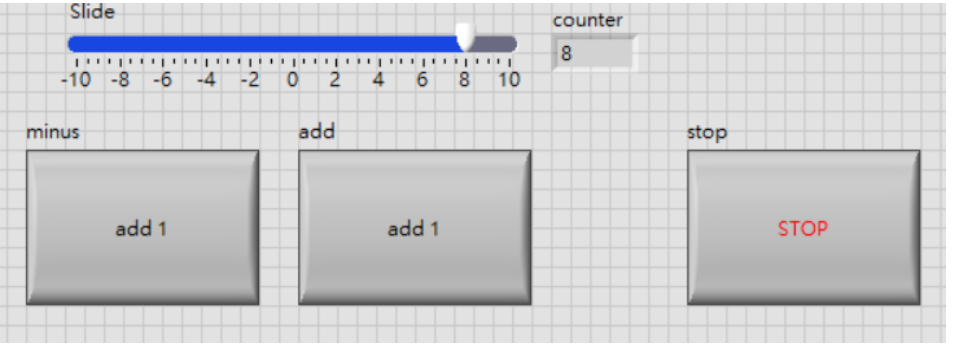


使用者每按一次 add 鍵 , counter 就 +1









利用 timeout 來更新讀值圖像化預覽

Event structure 的優點？

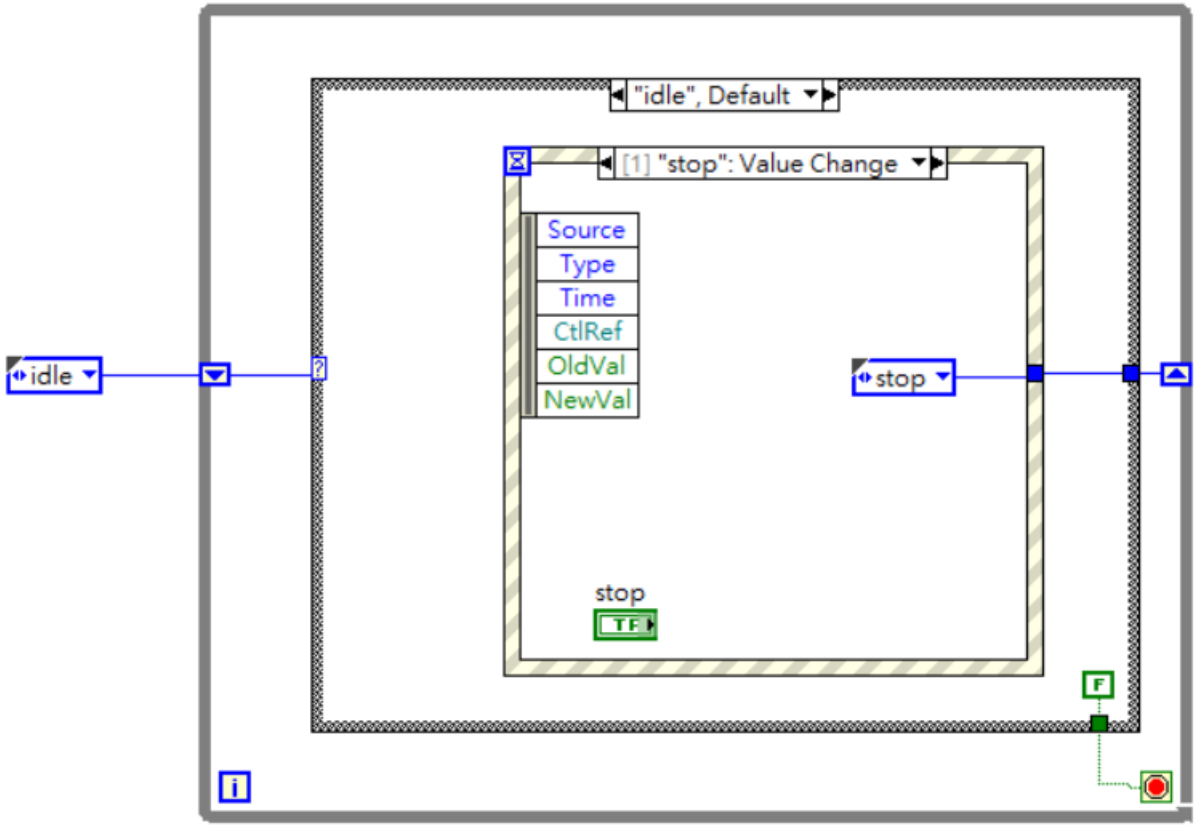
使用者觸發 front panel 的行為都會被記錄 (隱藏的 queue)
再非同步依序執行對應的動作

能不能用變數變化 (非人為改變其值) 觸發？
要用 user event

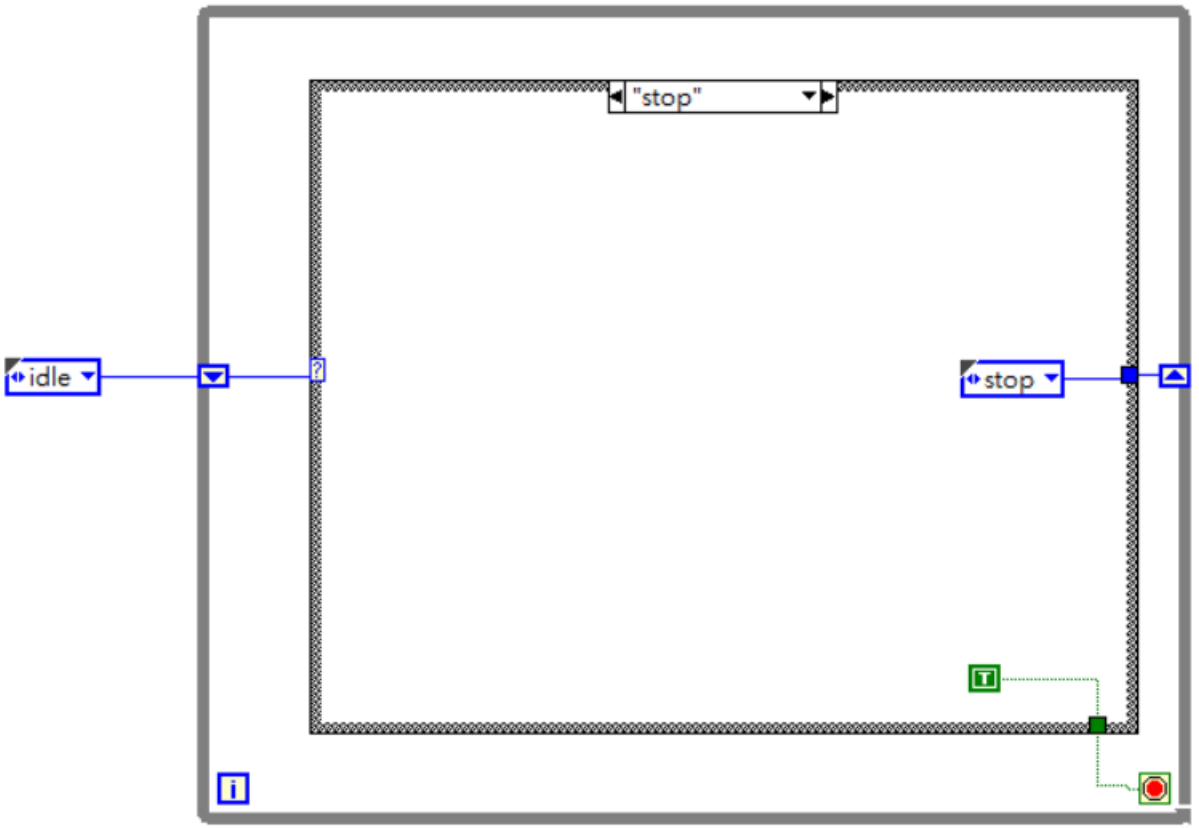
如果有一項動作計算會很耗時 , 可以將它塞入 event structure 嗎？
通常 event structure 只負責 HMI 抓動作 , 配合 queue 跟 state Machine 才能進行更複雜的控制

Ex: queue driven state machine
Ex: producer-consumer model

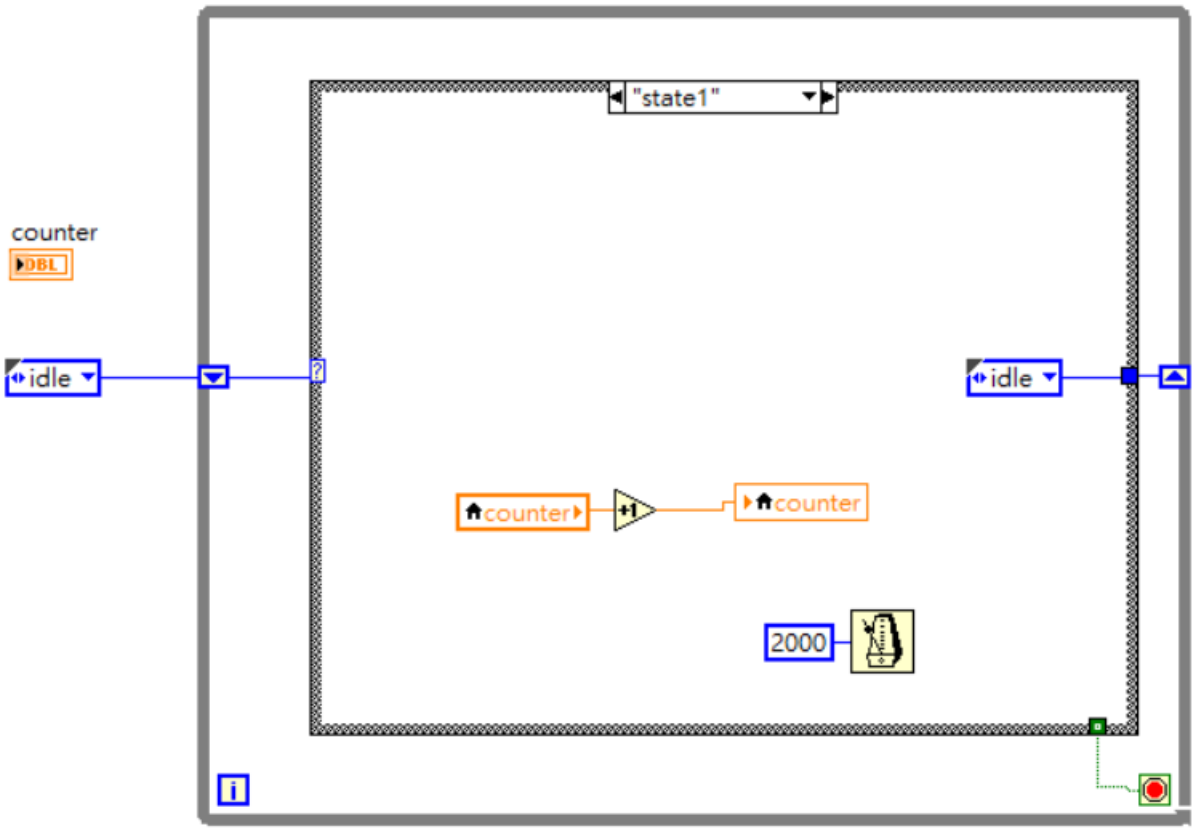
Event structure 進階版 - 有 idle 的 state machine



有一個 idle state, 任何流程最後都會回到這個 idle state
Event structure 裝在 idle state 裡，偵測到任何動作便送出對應的 enum 指令
讓系統跳到下一個 state



只有 stop 跟 ilde 兩個 state 的主架構



非 stop state 最後都會回到 idle

有 idle 的 state machine 的優缺點？

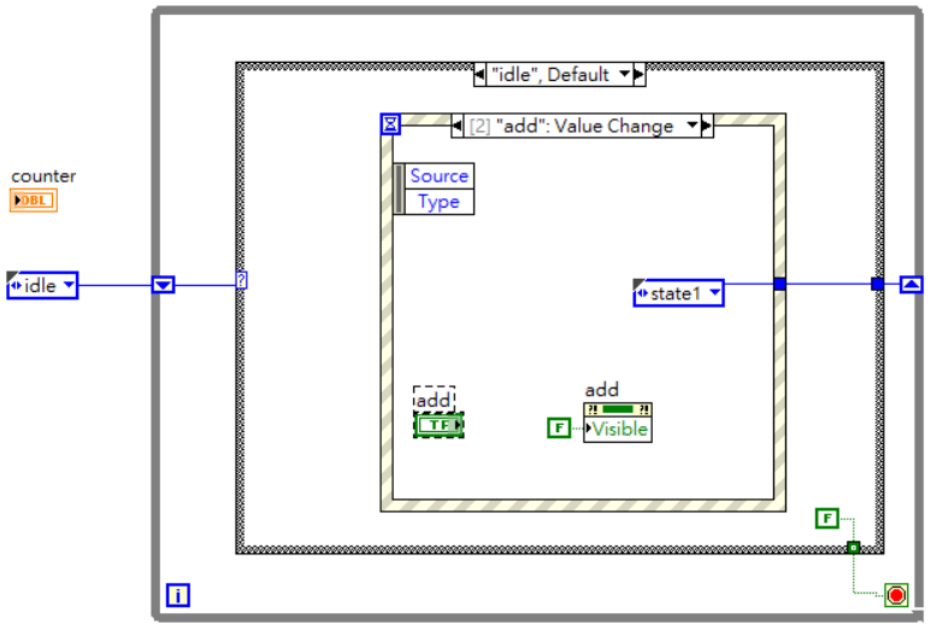
在執行動作時 (非 idle state) 時面板的輸入回不會被記錄？

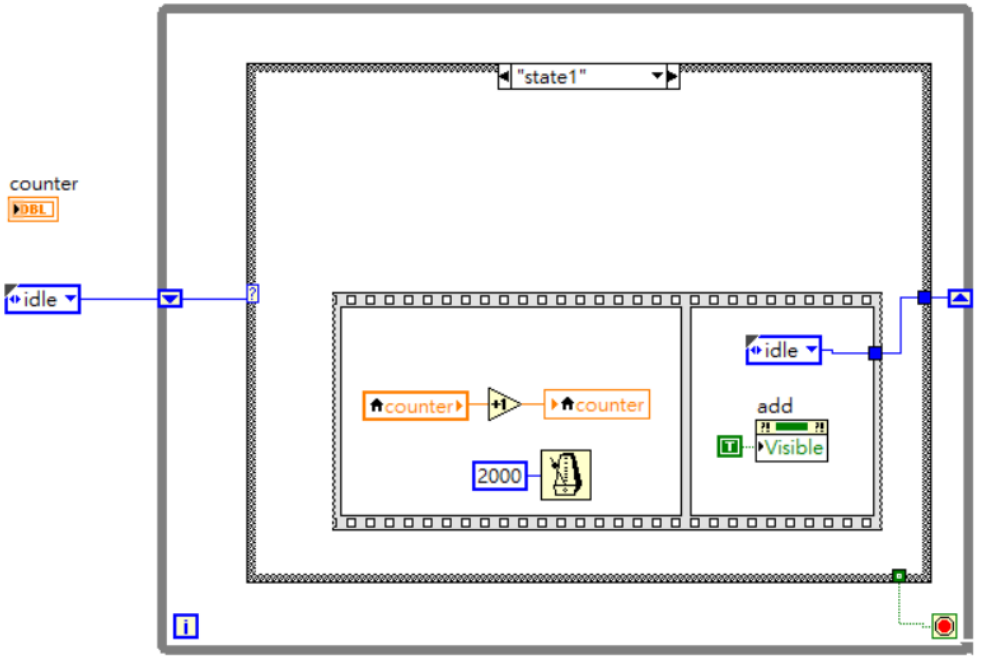
State 若執行很久面板操作流暢度？

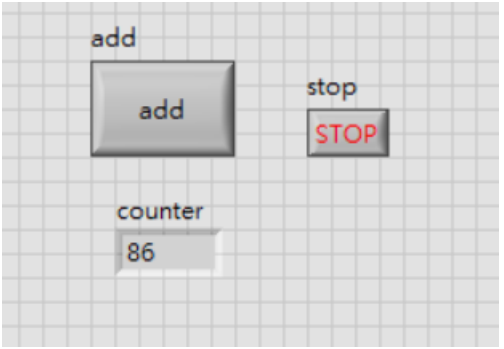
能不能在非 idle state 按照正常流程終止程式？

跨 state 資料傳遞？

特殊技巧 (讓畫面更順暢)

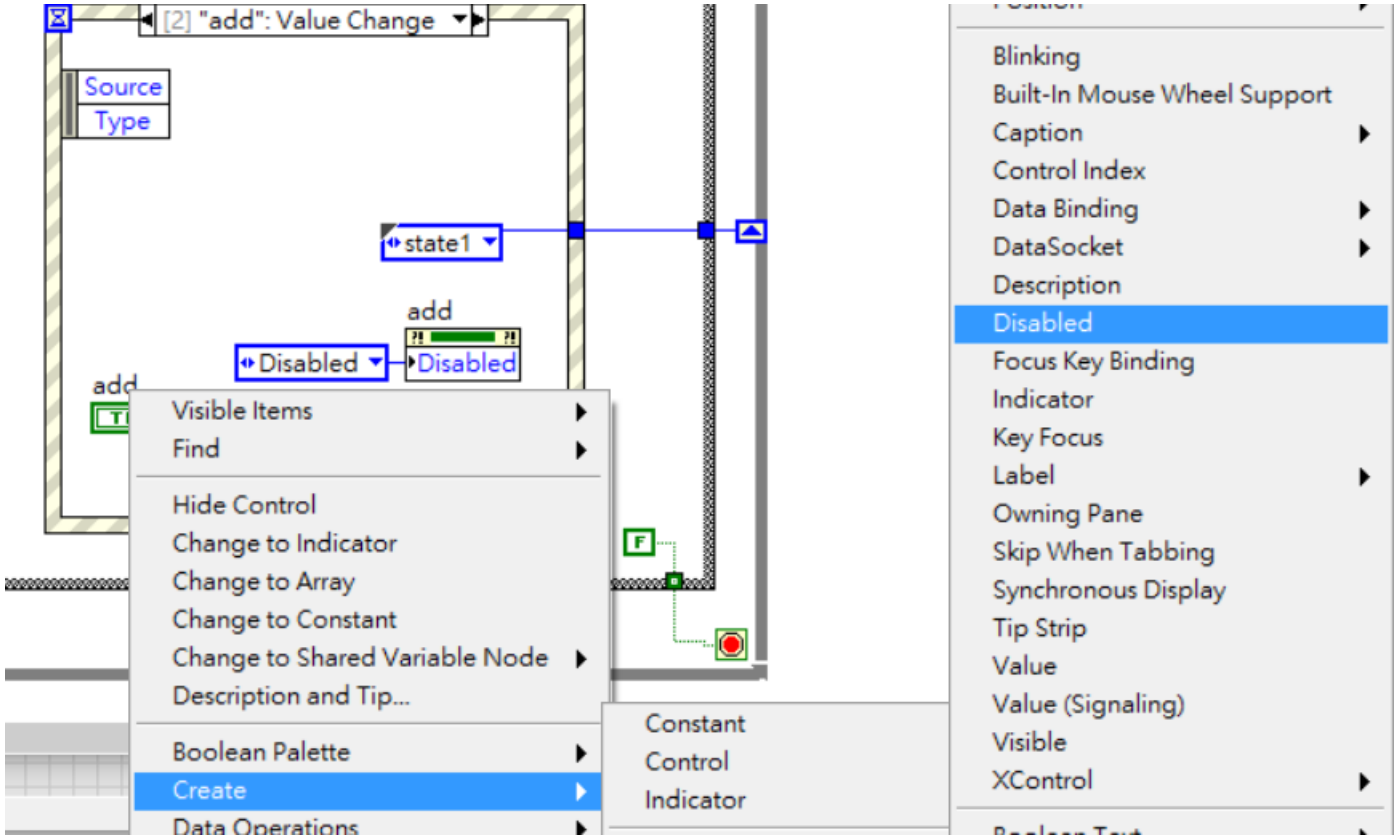






進入 state 時讓 button
消失不給按
(當然也可以 gray out)

Property node

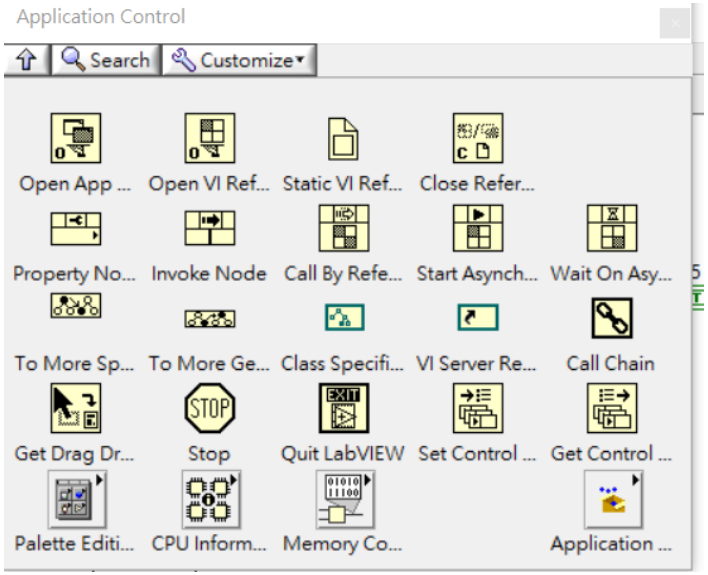


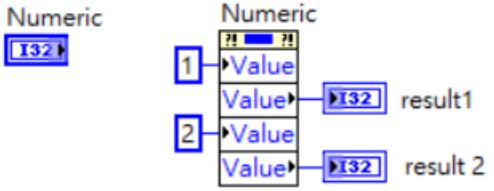
調整物件屬性 . Disable (enable, disable, disable & grayed out)

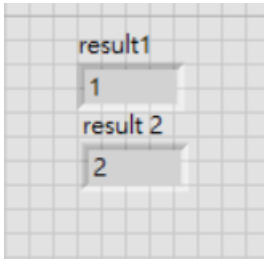
Property node

在 LabVIEW 中每個物件 (變數) 都可以利用右鍵選單 create 出 property node 或者是 invoke node, 你可以把它們看做是一些針對該物件 (變數) 性質的參數改變或者操作的函數 . 不同類型的變數 (比如 numeric 或 string) 可能擁有相同的 property node 項目 (比如說 value)

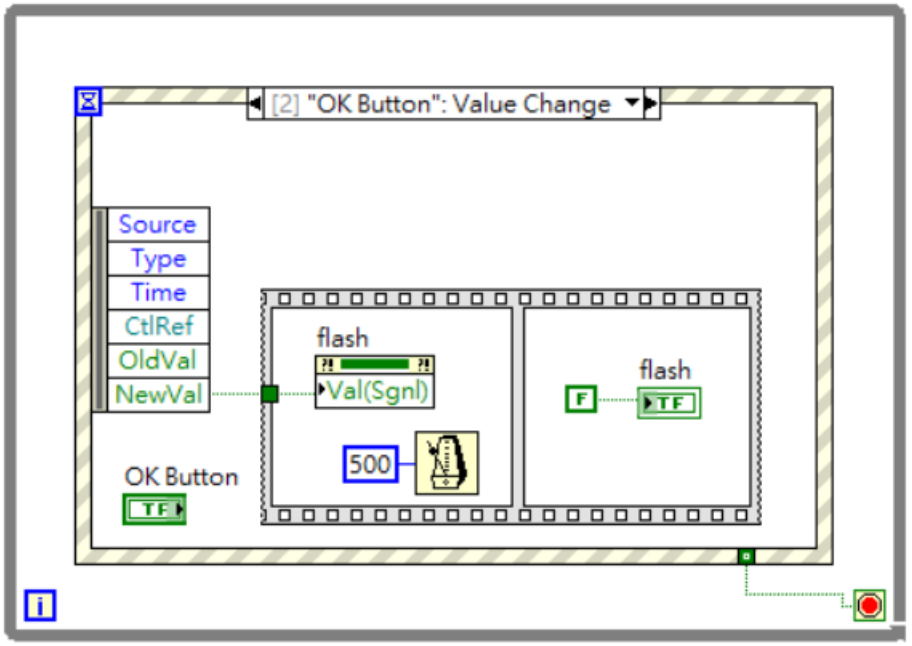
property node 可以從 application control 選單點選
property node 可以從右鍵選單選擇要 read 還是 write
也可以擴展拉出單一變數的多個 property, 要注意的是執行的順序是由上而下

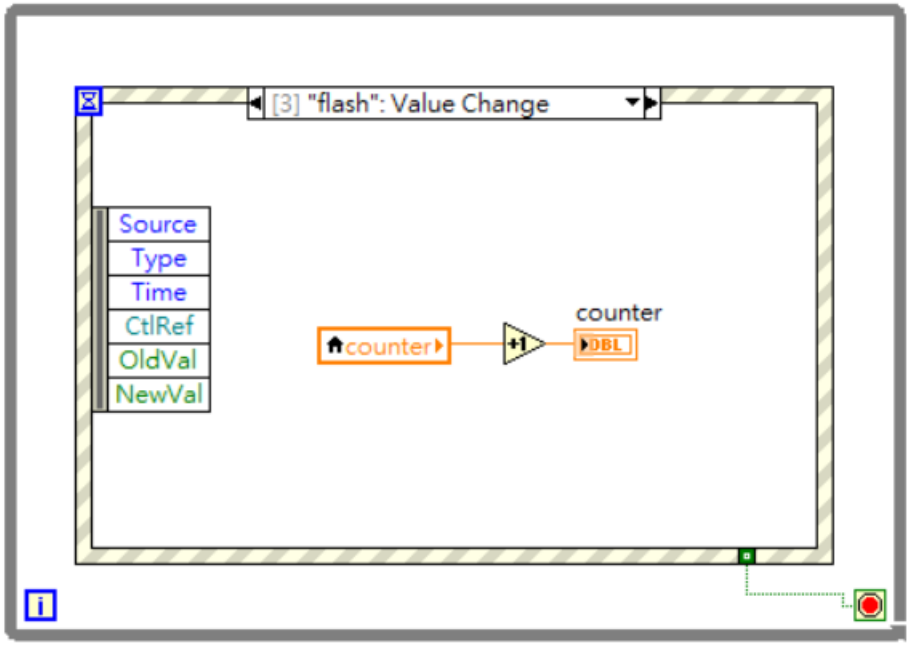


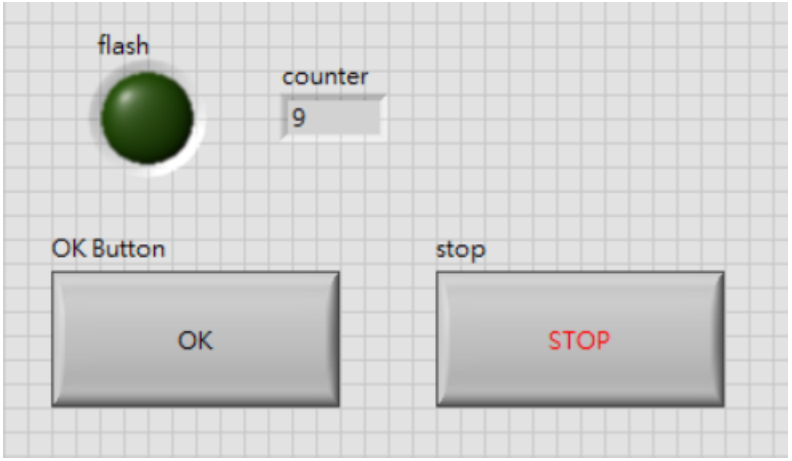




Property node 的 value & value(signaling)



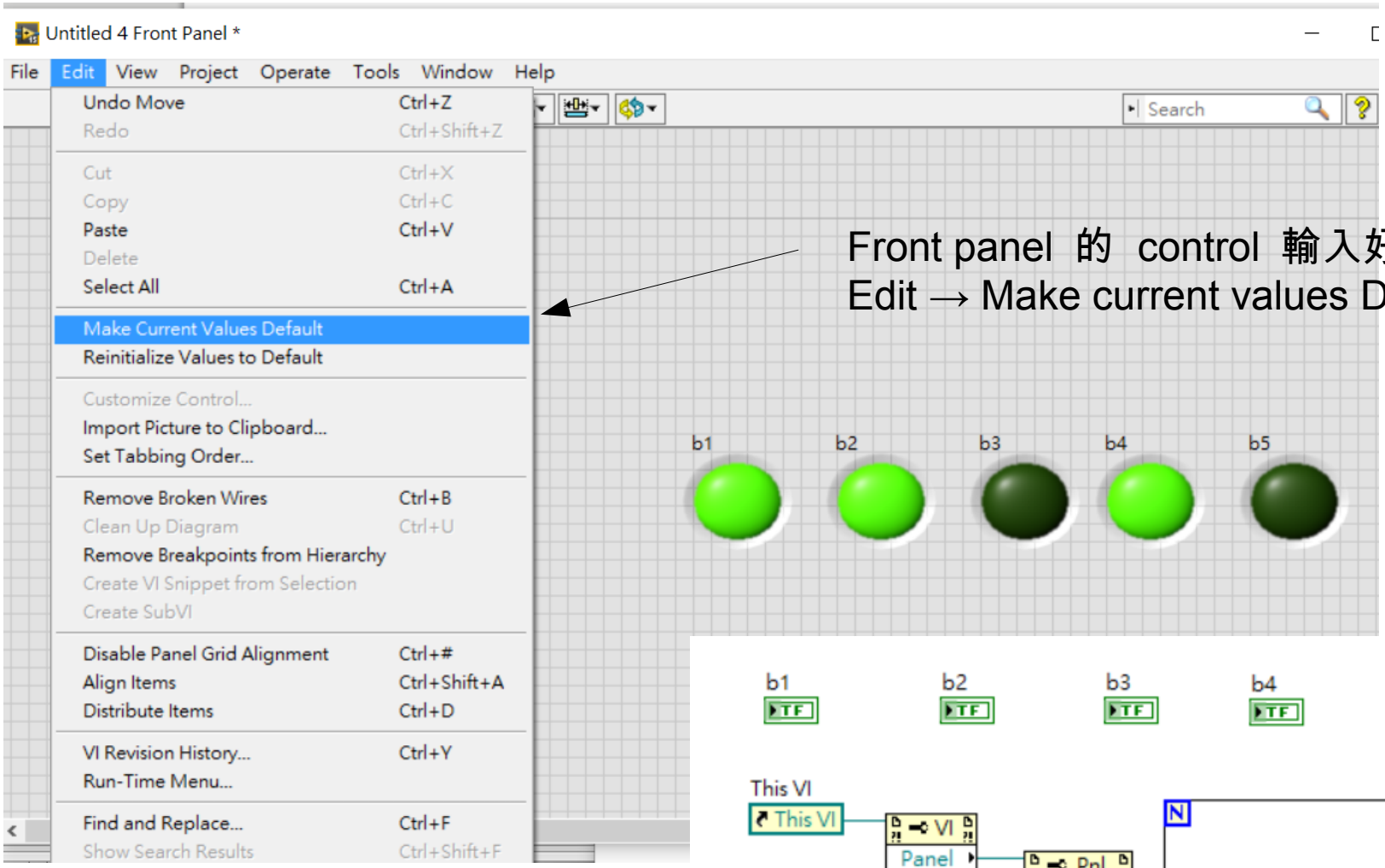




按一次 OK button 觸發一次 flash indicator, 使用 Val (Sgnl) = value (signaling) 來寫入 T, 同時有另外一個 event 偵測 flash 若是有發生 value change 則觸發 counter 加一

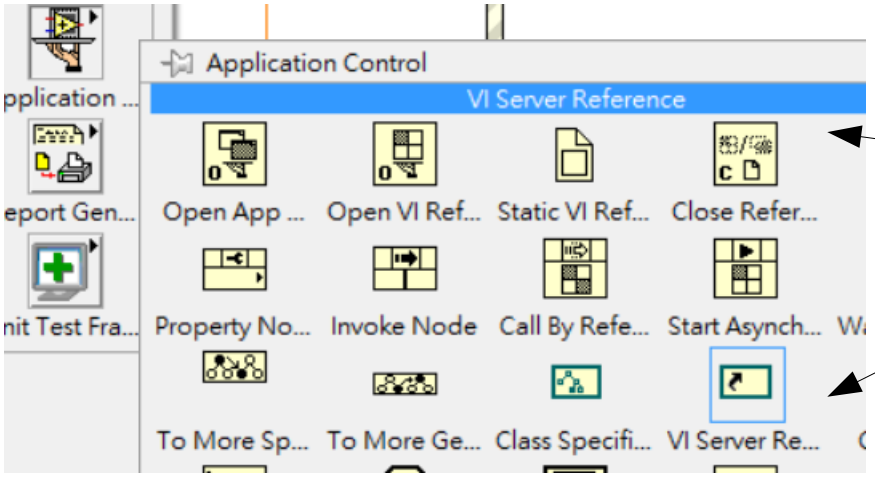
一般情況 indicator 的值時沒辦法由面板變更
但是 value (signaling) 可以觸發
你會發現 local variable 跟 value (property node) 都沒辦法觸發

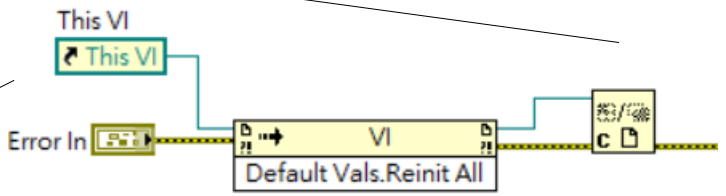
Property 與 invoke node 的應用：初始化

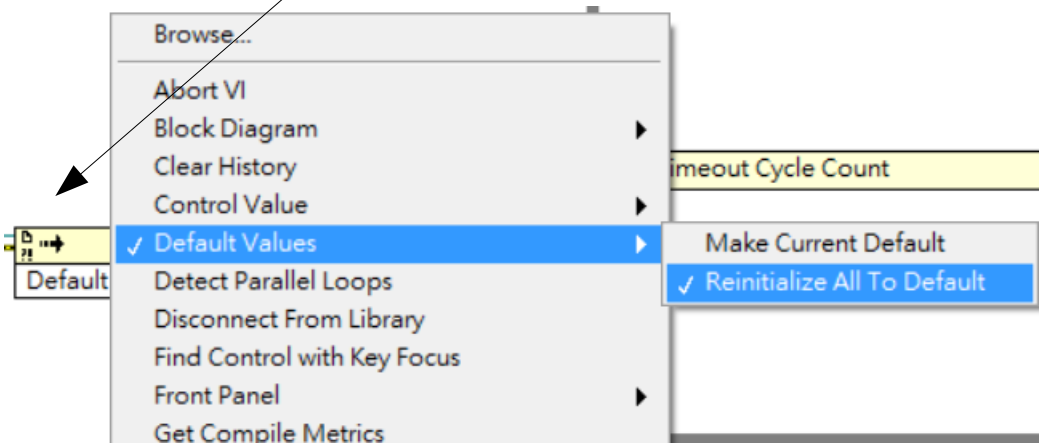


利用 vi 層級的 property node
抓出所有 control ref 然後下回復初始值的 invoke node

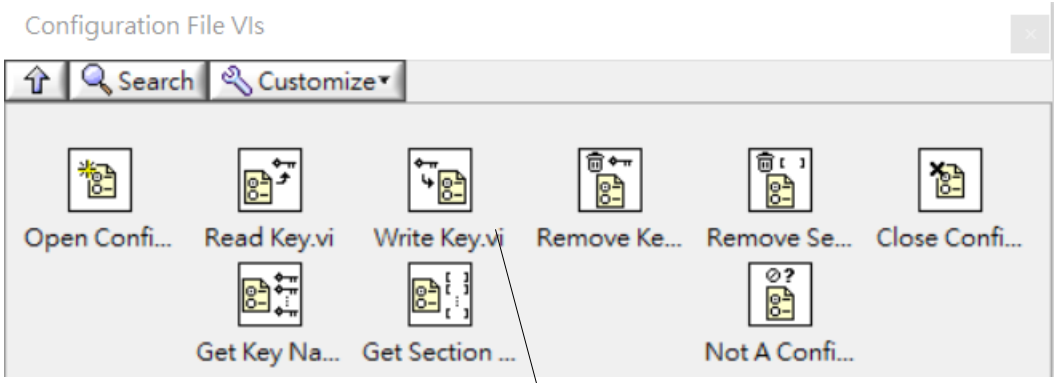
或者直接利用 vi 層級的 invoke node 下回復初始值的





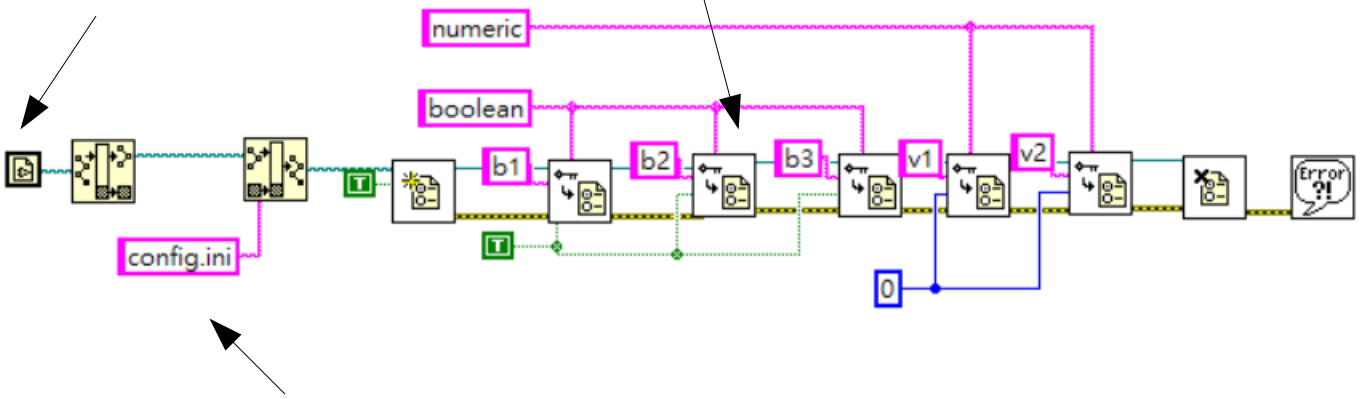


正規的初始化方法 (CLD 有時會指定此法儲存資料)



File IO→ configuration file VIs

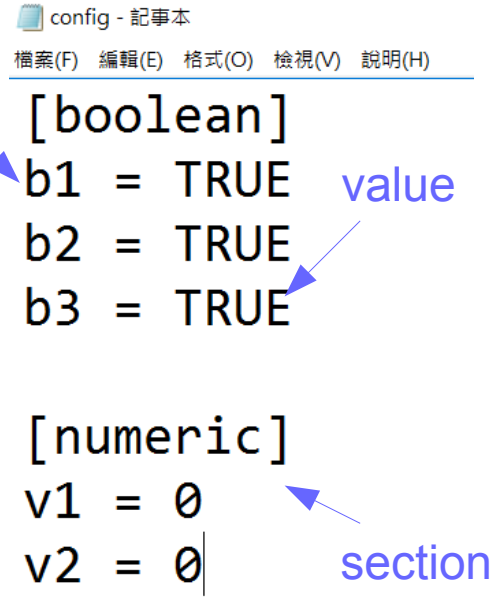
Current vi's path



取出 VI 上層目錄加入 ini 檔名

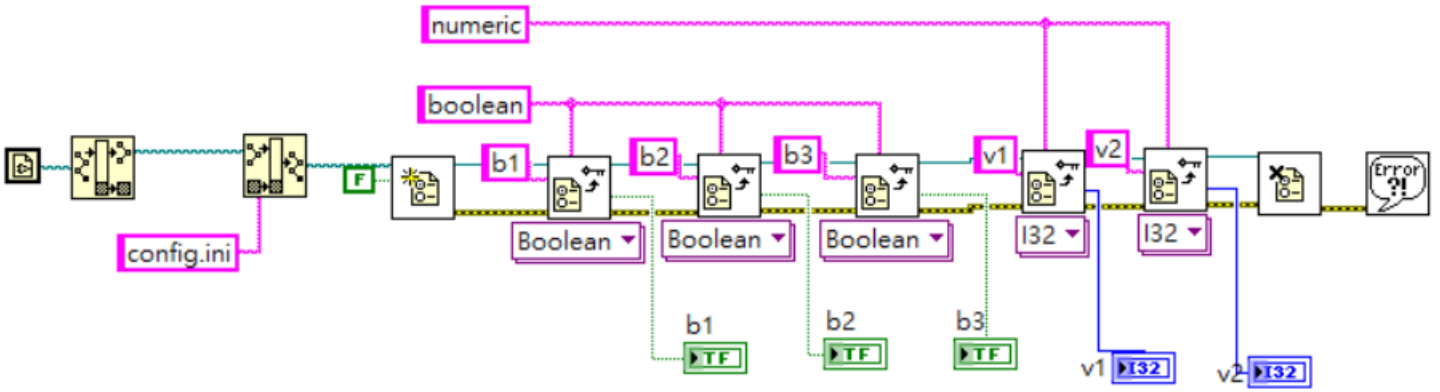
輸出 ini 結果

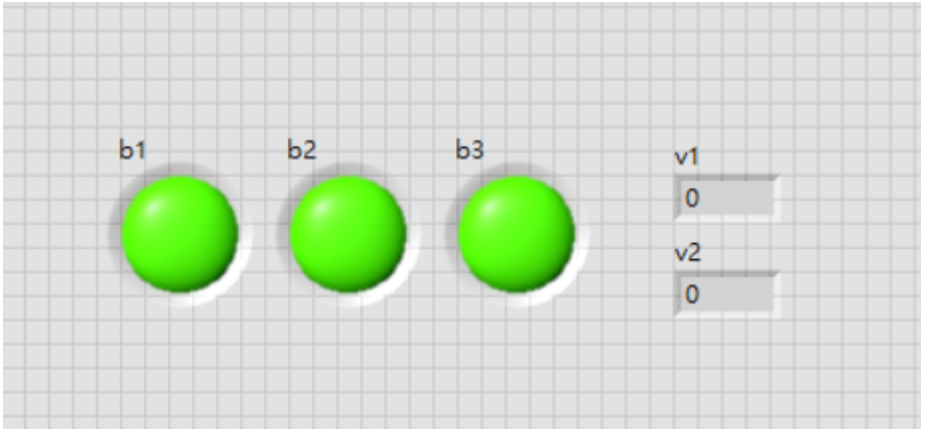
key



value

section

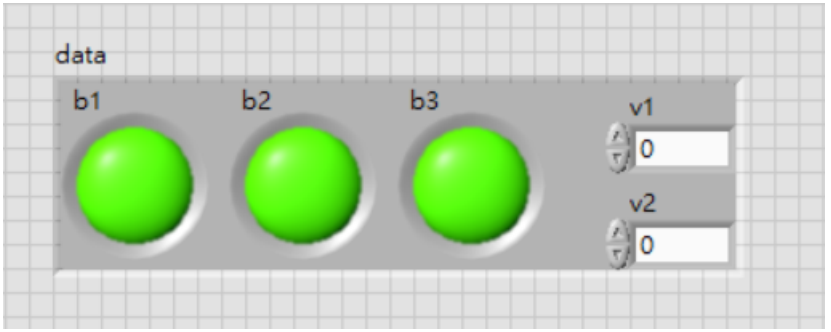




讀取資料反過來使用 read key.vi 即可．支援的類型有 numeric, string, boolean 等等，其他特定格式其實都可以透過 string 的方式讀取再來做後處理

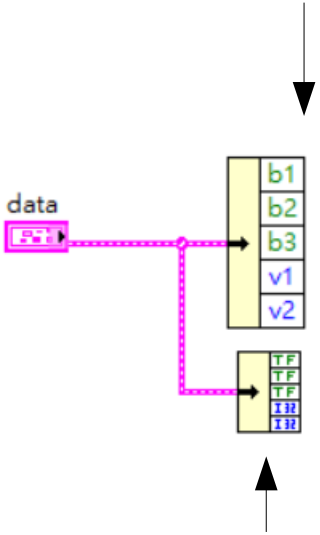
此法優點在於可以使用記事本直接編輯參數（方便）

使用 cluster 來封裝資料



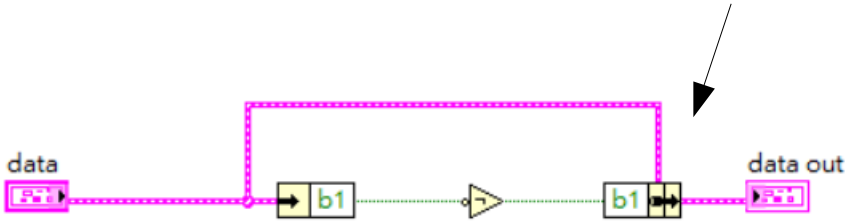
不同類型的資料可以包在起 , 在呼叫 subVI 輸出輸入資料時或者配合 shift register 使用可以有效降低引線數目
也常配合 type definition 使用

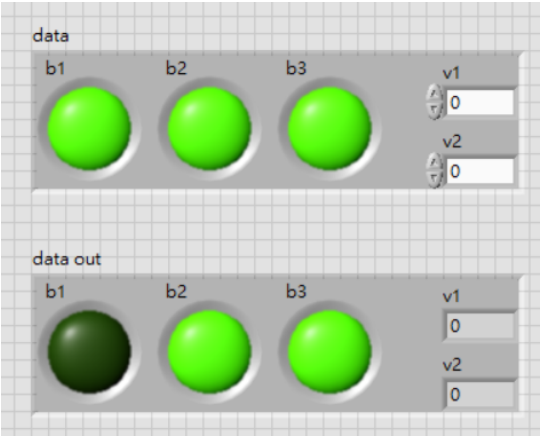
Unbundle by name

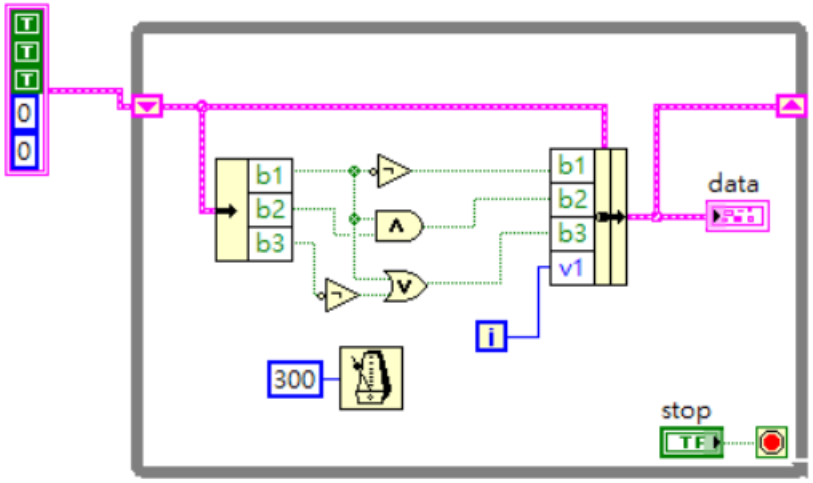


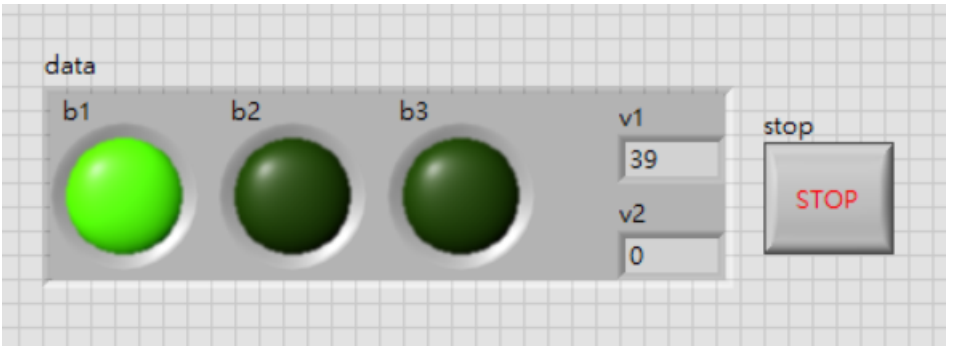
unbundle

重新 bundle 要提供 cluster 格式



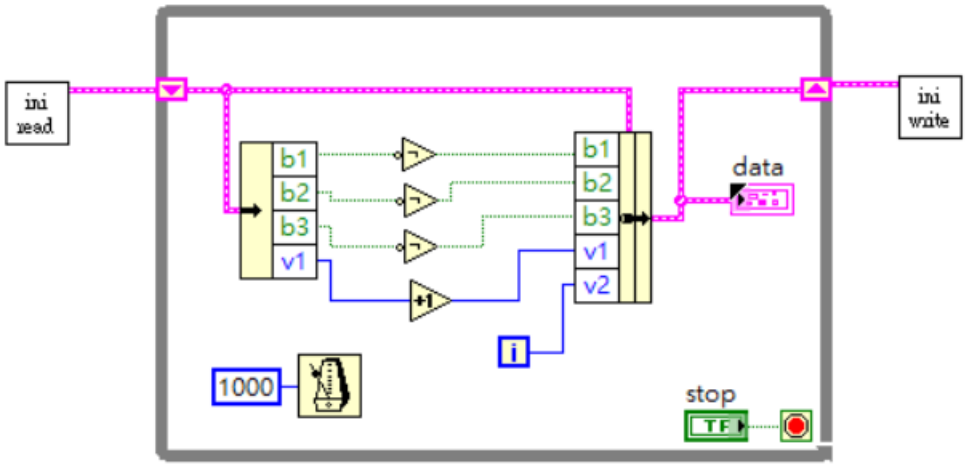






Cluster 應用在 shift register 可以有效降低接線

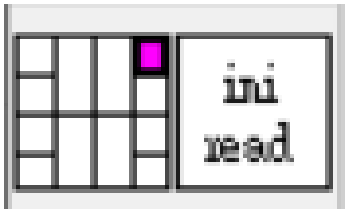
使用 cluster 配合 configuration data



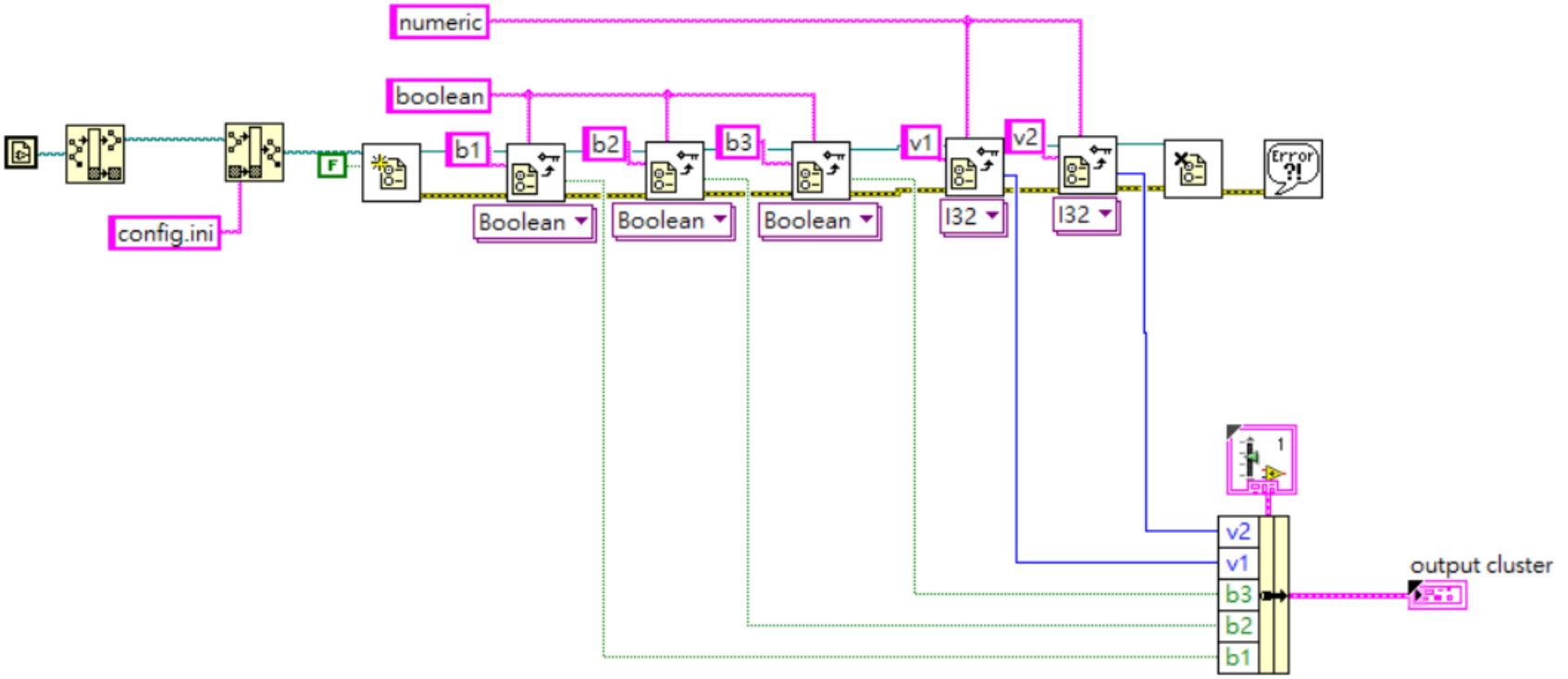
subVI, readini.vi

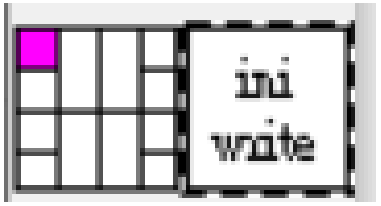
subVI, writeini.vi

v1 紀綠執行的總次數，v2 記錄此次執行程式的執行次數

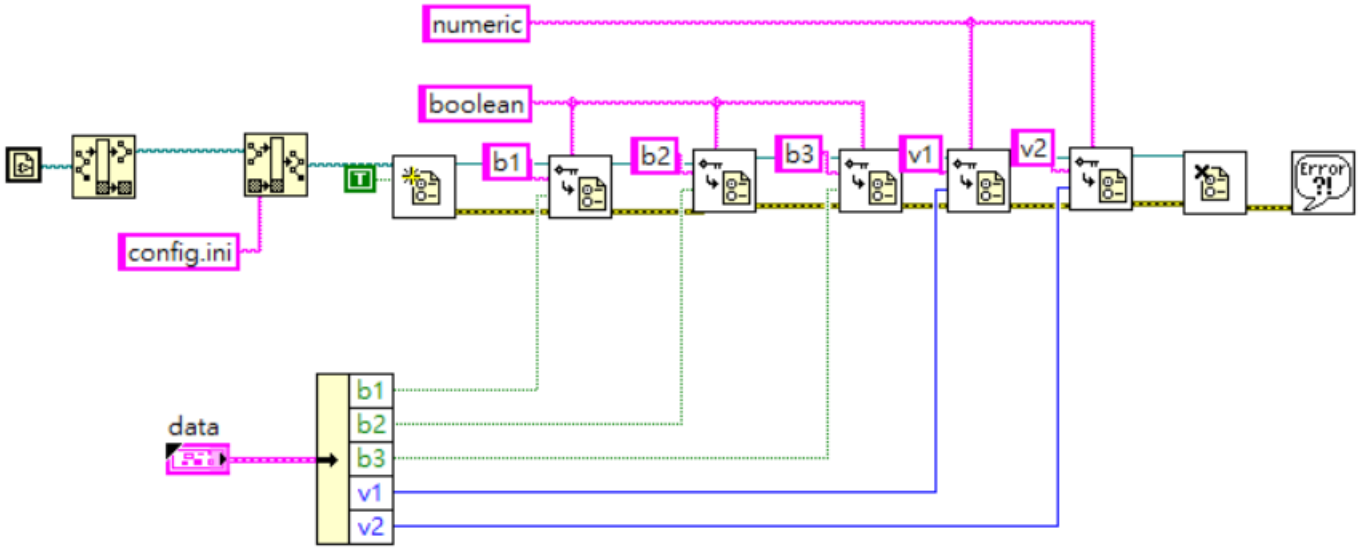

ini
read

readini.vi

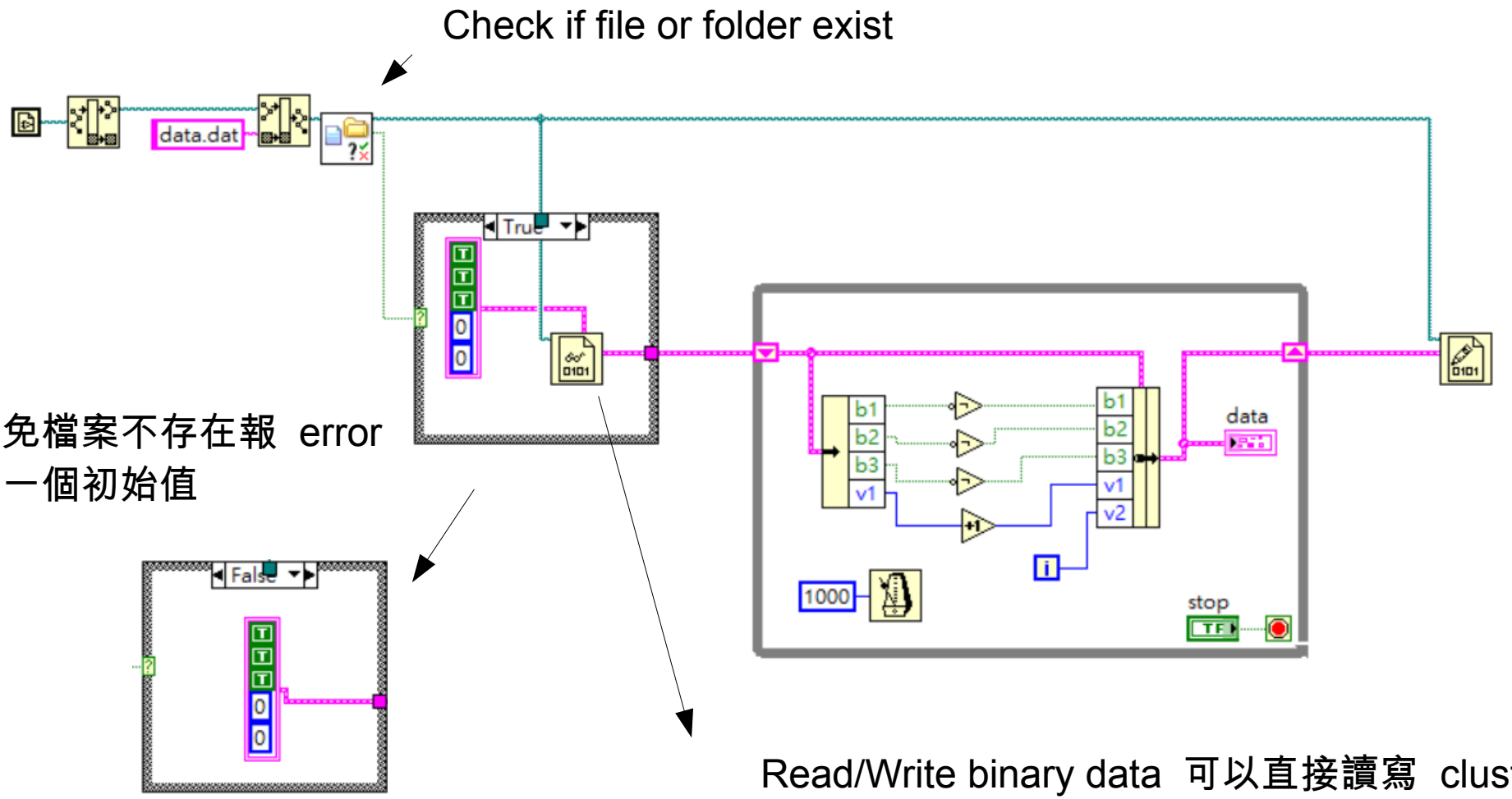

numeric
boolean
b1
b2
b3
v1
v2
Boolean
Boolean
Boolean
I32
I32
Error
config.ini
F
1
v2
v1
b3
b2
b1
output cluster


ini
write

Writeini.vi


numeric
boolean
b1
b2
b3
v1
v2
config.ini
Error
data
b1
b2
b3
v1
v2

比較偷懶的設定檔儲存方法

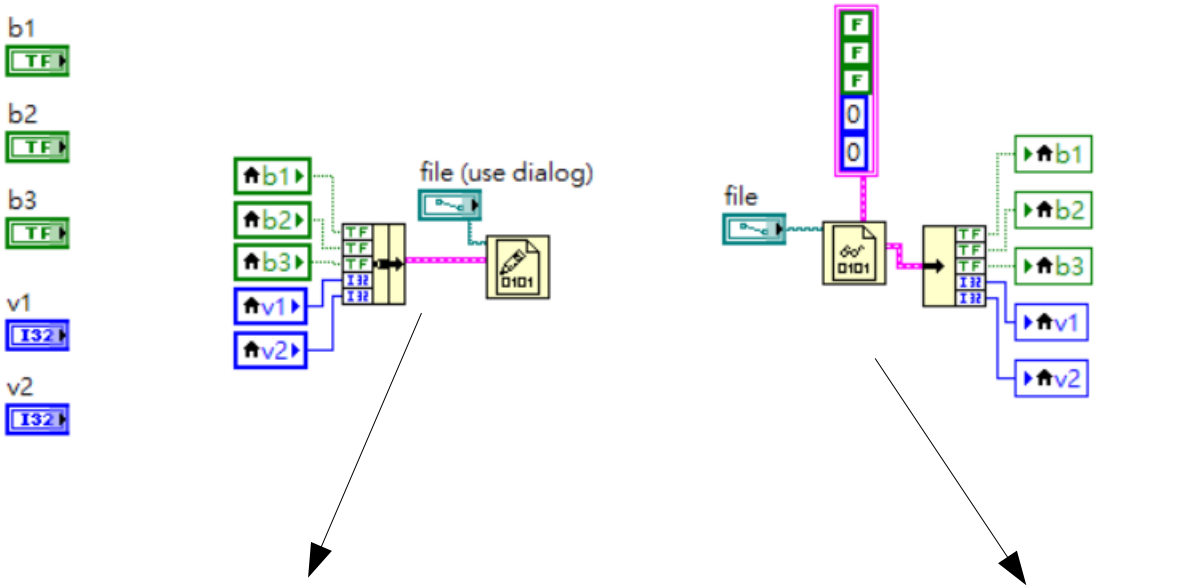


避免檔案不存在報 error
給一個初始值

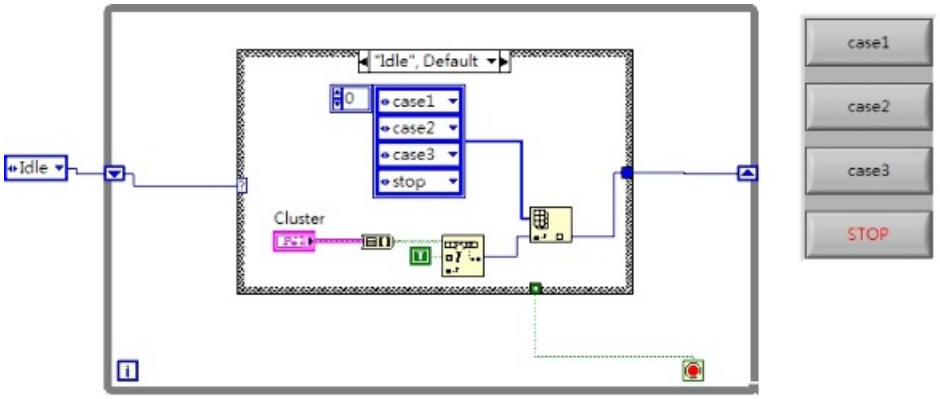
Read/Write binary data 可以直接讀寫 cluster
但是寫入時要記得給資料類型 (創一個
constant 當作輸入)

就算沒有 bundle 也可以偷懶直接把 front panel 所有你想儲存狀態的 control/indicator 建成 local variable 再 bundle 起來存成 binary file (通常少於 20 個變數可以這樣做)

優點 : 無腦又快 , 加點變化可讓使用者自訂多組設定檔 , 不像 reinit to default 只能存一組
缺點 : 存檔內容只有 LabVIEW 才能開啟辨識



不用 event structure 的多 button state machine



http://labview360.com/forum/forum_posts.asp?TC=R0HWWK0IW5SI&FID=23

練習

1. 輸出 99 乘法表 (一個 2D 陣列顯示乘法結果)

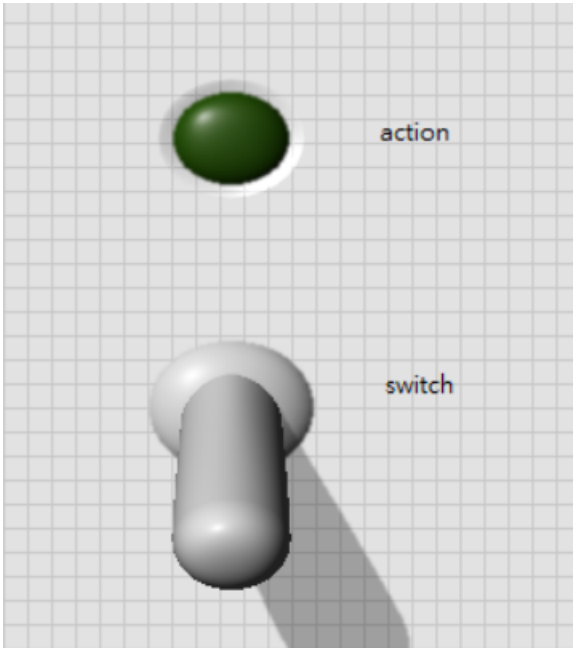
2. 輸出完整的 99 乘法表 (一個 2D 字串陣列顯示乘法結果 , ex.1*1=1,1*2=2,…)

3. 使用者輸入介面 , 可輸出任意 M*N 乘法表
4. 輸出 1-100 間所有質數

練習

Useless machine

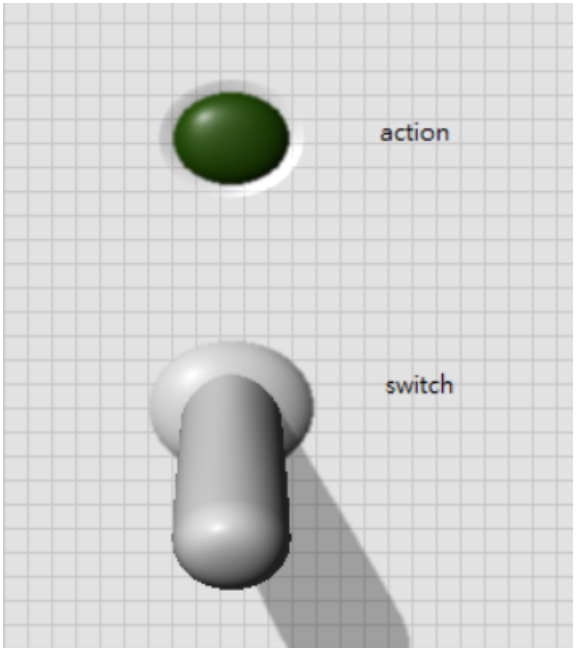
面板有一個 boolean indicator 跟一個 switch ON/OFF boolean control
當 switch 被切換至 ON 時 ,indicator 閃燈兩秒後 switch 被自動切回 OFF



練習

Useless machine

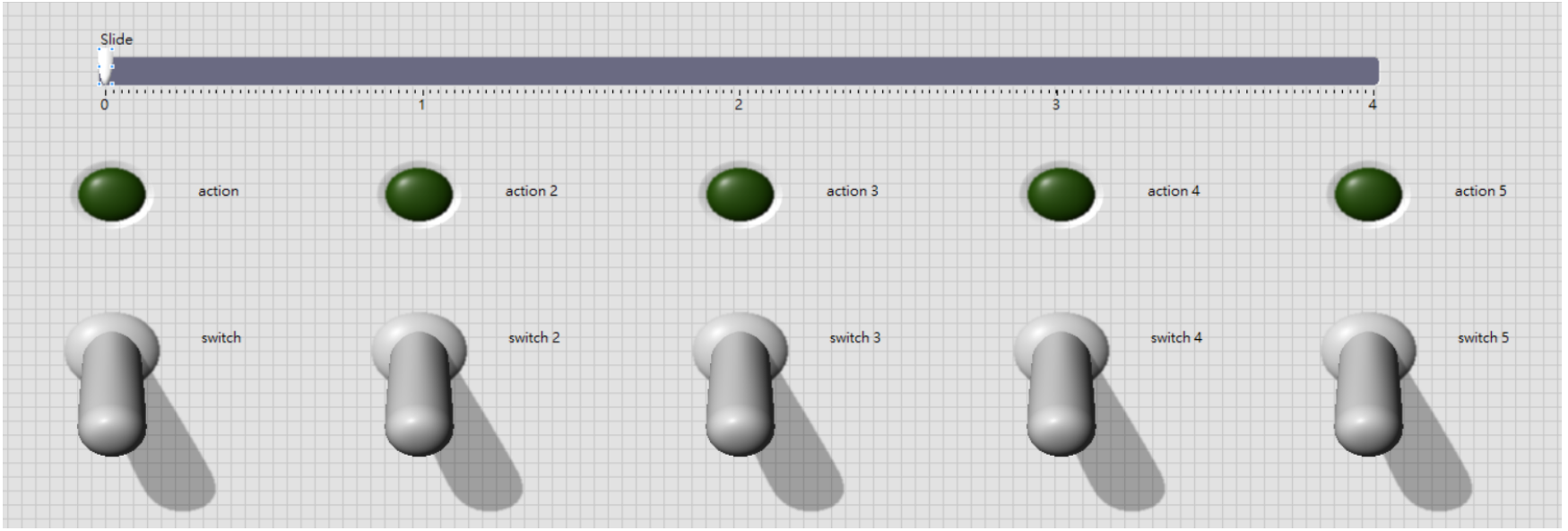
面板有一個 boolean indicator 跟一個 switch ON/OFF boolean control
當 switch 被切換至 ON 時 ,indicator 閃燈兩秒後 switch 被自動切回 OFF



練習

Useless machine 加強版

面板有 5 個 boolean indicator 跟 5 個 switch ON/OFF boolean control
當 switch 被切換至 ON 時 , slide 指向對應編號的 switch 位置 (每移動一單位耗時 1 秒 . 對應 indicator 閃燈兩秒後 switch 被自動切回 OFF



練習

CLD prepare kit 第 4 題 Event Structure Time Out
CLD prepare kit 第 9 題 Step Sequencer with Express Timer

練習

撇開資料存取之類的不談，幾乎只要靠 **event structure** 就可以把功能寫出來
最難的是要在 **4 hour** 內寫完 **(有經驗的大概也都要花 2~3 hour)**

Preparation E-kit for the NI Certified LabVIEW Developer (CLD) Exam

http://www.ni.com/gate/gb/GB_EKITCLDEXMPRP/US