九齊科技股份有限公司
Nyquest Technology Co., Ltd.

使用手冊

Q-Code

Power-Speech Format Programmer

Version 8.4

Aug. 29, 2025

目　錄

1 簡介

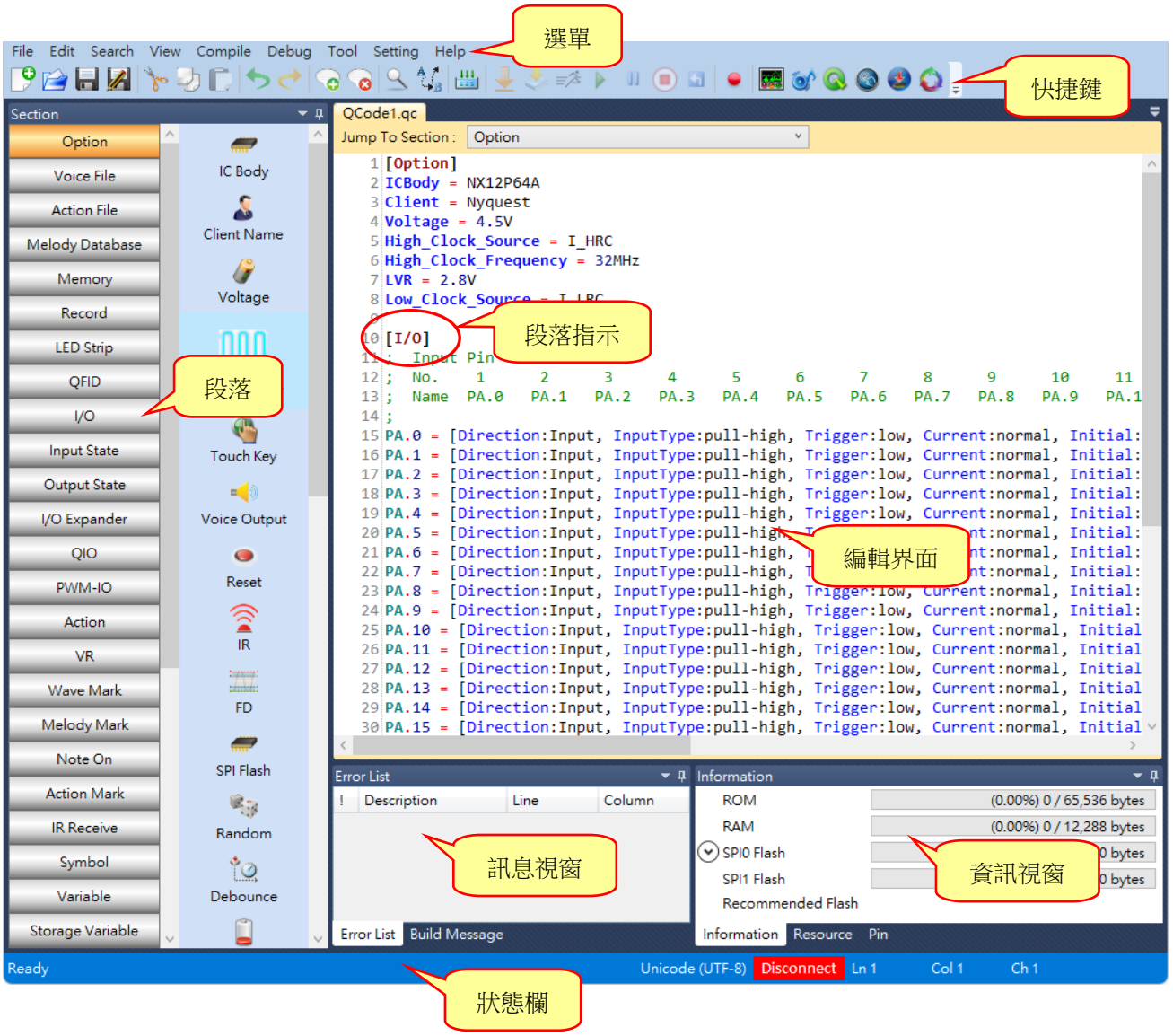
Q-Code 為九齊科技股份有限公司所提供，它是針對九齊科技的 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1 系列而開發的軟體工具。它提供簡易的圖形處理介面來完成應用程式的開發。無需瞭解硬體結構和組合語言的編寫，工程師依然可以利用 *Q-Code* 強大的功能來完成應用程式的開發，簡化了產品開發流程，提高了產品開發效率。對於初級工程師只需要進行簡單的培訓，在短時間內就可以完成某一產品的開發。

您可以聯繫九齊科技來獲得 *Q-Code* 的安裝程式檔，雙擊執行後進入安裝程式嚮導，然後依照畫面指示將可輕易完成安裝流程。

系統需求：

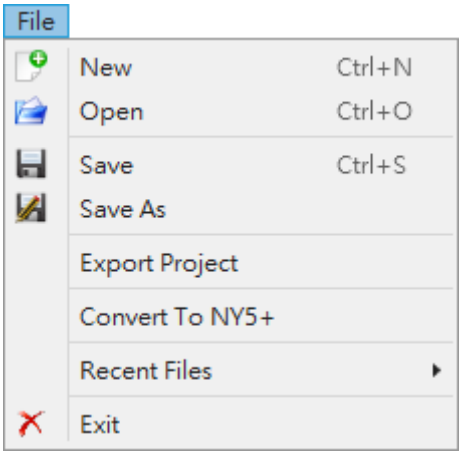
- Pentium 1.3GHz 或更高級處理器，Win7、Win8、Win10、Win11 作業系統。
- 至少 1G SDRAM。
- 至少 2G 硬碟空間。
- 顯示器和顯示卡支援解析度 1366x768 或更高。
- 需安裝 .Net Framework 4.8。

2 *Q-Code* 介面外觀


選單
快捷鍵
段落
段落指示
編輯界面
訊息視窗
資訊視窗
狀態欄
File Edit Search View Compile Debug Tool Setting Help
Section
Option
Voice File
Action File
Melody Database
Memory
Record
LED Strip
QFID
I/O
Input State
Output State
I/O Expander
QIO
PWM-IO
Action
VR
Wave Mark
Melody Mark
Note On
Action Mark
IR Receive
Symbol
Variable
Storage Variable
IC Body
Client Name
Voltage
Touch Key
Voice Output
Reset
IR
FD
SPI Flash
Random
Debounce
QCode1.qc
Jump To Section : Option
[Option]
ICBody = NX12P64A
Client = Nyquest
Voltage = 4.5V
High_Clock_Source = I_HRC
High_Clock_Frequency = 32MHz
LVR = 2.8V
[I/O]
; No. 1 2 3 4 5 6 7 8 9 10 11
; Name PA.0 PA.1 PA.2 PA.3 PA.4 PA.5 PA.6 PA.7 PA.8 PA.9 PA.1
PA.0 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial:
PA.1 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial:
PA.2 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial:
PA.3 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial:
PA.4 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial:
PA.5 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial:
PA.6 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial:
PA.7 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial:
PA.8 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial:
PA.9 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial:
PA.10 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial
PA.11 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial
PA.12 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial
PA.13 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial
PA.14 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial
PA.15 = [Direction:Input, InputType:pull-high, Trigger:low, Current:normal, Initial
Error List
! Description Line Column
Error List Build Message
Information
ROM (0.00%) 0 / 65,536 bytes
RAM (0.00%) 0 / 12,288 bytes
SPI0 Flash
SPI1 Flash
Recommended Flash
Information Resource Pin
Ready
Unicode (UTF-8) Disconnect Ln 1 Col 1 Ch 1

2.1 *Q-Code* 功能選單

2.1.1 檔案（File）



新增檔案（New）：新建一個 .qc 檔案。

開啟舊檔（Open）：開啟一個 .qc 檔案。

儲存檔案（Save）：儲存一個 .qc 檔案。

另存新檔（Save As）：將一個 .qc 檔案另存至其它檔案夾中。

匯出（Export Project）：將 .qc 以及其中所使用到的檔案匯出至另外的目錄中。

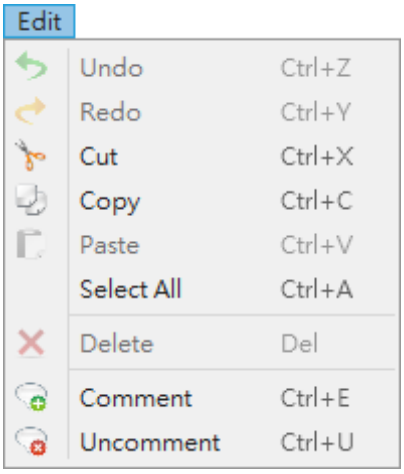
轉換到 NY5+（Convert To NY5+）：將 .qc 轉換成 NY5+專案。詳細說明請參閱 6.13。

注意：NY5 才支援此功能。

最近使用的檔案（Recent Files）：開啟最近使用過的 .qc 檔案。當關閉一個檔案後，該檔案會自動添加到“Recent Files”目錄中。

離開（Exit）：退出 *Q-Code*。

2.1.2 編輯（Edit）



復原（Undo）：復原最近一次編輯的動作。

取消復原（Redo）：取消最近一次復原的動作。

剪下（Cut）：剪下一選取區段文字內容。

複製（Copy）：複製一選取區段文字內容。

貼上（Paste）：貼上已剪下或複製的文字內容。

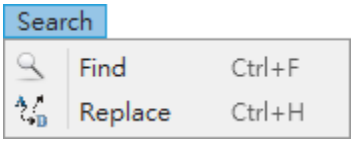
全選（Select All）：選取全部文字內容。

刪除（Delete）：刪除一選取區段文字內容。

註解（Comment Selection）：將游標所在位置或已選取多行進行註解。

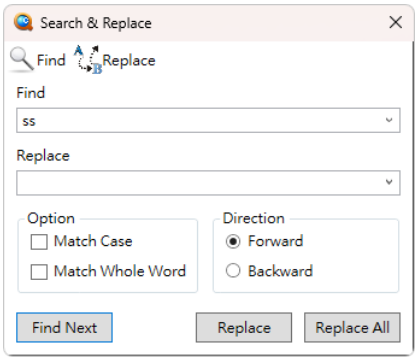
取消註解（Uncomment Selection）：將游標所在位置或已選取多行進行註解取消。

2.1.3 搜尋（Search）



尋找（Find）：在對話框內輸入欲尋找字串或數字。

取代（Replace）：把目前相符合的字串替換。



符合大小寫（Match Case）：在搜尋時區分大小寫。

符合單字（Match Whole Word）：只搜尋單字。（當未選取這個選項時，可能會搜尋到更長的單字）

向下搜尋（Forward）：向下找尋相符字串。

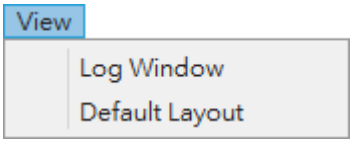
向上搜尋（Backward）：向上找尋相符字串。

繼續尋找（Find Next）：查找下一個。

取代（Replace）：把目前相符合的字串替換。

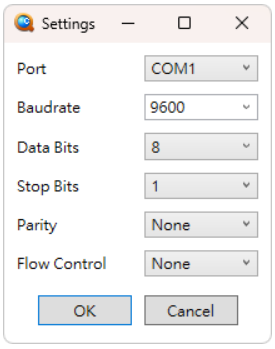
全部取代（Replace All）：把“相符合的字串”全部替換。

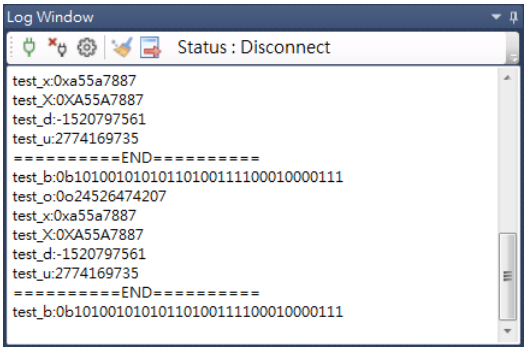
2.1.4 檢視（View）



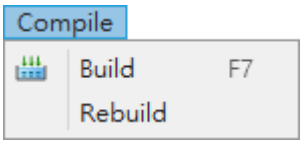
恢復預設視窗（Default Layout）：回復預設畫面。

紀錄視窗（Log Window）：接收由 UART 送出的訊息。使用設定畫面（如下圖）設定序列埠（Port）/ 鮑率（Baudrate）/ 同位檢察（Parity）等相關設定後，點擊連接按鈕即可開始進行資料接收。





2.1.5 編譯（Compile）

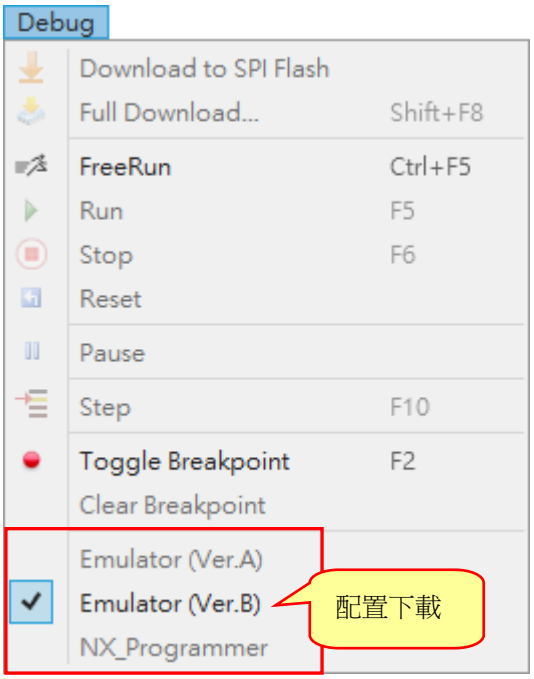


建置（Build）：將當前.qc 檔編譯成.bin 檔，以便下載到 ICE 或是 Flash Demo Board 作驗證。如果音源檔案被編譯過且未改變，就不會重新編譯，以加速編譯。編譯完成後，結果會顯示在 Build Message 視窗，而 ROM 的使用情況會顯示在訊息視窗（Information）。

重新建置（Rebuild）：不論音源檔案是否被編譯過，其已編譯資料都會被清除並重新編譯成.bin 檔。

2.1.6 偵錯（Debug）

此功能主要是當用戶在 *Q-Code* 的程式中可能需要用 ICE 來進行除錯。當使用此功能時，需先將 ICE 或 NX_Programmer 連接到電腦的 USB 埠（USB Port）。



下載到 SPI Flash（Download To SPI Flash）：下載程式到模擬器的 SPI Flash。

注意：僅 NY6 系列支援此功能。

完整下載（Full Download）：完整的下載程式和資料到模擬器和 SPI Flash。電腦需要連接 NX_Programmer。

注意：僅 NX1 系列支援此功能。

執行但不偵錯（Free Run）：ICE 執行程式，不可中斷進行偵錯。

開始偵錯（Run）：ICE 執行程式，可中斷進行偵錯。

注意：僅 NX1 系列支援此功能。

停止（Stop）：停止 ICE。

重置（Reset）：不關電源進行重置。

暫停（Pause）：暫停 ICE。

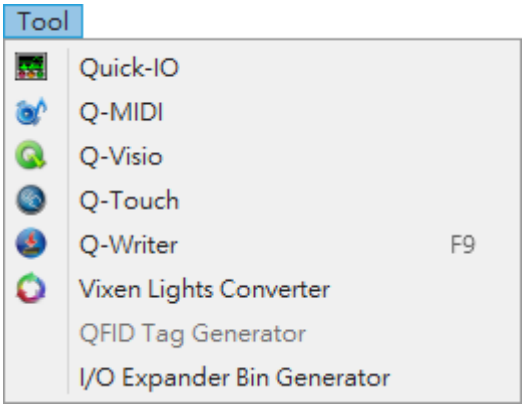
單步執行（Step）：逐行執行 ICE。

新增/刪除中斷點（Toggle Breakpoint）：新增或刪除中斷點。

清除所有中斷點（Clear Breakpoint）：清除所有中斷點。

配置下載：可透過此配置設定來選擇欲下載試聽的演示板。目前支援 Emulator Ver.A、Emulator Ver.B 和 NX_Programmer。在設定後每一次的下載都會依照此設定執行。

2.1.7 工具（Tool）



Quick-IO：此軟體是用來畫出輸出腳的信號，需結合.wav 檔使用，所產生的檔案為.nyq。

Q-MIDI：此軟體是用來編輯 MIDI 與音色庫，產生的檔案為.qmd，可在 NY5 / NY5+ / NY6 / NY7 / NX1 使用。

Q-Visio：此軟體是用來畫出輸出腳的信號，所產生的檔案為.vio。

Q-Touch：此軟件是用來掃描觸摸鍵的靈敏度，所產生的文件為.t9x 和.tnx。

Q-Writer：此軟體是用來將.bin 檔案燒錄在 Flash Demo Board、Romter 或 OTP 上以供驗證。

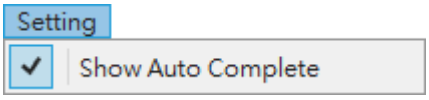
Vixen Lights Converter：此軟體是用來將.csv 檔案轉換成多個.csv 和.vio 檔案。

QFID Tag Generator：此處用來產生 QFID 應用中，Tag 所使用的.bin 檔案，僅可在 NY7 / NX1 使用。

I/O Expander Bin Generator：此工具用來產生 I/O Expander 所使用的 bin 檔。

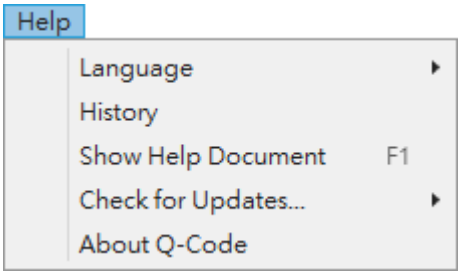
29

2.1.8 設定（Setting）



自動完成（Show Auto Complete）：搜尋建議功能的開關。

2.1.9 幫助（Help）



語言（Language）：切換 *Q-Code* 顯示語言。

改版紀錄（History）：檢視 *Q-Code* 最新改版訊息。

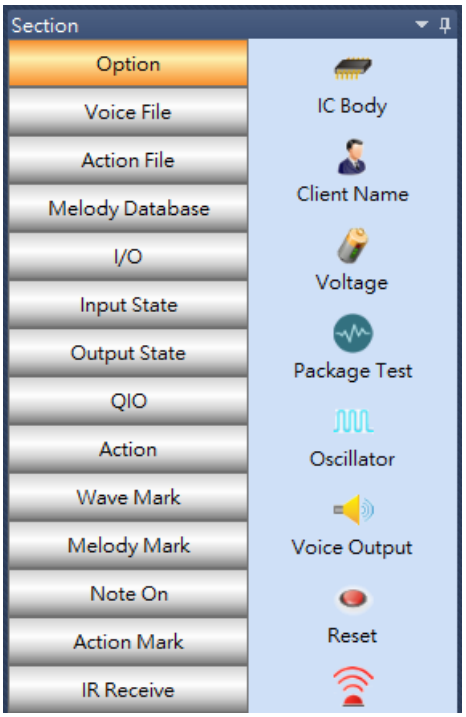
參考手冊（Show Help Document）：顯示 Q-Code Reference Manual。

檢查更新（Check for Updates…）：檢查是否有最新的 *Q-Code*、*NYASM*、*NYC_NX1* 版本，此功能需連上網路。

關於 *Q-Code* （About *Q-Code*）：顯示目前所安裝的 *Q-Code* 版本，以及技術支援的相關聯絡資訊。

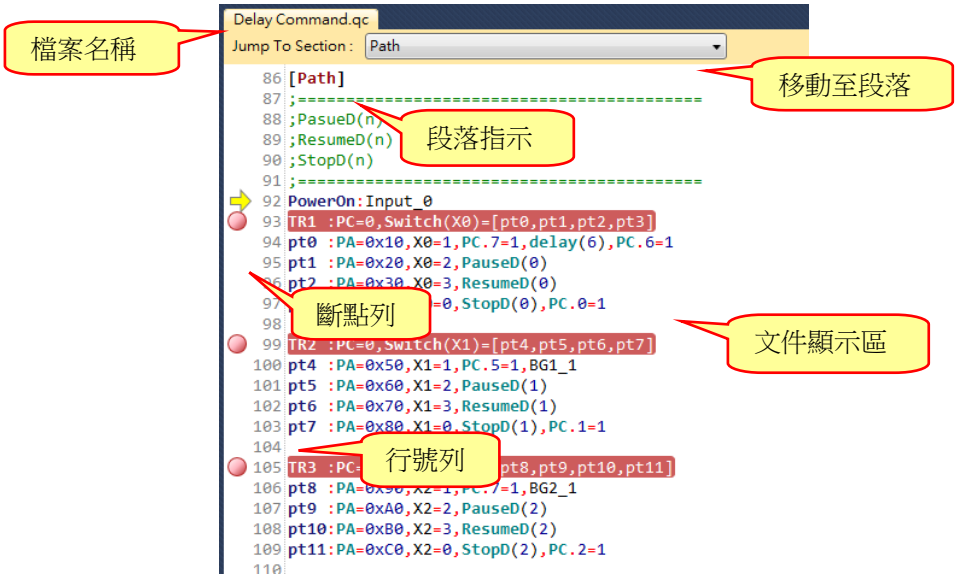
2.2 段落（Section）

段落提供多項簡易的使用者介面，讓使用者更容易完成 *Q-Code* 程式的開發。詳細操作方式請見段落描述。



2.3 編輯介面（Text Edit Area）

使用者編輯 *Q-Code* 程式的區塊，與一般編輯軟體的操作類似，編輯介面的左方有行號的顯示，*Q-Code* 程式相關的指令及關鍵字也會進行變色處理，除了功能表及快捷鍵的編輯功能外，使用者可以利用滑鼠右鍵功能表使用各項編輯功能，也可以利用段落來編輯 *Q-Code* 程式，關於 *Q-Code* 程式開發的詳細資訊請參閱 *Q-Code* 架構。



檔案名稱：顯示檔案名稱。

斷點列：程式運行／中斷點等圖示。

行號列：顯示行號。

移動至段落（Jump To Section）：將游標移動至選擇的段落。

文件顯示區：文件內容。

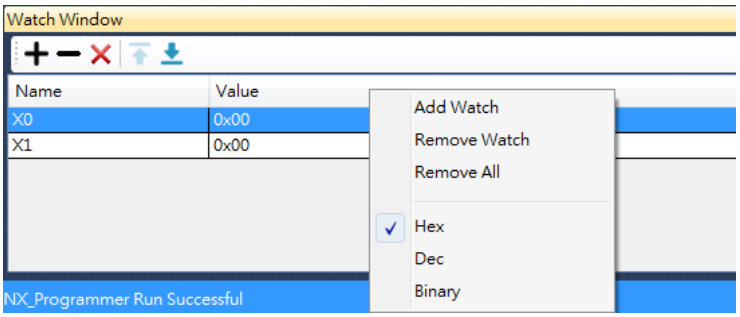
段落指示：顯示目前的段落。

2.4 偵錯模式（Debug Mode）

此功能支援 NX1。按下執行（Run）進入偵錯模式，就會出現兩個視窗：監看視窗（Watch Window）和暫存器視窗（Register Window）。

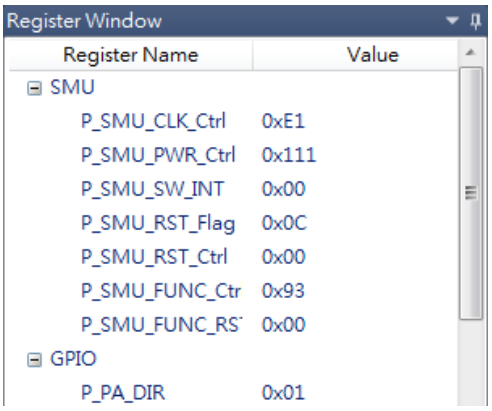
2.4.1 監看視窗（Watch Window）

使用者可以自行加入欲觀察的暫存器或是變數，方便追蹤。



2.4.2 暫存器視窗（Register Window）

偵錯過程中，顯示暫存器的數值，只有在 Pause 狀態下才會更新數值。

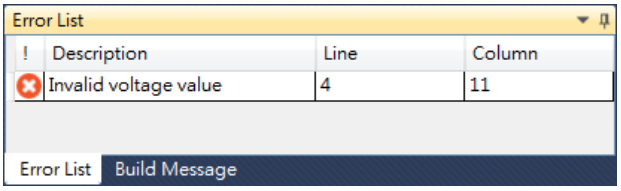


2.5 訊息視窗（Message Window）

分為兩個部份：錯誤清單（Error List）和建置訊息（Build Message）。

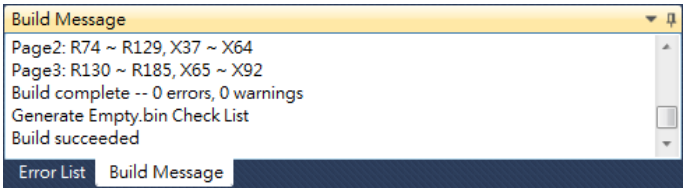
2.5.1 錯誤清單（Error List）

錯誤清單（Error List）負責顯示所有編輯介面要傳達給使用者的訊息，包括語法上的錯誤、錯誤訊息等等。使用者可以利用滑鼠左鍵點擊兩次錯誤的訊息，編輯介面就將可能產生錯誤的位置反白。



2.5.2 建置訊息（Build Message）

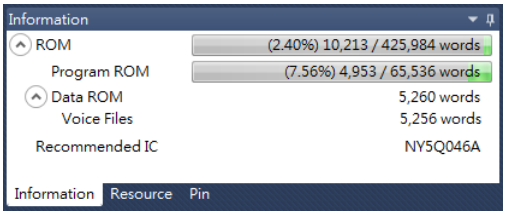
建置訊息（Build Message）負責所有編譯過程及下載到 ICE 過程需傳達給使用者的訊息。



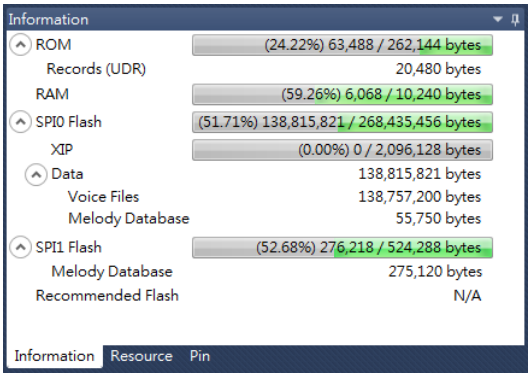
2.6 資訊視窗（Information Window）

2.6.1 記憶體使用資訊（Information）

編譯完成後會將記憶體的使用狀況顯示於此視窗，若有超過記憶體容量的部分，會以紅色字體顯示。Recommended IC 是依據程式使用的 ROM 大小，在同系列母體中提供選擇參考。

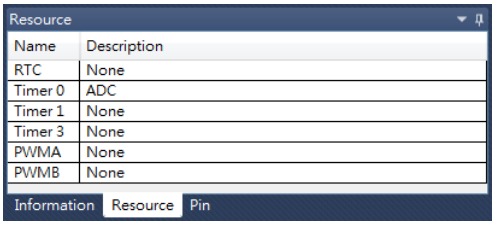


若為 NX1 母體，則顯示會改為 Recommended Flash 項目。此項目依據 SPI_Encoder 專案中的設置或是使用的 SPI Flash 容量，顯示適合的 N25Q 系列母體供選擇參考。



2.6.2 資源使用資訊（Resource）

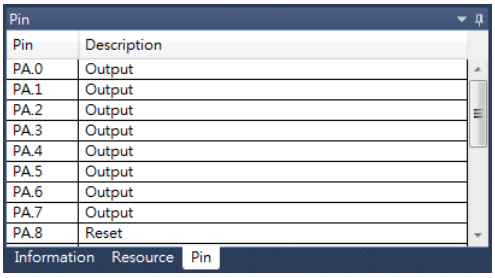
NX1 母體，會顯示目前 Timer 的使用資訊。



注意：RTC 以外的 Timer 分配，均採用獨占處理，如有衝突請重新安排相關應用程序。

2.6.3 腳位資訊（Pin）

會將目前腳位的使用狀況顯示於此視窗。

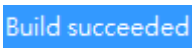


2.7 *Q-Code* 狀態欄（*Q-Code* Status Bar）

顯示 *Q-Code* 目前的各項狀態。包含 Build 結果（或 ICE 狀態）、ICE 圖示狀態、編輯行列、字數、INS 等等資訊。

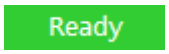
2.7.1 編譯結果與 ICE 狀態（Build Result and ICE Status）

下圖所示為.qc 檔案編譯後 OK or Fail：



2.7.2 ICE 連接狀態（ICE Connect Status）

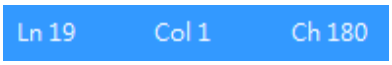
下圖所示為 ICE 已連接：



下圖所示為 ICE 未連接：

2.7.3 行列字元顯示（Row, Column, Characters）

顯示游標目前所在的列、行及字元所在位置。



3 *Q-Code* 架構

Q-Code 由不同的段落組成，各個段落代表不同的功能描述，某些段落僅在某些系列支援。下表為所有段落的列表以及各系列的支援程度。

段落	NY4	NY5	NY5+	NY6	NY7	NY9T	NX1
Option	O	O	O	O	O	O	O
Voice File	O	O	O	O	O	X	O
Action File	O	O	O	O	O	O	O
Melody Database	X	O	O	O	O	X	O
TouchKey File	X	X	X	X	X	O	X
Memory	X	X	X	X	X	X	O
SPI Flash	X	X	X	O	X	X	X
Record	X	X	X	X	X	X	O
LED Strip	X	X	X	X	X	X	O
QFID	X	X	O	X	O	X	O
I/O	O	O	O	O	O	O	O
Input State	O	O	O	O	O	O	O
Output State	O	O	O	O	O	O	O
I/O Expander	X	X	O	X	X	X	O
Input State Exp0~3	X	X	O	X	X	X	O
Output State Exp0~3	X	X	O	X	X	X	O
QIO	O	O	O	X	X	X	O
PWM-IO	X	X	O	X	X	X	O
Action	O	O	O	O	O	O	O
VR	X	X	X	X	X	X	O
Action Mark	O	O	O	O	O	O	O
Wave Mark	O	O	O	X	X	X	O
Melody Mark	X	O	O	O	O	X	O
Note On	X	O	O	O	O	X	O
PWM-IO Mark	X	X	O	X	X	X	X
IR Receive	O	O	O	O	O	X	O
Symbol	O	O	O	O	O	O	O
Variable	X	X	X	X	X	X	O
Storage Variable	X	X	X	X	X	X	O
Table	O	O	O	O	O	O	O
ASM	O	O	O	O	O	O	X

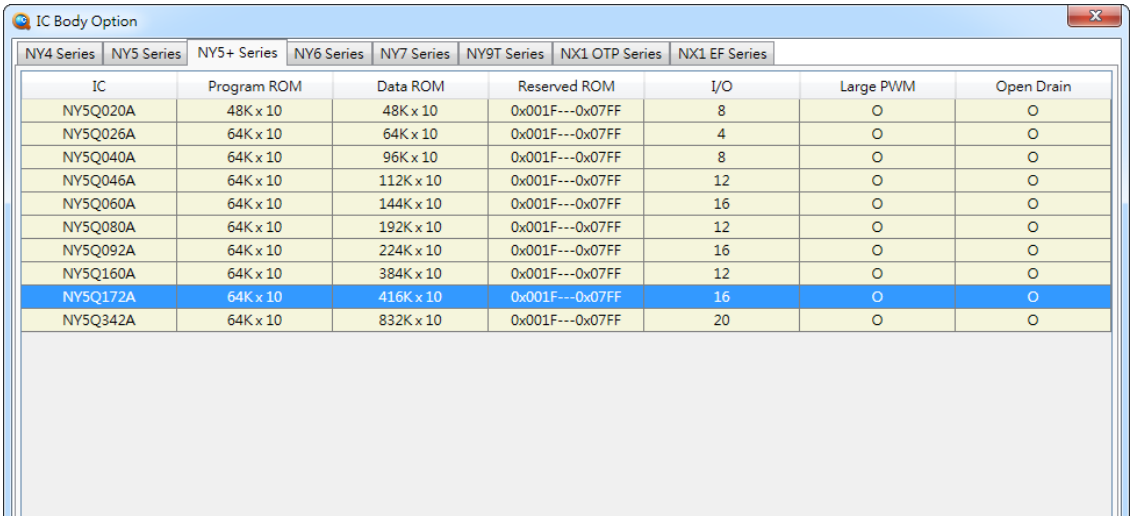
段落	NY4	NY5	NY5+	NY6	NY7	NY9T	NX1
C-Code	X	X	X	X	X	X	O
Macro	O	O	O	O	O	**O**	O
Sentence	O	O	O	O	O	**O**	O
Subroutine	O	O	O	O	O	**O**	O
QIO Custom	O	O	O	X	X	X	X
Path	O	O	O	O	O	O	O
Background1	O	O	O	O	O	O	O
Background2	O	O	O	O	O	O	O
Background3	X	X	X	X	X	X	O
Interrupt	X	X	X	X	X	X	O

3.1 Option

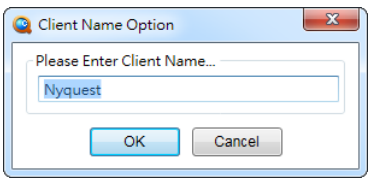
3.1.1 IC Body

選擇 IC 的型號。

例. **ICBody** = NY5Q172A



3.1.2 Client

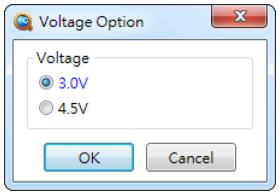


使用者或是公司名稱。必須填入客戶名稱，否則編譯將會不能通過。(此舉是用來保護程式開發者的所有權益)

例. **Client** = Nyquest

3.1.3 Voltage

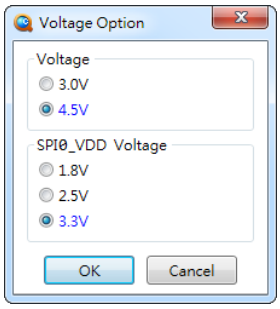
3.1.3.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1 EF



Voltage：選擇 IC 實際應用的工作電壓。(IC 的工作頻率在 3.0V 和 4.5V 會有點不同)

例. **Voltage** = 3.0V

3.1.3.2 NX1 OTP



Voltage：選擇 IC 實際應用的工作電壓。(IC 的工作頻率在 3.0V 和 4.5V 會有點不同)

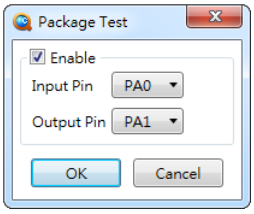
SPI0_VDD Voltage：選擇透過 SPI0 接腳 (PB.0 ~ PB.3) 連接的外部 SPI flash 工作電壓，SPI Flash 電源將由內部 LDO 提供。

例. **Voltage** = 4.5V

SPI0_VDD_Voltage = 4.5V

3.1.4 Package Test

3.1.4.1 NY4 / NY5 / NY7



當使用者需要代客封裝後的 IC 做基本功能檢測時，可開啟此功能，讓封裝後的 IC 能進行 Open / Short、Standby Current、OSC 頻飄和 Checksum 的驗證。

注意：

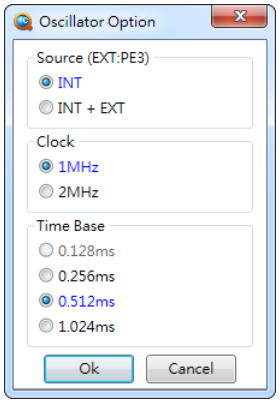
1. ***開啟此功能後，IC 的上電初始化時間會增加額外的 80ms。***
2. ***開啟此功能後，測試程式會佔用一小部分的 Program ROM Size。***
3. ***只有使用九齊公版封裝且有開啟封裝測試功能並用預設腳位出 Pin 的代客封裝需求，才被視為標準流程，出貨前才會進行封裝測試。***

*4. 由於 **NY5** 的腳位組態選定後就不能再變更，因此輸入與輸出腳位組態必需符合下表：*

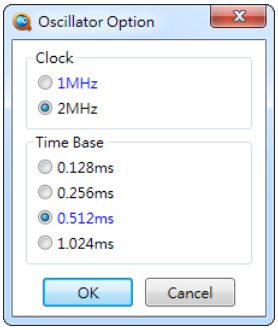
封裝測試腳位	組態
指令輸入腳位	Direct Input
	Direct Input IO
	Matrix Input
	Matrix Output
	Serial Control (SPI_Like) Clock
	Serial Control (SPI_Like) Data
	Serial Control (IR_Trigger) Data
	IR RX
訊號輸出腳位	Direct Input IO
	Direct Output
	Matrix Output

3.1.5 Oscillator

3.1.5.1 NY5



NY5



NY5PxxxB / NY5PxxxJ

Source : 選擇為 INT 表示 OSC 使用內部震盪電阻，選擇 INT+EXT 則表示 OSC 既可使用內部震盪電阻也可使用外部震盪器。(預設 INT)。

*注意：只有 **NY5** 支援 **EXT** 選項。*

例. **Oscillator** = INT + EXT

Clock：設定系統頻率。支援 1MHz / 2MHz（預設 1MHz）。

例. **Clock** = 1MHz

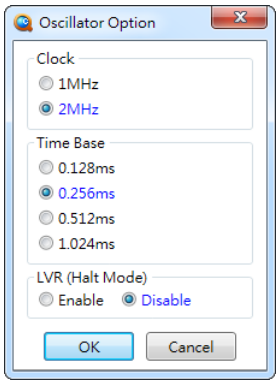
Time Base：*Q-Code* 運作的基礎計時時間，所有的時間計數皆參照此時間計時器。使用者可以自行設定所需的計時器。支援 0.128ms / 0.256ms / 0.512ms / 1.024ms（預設 0.512ms）。

例. **TimeBase** = 0.512ms

注意：

1. *因為效能問題，僅在 Clock=2MHz 下，才能使用 0.128ms。*
2. *計時器越快，則越佔 CPU 效能。*
3. *計時器與 Action 計時器共用，所以改變計時器的基準時間會連帶改變 Action 的計時基準。*
4. *Time Base 設定不會影響 Melody 的計時的基準時間。*

3.1.5.2 NY5+



Clock：設定系統頻率。支援 1MHz / 2MHz（預設 2MHz）。

例. **Clock** = 2MHz

Time Base：*Q-Code* 運作的基礎計時時間，所有的時間計數皆參照此時間計時器。使用者可以自行設定所需的計時器。支援 0.128ms / 0.256ms / 0.512ms / 1.024ms（預設 0.256ms）。

例. **TimeBase** = 0.512ms

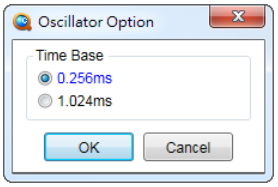
LVR (Halt Mode)：進 Halt Mode 時，低電壓復位功能的開關選項。

例. **LVR_When_Halt** = Enable

注意：

1. *計時器越快，則越佔 CPU 效能。*
2. *計時器與 Action 計時器共用，所以改變計時器的基準時間會連帶改變 Action 的計時基準。*
3. *Time Base 設定不會影響 Melody 的計時的基準時間。*
4. *LVR On 在 Halt Mode 將增加 Standby Current。*

3.1.5.3 NY6



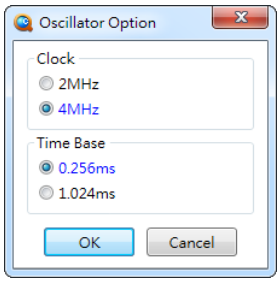
Time Base：*Q-Code* 運作的基礎計時時間，所有的時間計數皆參照此計時器。使用者可以自行設定所需的計時器。支援 0.256ms / 1.024ms（預設 0.256ms）。

例. **TimeBase** = 0.256ms

注意：

1. *計時器越快，則越佔 CPU 效能。*
2. *計時器與 Action 計時器共用，所以改變計時器的基準時間會連帶改變 Action 的計時基準。*
3. *Time Base 設定不會影響 Melody 的計時的基準時間。*

3.1.5.4 NY7



Clock：設定系統頻率。支援 2MHz / 4MHz（預設 4MHz）。

例. **Clock** = 4MHz

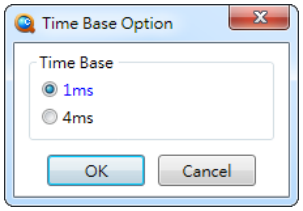
Time Base：*Q-Code* 運作的基礎計時時間，所有的時間計數皆參照此計時器。使用者可以自行設定所需的計時器。支援 0.128ms / 0.256ms / 1.024ms（預設 0.256ms）。

例. **TimeBase** = 0.256ms

注意：

1. *計時器越快，則越佔 CPU 效能。*
2. *計時器與 Action 計時器共用，所以改變計時器的基準時間會連帶改變 Action 的計時基準。*
3. *Time Base 設定不會影響 Melody 的計時的基準時間。*

3.1.5.5 NY9T



Time Base：*Q-Code* 運作的基礎計時時間，所有的時間計數皆參照此計時器。使用者可以自行設定所需的計時器。支援 1ms / 4ms（預設 1ms）。

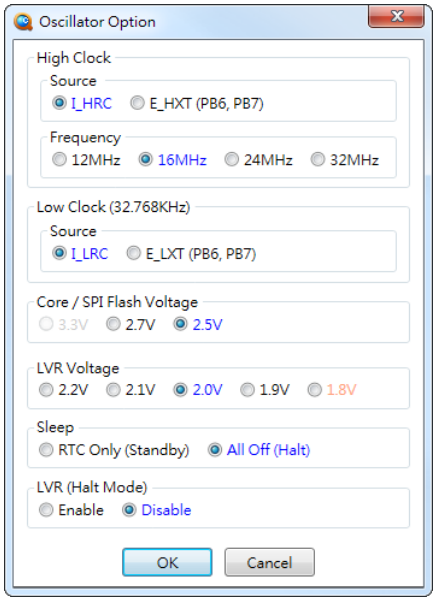
例. **TimeBase** = 1ms

注意：

1. *計時器越快，則越佔 CPU 效能。*
2. *計時器與 Action 計時器共用，所以改變計時器的基準時間會連帶改變 Action 的計時基準。*
3. *Time Base 設定不會影響 Melody 的計時的基準時間。*

3.1.5.6 NX1 OTP

NX1 震盪器選項分為二組 High_Clock 和 Low_Clock。High_Clock 及 Low_Clock 皆可使用外部振盪器。然而 IC 上僅有一組外部震盪連接腳位，僅有 High_Clock 或 Low_Clock 其中之一可使用外部振盪器。



High_Clock_Source：正常模式使用的時脈來源，可選擇內部（I_HRC）或外部（E_HXT）。

例. **High_Clock_Source** = I_HRC

High_Clock_Frequency：正常模式系統運作頻率。

例. **High_Clock_Frequency** = 24MHz

Low_Clock_Source：低速模式使用的時脈來源，可選擇內部（I_LRC）或外部（E_LXT）。頻率固定為 32.768KHz。

例. **Low_Clock_Source** = I_LRC

Core / SPI Flash Voltage：核心暨外部 SPI Flash 工作電壓，電源由 NX1 內部 LDO 提供。

注意：NX11P21A / NX11S / NX11M / NX12M / NX13M 支援 Core / SPI Flash Voltage 選項。

例. **SPI_Flash_Voltage** = 2.8V

LVR Voltage：低電壓復位的閥值，High Clock Frequency 的選擇會影響可用的閥值。

例. **LVR** = 2.4V

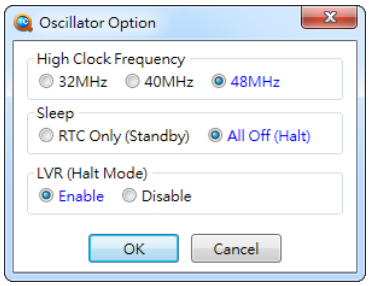
Sleep：IC 進入睡眠時，計時器運作的模式。設定為 RTC Only 時，會保留 RTC_2Hz 和 RTC_64Hz 運作。若是設定為 All Off 則所有的計時器都會停止，在此狀態下，只有 IO 中斷會喚醒 IC。

例. **Sleep** = All_Off

LVR (Halt Mode)：進 Halt Mode 時，低電壓復位功能的開關選項。Sleep 必須選擇 All Off 才能設定。

例. **LVR_When_Halt** = Enable

3.1.5.7 NX1 EF



High_Clock_Frequency：正常模式系統運作頻率。

例. **High_Clock_Frequency** = 48MHz

Sleep：IC 進入睡眠時，計時器運作的模式。設定為 RTC Only 時，會保留 RTC_2Hz 和 RTC_64Hz 運作。若是設定為 All Off 則所有的計時器都會停止，在此狀態下，只有 IO 中斷會喚醒 IC。

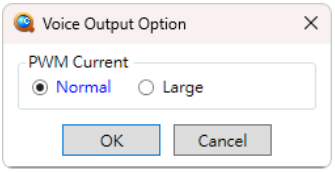
例. **Sleep** = All_Off

LVR (Halt Mode)：進 Halt Mode 時，低電壓復位功能的開關選項。Sleep 必須選擇 All Off 才能設定。

例. **LVR_When_Halt** = Enable

3.1.6 Voice Output

3.1.6.1 NY4

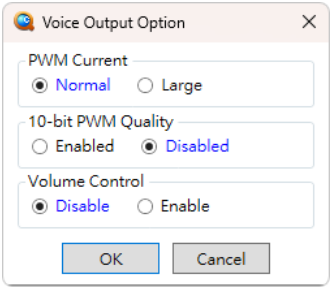


PWM Current：PWM 輸出電流。支援 Normal / Large。

*注意：**NY4A 不支持 PWM Current 選項。***

例. **PWM Current** = Normal

3.1.6.2 NY4PxxxC



PWM Current：PWM 輸出電流。支援 Normal / Large。

例. **PWM Current** = Normal

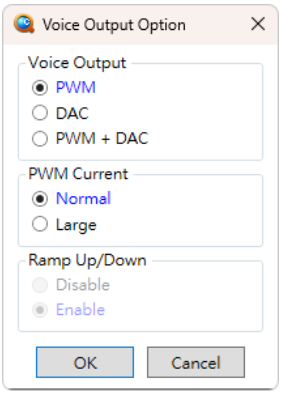
10-Bit PWM Quality：設置 10-bit PWM 是否開啟。

例. **PWM_10Bit** = Enable

Volume Control：設置音效控制是否開啟。

例. **Volume_Control** = Enable

3.1.6.3 NY5



Voice Output：設定喇叭的驅動型態。PWM 輸出可直接接上喇叭輸出，DAC 輸出需要加上放大電路。PWM+DAC 為自動偵測外部元件是使用 PWM 或者是 DAC 輸出。

例. **Voice Output** = PWM

PWM Current：PWM 輸出電流。支援 Normal / Large。

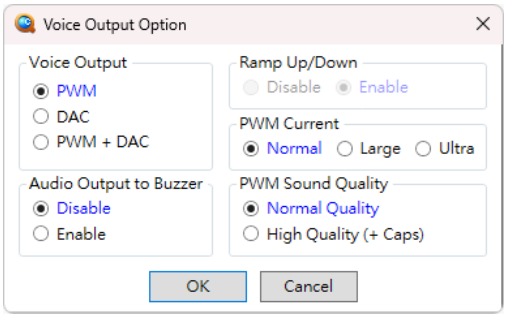
例. **PWM Current** = Normal

Ramp Up/Down：只在 DAC 模式下，可允許關閉自動 Ramp Up/Down 功能。

例. **RampUp/Down** = Disable

注意：並不是所有的應用需要關閉 RampUp / RampDown 功能，該功能針對外接 OP Amp 輸出時，要消除播音瞬間，因為 Ramp Up/Down 的動作所造成的"bo"聲。

3.1.6.4 NY5+



Voice Output：設定喇叭的驅動型態。PWM 輸出可直接接上喇叭輸出，DAC 輸出需要加上放大電路。PWM+DAC 為自動偵測外部元件是使用 PWM 或者是 DAC。

例. **Voice Output** = PWM

PWM Current：PWM 輸出電流。支援 Normal / Large / Ultra。

例. **PWM Current** = Normal

PWM Sound Quality: PWM 音質。支援 Normal / High。

例. **PWM_Sound_Quality** = Normal

Ramp Up/Down：只在 DAC 模式下，可允許關閉自動 Ramp Up/Down 功能。

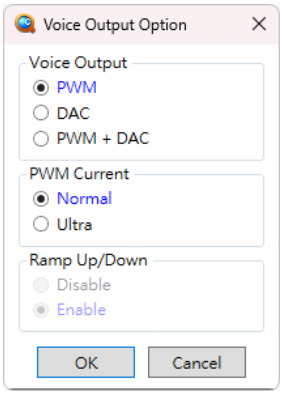
例. **RampUp/Down** = Disable

注意：並不是所有的應用都需要關閉 RampUp / RampDown 功能，該功能針對外接 OP Amp 輸出時，要消除播音瞬間，因為 RampUp / RampDown 的動作所造成的"bo"聲。

Audio Output to Buzzer：聲音由蜂鳴片輸出時，請啟用此選項。

例. **Audio_Output_Buzzer** = Enable

3.1.6.5 NY6



Voice Output：設定喇叭的驅動型態。PWM 輸出可直接接上喇叭輸出，DAC 輸出需要加上放大電路。PWM+DAC 為自動偵測外部元件是使用 PWM 或者是 DAC。NY6A 系列僅提供 PWM，NY6B / NY6C 系列提供 PWM / DAC / PWM+DAC 選項。

注意：NY6 的 Voice Output 僅設定初始輸出方式，若使用者往後需要變更時，可用 AudioMode 指令來變更輸出方式。

例. **Voice Output** = PWM

PWM Current：PWM 輸出電流。NY6 系列支援 Normal / Ultra。

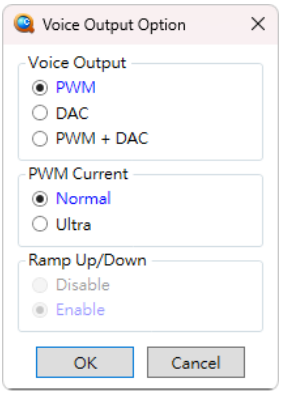
例. **PWM Current** = Ultra

Ramp Up/Down：只在 DAC 模式下，可允許關閉自動 Ramp Up/Down 功能。

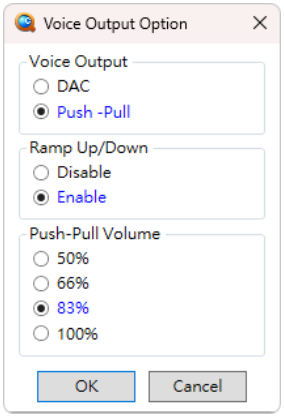
例. **RampUp/Down** = Disable

注意：並不是所有的應用都需要關閉 RampUp / RampDown 功能，該功能針對外接 OP Amp 輸出時，要消除播音瞬間，因為 Ramp Up/Down 的動作所造成的"bo"聲。

3.1.6.6 NY7



NY7A



NY7B / NY7C

Voice Output：設定喇叭的驅動型態。PWM 輸出可直接接上喇叭輸出，DAC 輸出需要加上放大電路。PWM+DAC 為自動偵測外部元件是使用 PWM 或者是 DAC。Push-Pull 則是類比推挽式輸出。NY7A 系列提供 PWM / DAC / PWM+DAC 選項(預設值 PWM)。NY7B / NY7C 提供 DAC 與 Push-Pull 選項。同時間只能有一種輸出方式（預設值 Push-Pull）。

注意：NY7 的 Voice Output 僅設定初始輸出方式，若使用者往後需要變更時，可用 AudioMode 指令來變更輸出方式。

例. **Voice Output** = PWM

例. **Voice Output** = Push-Pull

PWM Current：PWM 輸出電流。NY7A 系列支援 Normal / Ultra。

例. **PWM Current** = Ultra

注意：NY7B / NY7C 不支持 PWM Current 設置。

Ramp Up/Down：只在 DAC 模式下，可允許關閉自動 Ramp Up/Down 功能。

例. **RampUp/Down** = Disable

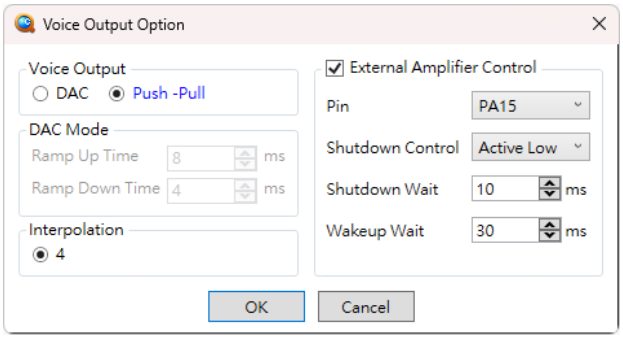
注意：並不是所有的應用都需要關閉 RampUp / RampDown 功能，該功能針對外接 OP Amp 輸出時，要消除播音瞬間，因為 RampUp / RampDown 的動作所造成的"bo"聲。

Push-Pull Volume：調整 Push-Pull 功放的 Analog 輸出音量設定。提供 100%、83%、66%、50% 選項（預設值 83 %）。

注意：NY7A 不支持 Push-Pull Volume 設置。

例. **Gain** = 83%

3.1.6.7 NX1 OTP



Voice Output：設定喇叭的驅動型態。DAC 輸出需要加上放大電路，Push-Pull 則是類比推挽式輸出。

例. **Voice Output** = Push-Pull

例. **Voice Output** = DAC

Ramp Up Time：設定 Ramp Up 的時間。會在設定的時間內完成 Ramp Up 功能。

例. **RampUp_Time** = 8ms

Ramp Down Time：設定 Ramp Down 的時間。會在設定的時間內完成 Ramp Down 功能。

例. **RampDown_Time** = 4ms

Interpolation：設定（倍頻）插點。NX1 OTP 固定為 4。

例. **Interpolation**= 4

External Amplifier Control：外部放大器控制功能。

例. **ExtAmp** = **Enable**

Pin：外部放大器控制腳位。

例. **ExtAmp_Pin** = **PA**.15

Shutdown Control：外部放大器控制方式，可使用 Active High / Active Low (預設為 Active Low)。

例. **ExtAmp_Shutdown_Control** = Active_Low

例. **ExtAmp_Shutdown_Control** = Active_High

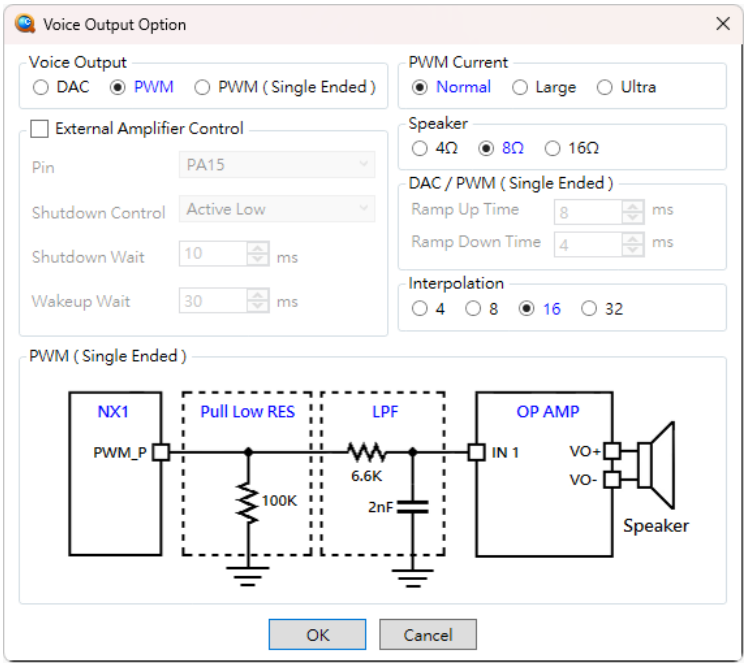
Shutdown Wait：關閉外部放大器等待時間。

例. **ExtAmp_Shutdown_Time** = 10ms

Wakeup Wait：喚醒外部放大器等待時間。

例. **ExtAmp_Wakeup_Time** = 30ms

3.1.6.8 NX1 EF



Voice Output：為設定喇叭的驅動型態。PWM 輸出可直接接上喇叭輸出，DAC 輸出需要加上放大電路，PWM (Single Ended) 輸出需要加上放大電路及另外的電阻及電容。

例. **Voice Output** = PWM

例. **Voice Output** = DAC

例. **Voice Output** = PWM_SE

Ramp Up Time：設定 Ramp Up 的時間。會在設定的時間內完成 Ramp Up 功能。

例. **RampUp_Time** = 8ms

Ramp Down Time：設定 Ramp Down 的時間。會在設定的時間內完成 Ramp Down 功能。

例. **RampDown_Time** = 4ms

Interpolation：設定（倍頻）插點。分為 4、8、16 與 32 四種（倍頻）插點供選擇，數值越大插點越多，反之數值越小插點越少，預設值 16。

例. **Interpolation**= 16

External Amplifier Control：外部放大器控制功能。

例. **ExtAmp** = **Enable**

Pin：外部放大器控制腳位。

例. **ExtAmp_Pin** = **PA**.15

Shutdown Control：外部放大器控制方式，可使用 Active High / Active Low (預設為 Active Low)。

例. **ExtAmp_Shutdown_Control** = Active_Low

Shutdown Wait：關閉外部放大器的等待時間。

例. **ExtAmp_Shutdown_Time** = 10ms

Wakeup Wait：喚醒外部放大器的等待時間。

例. **ExtAmp_Wakeup_Time** = 30ms

PWM Current：PWM 輸出電流。支援 Normal / Large / Ultra。

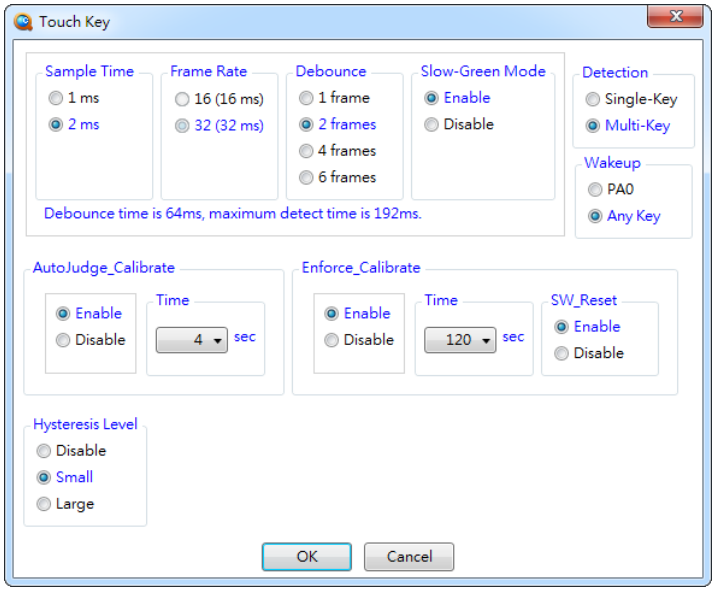
例. **PWM Current** = Large

Speaker：設定 Speaker 的阻抗值。

例. **Speaker** = 8ohm

3.1.7 Touch Key

3.1.7.1 NY9T



Sample Time：掃描一個觸摸鍵的時間。較長的取樣時間觸摸更精確，但偵測到 TouchKey 被觸摸的時間會較久。（預設 2ms）。

Frame Rate：在一個 Frame 要取樣的數量，即完成所有觸摸鍵的掃描時間。取樣的數量愈多則較不耗電，但偵測到 TouchKey 被觸摸的時間會較久。8/16 samples（Sample Time 1ms），16/32 samples（Sample Time 2ms）。

Debounce：連續幾個 Frame 被偵測到，按鍵才會成立。愈多 Frame 觸摸更精確，但偵測到 TouchKey 被觸摸的時間會愈久，對於較容易有外部干擾的環境建議選擇多些 Frame。

Slow-Green Mode：只有在 TouchKey 設定使用超過 8 個鍵時（含 8 個），才能開啟或關閉此項功能。若使用者有下達“TouchKey_Scan_Slow”指令：則 Slow-Green Mode 開啟時表示在進睡眠後會維持 Slow-Green Mode，反之關閉 Slow-Green Mode 時表示進睡眠後會維持 Slow Mode。Slow-Green Mode 比 Slow Mode 更省電，但偵測到 TouchKey 被觸摸的時間會較久。（預設 Enable）

TouchKey_DebounceTime 為一般模式下偵測到 TouchKey 被觸摸的時間；而 TouchKey_DetectTime 為進入睡眠後的最長偵測時間，實際上可能少於最長偵測時間。公式如下：

Normal Mode = SampleTime x FrameRate x Debounce

Slow Mode = SampleTime x FrameRate x (3 + Debounce)

Slow-Green Mode = SampleTime x FrameRate x (4 + Debounce)

注意：

1. *選用 NY9T016A 時，Frame Rate 固定為 16 samples。*
2. *NY9T001A/NY9T004A 無 Slow-Green Mode 設定。*
3. *Slow-Green Mode 只有在進入睡眠後，且切換到 TouchKey_Scan_Slow 才有效。*
4. *正常工作模式下，維持 Normal Mode Scan，不管有沒有下達*"TouchKey_Scan_Slow"指令。

例.

```
TouchKey_DebounceTime = 64ms { SampleTime=2ms, FrameRate=32ms, Debounce=2,
Slow-Green_Mode=Enable }          ; 一般模式偵測觸摸的掃描時間為 64ms。
TouchKey_DetectTime = 192ms       ; 睡眠模式偵測觸摸的最長掃描時間為 192ms。
```

注意：

1. *Sample Time、Frame Rate、Debounce、Slow-Green Mode 和 Scan Count(Q-Touch Option) 皆會影響觸摸鍵的反應時間及耗電流。*
2. *觸摸鍵的 Detect Time 時間，會因為 Q-Touch 選項中 Scan Count 的設置而使得掃描時間變為 1~4 倍，按鍵反應會比較慢，但按鍵準確率會提高。*

Detection：使用者可以選擇觸摸鍵的偵測方式，共有 2 種偵測方式。Single-Key 為同時間只能接受單一觸摸鍵被按下，Multi-Key 為可同時間多個觸摸鍵被按下。（預設值 Multi-Key）

```
例. TouchKey_Detection = Multi-Key      ; 可同時接受多個按鍵被觸摸。
例. TouchKey_Detection = Single-Key     ; 可同時接受一個按鍵被觸摸。
```

Wakeup：使用者可以選擇只用 PA0 來喚醒或是全部的觸摸鍵皆可喚醒。（預設值 AnyKey）

```
例. TouchKey_Wakeup = PA0       ; 觸摸鍵只有 PA0 可以喚醒 IC。
例. TouchKey_Wakeup = AnyKey    ; 所有的觸摸鍵皆可喚醒 IC。
```

注意：

1. *NY9T001A 無此設定。*
2. *使用單鍵（PA0）喚醒可以獲得更佳的省電效能。*
3. *需要使用 TouchKey_Scan_Slow 指令開啟觸摸鍵慢速掃描功能，PA0 Wakeup 才會生效。*

AutoJudge_Calibrate：觸摸鍵自動環境校正。使用觸摸鍵時，建議經過一段時間後要進行觸摸鍵校正。系統提供自動校正，並可設定多久作一次自動校正（時間 0.5 ~ 60 秒，預設 4 秒）。使用者可以將自動校正功能關閉，並自行利用時間路徑來計數更長的時間。

; 觸摸鍵每 4 秒自動校正一次。

例. **AutoJudge_Calibrate** = 4s

; 關閉觸摸鍵自動校正功能。

; 關閉 4 秒自動校正功能。(可使用程式計算來決定校正的時間，請參考 4.2.8)

例. **AutoJudge_Calibrate** = Disable

注意：觸摸鍵功能被關閉後，則自動校正功能也會被關閉。

Enforce_Calibrate：強制進行觸摸鍵環境校正。為了避免觸摸鍵非人為按下，而導致 IC 非預期性的工作，建議經過一段時間後要進行強制觸摸鍵校正，若觸摸鍵被誤觸，則因強制校正將此誤觸視為環境參數，可避免 IC 無法進入睡眠而持續耗電。若不使用系統提供的設定時間可自行將強迫校正功能關閉，由使用者利用時間路徑來計時，以進行校正。(時間 4 ~ 120 秒，預設 120 秒)

例. **Enforce_Calibrate** = 60s ; 觸摸鍵每 60 秒自動校正一次。

例. **Enforce_Calibrate** = Disable ; 關閉觸摸鍵自動校正功能。

*注意：當觸摸鍵被按住時，**Enforce_Calibrate** 計時器才會開始計數，按鍵放開則會重置計時器。*

SW_Reset：在執行觸摸鍵的強制校正後，是否進行重置 RAM 及 IO 狀態。(預設值 Enable)

例. **SW_Reset** = Enable ; 觸摸鍵強制校正後，進行重置。

例. **SW_Reset** = Disable ; 觸摸鍵強制校正後，不進行重置。

*注意：使用者如需要記憶設定或操作模式，可將 **SW_Reset** 功能關閉，改成在 **Enforce_Calibrate** 路徑中來進行手動重置。請參考 **4.6.1** 強制校正路徑的範例。*

Hysteresis Level：觸摸鍵遲滯功能。可以調整觸摸鍵按下後的靈敏度，避免摸到觸摸點邊緣部分時，可能會造成觸摸鍵處於不穩定狀態。(預設值 Small)

例. **TouchKey_Hysteresis_Level** = Disable ; 觸摸鍵觸發後，靈敏度不自動調整。

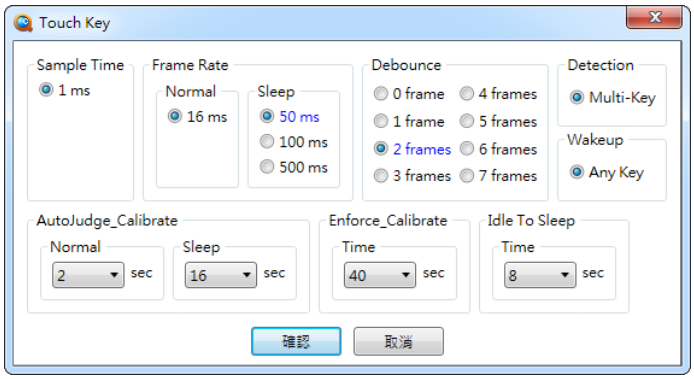
例. **TouchKey_Hysteresis_Level** = Small ; 觸摸鍵觸發後，該觸摸鍵靈敏度提高一階。

例. **TouchKey_Hysteresis_Level** = Large ; 觸摸鍵觸發後，該觸摸鍵靈敏度增強 50%。

注意：

1. *通常觸摸部位離 IC 的觸摸腳位較遠或是觸摸鍵較為敏感時，會需要開啟該功能。使用者要實際測試產品，比較容易決定要使用哪一個設定。*
2. ***NY9T004A** 不支援 **Hysteresis Level** 功能。*

3.1.7.2 NX1



Sample Time：掃描一個觸摸鍵的時間。較長的取樣時間觸摸更精確，但偵測到 TouchKey 被觸摸的時間會較久。僅支援 1ms。

Frame Rate：在一個 Frame 要取樣的數量，即完成所有觸摸鍵的掃描時間。取樣的數量愈多則較不耗電，但偵測到 TouchKey 被觸摸的時間會較久。

例. **TouchKey_FrameRate_Sleep** = 50ms

Debounce：連續幾個 Frame 被偵測到，按鍵才會成立。愈多 Frame 觸摸更精確，但偵測到 TouchKey 被觸摸的時間會愈久，對於較容易有外部干擾的環境建議選擇多些 Frame。

例. **TouchKey_Debounce** = 2

Detection：觸摸鍵的偵測方式， Multi-Key 為可同時間多個觸摸鍵被按下。

Wakeup：Any-Key 全部的觸摸鍵皆可喚醒。

AutoJudge_Calibrate：觸摸鍵自動環境校正。使用觸摸鍵時，建議經過一段時間後要進行觸摸鍵校正。系統提供自動校正，並可設定多久作一次自動校正。Normal Mode 可設定每隔 1 ~ 8 秒執行一次校正（預設值 2 秒）。Sleep Mode 可設定每 8 ~ 60 秒執行一次校正（預設值 16 秒）

例. **TouchKey_AutoJudge_Calibrate** = 4s ；**觸摸鍵每 4 秒自動校正一次。**

例. **TouchKey_AutoJudge_Calibrate_Sleep** = 16s

注意：觸摸鍵功能被關閉後，則自動校正功能也會被關閉。

Enforce_Calibrate：強制進行觸摸鍵環境校正。為了避免觸摸鍵非人為按下，而導致 IC 非預期性的工作，建議經過一段時間後要進行強制觸摸鍵校正，若觸摸鍵被誤觸，則因強制校正將此誤觸視為環境參數，可避免 IC 無法進入睡眠而持續耗電。若不使用系統提供的設定時間可自行將強迫校正功能關閉，由使用者利用時間路徑來計時，以進行校正。（時間 4 ~ 120 秒，預設值 40 秒）

例. **TouchKey_Enforce_Calibrate** = 60s ；**觸摸鍵每 60 秒自動校正一次。**

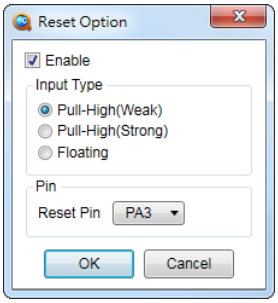
注意：當觸摸鍵被按住時，Enforce_Calibrate 計時器才會開始計數，按鍵放開則會重置計時器。

Idle To Sleep：進入休眠模式的時間。可以調整持續多久沒有被觸發，會進入休眠模式。（時間 1 ~ 16 秒，預設值 8 秒）

例. **TouchKey_ IdleToSleep** = 7s ；**持續 7s 未觸發，會進入休眠。**

3.1.8 Reset

3.1.8.1 NY4



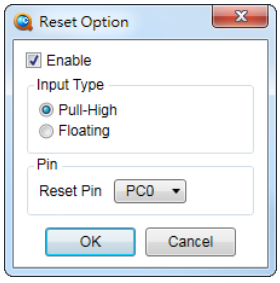
外部強制復位 IC 的腳位。Reset pin 可選擇有 Pull-High 電阻 Weak 或是 Strong 或是 Floating。若使用者選擇 Floating，則需要外接 1M 電阻（預設值 Disable）。Reset 為固定腳位

例. **Reset = P:W ; Pull-High Weak Resistor**

例. **Reset = P:S ; Pull-High Strong Resistor**

例. **Reset = Floating**

3.1.8.2 NY5 / NY6 / NY7



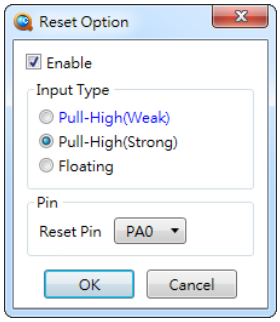
外部強制復位 IC 的腳位。Reset pin 可選擇有 Pull-High 電阻或是 Floating。若使用者選擇 Floating，則需要外接 1M 電阻。

例. **Reset = Pull-High**

NY5 可以選擇 Reset pin 的腳位，NY6 / NY7 的 Reset pin 為固定腳位。

例. **Reset_Pin = PC.0**

3.1.8.3 NY5+

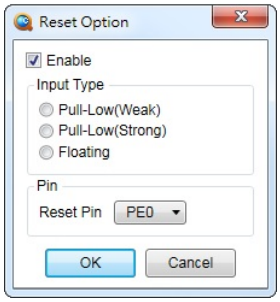


外部強制復位 IC 的腳位。Reset pin 可選擇有 Pull-High 電阻 Weak 或是 Strong 或是 Floating。若使用者選擇 Floating，則需要外接 1M 電阻（預設 Disable）。可以選擇 Reset pin 的腳位。

例. **Reset = Floating**

例. **Reset_Pin = PA.0**

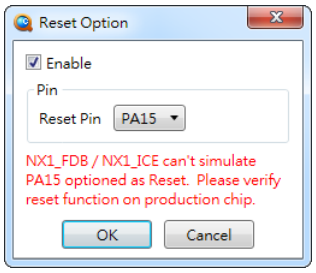
3.1.8.4 NY9T



外部強制復位 IC 的腳位。Reset pin 可選擇有 Pull-Low 電阻或是 Floating。若使用者選擇 Floating，則需要外接 1M 電阻。

例. **Reset = F ; Floating**

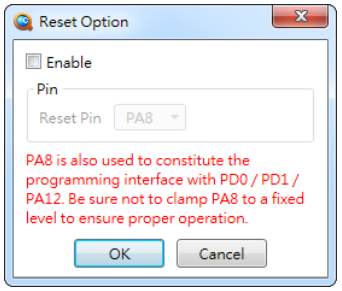
3.1.8.5 NX1 OTP



外部強制復位 IC 的腳位。

例. **Reset_Pin** = PA.15

3.1.8.6 NX1 EF



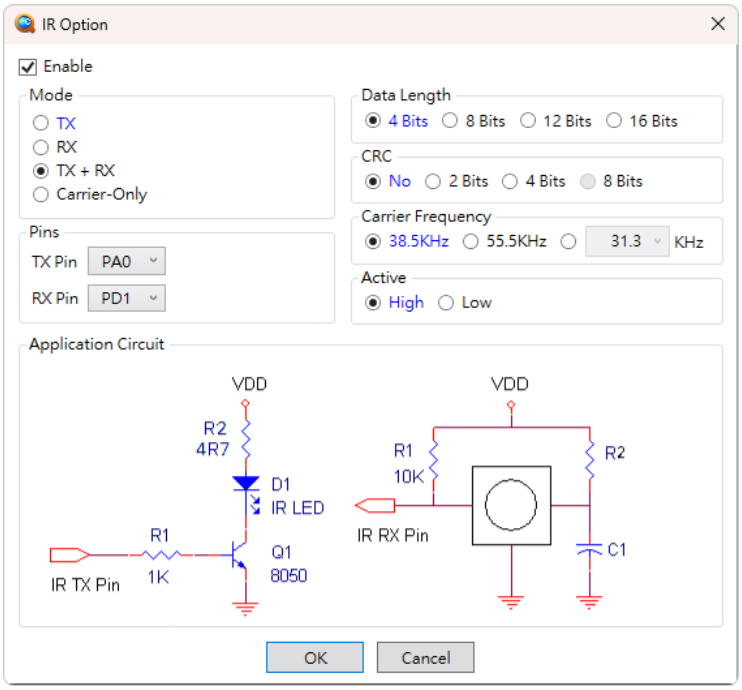
外部強制復位 IC 的腳位。

例. **Reset_Pin** = PA.15

3.1.9 IR

設定紅外線發射及接收功能。系統已預設特定的 I/O 作為發射或接收腳，但需用戶指定 IR 編碼資料的長度與 IR 載波頻率。

3.1.9.1 NY4 / NY5 / NY6 / NY7



Mode：選擇 IR 模式 TX / RX or Carrier-Only。

例. **IR_Mode** = TX + RX

例. **IR_Mode** = Carrier_Only

TX Pin：選擇 TX 腳位。只有 NY5 可以選擇腳位。

例. **IR_TX** = PA.1

RX Pin：選擇 RX 腳位。

例. **IR_RX** = PA.2

Data Length：選擇 IR 編碼長度。

例. **IR_Bit** = 8

CRC：選擇 IR 檢查碼的長度，預設為關閉檢查碼功能。

例. **IR_CRC** = 2

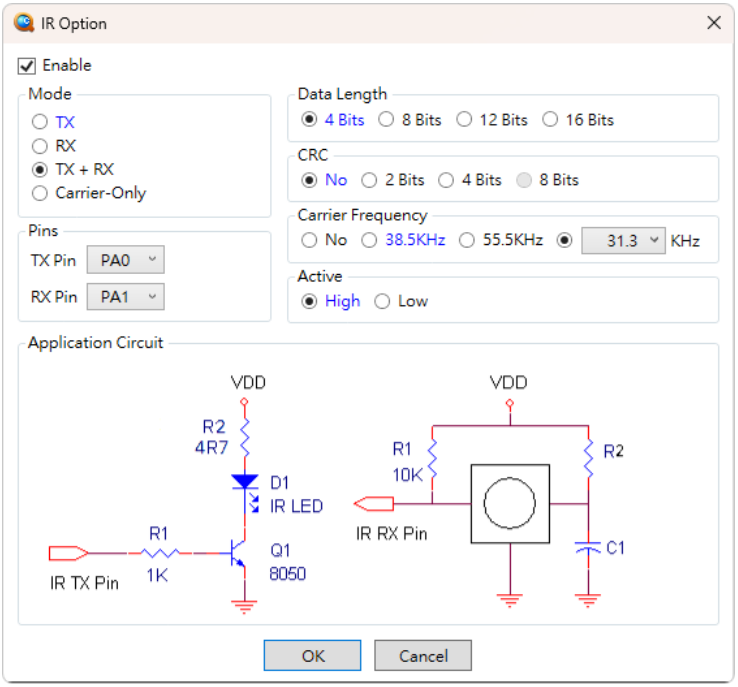
Carrier Frequency：選擇 IR 載波頻率。

例. **IR_Carrier_Freq** = 38.5K

Active：選擇由 NPN 或者 PNP 電晶體來驅動 IR LED。

例. **IR_Active** = High

3.1.9.2 NY5+



Mode：選擇 IR 模式 TX / RX or Carrier-Only。

例. **IR_Mode** = TX

例. **IR_Mode** = TX + RX

例. **IR_Mode** = Carrier_Only

TX Pin：選擇 TX 腳位。

例. **IR_TX** = PA.1

RX Pin：選擇 RX 腳位。

例. **IR_RX** = PA.2

Data Length：選擇 IR 編碼長度。

例. **IR_Bit** = 8

CRC：選擇 IR 檢查碼的長度，預設為關閉檢查碼功能。

例. **IR_CRC** = 2

Carrier Frequency：選擇 IR 載波頻率。

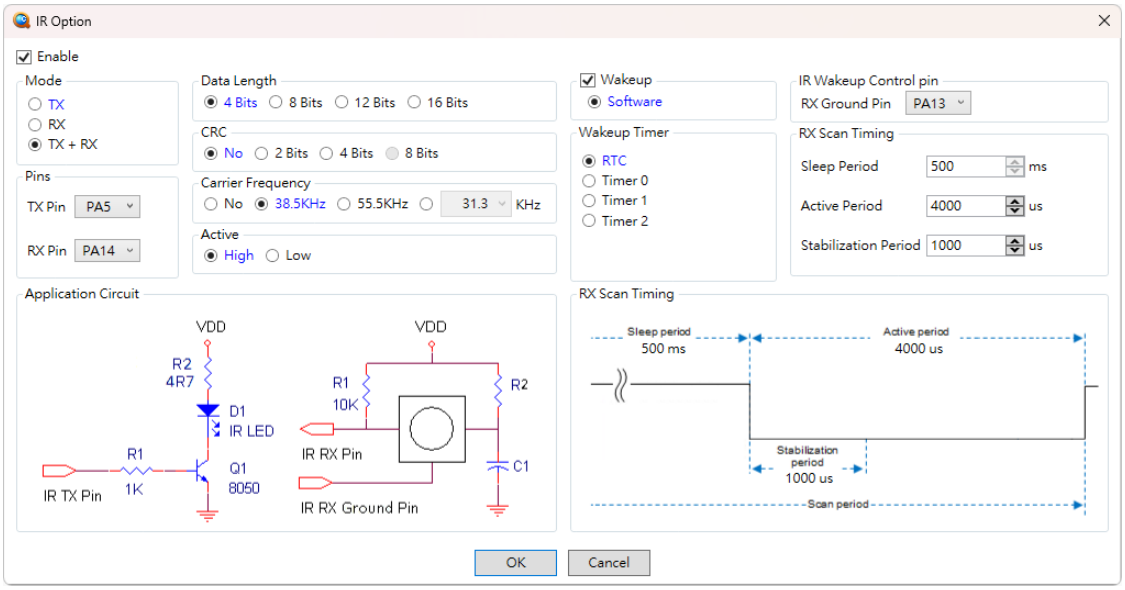
例. **IR_Carrier_Freq** = 31.3K

例. **IR_Carrier_Freq** = **Disable**

Active：選擇由 NPN 或者 PNP 電晶體來驅動 IR LED。

例. **IR_Active** = High

3.1.9.3 NX1 OTP



Mode：選擇開啟 TX or RX 功能。

例. **IR_Mode** = TX

例. **IR_Mode** = TX + RX

TX Pin：選擇 TX 腳位。

例. **IR_TX** = PA.1

RX Pin：選擇 RX 腳位。

例. **IR_RX** = PA.2

Data Length：選擇 IR 編碼長度。

例. **IR_Bit** = 8

CRC：選擇 IR 檢查碼的長度，預設為關閉檢查碼功能。

例. **IR_CRC** = 2

Carrier Frequency：選擇 IR 載波頻率。

例. **IR_Carrier_Freq** = 38.5K

例. **IR_Carrier_Freq** = **Disable**

Active：選擇由 NPN 或者 PNP 電晶體來驅動 IR LED。

例. **IR_Active** = High

Wakeup：選擇開啟 IR Wakeup 功能。

例. **IR_Wakeup** = Enable

IR_Wakeup_Type = SW

Wakeup Timer：選擇 IR Wakeup 使用的 Timer。

例. **IR_Wakeup_Timer** = RTC

IR Wakeup Control Pin：選擇 RX Ground 腳位。

例. **IR_Wakeup_Pin** = PA.3

Sleep Period (RX Scan Timing)：選擇 RX Sleep Period。

例. **IR_Wakeup_Sleep_Period** = 500ms

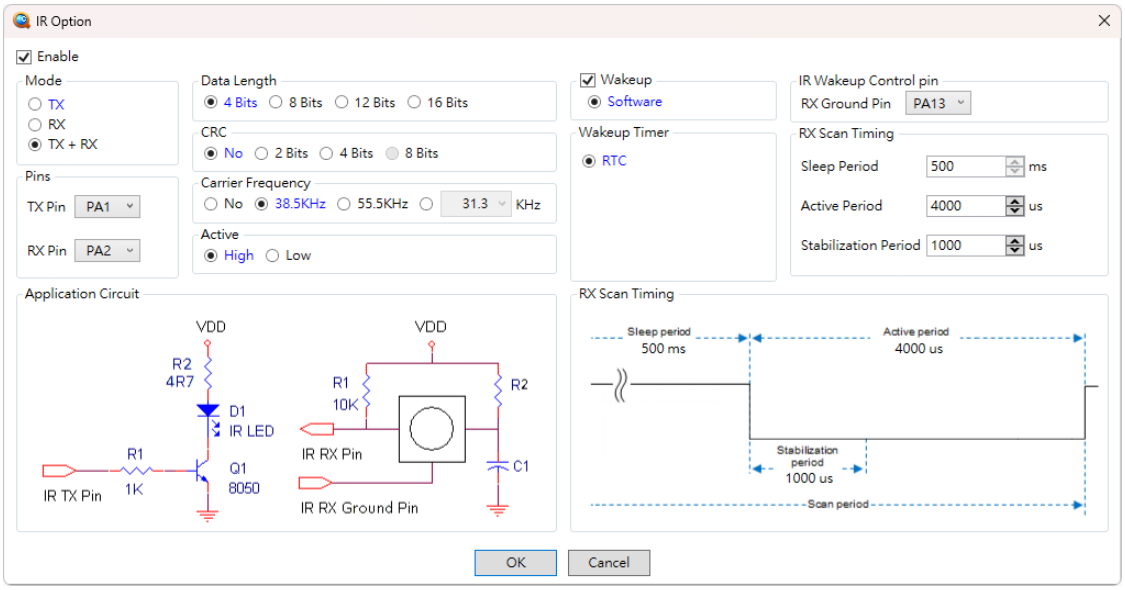
Active Period (RX Scan Timing)：選擇 RX Active Period。

例. **IR_Wakeup_Active_Period** = 1000us

Stabilization Period (RX Scan Timing)：選擇 RX Stabilization Period。

例. **IR_Wakeup_Stabilization_Period** = 4000us

3.1.9.4 NX1 EF



Mode：選擇開啟 TX or RX 功能。

例. **IR_Mode** = TX

例. **IR_Mode** = RX

例. **IR_Mode** = TX + RX

TX Pin：選擇 TX 腳位。

例. **IR_TX** = PA.1

RX Pin：選擇 RX 腳位。

例. **IR_RX** = PA.3

Data Length：選擇 IR 編碼長度。

例. **IR_Bit** = 8

CRC：選擇 IR 檢查碼的長度，預設為關閉檢查碼功能。

例. **IR_CRC** = 2

Carrier Frequency：選擇 IR 載波頻率。

例. **IR_Carrier_Freq** = 38.5K

例. **IR_Carrier_Freq** = 52.6K

例. **IR_Carrier_Freq** = **Disable**

Active：選擇由 NPN 或者 PNP 電晶體來驅動 IR LED。

例. **IR_Active** = High

例. **IR_Active** = Low

Wakeup：選擇開啟 SW / HW IR Wakeup 功能。

例. **IR_Wakeup** = **Enable**

IR_Wakeup_Type = HW

注意：NX11FS20A / NX11FS21A / NX11FS22B 才支援 HW IR Wakeup。

Wakeup Timer：選擇 IR RX Wakeup 使用的 Timer。

例. **IR_Wakeup_Timer** = RTC

例. **IR_Wakeup_Timer** = Timer1

注意：

1. NX11FS20A / NX11FS21A / NX11FS22B 才支援 Timer0 和 Timer1。

2. HW IR Wakeup 固定使用 PWMA。

IR Wakeup Control Pin：選擇 RX Ground 腳位。

例. **IR_Wakeup_Pin** = PA.3

Sleep Period (RX Scan Timing)：選擇 RX Sleep Period。

例. **IR_Wakeup_Sleep_Period** = 500ms

Active Period (RX Scan Timing)：選擇 RX Active Period。

例. **IR_Wakeup_Active_Period** = 4000us

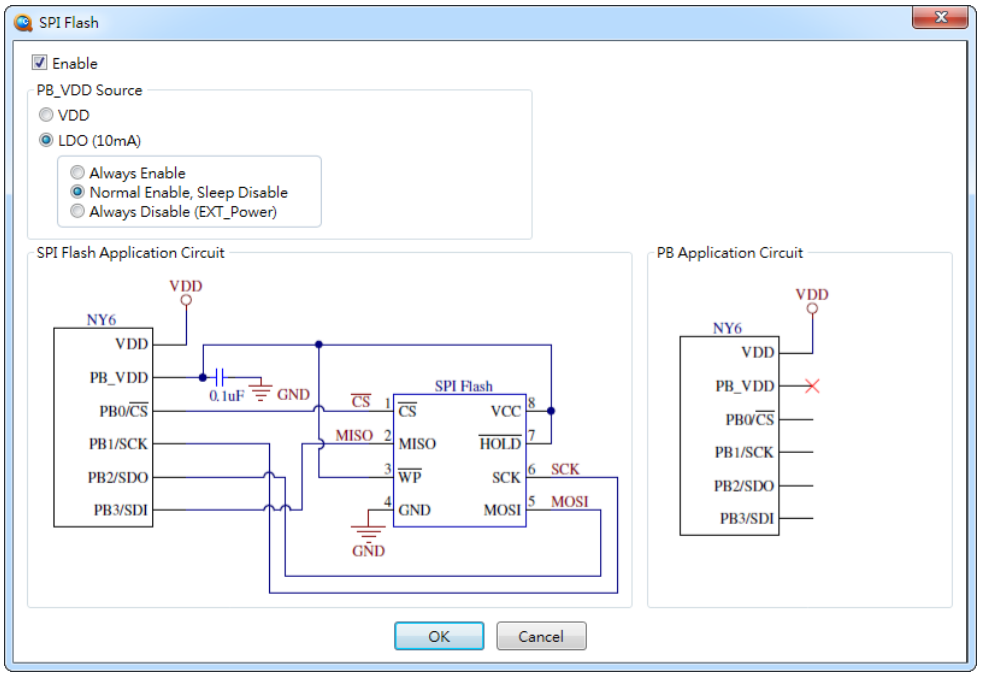
Stabilization Period (RX Scan Timing)：選擇 RX Stabilization Period。

例. **IR_Wakeup_Stabilization_Period** = 1000us

3.1.10 SPI Flash

串列匯流排介面（Serial Peripheral Interface），透過此 SPI 視窗配合 *Voice_Encoder* V3.30 或之後的版本，進行播放儲存於外面 SPI Flash 的 Voice。亦可透過 SPI 傳輸介面以 Master Mode 和其他 SPI 的裝置進行通訊。

3.1.10.1 NY6



PB_VDD Source：PB 電源之來源。

1. VDD：PB 電源由 VDD 提供。（Real chip 需自行將 PB_VDD 綁定到 VDD）
2. LDO Always Enable：PB 電源由 IC 內部提供。
3. LDO Normal Enable, Sleep Disable：PB 電源由 IC 內部提供，當 IC 進入休眠時，LDO 關閉。
4. LDO Always Disable (EXT_Power)：PB 電源不由 VDD 提供，也不由 LDO 提供，PB_VDD 可接其他外部電源，若外接電源大於 IC VDD，會有漏電的情形發生。

注意：

1. 當 PB 作為 SPI Flash 應用，並使用 IC 內部提供電源時，PB_VDD 需要外加一顆 0.1uF 電容。

2. 固定使用 PB 所有的腳位作為 SPI 溝通的腳位，各腳位對應如下：

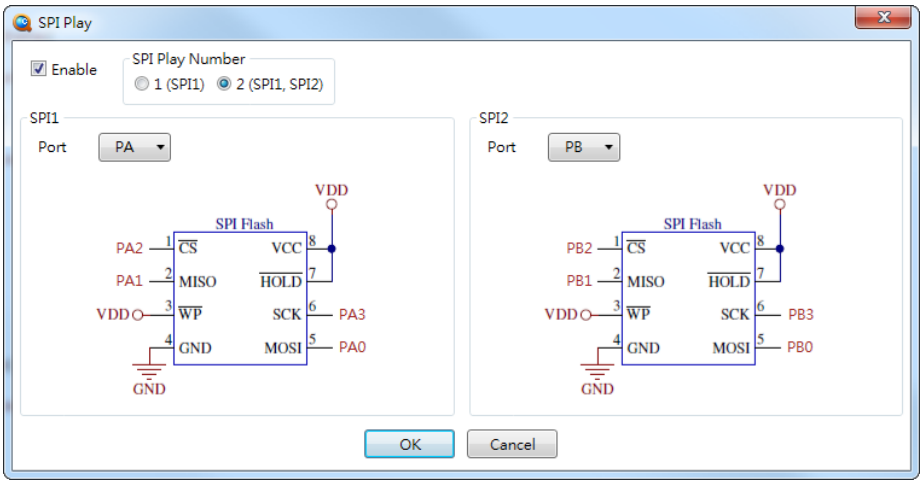
CSB：PB.0 SCK：PB.1 MOSI：PB.2 MISO：PB.3

3. PB 當一般 IO 使用，不當 SPI 使用時，PB 的電源亦可選擇。

例. **SPI** = Enable

PB_VDD = VDD

3.1.10.2 NY7



SPI Paly Number：支援二組 SPI 設定。

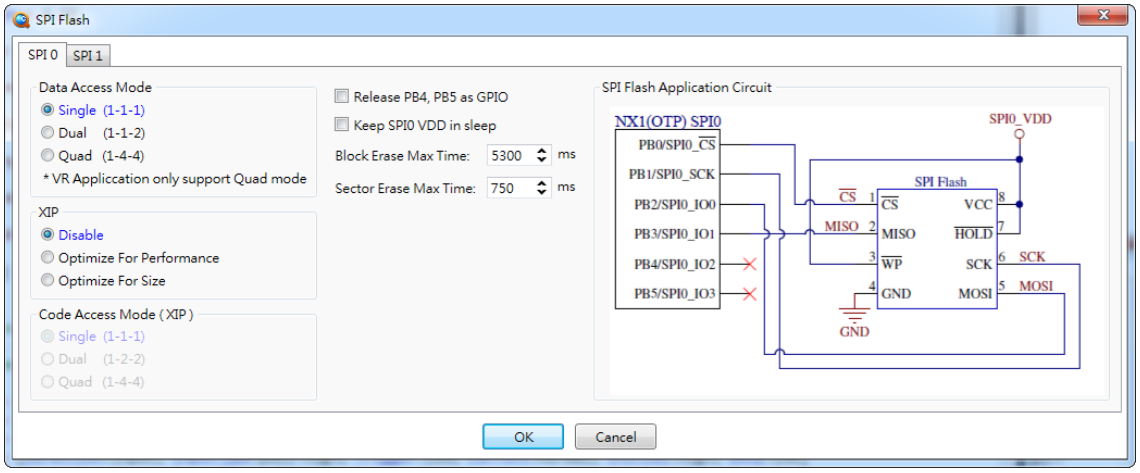
注意：目前只支援 SPI Dual Mode，固定使用完整的 port 作為 SPI 溝通的腳位，各腳位對應如下：

MOSI：Px.0 MISO：Px.1 CSB：Px.2 CLK：Px.3

例. **SPI1 = PA**

3.1.10.3 NX1 OTP

- **SPI 0**



Data Access Mode：可選擇 Single / Dual / Quad。

例. **SPI0_Data_Access_Mode** = Quad

XIP：可選擇 Disable / Optimze For Performance / Optimze For Size。

1. Optimze For Performance 可將部份使用者程序放到 SPI Flash。
2. Optimze For Size 可將更多程序放到 SPI Flash，但 CPU 負載相對會上升。

例. **XIP** = Speed_Optimize

Code Access Mode：可選擇 Disable / Single / Dual。

例. **SPI0_Code_Access_Mode** = Single

注意：使用 XIP 建議選擇 Quad，執行速度會比 Single 和 Dual 快一些，CPU 負載可相對下降一些。

Release PB4, PB5 as GPIO：當 Data Access Mode 與 Code Access Mode 不選用 Quad 時，將 PB.4 及 PB.5 開放為 GPIO 使用。

例. **Release_PB4_PB5** = Disable

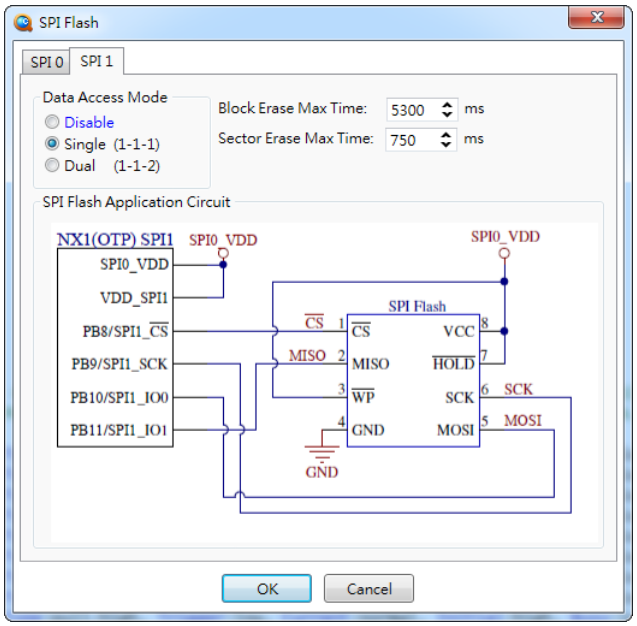
Block Erase Max Time：使用 Block Erase 擦除指令發生 Flash 等待逾時，將停止等待並立刻返回。建議依照 Flash 規格進行合適的時間設定。

例. **SPI0_BE_MaxTime** = 5300ms

Sector Erase Max Time：使用 Sector Erase 擦除指令發生 Flash 等待逾時，將停止等待並立刻返回。建議依照 Flash 規格進行合適的時間設定。

例. **SPI0_SE_MaxTime** = 750ms

- **SPI 1**



Data Access Mode：可選擇 Disbale / Single / Dual。

例. **SPI1_Data_Access_Mode** = Single

Block Erase Max Time：使用 Block Erase 擦除指令發生 Flash 等待逾時，將停止等待並立刻返回。建議依照 Flash 規格進行合適的時間設定。

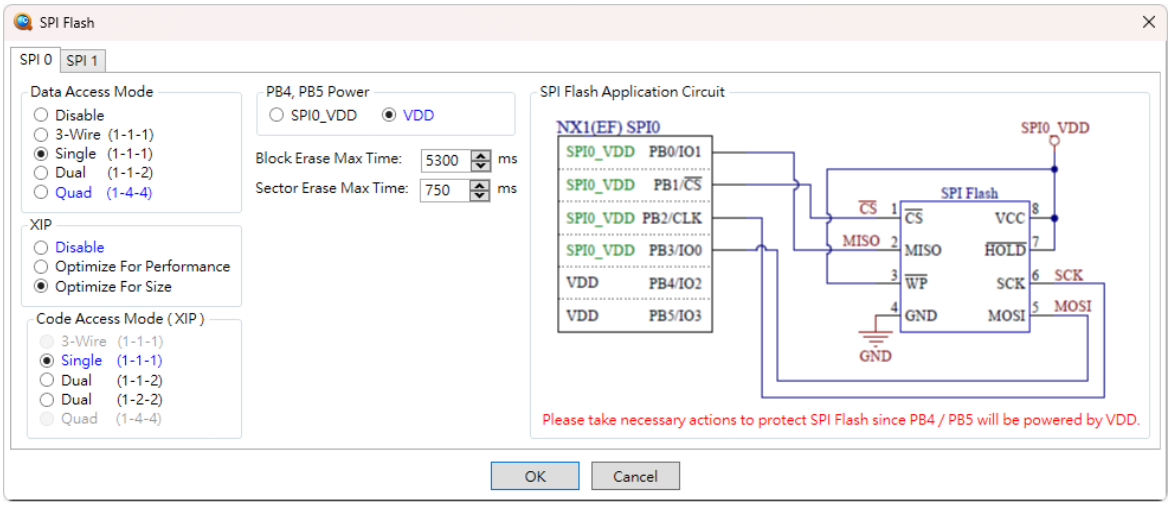
例. **SPI1_BE_MaxTime** = 5300ms

Sector Erase Max Time：使用 Sector Erase 擦除指令發生 Flash 等待逾時，將停止等待並立刻返回。建議依照 Flash 規格進行合適的時間設定。

例. **SPI1_SE_MaxTime** = 750ms

3.1.10.4 NX1 EF

- **SPI 0**



Data Access Mode：可選擇 Disable / 3-Wire / Single / Dual / Quad。

例. **SPI0_Data_Access_Mode = Quad_144**

例. **SPI0_Data_Access_Mode = Single_3Wire**

例. **SPI0_Data_Access_Mode = Single**

XIP：可選擇 Disable / Optimze For Performance / Optimze For Size

1. Optimze For Performance 可將部份使用者程序放到 SPI0 Flash。
2. Optimze For Size 可將更多程序放到 SPI0 Flash，但 CPU 負載相對會上升。

例. **XIP** = Speed_Optimize

Code Access Mode：可選擇 3-Wire / Single / Dual (1-1-2) / Dual (1-2-2) / Quad。

例. **SPI0_Code_Access_Mode = Single**

注意：使用 XIP 建議選擇 Quad，執行速度會比 Single 和 Dual 快一些，CPU 負載可相對下降一些。

PB4, PB5 Power：當 Data Access Mode 與 Code Access Mode 選擇 3-Wire / Single / Dual 時，將可以設定 PB4, PB5 的電源。

例. **PB45_Power_Source** = SPI0_VDD

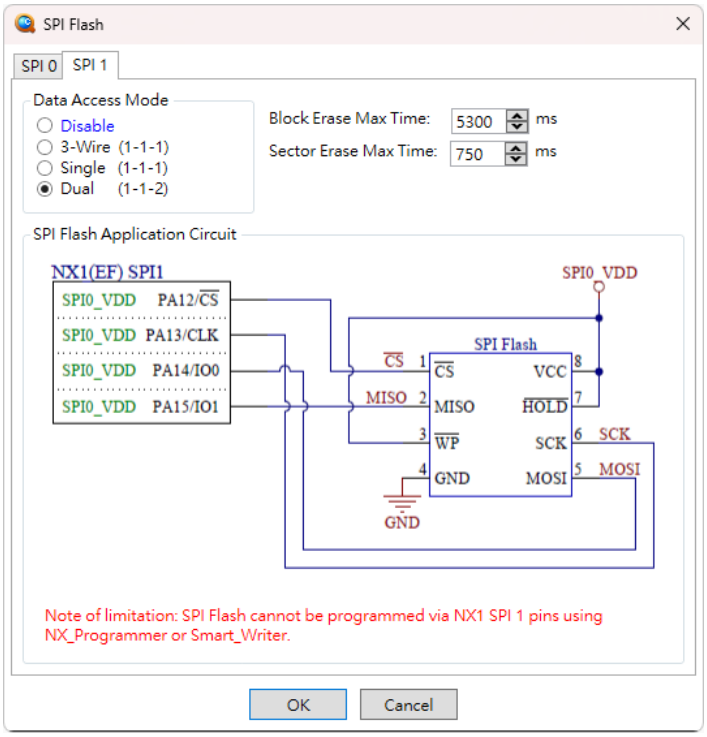
Block Erase Max Time：當 Block Erase 指令發生 Flash 等待逾時，將停止等待並立刻返回。建議依照 Flash 規格進行合適的時間設定。

例. **SPI0_BE_MaxTime** = 5300ms

Sector Erase Max Time：當 Sector Erase 指令發生 Flash 等待逾時，將停止等待並立刻返回。建議依照 Flash 規格進行合適的時間設定。

例. **SPI0_SE_MaxTime** = 750ms

- **SPI 1**



Data Access Mode：可選擇 Disable / 3-Wire / Single / Dual。

例. **SPI1_Data_Access_Mode = Disable**

例. **SPI1_Data_Access_Mode = Single_3Wire**

例. **SPI1_Data_Access_Mode = Single**

例. **SPI1_Data_Access_Mode = Dual_112**

Block Erase Max Time：當 Block Erase 指令發生 Flash 等待逾時，將停止等待並立刻返回。建議依照 Flash 規格進行合適的時間設定。

例. **SPI1_BE_MaxTime** = 5300ms

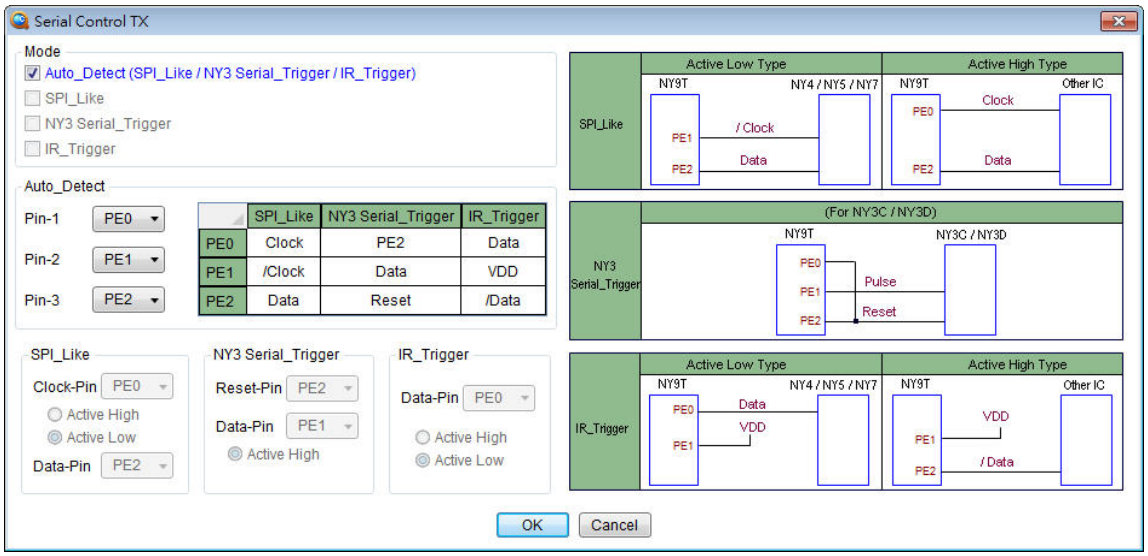
Sector Erase Max Time：當 Sector Erase 指令發生 Flash 等待逾時，將停止等待並立刻返回。建議依照 Flash 規格進行合適的時間設定。

例. **SPI1_SE_MaxTime** = 750ms

3.1.11 Serial Control TX

3.1.11.1 NY9T

可連接一般微控制器，將 NY9T 當成微控制器的按鍵。提供 3 種不同的傳輸協定，分別是 SPI_Like、NY3 Serial_Trigger、IR_Trigger。使用者可以設定成自動偵測或是特定的傳輸協定，通訊時序圖請參考串列訊號控制通訊協定。



Auto Detect：可由 3 根腳位的設定方式來決定，NY9T 要使用何種傳輸協定。自動偵測的腳位設定方式如下表所示：

自動模式偵測			
Mode	**Pin-3**	**Pin-2**	**Pin-1**
IR_Trigger	X	VDD	X
NY3 Serial_Trigger	Pin-3 connected to Pin-1	X	Pin-3 connected to Pin-1
SPI_Like	No Connected	No Connected	No Connected

例. **SerialControl** = Auto

SerialControl_Pin1 = PF.1

SerialControl_Pin2 = PF.2

SerialControl_Pin3 = PF.3 ；**自動偵測傳輸協定。**

注意：Auto 模式時，3 根腳位都必須在同一個埠。

SPI_Like：使用 2 根腳位，達成類似 SPI 傳輸協定。

例. **SerialControl** = SPI_Like

SerialControl_SPI_Clock = PF.2

SerialControl_SPI_Data = PF.3

SerialControl_SPI_ClockTrigger = High or Low ；**傳輸協定為正緣觸發或是負緣觸發。**

注意：SPI_Like 模式時，2 根腳位都必須在同一個埠。

NY3 Serial_Trigger：使用者可以透過串列訊號與 NY3 連線，將 NY9T 當成 NY3 的觸發按鍵。

例. **SerialControl** = NY3

SerialControl_NY3_Reset = PF.3

SerialControl_NY3_Data = PF.2

IR_Trigger：使用 1 個腳位來模擬 IR 的接收訊號，可直接使用目前的 NY4/5/7 系列的 IR 通訊。

例. **SerialControl** = IR_Trigger

SerialControl_IR_Data = PF.3

SerialControl_IR_DataBusy = High or Low ；傳輸協定為正緣觸發或是負緣觸發。

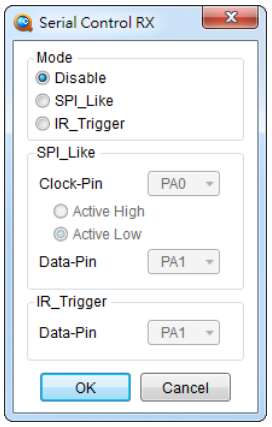
注意：

1. *同一時間只能有一種傳輸協定。*
2. *開啟 Auto-Detect 功能時，PIN-1 會被設成 Input Floating，PIN-2/PIN-3 會被設成 Input Weak-Pull-Low。*
3. *開啟 Auto-Detect 功能時，接收端的 IC，必須將連接腳設定成 Input floating。*
4. *NY9T001A/NY9T004A 不支援 Serial Control 功能。*

3.1.12 Serial Control RX

3.1.12.1 NY4/ NY5 / NY5+ / NY6 / NY7

接收外部送出的串列資料。可使用 SPI_Like / IR_Trigger 兩種傳輸協定，通訊時序圖請參考串列訊號控制通訊協定。



SPI_Like：使用 2 根腳位，類似 SPI 傳輸協定。

例. **Serial_Rx** = SPI_Like

Serial_Rx_Data = PA.1

Serial_Rx_Clock = PA.0

Serial_Rx_ClockTrigger = High

IR_Trigger：使用 1 根腳位，使用 IR 協定。

例. **Serial_Rx** = IR_Trigger

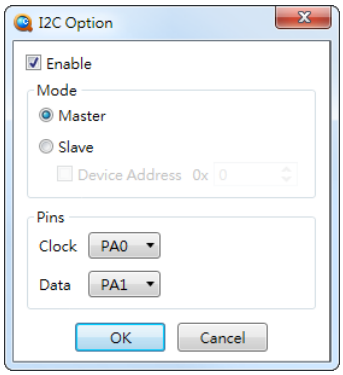
Serial_Rx_Data = PF.1

注意：

1. *NY5+ / NY6 / NY7 使用 SPI_Like 時，Clock / Data pin 必需在相同的 port。*
2. *IR_Trigger 接收模式不能與 IR_RX 一起使用。*

3.1.13 I2C

3.1.13.1 NY5+



Mode：可以選擇 Master 模式或 Slave 模式。傳輸速度為 250Hz (傳輸一筆資料約 36ms)。

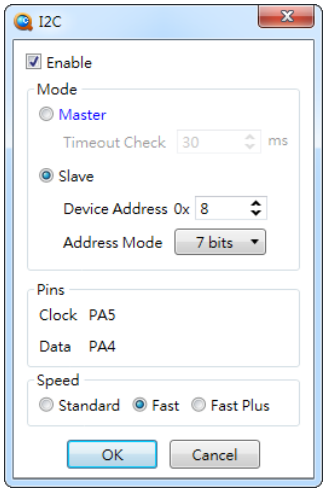
Device Address：作為 Slave 端時的 Address，可為 0 ~ 127。

Clock：選擇 Clock(SCL) 腳位。

Data：選擇 Data(SDA) 腳位。

注意：Clock 與 Data 腳位必須為相同的 Port。

3.1.13.2 NX1



Mode：可以選擇 Master 模式或 Slave 模式。傳輸速度可支援 Standard (100kHz) / Fast (400kHz) / Fast+ (1MHz)。

Timeout Check：作為 Master 端時，間隔多久無回應即視為逾時，1 ~ 30ms，設 0 則永不逾時。

Device Address：作為 Slave 端時的 Address。

Device Address Bit：做為 Slave 端時，指定 Address 為 7 / 10 bits。

Clock：Clock(SCL) 腳位。

Data：Data(SDA) 腳位。

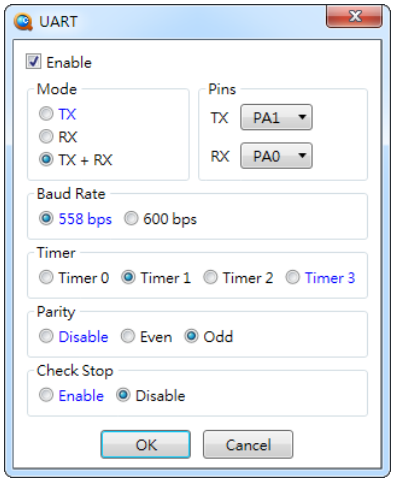
Speed：傳輸速度，支援 Standard / Fast / Fast Plus，預設為 Fast。

注意：作為 Slave 端時，Master 端需支援 Clock Stretching，否則行為會異常。

3.1.14 UART

Q-Code 支援 UART(Universal Asynchronous Receiver / Transmitter)通訊協定。

3.1.14.1 NY5+



Mode：可選擇 TX / RX / TX+RX。

例. **UART_Mode** = TX

例. **UART_Mode** = RX

例. **UART_Mode** = TX + RX

Pins：UART 的 TX 和 RX pin。

例. **UART_TX** = PA.1

例. **UART_RX** = PA.0

Baud Rate：UART 的 Baud Rate，支援 558bps / 600bps。

例. **UART_BaudRate** = 558bps

Timer：用來產生所選擇 Baud Rate 的計時器。

例. **UART_Timer_Channel** = 1

Parity：是否進行奇偶校驗，可選擇 Disable / Even / Odd。

例. **UART_ Parity** = Odd

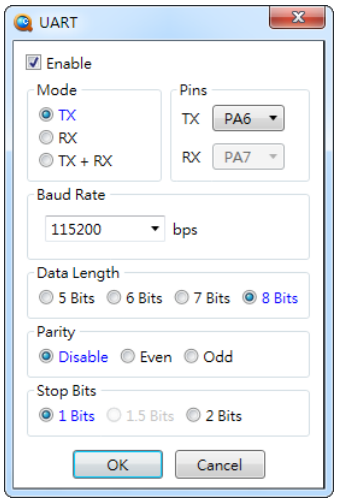
Check Stop：是否檢查 StopBit。

例. **UART _Check_Stop** = **Disable**

注意：

1. *TX 與 RX 腳位必須為相同的 Port。*
2. *UART 使用時，所選擇的計時器無法進行 Voice 播放或 Timer 應用。*

3.1.14.2 NX1



Mode：可選擇 TX / RX / TX+RX。

例. **UART_Mode** = TX

例. **UART_Mode** = RX

例. **UART_Mode** = TX + RX

Pins：UART 的 TX 和 RX pin。

例. **UART_TX** = PA.6

例. **UART_RX** = PA.7

注意：

1. ***NX1 EF 系列 同時使用 TX 與 RX 時，必須為相同的 Port。***
2. ***NX1 EF 系列 TX 僅支援 PA.6 / PD.0，若要使用 PD.0 作為 TX，必須先在 I/O 段落將 PD.0 設定為 GPIO。***
3. ***NX1 EF 系列 RX 僅支援 PA.7 / PD.1，若要使用 PD.1 作為 RX，必須先在 I/O 段落將 PD.1 設定為 GPIO。***

Baud Rate：UART 的 Baud Rate。

例. **UART_BaudRate** = 9600bps

Data Length：每筆傳輸資料的長度。

例. **UART_ Bit** = 7

Parity：是否進行奇偶校驗，可選擇 Disable / Even / Odd。

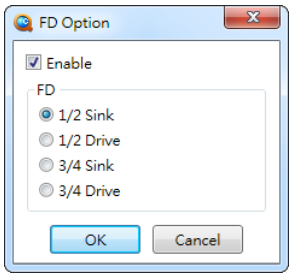
例. **UART_ Parity** = Odd

Stop Bits：UART 的 Stop Bit。

例. **UART _Stop_Bit** = 1

3.1.15 FD

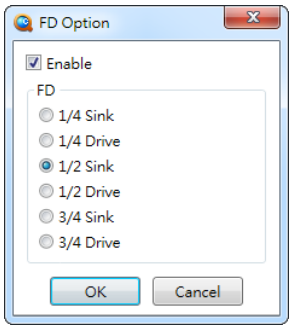
3.1.15.1 NY4 / NY5



Flash with Dynamic 隨著音量的大小來閃動。可以選擇大於 1 / 2 或者是 3 / 4 音量來閃動，並且可以設定輸出型態為 Drive 或是 Sink 的方式。(D 為 Drive，S 為 Sink。預 Disable)

例. **FD** = 1/2D

3.1.15.2 NY5+ / NY6 / NY7 / NX1

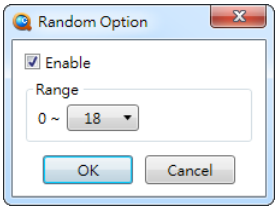


Flash with Dynamic 隨著音量的大小來閃動。可以選擇大於 1 / 4、1 / 2 或者是 3 / 4 音量來閃動，並且可以設定輸出型態為 Drive 或是 Sink 的方式。(D 為 Drive，S 為 Sink。預設值 Disable)

例. **FD** = 1/4D

3.1.16 Random

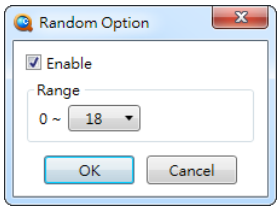
3.1.16.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T



Random 表示產生出來的隨機數範圍，預設值 Disable，可選 0 ~ 255 之間。

例. **Random** = 122

3.1.16.2 NX1

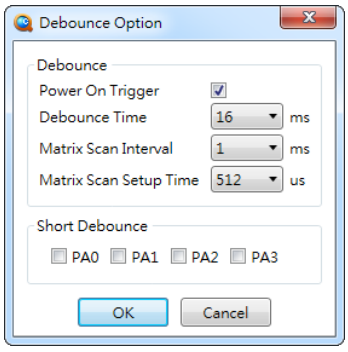


Random 表示產生出來的隨機數範圍，預設值 Disable，可選 0 ~ 65535 之間。

例. **Random** = 822

3.1.17 Debounce

3.1.17.1 NY4 / NY5



Power On Trigger：選擇是否要有 Power on Trigger 功能。當功能開啟後，如果使用者按著按鍵後開機，則開機後會馬上去執行該按鍵的路徑；反之，則不執行該按鍵路徑。（預設 Enable）

注意：如果開啟此功能，則 PowerOn 路徑必須設定 input state，否則會因為找不到設定路徑而無效。

例. **PowerOnTrigger** = Disable ； **關閉所有按鍵的上電觸發功能。**

Debounce Time：使用者可以選擇按鍵的 debounce 時間，時間範圍：1~1000ms（預設值 16ms）。

注意：Q-Code 實際上是採用輪詢（Polling）的方式。原則上，實際按鍵的時間必須大於設定的 debounce 時間。如需要很精準的 debounce 時間或不同的 debounce 時間，需用組合語言來編寫。

例. **Debounce** = 16 ms ； **全部按鍵的 Debounce 時間為 16ms。**

Matrix Scankey Interval：使用者可以選擇 Matrix Key 的掃描時間間隔，可支援 0 ~ 4ms（預設值 1ms）。

注意：Q-Code 實際上是採用輪詢（Polling）的方式處理此時間，所以實際時間會根據當時的負載輕重而有所誤差。

例. **ScankeyInterval** = 4ms ； **按鍵的掃描時間間隔為 4ms。**

Matrix Scan Setup Time：使用者可以選擇矩陣掃描的輸出設定時間；假設此時間設定為 256us，則

每支輸出腳設定為輸出低準位後，會延遲 256us 才抓取輸入腳的狀態，以避免輸入腳在下拉過程中，尚未穩定的情況下抓取狀態造成誤判。支援 0 / 512us / 1024us / 1536us（預設 512us）。

注意：Q-Code 實際上是採用輪詢（Polling）的方式處理此時間，所以實際時間會根據當時的負載輕重而有所誤差。

例. **Matrix_Scan_SetupTime** = 256us ; **設定矩陣掃描設定時間為 256us。**

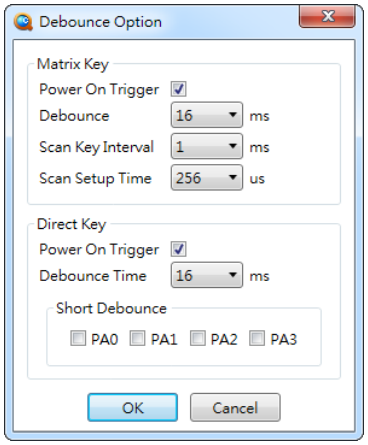
Short Debounce：使用者可以選擇哪一個獨立按鍵的 Short Debounce，時間範圍：0~1ms。

注意：

1. *此功能不支援 Matrix Key*
2. *Q-Code 實際上是採用輪詢（Polling）的方式，所以實際 debounce 時間會有所誤差。如需要很精準的 debounce 時間或不同的 debounce 時間，需用組合語言來編寫。*

例. **ShortDebounce** = PA.0, PA.1, PA.2 ; **PA.0, PA.1, PA.2 為 Short Debounce 功能。**

3.1.17.2 NY5+ / NY6 / NY7



Power On Trigger：選擇是否要有 Power on Trigger 功能。當功能開啟後，如果使用者按著按鍵後開機，則開機後會馬上去執行該按鍵的路徑；反之，則不執行該按鍵路徑。允許個別設定 Direct Key 及 Matrix Key 是否開啟此功能。（預設值 Enable）

注意：如果開啟此功能，則 PowerOn 路徑必須設定 input state，否則會因為找不到設定路徑而無效。

例. **Direct_PowerOnTrigger** = Disable ; **關閉 Direct Key 上電觸發功能。**

例. **Matrix_PowerOnTrigger** = Disable ; **關閉 Matrix Key 上電觸發功能。**

Debounce Time：使用者可以選擇按鍵的 debounce 時間，時間範圍：1~1000ms。允許 Direct Key 及 Matrix Key 設定不同的 debounce 時間。（預設值 16ms）。

注意：Q-Code 實際上是採用輪詢（Polling）的方式。原則上，實際按鍵的時間必須大於設定的 debounce 時間。如需要很精準的 debounce 時間或不同的 debounce 時間，需用組合語言來編寫。

例. **Direct_Debounce** = 16 ms ；**Direct Key 的 Debounce 時間為 16ms。**

例. **Matrix_Debounce** = 16 ms ；**Matrix Key 的 Debounce 時間為 16ms。**

Scankey Interval：使用者可以選擇 Matrix Key 掃描的時間間隔，可支援 0 ~ 4ms（預設值 1ms）。

*注意：**Q-Code 實際上是採用輪詢（Polling）的方式處理此時間，所以實際時間會根據當時的負載輕重而有所誤差。***

例. **ScankeyInterval** = 4ms ；**按鍵的掃描時間間隔為 4ms。**

Scan Setup Time：使用者可以選擇 Matrix Key 掃描的輸出設定時間；假設此時間設定為 256us，則每支輸出腳設定為輸出低準位後，會延遲 256us 才抓取輸入腳的狀態，以避免輸入腳在下拉過程中，尚未穩定的情況下抓取狀態造成誤判。0 / 256us / 512us / 768us / 1024us（預設 256us）。

*注意：**Q-Code 實際上是採用輪詢（Polling）的方式處理此時間，所以實際時間會根據當時的負載輕重而有所誤差。***

例. **Matrix_Scan_SetupTime** = 256us ；**設定矩陣掃描設定時間為 256us。**

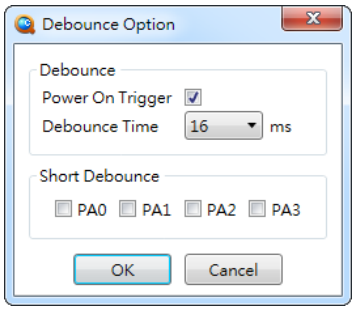
Short Debounce：使用者可以選擇哪一個獨立按鍵的 Short Debounce，時間範圍：0~1ms。

注意：

1. *此功能不支援 **Matrix Key**。*
2. *Q-Code 實際上是採用輪詢（Polling）的方式，所以實際 **debounce** 時間會有所誤差。如需要很精準的 **debounce** 時間或不同的 **debounce** 時間，需用組合語言來編寫。*

例. **ShortDebounce** = PA.0, PA.1, PA.2 ；**PA.0, PA.1, PA.2 為 Short Debounce 功能。**

3.1.17.3 NY9T



Power On Trigger：選擇是否要有 Power on Trigger 功能。當功能開啟後，如果使用者按著按鍵後開機，則開機後會馬上去執行該按鍵的路徑；反之，則不執行該按鍵路徑。（預設 Enable）

注意：

1. *上電觸發功能僅對一般按鍵有效，觸摸鍵不支援該功能。*
2. *如果開啟此功能，則 **PowerOn** 路徑必須設定 **input state**，否則會因為找不到設定路徑而無效。*

例. **PowerOnTrigger** = Disable ；**關閉所有按鍵的上電觸發功能。**

Debounce Time：使用者可以選擇按鍵的 debounce 時間，時間範圍：1~1000ms（預設值 16ms）。

注意：Q-Code 實際上是採用輪詢（Polling）的方式。原則上，實際按鍵的時間必須大於設定的 debounce 時間，如需要很精準的 debounce 時間或不同的 debounce 時間，需用組合語言來編寫。

例. **Debounce** = 16 ms ; **全部按鍵的 Debounce 時間為 16ms。**

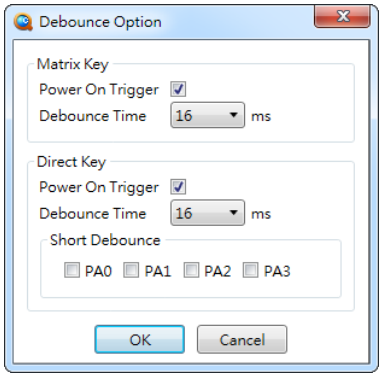
Short Debounce：使用者可以選擇哪一個獨立按鍵的 Short Debounce，時間範圍：0~1ms。

注意：

1. ***此功能不支援 Matrix Key***
2. ***Q-Code 實際上是採用輪詢（Polling）的方式，所以實際 debounce 時間會有所誤差。如需要很精準的 debounce 時間或不同的 debounce 時間，需用組合語言來編寫。***

例. **ShortDebounce** = PA.0, PA.1, PA.2 ; **PA.0, PA.1, PA.2 為 Short Debounce 功能。**

3.1.17.4 NX1



Power On Trigger：選擇是否要有 Power on Trigger 功能。當功能開啟後，如果使用者按著按鍵後開機，則開機後會馬上去執行該按鍵的路徑；反之，則不執行該按鍵路徑。允許個別設定 Direct Key 及 Matrix Key 是否開啟此功能。（預設值 Enable）

注意：如果開啟此功能，則 PowerOn 路徑必須設定 input state，否則會因為找不到設定路徑而無效。

例. **Direct_PowerOnTrigger** = Disable ; **關閉 Direct Key 上電觸發功能。**

例. **Matrix_PowerOnTrigger** = Disable ; **關閉 Matrix Key 上電觸發功能。**

Debounce Time：使用者可以選擇按鍵的 debounce 時間，時間範圍：1~1000ms（預設值 16ms）。允許 Direct Key 及 Matrix Key 設定不同的 debounce 時間。

注意：Q-Code 實際上是採用輪詢（Polling）的方式。實際按鍵的時間必須大於設定的 debounce 時間，如需要很精準的 debounce 時間或不同的 debounce 時間，需用組合語言來編寫。

例. **Direct_Debounce** = 16 ms ; **Direct Key 的 Debounce 時間為 16ms。**

例. **Matrix_Debounce** = 16 ms ; **Matrix Key 的 Debounce 時間為 16ms。**

Short Debounce：使用者可以選擇哪一個獨立按鍵的 Short Debounce，時間範圍：0~1ms。

注意：

1. *此功能不支援 **Matrix Key***
2. *Q-Code 實際上是採用輪詢（Polling）的方式，所以實際 **debounce** 時間會有所誤差。如需要很精準的 **debounce** 時間或不同的 **debounce** 時間，需用組合語言來編寫。*

例. **ShortDebounce** = PA.0, PA.1, PA.2 ; **PA.0, PA.1, PA.2 為 Short Debounce 功能。**

3.1.18 Interrupt Service

3.1.18.1 NY5

提供 3 種不同功能的中斷服務，分別為 Action 與 QIO 及 Melody。若對於精確度有較高的要求，可以將該功能改由中斷來處理。

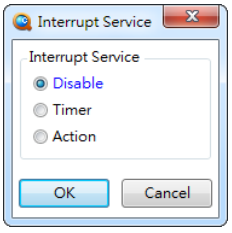
注意：

1. ***NY5+ / NY6 / NY7 Q-Code** 會固定將中斷服務打開，以確保所有功能皆能達到較高的精確度，故不支援此選項。*
2. *中斷服務同時僅能支援一種功能，設定後程式執行過程無法變更。*
3. *中斷服務於功能啟動時，系統自動致能，無須下達 **INT_ON** 指令。*
4. *使用中斷服務會佔用較多的 **RAM**。*
5. ***NY5** 的 **ADSR** 功能預設為開啟中斷，**PCT** 模式預設為關閉，其餘功能預設皆為關閉。*
6. *與時間中斷指令混用會導致無法預期的結果。*

例. **InterruptService** = Action ; **Action 功能使用中斷。**
例. **InterruptService** = QIO ; **QIO 功能使用中斷。**
例. **InterruptService** = Melody ; **Melody 功能使用中斷。**

3.1.18.2 NY9T

提供 2 種不同功能的中斷服務，分別為 Action 與 Timer。若對於精確度有較高的要求，可以將該功能改由中斷來處理。



注意：

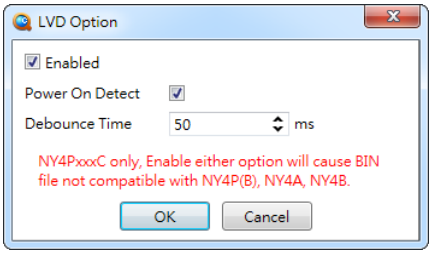
1. *中斷服務同時僅能支援一種功能，設定後程式執行過程無法變更。*
2. *中斷服務於功能啟動時，系統自動致能，無須下達 **INT_ON** 指令。*
3. *使用中斷服務會佔用較多的 **RAM**。*
4. *與時間中斷指令混用會導致無法預期的結果。*
5. *使用 **Action** 中斷服務會佔用較多的 **RAM**，且會一併啟動 **Timer** 中斷服務。*

例. **InterruptService** = Action ; **Action 功能使用中斷。**

3.1.19 LVD

使用者可以透過 IC 內建 LVD 功能對系統進行監控，避免電壓不穩定導致系統出錯，當系統電壓有變動時，程式跳至對應的系統路徑。

3.1.19.1 NY4PxxxC



Enabled：開啟或關閉偵測系統電壓。

例. **LVD = Enable**

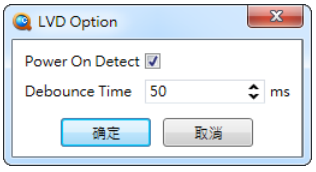
Power On Detect：上電隨即偵測目前的系統電壓，並跳至對應的系統路徑執行程式。

例. **PowerOnLVD = Enable**

Debounce：電壓變化需維持一段時間，才會判定電壓確實有變化。

例. **LVD_Debounce** = 50ms

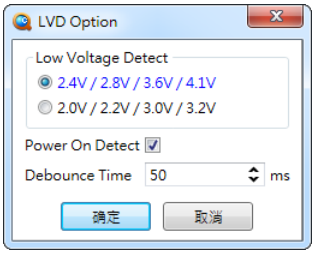
3.1.19.2 NY5+ / NY6B / NY6C / NX1



Power On Detect：上電隨即偵測目前的系統電壓，並跳至對應的系統路徑執行程式。

Debounce：電壓變化需維持一段時間，才會判定電壓確實有變化。

3.1.19.3 NY6P025A



NY6P025A 支援 2 組電壓偵測。

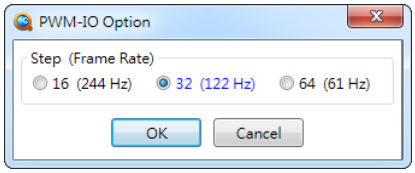
例. **LVD_Selection** = LVD2

Power On Detect：上電隨即偵測目前的系統電壓，並跳至對應的系統路徑執行程式。

Debounce：電壓變化需維持一段時間，才會判定電壓確實有變化。

3.1.20 PWM-IO

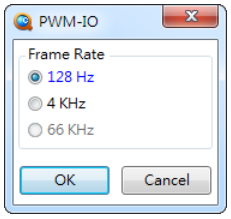
3.1.20.1 NY4 / NY5 / NY6 / NY7



PWMIO_Step：設定 PWMIO 的階數，可為 16 / 32 / 64，分別對應的 frame rate 為 244Hz / 122Hz / 61Hz。

例. **PWMIO_Step** = 32 ; **PWM 設定輸出階數 32。**

3.1.20.2 NY9T



Frame Rate：PWM-IO 訊號的更新率（NY9T001A 預設為 4KHz，其餘系列預設為 128Hz）。

例. **PWMIO_FrameRate** = 128Hz ; **PWM 更新率為 128Hz。**

例. **PWMIO_FrameRate** = 4KHz ; **PWM 更新率為 4KHz。**

注意：

1. ***更新率越高，LED 越不容易閃爍。***
2. ***NY9T001A 設定 FrameRate=66KHz 時，PWM-IO 僅能提供一個 PWM-IO 輸出，但能設定於 PE.0 / PE.1 / PE.2 中的任一腳位，其餘腳位僅能設定成一般 IO。（僅有 NY9T001A 提供 66K 選項）***
3. ***NY9T001A / NY9T008A / NY9T016A 設定 FrameRate=4KHz 時，PWM-IO 僅能使用 PE.0 / PE.1 / PE.2 腳位，其餘腳位僅能設定成一般 IO。（NY9T004A 僅提供 128Hz）***

NY9T 母體支援之 PWM 更新率和可設置的腳位:

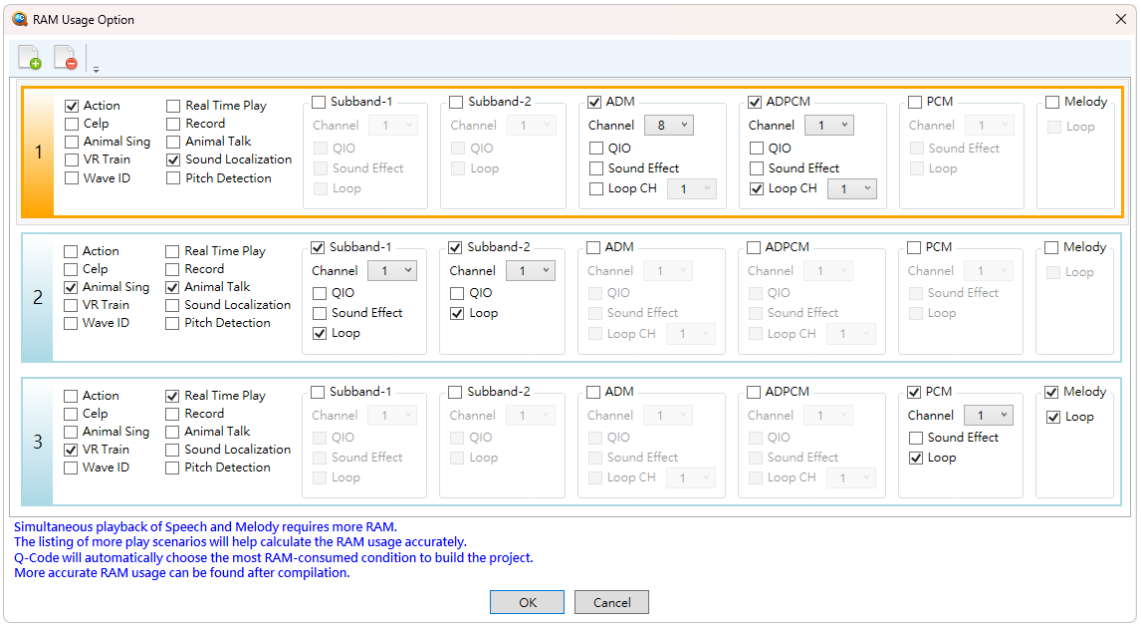
	NY9T001A		NY9T004A	NY9T008A		NY9T016A	
Frame Rate	4KHz	66KHz	128Hz	128Hz	4KHz	128Hz	4KHz
Pin	PE.0 ~ PE.2	one of PE.0 ~ PE.2 pin only	PE	PE ~ PF	PE.0 ~ PE.2	PE ~ PF	PE.0 ~ PE.2

3.1.21 RAM Usage

3.1.21.1 NX1

指定各種功能同時使用的狀況，依據此設定可推估執行期間佔用的記憶體空間。當有同時使用到兩個 channel 或兩種功能以上時，必須使用 RAM Usage 設定同時間使用的功能組合。每一種功能組合表示同一個時間，同時會執行的功能。

假設同時播放 SBC 20sec / ADPCM 10 sec，ADPCM 播放完後馬上播放 PCM(SBC 還未結束)，這樣要選擇 SBC + ADPCM + PCM。



Action：開啟動作訊號輸出功能。

Celp：使用 Celp 播放。

Animal Sing：開啟動物唱歌功能。

VR Train：開啟語音標籤訓練功能。

WaveID：開啟 WaveID 功能。

Real Time Play：開啟即時播放(大聲公)功能。

Record：開啟錄音功能。

Animal Talk：開啟動物音功能。

Sound Localization：開啟聽聲辨位功能。

Pitch Detection：開啟音高偵測功能。

Subband-1：選擇幾個通道使用 Subband-1 播放，最多 2 通道。

Subband-2：選擇幾個通道使用 Subband-2 播放，最多 2 通道。

ADM：選擇幾個通道使用 ADM 播放，最多 8 通道。

ADPCM：選擇幾個通道使用 ADPCM 播放，最多 8 通道。

PCM：選擇幾個通道使用 PCM 播放，最多 3 通道。

Melody：播放 MIDI。

QIO：對應的演算法開啟 QIO 功能。

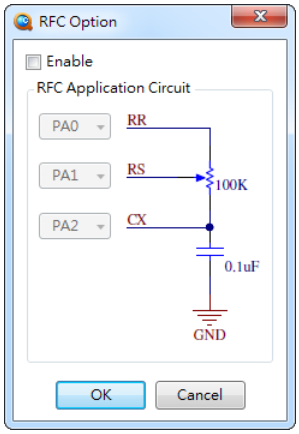
Sound Effect：對應的壓縮演算法開啟音效功能。

Loop：對應的演算法或 MIDI 開啟 Loop 功能。ADM / ADPCM 可以支援多個通道 Loop。

例. **RAM_Usage** = ADMx8+ADPCM+ADPCM_Loop+**Action**+**SoundLoc**,
Subband+Subband_Loop+SBC2+SBC2_Loop+AnimalTalk+AnimalSing,
Melody+Melody_Loop+**PCM**+PCM_Loop+**RT_Play**+VT_Train

3.1.22 RFC

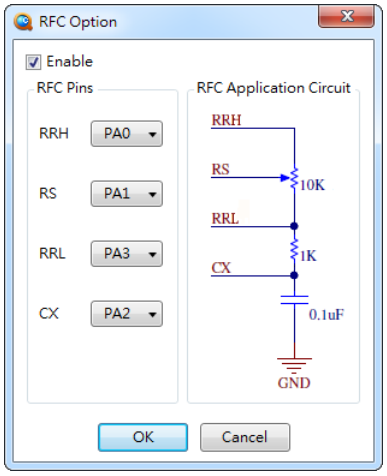
3.1.22.1 NY5+ / NY6 / NY7



RFC（Resistor to Frequency Converter）模組需外接一顆電容器及電位器，利用 I/O 對電容器充電的時間常數來偵測電位器旋鈕的位置，解析度為 16 階，簡而言之，就是電位器的旋鈕會被分為 16 個區域，使用者可透過 API 讀取目前旋鈕所在的區域來執行不同的控制，如音量大小及節奏快慢等。

注意：電容器及電位器的數值建議為 0.1uF 及 100KΩ，如使用者有必要調整，需儘量維持相同的充電常數；例如，電位器數值降低為一半，則電容器數值就需增加一倍，以此類推。

3.1.22.2 NX1

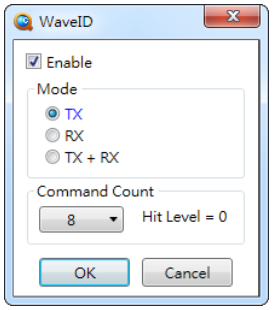


RFC（Resistor to Frequency Converter）模組需外接一顆電容器及電位器，利用 I/O 對電容器充電的時間常數來偵測電位器旋鈕的位置，解析度為 16 階，簡而言之，就是電位器的旋鈕會被分為 16 個區域，使用者可透過 API 讀取目前旋鈕所在的區域來執行不同的控制，如音量大小及節奏快慢等。

注意：電容器及電位器的數值建議為 0.1uF 及 10KΩ，如使用者有必要調整，需儘量維持相同的充電常數；例如，電位器數值降低為一半，則電容器數值就需增加一倍，以此類推。

3.1.23 WaveID

3.1.23.1 NX1



Wave ID 使用喇叭傳送 ID，透過麥克風接收 ID，達成 ID 的傳送和接收。喇叭傳送的頻率為人耳不靈敏的頻帶(16K ~ 20KHz)，應用上可與一般語音一起由喇叭輸出，搭配的語音只能由 Ch0 播放。

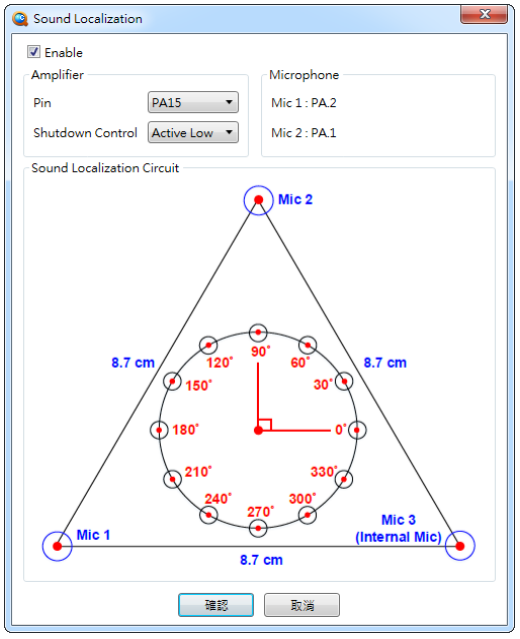
Mode：選擇 TX or RX 功能。

Command Count：選擇 ID 傳送的數量，最多 38 個 ID。ID 越多，會降低接收的成功率。

ID	Hit Rate
<= 12	95%
<= 19	90%
<= 30	85%
<= 38	80%

3.1.24 Sound Localization

3.1.24.1 NX1 OTP

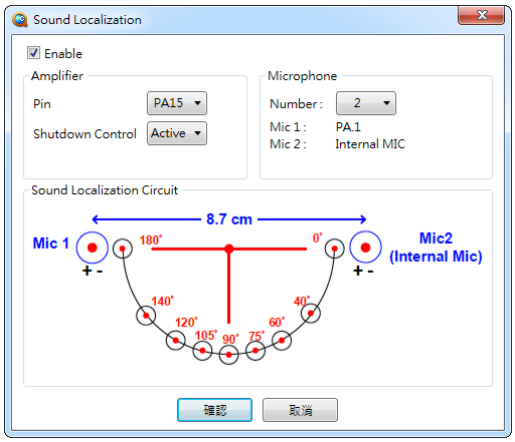


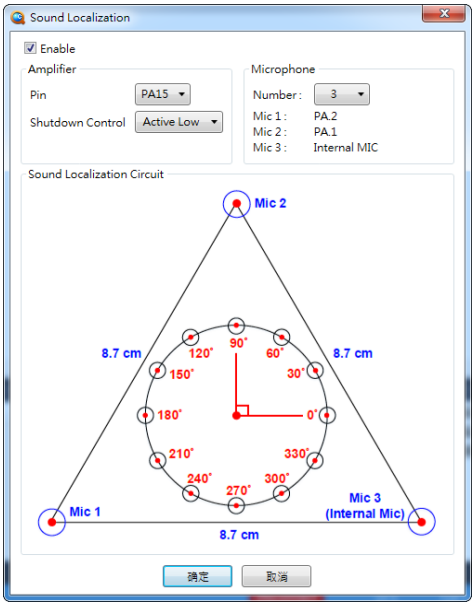
Amplifier Pin：外部放大器控制腳位。

Amplifier Shutdown Control：外部放大器控制方式。(預設 Active Low)。

3.1.24.2 NX1 EF

這功能可以選擇使用 2 或 3 組麥克風，透過外部放大器放大輸入訊號，再搭配內建的麥克風，可推算出聲音來源的角度。





Amplifier Pin：外部放大器控制腳位。

Amplifier Shutdown Control：外部放大器控制方式。(預設 Active Low)。

Microphone Number：麥克風數量。

注意：2 組麥克風需佔用 PA.1，3 組麥克風需佔用 PA.1、PA.2。

例.

SoundLoc = Enable

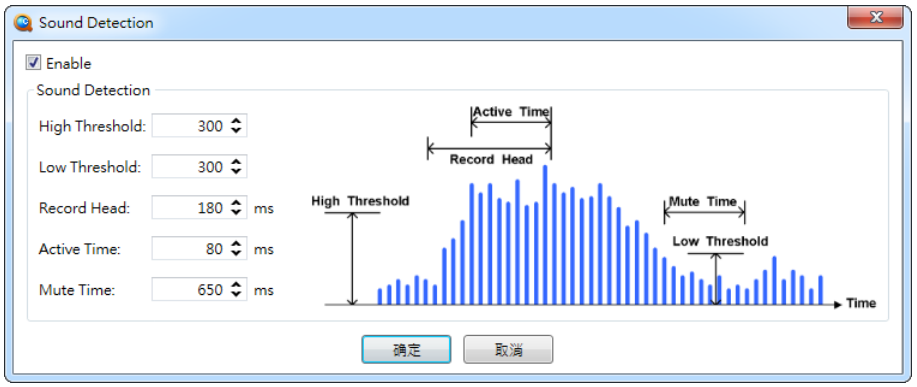
SoundLoc_Amp_Pin = PA.15

SoundLoc_Amp_Shutdown_Control = Active_Low

SoundLoc_MIC = 2

3.1.25 Sound Detection

3.1.25.1 NX1



High Threshold：偵測有語音的門檻值。高於此門檻且超過 Active Time，會被判定為有效的語音。

Low Threshold：偵測無語音的門檻值。低於此門檻且超過 Mute Time，會被判定為僅剩背景音，而停止錄音。

Active Time：偵測有語音的時間。語音高於 High Threshold 且超過偵測時間，會被判定為有效的語音。

Mute Time：偵測無語音的時間。語音低於 Low Threshold 且超過偵測的時間，會被判定為僅剩背景音，而停止錄音。

例.

SoundDetect = Enable

SoundDetect_High_Threshold = 300

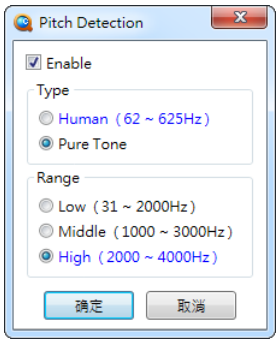
SoundDetect_Low_Threshold = 300

SoundDetect_Active_Time = 80ms

SoundDetect_Mute_Time = 650ms

3.1.26 Pitch Detection

3.1.26.1 NX1



Type：選擇偵測人聲（Human）或純音頻（Pure Tone）。

例. **PitchDetect** = Tone

Range：設定純音頻（Pure Tone）的偵測範圍。

例. **PitchDetect_Range** = Low

3.1.27 Maximum Single Note

3.1.27.1 NY5+ / NY6 / NY7

使用者可以選擇最多可同時播放的琴鍵音數目，設定範圍：1 ~ 8。(預設值=4)

注意：

1. ***NY5+最多四個通道，沒有在播放琴鍵音或語音的通道才能用來播放 melody 使用。***
2. ***NY6 最多六個通道，沒有在播放琴鍵音或語音的通道才能用來播放 melody 使用。***
3. ***NY7 最多八個通道，沒有在播放琴鍵音或語音的通道才能用來播放 melody 使用。***

例. **Melody_MaxSingleNote** = 4 ；**設定最大琴鍵音數目為 4 個。**

3.1.28 Melody_OKON_BgNote

3.1.28.1 NY5+ / NY6 / NY7

預設情況下，當使用 One-Key-One-Note 功能時，僅會播放 MIDI 通道 1。使用者可以開啟此選項，一併播放其它通道。

例. **Melody_OKON_BgNote** = Enable ；開啟 OKON 背景播放功能。

3.1.29 Variable Compatible

3.1.29.1 NX1

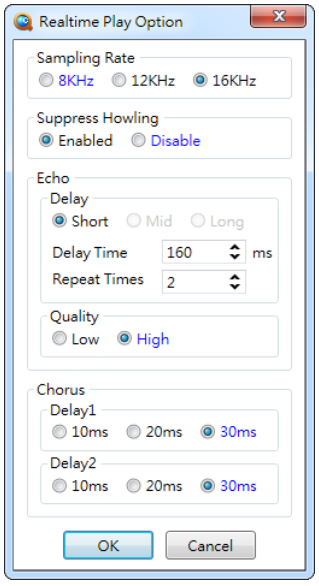
變數相容性提供了 NX1 系列對傳統 4bit 變數運算的相容性，如果您在 NX1 系列仍然需要使用 4 位元的資料型態，請設定在 NX1 系列，變數最小長度的變數是 8 位元。如果需要 4 位元的變數，請設定 Variable_Compatible。*Q-Code* 將會產生 R0、R1、X0、X0H、X0L 之類的變數，讓數學運算的指令與 NY4 / 5 / 6 / 7 / 9T 一致。

例. **Variable_Compatible** = Enable ；開啟變數運算相容性。

3.1.30 Realtime Play

3.1.30.1 NX1

即時播放(大聲公)功能。



Sampling Rate：選擇採樣率。

例. **RT_Sampling_Rate** = 8000

Suppress Howling：囂叫聲抑制。

例. **RT_SuppressHowling** = Enable

Echo Delay Time：迴聲的長度。

例. **RT_Echo_DelayTime** = 200ms

Echo Repeat Times：迴聲的重複次數。

例. **RT_ECHO_Repeat** = 2

Echo Quality：迴聲的品質。

例. **RT_ECHO_Quality** = High

Chorus Delay1：合音頻道 1 與頻道 2 之間的延遲時間，支援 10/20/30ms。

例. **RT_Chorus_Delay1** = 30ms

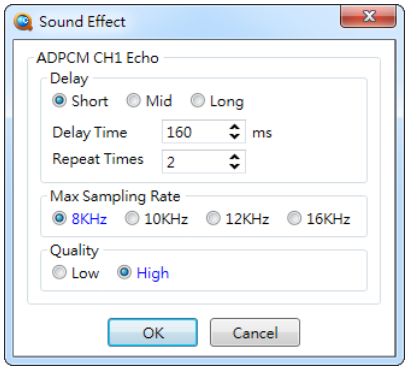
Chorus Delay2：合音頻道 2 與頻道 3 之間的延遲時間，支援 10/20/30ms。

例. **RT_Chorus_Delay2** = 30ms

3.1.31 Sound Effect

3.1.31.1 NX1

音效設定。



ADPCM CH1 Echo Delay Time：使用者自訂 ADPCM CH1 迴聲的長度。

例. **SFX_ADPCM_Echo_DelayTime** = 200ms

ADPCM CH1 Echo Repeat Times：使用者自訂 ADPCM CH1 迴聲的重複次數。

例. **SFX_ADPCM_Echo_Repeat** = 4

ADPCM CH1 Echo Max Sampling Rate：選擇 ADPCM CH1 迴聲的最大採樣率。

例. **SFX_ADPCM_Echo_Max_SR** = 8000

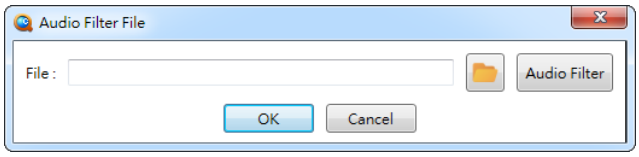
ADPCM CH1 Echo Advance Quality：選擇高品質的 ADPCM CH1 迴聲。

例. **SFX_ADPCM_Echo_Quality** = High

3.1.32 Audio Filter

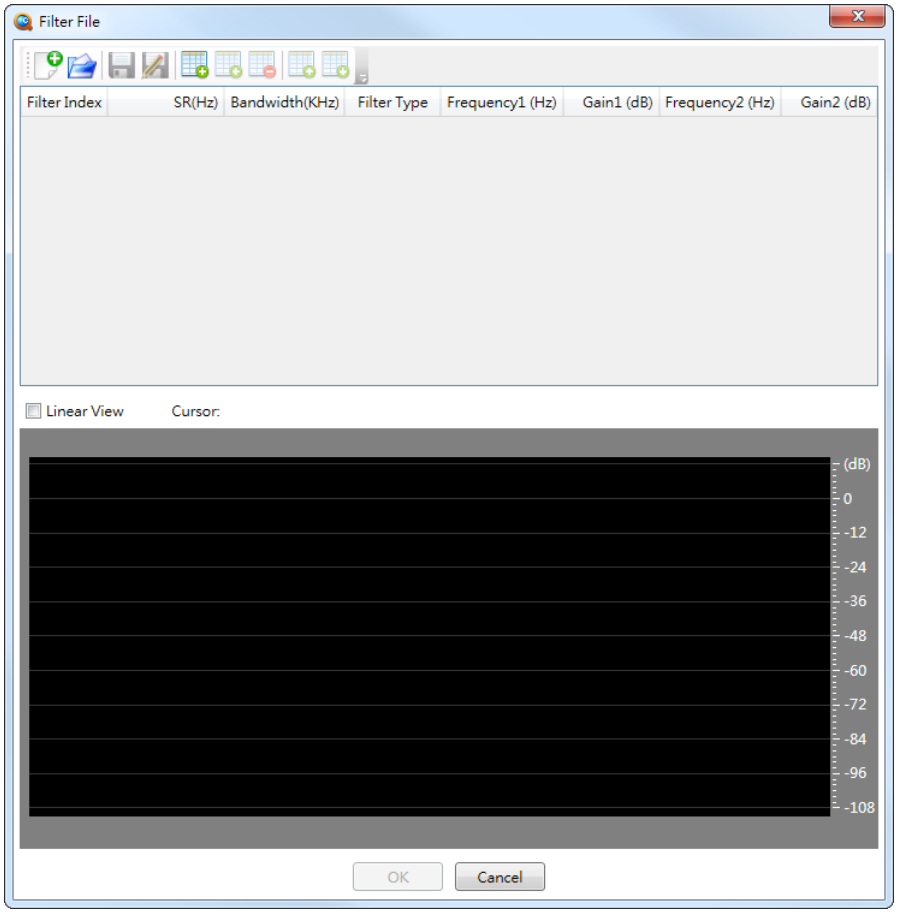
3.1.32.1 NX1

設定畫面如下圖所示，可直接選擇已編輯好的.fnx 檔案，或是呼叫音頻濾波器編輯視窗來產生.fnx 檔案。



點擊上圖音頻濾波器按鈕會跳出編輯視窗畫面如下圖所示。可設定多組的濾波器，每組濾波器可設定取樣頻率、頻寬以及濾波器類型，目前支援低通濾波器(LPF)、高通濾波器(HPF)、帶通濾波器(BPF)以及等化器(EQ)四種類型。選擇類型後，再輸入所需過濾的頻率或頻率範圍與增益值即完成設定。設定完成可將此設定保存為副檔名為.fnx 的檔案，方便之後重複使用。

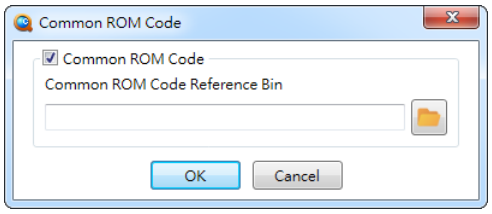
例. **Custom_Filter** = Filter.fnx



3.1.33 Common ROM Code

3.1.33.1 NX1

ROM 程式共用功能用於主要功能相同，僅需替換資源內容的應用。替換的內容放置於 SPI Flash 中，只需替換 SPI Flash 就可以有不同的效果。為了維持主要 ROM 內容一致，會彈出以下畫面進行設定。



ROM 程式共用參考 bin 檔（Common ROM Code Reference Bin）則是選擇之前建立的 bin 檔案，*Q-Code* 會自動對目前與選擇 bin 檔中的 ROM 資料進行比對，以確保 ROM 的資料完全一致。

基本開發 ROM 程式共用應用流程如下：

1. 首次開發時開啟 ROM 程式共用功能。
2. 後續開發的程式，需開啟 ROM 程式共用功能，並選擇首次開發產生的 bin 檔作為比對檔案。

按照以上的操作，*Q-Code* 會檢查產出的 ROM 資料皆為一致，否則會發出錯誤。

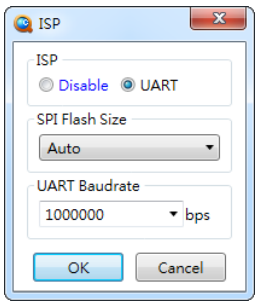
例.

Common_ROM_Code = Enable

Common_ROM_Ref_Bin = D:\ NX1\bin\New_Jump_Cmd_NX1.bin

3.1.34 ISP

3.1.34.1 NX1 EF



設置 ISP 的通訊類型。目前僅支援 UART，而 UART 通訊內建使用 PD.0/1 這組腳位，更詳細的 ISP 操作與說明，請參考 NYISP 使用手冊目錄。

例.

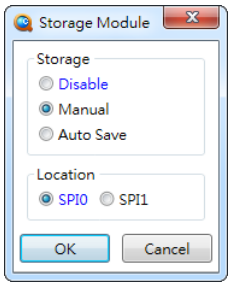
ISP = UART

ISP_UART_Baundrate = 1000000bps

ISP_FlashSize = 256Mbit

3.1.35 Storage Module

3.1.35.1 NX1 OTP



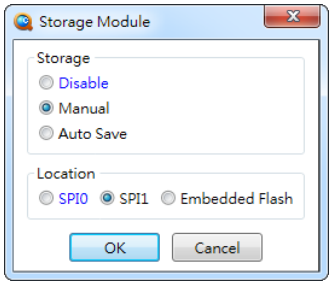
Storage：設定儲存的模式。

例. **Storage** = Disable

Locaiton：設定儲存的位置。

例. **Storage_Location** = SPI0

3.1.35.2 NX1 EF



Storage：設定儲存的模式。

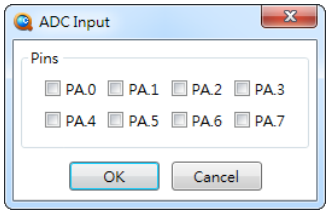
例. **Storage** = Manual

Locaiton：設定儲存的位置。

例. **Storage_Location** = EF

3.1.36 ADC Input

3.1.36.1 NX1



設定類比數位轉換器的類比訊號輸入腳。

例. **ADC_Input_Pins** = PA.0, PA.1, PA.2

3.2 Voice File

播放用的聲音檔案，一般副檔名為.wav，也可以是經過編碼後的.v4x / .v5x / .v5px / .v6x / .v7x / .vnx 檔案，或是經由 *Q-Sound* 產生的.nyw 檔案。若是有 *Quick-IO* 功能(QIO 信號或 WaveMark 信號)的副檔名為.nyq。如果加進來的是.nyq 檔案，將會在播放該檔案時，啟動 QIO 或 Wave Mark 功能。

單擊 Voice File 段落功能表中的 Add File，開啟 Voice File，可以向程式中插入 Voice 音源。如果想要刪除一個或多個音源，則只需要先選中您想要刪除的音源檔案，點擊 Delete Files 就可移除不需要的音源。

注意：Voice Label 使用者可以自行定義。

點擊 Add File 後會出現 Voice File 的對話框，可在 Voice File 的對話框中加入所需的 Voice 音源。不同系列的 Voice File 對話框會顯示不同的欄位。

加入檔案：新增音源檔。

刪除檔案：刪除選取的檔案。

快速更名：快速更名多個 Label 的工具。

反相：將目前選取的檔案取消選取，並選取其餘未選取的檔案。

向上移動：選取的項目向上移動。

向下移動：選取的項目向下移動。

向上移動檔案：向上移動選取的檔案。

向下移動檔案：向下移動選取的檔案。

Q-Sound：是針對音源檔切割而開發的軟體工具。它提供簡易的圖形處理介面來達成切割檔案的功能。使用時僅需在 Q-Sound 完成標記編輯後儲存。

試聽鈕：可試聽所選擇的音源檔，拖曳 indicator 可控制播放進度。

十進制 / 十六進制：將 ROM Size 的數值切換成十進制或十六進制。

Selected ROM size：選取的音源檔預估 ROM Size 的總和。

Total ROM Size：所有音源檔預估 ROM Size 的總和。

注意：

1. Q-Code 內建的 Q-Sound 功能，支援外部 Q-Sound 切好的.nyw 檔。

2. *所有插入的標記（Mark）須在儲存後才會記錄。*
3. *若使用者添加的是 .v5px / .v6x / .v7x / .vnx 檔，或是 Voice_Encoder 2.20 或更新版本所編碼的.v4x / .v5x，程式會自動使用原始音源的採樣率。例如：PlayV(Ch0, $V0)。*
4. *若是用 Voice_Encoder 2.20 之前的版本編碼的 .v4x / .v5x，則需要手動寫上原始音源的採樣率。例如：PlayV(Ch0, $V0, 6K)。*

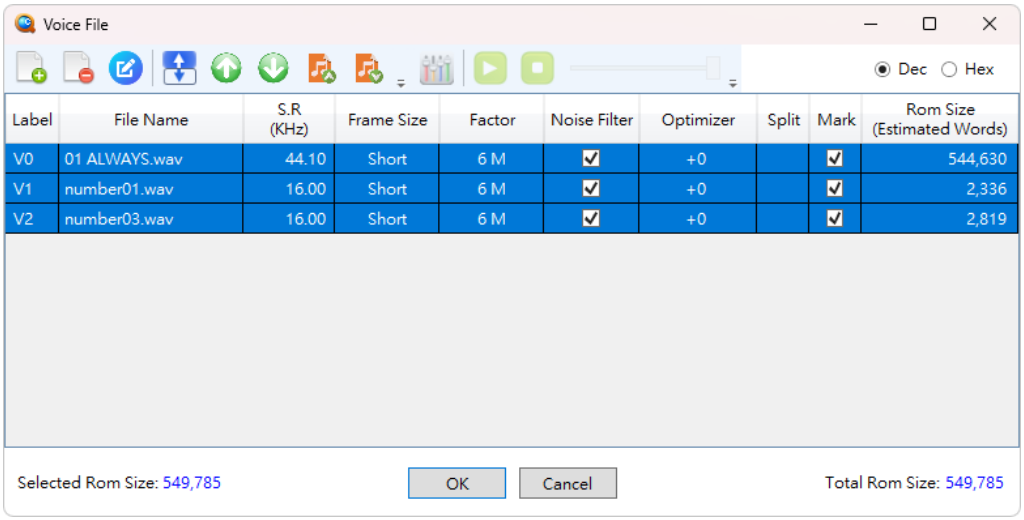
例.

[Voice File]

V0 = D:\NyquestFile\WAV\apple.nyw /6

V1 = WAV\B13.wav /PCM

3.2.1 NY4



S.R（Sample Rate）：顯示加入的音源檔案的採樣率，採樣率越高所佔的 ROM Size 越大。

Frame Size：供兩種不同的 Frame Size，分別是 Short 與 Long。

1. Short Frame Size：音質較佳，但是佔 ROM 較多的聲音壓縮方式。
2. Long Frame Size：音質一般，但是佔 ROM 較少的聲音壓縮方式。

Factor：語音壓縮率。以提供 12 種不同的壓縮率，其數字越大，則代表壓縮比越低，所佔的 ROM Size 也越高，則語音品質也隨之提高。另外，還提供 PCM 模式，則是對語音不採取壓縮的方式，其所佔的 ROM Size 最大，聲音品質也是最好。(聲音的品質與壓縮率乃是成正比，預設值為 6)

Noise Filter：提供雜訊濾波功能（降噪）。在某些情況下，聲音的輸出聽起來會有細微的背景雜音混雜在音源裡面，所以我們會使用 Noise Filter 來將雜音濾掉。

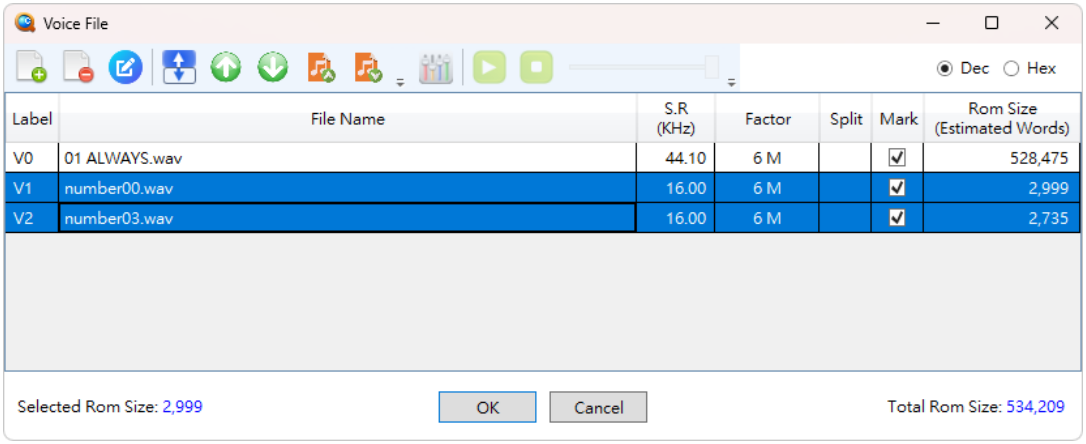
Optimizer：將聲音依設定值進行優化。聲音優化功能可對音樂訊號進行音色調整，以增強對比度（sharp），或者讓聲音更柔和（smooth）。

Split：語音檔切割功能。可針對單一檔案開啟或關閉。

Mark：開啟.wav 檔的 Wave Mark 功能。

Estimated ROM Size：語音檔被加入後，預估所需的 ROM Size。

3.2.2 NY5 / NY5+



S.R（Sample Rate）：顯示加入的音源檔案的採樣率，採樣率越高所佔的 ROM Size 越大。

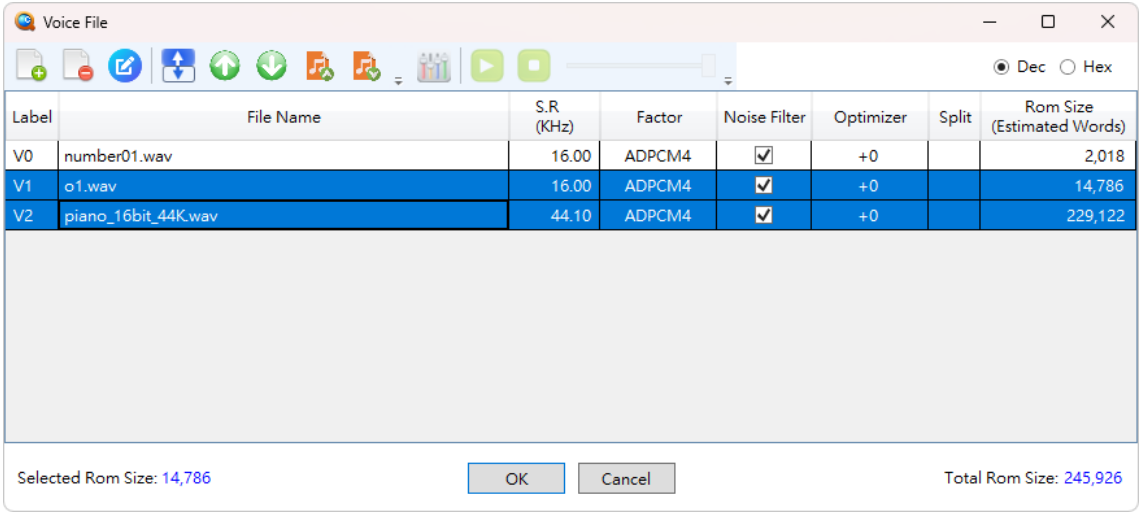
Factor：語音壓縮率。提供 12 種不同的壓縮率，其數字越大，則代表壓縮比越低，所佔的 ROM Size 也越高，則語音品質也隨之提高。另外，還提供 PCM 模式，則是對語音不採取壓縮的方式，其所佔的 ROM Size 最大，聲音品質也是最好。(聲音的品質與壓縮率乃是成正比，預設值為 6)。

Split：語音檔切割功能。可針對單一檔案開啟或關閉。

Mark：開啟.wav 檔的 Wave Mark 功能

Estimated ROM Size：語音檔被加入後，預估所需的 ROM Size。

3.2.3 NY6



S.R（Sample Rate）：顯示加入的音源檔案的採樣率，採樣率越高所佔的 ROM Size 越大。

Factor：語音壓縮率。預設使用 ADPCM4 模式。另外，還提供 ADPCM5 / PCM 模式。PCM 採取不壓縮的方式，其所佔的 ROM Size 最大，聲音品質也是最好。

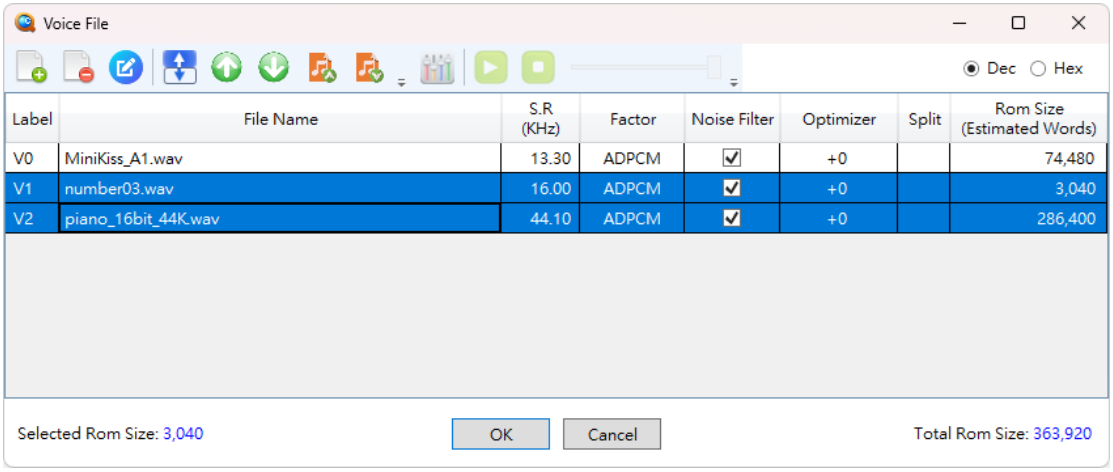
Noise Filter：提供雜訊濾波功能（降噪）。在某些情況下，聲音的輸出聽起來會有細微的背景雜音混雜在音源裡面，所以我們會使用 Noise Filter 來將雜音濾掉。

Optimizer：將聲音依設定值進行優化。聲音優化功能可對音樂訊號進行音色調整，以增強對比度（sharp），或者讓聲音更柔和（smooth）。

Split：語音檔切割功能。可針對單一檔案開啟或關閉。

Estimated ROM Size：語音檔被加入後，預估所需的 ROM Size。

3.2.4 NY7



S.R（Sample Rate）：顯示加入的音源檔案的採樣率，採樣率越高所佔的 ROM Size 越大。

Factor：語音壓縮率。預設使用 ADPCM 模式。另外，還提供 PCM 模式，則是對語音不採取壓縮的方式，其所佔的 ROM Size 最大，聲音品質也是最好。

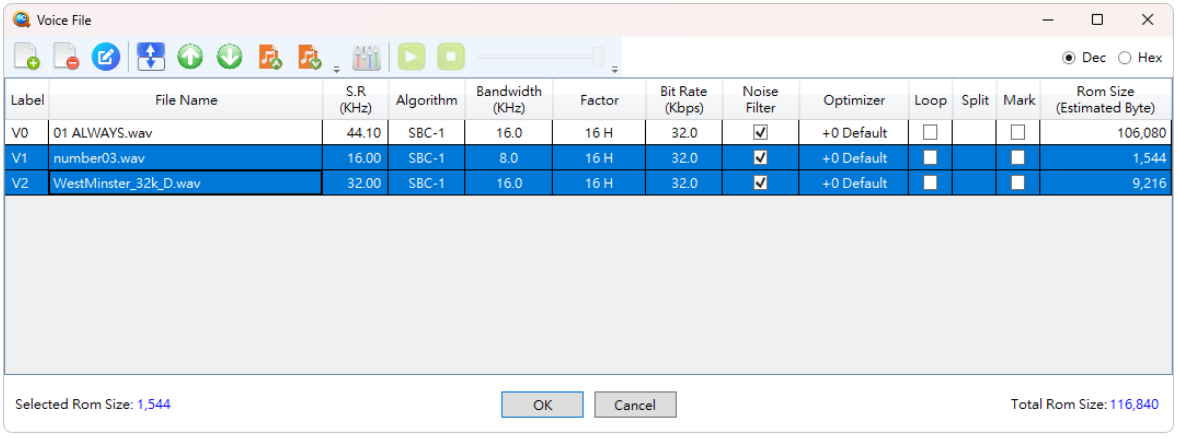
Noise Filter：提供雜訊濾波功能（降噪）。在某些情況下，聲音的輸出聽起來會有細微的背景雜音混雜在音源裡面，所以我們會使用 Noise Filter 來將雜音濾掉。

Optimizer：將聲音依設定值進行優化。聲音優化功能可對音樂訊號進行音色調整，以增強對比度（sharp），或者讓聲音更柔和（smooth）。

Split：語音檔切割功能。可針對單一檔案開啟或關閉。

Estimated ROM Size：語音檔被加入後，預估所需的 ROM Size。

3.2.5 NX1



S.R（Sample Rate）：顯示加入的音源檔案的採樣率，採樣率越高所佔的 ROM Size 越大。

Algorithm：除了常用的 PCM 和 ADPCM 格式以外，NX1 也提供更高壓縮率的 SBC-1 / SBC-2 / CELP 格式供客戶選擇。子帶編碼（Sub-Band Coding）是一種以信號頻譜為依據的波形編碼方法，

它首先用一組帶通濾波器將輸入信號按頻譜分開，然後讓每路子信號通過各自的自我調整 PCM 編碼器（ADPCM）編碼，經過分接和解碼再複合成原始信號。經由高效能的 NX1 以軟件的方式來實現 Sub-Band Coding，能夠達到更低的 Bit Rate 來節省內存的需求量，而且以較高的採樣率、更寬的頻帶來重現高頻的音域表現，尤其是高音域的器樂更是如此。SBC-2 比 SBC-1 降低了 ROM 空間、RAM 空間和 CPU 負載，但 SBC-2 音質會比 SBC-1 差一些。CELP 則是針對人聲所特定發展出來的超低 Bit Rate 壓縮演算法，不過，CELP 並不適用於非人聲的應用。

Bandwidth：單位為 KHz。

Factor：使用動態壓縮的演算法，依據選擇的演算法有不同的壓縮率供選擇，SBC-1 / SBC-2 共有 19 階參數（-2~16）；ADPCM4 和 ADPCM5 共有 12 階參數（1~12）；CELP 為固定壓縮率。客戶可以根據產品的需求，選定合適的演算法，再依聲音品質的要求，調整 Factor 的高低來達到優化記憶體的使用。

Bit Rate：單位時間播放壓縮後語音的位元數量，它相當於語音播放時的頻寬消耗量。可以想像的是，較高的位元率可容納更高的語音品質，單位為 Kbps。

Noise Filter：提供雜訊濾波功能（降噪）。在某些情況下，聲音的輸出聽起來會有細微的背景雜音混雜在音源裡面，所以我們會使用 Noise Filter 來將雜音濾掉。

Optimizer：將聲音依設定值進行優化。聲音優化功能可對音樂訊號進行音色調整，以增強對比度（sharp），或者讓聲音更柔和（smooth）。

Loop：選擇是否開啟循環播放功能，當開啟功能時將會耗用較多的 ROM Size。

Split：語音檔切割功能。可針對單一檔案開啟或關閉。

Mark：開啟.wav 檔的 Wave Mark 功能。

Estimated ROM Size：語音檔被加入後，預估所需的 ROM Size。

3.2.6 *Q*-Code 如何播放 *Q-Sound* 處理後的檔案

Q-Code 提供簡易的方式，讓使用者可以播放 *Q-Sound* 處理後的檔案。

第一步：在 Voice File 對話框中，先將要播放的 Voice File 加入，加入後將 Split 功能打開。

第二步：開啟 *Q-Sound* 並設定好要切割的音源。

第三步：在 *Q-Code* 中下達 PlayV 指令。

例.

[Voice File]

V01 = C:\MyQ-CodeProjects\Prj1\voice1.nyw /s

;在 Voice 檔後面帶了 /s 參數，表示有使用 Split 功能。

[Path]

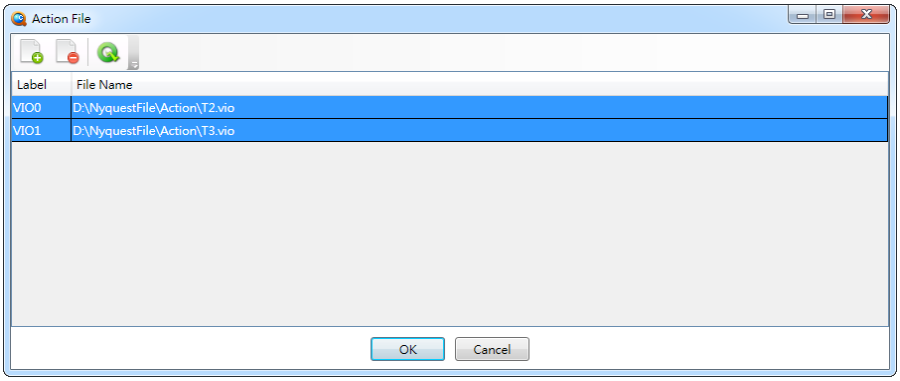
PowerOn: PlayV(Ch1, $V01)

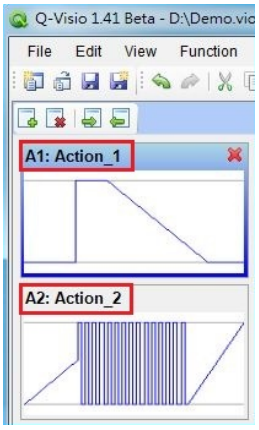
;要播放 Label“$V01”音檔，僅需要下 PlayV 指令，系統會自動將切完的檔案展開，使用者不需要另外處理。

3.3 Action File

3.3.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1

播放用的訊號檔，一般訊號檔的副檔名為 .vio。關於 Action 的編輯，請參考 *Q-Visio* 使用手冊。





例.

[Action File]

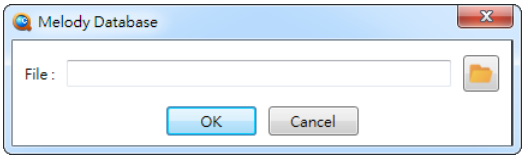
VIO0 = C:\Project.vio ； 加入 **Action 檔。**

[Path]

PowerOn: PlayA(PB.0, CH1, $A1) ； **上電後立即播放 Action 檔，並將 A1 的訊號輸出至 PB.0。**

3.4 Melody Database

播放 melody 時，需要音樂資料庫，其中包含音色、包絡、以及樂曲等資訊。使用 Q-Code 寫程式時需添加 Melody Database 的資料，才能正常播放 melody。



例.

[Melody Database]

C:\Project.md2

; m0 = mla_a.mid, 2 channel

; m1 = song14_1ch.mid, 1 channel

3.4.1 NY5

使用的音樂資料庫為 *Q-MIDI* 產生的.md2 檔案。

注意：使用者需要通過 Q-MIDI 自行製作音色庫，產生一個.idb 檔，之後將 melody 會用到的音色與音色庫聯結並產生一個.md2 音色檔，即可在 Q-Code 程式中添加這個音色檔（詳情請見 Q-MIDI 使用手冊）。

3.4.2 NY5+ / NY6 / NY7 / NX1

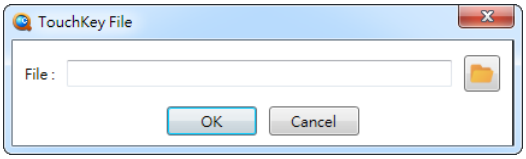
使用的音樂資料庫為 *Q-MIDI* 產生的.qmd 檔案。

注意：使用者需要通過 Q-MIDI 的 Instrument Database 頁面自行製作音色庫，產生一個.idb3 檔，之後用 Q-MIDI 的 Music 頁面添加需使用的.mid 檔案，Q-MIDI 會用到的音色與音色庫聯結並產生一個.qmd 音樂資料檔，即可在 Q-Code 程式中添加這個音樂資料檔（詳情請見 Q-MIDI 使用手冊）。

3.5 TouchKey File

3.5.1 NY9T

觸摸鍵靈敏度設定檔案，可替換檔案來對應不同的靈敏度。副檔名為.t9x。



例.

[TouchKey File]

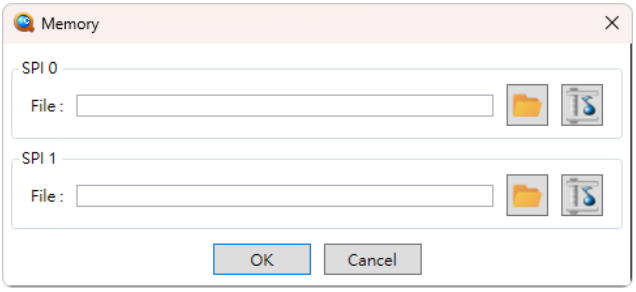
; TouchKey Number = 1

; Scane Mode = Preset。

C:\Normal.t9x

3.6 Memory

3.6.1 NX1 OTP



可將資源放在 SPI Flash 上，支援 SPI0 / SPI1。

需開啟 *SPI_Encoder* 編輯所需要的資源，再加入專案檔，副檔名為.spiprj，建置後會產生 _SPI.bin 檔，可以經由 *Q-Writer* 或是 *Q-Code* 下載。

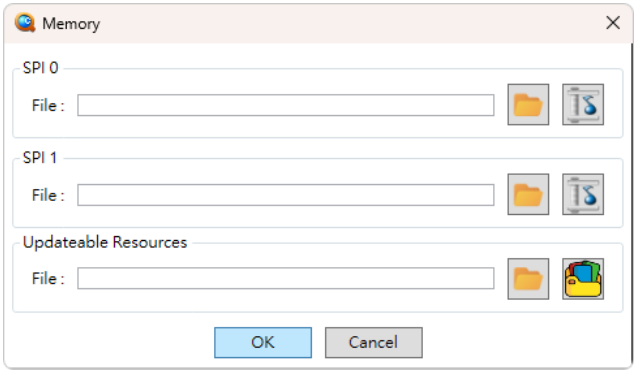
例.

[Memory]

SPI0_File = C:\Users\Desktop\Untitled.spiprj

SPI1_File = C:\Users\Desktop\Untitled.spiprj

3.6.2 NX1 EF



可將資源放在 SPI Flash 上，支援 SPI0 / SPI1，另外支援 Updateable Resources。

需開啟 *SPI_Encoder* 編輯所需要的資源，再加入專案檔，副檔名為.spiprj，建置後會產生 _SPI.bin 檔，可以經由 *Q-Writer* 或是 *Q-Code* 下載。

Updateable Resources 可按下圖所示按鈕編輯內容，並存為.udrprj 檔案。

例.

[Memory]

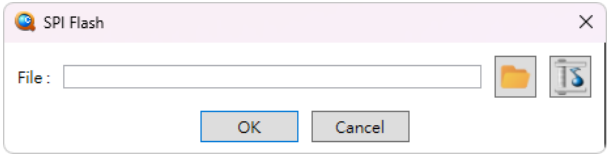
SPI0_File = C:\Users\Desktop\Untitled.spiprj

SPI1_File = C:\Users\Desktop\Untitled1.spiprj

UDR_FIle = C:\Users\Desktop\Untitled2.spiprj

3.7 SPI Flash

3.7.1 NY6



加入 *SPI_Encoder* 使用的專案檔，副檔名為.spiprj，建置後會產生 _SPI.bin 檔，可以經由 *Q-Writer* 或是 *Q-Code* 下載。

關於 *SPI_Encoder* 的編輯，請參考 *SPI_Encoder* 使用手冊。

例.

[SPI Flash]

C:\Users\Desktop\Untitled.spiprj

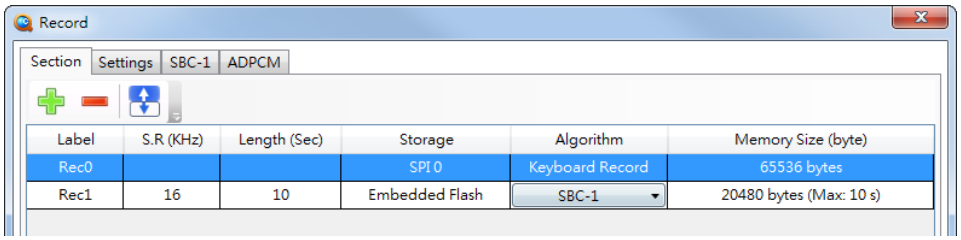
3.8 Record

3.8.1 NX1

NX1 提供了簡單的方式進行錄音。當使用 SPI Flash 做為錄音區段的儲存空間時，會將錄音的結果寫至 SPI Flash 上。為了進行錄音，SPI Flash 上需要保留空間。所以在 Build 後會產生_SPI.bin。

3.8.1.1 Section

需要定義錄音的區段。每次進行錄音或擦除時都是以區段為單位。



Length：語音錄音的時間長度，區段的大小由所選定的錄音演算法與錄音的時間長度所決定。而琴鍵錄音區段的大小固定為 64kB。

Storage：錄音區段的儲存空間，可支援 SPI0 / SPI1 / Embedded Flash。琴鍵錄音只支援 SPI0。

Algorithm：SBC-1 / ADPCM 是語音錄音，Keyboard Record 則是做為琴鍵錄音使用。

Memory Size：實際佔用的 Size。

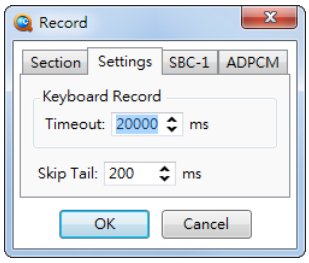
例.

[Record]

Rec0 = [Algorithm: KRecord, Storage: SPI0, Size: 64kb]

Rec1 = [Algorithm: SBC-1, Storage: EF, Length: 10s]

3.8.1.2 Settings



Skip Tail：會刪除錄音的結尾，可以避免將結束的按鍵音一併錄進去。

Keyboard Record Timeout: 當使用琴鍵錄音功能時，設定多久未收到 InstNoteOn / InstNoteOff 即結束錄音功能。

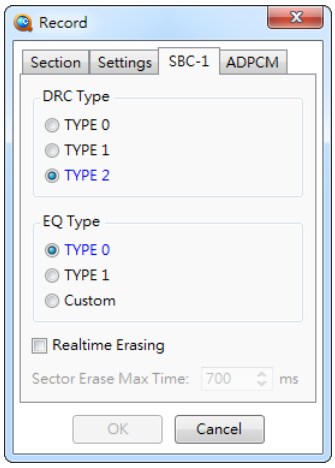
例.

[Record]

KRecord_Timeout = 20000ms

Skip_Tail = 200ms

3.8.1.3 SBC-1



DRC Type：針對錄音播放，TYPE 0 不做任何處理，可以減少資源消耗。TYPE 1 會減少最大聲與最小聲之間的差距，使聲音更加厚實、宏亮。TYPE 2 僅放大小音量部份，使小音量變更清楚。(預設值 TYPE 2)

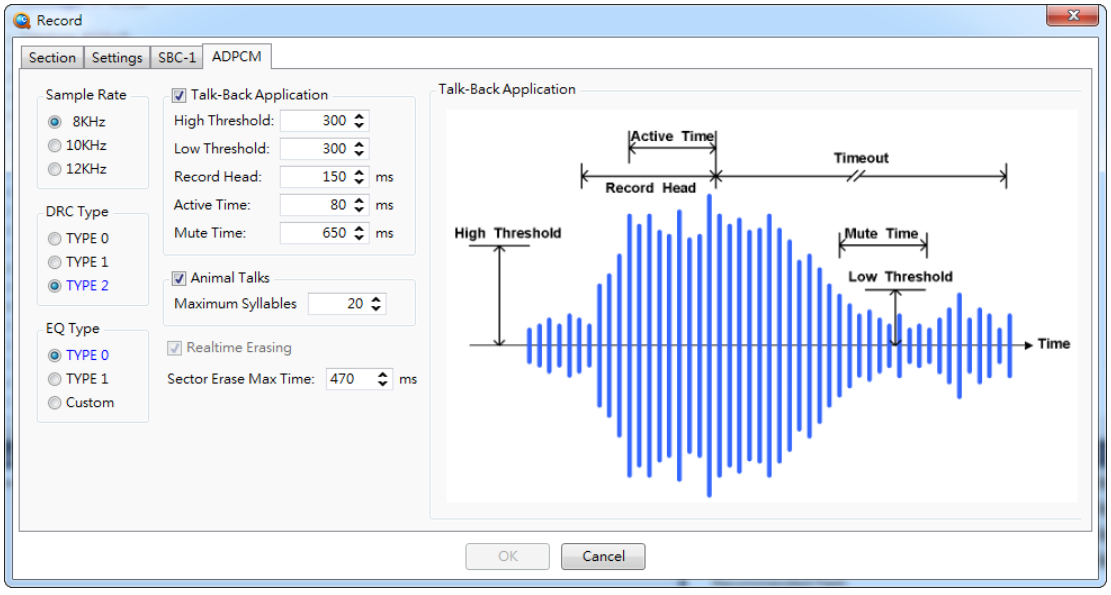
EQ Type：針對錄音播放套用 EQ 效果。TYPE 0 會銳化音色特徵，使聲音較為明亮，適合用於樂曲類型；TYPE 1 加強低音，使聲音較為厚實，使人聲更為突出；Customize 則使用音頻濾波器（Audio Filter）進行音訊處理。(預設值 TYPE 0)

Realtime Erasing：是否開啟隨擦即錄功能。未開啟時，使用者要自行在錄音前使用 EraseR / EraseRS 等指令將要錄音的區段進行擦除。開啟後 Q-Code 會在錄音前視需要進行錄音區段擦除的動作。開啟此功能會耗費較多的 RAM。

Sector Erase Max Time：使用隨擦即錄功能時，於擦除階段發生 Flash 等待逾時將停止等待並立刻返回，建議依照 Flash 規格進行合適的時間設定。

注意：SBC-1 錄音僅支援 16k sample rate。

3.8.1.4 ADPCM



Sample Rate：設定 ADPCM Sample Rate，支援 8KHz / 10KHz / 12KHz。

DRC Type：同 SBC-1 同名設定。

EQ Type：同 SBC-1 同名設定。

Talk-Back Application：開啟自動應答功能。

High Threshold：偵測有語音的門檻值。高於此門檻且超過 Active Time，會被判定為有效的語音。

Low Threshold：偵測無語音的門檻值。低於此門檻且超過 Mute Time，會被判定為僅剩背景音，而停止錄音。

Record Head：錄音檔最前面的語音。當有效的語音被偵測，實際錄音的起始點，會再往前 Record Head 的時間。

Active Time：偵測有語音的時間。語音高於 High Threshold 且超過偵測時間，會被判定為有效的語音。

Mute Time：偵測無語音的時間。語音低於 Low Threshold 且超過偵測的時間，會被判定為僅剩背景音，而停止錄音。

Animal Talks：開啟動物音。

Maximum Syllables：音節的最大個數。

Realtime Erasing：同 SBC-1 同名設定。

Sector Erase Max Time：同 SBC-1 同名設定。

注意：

1. ***錄音功能需要搭配 SPI_Encoder 1.53 或更新的版本。***
2. ***錄音不能與播音(Voice / MIDI)同時。***
3. ***當 Q-Code 正在處理 SPI Flash 的擦除動作時，XIP 的程式會暫停大約 30 ~ 100ms(開啟隨擦即錄) / 100 ~ 200ms (未開啟隨擦即錄)左右 (依 SPI Flash 而定)。***
4. ***當擦除正在進行中，無法對 SPI Flash 進行操作。***
5. ***播放 SPI Flash 上的內容 (voice / melody / action ...)時，Q-Code 處理 SPI Flash 的擦除動作會造成播放不正常。***
6. ***開啟隨擦即錄功能時，無法使用琴鍵錄音功能。***

例.

[Record]

ADPCM_Realtime_Erasing = Enable

ADPCM_DRC_Type = TYPE2

ADPCM_EQ_Type = TYPE0

Record_ADPCM_SR = 8000

ADPCM_SE_Time = 470ms

ADPCM_AnimalTalks = Enable

ADPCM_AnimalTalks_Syllable = 20

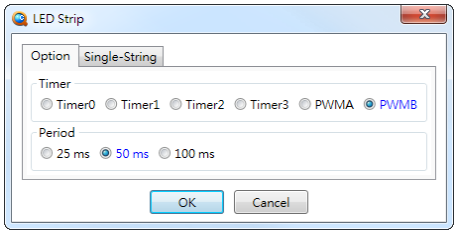
3.9 LED Strip

LED Strip 可透過輸出腳位控制多顆 RGB LED 所組成的燈串，燈串資料由 Vixen Lights 產生，副檔名為.csv，被控制的 RGB LED 須支援特定的通訊格式。

注意：LED Strip 的最低 Reset Time 需大於 60us。

3.9.1 Option

3.9.1.1 NX1

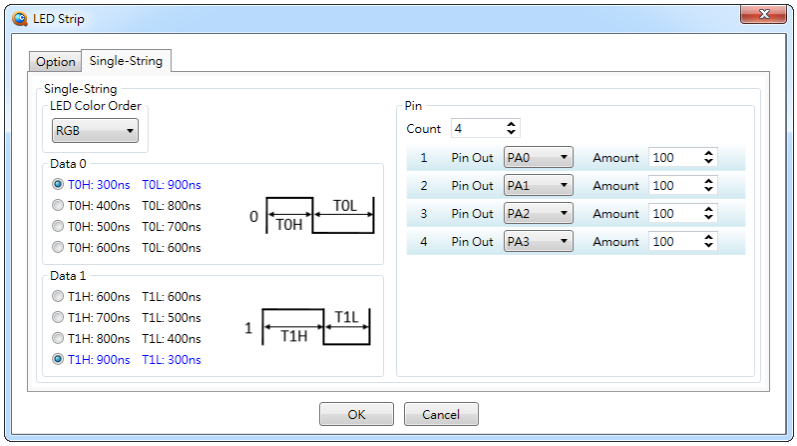


Timer：選擇計時器來源，需避免其他功能已使用的計時器。

Period：選擇燈串 LED 的更新週期。

3.9.2 Single-String

3.9.2.1 NX1



LED Color Order：燈串上 LED 的 RGB 排列順序。

LED Data 0：選擇燈串 Data 0 的通訊格式。

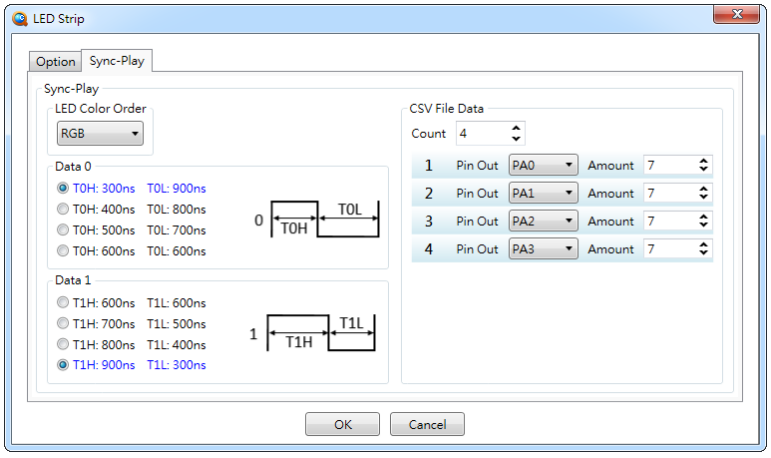
LDE Data 1：選擇燈串 Data 1 的通訊格式。

Pin：設定要使用 Single-String 功能的腳位，以及該腳位外接的燈串上的 LED 數量。所有腳位輸出的 LED 數量總數，當 Period 為 25ms 最多支援 399 顆，Period 為 50ms 最多支援 799 顆，Period 為 100ms 最多支援 1599 顆，另外，輸出腳位數量也會影響最大 LED 數量。

注意：LED 燈串的最低 Reset Time 需大於 60us。

3.9.3 Sync-Play

3.9.3.1 NX1



LED Color Order：燈串上 LED 的 RGB 排列順序。

LED Data 0：選擇燈串 Data 0 的通訊格式。

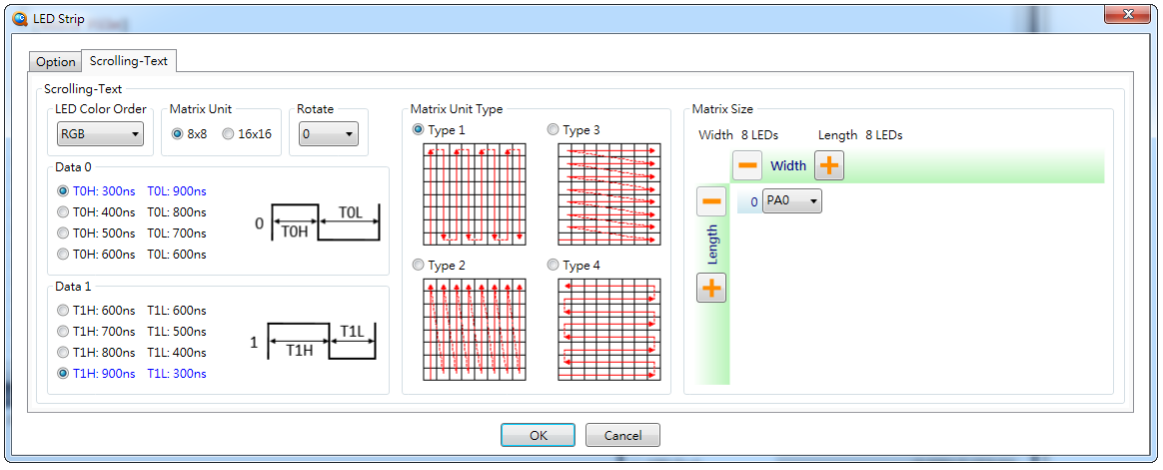
LDE Data 1：選擇燈串 Data 1 的通訊格式。

CSV File Data：設定要 CSV File 使用的腳位，以及該腳位燈串的 LED 數量。每個腳位最多支援 300 顆 LED 且必須使用同一個 Port 的腳位。

注意：LED 燈串的最低 Reset Time 需大於 60us。

3.9.4 Scrolling-Text

3.9.4.1 NX1



LED Color Order：燈串上 LED 的 RGB 排列順序。

Matrix Unit：選擇矩陣單元。

Rotate：矩陣單元的旋轉角度。

LED Data 0：選擇燈串 Data 0 的通訊格式。

LDE Data 1：選擇燈串 Data 1 的通訊格式。

Matrxi Unit type：選擇矩陣單元燈串排列的類型。

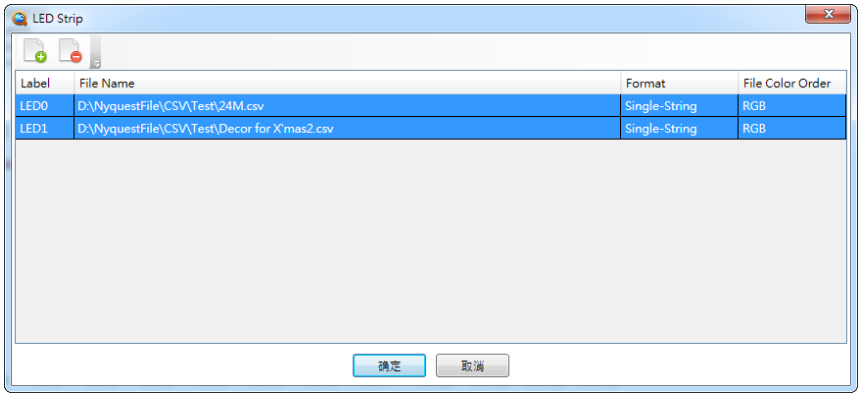
Matrix Size：選擇矩陣的長度和寬度，並設定每個矩陣單元資料輸出的腳位。最大的寬度為 256 個 LED 數量，最大的長度為 256 個 LED。必須使用同一個 Port 且連續的腳位。

注意：LED 燈串的最低 Reset Time 需大於 60us。

3.9.5 Add File

3.9.5.1 NX1

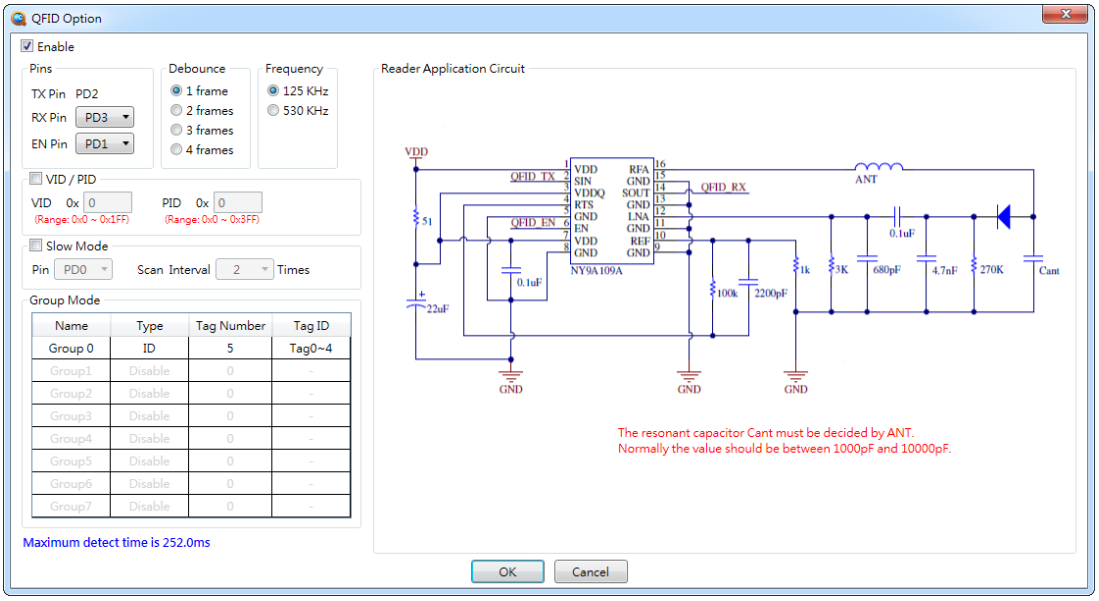
加入 .csv 和 .ledstr 檔案，File Color Order 指定檔案的 RGB 排列順序。



3.10 QFID

QFID 為九齊為了 RFID 應用所推出的功能，搭配九齊所提供的應用電路以及對應的 Tag 使用，即可進行 RFID 相關應用的開發。

3.10.1 NY5+ / NY7 / NX1



TX Pin：用來產生 QFID 載波使用。

- NY5+可選用 PA.0 / PB.0 / PC.0 / PD.0 / PE.0。
- NY7A 固定為 PB.2，NY7B 固定為 PD.2，NY7C 固定為 PF.2。

- NX1 OTP 固定為 PA.5。
- NX1 EF 可選用 PA.1 / PA.2 / PC.0 / PC.1。

RX Pin：接收 tag 回傳資料使用，使用者僅需將指定的 I/O 連接到 NY9A109A 的 SOUT 腳位即可。

En Pin：用來開關 NY9A109A 的功能，*Q-Code* 在掃描時會自動控制，使用者僅需將指定的 I/O 連接到 NY9A109A 的 EN 腳位即可。

Debounce：QFID 掃描時的 debounce 時間。NY5+ / NY7 可設為 1 / 2 / 3 / 4 frames，一併套用在 tag 辨識及移除。NX1 可分別設定辨識及移除時的 debounce 時間，支援 1 ~ 50ms。

Frequency：選擇 QFID 的載波頻率，可選為 125 KHz 或 530KHz。當選擇 530KHz 時，雖然需外加一顆 NY8A051D，但 Tag 可選擇用 PCB 天線製作，並在專案建置後，*Q-Code* 自動產生額外的 CarrierGen-530KHz.bin 供燒錄 NY8A051D 使用。

Timer：NX1 可選擇 QFID 功能要使用的系統計時器。

Slow Pin：QFID 的低速掃描的休息時間是利用指定 I/O 的 RC 充放電來計數，使用低速掃描前，須在指定 I/O 上並接一顆 100KΩ 電阻及 0.1uF 電容。

Scan Interval：QFID 的低速掃描會以掃描一次休息數次的方式動作，休息的次數由 Scan Interval 的選項決定；例如，Scan Interval 為 2 則表示掃描一次，休息兩次，假設掃描一次的時間為 200ms，則休息時間則約為 400ms，而掃描時間由設定的 Tag 數量決定，使用者亦可變更電阻及電容值以調整 Interval 單位時間，變更後單位時間計算公式如下。

$$Interval_{(new)} = (Interval_{(original)} \times R \times C) / 0.01$$

注意，休息時間為約略值，可能會有些許誤差。

QFID_GroupX_Mode：QFID 模式，ID 為僅辨識 Tag，ID+Input 為辨識 Tag 加上讀取 tag 上的 I/O 狀態。

QFID_GroupX_Tags：可使用的 tag 數量。QFID_GroupX_Mode 為 ID 時可使用 1~16，對應的 tag 為 Tag0 ~ Tag15。QFID_GroupX_Mode 為 ID+Input 時可使用 1~8，對應的 tag 為 Tag0 ~ Tag7。

QFID_VID：設定產品的 Vender ID，範圍為 0x0~0x1FF，如果 Tag 的 Vender ID 與 reader 的設定不同則該 tag 會無法辨識。

QFID_PID：設定產品的 Project ID，範圍為 0x0~0x3FF，如果 Tag 的 Project ID 與 reader 的設定不同則該 tag 會無法辨識。

QFID_VID / QFID_PID 是用來避免不同產品間的 Tag 互相干擾，tag 與 reader 設定的 VID / PID 必須完全相同才能辨識，使用者也可以選擇關閉 VID / PID 功能。

注意：

1. NX1 不支援 Slow Pin 與 Scankey Interval 選項。

2. NY5+ / NY7 僅支援一組 QFID Group。

[QFID] 段落中亦需定義相對應的觸發狀態。Tag 觸發型態是指當相對應的 Tag 被 reader 辨識到或從 reader 移除時所應該作出的反應，觸發的數目不可超過 QFID_GroupX_Tags 所設定的數量，且觸發事項之間必須以空格（TAB 或 Space）來分開。

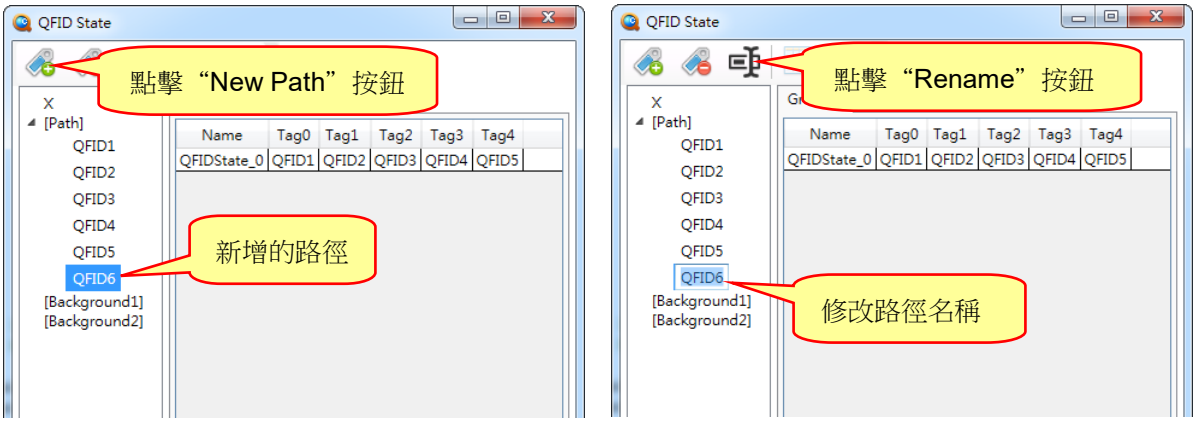
觸發事項如下表所示：

格式	Tag 觸發型態	註釋
X	忽略	無動作。
Path1	辨識	當 Tag 辨識成立時，執行 Path1 路徑。
/Path2	移除	當 Tag 移除時，執行 Path2 路徑。
Path1/Path2	辨識&移除	當辨識到 Tag 時，執行 Path1 路徑。 當 Tag 移除時，執行 Path2 路徑。

操作步驟：QFID→QFID State→Add State Name→Add PathName→OK，具體操作如下所示：

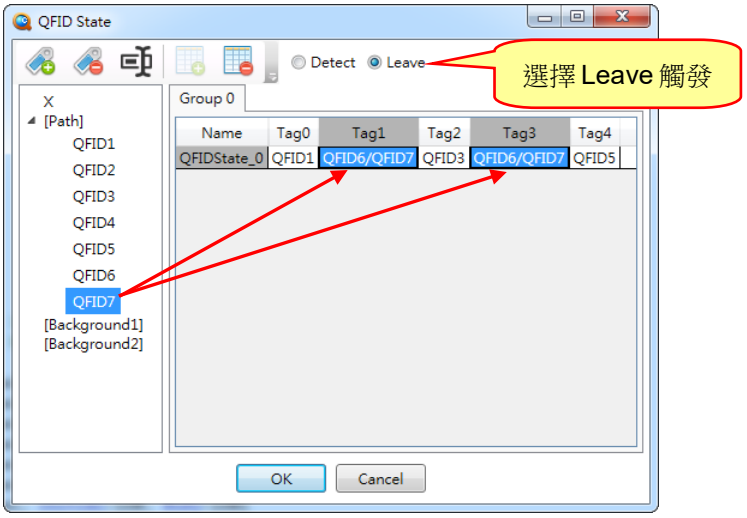
第一步：新增 **QFID State**，如下圖：

第二步：添加路徑或修改路徑名稱，如下圖：



第三步：修改對應路徑。系統已經預設路徑，使用者也可以根據需求修改對應路徑。如下圖的步驟：

如果要設定移除時的觸發方式，只需要在第 2 個動作的時候選擇 Leave 按鈕，如下圖所示：



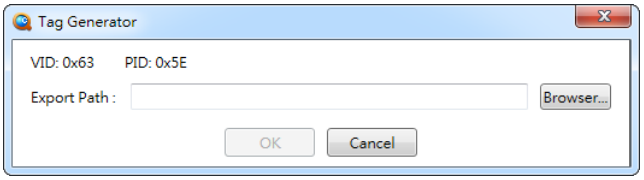
例.

[QFID]

Group0: TR1R X /TR3F TR4R/TR4F

QFID 功能僅能搭配 *Q-Code* 產生的 Tag .bin 檔使用，*Q-Code* 提供 Tag Generator 工具產生 Tag .bin 檔案，使用者可自行透過 *Q-Writer* 燒錄至 Tag IC。

使用方式如下：Tool→QFID Tag Generator



VID：對應的 QFID Vender ID。

PID：對應的 QFID Project ID。

Export Path：設定 Tag .bin 產生的路徑。

輸入完畢，按下 OK 按鈕，即會在 Location 所指定的路徑下產生 Tag .bin 檔案與 Tag 的應用電路圖。

Tag_Schematic_ID.jpg。

檔名格式：{VID_}{PID_}{TagID}.bin。

3.11 I/O

3.11.1 Matrix Key

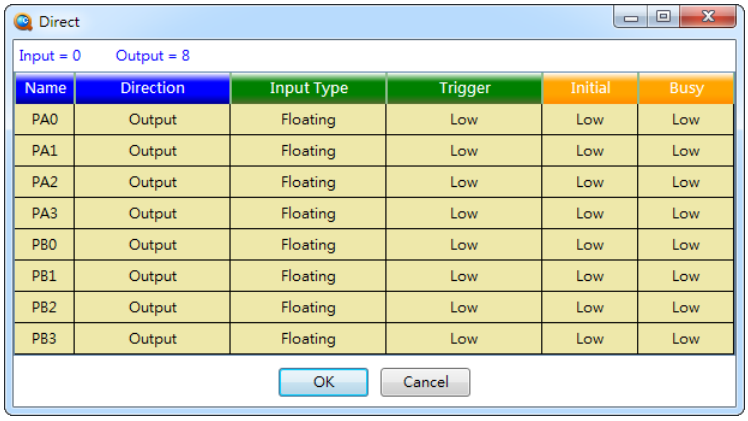
3.11.1.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NX1

點擊 I/O 中的 Matrix，在彈出的視窗中點擊+/-按鍵來設置 Matrix。點擊 Available Pins 按鍵來設置腳位。如下圖所示：

例. **Matrix =** [PA.0, PA.1] * [GND, PB.0, PA.2] ；使用 **GND** 來做 **Matrix** 按鍵掃描。

3.11.2 Direct Key

3.11.2.1 NY4



Direction：設定初始時 I/O 的輸出入方向。Input 初始時為輸入模式，Output 初始時為輸出模式。

Input Type：設定一般 I/O 的輸入模式中的上拉電阻。

- Pull-High 有上拉電阻的輸入模式（Input mode with Pull-High Resistor）。
- Floating 無上拉電阻的輸入模式（Input mode with Floating）。

Trigger：設定輸入腳位的觸發準位。

- Low 低準位觸發（Low Trigger）。
- High 高準位觸發（High Trigger）。

Initial：初始輸出電壓設定。

- Low 初始輸出電壓為（Initial Low）。
- High 初始輸出電壓為高準位（Initial High）。

Busy：Action 訊號的輸出方式。

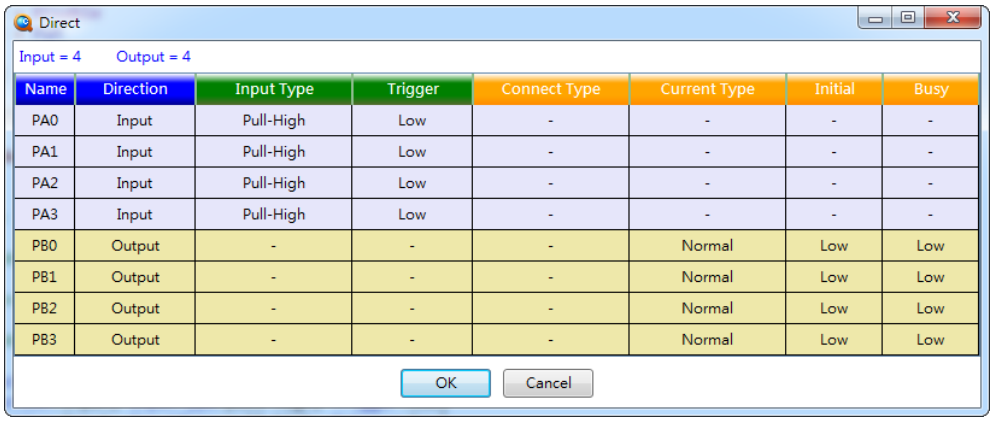
- High 設定 action 腳位的組態為 Busy High。
- Low 設定 action 腳位的組態為 Busy Low。

例.

PA.0 **=** [Direction: Input, InputType: Pull-high, Trigger: Low, Initial: Low, Busy: High]

PA.1 **=** [Direction: Output, InputType: Floating, Trigger: Low, Initial: Low, Busy: High]

3.11.2.2 NY5



Direction：設定初始時 I/O 的輸出入方向。Input 為輸入模式，Output 為輸出模式。

Input Type：設定一般 I/O 的輸入模式中的下拉電阻。

- Pull-High 有上拉電阻的輸入模式（Input mode with Pull-High Resistor）。
- Floating 無上拉電阻的輸入模式（Input mode with Floating）。
- IO Pull-High 有上拉電阻的 I/O 模式（I/O mode with Pull-High resistor）。
- IO Open-Drain 無上拉電阻的開汲閘 I/O 模式（Input mode without Pull-High resistor and with Open Drain）。

Trigger：設定輸入腳位的觸發準位。

- Low 低準位觸發（Low Trigger）。
- High 高準位觸發（High Trigger）。

Connect Type：設定外部元件的驅動方式。Sink 外部元件的驅動方式為 Active Low。

Current Type：設定電流輸出方式。

- Normal 一般電流輸出（Normal current output）。
- Large 大電流（Large Current output）。

Initial：初始輸出電壓設定。

- Low 初始輸出電壓為低準位（Initial Low）。
- High 初始輸出電壓為高準位（Initial High）。

Busy：Action 訊號的輸出方式。

- High 設定 action 腳位的組態為 Busy High。
- Low 設定 action 腳位的組態為 Busy Low。

注意：若 I/O 腳位要使用 input 模式，則只能將該 I/O 腳的 input mode 設成 Pull-High resistor。若是要使用 output，則只能將該 I/O 腳的 output mode 設成 Normal current output 和 Initial Low，且 Normal current output 僅提供 Sink output。若 I/O 腳位一直使用 input 模式而未

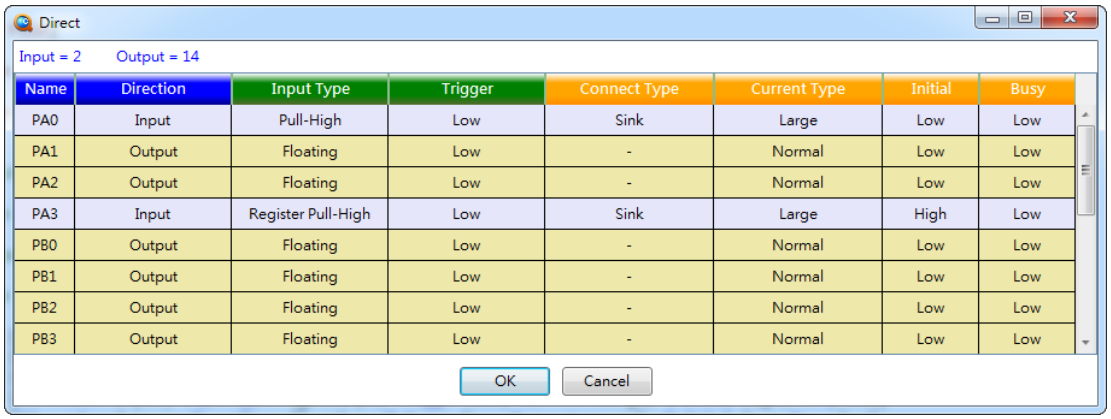
切換成 output 模式，按鍵一直壓住不放不能進入 Sleep mode。另外，IO Pull-High 的預設值（Current: Normal, Initial: Low）的初始狀態是 input 模式。這點請特別注意！

例.

PA.0 = [Direction: Input, InputType: Pull-high, Trigger: Low]

PA.1 = [Direction: Output, Current: Normal, Initial: Low, Busy: High]

3.11.2.3 NY5+



Direction：設定初始時 I/O 的輸出入方向。Input 初始時為輸入模式，Output 初始時為輸出模式。

Input Type：設定一般 I/O 的輸入模式中的下拉電阻。

- Pull-High 有上拉電阻的輸入模式（Input mode with Pull-High Resistor）。
- Floating 無上拉電阻的輸入模式（Input mode with Floating）。
- Register Pull-High 上拉電阻控制模式（Pull-High Resistor Control）。設定該模式後，上拉電阻可由 IO 暫存器來控制開啟或是關閉。

  例.

  [Path]

  TR1: PA=0xF ; **開啟 PA.0、PA.1 的上拉電阻。**

  TR2: PA=0x0 ; **關閉 PA.0、PA.1 的上拉電阻。**

Trigger：設定輸入腳位的觸發準位。

- Low 低準位觸發（Low Trigger）。
- High 高準位觸發（High Trigger）。

注意：InputType: Register Pull-High / Trigger: Low 模式下僅能接受低準位電壓來喚醒 IC。

Connect Type：設定外部元件的驅動方式。

- Sink 外部元件的驅動方式為 Active Low。
- Drive 外部元件的驅動方式為 Active High。

Current Type：設定電流輸出方式。

- Normal 一般電流輸出（Normal current output）。
- Large 大電流（Large Current output）。

Initial：初始輸出電壓設定。

- Low 初始輸出電壓為低準位（Initial Low）。
- High 初始輸出電壓為高準位（Initial High）。

Busy：Action 訊號的輸出方式。

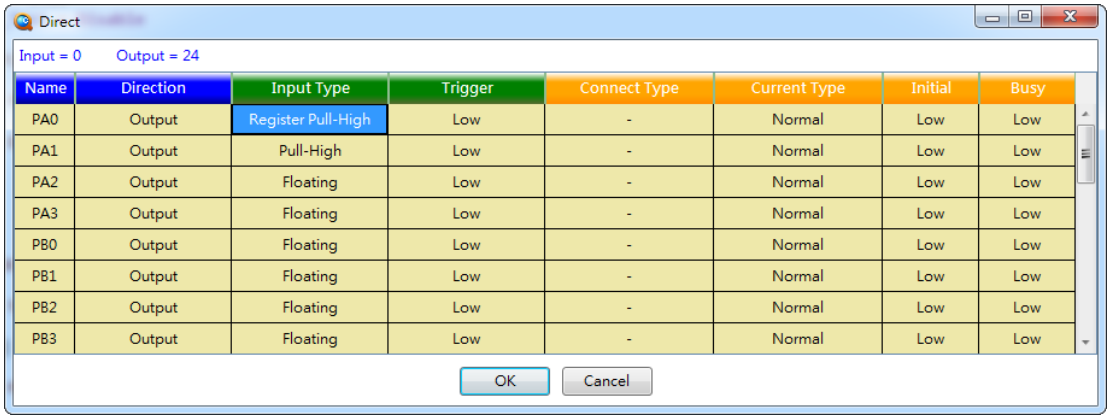
- High 設定 action 腳位的組態為 Busy High。
- Low 設定 action 腳位的組態為 Busy Low。

例.

PA.0 = [Direction:Input, InputType:Pull-high, Trigger:Low, Current:Large, Initial:Low, Busy:High]

PA.1 = [Direction:Output, InputType:Floating, Trigger:Low, Current:Normal, Initial:Low, Busy:High]

3.11.2.4 NY6 / NY7



Direction：設定初始時 I/O 的輸出入方向。Input 初始時為輸入模式，Output 初始時為輸出模式。

Input Type：設定一般 I/O 的輸入模式中的下拉電阻。

- Pull-High 有上拉電阻的輸入模式（Input mode with Pull-High Resistor）。
- Floating 無上拉電阻的輸入模式（Input mode with Floating）。
- Register Pull-High 上拉電阻控制模式（Pull-High Resistor Control）。設定該模式後，上拉電阻可由 IO 暫存器來控制開啟或是關閉。

  例.

  [Path]

  TR1: PA=0xF **；開啟 PA.0、PA.1 的上拉電阻。**

  TR2: PA=0x0 **；關閉 PA.0、PA.1 的上拉電阻。**

Trigger：設定輸入腳位的觸發準位。

- Low 低準位觸發（Low Trigger）。
- High 高準位觸發（High Trigger）。

注意：InputType: Register Pull-High / Trigger: Low 模式下僅能接受低準位電壓來喚醒 IC。

Connect Type：設定外部元件的驅動方式。

- Sink 外部元件的驅動方式為 Active Low。

- Drive 外部元件的驅動方式為 Active High。

Current Type：設定電流輸出方式。

- Normal 一般電流輸出（Normal current output）。
- Large 大電流（Large Current output）。
- Constant：定電流（Constant Sink current output）。

Initial：初始輸出電壓設定。

- Low 初始輸出電壓為低準位（Initial Low）。
- High 初始輸出電壓為高準位（Initial High）。

Busy：Action 訊號的輸出方式。

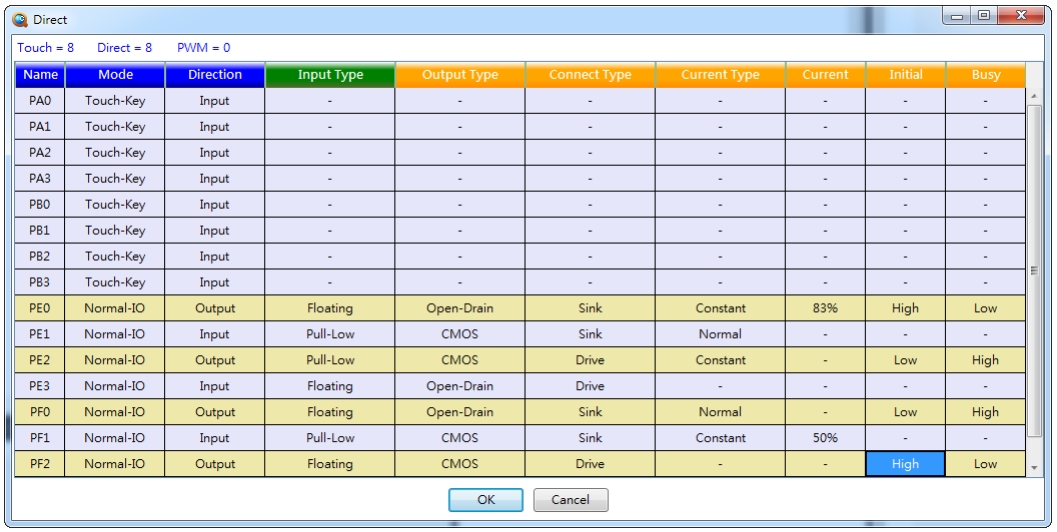
- High 設定 action 腳位的組態為 Busy High。
- Low 設定 action 腳位的組態為 Busy Low。

例.

PA.0 = [Direction:Input, InputType:RegisterPH, Trigger:Low, Current:Large, Initial:Low, Busy:High]

PA.1 = [Direction:Output, InputType:Large, Trigger:High, Current:Normal, Initial:High, Busy:Low]

3.11.2.5 NY9T



Name	Mode	Direction	Input Type	Output Type	Connect Type	Current Type	Current	Initial	Busy
PA0	Touch-Key	Input	-	-	-	-	-	-	-
PA1	Touch-Key	Input	-	-	-	-	-	-	-
PA2	Touch-Key	Input	-	-	-	-	-	-	-
PA3	Touch-Key	Input	-	-	-	-	-	-	-
PB0	Touch-Key	Input	-	-	-	-	-	-	-
PB1	Touch-Key	Input	-	-	-	-	-	-	-
PB2	Touch-Key	Input	-	-	-	-	-	-	-
PB3	Touch-Key	Input	-	-	-	-	-	-	-
PE0	Normal-IO	Output	Floating	Open-Drain	Sink	Constant	83%	High	Low
PE1	Normal-IO	Input	Pull-Low	CMOS	Sink	Normal	-	-	-
PE2	Normal-IO	Output	Pull-Low	CMOS	Drive	Constant	-	Low	High
PE3	Normal-IO	Input	Floating	Open-Drain	Drive	-	-	-	-
PF0	Normal-IO	Output	Floating	Open-Drain	Sink	Normal	-	Low	High
PF1	Normal-IO	Input	Pull-Low	CMOS	Sink	Constant	50%	-	-
PF2	Normal-IO	Output	Floating	CMOS	Drive	-	-	High	Low

Mode：設定腳位 I/O 模式。

- Normal-IO 通用輸出入腳位，由 Direction 參數決定該腳位為 Input 或是 Output。
- Touch-Key 觸摸鍵，Direction 固定為 Input。
- PWM-IO 輸出 PWM 腳位。

注意：PA.0 ~ PD.3 為觸摸鍵與一般 IO 共用腳，PE.0 ~ PF.3 為一般 IO 與 PWM-IO 共用腳。Touch-Key 或 PWM-IO 無法在程式中更改狀態。Normal-IO 可在程式中變更輸出入狀態。

Direction：設定初始時 Normal-IO 的模式。Input 初始時為輸入模式，Output 初始時為輸出模式。

Input Type：設定 Normal-IO 輸入模式的下拉電阻。

- Pull-Low 有上拉電阻的輸入模式（Input mode with Pull-Low Resistor）。

- Floating 無上拉電阻的輸入模式（Input mode with Floating）。

Output Type：設定 Normal-IO 的輸出狀態。

- CMOS 一般 CMOS 的 I/O 功能。
- Open-Drain 無上拉電阻的開漏極 I/O 模式。

Connect Type：設定 Normal-IO 外部元件的驅動方式。

- Sink 外部元件的驅動方式為 Active Low。
- Drive 外部元件的驅動方式為 Active High。

Current Type：設定電流輸出方式。

- Normal 一般電流輸出（Normal current output）。
- Large 大電流（Large Current output）。
- Constant 定電流（Constant Sink current output）。

Current：設定電流輸出能力，提供四種不同的電流輸出模式，僅在 Constant 與 Large 模式時有支援。

- 33% 提供 33%的電流輸出。
- 50% 提供 50%的電流輸出。
- 83% 提供 83%的電流輸出。
- 100% 提供 100%的電流輸出。

Initial：初始輸出電壓設定。

- Low：初始輸出電壓為低準位（Initial Low）。
- High：初始輸出電壓為高準位（Initial High）。

Busy：Action 訊號的輸出方式。

- High：設定 action 腳位的組態為 Busy High。
- Low：設定 action 腳位的組態為 Busy Low。

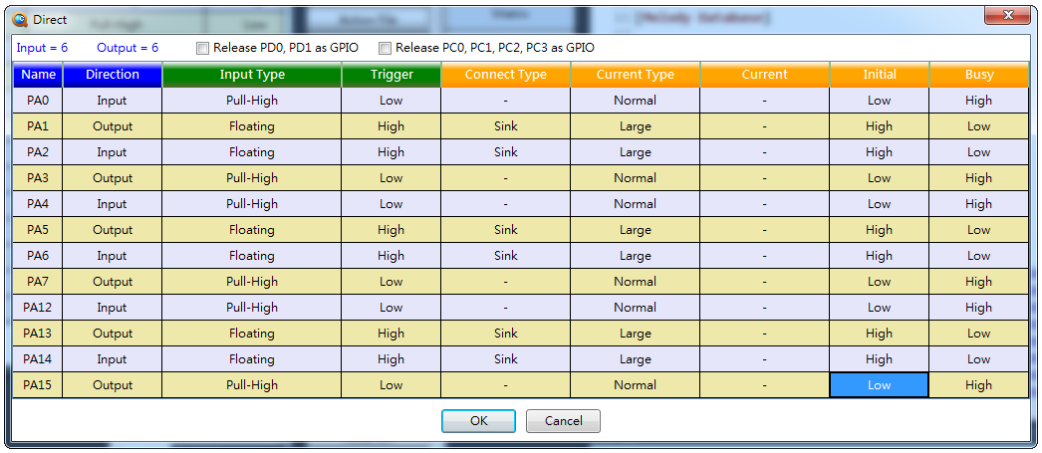
NY9T 腳位對於輸出電流的支援如下表：

	NY9T001A	***NY9T004A***	***NY9T008A***	***NY9T016A***
Constant Drive	PE.1 ~ PE.2	Not Available	PE.2	Not Available
Constant Sink	PE.0 ~ PE.2	PE.0 ~ PE.3	PE.0 ~ PF.3	PE.0 ~ PF.3

例.

```
PA.0 = [ Mode:Touch ]
PB.0 = [ Mode:Touch ]
PE.0 = [ Direction:Output, InputType:Pull-Low, OutputType:cmos, Connect:Sink,
         Current:constant_100%, Initial:Low, Busy:High ]
PE.1 = [ Direction:Input, InputType:Pull-Low, OutputType:cmos, Connect:Sink, Current:Normal,
         Initial:Low, Busy:High ]
PE.3 = [ Mode:PWM, Connect:sink, Current:constant_100% ]
```

3.11.2.6 NX1



Name	Direction	Input Type	Trigger	Connect Type	Current Type	Current	Initial	Busy
PA0	Input	Pull-High	Low	-	Normal	-	Low	High
PA1	Output	Floating	High	Sink	Large	-	High	Low
PA2	Input	Floating	High	Sink	Large	-	High	Low
PA3	Output	Pull-High	Low	-	Normal	-	Low	High
PA4	Input	Pull-High	Low	-	Normal	-	Low	High
PA5	Output	Floating	High	Sink	Large	-	High	Low
PA6	Input	Floating	High	Sink	Large	-	High	Low
PA7	Output	Pull-High	Low	-	Normal	-	Low	High
PA12	Input	Pull-High	Low	-	Normal	-	Low	High
PA13	Output	Floating	High	Sink	Large	-	High	Low
PA14	Input	Floating	High	Sink	Large	-	High	Low
PA15	Output	Pull-High	Low	-	Normal	-	Low	High

Direction：設定初始時 I/O 的輸出入方向。Input 初始時為輸入模式，Output 初始時為輸出模式。

Input Type：設定一般 I/O 的輸入模式中的下拉電阻。

- Pull-High 有上拉電阻的輸入模式（Input mode with Pull-High Resistor）。
- Floating 無上拉電阻的輸入模式（Input mode with Floating）。

Trigger：設定輸入腳位的觸發準位。

- Low 低準位觸發（Low Trigger）。
- High 高準位觸發（High Trigger）。

Connect Type：設定外部元件的驅動方式。

- Sink 外部元件的驅動方式為 Active Low。
- Drive 外部元件的驅動方式為 Active High。

Current Type：設定電流輸出方式。

- Normal 一般電流輸出（Normal current output）。
- Large 大電流（Large Current output）。
- Constant 定電流（Constant Sink current output）。

Current：設定電流輸出能力，提供四種不同的電流輸出模式，僅在 Constant 與 Large 模式時有支援。

- 33% 提供 33%的電流輸出。
- 50% 提供 50%的電流輸出。
- 83% 提供 83%的電流輸出。
- 100% 提供 100%的電流輸出。

Initial：初始輸出電壓設定。

- Low 初始輸出電壓為低準位（Initial Low）。
- High 初始輸出電壓為高準位（Initial High）。

Busy：Action 訊號的輸出方式。

- High 設定 action 腳位的組態為 Busy High。
- Low 設定 action 腳位的組態為 Busy Low。

例.

PA.0 = [Direction:Input, InputType:Pull-high, Trigger:Low, Current:Normal, Initial:High, Busy:Low]

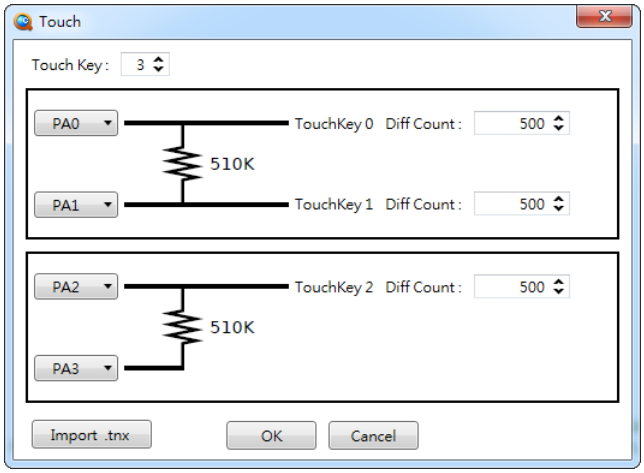
PA.1 = [Direction:Output, InputType:Floating, Trigger:High, Current:Large, Initial:Low, Busy:High]

PA.2 = [Direction:Input, InputType:Pull-high, Trigger:High, Current:Large, Initial:Low, Busy:High]

3.11.3 Touch

3.11.3.1 NX1

點擊 I/O 中的 Touch 設置 Touch Key 的數量和腳位。



Touch Key：設定 Touch Key 的數量。

Import .tnx：匯入 .tnx 檔案。

注意：Touchkey 中，每組 Touch 需要 2 個腳位，可以使用兩個 Touch Key。

例.

TouchKey = 3

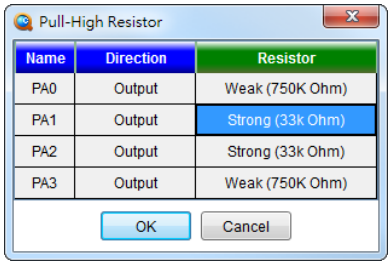
TouchGroup = [PIN0:PA.0, PIN1:PA.1, Diff0:500, Diff1:500] **; TouchKey0, TouchKey1**

TouchGroup = [PIN0:PA.2, PIN1:PA.3, Diff0:500] **; TouchKey2**

3.11.4 Pull-High Resistor

3.11.4.1 NY4

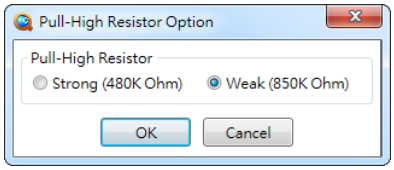
NY4 可針對每根腳位設定上拉電阻。Weak 為弱上拉 750k Ohm，Strong 為強上拉 33k Ohm。



例. **Input_Resistor** = [PA.0:weak, PA.1:Strong, PA.2: Strong, PA.3:weak]

3.11.4.2 NY5

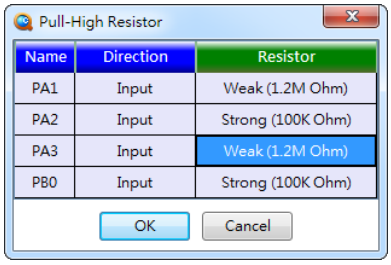
NY5 僅可使用全域的上拉電阻選項來設定電阻強度。Weak 所有腳位的上拉電阻皆為弱上拉 850k Ohm，Strong 所有腳位的上拉電阻皆為強上拉 480k Ohm。



例. **Input_Resistor** = strong

3.11.4.3 NY5+

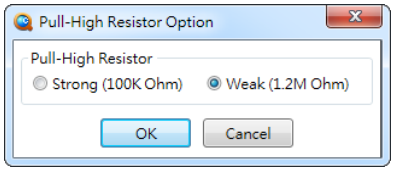
NY5+可針對每根腳位設定上拉電阻。Weak 為弱上拉 1.2M Ohm，Strong 為強上拉 100k Ohm。



例. **Input_Resistor** = [PA.0:weak, PA.1:Strong, PA.2:weak, PA.3:weak]

3.11.4.4 NY6

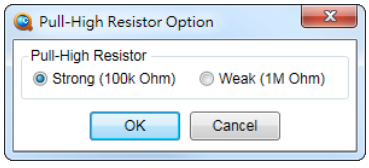
NY6 僅可使用全域的上拉電阻選項來設定電阻強度。Weak 所有腳位的上拉電組皆為弱上拉 1.2M Ohm，Strong 所有腳位的上拉電阻皆為強上拉 100k Ohm。



例. **Input_Resistor** = strong

3.11.4.5 NY7

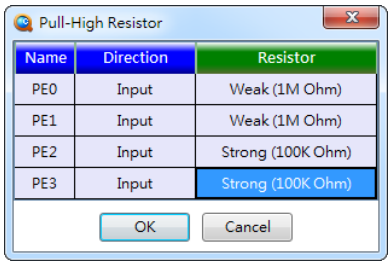
NY7 僅可使用全域的上拉電阻選項來設定電阻強度。Weak 所有腳位的上拉電組皆為弱上拉 1M Ohm，Strong 所有腳位的上拉電阻皆為強上拉 100k Ohm。



例. **Input_Resistor** = strong

3.11.4.6 NY9T

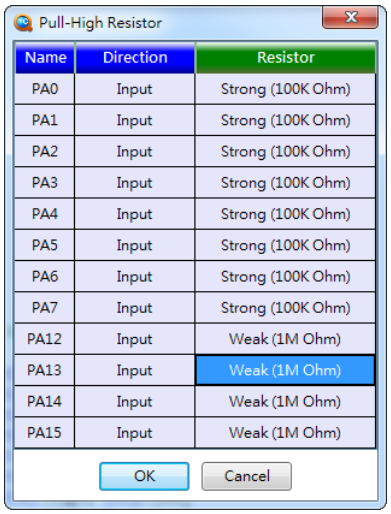
NY9T 可針對每根腳位設定上拉電阻。Weak 為弱上拉 1M Ohm，Strong 為強上拉 100k Ohm。



例. **Input_Resistor** = [PE.0:weak, PE.1:Strong, PE.2:weak, PE.3:weak]

3.11.4.7 NX1

NX1 可設定腳位上拉電阻，但 PA / PB / PC 的 0 ~ 7 腳位必須相同，PA / PB 的 8 ~ 15 腳位亦須相同。Weak 為弱上拉 1M Ohm，Strong 為強上拉 100k Ohm。

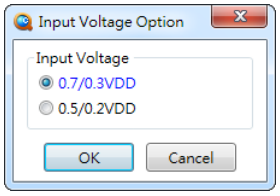


例. **Input_Resistor** = [PA.0:Strong, PA.1:Strong, PA.2:Strong, PA.3:Strong,
PA.4:Strong, PA.5:Strong, PA.6:Strong, PA.7:Strong,
PA.12:weak, PA.13:weak, PA.14:weak, PA.15:weak]

3.11.5 Input Voltage

3.11.5.1 NX1

設定輸入電壓準位。



例. **Input_Voltage** = VIH7_VIL3 **;高電位 0.7VDD, 低電位 0.3VDD**

3.12 Input State

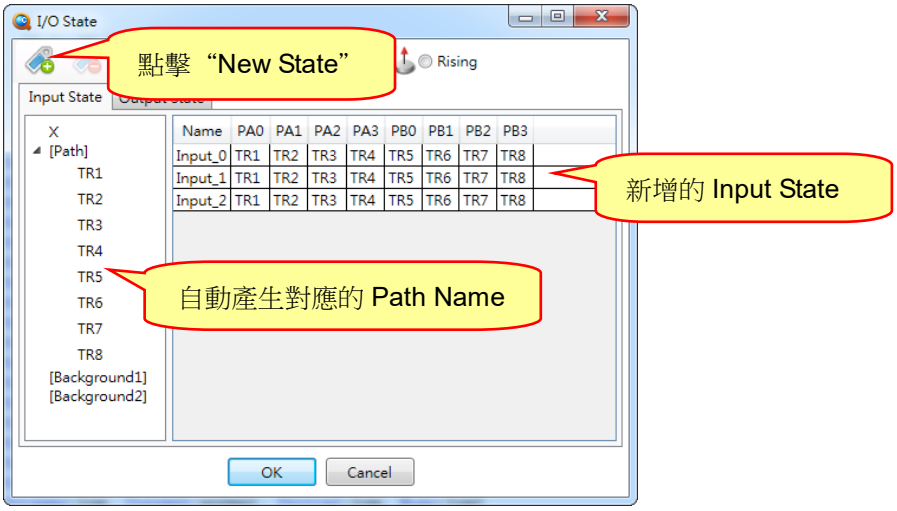
3.12.1 NY4 / NY5 / NY5+ / NY6 / NY7 / N9T / NX1

定義相對應的按鍵動作，最多可定義 255 組 input state。按鍵觸發型態是指當相應的輸入鍵被按下或釋放時所應該作出的反應，觸發的數目必須與輸入腳數相同，且觸發事項之間必須以空格（TAB 或 Space）來分開。觸發事項如下表所示：

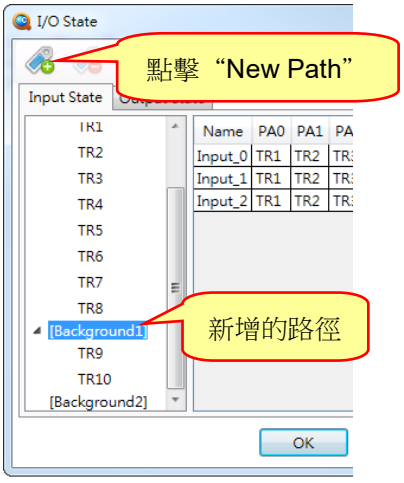
格式	按鍵觸發型態	註釋
X	忽略	無動作。
Path1	下降緣	當輸入腳從高電位到低電位時，執行 Path1 路徑。
/Path2	上升緣	當輸入腳從低電位到高電位時，執行 Path2 路徑。
Path1/Path2	下降緣&上升緣	當輸入腳從高電位到低電位時，執行 Path1 路徑。 當輸入腳從低電位到高電位時，執行 Path2 路徑。

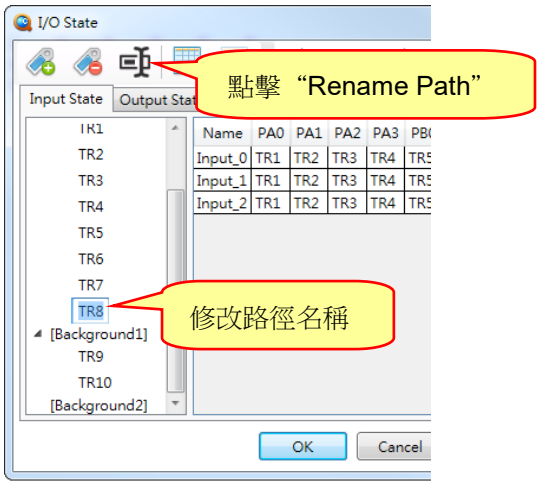
具體操作如下所示：

第一步：新增 **Input State**。

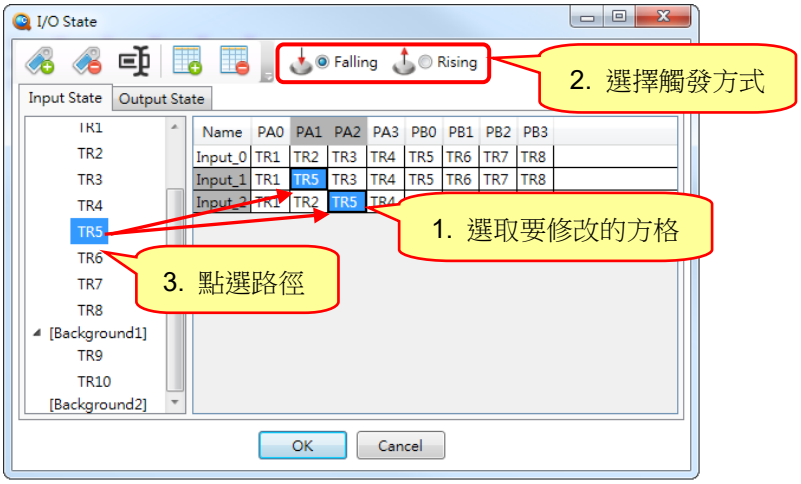


第二步：添加路徑或修改路徑名稱。

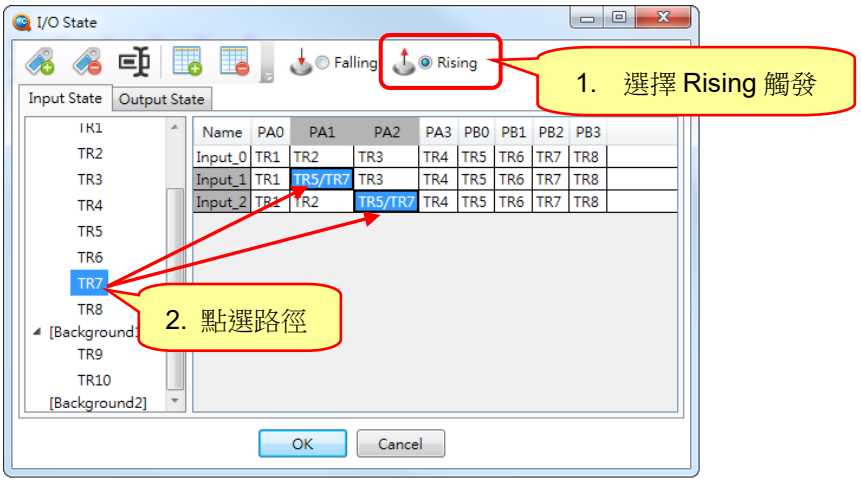




第三步：修改按鍵路徑。系統已經預設路徑，使用者也可以根據需求修改按鍵路徑。



如果想使用 Rising 觸發方式，只需要在第 2 個動作的時候選擇 Rising 按鈕，如下圖所示：



例.

[Input State]

Input_0: TR1R X /TR3F TR4R/TR4F

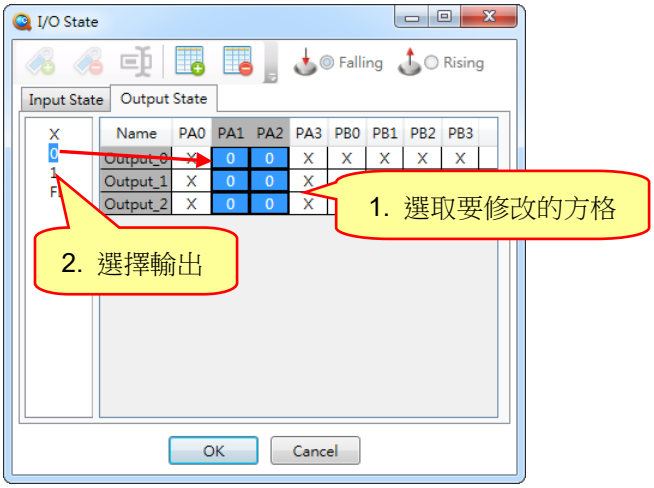
3.13 Output State

使用者可以在 **[Output State]** 段落中定義輸出的狀態。最多可定義 255 組 output state。在 **[Path]** 與 **[Background]** 都可以呼叫 **[Output State]** 中所定義的 output state。每個 output state 都由一個 output state 名稱，一個冒號":"及指定的輸出值組成。輸出值是指當該 output state 被呼叫時，每個輸出腳應該輸出的狀態值。輸出值的數目必須與輸出腳數相同，且輸出值之間必須以空格（TAB 或 space）分開。

具體操作步驟如下所示：

第一步：新增 **Output State**。

第二步：修改輸出訊號。



例.

[Output State]

OState0:	0	1	0	FD
OState1:	X	X	0	X
OState2:	0	1	0	X

3.13.1 NY4 / NY5

下列表格是 output state 的定義

輸出值	註釋
X	表示該腳位維持原來的狀態。
0	表示該腳位輸出低電位。
1	表示該腳位輸出高電位。
A	表示該腳位輸出 Action。（該功能僅對於播放整個 VIO 檔有效）
FD	表示隨播放的聲音音量大小變化。（Flash with Dynamic）
Q	表示該腳位跟隨 Quick-IO 的 QIO 訊號變化。

3.13.2 NY5+

下列表格是 output state 的定義

輸出值	註釋
X	表示該腳位維持原來的狀態。
0	表示該腳位輸出低電位。
1	表示該腳位輸出高電位。
A	表示該腳位輸出 Action（該功能僅對於播放整個 VIO 檔有效）。
P	表示該腳位輸出 PWM-IO（該功能僅對 PlayPWM 播放 MPIOx 時有效）。
FD	表示隨播放的聲音音量大小變化（Flash with Dynamic）。
Q	表示該腳位跟隨 Quick-IO 的 QIO 訊號變化。

3.13.3 NY6 / NY7

下列表格是 output state 的定義

輸出值	註釋
X	表示該腳位維持原來的狀態。
0	表示該腳位輸出低電位。
1	表示該腳位輸出高電位。
A	表示該腳位輸出 Action（該功能僅對於播放整個 VIO 檔有效）。
FD	表示隨播放的聲音音量大小變化（Flash with Dynamic）。

3.13.4 NY9T

下列表格是 output state 的定義

輸出值	註釋
X	表示該腳位維持原來的狀態。
0	表示該腳位輸出低電位。
1	表示該腳位輸出高電位。
A	表示該腳位輸出 Action（該功能僅對於播放整個 VIO 檔有效）。

3.13.5 NX1

下列表格是 output state 的定義

輸出值	註釋
X	表示該腳位維持原來的狀態。
0	表示該腳位輸出低電位。
1	表示該腳位輸出高電位。
FD	表示隨播放的聲音音量大小變化（Flash with Dynamic）。
Q	表示該腳位跟隨 Quick-IO 的 QIO 訊號變化。

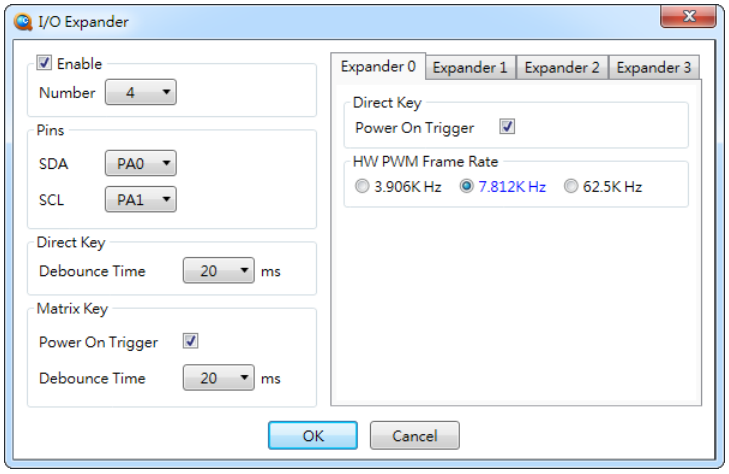
3.14 I/O Expander

3.14.1 I/O Expander

Q-Code 支援 I/O 擴展晶片功能，當 IC 腳位不足時，可透過 I/O 擴展晶片增加可用腳位。

3.14.1.1 NY5+ / NX1

可支援 4 組 I/O 擴展晶片。



Number：要使用的 I/O 擴展晶片數量。

SDA 腳位：連接至 I/O 擴展晶片的 SDA 腳位。

SCL 腳位：連接至 I/O 擴展晶片的 SCL 腳位。

Debounce Time：使用者可以選擇 Direct Key 和 Matrix Key 的 debounce 時間，範圍：1~1000ms。

Expander Power On Trigger：選擇 I/O 擴展晶片是否要開啟 Direct Key 和 Matrix Key 的 Power on Trigger 功能。當功能開啟後，如果使用者按著 I/O 擴展晶片上的按鍵後開機，則開機後會馬上去執行該按鍵對應的路徑。可以獨立設定每個 I/O 擴展晶片 Direct Key 是否要開啟 Power on Trigger 功能。

Expander HW PWM Frame Rate：設定 I/O 擴展晶片腳位作為硬體 PWM 輸出時的 Frame Rate。（預設 7.812kHz）

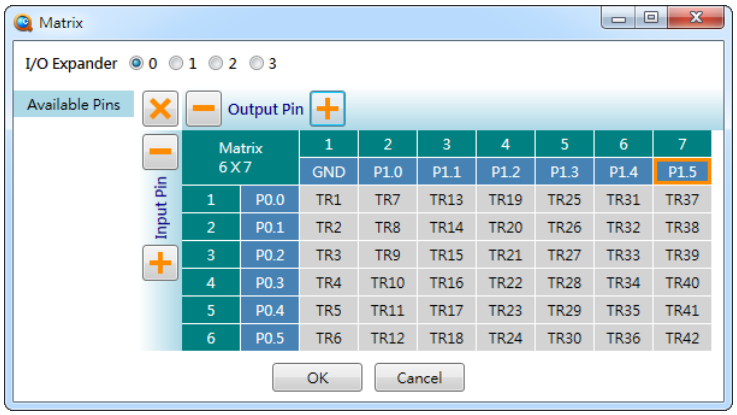
例.

```
IO_Expander_SDA = PA.0
IO_Expander_SCL = PA.1
IO_Expander0 = Enable
IO_Expander0_PowerOnTrigger = Enable
IO_Expander0_HW_PWM_FrameRate = 7812
IO_Expander_Direct_Debounce = 20ms
IO_Expander_Matrix_Debounce = 20ms
IO_Expander_Matrix_PowerOnTrigger = Enable
```

3.14.2 Matrix

設定 I/O 擴展晶片的 Mrtix 功能，僅支援一組 I/O 擴展晶片設置 Matrix。

3.14.2.1 NY5+ / NX1



Matrix 6X7		1	2	3	4	5	6	7
		GND	P1.0	P1.1	P1.2	P1.3	P1.4	P1.5
1	P0.0	TR1	TR7	TR13	TR19	TR25	TR31	TR37
2	P0.1	TR2	TR8	TR14	TR20	TR26	TR32	TR38
3	P0.2	TR3	TR9	TR15	TR21	TR27	TR33	TR39
4	P0.3	TR4	TR10	TR16	TR22	TR28	TR34	TR40
5	P0.4	TR5	TR11	TR17	TR23	TR29	TR35	TR41
6	P0.5	TR6	TR12	TR18	TR24	TR30	TR36	TR42

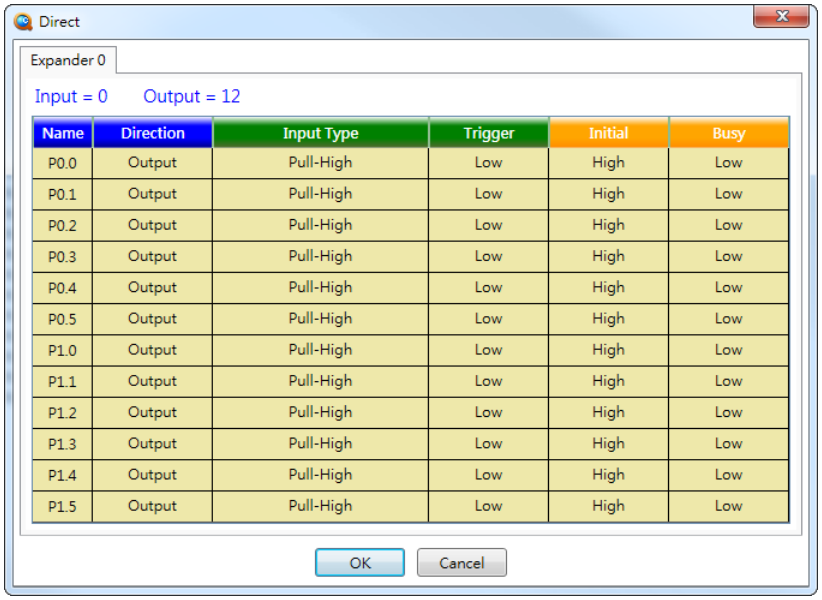
例.

IO_Expander0_Matrix = [EXP0_P0.0, EXP0_P0.1, EXP0_P0.2, EXP0_P0.3, EXP0_P0.4, EXP0_P0.5]*[GND, EXP0_P1.0, EXP0_P1.1, EXP0_P1.2, EXP0_P1.3, EXP0_P1.4, EXP0_P1.5]

3.14.3 Direct

設置對各個 I/O 擴展晶片的腳位。

3.14.3.1 NY5+ / NX1



Name	Direction	Input Type	Trigger	Initial	Busy
P0.0	Output	Pull-High	Low	High	Low
P0.1	Output	Pull-High	Low	High	Low
P0.2	Output	Pull-High	Low	High	Low
P0.3	Output	Pull-High	Low	High	Low
P0.4	Output	Pull-High	Low	High	Low
P0.5	Output	Pull-High	Low	High	Low
P1.0	Output	Pull-High	Low	High	Low
P1.1	Output	Pull-High	Low	High	Low
P1.2	Output	Pull-High	Low	High	Low
P1.3	Output	Pull-High	Low	High	Low
P1.4	Output	Pull-High	Low	High	Low
P1.5	Output	Pull-High	Low	High	Low

Direction：設定初始時腳位的輸出入方向。Input 初始時為輸入模式，Output 初始時為輸出模式。

Input Type：設定腳位的輸入模式中的上下拉電阻。

- Pull-High 有上拉電阻的輸入模式（Input mode with Pull-High Resistor）。
- Pull-Low 有下拉電阻的輸入模式（Input mode with Pull-Low Resistor）。

- Floating 無上下拉電阻的輸入模式（Input mode with Floating）。

Trigger：設定輸入腳位的觸發準位。

- Low 低準位觸發（Low Trigger）。
- High 高準位觸發（High Trigger）。

Initial：初始輸出電壓設定。

- Low 初始輸出電壓為低準位（Initial Low）。
- High 初始輸出電壓為高準位（Initial High）。

Busy：Action 訊號的輸出方式。

- High 設定 action 腳位的組態為 Busy High。
- Low 設定 action 腳位的組態為 Busy Low。

例.

EXP1_P0.0 = [Direction:Output, InputType:pull-high, Trigger:low, Initial:high, Busy:low]

3.14.4 Input State

開啟 I/O 擴展晶片功能後，可以在 **[Iutput State ExpID]** 段落中定義各個 I/O 擴展晶片對應的按鍵動作。*Q-Code* 會依據 I/O 擴展晶片使用的數量，自動產生相對應的段落。

按鍵觸發型態是指當相應的輸入鍵被按下或釋放時所應該作出的反應，觸發的數目必須與輸入腳數相同，且觸發事項之間必須以空格（TAB 或 Space）來分開。

3.14.4.1 NY5+ / NX1

NY5+最多可定義 255 組 input state，NX1 則無限制。UI 的操作方式同 Input State。觸發事項如下表所示：

格式	按鍵觸發型態	註釋
X	忽略	無動作。
Path1	下降緣	當輸入腳從高電位到低電位時，執行 Path1 路徑。
/Path2	上升緣	當輸入腳從低電位到高電位時，執行 Path2 路徑。
Path1/Path2	下降緣&上升緣	當輸入腳從高電位到低電位時，執行 Path1 路徑。 當輸入腳從低電位到高電位時，執行 Path2 路徑。

例.

[Input State Exp0]

InputExp0_0: TR1 TR2 TR3 TR4 TR5 TR6

3.14.5 Output State

開啟 I/O 擴展晶片功能後，使用者可以在 **[Output State ExpID]** 段落中定義各個 I/O 擴展晶片輸出腳的狀態。*Q-Code* 會依據 I/O 擴展晶片使用的數量，自動產生相對應的段落。

在 **[Path]** 與**[Background]** 都可以呼叫 **[Output State ExpID]** 中所定義的 output state。每個 output state 都由一個 output state 名稱，一個冒號":"及指定的輸出值組成。輸出值是指當該 output state 被呼

叫時，每個輸出腳應該輸出的狀態值。輸出值的數目必須與輸出腳數相同，且輸出值之間必須以空格（TAB 或 space）分開。

3.14.5.1 NY5+ / NX1

UI 的操作方式同 Output State。下列表格是 I/O 擴展晶片的 output state 可用的輸出狀態。

輸出值	註釋
X	表示該腳位維持原來的狀態。
0	表示該腳位輸出低電位。
1	表示該腳位輸出高電位。

例.

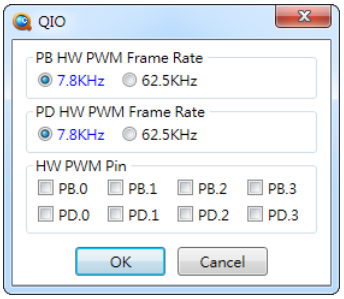
[Output State Exp0]

OutputExp0_0: **X X X X X X**

3.15 QIO

3.15.1 Configure

3.15.1.1 NY5+



PB HW PWM Frame Rate：Port B Hardware PWM 訊號的更新率。

PD HW PWM Frame Rate：Port D Hardware PWM 訊號的更新率。

HW PWM Pin：選擇輸出 Hardware PWM 訊號的腳位。

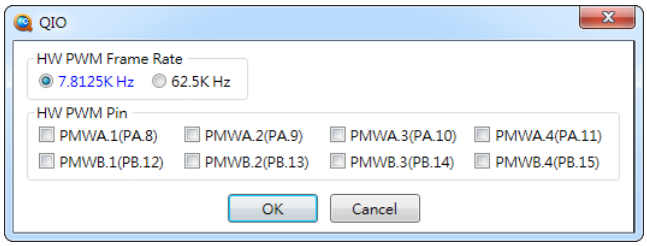
例.

HW_PWM_Pins = PB.1, PD.1

HW_PWM_PB_FrameRate = 7.8KHz

HW_PWM_PD_FrameRate = 7.8KHz

3.15.1.2 NX1



HW PWM Frame Rate：Hardware PWM 訊號的更新率。

HW PWM Pin：選擇輸出 Hardware PWM 訊號的腳位。

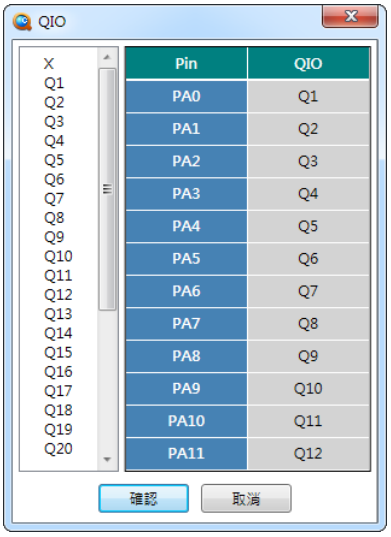
例.

HW_PWM_Pins = **PA**.9, **PA**.10

HW_PWM_FrameRate = 7812

3.15.2 QIO Table

設定 QIO 訊號與輸出腳的對映組態。僅支援一組設定。專案中只要輸出 QIO 即會套用此處的設定。



例.

[QIO]

MQIO = [PA.0: Q1, PA.1: Q2, PA.2:Q3, PA.3: Q4]

3.15.2.1 NY4

當 **[QIO]** 未設定時，QIO 信號與腳位的對映關係如下：

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
NY4A	PA.0	PA.1	PA.2	PA.3				
NY4B	PA.0	PA.1	PA.2	PA.3	PB.0	PB.1	PB.2	PB.3

3.15.2.2 NY5

當 **[QIO]** 未設定時，QIO 信號與腳位的對映關係如下：

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
PA.0	PA.1	PA.2	PA.3	PB.0	PB.1	PB.2	PB.3
Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16
PC.0	PC.1	PC.2	PC.3	PD.0	PD.1	PD.2	PD.3
Q17	Q18	Q19	Q20	Q21	Q22	Q23	Q24
PE.0	PE.1	PE.2	PE.3	PF.0	PF.1	PF.2	PF.3

3.15.2.3 NY5+

當 **[QIO]** 未設定時，QIO 信號與腳位的對映關係如下：

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
PA.0	PA.1	PA.2	PA.3	PB.0	PB.1	PB.2	PB.3
Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16
PC.0	PC.1	PC.2	PC.3	PD.0	PD.1	PD.2	PD.3

3.15.2.4 NX1

當 **[QIO]** 未設定時，QIO 信號與腳位的對映關係如下：

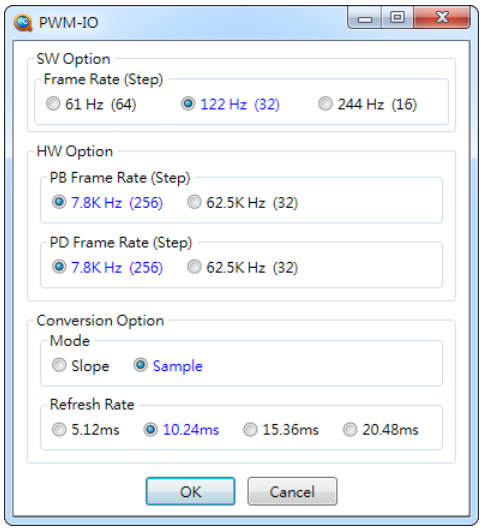
Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
PA.0	PA.1	PA.2	PA.3	PA.4	PA.5	PA.6	PA.7
Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16
PA.8	PA.9	PA.10	PA.11	PA.12	PA.13	PA.14	PA.15
Q17	Q18	Q19	Q20	Q21	Q22	Q23	Q24
PB.6	PB.7	PB.8	PB.9	PB.10	PB.11	PB.12	PB.13
Q25	Q26	Q27	Q28	Q29	Q30	Q31	Q32
PB.14	PB.15	PC.0	PC.1	PC.2	PC.3	PC.4	PC.5

3.16 PWM-IO

用於設定 PWM-IO 輸出的組態。設定好的輸出組態可以在 PlayPWM / PlayPWMS 及 PWMOut / PWMOutS 中使用。

3.16.1 Configure

3.16.1.1 NY5+



SW Frame Rate (Step)：當使用 PWMOut / PWMOutS 並設定非 PB 及 PD 接腳，才會依照此設定控制 PWM 週期。Frame Rate 越高則輸出較不易閃爍，但可用階數較少且系統的負載也高，Frame Rate 越低則有較高的階數，但較易出現閃爍。

HW PB Frame Rate (Step)：當使用 PlayPWM / PlayPWMS 及 PWMOut / PWMOutS 並設定 PB 接腳，會依照此設定控制 PWM 週期。Frame Rate 越高則輸出較不易閃爍但可用階數較少，Frame Rate 越低則輸出階數較多但較易出現閃爍。

HW PD Frame Rate (Step)：當使用 PlayPWM / PlayPWMS 及 PWMOut / PWMOutS 並設定 PD 接腳，會依照此設定控制 PWM 週期。Frame Rate 越高則輸出較不易閃爍但可用階數較少，Frame Rate 越低則輸出階數較多但較易出現閃爍。

Conversion Option：為資料轉換時的選項，僅適用於 PlayPWM / PlayPWMS 指令。

Mode：Slope 使用斜率方式做資料轉換，Sample 則是以對準取樣點的方式做資料轉換。當使用 Sample 模式時，耗用的 ROM 較多，而使用 Slope 模式時，耗用的 RAM 較多。

Refresh Rate：資料轉換時隔多久取樣一次。間隔越小，資料量越大。

例.

SW_FrameRate = 122Hz

PB_FrameRate = 7.8KHz

PD_FrameRate = 7.8KHz

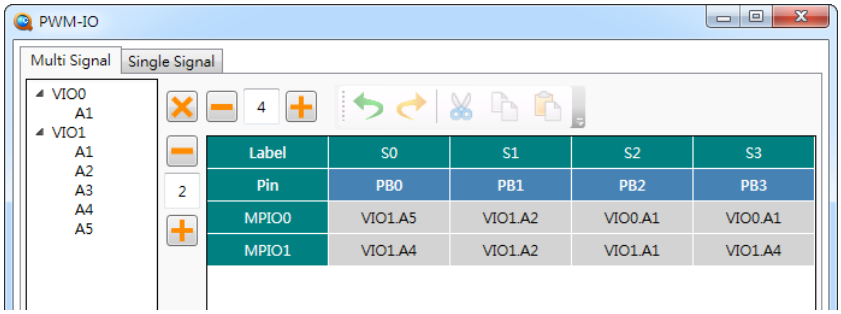
Conversion_Mode = Sample

Refresh_Rate = 10.24ms

3.16.2 Multi Signal

3.16.2.1 NY5+

可以設定多訊號的輸出組態。播放時會同時輸出至對應的腳位。輸出組態最少需包含 2 隻腳位，上方+/-可調整腳位數量。Multi Signal 當透過 PlayPWM 指令播放時，會依照此處的設定，將指定的訊號輸出至指定的腳位。



例.

[PWM-IO]

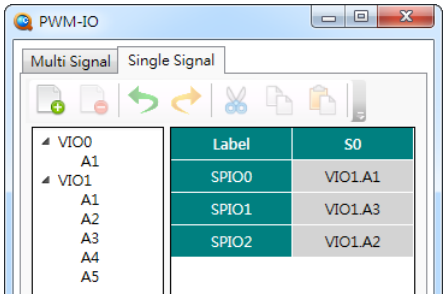
Multi_Pins = [PB.0, PB.1, PB.2, PB.3]

MPIO0 = [VIO0.A1, VIO0.A1, X, X]

3.16.3 Single Signal

3.16.3.1 NY5+

可以設定單一訊號的輸出組態。Single Signal 不指定輸出腳位，因此使用 PlayPWM 指令時，需要指定輸出腳位。



例.

[PWM-IO]

SPIO0 = [VIO0.A1]

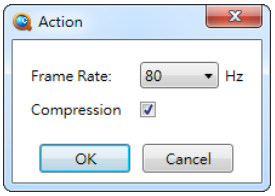
SPIO1 = [VIO1.A1]

3.17 Action

設定 Action 輸出的組態。設定好的輸出組態可以在 PlayA / PlayAS 中使用。

3.17.1 Configure

3.17.1.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T



Frame Rate：訊號每秒的更新頻率。Frame Rate 越高則輸出較不易閃爍，但可用階數較少且系統的負載也高。

Compression：開啟 Action 訊號壓縮功能。(預設值 Enable)

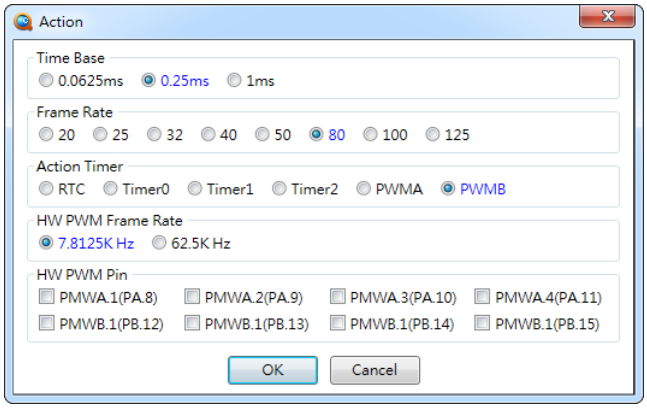
例.

[Action]

Compression = Enable

FrameRate = 80

3.17.1.2 NX1



Time Base：時間計數的最小單位，單位為毫秒(ms)。

Frame Rate：訊號每秒的更新頻率。Frame Rate 越高則輸出較不易閃爍，但可用階數較少且系統的負載也高。

Action Timer：設定 Action 使用的 Timer。

HW PWM Frame Rate：HW PWM 訊號每秒的更新頻率。

HW PWM Pin：設定使用 HW PWM 的腳位。

例.

[Action]

FrameRate = 80

TimeBase = 0.25ms

Timer = PWMB

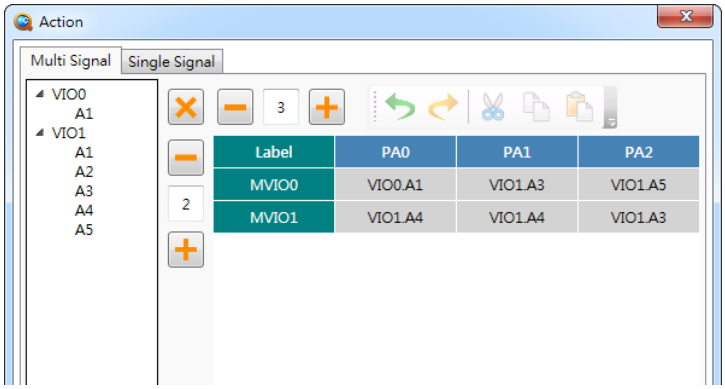
HW_PWM_FrameRate = 7812

HW_PWM_Pins = **PA**.8, **PA**.9

3.17.2 Multi Signal

3.17.2.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T

可以設定多訊號的輸出組態。播放時會同時輸出至對應的腳位。輸出組態最少需包含 2 隻腳位，上方 +/-可調整腳位數量。Multi Signal 當透過 PlayA 指令播放時，會依照此處的設定，將指定的訊號輸出至指定的腳位。



例.

[Action]

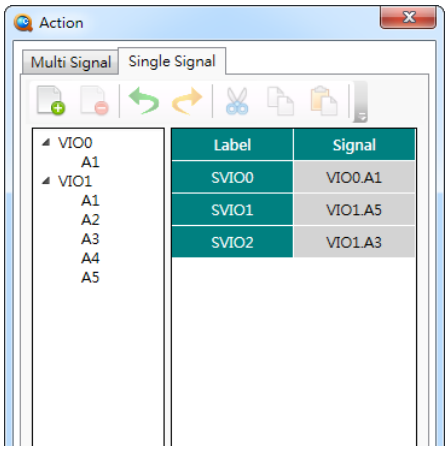
Multi_Pins = [PB.0, PB.1, PB.2, PB.3]

MVIO0 = [VIO0.A1, VIO0.A1, X, X]

3.17.3 Single Signal

3.17.3.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T

可以設定單一訊號的輸出組態。Single Signal 不指定輸出腳位，因此使用 PlayA 指令時，需要指定輸出腳位。



例.

[Action]

SVIO0 = [VIO0.A1]

SVIO1 = [VIO1.A1]

3.18 VR

語音辨識段落（VR，Voice Recognition）可引用 *Cyberon CSpotter Modeling Tool* 所產生的 .cvr 檔案（VR File），並設定語音辨識狀態（VR State）指定語音指令辨識成功所要執行的路徑。

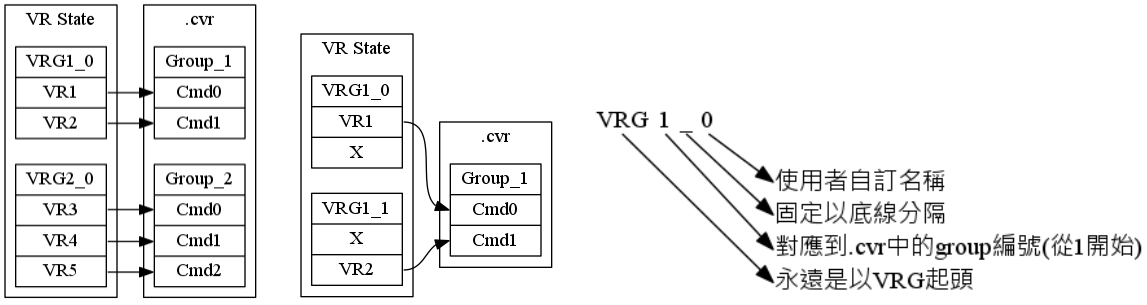
語音辨識檔案.cvr 檔案可選擇存放於 IC 內部的 ROM，亦可選擇存放於 SPI Flash。使用者可以在 VR File 選項中指定將.cvr 檔案存放於 IC 內部的 ROM 或外部 SPI Flash，兩者只能選擇其一。

除了離線建立語音指令，另外可以在 IC 運作時，線上訓練出語音指令，Voice Tag 提供這樣的功能。Voice Password 提供線上訓練語音指令並記錄聲紋特徵，訓練出來的語音指令，須使用相似的聲音才能成功辨識。

設定語音指令啟用狀態及其對應路徑的方式有兩種，VR State 與 VRGC state (VR Group Combo state)。*Cyberon CSpotter Modeling Tool* 專案中可以有多個 group，VR State 指定啟用其中一個語音指令群組(VR Group)，程式中可設定多個 VR State，每一個 VR State 都可以指定對應到不同的語音指令群組。VRGC state 則是設定啟用多個語音指令群組（最多三個群組），例如，如果設定結合 2 個 group，這兩個 group 的所有指令都會被判斷和辨識，VRGC 的所有群組的每個指令也都有對應的路徑。

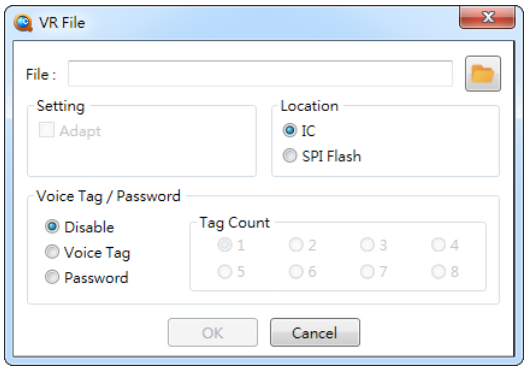
VR State 的名稱必須與.cvr 檔案的指令群組有所對應，例如 VRG1_0 的 1 代表第一個語音指令群組；VRG2_0 的 2 代表第二個語音指令群組。路徑名稱默認為 VR1、VR2、……依序編號，如果想關閉某條語音指令，可在 VR State 對應的路徑寫 X，代表不動作。VR State 名稱必須符合規範，以 VRG 起頭，跟著一數字代表指令群組編號，底線分隔後接著使用者自訂名稱。

當使用 VoiceTag / Password 模式時，可以使用語音指令群組 VTG，所對應的 VR State 名稱為 VTG_0、VTG_1、…。VTG 語音群組亦可使用於 VRGC state 中。



3.18.1 VR File

3.18.1.1 NX1



File：.cvr 檔的路徑。

Location：選擇 cvr 存放的位置，選擇放置於外部 SPI Flash 時，會複寫 .spiprj 檔的內容。(若 .spiprj 檔不存在，需先選擇要開啟的 .spiprj 檔)。選擇放置於 IC 內部會產生範例內容。

Adapt：選擇要不要使用 Adpat 功能。

Voice Tag / Password：選擇使用 Voice Tag 或 Password。

Tag Count：當使用 Voice Tag / Password 時選擇對應的語音指令群組所包含的指令數量。Voice Tag 最多 8 個，Password 最多 1 個。

例.

[VR]

VR File = C:\test.cvr

VR_Application = VoiceTag

VT_TagCount = 6

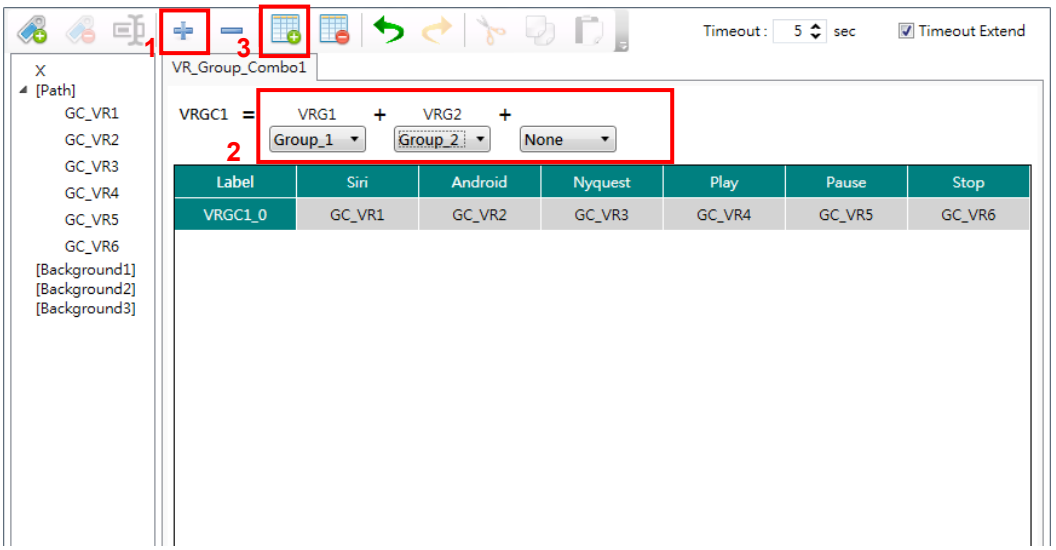
3.18.2 VR Combo

3.18.2.1 NX1

VRGC state 為多個 VR Group 的集合，必須先設定 VRGC 由哪些 VR Group 組成，然後指定所有群組的每一條語音指令所對應的路徑。不論語音指令是屬於哪一個群組，一旦被辨認成功都會立刻執行相對應的路徑。

編輯 VRGC State 流程：

1. 按下新增將自動產生名稱為 VRGC1 的 VR Group Combo。
2. 選擇 VR Group Combo 要結合的群組（最多三個群組）。
3. 按下新增將自動產生名稱為 VRGC1_0 的狀態，以下範例中，第一個群組有三個語音指令分別對應到 G1_VR1、G1_VR2、G1_VR3 三條路徑，第二個群組有三個語音指令分別對應到 G2_VR1、G2_VR2、G3_VR3 三條路徑。



注意：.cvr 檔必須有兩個以上的 Group 才能使用此功能。

VRGC Timeout 為計數逾時，分為兩個 VR Group Combo 和三個 VR Group Combo 來說明。

兩個 Group Combo：第一組 VR Group 中的指令成功辨識後，開始計數 timeout，直到指令 VRGC_Timeout_CLR 被呼叫後，停止計數。若計數時間到，仍未有呼叫 VRGC_Timeout_CLR，則跳到 timeout 路徑。

三個 Group Combo：第一組 VR Group 中的指令成功辨識後，開始計數 timeout，若第二組 VR Group 中的指令成功辨識後，會再重新開始計時，直到指令 VRGC_Timeout_CLR 被呼叫後，停止計數。若第一組 VR Group 中的指令沒有先成功辨識，反而是第二組 VR Group 中的指令先成功辨識，timeout 也會停止計數。

Timeout Extend 為延長 timeout，延長次數以一次為限。若 VRGC timeout 開始計數到時間結束期間，若有聲音超過 VAD 門檻，會自動延長一次。

例.

```
VRGC1 = VRG1 + VRG2              ; VRGC1 結合 VRG1 和 VRG2 兩個 group
VRGC1_0: G1VR1 G1VR2             ; VRG1 兩條語音指令的路徑。
         G2VR1 G2VR2             ; VRG2 兩條語音指令的路徑。
```

VRGC state 提供逾時機制。以上面的範例來說，若 VRG1(第一組 VR Group)之中的指令成功辨認，但並未在時限內辨認到 VRG2(第二組 VR Group)的指令，將會自動執行 VRGC_Timeout 路徑。如果第二組 Group 之中的指令先成功辨認，則 Timeout 條件不會成立，動作上一定要第一組 VR Group 之中的指令先成功辨識，VRGC_Timeout 路徑才會成立。逾時時間的長度單位可為秒(s)，最長可設為 60 秒。Timeout Extend 功能開啟後，當 VRG1(第一組 VR Group)之中的指令辨認成功後，若有聲音大於 VAD 閥值，則自動延長逾時時間的長度。

```
例. VRGC_Timeout = 5s            ; 指定 timeout 時間 5 秒。
例. VR_Timeout_Extend = Enable   ; 開啟自動延長逾時時間功能。
```

VRGC state 也可用來訂定指令群組被辨識的順序，當特定指令被辨識後才可觸發真正指令。例如：“Hi Siri, what time is it?”，Hi Siri 可作為第一個群組指令，當被辨識後才接受第二個群組指令 what time is it?。程式撰寫上可使用旗標變數紀錄第一個群組指令是否已被辨識的狀態，如以下範例。

例.

```
[VR]
VRGC_Timeout = 5s                ; 指定 timeout 時間 5 秒。
VRGC1 = VRG1 + VRG2              ; VRGC1 結合 VRG1 和 VRG2 兩個 group。
                                 ; VRG1 為前置詞的 group。
                                 ; VRG2 為後續指令的 group。
VRGC2 = VRG1 + VRG3              ; VRGC2 結合 VRG1 和 VRG3 兩個 group。
VRGC1_0: G1_VR1 G1_VR2           ; VRG1 的 state 中定義所有指令的 path。
         G2_VR1 G2_VR2           ; 第二個群組指令對應的 path。

[Variable]
```

VAR8: flag

[Path]

PowerOn: VRGC1_0 ; 啟用 **VRGC1_0** 狀態。

G1_VR1: flag = 1 ; 前置詞出現，設定 **flag**。

G1_VR2: flag = 1 ; 前置詞出現，設定 **flag**。

G2_VR1: flag = 1 ? G2_VR1_True ; 如果有前置詞才執行路徑。

G2_VR2: flag = 1 ? G2_VR2_True ; 如果有前置詞才執行路徑。

VRGC_Timeout: flag = 0 ; 如果講了前置詞，卻沒有後續指令，時間到會執行 **;VRGC_Timeout** 路徑讓使用者可以重設狀態。

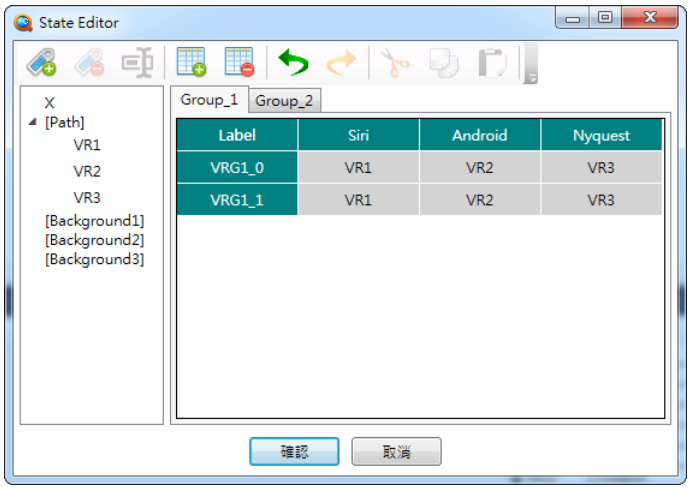
G2_VR1_True: flag = 0, VRGC_Timeout_CLR, PlayV(ch0, $V1) ; 當被辨識到就播放聲音。

G2_VR2_True: flag = 0, VRGC_Timeout_CLR, PlayV(ch0, $V1) ; 當被辨識到就播放聲音。

3.18.3 VR State

3.18.3.1 NX1

VR State 編輯器。



例.

[VR]

VR File = C:\test.cvr

VRG1_0: VR1 VR2

VRG2_0: VR3 VR4 VR3

[Path]

PowerOn: VRG1_0, VR_On

VR1: PlayV(ch0, $V0)

VR2: PlayV(ch0, $V1)

VR3: PlayV(ch0, $V2)

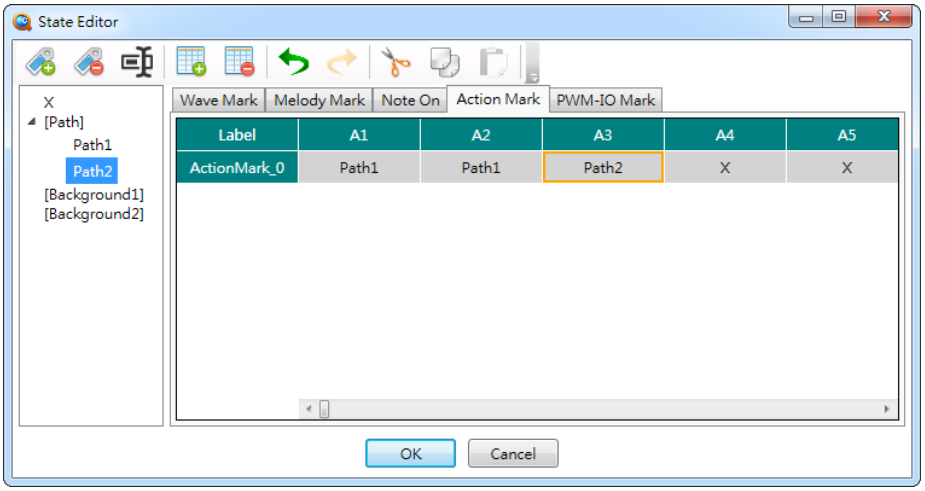
VR4: PlayV(ch0, $V3)

3.19 Action Mark

3.19.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1

當使用者使用 Q-Visio 編輯檔案時，可以隨意在 action 檔案中插入 Mark 標記。當 Q-Code 讀到標記時，會依照當前的 Action Mark 狀態自動執行對應的路徑。使用者可以在 **[Action Mark]** 段落中定義 Action Mark 的狀態。

Action Mark 的標記編號最多可有 255 個，從 A1 至 A255。在 *Q-Code* 中，使用者僅需要加入.vio 檔及定義好標記對應的路徑即可。最多可以定義 255 組設定。



注意：

1. ***Action Mark 狀態的切換，等同 Input State 的用法。***
2. ***標記編號必須為 A1~A255。***

例. 在 Q-Visio 中，插入 A1~A4 的碼。

```
[Action Mark]
;                A1      A2      A3      A4
ActionMark1:     AM1     AM2     AM3     AM4

[Path]
PowerOn: ActionMark1, PlayA(PA.0, Ch1, $A0)

[Background1]
AM1: PB=0x1                  ; 當讀取到 Action Mark，且 Mark 編號為 1，即會執行 AM1。
AM2: PB=0x2                  ; 當讀取到 Action Mark，且 Mark 編號為 2，即會執行 AM2。
AM3: PB=0x3                  ; 當讀取到 Action Mark，且 Mark 編號為 3，即會執行 AM3。
AM4: PB=0x0                  ; 當讀取到 Action Mark，且 Mark 編號為 4，即會執行 AM4。
```

3.20 Wave Mark

3.20.1 NY4 / NY5 / NY5+ / NX1

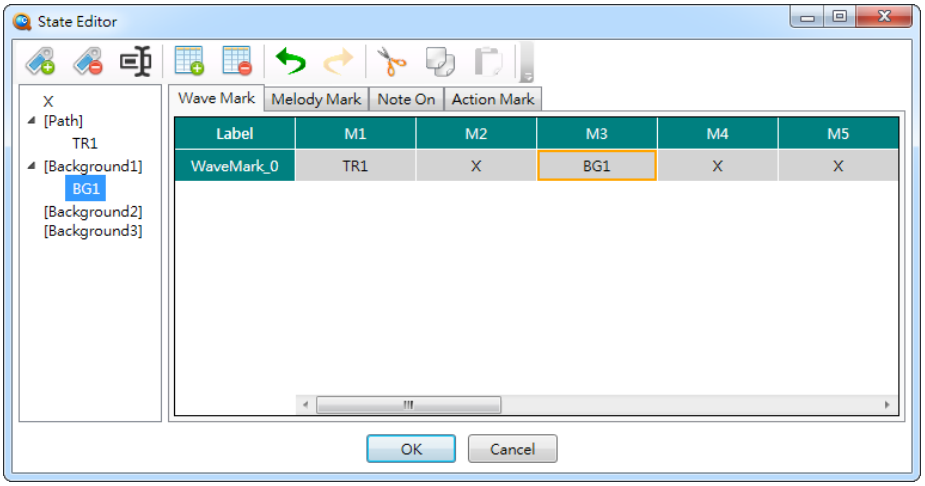
當使用者使用 *Quick-IO* 來編輯檔案時，使用者可以隨意在 Voice 檔案的任何地方插入 Mark 標記。當 *Q-Code* 讀到標記時，會依照當前的 Wave Mark 狀態自動執行對應的路徑。使用者可以在 Wave Mark 段落中定義 Wave Mark 的狀態。

Wave Mark 的標記編號，如 M1；此標記編號最多可有 255 個，從 M1 至 M255。在 *Q-Code* 中，使用者僅需要加入.nyq 檔及定義好標記對應的路徑即可。最多可以定義 255 組設定。

注意：

1. Wave Mark 狀態的切換，等同 Input State 的用法。

2. 標記編號必須為 M1~M255。

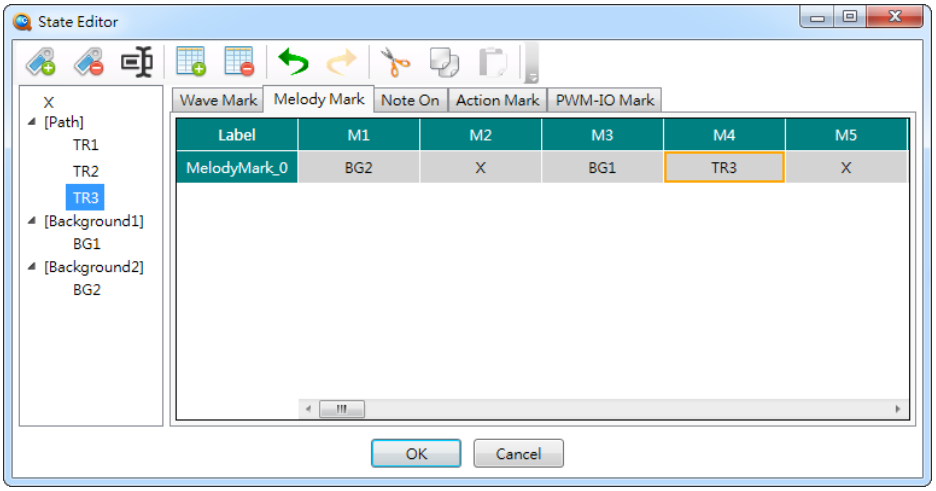


例. 使用 *Quick-IO* 在 Wave 檔中，插入 M1~M4 的碼。

```
[Wave Mark]
;              M1      M2      M3      M4
WaveMark_0:    WM1     WM2     WM3     WM4

[Path]
PowerOn: WaveMark_0, PlayV($V0)

[Background1]
WM1: PB=0x1                    ; 當 QIO 發生，且 Mark 編號為 1，即會執行 WM1。
WM2: PB=0x2                    ; 當 QIO 發生，且 Mark 編號為 2，即會執行 WM2。
WM3: PB=0x3                    ; 當 QIO 發生，且 Mark 編號為 3，即會執行 WM3。
WM4: PB=0x0                    ; 當 QIO 發生，且 Mark 編號為 4，即會執行 WM4。
```

3.21 Melody Mark

3.21.1 NY5 / NY5+ / NY6 / NY7 / NX1



當用 *CakeWalk* 來編輯 MIDI 音樂檔時，使用者可以隨意在任何一個音符中插入控制碼來控制輸出腳的狀態或執行其它操作。播放 Melody 檔案時，按照 Melody 檔案中的標記來執行相應的馬達旋轉或 LED 閃燈動作。

Melody Mark 動作的標記編號，如 M1；與 *CakeWalk* 或 *Q-MIDI* 中插入的控制碼相對應；Melody Mark 頁面中標記編號最多可有 255 個，從 M1 至 M255，每一標記編號有其相對應的路徑。當定義了 Melody Mark 後，使用者僅需要在 MIDI 檔案中插入相應的控制碼來啟動 Melody Mark 功能。最多可以定義 255 組設定。

注意：

1. Melody Mark 狀態的切換，等同 Input State 的用法。

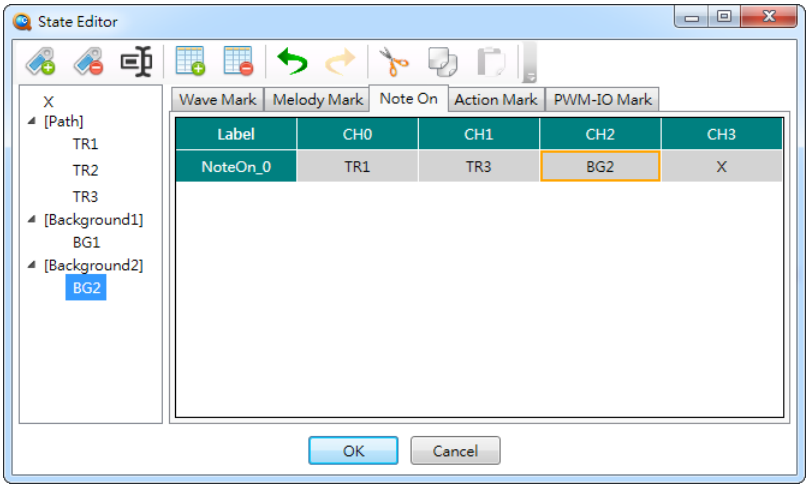
2. 標記編號必須為 M1~M255。

例. 在 *CakeWalk* 或是 *Q-MIDI* 中，插入 M1~M4 的碼。

```
[Melody Mark]
;                       M1      M2      M3
NewMelodyMark1:         MM1     MM2     MM3
; 當讀取到 Melody Mark 時，且 Mark 編號為 1，即會執行 MM1。
; 當讀取到 Melody Mark 時，且 Mark 編號為 2，即會執行 MM2。
; 當讀取到 Melody Mark 時，且 Mark 編號為 3，即會執行 MM3。

[Path]
PowerOn: NewMelodyMark1, PlayM($M0)

[Background1]
MM1: PB=0x1
MM2: PB=0x2
MM3: PB=0x3
```

3.22 Note On

3.22.1 NY5 / NY5+ / NY6 / NY7 / NX1



當播放 Melody 檔案時，可以不需要使用 *Q-MIDI* 來對每個 MIDI 通道的音符進行插碼，來達到節省 ROM Size 的目的。使用 **[Note On]** 段落可直接在每個 MIDI 通道讀取音符後，執行該 MIDI 通道所對應的動作。最多可以定義 255 組設定。

注意：

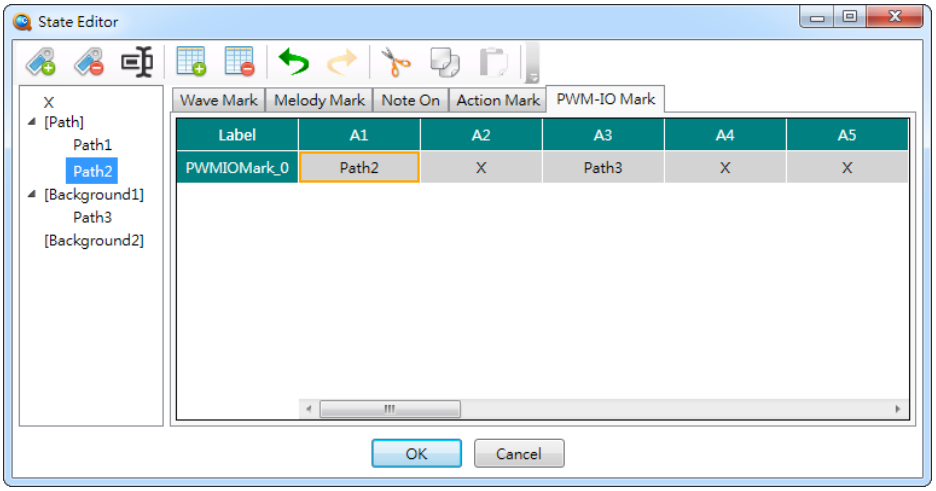
1. ***Note On 狀態的切換，等同 Input State 的用法。***
2. ***NY5 系列支援 CH0 ~ CH3，NY5+系列支援 CH1 ~ CH4 和 CH10，NY6 系列支援 CH1 ~ CH6 和 CH10，NY7 / NX1 系列支援 CH1 ~ CH16。***
3. ***若是在 [Note On] 中，直接執行 play 程序，則有可能會中斷當前的 Melody 播放。***
4. ***休止符不會動作。***

例.

```
[Note On]
;          CH1        CH2        CH3        CH4
NoteOn1:   BG1        X          X          X
NoteOn2:   X          BG2        X          X

[Path]
TR1: PlayM($M0)                              ; 播放 Melody。
TR2: SWITCH(R0)=[NO1, NO2], R0=0, TR2        ; 切換 Note On 狀態。
NO1: NoteOn1, R0=1                           ; 執行 NoteOn1。
NO2: NoteOn2, R0=0                           ; 執行 NoteOn2。

[Background1]
BG1:PB=[ X X X 1], DELAY(0.08), PB=[ X X X 0]

[Background2]
BG2:PB=[ X X 1 X], DELAY(0.08), PB=[ X X 0 X]
```

3.23 PWM-IO Mark

3.23.1 NY5+



當使用者使用 Q-Visio 編輯檔案時，可以隨意在 action 檔案中插入 Mark 標記。當 Q-Code 使用 PlayPWM 播放時讀到標記，會依照當前的 PWM-IO Mark 狀態自動執行對應的路徑。使用者可以在 **[PWM-IO Mark]** 段落中定義 PMW-IO Mark 的狀態。

PWM-IO Mark 的標記編號最多可有 255 個，從 A1 至 A255。最多可以定義 255 組設定。

注意：

1. PWM-IO Mark 狀態的切換，等同 Input State 的用法。

2. 標記編號必須為 A1~A255。

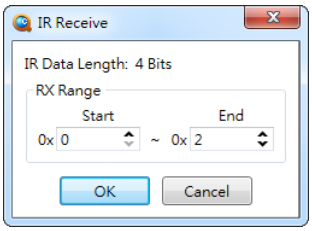
例. 在 Q-Visio 中，插入 A1~A4 的碼。

```
[PWM-IO]
SPIO0 = [VIO0.A1]

[PWM-IO Mark]
;                A1      A2      A3      A4
PWMIOMark_0:  PM1     PM2     PM3     PM4

[Path]
PowerOn: PWMIOMark_0, PlayPWM(PA.0, Ch1, $SPIO0)

[Background1]
PM1: PB=0x1                ; 當讀取到 PWM-IO Mark，且 Mark 編號為 1，即會執行 PM1。
PM2: PB=0x2                ; 當讀取到 PWM-IO Mark，且 Mark 編號為 2，即會執行 PM2。
PM3: PB=0x3                ; 當讀取到 PWM-IO Mark，且 Mark 編號為 3，即會執行 PM3。
PM4: PB=0x0                ; 當讀取到 PWM-IO Mark，且 Mark 編號為 4，即會執行 PM4。
```

3.24 IR Receive

3.24.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NX1

編寫 IR 接收指令。必須先在 Option 段落中選擇需要的 IR Mode（紅外線發射或接收功能，編碼長度與載波頻率）。



使用者只能在 **[Option]** 段落中選擇 IR 資料的編碼長度，視窗會顯示先前所選擇的資料長度。

1. IR 資料為 4-bit 編碼時，RX 的代碼範圍為：0x0~0xF。
2. IR 資料為 8-bit 編碼時，RX 的代碼範圍為：0x00~0xFF。
3. IR 資料為 12-bit 編碼時，RX 的代碼範圍為：0x000~0xFFF。
4. IR 資料為 16-bit 編碼時，RX 的代碼範圍為：0x0000~0xFFFF。

例.

[IR Receive]

Code[0x0]: **PlayV**(CH0, $V0)

Code[0x1]: **PlayV**(CH0, $V1)

注意：要發射 IR，只要在 Option 選項中選擇 Enable IR Tx mode，然後在 Path 段落下達 TX=data。

3.25 Symbol

3.25.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1

定義在條件語句和系統指令中使用的常數（Constant）或變數（Variable）。請嘗試在程式中使用 Symbol，因為這樣可以提高程式的可讀性和靈活性。

例.

[Symbol]

On = 5

Off = 15

使用者可以用自訂的變數（Variable）名稱來替代使用者 RAM 的名稱（Ri / Xi）。

例. Counter 為 8-bit 變數，CounterH 為 Counter 的高位 nibble，CounterL 為 Counter 的低位 nibble。

[Symbol]

Counter = X0

CounterL = R0

CounterH = R1

3.26 Variable

3.26.1 NX1

變數段落是用於定義變數名稱及資料大小。可用的資料型態如下表：

型態	大小	範圍
VAR8	8-bit (1-byte)	0 ~ 0xFF
VAR16	16-bit (2-byte)	0 ~ 0xFFFF
VAR32	32-bit (4-byte)	0 ~ 0xFFFFFFFF

可用變數名稱為 A-Z 起頭，第二字元開始可使用 A-Z、0-9 及底線，不可與保留字衝突。在 NX1 所使用的變數皆須於此段落宣告，並定義其資料大小。

例. Buf0、Buf1 為 8-bit 變數，Buf2 為 16-bit 變數。

```
[Variable]
VAR8: Buf0, Buf1
VAR16: Buf2
```

3.27 Storage Variable

3.27.1 NX1

此段落中所定義的變數可使用 Storage 指令儲存在 SPI Flash 或是 Embedded Flash 上，在 IC 斷電重開後，系統會自動重新載入，可用於保存系統狀態。

Note: 當使用 COC 功能時，第二版之後的程式所使用的 Variable 不允許進行變更（包括數量、初始值）。

例.

```
[Option]
Storage = Auto                          ; 系統會在進入睡眠之前，自動將變數存至 SPI0。
Storage_Location = SPI0
[Storage Variable]
Var8: var0
[Path]
PowerOn: SPIPlay(ch0, var0)             ; 上電後自動播放下一首。
TR1: SPIPlay(ch0, var0++)
```

3.28 Table

3.28.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1

Table 段落可以定義以二維方式排列用於查詢操作的 table。針對 NY4 / NY5 / NY5+ / NY6 / NY9T 每一個數值可以是 0x000 到 0x3FF 的任意整數，針對 NY7 每一個數值可以是 0x000 到 0xFFF 的任意整數，

針對 NX1 每一個數值可以是 0 ~ 0xFFFFFFFF (32-bit)。對於 4-bit 指令來說，排列方式不能夠超過 16x16 陣列，而對於 8-bit 指令來說，排列方式不能夠超過 256x256 陣列，十進位元和十六進位的數值都可以在 *Q-Code* 中使用。

例.

```
[Table]
Trans:
{
   [0x1, 0x3, 0x153, 0x17, 0x9],
   [0x4, 0x5, 0x3F6, 0x8, 0x10],
   [0x0F, 0x1C, 0x2, 0x3, 0xF4]
}
[Path]
P1: R0=2, R1=1, TableL(Trans,R0,R1,R2)     ; R2 等於 0x6。
P2: R0=2, R1=1, TableM(Trans,R0,R1,R2)     ; R2 等於 0xF。
P3: R0=2, R1=1, TableH(Trans,R0,R1,R2)     ; R2 等於 0x3。
```

Table command: Table { Return_Type } (Table_Name, Col_Index, Row_Index, Return_Reg)

ReturnType: {L, M, H} ; L：取得 b3~b0， M：取得 b7~b4，H：取得 b11~b8。

Table_Name: {Table 名稱}

Col_Index: {常數，RAM} ; 0<= 常數 <=15, RAM = User RAM。

Row_Index: {常數，RAM} ; 0<= 常數 <=15, RAM = User RAM。

Return_Reg: {RAM}

注意：

1. 陣列必須要填滿，該位址若是空的，則 Q-Code 會自動填 0。

2. 當所讀取的位址超出陣列大小時則會讀到 0。

3.29 ASM

3.29.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T

有時當使用者開發某些專案時，由於 *Q-Code* 是一種簡易的高階語言，其對於時序控制的準確性及效率等可能未必能滿足使用者的需要。此時，使用者可考慮在 ASM 段落中直接編寫一些組合語言副程式來滿足這個需求，而不用重新以組合語言編寫整個程式。在 ASM 段落內編寫的組合語言副程式，將可以在 Path，Background1 和 Background2 等段落中調用。

由於這些組合語言副程式有別於一般的 *Q-Code* 語法，所以在編寫這些代碼時必須將組合語言副程式的名稱和“{”“}”以及每句組合語言各自置於一個獨立行中。

語法：

```
[ASM]
```

```
Name
{
    ...
    assembly code
    ...
}
```

例.

```
[ASM]
ABC
{
    mvla    5
    ……
    mvam    pa
}

[Path]
P1: ABC                                    ; 呼叫 Assembly。
```

注意：由於 Q-Code 的 User RAM 採取動態配置的方式，所以，當使用者要在 Q-Code 中插入 ASM，則需要看 Q-Code 的 Build Message 中，該 RAM 被定義至哪一個 PAGE。使用者在 Assembly Code 中需要自行切換 RAM PAGE。

3.30 C-Code

3.30.1 NX1

C 語言段落允許編寫任意 C 語言程式碼，此段落內的文字將會完整的傳遞給 C 語言編譯器。Q-Code 僅解析其中的函數定義，在 **[Path]** 可以直接的呼叫 C 語言段落內定義的函數。

關於大小寫的處理，C 語言是有大小寫區分，而 Q-Code 無大小寫分別。當 C 語言段落定義了兩個同名、大小寫不相符的函數，Q-Code 在處理 **[Path]** 內函數呼叫將永遠配對到第一個出現的 C 語言函數。建議 C 語言段落不要定義大小寫不同的同名函數。C 語言區段的程式可以直接存取變數段落(Variable)，是用於定義變數名稱及資料大小。可用的資料型態如下表：

型態	大小	範圍
VAR8	8-bit (1-byte)	0 ~ 0xFF
VAR16	16-bit (2-byte)	0 ~ 0xFFFF
VAR32	32-bit (4-byte)	0 ~ 0xFFFFFFFF

可用變數名稱為 A-Z 起頭，第二字元開始可使用 A-Z、0-9 及底線，不可與保留字衝突。在 NX1 所使用的變數皆須於此段落宣告，並定義其資料大小。

例. Buf0、Buf1 為 8-bit 變數，Buf2 為 16-bit 變數。

```
[Variable]
VAR8: Buf0, Buf1
VAR16: Buf2
```

定義的變數。存取變數時，必須使用小寫字母。

實際範例請參考 5.4.2。

3.31 Macro

3.31.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1

當使用者開發專案時，為了程式閱讀方便。可考慮在 Macro 段落中直接編寫一些 *Q-Code* 程式來滿足這個需求，而不用每次在程式段中寫相同的程式來完成專案。在 Macro 段落內編寫的 *Q-Code* 程式，將可以在 Path，Background1，Background2 和 Background3 等段落中直接使用。

每一個巨集皆由 path 名稱加上一個冒號":"及一個或數個步驟（steps）所組成。

注意：Macro 段落中的程式將會用巨集的方式展開，每次展開皆會佔用 ROM Size。

在以下範例中，Macro 段落中的程式可在前景或是背景中呼叫，且可呼叫相同的 Macro；也可以利用 Switch 指令來進行呼叫。

例.

```
[Macro]
MACRO1: PlayV(Ch1, $V2), DELAY(1)
MACRO2: PlayV(Ch1, $V3), DELAY(1)
MACRO3: PlayV(Ch1, $V4), DELAY(1)
MACRO4: PB=1

[Path]
PowerOn: KEY1, R0=0
TR1: SWITCH(R0)=[SW1, SW2, SW3, SW4]  ; R0 的內容值來決定要跳躍的 Label。
SW1: R0=1, MACRO1                    ; 呼叫 Macro1。
SW2: R0=2, BG1                       ; 呼叫 BG1。
SW3: R0=3, BG2                       ; 呼叫 BG2。
SW4: R0=0, PlayV(Ch1, $V1)           ; 播放 PlayV。
TR2: VOICE? MACRO4, PB=0             ; Voice 播放中，則執行 Macro4 完後，繼續執行接在後
                                     ; 面的指令。
[Background1]
BG1: MACRO2                          ; 呼叫 Marco2。

[Background2]
BG2: MACRO3                          ; 呼叫 Macro3。
```

3.32 Sentence

當使用者開發專案時，為了程式閱讀方便。可考慮在 Sentence 段落中直接編寫一些組合句子來滿足這個需求，而不用每次在程式段中寫相同的程式來完成專案。在 Sentence 段落內編寫的 *Q-Code* 程式，將可以使用 PlayS 指令在 Path，Background1，Background2 和 Background3 段落中直接使用。

3.32.1 NY4 / NY5

Sentence 組合句子皆由 path 名稱加上一個冒號":"及一個或數個步驟（steps）所組成。

例.

[Sentence]

S1: PlayV(CH2,$V0), DELAY(0.1), PlayV(CH2,$V1), DELAY(0.2), PlayV(CH2,$V2)

S2: PlayV(CH2,$V3), PlayV(CH2,$V4), PlayV(CH2,$V5)

S3: PlayV(CH2,$V6), DELAY(0.1), PlayV(CH2,$V7)

[Path]

PowerOn: KEY1

TR1: PlayS($S1) ; 播放 Sentence "S1"。

[Background1]

BG1: PlayS($S1), PlayS($S3) ; 播放 Sentence "S1"與 Sentence "S3"。

[Background2]

BG3: PlayS($S1), PlayS($S2) ; 播放 Sentence "S1"與 Sentence "S2"。

3.32.2 NY5+ / NY6 / NY7 / NX1

Sentence 組合句子皆由 path 名稱加上一個冒號":"及一個或數個步驟（steps）所組成。

*注意：句子功能主要用來將多個資源檔案（**Melody**、**Speech** 及 **Action** 等）組合成一個句子使用，因此 **Sentence** 段落中只允許 **Play** 及 **Delay** 等指令，其餘指令不可使用在 **Sentence** 段落內。*

例.

[Sentence]

S1: PlayV(CH2,$V0), DELAY(0.1), PlayV(CH2,$V1), DELAY(0.2), PlayV(CH2,$V2)

S2: PlayV(CH2,$V3), PlayV(CH2,$V4), PlayV(CH2,$V5)

S3: PlayV(CH2,$V6), DELAY(0.1), PlayV(CH2,$V7)

[Path]

PowerOn: KEY1

TR1: PlayS($S1) ; 播放 Sentence "S1"。

[Background1]

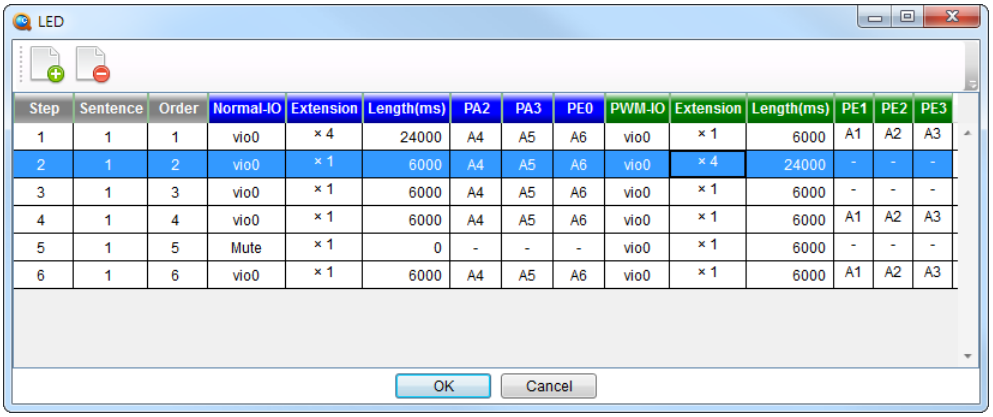
BG1: PlayS($S1), PlayS($S3) ; 播放 Sentence "S1"與 Sentence "S3"。

[Background2]

BG3: PlayS($S1), PlayS($S2) ; 播放 Sentence "S1"與 Sentence "S2"。

3.32.3 NY9T

提供編排 Action 及 PWM 輸出用的 LED Sentence 編輯工具。



滑鼠右鍵功能表如下表所示：

功能表選項	功能描述	熱鍵
New Step	在全部段落的最後新'增一個 LED 動作格	＋
Insert Step	在選取的 LED 動作格上插入一個 LED 動作格	Insert
Delete Step	移除選取的 LED 動作格	－, Delete
Delete Sentence	移除選取的 LED 組合	-
Remove All	移除全部的 LED 組合	-

例.

[Sentence]

L1: PlayPWMS(@000,$VIO0,1), PA=[A5 A4 0 0], PE=[0 0 0 A6], PlayA(Ch1,$VIO0,4), WaitPN

L2: PlayPWMS(@000,$VIO0,4), PA=[A5 A4 0 0], PE=[0 0 0 A6], PlayA(Ch1,$VIO0,1), WaitPN

L3: PA=[A5 A4 0 0], PE=[0 0 0 A6], PlayA(Ch1,$VIO0,1)

L4: PlayPWMS(@111,$VIO0,1), PA=[A5 A4 0 0], PE=[0 0 0 A6], PlayA(Ch1,$VIO0,1), WaitPN, Delay(300ms), PlayPWMS(@111,$VIO0,1), PA=[A5 A4 0 0], PE=[0 0 0 A6], PlayA(Ch1,$VIO0,1), WaitPN

[Path]

PowerOn: KEY1, PlayS($L1), L4 ; **播放 Sentence "L1"及"L4"。**

TR1: BG1 ; **播放 BG1。**

TR2: BG2 ; **播放 BG2。**

[Background1]

BG1: PlayS($L2), PlayS($L3), L4 ; **播放 Sentence "L2"與 Sentence "L3"及"L4"。**

[Background2]

BG2: PlayS($L3), PlayS($L2), L4 ; **播放 Sentence "L3"與 Sentence "L2"及"L4"。**

3.33 Subroutine

3.33.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1

當使用者開發專案時，為了節省程式空間。可考慮在 Subroutine 段落中直接編寫一些 *Q-Code* 副程式來滿足這個需求，而不用每次在程式段中寫相同的程式來完成專案。在 Subroutine 段落內編寫的 *Q-Code* 程式，將可在 Path，Background1，Background2 和 Background3 段落中直接使用。

Subroutine 副程式皆由 path 名稱加上一個冒號":"及一個或數個步驟（steps）所組成。

在以下範例中，BG1 與 BG2 都是呼叫相同的 Subroutine。但是 *Q-Code* 會自動將 Program 展開成 Background1 與 Background2 的副程式。不需要使用者自己寫兩種不同層級的副程式。

例.

```
[Subroutine]
SUB1: PA=0xF, DELAY(0.5), PA=0x0, DELAY(0.5), PlayV(Ch0,$V1)
SUB2: PA=0xF, DELAY(0.5), PA=0x0, DELAY(0.5), PlayV(Ch1,$V1)

[Path]
PowerOn: BG(BG1,BG2), SUB1              ; 播放 Subroutine "SUB1"

[Background1]
BG1:SUB2                                ; 播放 Subroutine "SUB2"

[Background2]
BG2:SUB2                                ; 播放 Subroutine "SUB2"
```

3.34 QIO Custom

3.34.1 NY4 / NY5 / NY5+

讓使用者自行對 QIO / Wave Mark 來進行解碼。當使用此功能時，使用者必須使用組合語言來編譯 QIO 解碼程式，原本 *Q-Code* 中的解碼程式將會被替換。

注意：

1. ***當使用 QIO Custom 時，會套用至整個專案，使用者必須自己控制 QIO 的解碼及輸出流程。***
2. ***要使用該功能必須對 QIO 的格式及 Q-Code 架構有一定程度的瞭解，詳情請聯絡九齊科技。***

3.35 Path

3.35.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1

段落中所包含的路徑（paths）是用來描述 *Q-Code* 程式執行過程中的流程，每一個流程稱之為一個路徑。路徑可分為兩種類型，一種是前景路徑（Path 或 Foreground Path），多數用來播放 Voice 或 Melody；另一種是背景路徑（Background Path），用來配合前景路徑做一些背景動作。

每一個前景路徑（Foreground path）皆由 path 名稱加上一個冒號"："及一個或數個步驟（steps）所組成。一個 path 的執行通常是在其他段落中定義的外部觸發事項，或其它 path 直接呼叫所觸發的。

注意：路徑段落中的每個路徑在執行時，若是沒有出現會 PlayV、PlayM、PlayA、PlayS、Delay 指令，來停止當前的播放。其餘指令皆會執行完後，皆會返回原本執行 Play 或 Delay 指令的位置，來執行下一個指令。在 Q-Code 系統中，不需要額外的特殊路徑來做返回動作。

例.

[Input State]

KEY1: TR1R TR2R

[Path]

PowerOn: KEY1

TR1R: PlayV(Ch1, $V0), Delay(1) ; TR1 播放$V0 後，執行延遲 1 秒。

TR2R: R0=5 ; 執行 R0=5 完畢後，若是 PlayV 或 Delay 仍在執行，則會返回。

3.36 Background1

3.36.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1

背景路徑，Background1 是用來定義第 1 個背景路徑，每個背景路徑（Background Path）皆由 path 名稱加上一個冒號“:”及一個或數個步驟（steps）所組成。一個 path 的執行通常是由前景路徑（Foreground Path）中的外部觸發事項或由其它背景路徑（Background Path）直接呼叫所引起的。

例.

[Path]

P0: R0**=**0**,** PlayV(Ch0**,** Indian**,** 8K) & [BG1**,** X]**,** PlayV(Ch0**,** ABC**,** 7K) & [BG3**,** X]

[Background1]

BG1: output1**,** delay(200ms)**,** output2**,** delay(200ms)**,** BG1

BG3: output3**,** delay(200ms)**,** output4**,** delay(200ms)**,** BG3

3.37 Background2

3.37.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1

此段落與 Background1 用法相容，Background2 是用來定義第 2 個背景路徑。

例.

[Path]

P0: R0**=**0**,** PlayV(Indian**,** 8K) & [X**,** BG2]

P1: BG(X**,** BG2)

[Background2]

BG2: PlayV(Ch0**,** $V0**,** 12k)

3.38 Background3

3.38.1 NX1

此段落與 Background1 用法相容，Background3 是用來定義第 3 個背景路徑。

例.

[Path]

P0: R0=0, PlayV(Indian, 8K) & [X, X, BG3]

P1: BG(X, X, BG3)

[Background3]

BG3: PlayV(Ch0, $V0, 12k)

3.39 Background4

3.39.1 NX1

此段落與 Background1 用法相容，Background4 是用來定義第 4 個背景路徑。

例.

[Path]

P0: R0=0, PlayV(Indian, 8K) & [X, X, X, BG4]

P1: BG(X, X, X, BG4)

[Background4]

BG4: PlayV(Ch0, $V0, 12k)

3.40 Background5

3.40.1 NX1

此段落與 Background1 用法相容，Background5 是用來定義第 5 個背景路徑。

例.

[Path]

P0: R0=0, PlayV(Indian, 8K) & [X, X, X, X, BG5]

P1: BG(X, X, X, X, BG5)

[Background5]

BG5: PlayV(Ch0, $V0, 12k)

3.41 Interrupt

3.41.1 NX1

中斷段落同路徑段落，中斷路徑以及會使用到的路徑需在這個段落中。

在 NX1 中，中斷發生時的程式必須位於 ROM 中，當使用 XIP 或是 COC 時，Q-Code 會將路徑放至 SPI Flash，而當中斷路徑呼叫時有可能出現問題。

例. 中斷呼叫有可能發生錯誤

```
[Option]
XIP = Speed_Optimize                ; 開啟 XIP 功能。
SPI0_Code_Access_Mode = Dual

[Path]
PowerOn: TMR_On(TMR0, 256us)        ; 開啟 TMR0 中斷。
Int_TMR0: Var0=1?Path1
Path1: …                            ; Q-Code 會將 Path1 放至 SPI Flash 中。
                                    ; 當 TMR0 中斷發生時執行 Path1，程式可能發生錯誤。
```

因此將中斷路徑放以及相關路徑放至中斷段落中，*Q-Code* 會確保將其置於 ROM 中，避免程式的不正常行為。正確的寫法如下：

例.

```
[Option]
XIP = Speed_Optimize                ; 開啟 XIP 功能。
SPI0_Code_Access_Mode = Dual

[Path]
PowerOn: TMR_On(TMR0, 256us)        ; 開啟 TMR0 中斷。

[Interrupt]
Int_TMR0: Var0=1?Path1
Path1: …
```

以下是適用的中斷路徑：

- INT_TMR0
- INT_TMR1
- INT_TMR2
- INT_TMR3
- INT_PWMA
- INT_PWMB
- RTC_2Hz
- RTC_64Hz
- RTC_1024Hz
- RTC_4096Hz
- RTC_16384Hz

4 Q-Code 特殊路徑（Q-Code Special Paths）

4.1 通用路徑

4.1.1 開機（PowerOn）

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

“**PowerOn**”路徑名稱必須出現在 **[Path]** 段落中，並且是 *Q-Code* 程式開始執行的起始語句。當 IC 的電源開啟後，“**PowerOn**”路徑是 **[Path]** 段落中首先被執行的語句。

注意：PowerOn 為系統保留路徑，使用者不得設置相同的路徑名稱。

例.

```
[Path]
PowerOn: R0=0, R1=0, R2=0, R3=0xF
P0: R0=0, PlayV(Ch1, $Indian,8K) & [BG1, x]
P1: PlayV(Ch1, $Star,6K), PA=1, Delay(0.5), PA=0
P2: R0++, R0=2 ? P1, P0
P3: R1=R2, R3=R1|R2, PlayV(Ch1, $ABC,7K)
```

4.1.2 開機前（Before_PowerOn）

[NX1]

“**Before_PowerOn**”路徑名稱僅能出現在 **[Path]** 段落中，“**Before_PowerOn**”路徑會在“**PowerOn**”之前執行。當 XIP(Execute in place)功能開啟，“**PowerOn**”路徑的程式將會存放於 SPI Flash，而“**Before_PowerOn**”路徑則存放於 IC 內部 ROM。如果使用者所選用的 SPI Flash 晶片需要以特殊指令初始化，可以在“**Before_PowerOn**”撰寫初始化 SPI Flash 的指令。

一旦 “**Before_PowerOn**” 執行結束後，系統會自動執行 “**PowerOn**” 。使用者不須要在 “**Before_PowerOn**” 的結尾自行跳躍到 “**PowerOn**” 。

例. 在系統啟動前對 SPI Flash 進行初始化。

```
[Path]
PowerOn:
Before_PowerOn: SPI_WRSR(0x31,0x2)
```

4.1.3 主迴圈（Main）

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

“**Main**”路徑名稱僅能出現在 **[Path]** 段落中，每次主迴圈執行完，系統會處理該路徑一次。

注意：

1. ***請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低。***
2. ***由於系統還需要執行其它的功能，且依照所使用的功能不同，所以該時間不會每次都相同。***

3. 不允許在此路徑直接執行任何播放或 Delay 指令，但可執行條件判斷指令。

例.

```
[Path]
PowerOn:
Main: !PA                    ; 每次主迴圈執行完，PA 就反相輸出。
```

4.1.4 休眠（Sleep）

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

“**Sleep**”路徑名稱僅能出現在 **[Path]** 段落中，每次 IC 進入休眠前，皆會處理該路徑一次。

例.

```
[Path]
PowerOn: PB=0xF, DELAY(1)    ; Power On 後，PB 輸出 1 秒後，進入休眠。
SLEEP: PB=0                  ; 進入休眠前將 PB 輸出設為低電位。
```

4.1.5 喚醒（WakeUp）

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

“**WakeUp**”路徑名稱僅能出現在 **[Path]** 段落中，每次 IC 被喚醒後，皆會處理該路徑一次。

注意：若有使用 I/O 擴展晶片，需延遲 1ms 才能對 I/O 擴展晶片進行操作。

例.

```
[Path]
PowerOn: KEY1                ; Power On 後，立即進入休眠。
WakeUp: PB=0xF               ; 每次 IC 被喚醒後，將 PB 設為高電位。
```

4.2 計時器路徑

4.2.1 512 微秒（512us）

[NY7]

“**512us**”路徑名稱僅能出現在 **[Path]** 段落中，每次 512us 時間，系統會處理該路徑一次。

注意：

1. 請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低。

2. 由於系統還需要執行其它的功能，所以該 512us 並不會非常準確。

3. 不可使用的指令請參考特殊路徑中不可使用的功能。

例.

```
[Path]
PowerOn:
512us: !PA                   ; 每 512us，PA 就反相輸出。
```

4.2.2 1 毫秒（1ms）

[NY5+ / NY6 / NY7 / NY9T / NX1]

“**1ms**”路徑名稱僅能出現在 **[Path]** 段落中，每次 1ms 時間，系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低。*
2. *由於系統還需要執行其它的功能，所以該 **1ms** 並不會非常準確。*
3. *不可使用的指令請參考特殊路徑中不可使用的功能。*
4. ***NY9T** 使用 **1ms** 路徑後，一定要執行 **1ms_RET** 指令，否則系統會出現不能預期的錯誤。*
5. ***NY9T** 若 **Timebase=4ms**，則 **1ms** 路徑不會被執行。*

例.

```
[Path]
PowerOn:
1ms: !PA                    ; 每 1ms，PA 就反相輸出。
```

4.2.3 2 毫秒（2ms）

[NY5+ / NY6 / NY7 / NX1]

“**2ms**”路徑名稱僅能出現在 **[Path]** 段落中，每次 2ms 時間，系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低。*
2. *由於系統還需要執行其它的功能，所以該 **2ms** 並不會非常準確。*
3. *不可使用的指令請參考特殊路徑中不可使用的功能。*

例.

```
[Path]
PowerOn:
2ms: !PA                    ; 每 2ms，PA 就反相輸出。
```

4.2.4 4 毫秒（4ms）

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

“**4ms**”路徑名稱僅能出現在 **[Path]** 段落中，每次 4ms 時間，系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低。*
2. *由於系統還需要執行其它的功能，所以該 **4ms** 並不會非常準確。*
3. *不可使用的指令請參考特殊路徑中不可使用的功能。*
4. *在 **NY9T** 中使用 **4ms** 路徑，一定要執行 **4ms_RET** 指令，否則系統將會出現不能預期的錯誤。*

例.

[Path]

PowerOn:

4ms: !PA **；每 4ms，PA 就反相輸出。**

4.2.5 8 毫秒（8ms）

[NY5+ / NY6 / NY7 / NX1]

“**8ms**”路徑名稱僅能出現在 **[Path]** 段落中，每次 8ms 時間，系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低。*
2. *由於系統還需要執行其它的功能，所以該 **8ms** 並不會非常準確。*
3. *不可使用的指令請參考特殊路徑中不可使用的功能。*

例.

[Path]

PowerOn:

8ms: !PA **；每 8ms，PA 就反相輸出。**

4.2.6 500 毫秒（500ms）

[NY9T / NX1]

“**500ms**”路徑名稱僅能出現在 **[Path]** 段落中，每次 500 毫秒時間，系統會處理該路徑一次。

例.

[Path]

PowerOn:

500ms: R0++, R0=2?Timer_1sec **；每 500 毫秒 R0 就計數一次，計數至 2 次表示 1 秒鐘。**

Timer_1sec: R0=0, User Program.....

注意：

1. *用於長時間的計時，可利用內部 **16KHz** 的 **RC** 計時器獲得較佳的省電效能。由於 **16KHz** 的 **RC** 計時器存在某種程度的誤差，若是需要較為精確的計時，請勿使用 **16KHz** 的 **RC** 計時器功能來計時。*
2. *不可使用的指令請參考特殊路徑中不可使用的功能。*

4.2.7 1 秒（1sec）

[NY9T]

“**1sec**”路徑名稱僅能出現在 **[Path]** 段落中，每次 1 秒時間，系統會處理該路徑一次。

例.

[Path]

```
PowerOn:
1sec: X0++, X0=60?Timer_1min          ; 每 1 秒 X0 就計數一次，計數至 60 次表示 1 分鐘。
Timer_1min: X0=0, User Program.....
```

注意：

1. *用於長時間的計時，可利用內部 16KHz 的 RC 計時器獲得較佳的省電效能。由於 16KHz 的 RC 計時器存在某種程度的誤差，若需要較為精確的計時，請勿使用 16KHz 的 RC 計時器功能來計時。*
2. *不可使用的指令請參考特殊路徑中不可使用的功能。*

4.2.8 4 秒（4sec）

[NY9T]

"**4sec**"路徑名稱僅能出現在 **[Path]** 段落中，每次 4 秒時間，系統會處理該路徑一次。

例.

```
[Path]
PowerOn:
4sec: R0++, R0=15?Timer_1min          ; 每 4 秒 R0 就計數一次，計數至 15 次表示 1 分鐘。
Timer_1min: R0=0, User Program.....
```

注意：

1. *用於長時間的計時，可利用內部 16KHz 的 RC 計時器獲得較佳的省電效能。由於 16KHz 的 RC 計時器存在某種程度的誤差，若是需要較為精確的計時，請勿使用 16KHz 的 RC 計時器功能來計時。*
2. *不可使用的指令請參考特殊路徑中不可使用的功能。*

4.3 中斷路徑

4.3.1 中斷路徑（Interrupt）

[NY5 / NY5+ / NY7]

"**Interrupt**"路徑名稱僅能出現在 **[Path]** 段落中，每次 IC 進入中斷，皆會處理該路徑一次。

注意：

1. *由於 IC 僅提供 1 階的中斷，若選擇使用中斷來執行 PlayA / PlayAS、PlayM / PlayMS 及 QIO 時，一樣會執行 Interrupt 路徑中的指令，使用者在使用時需要特別注意!!*
2. *NY5 中，若是執行 PlayA / PlayAS、PlayM / PlayMS 及 QIO 功能後，原本設定的中斷時間將會被系統修改。*
3. *不可使用的指令請參考特殊路徑中不可使用的功能。*

例.

```
[Path]
PowerOn: KEY1, INT_ON, INT=1.024
Interrupt: !PA        ; 每 1ms 發生時間中斷一次，且將 PA 反相輸出。
```

4.3.2 256 微秒中斷路徑（Int_256us）

[NY6]

“**Int_256us**”路徑名稱僅能出現在 **[Path]** 段落中，當 256us 中斷發生時，系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。*
2. *不可使用的指令請參考特殊路徑中不可使用的功能。*

例.

```
[Path]
PowerOn:
Int_256us: !PA                    ; 當 256us 中斷發生時，PA 就反相輸出。
```

4.3.3 512 微秒中斷路徑（Int_512us）

[NY6]

“**Int_512us**”路徑名稱僅能出現在 **[Path]** 段落中，當 512us 中斷發生時，系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低，並影響其他中斷的發生。*
2. *不可使用的指令請參考特殊路徑中不可使用的功能。*

例.

```
[Path]
PowerOn:
Int_512us: !PA                    ; 當 512us 中斷發生時，PA 就反相輸出。
```

4.3.4 1 毫秒中斷路徑（Int_1ms）

[NY6]

“**Int_1ms**”路徑名稱僅能出現在 **[Path]** 段落中，當 1ms 中斷發生時，系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。*
2. *不可使用的指令請參考特殊路徑中不可使用的功能。*

例.

```
[Path]
PowerOn:
Int_1ms: !PA                      ; 當 1ms 中斷發生時，PA 就反相輸出。
```

4.3.5 16 毫秒中斷路徑（Int_16ms）

[NY6]

“**Int_16ms**”路徑名稱僅能出現在 **[Path]** 段落中，當 16ms 中斷發生時，系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。*
2. *不可使用的指令請參考特殊路徑中不可使用的功能。*

例.

```
[Path]
PowerOn:
Int_16ms: !PA                    ; 當 16ms 中斷發生時，PA 就反相輸出。
```

4.3.6 比較器中斷路徑（Int_CMP）

[NY6B / NY6C]

“**Int_CMP**”路徑名稱僅能出現在 **[Path]** 段落中，當 Comparator 中斷發生時，系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。*
2. *不允許在此路徑直接執行任何播放或 Delay 指令，但可執行條件判斷指令。*

例.

```
[Path]
PowerOn:
Int_CMP: !PA                     ; 當 CMP 中斷發生時，PA 就反相輸出。
```

4.3.7 計時器中斷路徑（Int_TMR）

[NY6]

“**Int_TMR**”路徑名稱僅能出現在 **[Path]** 段落中，當 Timer 中斷發生時（Timer 時間可由程式設定），系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。*
2. *不允許在此路徑直接執行任何播放或 Delay 指令，但可執行條件判斷指令。*

例.

```
[Path]
PowerOn:
Int_TMR: !PA                     ; 當 TMR 中斷發生時，PA 就反相輸出。
```

4.3.8 計時器 0 中斷路徑（Int_TMR0）

[NY5+ / NX1]

NY5+系列，“**Int_TMR0**”路徑名稱僅能出現在 **[Path]**，而在 NX1 系列僅能出現在 **[Interrupt]** 段落中。

當 Timer0 中斷發生時（Timer 時間可由程式設定），系統會處理該路徑一次。

注意：

1. ***請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。***
2. ***不允許在此路徑直接執行任何播放或 Delay 指令，但可執行條件判斷指令。***
3. ***NY5+指定的時間範圍為 128us ~ 256us。***
4. ***NX1 OTP 計時時間隨系統頻率而有不同：***
   - ***High Clock Frequency 為 12MHz 時，為 64 ~ 5461us。***
   - ***High Clock Frequency 為 16MHz 時，為 64 ~ 4096us。***
   - ***High Clock Frequency 為 24MHz 時，為 64 ~ 2730us。***
   - ***High Clock Frequency 為 32MHz 時，為 64 ~ 2048us。***
5. ***NX1 EF 支援 64 ~ 1365us。***

例.

```
[Path]
PowerOn:
TR0: TMR_ON(TMR0, 128us)
Int_TMR0: !PA.0                    ; 每 128us 觸發一次，當 Int_TMR0 觸發，PA.0 就反向輸出。
```

4.3.9 計時器 1 中斷路徑（Int_TMR1）

[NY5+ / NX1]

NY5+系列，“**Int_TMR1**”路徑名稱僅能出現在 **[Path]**，而在 NX1 系列僅能出現在 **[Interrupt]** 段落中。

當 Timer1 中斷發生時（Timer 時間可由程式設定），系統會處理該路徑一次。

注意：

1. ***請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。***
2. ***不允許在此路徑直接執行任何播放或 Delay 指令，但可執行條件判斷指令。***
3. ***NY5+指定的時間範圍為 128us ~ 256us。***
4. ***NX1 OTP 計時時間隨系統頻率而有不同：***
   - ***High Clock Frequency 為 12MHz 時，為 64 ~ 5461us。***
   - ***High Clock Frequency 為 16MHz 時，為 64 ~ 4096us。***
   - ***High Clock Frequency 為 24MHz 時，為 64 ~ 2730us。***
   - ***High Clock Frequency 為 32MHz 時，為 64 ~ 2048us。***
5. ***NX1 EF 支援 64 ~ 1365us。***

例.

[Path]

PowerOn:

TR0: TMR_ON(TMR1, 128us)

Int_TMR1: !PA.1 ; 每 128us 觸發一次，當 Int_TMR1 觸發，PA.1 就反向輸出。

4.3.10 計時器 2 中斷路徑（Int_TMR2）

[NY5+ / NX1]

NY5+系列，“**Int_TMR2**”路徑名稱僅能出現在 **[Path]**，而在 NX1 系列僅能出現在 **[Interrupt]** 段落中。

當 Timer2 中斷發生時（Timer 時間可由程式設定），系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。*
2. *不允許在此路徑直接執行任何播放或 Delay 指令，但可執行條件判斷指令。*
3. *NY5+指定的時間範圍為 128us ~ 256us。*
4. *NX1 OTP 計時時間隨系統頻率而有不同：*
   - *High Clock Frequency 為 12MHz 時，為 64 ~ 5461us。*
   - *High Clock Frequency 為 16MHz 時，為 64 ~ 4096us。*
   - *High Clock Frequency 為 24MHz 時，為 64 ~ 2730us。*
   - *High Clock Frequency 為 32MHz 時，為 64 ~ 2048us。*
5. *NX1 EF 支援 64 ~ 1365us。*

例.

[Path]

PowerOn:

TR0: TMR_ON(TMR2, 128us)

Int_TMR2: !PA.2 ; 每 128us 觸發一次，當 Int_TMR2 觸發，PA.2 就反向輸出。

4.3.11 計時器 3 中斷路徑（Int_TMR3）

[NY5+ / NX1]

NY5+系列，“**Int_TMR3**”路徑名稱僅能出現在 **[Path]**，而在 NX1 系列僅能出現在 **[Interrupt]** 段落中。

當 Timer3 中斷發生時（Timer 時間可由程式設定），系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。*
2. *不允許在此路徑直接執行任何播放或 Delay 指令，但可執行條件判斷指令。*
3. *NY5+指定的時間範圍為 128us ~ 256us。*
4. *NX1 OTP 計時時間隨系統頻率而有不同：*
   - *High Clock Frequency 為 12MHz 時，為 64 ~ 5461us。*

- *High Clock Frequency 為 16MHz 時，為 64 ~ 4096us。*
- *High Clock Frequency 為 24MHz 時，為 64 ~ 2730us。*
- *High Clock Frequency 為 32MHz 時，為 64 ~ 2048us。*

5. NX1 EF 支援 64 ~ 1365us。

例.

[Path]

PowerOn:

TR0: TMR_ON(TMR3, 128us)

Int_TMR3: !PA.3 ；每 128us 觸發一次，當 Int_TMR3 觸發，PA.3 就反向輸出。

4.3.12 PWMA 中斷路徑（Int_PWMA）

[NX1]

“**Int_PWMA**”路徑名稱僅能出現在 **[Interrupt]** 段落中，當 PWMA 計時器中斷發生時（Timer 時間可由程式設定），系統會處理該路徑一次。

注意：

1. 請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。

2. 不允許在此路徑直接執行任何播放或 Delay 指令，但可執行條件判斷指令。

3. NX1 OTP 計時時間隨系統頻率而有不同：

- *High Clock Frequency 為 12MHz 時，為 64 ~ 5461us。*
- *High Clock Frequency 為 16MHz 時，為 64 ~ 4096us。*
- *High Clock Frequency 為 24MHz 時，為 64 ~ 2730us。*
- *High Clock Frequency 為 32MHz 時，為 64 ~ 2048us。*

4. NX1 EF 支援 64 ~ 1365us。

例.

[Path]

TR0: TMR_ON(PWMA, 128us)

[Interrupt]

Int_PWMA: !PA.3 ；當 PWMA 計時器中斷每 128us 觸發，PA.3 就反向輸出。

4.3.13 PWMB 中斷路徑（Int_PWMB）

[NX1]

“**Int_PWMB**”路徑名稱僅能出現在 **[Interrupt]** 段落中，當 PWMB 計時器中斷發生時（Timer 時間可由程式設定），系統會處理該路徑一次。

注意：

1. 請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。

2. *不允許在此路徑直接執行任何播放或 Delay 指令，但可執行條件判斷指令。*
3. *NX1 OTP 計時時間隨系統頻率而有不同：*
   - *High Clock Frequency 為 12MHz 時，為 64 ~ 5461us。*
   - *High Clock Frequency 為 16MHz 時，為 64 ~ 4096us。*
   - *High Clock Frequency 為 24MHz 時，為 64 ~ 2730us。*
   - *High Clock Frequency 為 32MHz 時，為 64 ~ 2048us。*
4. *NX1 EF 支援 64 ~ 1365us。*

例.

```
[Path]
TR0: TMR_ON(PWMB, 128us)
[Interrupt]
Int_PWMB: !PA.3                ; 當 PWMB 計時器中斷每 128us 觸發，PA.3 就反向輸出。
```

4.3.14 實時時鐘中斷路徑（RTC_2Hz / RTC_64Hz / RTC_1024Hz / RTC_4096Hz / RTC_16384Hz）

[NX1]

“**RTC_2Hz / RTC_64Hz / RTC_1024Hz / RTC_4096Hz / RTC_16384Hz**”路徑名稱僅能出現在 **[Interrupt]** 段落中，當 RTC 中斷發生時，系統會處理該路徑一次。

注意：

1. *請勿在此路徑底下執行過多的指令，執行過多容易讓系統執行效率降低並影響其他中斷的發生。*
2. *不允許在此路徑直接執行任何播放或 Delay 指令，但可執行條件判斷指令。*
3. *RTC 中斷由系統初始化時自動啟用，無需開關。*
4. *RTC_4096Hz 僅 NX12/3P44A 與 NX11P22A 支援。RTC_16384Hz 除了 NX12/3P44A 與 NX11P22A 以外，其他 NX1 可支援。*

例.

```
[Interrupt]
RTC_2Hz: !PA.6                 ; 每秒觸發兩次，當 RTC_2Hz 觸發，PA.6 就反向輸出。
```

4.4 電壓偵測路徑

4.4.1 特定電壓偵測（LVD_mVn / LVD_Max）

[NY4PxxxC / NY5+ / NY6P025A / NY6B / NY6C / NX1]

“**LVD_mVn**” / “**LVD_Max**”路徑名稱僅能出現在 **[Path]** 段落中，當 VDD 變化至指定範圍時，系統會處理該路徑一次。

不同系列 IC 可支援的電壓偵測系統路徑及對應的電壓範圍如下：

電壓範圍	NY4PxxxC / NY5+	NY6P025A LVD1	NY6P025A LVD2	NY6B / NY6C	NX1 OTP	NX1 EF
<2.0V	LVD_2V0	LVD_2V4	LVD_2V0	LVD_2V4	LVD_2V2	LVD_2V0
2.0V ~ 2.2V	LVD_2V2		LVD_2V2			LVD_2V2
2.2V ~ 2.4V	LVD_2V4		LVD_3V0		LVD_2V4	LVD_2V4
2.4V ~ 2.6V	LVD_2V8	LVD_2V8		LVD_2V7	LVD_2V6	LVD_2V8
2.6V ~ 2.7V					LVD_3V2	
2.7V ~ 2.8V				LVD_3V6		
2.8V ~ 3.0V	LVD_3V0	LVD_3V6				LVD_3V0
3.0V ~ 3.2V	LVD_3V3		LVD_3V2			LVD_3V2
3.2V ~ 3.3V			LVD_Max		LVD_3V4	LVD_3V4
3.3V ~ 3.4V	LVD_3V6					
3.4V ~ 3.6V					LVD_3V6	LVD_3V6
3.6V ~ 4.1V	LVD_Max	LVD_4V1		LVD_4V1	LVD_Max	LVD_Max
>4.1V		LVD_Max		LVD_Max		

注意：

1. *若上電後即觸發此路徑，則 PowerOn 路徑可能會執行不完全。*
2. *以 NY5+為例，當指定 LVD_2V8 及 LVD_2V4 時，LVD_2V4 也涵蓋<2.0V ~ 2.4V 的變化。*

4.4.2 電壓偵測（LVD）

[NY5+ / NY6B / NY6C / NX1]

"**LVD**"路徑名稱僅能出現在 **[Path]** 段落中，當 VDD 電壓在不同 LVD 階數變化時，系統會處理該路徑一次。

例.

```
[Path]
PowerOn:
LVD: PA=1                    ; 當電壓在不同 LVD 階數間變化時，PA 輸出 1。
```

4.5 資料傳輸路徑

4.5.1 紅外線接收（IR_RX）

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

"**IR_RX**"路徑名稱僅能出現在 **[Path]** 段落中，每次 IR 接收到一筆資料，皆會處理該路徑一次，使用者可在此路徑用指令讀取 IR 內的資料。

例.

```
[Path]
```

```
IR_RX: [R1, R0] = IR_RX                    ; 在接收到 IR 資料時，將資料存到 R0 及 R1。
```

4.5.2 串列資料接收（SC_RX）

[NY4 / NY5 / NY5+ / NY6 / NY7]

“**SC_RX**”路徑名稱僅能出現在 **[Path]** 段落中，每次 SC 接收到一筆資料，皆會處理該路徑一次，使用者在此路徑用指令讀取 SC 內的資料。

例.

```
[Path]
SC_RX: [R1, R0] = SC_RX                    ; 在接收到 SC 資料時，將資料存到 R0 及 R1。
```

4.5.3 I2C 資料接收（I2C_RX）

[NY5+ / NX1]

“**I2C_RX**”路徑名稱僅能出現在 **[Path]** 段落中，每次 I2C 接收到一筆資料，皆會處理該路徑一次，使用者在此路徑用指令讀取 I2C 接收到的資料。

例.

```
[Path]
I2C_RX: [R1, R0] = I2C_RX                  ; 在接收到 I2C 資料時，將資料存到 R0 及 R1。
```

4.5.4 I2C 資料傳輸完成（I2C_TX_Ok）

[NY5+ / NX1]

“**I2C_TX_Ok**”路徑名稱僅能出現在 **[Path]** 段落中，每次 I2C 傳送完成一筆資料，皆會處理該路徑一次，使用者在此路徑可以判斷是否有 ACK。

例.

```
[Path]
I2C_TX_OK: I2C_Ack?Path1                   ; I2C 資料傳送完成，判斷是否有回應 ACK 訊號。
Path1:     I2C_TX(0x5A)                    ; 有 ACK 訊號則傳送 0x5A 的資料。
```

4.5.5 I2C 資料傳輸錯誤（I2C_Error）

[NX1]

“**I2C_Error**”路徑名稱僅能出現在 **[Path]** 段落中，當 I2C 因 Clock Stretching 發生 Timeout，或 I2C Bus arbitration 失敗等狀況發生時，皆會處理該路徑一次。

例.

```
[Path]
I2C_Error: I2C_Reset                       ; I2C 資料傳送失敗，將 I2C 傳輸重設。
```

4.5.6 UART 資料接收（UART_RX）

[NY5+ / NX1]

“**UART_RX**”路徑名稱僅能出現在 **[Path]** 段落中，每次 UART 接收到一筆資料，皆會處理該路徑一次，使用者在此路徑用指令讀取 UART 接收到的資料。

例.

```
[Path]
UART_RX: [R1, R0] = UART_RX          ; 在接收到 UART 資料時，將資料存到 R0 及 R1。
```

4.5.7 WaveID 資料接收（WaveID_RX）

[NX1]

“**WaveID_RX**”路徑名稱僅能出現在 **[Path]** 段落中，每次 WaveID 接收到一筆資料，皆會處理該路徑一次，使用者在此路徑用指令讀取 WaveID 的資料。

例.

```
[Path]
WaveID_RX: R0= WaveID_RX             ; 在接收到 WaveID 資料時，將資料存到 R0。
```

4.5.8 Sound Localization 資料接收（SL_RX）

[NX1]

“**SL_RX**”路徑名稱僅能出現在 **[Path]** 段落中，每次 Sound Localization 接收到一筆資料，皆會處理該路徑一次，使用者在此路徑用指令讀取聲音來源的角度。

例.

```
[Path]
SL_RX: R0= SL_RX                     ; 在接收到 Sound Localization 資料時，將偵測到的角度存到 R0。
```

4.6 觸控路徑

4.6.1 強制校正（Enforce_Calibrate）

[NY9T]

“**Enforce_Calibrate**”路徑名稱僅能出現在 **[Path]** 段落中，每次 Enforce_Calibrate 執行完畢後，皆會處理該路徑一次。

注意：

1. *當 SW_Reset 功能關閉後，強制校正路徑方可執行。*
2. *若使用者需要針對不同的應用，而於執行完 Enforfce_Calibrate 後所需要的重置行為不同時，可將 SW_Reset 功能關閉，改由使用者手動進行系統重置。*

例. 無需記憶設定或模式時，可用 SW_Reset 來進行重置。

[Path]

```
PowerOn: KEY1                          ; Power On 後，立即進入休眠。
TR1R: PE.0=1                           ; 觸摸鍵按下時，PE.0 輸出 1。
TR1F: PE.0=0                           ; 觸摸鍵放開時，PE.0 輸出 0。
Enforce_Calibrate: SW_Reset            ; 每次 Enforce_Calibrate 執行後，隨即進行軟體
                                       重置。
```

例. 需記憶設定或模式，且執行完 Enforce_Calibrate 後，持續維持原本狀態。

[Path]

```
PowerOn: KEY1                          ; Power On 後，立即進入休眠。
TR1R: PE.0=1                           ; 觸摸鍵按下時，PE.0 輸出 1。
TR1F: PE.0=0                           ; 觸摸鍵放開時，PE.0 輸出 0。
TR2R: R0++                             ; 切換模式。
Enforce_Calibrate: TouchKey_CLR        ; 每次 Enforce_Calibrate 執行後，隨即進行觸摸
                                       鍵狀態清除。
```

例. 需記憶設定或模式，且執行完 Enforce_Calibrate 後，只清除 IO 狀態。若執行完 Enforce_Calibrate 後不清除觸摸鍵，會自動依序執行觸摸鍵按鍵放開路徑。

[Path]

```
PowerOn: KEY1                          ; Power On 後，立即進入休眠。
TR1R: PE.0=1                           ; 觸摸鍵按下時，PE.0 輸出 1。
TR1F: PE.0=0                           ; 觸摸鍵放開時，PE.0 輸出 0。
TR2R: R0++                             ; 切換模式。
Enforce_Calibrate: PE=0                ; 每次 Enforce_Calibrate 執行後，隨即進行 IO 狀
                                       態清除。
```

例. 需記憶設定或模式，且執行完 Enforce_Calibrate 後，清除 IO 及按鍵狀態，但不執行按鍵放開路徑。

[Path]

```
PowerOn: KEY1                          ; Power On 後，立即進入休眠。
TR1R: PE.0=1                           ; 觸摸鍵按下時，PE.0 輸出 1。
TR1F: PE.0=0                           ; 觸摸鍵放開時，PE.0 輸出 0。
TR2R: R0++                             ; 切換模式。
   Enforce_Calibrate: PE=0, TouchKey_CLR ; 每次 Enforce_Calibrate 執行後，隨即進行 IO 與
                                       觸摸鍵狀態清除。
```

4.7 SPI Flash 路徑

4.7.1 抹除完成（SPI_EraseEnd）

[NY6B / NY6C / NY7 / NX1]

"SPI_EraseEnd"路徑名稱僅能出現在 **[Path]** 段落中。當 SPI Flash 抹除指令完成後，皆會處理該路徑一次，以便使用者知道抹除完成的時間點，進行後續動作。抹除指令包含 SPI_CE、SPI_SE 及 SPI_BE。

例.

```
[Path]
Erase: SPI_CE, R0=1
SPI_EraseEnd: R0=0          ; 執行全記憶體抹除動作，並將 R0 設為 1，抹除完成將 R0 清為 0。
```

4.7.2 抹除逾時（SPI_EraseTimeout）

[NX1]

"SPI_EraseTimeout"路徑名稱僅能出現在 **[Path]** 段落中。當 SPI Flash 抹除等待超過 Option->SPI Flash 所設定的抹除時間，會立刻處理該路徑一次，以便使用者知道抹除發生逾時。此路徑用於 SPI Flash 硬體異常的提示，不允許有跳躍行為。

例.

```
[Path]
TR1: EraseR($Rec0)          ; 擦除位於 SPI Flash 的 Rec0 錄音空間。
SPI_EraseTimeout: PA.0=1    ; 發生抹除逾時，將 PA.0 設為 1。
```

4.8 語音辨識路徑

4.8.1 語音指令群組逾時路徑（VRGC_Timeout）

[NX1]

"VRGC_Timeout"路徑名稱僅能出現在 **[Path]** 段落中，若設定語音指令群組為 VRGC1 = VRG1+VRG2，當 VRG1 之中的指令成功辨認，但並未在時限內辨認到 VRG2 的指令，將會自動執行 VRGC_Timeout 路徑。

例.

```
[Path]
PowerOn:
VRGC_Timeout: PlayV(ch0, $V0)    ; 在語音指令不明時，提示使用者再說一次。
```

4.8.2 語音辨識指令未成功（VR_Unknown）

[NX1]

"**VR_Unknown**"路徑名稱僅能出現在 **[Path]** 段落中，當語音辨識有偵測到聲音，但是卻沒有辨識成功時，會執行該路徑一次。

注意：使用此功能，須先開啟 VAD。

例.

```
[Path]
PowerOn:
VR_Unknown: PlayV(ch0, $V0)                    ; 在語音指令不明時，提示使用者再說一次。
```

4.8.3 Voice Tag 錄音前（VT_BeforeRecord）

[NX1]

"**VT_BeforeRecord**"路徑名稱僅能出現在 **[Path]** 段落中，Voice Tag 訓練前會先錄音數次，錄音前會跳進此路徑。

注意：

1. ***此路徑不可再跳其它路徑。***
2. ***此路徑不可執行須等待的指令，除了 PlayV / PlayA / SpiPlay / SDelay。***
3. ***跳進此路徑後，除了該路徑的工作繼續執行外，其他所有工作將暫停。***

例.

```
[Path]
VT_BeforeRecord: SpiPlay(ch0, 0)               ; Voice Tag 錄音前，跳此 path。
```

4.8.4 Voice Tag 訓練前（VT_BeforeTraining）

[NX1]

"**VT_BeforeTraining**"路徑名稱僅能出現在 **[Path]** 段落中，當 Voice Tag 錄音後訓練前，會跳進此路徑。

注意：

1. ***此路徑不可再跳其它路徑。***
2. ***此路徑不可執行須等待的指令，除了 PlayV / PlayA / SpiPlay / SDelay。***
3. ***跳進此路徑後，除了該路徑的工作繼續執行外，其他所有工作將暫停。***

例.

```
[Path]
VT_BeforeTraining: SpiPlay(ch0, 0)             ; Voice Tag 訓練前，跳此 path。
```

4.8.5 Voice Tag 訓練失敗（VT_AddTagFail）

[NX1]

"**VT_AddTagFail**"路徑名稱僅能出現在 **[Path]** 段落中，Voice Tag 訓練失敗，會跳進此路徑。

例.

[Path]

VT_AddTagFail: SpiPlay(ch0, 0) **; Voice Tag 訓練失敗，跳此 path。**

4.9 錄音路徑

4.9.1 琴鍵錄音逾時（KRecord_Timeout）

[NX1]

“**KRecord_Timeout**”路徑名稱僅能出現在 **[Path]** 段落中，當琴鍵錄音超過時間沒有 InstNoteON / InstNoteOff 發生時，會跳進此路徑。

例.

[Path]

KRecord_Timeout: PlayKS($Rec0, 0) **; 發生琴鍵錄音逾時時，使用音色 0 播放琴鍵錄音。**

4.10 聲音偵測路徑

4.10.1 聲音偵測（Sound_Detected）

[NX1]

“**Sound_Detected**”路徑名稱僅能出現在 **[Path]** 段落中，當有偵測到聲音時，會跳進此路徑。

例.

[Option]

SoundDetect = Enable

SoundDetect_High_Threshold = 300

SoundDetect_Low_Threshold = 300

SoundDetect_Active_Time = 80ms

SoundDetect_Mute_Time = 650ms

[Path]

TR1: SoundDetect_On **; 啟用聲音偵測。**

Sound_Detected: PA.0 = 1 **; 偵測到聲音時，PA.0 輸出 1。**

Sound_Muted: PA.0 = 0 **; 當不再偵測到聲音超過 150ms 時，PA.0 輸出 1。**

4.10.2 聲音靜默（Sound_Muted）

[NX1]

“**Sound_Muted**”路徑名稱僅能出現在 **[Path]** 段落中，當不再偵測到聲音時，會跳進此路徑。

4.11 音高偵測路徑

4.11.1 音高偵測（Pitch_Detected）

[NX1]

“**Pitch_Detected**”路徑名稱僅能出現在 **[Path]** 段落中，當偵測到音高時，會跳進此路徑。

例.

```
[Option]
PitchDetect = Tone                    ; 偵測 Pure-Tone。
PitchDetect_Range = Middle            ; 偵測範圍 1000 ~ 3000Hz。
[Variable]
Var16: Pitch_Value
[Path]
TR1: PitchDetect_On                   ; 開啟音高偵測。
TR2: PitchDetect_Off                  ; 關閉音高偵測。
Pitch_Detected: ReadPitch(Pitch_Value)  ; 偵測到 1000 ~ 3000Hz 內的 pure-tone，
                                        ; 將偵測到的音高儲存至 Pitch_Value。
```

4.12 IO 擴充晶片路徑

4.12.1 IO 擴充晶片通訊失敗（IoExp_CommFail）

[NY5+ / NX1]

“IoExp_CommFail”路徑名稱僅能出現在 **[Path]** 段落中，當系統與 I/O 擴展晶片通訊失敗時，會跳進此路徑。

例.

```
[Path]
IoExp_CommFail: R0 = IoExp_ErrId      ; 當系統與 I/O 擴展晶片通訊失敗時，讀取通訊失敗的 I/O
                                        擴展晶片的 ID 到 R0。
```

5 *Q-Code* 指令（*Q-Code* Command）

Q-Code 提供有 4-bit 指令和 8-bit 指令。4-bit 指令所使用的 RAM 為 **Ri**，8-bit 指令所使用的 RAM 為 **Xi**。Xi 由兩個 4-bit RAM 所組成，對應方式如下：

X(i) = [X(i)L, X(i)H] = [R(i*2), R(i*2+1)]

例. **X10 = [R20, R21]**，其中 R20 / X10L 為 X10 低位的 nibble，R21 / X10H 為 X10 高位的 nibble。

完整的對應表請參考 Ri 與 Xi 對應表。

注意：

1. ***NY9T001A/004A 不提供 8-bit 乘除法指令。***
2. ***NX1 不預先定義 Ri / Xi 等變數，使用者需自行於 [Variable]中定義所需的變數。***

5.1 算數邏輯指令（Arithmetic Logic Command）

Arithmetic Logic Command			
Var1 = Var2	Var1 = Var2 + Var3	Var1 = Var2 – Var3	Var1 = Var2 * Var3
Var1 = Var2 / Var3	Var1 = Var2 % Var3	Var++	Var--
Var1 = Var2 & Var3	Var1 = Var2 | Var3	Var1 = Var2 ^ Var3	Var1 = Var2 << Var3
Var1 = Var2 >> Var3	!Var	Var=RandomL	Var=RandomH
Var=Random	-	-	-

5.1.1 變數運算（Variable Operation）

Q-Code 支援變數的賦值與運算操作，變數可以為：

- 1-bit 運算子：Ri.n / Xi.n / Pin
- 4-bit 運算子：Ri / Port
- 8-bit 運算子：Xi
- **[Variable]** 中定義的變數
- 系統變數

變數之間可進行賦值及下表所列的運算操作。

No.	變數	註釋	範例
1	Var1 = Var2	賦值操作	R0 = 0x4, R0 = R1, R0 = RandomL, R0 = RandomH
2	Var1 = Var2 + Var3	加法運算	R2 = R0 + 5, R2 = 5 + R0
3	Var1 = Var2 – Var3	減法運算	R2 = R0 – 3, R2 = 5 – R0
4	Var1 = Var2 * Var3	乘法運算	X1 = R0 * R1,

No.	變數	註釋	範例
	[Var1, Var2] = Var3 * Var4		X1 = R0 * 5 [R0, R1] = R2 * 15
5	Var1 = Var2 / Var3 [Var1, Var2] = Var3 / Var4	除法運算 第二種除法運算的商數存於 Var1，餘數存於 Var2	R2 = R0 / 5, X2 = X1 / R0, [R2, R1] = R0 / 5
6	Var1 = Var2 % Var3	餘數運算	R2 = R1 % 15
7	Var++	遞增運算	R0++
8	Var--	遞減運算	PA--
9	Var1 = Var2 & Var3	AND 運算	R2 = R0 & R1
10	Var1 = Var2 | Var3	OR 運算	R2 = R0 | R1
11	Var1 = Var2 ^ Var3	XOR 運算	R2 = R0 ^ R1
12	Var1 = Var2 << Var3	左移運算	R2 = R0 << 2, R2 = R0 << R1
13	Var1 = Var2 >> Var3	右移運算	R2 = R0 >> 2, R2 = R0 >> R1
14	!Var	反向運算	!R0, !R0.3

注意：

1. *可用的系統變數如下表：*

系統變數名稱	說明	支援性	範例
Vol	音量	NY5 / NY5+ / NY6 / NY7 / NX1	Vol = R1
V_Chx_Vol	語音的通道音量	NY5+ / NY6 / NY7	V_Ch0_Vol = X0
M_Chx_Vol	Melody 的個別通道音量	NY5+ / NY6 / NY7 / NX1	M_Ch10_Vol = R0
SPIVol	SPIPlay 播放的語音音量	NY6B / NY6C / NY7	SPIVol = R0
Px	IO Port	全系列	PA = R0 ^ 0x9
PxM	IO Port 輸出入狀態	NY4 / NY5+ / NY6 / NY7 / NY9T / NX1	PAM = PFM | 0x1
PxM.n	IO 腳位輸出入狀態	NY4 / NY5+ / NY6 / NY7 / NY9T / NX1	PAM.0 = PFM.0
RandomL	亂數的低位 nibble 唯讀，不可使用於 = 左邊。	全系列	R0 = RandomL ^ 0x5
RandomH	亂數的高位 nibble 唯讀，不可使用於 = 左邊。	全系列	R0 = RandomH | 0x9
Random	亂數 唯讀，不可使用於 = 左邊。	全系列	X0 = Random ^ 0x33

2. ***1-bit / 4-bit / 8-bit 變數互相運算時若是將較高 bit 的變數或運算結果賦值於較低 bit 的變數時，多餘的 bit 會被截去。***

Ex. **X0 = 0xFA, R3 = X0 ; R3 = 0xA**

5.1.2 Var = RandomL

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

產生亂數，並將結果的 low-nibble 存入 Var。亂數範圍由 Random 選項決定。

5.1.3 Var=RandomH

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

產生亂數，並將結果的 high-nibble 存入 Var。亂數範圍由 Random 選項決定。

5.1.4 Var=Random

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

產生亂數，並將結果存入 Var。當未指定 Min / Max 時，亂數範圍為 0 ~ Random 選項，若指定 Min / Max 時，亂數範圍為 Min ~ Max - 1。

Var=Random

Var=Random({Min, }Max)

Var：用來儲存亂數的變數。

Min：亂數最小值。若不指定則為 0。

Max：最大接受 0xFFFFFFFF。

Note:

1. ***NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 不支援設定 Min / Max 參數。***
2. ***NX1 指定 Min / Max 時，產生的亂數不受 Random 選項影響。***

5.2 流程控制指令（Flow Control Command）

Flow Control Command				
Var = Var ? Path	Var != Var ? Path	Var >= Var ? Path	Var <= Var ? Path	Var > Var ? Path
Var < Var ? Path	Px = data?Path	Px != data?Path	Px.n = 0?Path	Px.n != 0?Path
Px.n = 1?Path	Px.n != 1?Path	Vol = n?Path	Vol !=n ?Path	-
MixCtrl = data?Path	MixCtrl != data ?Path	RandomL=data?Path	RandomH=data?Path	
RandomL!=data?Path	RandomH!=data?Path	Random=data?Path	Random!=data?Path	
Px[1 X 0 X]?Path	ChUsed?Path	Voice?Path	PaueV?Path	Melody?Path
PauseM?Path	Delay? Path	PauseD?Path	Action?Path	PauseA?Path
PWMIO?Path	PausePWM?Path	HoldPWM?Path	SPIPlay?Path	SPIPause?Path
Pause?Path	Record?Path	KRecord?Path	Recorded?Path	PlayK?Path
EraseR?Path	VR_VAD?Path	LEDStr?Path	LEDSync?Path	LEDText?Path
Animaltalks_Record?Path		Animaltalks_Play?Path		-
Animalsings_Record?Path		Animalsings_Play?Path		Calibrate?Path
IoExp_Exists?Path		CheckSum?Path	SPI_Checksum?Path	
!Px[1 X 0 X]?Path	!ChUsed?Path	!Voice?Path	!PaueV?Path	!Melody?Path
!PauseM?Path	!Delay? Path	!PauseD?Path	!Action?Path	!PauseA?Path
!PWMIO?Path	!PausePWM?Path	!HoldPWM?Path	!SPIPlay?Path	!SPIPause?Path
!Pause?Path	!Record?Path	!KRecord?Path	!Recorded?Path	!PlayK?Path
!EraseR?Path	!VR_VAD?Path	!LEDStr?Path	!LEDSync?Path	!LEDText?Path
!Animaltalks_Record?Path		!Animaltalks_Play?Path		-
!Animalsings_Record?Path		!Animalsings_Play?Path		!Calibrate?Path
!IoExp_Exists?Path		!CheckSum?Path	!SPI_Checksum?Path	
If-Else	-	-	-	-
Switch(Ri) = [Path0, Path1, Path2,..Path15]		Switch(Px) = [Path0, Path1, Path2,..Path15]		
Switch(RandomL) = [Path0, Path1,..Path15]		Switch(RandomH) = [Path0, Path1,..Path15]		
Switch(Px[d x d x]) = [Path0, Path1, Path2….Path15]				
Switch(Xi)=[Path0, Path1, Path2,..Path255]		Switch(Random)=[Path0, Path1,Path2,..Path255]		
While	Do-While	For	-	-

語法：

R op T ? True_Branch { : False_Branch }

R：RAM 或變數（Variable）。

Op：運算子，見下列表格。

T：R 或常數（Constant）。常數可以接受 2 進制，10 進制與 16 進制的數值。

?：判斷符號。

True_Branch { : False_Branch }：可以是 Path 名稱或 ASM。

{ }：中括弧裏面的 False 子句可以被省略。

條件跳躍指令是用條件式的 True 或 False 值來選擇應執行成立分支或不成立分支，若不成立分支省略時且條件式之值為 False，則會跳到目前路徑的下一個步驟。

分支路徑除了使用一般路徑名稱，亦可使用以大括號包住的一連串指令集合。指令集合會被自動展開為匿名路徑。底下的範例示範此種寫法，左右兩邊結果是等價的。

P1: R0=1? P2, P3 P2: PlayV(ch0, $V0), P4 P3: PlayV(ch0, $V1), P4 P4: PlayV(ch0, $V2)	P1: R0=1? {PlayV(ch0, $V0) } : {PlayV(ch0, $V1)}, PlayV(ch0, $V2)

5.2.1 比較運算（Comparison）

比較運算	定 義	例 子
Var1 = Var2	Var1 等於 Var2	R0 = 2 ?TRUE, X0 = R1 ?TRUE, R0.0 = 1 ? True
Var != Var	Var1 不等於 Var2	R0 != R1?TRUE ; R0 != 3 ?TRUE ; PA.0 = 1 ?TRUE
Var >= Var	Var1 大於等於 Var2	Vol >= R1?TRUE ; R0 >= 3 ?TRUE
Var > Var	Var1 大於 Var2	Random > 128 ?TRUE ; R0 > 5 ?TRUE
Var <= Var	Var1 小於等於 Var2	R0 <= R1?TRUE ; R0 <= 5 ?TRUE
Var < Var	Var1 小於 Var2	R0 < R1 ?TRUE ; R0 = 7, R0 < 5 ?TRUE

Q-Code 支援變數的比較，變數可以為：

- 1-bit 運算子：Ri.n / Xi.n / Px.n
- 4-bit 運算子：Ri / Px
- 8-bit 運算子：Xi
- **[Variable]** 中定義的變數。
- 系統變數

注意：可用的系統變數如下表：

系統變數名稱	說明	支援性	范例
Vol	音量	NY5 / NY5+ / NY6 / NY7 / NX1	Vol = R1
V_Chx_Vol	語音的通道音量	NY5+ / NY6 / NY7	V_Ch0_Vol = X0
M_Chx_Vol	Melody 的個別通道音量	NY5+ / NY6 / NY7 / NX1	M_Ch10_Vol = R0
SPIVol	SPIPlay 播放的語音音量	NY6B / NY6C / NY7	SPIVol = R0
Px	IO Port	全系列	PA = R0 ^ 0x9
PxM	IO Port 輸出入狀態	NY4 / NY5+ / NY6 / NY7 /	PAM = PFM | 0x1

		NY9T / NX1	
PxM.n	IO 腳位輸出入狀態	NY4 / NY5+ / NY6 / NY7 / NY9T / NX1	PAM.0 = PFM.0
RandomL	亂數的低位 nibble 唯讀，不可使用於 = 左邊。	全系列	R0 = RandomL ^ 0x5
RandomH	亂數的高位 nibble 唯讀，不可使用於 = 左邊。	全系列	R0 = RandomH | 0x9
Random	亂數 唯讀，不可使用於 = 左邊。	全系列	X0 = Random ^ 0x33

5.2.2 Px = data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Px 的值為某個值時，跳至 Path。

例. **PA = 3? Path** **; PA 的值為 3 時，跳至 Path。**

5.2.3 Px != data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Px 的值不為某個值時，跳至 Path。

例. **PA != 3? Path** **; PA 的值不為 3 時，跳至 Path。**

5.2.4 Px.n = 0?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Px 的某一個腳的值為 0 時，跳至 Path。

例. **PA.0 = 0? Path** **; PA.0 的值為 0 時，跳至 Path。**

5.2.5 Px.n != 0?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Px 的某一個腳的值不為 0 時，跳至 Path。

例. **PA.0 != 0? Path** **; PA.0 的值不為 0 時，跳至 Path。**

5.2.6 Px.n = 1?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Px 的某一個腳的值為 1 時，跳至 Path。

例. **PA.0 = 1? Path** **; PA.0 的值為 1 時，跳至 Path。**

5.2.7 Px.n != 1?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Px 的某一個腳的值不為 1 時，跳至 Path。

例. **PA.0 != 1? Path** **; PA.0 的值不為 1 時，跳至 Path。**

5.2.8 Vol = n?Path

[NY5 / NY5+ / NY6 / NY7 / NX1]

若是目前音量等於 n 的時候，則跳躍至 Path。n=0~15。

例. 若音量等於第五階時。

Vol=5?True **; 音量等於第 5 階時，則跳躍至“True”。**

5.2.9 Vol != n?Path

[NY5 / NY5+ / NY6 / NY7 / NX1]

若是目前音量不等於 n 的時候，則跳躍至 Path。n=0~15。

例. 若音量等於第五階時。

Vol!=5?True **; 音量不等於第 5 階時，則跳躍至“True”。**

5.2.10 MixCtrl = data?Path

[NY5]

MixCtrl 設定值等於 data，則跳躍到 Path。

例.

```
MixCtrl=0x2?Set_MixCtrl      ; MixCtrl 值若是等於 0x2，則跳躍至“Set_MixCtrl”。
Set_ MixCtrl: MixCtrl(0,0,4,0)   ; 設定成 Channel2=100%音量。
```

5.2.11 MixCtrl != data?Path

[NY5]

MixCtrl 設定值不等於 data，則跳躍到 Path。

例.

```
MixCtrl!=0xE?Set_MixCtrl     ; MixCtrl 值若是不等於 0xE，則跳躍至“Set_MixCtrl”。
Set_MixCtrl: MixCtrl(1,1,1,1)    ; 設定成 Ch0 / Ch1 / Ch2 / Ch3=25%音量。
```

5.2.12 RandomL = data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

這是由隨機產生器產生的隨機數，隨機數可在 **[Option]** 頁面中設定。RandomH 提供高位的 nibble 而 RandomL 提供低位的 nibble。

隨機產生器的 Low-Nibble 若是等於 data，則跳躍至 Path。data=0~15。

例. 隨機數產生器所產生的隨機數，若是 RandomL 等於 0xA，則跳躍到路徑。

RandomL=0xA?True ; 取到的值若是等於 Low-Nibble 則跳躍至“True”。

5.2.13 RandomH = data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

這是由隨機產生器產生的隨機數，隨機數可在 **[Option]** 頁面中設定。RandomH 提供高位的 nibble 而 RandomL 提供低位的 nibble。

隨機產生器的 High-Nibble 若是等於 data，則跳躍至 Path。data=0~15。

例. 隨機數產生器所產生的隨機數，若是 RandomH 等於 0x3，則跳躍到該路徑。

RandomH=0x3?True ; 取到的值若是等於 High-Nibble 則跳躍至“True”。

5.2.14 RandomL != data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

這是由隨機產生器產生的隨機數，隨機數可在 **[Option]** 頁面中設定。RandomH 提供高位的 nibble 而 RandomL 提供低位的 nibble。

隨機產生器的 Low-Nibble 若是不等於 data，則跳躍至 Path。data=0~15。

例. 隨機數產生器所產生的隨機數，若是 RandomL 不等於 0xA，則跳躍到該路徑。

RandomL!=0xA?True ; 取到的值若是不等於 Low-Nibble 則跳躍至“True”。

5.2.15 RandomH != data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

這是由隨機產生器產生的隨機數，隨機數可在 **[Option]** 頁面中設定。RandomH 提供高位的 nibble 而 RandomL 提供低位的 nibble。

隨機產生器的 High-Nibble 若是不等於 data，則跳躍至 Path。data=0~15。

例. 隨機數產生器所產生的隨機數，若是 RandomH 不等於 0x3，則跳躍到該路徑。

RandomH!=0x3?True ; 取到的值若是不等於 High-Nibble 則跳躍至“True”。

5.2.16 Random = data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

這是由隨機產生器產生的隨機數，隨機數可在 **[Option]** 頁面中設定。隨機產生器所產生的數值若是等於 data，則跳躍至 Path。

Random=Data?Path

Random(Min, Max)=Data?Path

Random(Max)=Data?Path

Data: number.

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支援 0 ~ 255。
- NX1 支援 0 ~ 0xFFFFFFFE，當有指定 Min / Max 時，則為 Min ~ Max - 1。

Min：亂數最小值。若不指定則為 0。

Max：最大接受 0xFFFFFFFF。

Note:

1. NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 不支援設定 Min / Max 參數。

2. NX1 指定 Min / Max 時，產生的亂數不受 Random 選項影響。

例. Random=0x3A

Random=0x3A?True ；取到的值若是等於 **0x3A**，則跳躍至**“True”**。

例. NX1

Random(0x10, 0x1000)=0x100?True ；取到的值若是等於 **0x100**，則跳轉至**“True”**。

5.2.17 Random != data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

這是由隨機產生器產生的隨機數，隨機數可在 **[Option]** 頁面中設定。隨機產生器所產生的數值若是不等於 data，則跳躍至 Path。

Random != Data?Path

Random(Max) != Data?Path

Random(Min, Max) != Data?Path

Data: number.

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支援 0 ~ 255。
- NX1 支援 0 ~ 0xFFFFFFFE，當有指定 Min / Max 時，則為 Min ~ Max - 1。

Min：亂數最小值。若不指定則為 0。

Max：最大接受 0xFFFFFFFF。

Note:

1. NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 不支援設定 Min / Max 參數。

2. NX1 指定 Min / Max 時，產生的亂數不受 Random 選項影響。

例. Random=0x2A

Random!=0x3A?True ；取到的值若是不等於 **0x3A**，則跳躍至**“True”**。

例. NX1

Random(0x10, 0x1000)!=0x100?True ; 取到的值若是不等於 0x100，則跳躍至“True”。

5.2.18 Px[1 X 0 X]?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

可以使用該指令來讀取 Port 的特定 IO 腳位的狀態，並且跳躍到對應的路徑。

例. 讀取 PB.1 與 PB.3。

TR1: PB[1 X 0 X]?True ; PB.3=1 and PB.1=0，則跳躍至“True”。

True: PlayV(Ch0, $V0)

5.2.19 ChUsed(Ch)?Path

[NY5+ / NY6 / NY7 / NX1]

指定的通道若是有 Voice 或 Melody 播放中的話，則跳躍至 Path。

Ch：指定語音通道，不指定，則視同全部通道。

- NY5+支援 Ch= Ch0 ~ Ch3。
- NY6 支援 Ch= Ch0 ~ Ch5。
- NY7 支援 Ch= Ch0 ~ Ch7。
- NX1 支援 Ch= Ch0 ~ Ch7。

例. 使用 Ch4 來播放 Voice，使用 Ch0 ~ Ch3 來播放 Melody。

TR1: PlayV(ch4,$V0)

TR2: PlayM($M0)

TR3: ChUsed(3)?Channel_Playing ; 通道 3 正在使用中，則跳躍至“Channel_Playing”。

5.2.20 Voice(Ch)?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

若是指定的語音通道有 Voice 正在播放中的話，則跳躍至 Path。

Ch：指定語音通道，若不指定，則視同全部語音通道。

- NY4 不能指定通道。
- NY5 / NY5+支援 Ch= Ch0 ~ Ch3。
- NY6 支援 Ch= Ch0 ~ Ch5。
- NY7 支援 Ch= Ch0 ~ Ch7。
- NX1 支援 Ch= Ch0 ~ Ch7。

注意：此指令僅在程式中有使用 PlayV 或 PlayVS 時才有效。

例.

PowerOn: InputState

TR1: PlayV(ch0,$V0), PlayV(ch1,$V1), PlayV(ch2,$V2), PlayV(ch3, $V3)

```
TR2: Voice?Voice_Playing          ; 任一通道正在播放中，則跳躍至“Voice_Playing”。
TR3: Voice(0)?Ch0_Playing         ; 通道 0 正在播放中，則跳躍至“CH0_Playing”。
TR4: Voice(1)?Ch1_Playing         ; 通道 1 正在播放中，則跳躍至“CH1_Playing”。
TR5: Voice(ch2)?Ch2_Playing       ; 通道 2 正在播放中，則跳躍至“CH2_Playing”。
TR6: Voice(ch3)?Ch3_Playing       ; 通道 3 正在播放中，則跳躍至“CH3_Playing”。
```

5.2.21 PauseV(Ch)?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

若是指定的語音通道目前正在執行 PauseV 功能，則跳躍至 Path。

Ch：指定語音通道，若不指定，則視同全部語音通道。

- NY4 不能指定通道。
- NY5 / NY5+支援 Ch= Ch0 ~ Ch3。
- NY6 支援 Ch= Ch0 ~ Ch5。
- NY7 支援 Ch= Ch0 ~ Ch7。
- NX1 支援 Ch= Ch0 ~ Ch7。

注意：此指令僅在程式中有使用 PlayV 或 PlayVS 及 PauseV 時才有效。

例. 任一語音通道的 Voice 正在暫停播放中。

```
PowerOn: InputState
TR1: PlayV(ch0,$V0), PlayV(ch1,$V1), PlayV(ch2,$V2), PlayV(ch3, $V3)
TR2: PauseV?VoicePause            ; 任一通道正在暫停播放中，則跳躍至“VoicePause”。
TR3: PauseV(0)?Ch0_Pause          ; 通道 0 正在暫停播放中，則跳躍至“CH0_Pause”。
TR4: PauseV(1)?Ch1_Pause          ; 通道 1 正在暫停播放中，則跳躍至“CH1_Pause”。
TR5: PauseV(ch2)?Ch2_Pause        ; 通道 2 正在暫停播放中，則跳躍至“CH2_Pause”。
TR6: PauseV(ch3)?Ch3_Pause        ; 通道 3 正在暫停播放中，則跳躍至“CH3_Pause”。
```

5.2.22 Melody?Path

[NY5 / NY5+ / NY6 / NY7 / NX1]

若是 Melody 正在播放中，則跳躍至 Path。

例. Melody 正在播放中。

```
Melody?True                       ; Melody 播放中，則跳躍至“True”。
```

5.2.23 PauseM?Path

[NY5 / NY5+ / NY6 / NY7 / NX1]

若是目前正在執行 PauseM 功能，則跳躍至 Path。

例. Melody 正在暫停播放中。

```
PauseM?True                       ; Melody 暫停中，則跳躍至“True”。
```

5.2.24 Delay(n)?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

若是指定的前景或背景目前正在執行時間延遲功能，則跳躍至 Path。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，則表示所有的 Delay。

例.

Delay(0)?True ; 前景 Delay 執行中則跳躍至“True”。

Delay?True ; 所有前後背景中正在有 Delay 執行中則跳躍至“True”。

5.2.25 PauseD(n)?Path

[NY9T]

若是指定的前景或背景目前正在執行時間暫停功能，則跳躍至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，則表示所有的 Delay。

例. **PauseD(0)?True_Path** ; 前景 Delay 暫停執行中，則跳躍至“True_Path”。

例. **PauseD?True_Path** ; 所有前後背景中有 Delay 正在暫停執行中，則跳躍至“True_Path”。

5.2.26 Action(Ch)?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

若指定的 Action 通道正在播放中，則跳躍至 Path。

Ch：指定 Action 通道，若不指定，則表示全部 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 Ch1 ~ Ch8
- NX1 支援 Ch1 ~ Ch32。

注意：此指令僅在程式中有使用 PlayA 或 PlayAS 時才有效。

例. Action 通道 1 正在執行中。

Action(1)?StopOut ; Action 通道 1 正在執行中，則跳躍至“StopOut”。

Action?StopOut ; Action 的任一通道正在執行中，則跳躍至“StopOut”。

5.2.27 PauseA(Ch)?Path

[NY5+ / NY6 / NY7 / NY9T / NX1]

若是指定的 Action 通道目前正在執行 PauseA 功能，則跳躍至 Path。

Ch：指定 Action 通道，若不指定，則視同全部 Action 通道。

- NY5+ / NY6 / NY7 / NY9T 支持 Ch1 ~ Ch8
- NX1 支援 Ch1 ~ Ch32。

注意：此指令僅在程式中有使用 PlayA 或 PlayAS 時才有效。

例. 任一通道的 Action 正在暫停播放中。

```
PowerOn: InputState
TR1: PlayA(PC.0,ch1,$A1), PlayA(PC.0,ch2,$A1), PlayA(PC.0,ch3,$A1), PlayA(PC.0,ch4,$A1)
TR2: PauseA?ActionPause          ; 任一通道正在暫停播放中，則跳躍至“ActionPause”。
TR3: PauseA(1)?Ch1_Pause         ; 通道 1 正在暫停播放中，則跳躍至“CH1_Pause”。
TR4: PauseA(2)?Ch2_Pause         ; 通道 2 正在暫停播放中，則跳躍至“CH2_Pause”。
TR5: PauseA(ch3)?Ch3_Pause       ; 通道 3 正在暫停播放中，則跳躍至“CH3_Pause”。
TR6: PauseA(ch4)?Ch4_Pause       ; 通道 4 正在暫停播放中，則跳躍至“CH4_Pause”。
```

5.2.28 PWMIO?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

若是 PWMIO 正在播放中，則跳躍至 Path。語法如下：

PWMIO?Path

PWMIO(n)?Path

PWMIO(Pin)?Path

PWMIO(Ch)?Path

PWMIO(PIO)?Path

n：0=前景，1=背景 1，2=背景 2。n 不指定，則表示所有的 PlayPWM / PlayPWMS / PWMOut / PWMOutS。

Pin：指定腳位，表示該腳位是否在使用 PWMOut / PWMOutS 輸出 PWMIO。

- **Px.n:** IC 腳位。
- **EXPx_Py.n**：I/O 擴展晶片的腳位。

Ch：PlayPWM / PlayPWMS 所輸出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的輸出狀態。

注意：

1. ***NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支援指定 n。***
2. ***NY4 / NY5 / NY6 / NY7 / NY9T 不支援指定 Pin。***
3. ***NY4 / NY5 / NY6 / NY7 / NY9T / NX1 不支援 Ch / PIO 參數。***

例. PWMIO 正在播放中。

```
PWMIO?True                       ; PWMIO 播放中，則跳躍至“True”。
PWMIO(0)?True_Path               ; 前景 PlayPWM/PlayPWMS 執行中，則跳躍至
                                   “True_Path”。
PWMIO(PB.0)?True_Path            ; 若有 PWMOut 輸出於 PB.0，則跳躍至“True_Path”。
PWMIO(EXP0_P1.2)?True_Path       ; 若有 PWMOut 輸出於第 0 顆 I/O 擴展晶片 P1.2 腳位，則跳
                                   躍至“True_Path”。
```

5.2.29 PausePWM?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

若是目前正在執行 PausePWM 功能，則跳躍至 Path。語法如下：

PausePWM?Path

PausePWM(n)?Path

PausePWM(Pin)?Path

PausePWM(Ch)?Path

PausePWM(PIO)?Path

n：0=前景，1=背景 1，2=背景 2。n 不指定，則表示所有的 PausePWM / PlayPWMS / PWMOut / PWMOutS。

Pin：指定腳位，表示該腳位是否暫停由 PWMOut / PWMOutS 輸出的 PWM-IO。

- **Px.n**：IC 腳位。
- **EXPx_Py.n**：I/O 擴展晶片的腳位。

Ch：PlayPWM / PlayPWMS 所輸出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的輸出狀態。

注意：

1. NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支援指定 n。

2. NY4 / NY5 / NY6 / NY7 / NY9T 不支援指定 Pin。

3. NY4 / NY5 / NY6 / NY7 / NY9T / NX1 不支援 Ch / PIO 參數。

例. PWMIO 正在暫停播放中。

```
PausePWM?True_Path          ; PWMIO 暫停中，則跳躍至“True”。
```

5.2.30 HoldPWM(n)?Path

[NY9T]

若是指定的前景或背景目前正在執行 HoldPWM 功能，則跳躍至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，則表示所有的 HoldPWM。

注意：NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支援此指令。

```
例. HoldPWM(0)?True_Path    ; 前景 PlayPWM / PlayPWMS 暫停中，則跳躍至“True_Path”。
例. HoldPWM?True_Path       ; 所有前後背景中有 PlayPWM / PlayPWMS 正在暫停執行中，則跳躍至“True_Path”。
```

5.2.31 SPIPlay(Ch)?Path

[NX1]

若是指定的語音通道有 SPIPlay 正在播放中的話，則跳躍至 Path。

Ch：指定語音通道，若不指定，則視同透過 SPI 播放的全部語音通道與全部 Action 通道。

- NX1 支援 Ch0 ~ Ch3。

*注意：此指令僅在程式中有使用 **SPIPlay / SPIPlayS** 時才有效。*

例.

PowerOn: InputState

TR1: SPIPlay(ch0, 0), SPIPlay(ch1, 1), SPIPlay(ch2, 2), SPIPlay(ch3, 3)

TR2: SPIPlay?Voice_Playing ; 任一通道正在播放中，則跳躍至“Voice_Playing”。

TR3: SPIPlay(ch0)?Ch0_Playing ; 通道 0 正在播放中，則跳躍至“CH0_Playing”。

TR4: SPIPlay(ch1)?Ch1_Playing ; 通道 1 正在播放中，則跳躍至“CH1_Playing”。

TR5: SPIPlay(ch2)?Ch2_Playing ; 通道 2 正在播放中，則跳躍至“CH2_Playing”。

TR6: SPIPlay(ch3)?Ch3_Playing ; 通道 3 正在播放中，則跳躍至“CH3_Playing”。

5.2.32 SPIPause(Ch)?Path

[NX1]

若是指定的語音通道目前正在執行 SPIPause 功能，則跳躍至 Path。

Ch：指定語音通道，若不指定，則視同透過 SPI 播放的全部語音通道與全部 Action 通道。

- NX1 支援 Ch0 ~ Ch3。

*注意：此指令僅在程式中有使用 **SPIPlay / SPIPlayS / SPIPause** 時才有效。*

例. 任一語音通道的 SPIPlay 正在暫停播放中。

PowerOn: InputState

TR1: SPIPlay(ch0,0), SPIPlay(ch1,1), SPIPlay(ch2,2), SPIPlay(ch3, 3)

TR2: SPIPause?VoicePause ; 任一通道正在暫停播放中，則跳躍至“VoicePause”。

TR3: SPIPause(ch0)?Ch0_Pause ; 通道 0 正在暫停播放中，則跳躍至“CH0_Pause”。

TR4: SPIPause(ch1)?Ch1_Pause ; 通道 1 正在暫停播放中，則跳躍至“CH1_Pause”。

TR5: SPIPause(ch2)?Ch2_Pause ; 通道 2 正在暫停播放中，則跳躍至“CH2_Pause”。

TR6: SPIPause(ch3)?Ch3_Pause ; 通道 3 正在暫停播放中，則跳躍至“CH3_Pause”。

5.2.33 Pause(n)?Path

[NY9T]

若是指定的前景或背景目前正在執行任一播放或時間延遲功能，則跳躍至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，則表示所有前景與背景。

*注意：**NY4 / NY5 / NY5+ / NY6 / NY7 / NX1** 不支援此指令。*

例. **Pause(0)?True_Path** ; 前景暫停執行中，則跳躍至“True_Path”。

例. **Pause?True_Path** ; 所有前後背景的動作正在暫停執行中，則跳躍至“True_Path”。

5.2.34 Record?Path

[NX1]

若是錄音功能正在執行中，則跳躍至 Path。

*例.***Record?True_Path** **；錄音功能執行中，則跳躍至“True_Path”。**

5.2.35 KRecord?Path

[NX1]

若是琴鍵錄音功能正在執行中，則跳躍至 Path。

注意：當 KRecord / KRecordS 被執行時，必須要觸發 InstNoteOn 或 InstNoteOff 一次，才會正式進入琴鍵錄音模式。

例.

KRecord?True_Path **；錄音功能執行中，則跳躍至“True_Path”。**

5.2.36 Recorded($Rec)?Path

[NX1]

若是指定的錄音區段內含錄音資料，則跳躍至 Path。

例.

Recorded($Rec0)?True_Path **;Rec0 錄音區段內含錄音資料，則跳躍至“True_Path”。**

5.2.37 PlayK?Path

[NX1]

若是琴鍵回放功能正在執行中，則跳躍至 Path。

例.

PlayK?True_Path **；錄音功能執行中，則跳躍至“True_Path”。**

5.2.38 EraseR?Path

[NX1]

若是正在擦除 SPI Flash 上的錄音區段中，則跳躍至 Path。

例.

EraseR?True_Path **；若正在擦除錄音區段，則跳躍至“True_Path”。**

5.2.39 VR_VAD?Path

[NX1]

若是 VAD 超過門檻值，則跳躍至 Path。

例.

VR_VAD?True_Path ；若 VAD 超過門檻值，則跳躍至“True_Path”。

5.2.40 LEDStr?Path

[NX1]

若是正在進行 Single-String 燈串播放，則跳躍至 Path。

LEDStr?Path

LEDStr(Px.n)?Path

Px.n：指定檢查是否正在進行 Single-String 燈串播放的腳位，不指定則檢查所有 Single-String 輸出腳位。

例.

Path1: LEDStr?True_Path ；若燈串播放中，則跳躍至“True_Path”。

5.2.41 LEDSync?Path

[NX1]

若是正在進行 Sync-Play 燈串播放，則跳躍至 Path。

LEDSync?Path

例.

Path1: LEDSync?True_Path ；若燈串播放中，則跳躍至“True_Path”。

5.2.42 LEDText?Path

[NX1]

若是正在進行 Scrolling-Text 燈串播放，則跳躍至 Path。

LEDText?Path

例.

Path1: LEDText?True_Path ；若燈串播放中，則跳躍至“True_Path”。

5.2.43 Animaltalks_Record?Path

[NX1]

若是正在進行動物變音的錄音，則跳躍至 Path。

Animaltalks_Record?Path

例.

Path1: Animaltalks_Record?True_Path ；若動物變音錄音中，則跳躍至“True_Path”。

5.2.44 Animaltalks_Play?Path

[NX1]

若是正在進行動物變音的播放，則跳躍至 Path。

Animaltalks_Play?Path

例.

Path1: Animaltalks_Play?True_Path ；若動物變音播放中，則跳躍至“True_Path”。

5.2.45 Animalsings_Record?Path

[NX1]

若是正在進行動物變音的錄音，則跳躍至 Path。

Animalsings_Record?Path

例.

Path1: Animalsings_Record?True_Path ；若動物唱歌錄音中，則跳躍至“True_Path”。

5.2.46 Animalsings_Play?Path

[NX1]

若是正在進行動物變音的播放，則跳躍至 Path。

Animalsings_Play?Path

例.

Path1: Animalsings_Play?True_Path ；若動物唱歌播放中，則跳躍至“True_Path”。

5.2.47 Calibrate?Path

[NY9T]

若是觸摸鍵校正功能正在執行中，則跳躍至 Path。

例.

Calibrate?True_Path **；AutoJudge_Calibrate 或 Enforce_Calibrate 功能執行中，則跳躍至“True_Path”。**

5.2.48 IoExp_Exists?Path

[NY5+ / NX1]

檢查 ExpID 對應的 I/O 擴展晶片是否存在，若存在則跳至“Path”。

IoExp_Exists(ExpId)?Path

ExpId：I/O 擴展晶片 ID。

5.2.49 CheckSum?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

用於快速檢測 ROM DATA 是否有遭到損毀。CheckSum?Path 指令會自動判讀 ROM DATA 的正確性，若是 ROM DATA 正確則會跳至“Path”。

注意：

1. *當 CheckSum Command 被執行時，會停止當前所有動作。在執行 CheckSum 時，會需要一些時間。且執行完畢前將沒有任何回應。*
2. *NX1 EF 系列不會檢查整塊 ROM 區，因 UDR 區塊為可更新資料，不在檢查範圍之內。*

例.

TEST_ROUTINE: Checksum?OK, PlayV(Ch1,Fail.wav) **；Check Sum 正確會跳至“OK”。**
；反之，則執行下一個指令。

OK：PlayV(Ch1, OK.wav)

5.2.50 SPI_CheckSum?Path

[NX1]

用於快速檢測 SPI Flash 資料是否有遭到損毀。SPI_CheckSum?Path 指令會自動判讀 SPI Flash 資料的正確性，若是資料正確則會跳至“Path”。

注意：當 SPI CheckSum 指令執行時，會停止當前所有動作。在執行 SPI CheckSum 時，會需要一些時間。且執行完畢前將沒有任何回應。

例.

TEST_ROUTINE: SPI_CheckSum?OK, PlayV(Ch1, $Fail) **；SPI checksum 正確會跳至“OK”。**
；反之，則執行下一個指令。

OK: PlayV(Ch1, $OK)

5.2.51 !Px[1 X 0 X]?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

可以使用該指令來讀取 Port 的特定 IO 腳位的狀態，並且跳躍到對應的路徑。

例. 讀取 PB.1 與 PB.3。

```
TR1: !PB[1 X 0 X]?False_Path          ; PB.3!=1 或 PB.1!=0，則跳躍至“False_Path”。
False_Path: PlayV(Ch0, $V0)
```

5.2.52 !ChUsed(Ch)?Path

[NY5+ / NY6 / NY7 / NX1]

指定的通道若無 Voice 或 Melody 播放中的話，則跳躍至 Path。

Ch：指定語音通道，不指定，則視同全部通道。

- NY5+支援 Ch=Ch0 ~ Ch3。
- NY6 支援 Ch= Ch0 ~ Ch5。
- NY7 支援 Ch=Ch0 ~ Ch7。
- NX1 支援 Ch=Ch0 ~ Ch7。

例. 使用 Ch4 來播放 Voice，使用 Ch0 ~ Ch3 來播放 Melody。

```
TR1: PlayV(Ch4, $V0)
TR2: PlayM($M0)
TR3: !ChUsed(Ch3)?Channel_NotPlaying   ; 通道 3 非使用中，則跳躍至“Channel_NotPlaying”。
```

5.2.53 !Voice(Ch)?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

若是指定的語音通道無 Voice 正在播放中的話，則跳躍至 Path。

Ch：指定語音通道，若不指定，則視同全部語音通道。

- NY4 不能指定通道。
- NY5 / NY5+支援 Ch=Ch0 ~ Ch3。
- NY6 支援 Ch=Ch0 ~ Ch5。
- NY7 支援 Ch=Ch0 ~ Ch7。
- NX1 支援 Ch=Ch0 ~ Ch7。

注意：此指令僅在程式中有使用 PlayV 或 PlayVS 時才有效。

例.

```
PowerOn: InputState
TR1: PlayV(Ch0, $V0), PlayV(Ch1, $V1), PlayV(Ch2, $V2), PlayV(Ch3, $V3)
TR2: !Voice?Voice_NotPlaying          ; 任一通道非播放中，則跳躍至“Voice_NotPlaying”。
TR3: !Voice(Ch0)?Ch0_NotPlaying       ; 通道 0 非播放中，則跳躍至“Ch0_NotPlaying”。
```

TR4: !Voice(Ch1)?Ch1_NotPlaying ; 通道 1 非播放中，則跳躍至“Ch1_NotPlaying”。
TR5: !Voice(Ch2)?Ch2_NotPlaying ; 通道 2 非播放中，則跳躍至“Ch2_NotPlaying”。
TR6: !Voice(Ch3)?Ch3_NotPlaying ; 通道 3 非播放中，則跳躍至“Ch3_NotPlaying”。

5.2.54 !PauseV(Ch)?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

若是指定的語音通道目前並未執行 PauseV 功能，則跳躍至 Path。

Ch：指定語音通道，若不指定，則視同全部語音通道。

- NY4 不能指定通道。
- NY5 / NY5+支援 Ch=Ch0 ~ Ch3。
- NY6 支援 Ch=或 Ch0 ~ Ch5。
- NY7 支援 Ch=Ch0 ~ Ch7。
- NX1 支援 Ch=Ch0 ~ Ch7。

注意：此指令僅在程式中有使用 PlayV 或 PlayVS 及 PauseV 時才有效。

例. 任一語音通道的 Voice 正在暫停播放中。

PowerOn: InputState
TR1: PlayV(Ch0, $V0), PlayV(Ch1, $V1), PlayV(Ch2, $V2), PlayV(Ch3, $V3)
TR2: !PauseV?VoiceNotPause ; 所有通道並未暫停播放，則跳躍至“VoiceNotPause”。
TR3: !PauseV(Ch0)?Ch0_NotPause ; 通道 0 並未暫停播放，則跳躍至“Ch0_NotPause”。
TR4: !PauseV(Ch1)?Ch1_NotPause ; 通道 1 並未暫停播放，則跳躍至“Ch1_NotPause”。
TR5: !PauseV(Ch2)?Ch2_NotPause ; 通道 2 並未暫停播放，則跳躍至“Ch2_NotPause”。
TR6: !PauseV(Ch3)?Ch3_NotPause ; 通道 3 並未暫停播放，則跳躍至“Ch3_NotPause”。

5.2.55 !Melody?Path

[NY5 / NY5+ / NY6 / NY7 / NX1]

若是 Melody 非播放中，則跳躍至 Path。

例. Melody 非播放中。

!Melody?False_Path ; Melody 非播放中，則跳躍至“False_Path”。

5.2.56 !PauseM?Path

[NY5 / NY5+ / NY6 / NY7 / NX1]

若是目前未執行 PauseM 功能，則跳躍至 Path。

例. Melody 正在暫停播放中。

!PauseM?False_Path ; Melody 並未暫停，則跳躍至“False_Path”。

5.2.57 !Delay(n)?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

若是指定的前景或背景目前並未執行時間延遲功能，則跳躍至 Path。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，則表示所有的 Delay。

例.

```
!Delay(0)?False_Path        ; 前景無 Delay 執行中則跳躍至“False_Path”。
!Delay?False_Path           ; 所有前後背景中皆無 Delay 執行中則跳躍至
                            ; “False_Path”。
```

5.2.58 !PauseD(n)?Path

[NY9T]

若是指定的前景或背景目前並未執行 PauseD，則跳躍至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，則表示所有的 Delay。

```
例. !PauseD(0)?False_Path   ; 前景並未執行 PauseD 中，則跳躍至“False_Path”。
例. !PauseD?False_Path      ; 所有前後背景中並未執行 PauseD，則跳躍至
                            ; “False_Path”。
```

5.2.59 !Action(Ch)?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

若指定的 Action 通道不在播放中，則跳躍至 Path。

Ch：指定 Action 通道，若不指定，則表示全部 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 Ch1 ~ Ch8
- NX1 支援 Ch1 ~ Ch32。

注意：此指令僅在程式中有使用 PlayA 或 PlayAS 時才有效。

例. Action 通道 1 正在執行中。

```
!Action(Ch1)?StartOut       ; Action 通道 1 不在執行中，則跳躍至“StartOut”。
!Action?StartOut            ; Action 的全部通道都不在執行中，則跳躍至“StartOut”。
```

5.2.60 !PauseA(Ch)?Path

[NY5+ / NY6 / NY7 / NY9T / NX1]

若是指定的 Action 通道目前並未執行 PauseA 功能，則跳躍至 Path。

Ch：指定 Action 通道，若不指定，則視同全部 Action 通道。

- NY5+ / NY6 / NY7 / NY9T 支持 1 ~ 8 或 Ch1 ~ Ch8
- NX1 支援 1 ~ 20 或 Ch1 ~ Ch32。

注意：此指令僅在程式中有使用 PlayA 或 PlayAS 時才有效。

例. 任一通道的 Action 正在暫停播放中。

```
PowerOn: InputState
TR1: PlayA(PC.0, Ch1, $A1), PlayA(PC.0, Ch2, $A1), PlayA(PC.0,Ch3, $A1), PlayA(PC.0,Ch4,$A1)
TR2: !PauseA?ActionNotPause      ; 所有通道並未暫停播放，則跳躍至“ActionNotPause”。
TR3: !PauseA(1)?Ch1_NotPause     ; 通道 1 並未暫停播放，則跳躍至“Ch1_NotPause”。
TR4: !PauseA(2)?Ch2_NotPause     ; 通道 2 並未暫停播放，則跳躍至“Ch2_NotPause”。
TR5: !PauseA(ch3)?Ch3_NotPause   ; 通道 3 並未暫停播放，則跳躍至“Ch3_NotPause”。
TR6: !PauseA(ch4)?Ch4_NotPause   ; 通道 4 並未暫停播放，則跳躍至“Ch4_NotPause”。
```

5.2.61 !PWMIO?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

若是 PWMIO 非播放中，則跳躍至 Path。語法如下：

!PWMIO?Path

!PWMIO(n)?Path

!PWMIO(Pin)?Path

!PWMIO(Ch)?Path

!PWMIO(PIO)?Path

n： 0=前景，1=背景 1，2=背景 2。n 不指定，則表示所有的 PlayPWM / PlayPWMS / PWMOut / PWMOutS。

Pin：指定腳位，表示該腳位是否在使用 PWMOut / PWMOutS 輸出 PWMIO。

- **Px.n：**IC 腳位。
- **EXPx_Py.n：**I/O 擴展晶片的腳位。

Ch：PlayPWM / PlayPWMS 所輸出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的輸出狀態。

注意：

1. ***NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支援指定 n。***
2. ***NY4 / NY5 / NY6 / NY7 / NY9T 不支援指定 Pin。***
3. ***NY4 / NY5 / NY6 / NY7 / NY9T / NX1 不支援 Ch / PIO 參數。***

例. PWMIO 正在播放中。

```
!PWMIO?False_Path                ; PWMIO 非播放中，則跳躍至“False_Path”。
!PWMIO(0)?False_Path             ; 前景 PlayPWM/PlayPWMS 非執行中，則跳躍至“False_Path”。
!PWMIO(PB.0)?False_Path          ; 若無 PWMOut 輸出於 PB.0，則跳躍至“False_Path”。
!PWMIO(EXP0_P1.2)?False_Path     ; 若無 PWMOut 輸出於第 0 顆 I/O 擴展晶片 P1.2 腳位，則跳躍至“False_Path”。
```

5.2.62 !PausePWM?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

若是目前並未執行 PausePWM 功能，則跳躍至 Path。語法如下：

!PausePWM?Path

!PausePWM(n)?Path

!PausePWM(Pin)?Path

!PausePWM(Ch)?Path

!PausePWM(PIO)?Path

n：0=前景，1=背景 1，2=背景 2。n 不指定，則表示所有的 PausePWM / PlayPWMS / PWMOut / PWMOutS。

Pin：指定腳位，表示該腳位是否暫停由 PWMOut / PWMOutS 輸出的 PWM-IO。

- **Px.n**：IC 腳位。
- **EXPx_Py.n**：I/O 擴展晶片的腳位。

Ch：PlayPWM / PlayPWMS 所輸出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的輸出狀態。

注意：

1. NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支援指定 n。

2. NY4 / NY5 / NY6 / NY7 / NY9T 不支援指定 Pin。

3. NY4 / NY5 / NY6 / NY7 / NY9T / NX1 不支援 Ch / PIO 參數。

例. PWMIO 正在暫停播放中。

```
!PausePWM?False_Path                ; PWMIO 並未暫停，則跳躍至“False_Path”。
```

5.2.63 !HoldPWM(n)?Path

[NY9T]

若是指定的前景或背景目前並未執行 HoldPWM 功能，則跳躍至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，則表示所有的 HoldPWM。

注意：NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支援此指令。

```
例. !HoldPWM(0)?False_Path          ; 前景 PlayPWM / PlayPWMS 非暫停中，則跳躍至
                                      “False_Path”。
例. !HoldPWM?False_Path             ; 所有前後背景中無 PlayPWM / PlayPWMS 正在暫停執行
                                      中，則跳躍至“False_Path”。
```

5.2.64 !SPIPlay(Ch)?Path

[NX1]

若是指定的語音通道無 SPIPlay 正在播放中的話，則跳躍至 Path。

Ch：指定語音通道，若不指定，則視同透過 SPI 播放的全部語音通道與全部 Action 通道。

- NX1 支援 Ch0 ~ Ch3。

注意：此指令僅在程式中有使用 SPIPlay / SPIPlayS 時才有效。

例.

PowerOn: InputState

TR1: SPIPlay(Ch0, 0), SPIPlay(Ch1, 1), SPIPlay(Ch2, 2), SPIPlay(Ch3, 3)

TR2: !SPIPlay?Voice_NotPlaying ; 無 SPIPlay 播放，則跳躍至“Voice_NotPlaying”。

TR3: !SPIPlay(Ch0)?Ch0_NotPlaying ; 通道 0 無 SPIPlay 播放，則跳躍至“Ch0_NotPlaying”。

TR4: !SPIPlay(Ch1)?Ch1_NotPlaying ; 通道 1 無 SPIPlay 播放，則跳躍至“Ch1_NotPlaying”。

TR5: !SPIPlay(Ch2)?Ch2_NotPlaying ; 通道 2 無 SPIPlay 播放，則跳躍至“Ch2_NotPlaying”。

TR6: !SPIPlay(Ch3)?Ch3_NotPlaying ; 通道 3 無 SPIPlay 播放，則跳躍至“Ch3_NotPlaying”。

5.2.65 !SPIPause(Ch)?Path

[NX1]

若是指定的語音通道目前並非在執行 SPIPause 功能，則跳躍至 Path。

Ch：指定語音通道，若不指定，則視同透過 SPI 播放的全部語音通道與全部 Action 通道。

- NX1 支援 Ch0 ~ Ch3。

注意：此指令僅在程式中有使用 SPIPlay / SPIPlayS / SPIPause 時才有效。

例. 任一語音通道的 SPIPlay 正在暫停播放中。

PowerOn: InputState

TR1: SPIPlay(Ch0, 0), SPIPlay(Ch1, 1), SPIPlay(Ch2, 2), SPIPlay(Ch3, 3)

TR2: !SPIPause?VoiceNotPause ; 所有通道皆非暫停播放中，則跳躍至
; “VoiceNotPause”。

TR3: !SPIPause(Ch0)?Ch0_NotPause ; 通道 0 非暫停播放中，則跳躍至“Ch0_NotPause”。

TR4: !SPIPause(Ch1)?Ch1_NotPause ; 通道 1 非暫停播放中，則跳躍至“Ch1_NotPause”。

TR5: !SPIPause(Ch2)?Ch2_NotPause ; 通道 2 非暫停播放中，則跳躍至“Ch2_NotPause”。

TR6: !SPIPause(Ch3)?Ch3_NotPause ; 通道 3 非暫停播放中，則跳躍至“Ch3_NotPause”。

5.2.66 !Pause(n)?Path

[NY9T]

若是指定的前景或背景並未執行 Pause，則跳躍至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，則表示所有前景與背景。

注意：NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支援此指令。

例. **!Pause(0)?False_Path ; 前景並未執行 Pause，則跳躍至“False_Path”。**

例. **!Pause?False_Path ; 所有前後背景並未執行 Pause，則跳躍至“False_Path”。**

5.2.67 !Record?Path

[NX1]

若是錄音功能未在執行中，則跳躍至 Path。

例. **!Record?False_Path ; 無錄音功能執行中，則跳躍至“False_Path”。**

5.2.68 !KRecord?Path

[NX1]

若是琴鍵錄音功能非執行中，則跳躍至 Path。

注意：當 KRecord / KRecordS 被執行時，必須要觸發 InstNoteOn 或 InstNoteOff 一次，才會正式進入琴鍵錄音模式。

例.

!KRecord?False_Path ; 錄音功能非執行中，則跳躍至“False_Path”。

5.2.69 !Recorded($Rec)?Path

[NX1]

若是指定的錄音區段內不包含錄音資料，則跳躍至 Path。

例.

!Recorded($Rec0)?False_Path ; Rec0 錄音區段內不包含錄音資料，則跳躍至
; “False_Path”。

5.2.70 !PlayK?Path

[NX1]

若是琴鍵回放功能非執行中，則跳躍至 Path。

例.

!PlayK?False_Path ; 琴鍵回放功能非執行中，則跳躍至“False_Path”。

5.2.71 !EraseR?Path

[NX1]

若非正在擦除錄音區段中，則跳躍至 Path。

例.

!EraseR?False_Path **；若錄音區段擦除非進行中，則跳躍至“False_Path”。**

5.2.72 !VR_VAD?Path

[NX1]

若是 VAD 未超過門檻值，則跳躍至 Path。

例.

!VR_VAD?False_Path **；若 VAD 未超過門檻值，則跳躍至“False_Path”。**

5.2.73 !LEDStr?Path

[NX1]

若未進行 Single-String 燈串播放，則跳躍至 Path。

!LEDStr?Path

!LEDStr(Px.n)?Path

Px.n：指定檢查是否正在進行 Single-String 燈串播放的腳位，不指定則檢查所有 Single-String 輸出腳位。

例.

Path1: !LEDStr?False_Path **；若燈串非播放中，則跳躍至“False_Path”。**

5.2.74 !LEDSync?Path

[NX1]

若是未進行 Sync-Play 燈串播放，則跳躍至 Path。

!LEDSync?Path

例.

Path1: !LEDSync?False_Path **；若燈串 Sync-Play 非播放中，則跳躍至“False_Path”。**

5.2.75 !LEDText?Path

[NX1]

若是未進行 Scrolling-Text 燈串播放，則跳躍至 Path。

!LEDText?Path

例.

Path1: !LEDText?False_Path **；若燈串非播放中，則跳躍至“False_Path”。**

5.2.76 !Animaltalks_Record?Path

[NX1]

若非正在進行動物變音的錄音，則跳躍至 Path。

!Animaltalks_Record?Path

例.

Path1: !Animaltalks_Record?False_Path ；若未進行動物變音錄音中，則跳躍至“False_Path”。

5.2.77 !Animaltalks_Play?Path

[NX1]

若是未在進行動物變音的播放，則跳躍至 Path。

!Animaltalks_Play?Path

例.

Path1: !Animaltalks_Play?False_Path ；若動物變音非播放中，則跳躍至“False_Path”。

5.2.78 !Animalsings_Record?Path

[NX1]

若非正在進行動物唱歌的錄音，則跳躍至 Path。

!Animalsings_Record?Path

例.

Path1: !Animalsings_Record?False_Path ；若未進行動物唱歌錄音中，則跳躍至“False_Path”。

5.2.79 !Animalsings_Play?Path

[NX1]

若是未在進行動物唱歌的播放，則跳躍至 Path。

!Animalsings_Play?Path

例.

Path1: !Animalsings_Play?False_Path ；若動物唱歌非播放中，則跳躍至“False_Path”。

5.2.80 !Calibrate?Path

[NY9T]

若是觸摸鍵校正功能非執行中，則跳躍至 Path。

例.

!Calibrate?False_Path ; AutoJudge_Calibrate 或 Enforce_Calibrate 功能皆非執行中，則跳躍至“False_Path”。

5.2.81 !IoExp_Exists?Path

[NY5+ / NX1]

檢查 ExpID 對應的 I/O 擴展晶片是否存在，若不存在則跳至“Path”。當未指定 ExpID 時則檢查所有的 ID。

!IoExp_Exists?Path

!IoExp_Exists(ExpId)?Path

ExpId：I/O 擴展晶片 ID。

5.2.82 !CheckSum?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

用於快速檢測 ROM DATA 是否有遭到損毀。!CheckSum?Path 指令會自動判讀 ROM DATA 的正確性，若是 ROM DATA 不正確則會跳至“Path”。

注意：

1. ***當 CheckSum Command 被執行時，會停止當前所有動作。在執行 CheckSum 時，會需要一些時間。且執行完畢前將沒有任何回應。***
2. ***NX1 EF 系列不會檢查整塊 ROM 區，因 UDR 區塊為可更新資料，不在檢查範圍之內。***

例.

TEST_ROUTINE: Checksum?NotOK, PlayV(Ch1, Pass.wav); Check Sum 不正確會跳至“NotOK”。

; 反之，則執行下一個指令。

NotOK：PlayV(Ch1, Fail.wav)

5.2.83 !SPI_CheckSum?Path

[NX1]

用於快速檢測 SPI Flash 資料是否有遭到損毀。!SPI_CheckSum?Path 指令會自動判讀 SPI Flash 資料的正確性，若是資料不正確則會跳至“Path”。

注意：當 SPI CheckSum 指令執行時，會停止當前所有動作。在執行 SPI CheckSum 時，會需要一些時間。且執行完畢前將沒有任何回應。

例.

TEST_ROUTINE: !SPI_CheckSum?NotOK, PlayV(Ch1, $Pass); SPI checksum 不正確會跳至“NotOK”。

; 反之，則執行下一個指令。

```
NotOK: PlayV(Ch1, $Fail)
```

5.2.84 If-Else

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

If-Else 是 Condition?Path 的另一種寫法。

```
Voice?{ PA.0 = 0 }:{ PA.0 = 1 }
```

可寫成如下程式片段：

```
If(Voice)
{
   PA.0 = 0
}
Else
{
   PA.0 = 1
}
```

如果 Condition 成立時執行 If 區塊內的程式，若 Condition 不成立則執行 Else 區塊內的程式。執行後繼續執行 If-Else 之後的程式。

If(Condition) { … }

If(Condition) { … } Else { … }

Condition: 判斷條件。

例. Random number lottery.

```
[Option]
Random=15
[Voice File]
V1 = win.wav
V2 = draw.wav
V3 = lose.wav
[Path]
P1: R0 = RandomL,
    If(R0 > 7)
    {
      If(R0 > 13)
      {
        PlayV(Ch0, $v1)
      }
      Else
```

```
    {
      PlayV(Ch0, $v0)
    }
  }
  Else
  {
    PlayV(Ch0, $v3)
  }
```

5.2.85 Switch(Ri) = [Path0, Path1, Path2,... Path15]

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Switch 指令會依照 Ri 的值切換目前的路徑到相對應的路徑。這個路徑可以是一個前景路徑，背景路徑或“X”（代表沒有動作）。當全部切換後的路徑都為“X”時，表示此時的路徑都可以跳過，換言之，這個 Switch Statement 的步驟也可以不必寫。若是在 **[Path]** 內沒有被定義到的會執行後面的指令。

Ri 的內容若是 0 則跳躍至 Path0，若為 1 則跳躍至 Path1，依此類推。

```
例. R0=3, Switch(R0)=[ Path1, Path2, X, Path3], Path4      ; R0=3，跳躍至 Path3。
例. R0=4, Switch(R0)=[ Path1, Path2, X, Path3], Path4      ; R0 超出 Switch command 定義的範
                                                           ; 圍。
```

5.2.86 Switch(Px) = [Path0, Path1, Path2,… Path15]

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

用法同 Switch(Ri)，Switch 指令會依照 Px 的值切換目前的路徑到相對應的路徑。

Px 的內容若是 0 則跳躍至 Path0，若為 1 則跳躍至 Path1，依此類推。

```
例. PB =3, Switch(PB)=[Path1, Path2, X, Path3], Path4      ; PB =3，跳至 Path3。
例. PB =4, Switch(PB)=[Path1, Path2, X, Path3], Path4      ; PB 超出 Switch command 定義的
                                                           ; 範圍，執行下一個指令。
```

5.2.87 Switch(RandomL) = [Path0, Path1, Path2,… Path15]

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Switch 指令會依照 RandomL 的值切換目前的路徑到相對應的路徑。這個路徑可以是一個前景路徑，背景路徑或“X”（代表沒有動作）。若是在 **[Path]** 內沒有被定義到的會執行後面的指令。

RandomL 的內容若是 0 則跳躍至 Path0，若為 1 則跳躍至 Path1，依此類推。

```
例. Switch(RandomL)=[ Path1, Path2, X, Path3], Path4 ; 依照 RandomL 值切換到對應路徑。
```

5.2.88 Switch(RandomH) = [Path0, Path1, Path2,… Path15]

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Switch 指令會依照 RandomH 的值切換目前的路徑到相對應的路徑。這個路徑可以是一個前景路徑，背景路徑或“X”（代表沒有動作）。若是在 **[Path]** 內沒有被定義到的會執行後面的指令。

RandomH 的內容若是 0 則跳躍至 Path0，若為 1 則跳躍至 Path1，依此類推。

例. **Switch(RandomH)=[Path1, Path2, X, Path3], Path4; 依照 RandomH 值切換到對應路徑。**

5.2.89 Switch(Xi) = [Path0, Path1, Path2,... Path255]

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

其用法同 4-bit 的 Switch(Ri)。Switch 指令會依照 Xi 的值切換目前的路徑到相對應的路徑。這個路徑可以是一個前景路徑，背景路徑或“X”（代表沒有動作）。

Xi 的內容若是 0 則跳躍至 Path0，若為 1 則跳躍至 Path1，依此類推。

例. **X0=3, Switch(X0)=[Path1, Path2, X, Path3], Path4 ; X0=3，跳至 Path3。**

例. **X0=4, Switch(X0)=[Path1, Path2, X, Path3], Path4 ; X0=4，超出 Switch 所定義的範圍，**
; 跳過 Switch Command，執行下一個
; 指令。

5.2.90 Switch(Random) = [Path0, Path1, Path2,... Path255]

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Switch 指令會依照 Random 的值切換目前的路徑到相對應的路徑。這個路徑可以是一個前景路徑，背景路徑或“X”（代表沒有動作）。Random 的內容若是 0 則跳躍至 Path0，若為 1 則跳躍至 Path1，依此類推。

例. **Random=3, Switch(Random)=[Path1, Path2, X, Path3], Path4 ; Randon=3 至 Path3。**

例. **Random=4, Switch(Random)=[Path1, Path2, X, Path3], Path4 ; Random=4，超出 Switch 所定**
; 義的範圍，跳過 Switch
; Command，執行下一個指令。

5.2.91 Switch(Px[d x d x]) = [Path0, Path1, Path2,...Path15]

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

可以使用該指令來讀取 Port 的特定 IO 腳位的狀態，並將 IO 狀態組合成一個數值，根據此數值跳躍到對應的路徑。Switch 指令會依照有效位“d”的值切換目前的路徑到相對應的路徑。這個路徑可以是一個前景路徑，背景路徑或“X”（代表沒有動作）。此功能主要用來達到 bonding option 的功能。

例. 讀取 PB.1 與 PB.3，兩個位元總共可以組合出四種結果。

TR1: Switch(PB[d x d x]) = [P0, P1, P2, P3] ; 利用 PB.1 與 PB.3 來組合出四種結果。

```
P0: PlayV(Ch0, $V0)                ; 當 PB.1 及 PB.3 皆為 0，執行 P0 路徑。
P1: PlayV(Ch0, $V2)                ; 當 PB.1 為 1 及 PB.3 為 0，執行 P1 路徑。
P2: PlayV(Ch0, $V8)                ; 當 PB.1 為 0 及 PB.3 為 1，執行 P2 路徑。
P3: PlayV(Ch0, $V10)               ; 當 PB.1 及 PB.3 皆為 1，執行 P3 路徑。
```

5.2.92 While

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

重複執行 {...} 中的程式，直到 Condition 不成立為止。

表達式的測試會在每次執行迴圈之前進行。因此，While 迴圈會執行零次或多次。

執行程序如下所示：

1. 評估 Condition。 如果 Condition 不成立，While 會結束並執行其後的指令。
2. 執行 {...} 中的程式。
3. 跳回步驟 1。

在 {...} 中執行路徑或是條件跳躍等指令，While 也會終止。

While(Condition) { ... }

Condition：判斷條件。*Q-Code* 支援的判斷式同條件跳躍指令。

5.2.93 Do-While

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Do-While 陳述式可讓您重複 {...} 內的程式，直到 Condition 不成立為止。

執行 {...} 中的程式後，會評估 Condition 是否成立。因此，迴圈主體一律至少執行一次。

執行程序如下所示：

1. 執行 {...} 中的程式。
2. 接下來會評估 Condition。 如果 Condition 不成立，Do-While 會結束並執行其後的指令。如果 Condition 仍成立，則會從步驟 1 開始重複執行 {...} 內的程式。

在 {...} 中執行路徑或是條件跳躍等指令，Do-While 也會終止。

Do { ... } While(Condition)

Condition：判斷條件。*Q-Code* 支援的判斷式同條件跳躍指令。

例. 將 1 到 9 的加總儲存至 X0。

```
[Path]
P1: X0 = 0, X1 = 1,
    Do
    {
      X0 = X0 + X1,
      X1++
```

```
} While(X1 < 10)
```

5.2.94 For

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

For 迴圈。

進入迴圈前，Var 會被設定為 Start，迴圈進行中會逐次遞增或遞減 Var 的值，並執行 {...} 中的程式，直到 Var 等於 End 為止。

1. 1..10 會產生 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 的結果。
2. 1..<10 會產生 1, 2, 3, 4, 5, 6, 7, 8, 9 的結果。
3. 1..10 step 3 會產生 1, 4, 7, 10 的結果。
4. 1..<10 step 3` 會產生 1, 4, 7 的結果。
5. 1..10 step -1 不會進入迴圈。

For(Var in [Start..End]) { ... }

For(Var in [Start..<End]) { ... }

For(Var in [Start..End] Step Increment) { ... }

For(Var in [Start..<End] Step Increment) { ... }

Var：Var 在迴圈進行時，會依照 Increment 的值進行遞增或遞減。

Start：迴圈變數 Var 的初始值。可接受常數值或是變數。

End：迴圈變數 Var 的最終值。可接受常數值或是變數。

Increment：每次迴圈執行時，會使用 Increment 的值進行遞增或遞減，僅接受常數值，可為負值。不指定則為 1。

例. NY5 / NY5+ / NY6 / NY7 / NX1

```
[Voice File]
V0 = 00.wav
V1 = 01.wav
V2 = 02.wav
V3 = 03.wav
[Path]
TR1:
For(r0 in [3..1] Step -1)                    ; 反序播放 V1 ~ V3，每次間隔 0.5s。
{
   PlayV(Ch0, r0),
   Delay(0.5s)
}
```

5.3 I/O 指令（I/O Command）

4-bit I/O Command				
Ri = Px	Ri = PxM	Ri.n = Px.n	PxM = data	PxM = Ri
PxM.n = 0	PxM.n = 1	Px = data	Px = Ri	Px = Py
!Px	!Px.n	Px.n = 0	Px.n = 1	Px.n = Ri.n
Px.n = Py.n	Px = Py + Ri	Px = Py – Ri	Px = Py | Ri	Px = Py ^ Ri
Px = Py & Ri	Ri = Px + Rj	Ri = Px – Rj	Ri = Px | Rj	Ri = Px ^ Rj
Ri = Px & Rj	Px = Py + data	Px = Py – data	Px = Py | data	Px = Py ^ data
Px = Py & data	Ri = Px + data	Ri = Px – data	Ri = Px | data	Ri = Px ^ data
Ri = Px & data	Px = [1 x 0 FD An]	Px.n= 1KHz(time)	-	-
8-bit I/O Command				
XiL = Px	XiH = Px	XiL.n = Px.n	XiH.n = Px.n	Xi.n = Px.n
Xi = [Px, Py]	Px = XiL	Px = XiH	Px.n = XiL.n	Px.n = XiH.n
Px.n = Xi.n	[Px, Py] = Xi	-	-	-

針對 IO 腳位操作指令，Px 可用 Port 如下：

- **NY4A**：PA。
- **NY4B**：PA ~ PB。
- **NY5A**：PA ~ PB。
- **NY5B**：PA ~ PD。
- **NY5C**：PA ~ PE。
- **NY5C450x / NY5C520x / NY5C640x / NY5C720x**：PA ~ PF。
- **NY5Q026A**：PA。
- **NY5Q020A / NY5Q040A**：PA ~ PB。
- **NY5Q046A / NY5Q080A / NY5Q160A**：PA ~ PC。
- **NY5Q060A / NY5Q092A / NY5Q172A**：PA ~ PD。
- **NY5Q342A**：PA ~ PE。
- **NY6A**：PA ~ PB。
- **NY6B**：PA ~ PD。
- **NY6C**：PA ~ PF。
- **NY7A**：PA ~ PB。
- **NY7B**：PA ~ PD。
- **NY7C**：PA ~ PF。
- **NY9T001A**：PE。
- **NY9T004A**：PA、PE。
- **NY9T008A**：PA、PB、PE、PF。
- **NY9T016A**：PA ~ PF。

- **NX1:** PA ~ PC。

5.3.1 Ri = Px

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 的值存到 Ri。

例. **PB= 0x3, R0=PB** **; R0=0x3**

5.3.2 Ri = PxM

[NY4 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Port 輸出入模式存到 Ri。

例. **PBM= 0x3, R0=PBM** **; R0=0x3**

5.3.3 Ri.n = Px.n

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 的第 n 個 bit，輸出到 Ri 的第 n 個 bit。

例. **PB=0x3, R0=0x0, R0=PB** **; R0=0x3**

5.3.4 PxM = data

[NY4 / NY5+ / NY6 / NY7 / NY9T / NX1]

設定 I/O Port 的輸出入模式。

例. **PBM=0x5** **; PB.0=輸入模式，PB.1=輸出模式**
; PB.2=輸入模式，PB.3=輸出模式

5.3.5 PxM = Ri

[NY4 / NY5+ / NY6 / NY7 / NY9T / NX1]

由 Ri 來設定 Port 的輸出入模式。

例. **R0=0x3, PBM=R0** **; PBM=0x3**

5.3.6 PxM.n = 0

[NY4 / NY5+ / NY6 / NY7 / NY9T / NX1]

設定 Port 第 n 腳的設定成輸出模式。

例. **PBM.0=0** **; PB.0=輸出模式**

5.3.7 PxM.n = 1

[NY4 / NY5+ / NY6 / NY7 / NY9T / NX1]

設定 Port 第 n 腳的設定成輸入模式。

例. **PBM.0=1** ; PB.0=輸入模式

5.3.8 Px = data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Px 輸出數值 data。

例. **PB=0x5** ; PB=0x5

5.3.9 Px = Ri

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Px 輸出 Ri 的內容值。

例. **R0=0x3, PB=R0** ; PB=0x3

5.3.10 Px = Py

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py Port 的內容值輸出到 Px Port。

例. **PB=0x3, PA=PB** ; PA=0x3

5.3.11 !Px

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 值反相後輸出。

例. **PB=0x3, !PB** ; PB=0xC

5.3.12 !Px.n

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 的第 n 個 bit，反相後輸出。

例. **PB=0x3, !PB.0** ; PB=0x2

5.3.13 Px.n = 0

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 的第 n 個 bit 設為 0。

例. **PB=0xD, PB.3=0** **; PB=0x5**

5.3.14 Px.n = 1

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 的第 n 個 bit 設為 1。

例. **PB=0x5, PB.3=1** **; PB=0xD**

5.3.15 Px.n = Ri.n

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Ri 的第 n 個 bit，輸出到 Px 的第 n 個 bit。

例. **PB=0x0, R0=0x5, PB.2=R0.0** **; PB=0x4**

5.3.16 Px.n = Py.n

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 的第 n 個 bit，輸出到 Px 的第 n 個 bit。

例. **PB=0x0, PA=0x5, PB.2=PA.0** **; PB=0x4**

5.3.17 Px = Py + Ri

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 與 Ri 的值相加後輸出到 Px。

例. **PB=0x3, R0=0x4, PB=PB+R0** **; PB=0x7**

5.3.18 Px = Py – Ri

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 與 Ri 的值相減後輸出到 Px。

例. **PB=0x3, R0=0x2, PB=PB-R0** **; PB=0x1**

5.3.19 Px = Py | Ri

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 與 Ri 的值做 OR 後輸出到 Px。

例. **PB=0x3, R0=0x4, PB=PB | R0** **; PB=0x7**

5.3.20 Px = Py ^ Ri

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 與 Ri 的值做 XOR 後輸出到 Px。

例. **PB=0x3, R0=0x4, PB=PB^R0 ; PB=0x7**

5.3.21 Px = Py & Ri

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 與 Ri 的值做 AND 後輸出到 Px。

例. **PB=0x3, R0=0x4, PB=PB&R0 ; PB=0x0**

5.3.22 Ri = Px + Rj

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 與 Rj 的值相加後存到 Ri。

例. **R0=0x3, PB=0x4, R1=PB+R0 ; R1=0x7**

5.3.23 Ri = Px – Rj

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 與 Rj 的值相減後存到 Ri。

例. **R0=0x3, PB=0x4, R1=PB-R0 ; R1=0x1**

5.3.24 Ri = Px | Rj

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 與 Rj 的值 OR 後存到 Ri。

例. **PB=0x3, R0=0x4, R1=PB|R0 ; R1=0x7**

5.3.25 Ri = Px ^ Rj

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 與 Rj 的值 XOR 後存到 Ri。

例. **PB=0x3, R0=0x4, R1=PB^R0 ; R1=0x7**

5.3.26 Ri = Px & Rj

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 與 Rj 的值 AND 後存到 Ri。

例. **PB=0x3, R0=0x4, R1=PB&R0 ; R1=0x0**

5.3.27 Px = Py + data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 與 data 的值相加後輸出到 Px。

例. **PB=0x3, PB=PB+0x4 ; PB=0x7**

5.3.28 Px = Py – data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 與 data 的值相減後輸出到 Px。

例. **PB=0x3, PB=PB-0x2 ; PB=0x1**

5.3.29 Px = Py | data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 與 data 的值做 OR 後輸出到 Px。

例. **PB=0x3, PB=PB | 0x4 ; PB=0x7**

5.3.30 Px = Py ^ data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 與 data 的值做 XOR 後輸出到 Px。

例. **PB=0x3, PB=PB^0x4 ; PB=0x7**

5.3.31 Px = Py & data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Py 與 data 的值做 AND 後輸出到 Px。

例. **PB=0x3, PB=PB&0x4 ; PB=0x0**

5.3.32 Ri = Px + data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 與 data 的值相加後存到 Ri。

例. **PB=0x3, R1=PB+0x4 ; R1=0x7**

5.3.33 Ri = Px – data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 與 data 的值相減後存到 Ri。

例. **PB=0x3, PB=PB-0x2 ; R1=0x1**

5.3.34 Ri = Px | data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 與 data 的值 OR 後存到 Ri。

例. **PB=0x3, R1=PB | 0x4 ; R1=0x7**

5.3.35 Ri = Px ^ data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 與 data 的值 XOR 後存到 Ri。

例. **PB=0x3, R1=PB^0x4 ; R1=0x7**

5.3.36 Ri = Px & data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 與 data 的值 AND 後存到 Ri。

例. **PB=0x3, R1=PB&0x4 ; R1=0x0**

5.3.37 Px = [1 X 0 FD A P Q]

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

此指令針對 Px 的輸出控制。各腳位可選擇以下種類的輸出方式：

參數	說明	NY4	NY5	NY5+	NY6	NY7	NY9T	NX1
X	表示該腳位維持原來的狀態。	✓	✓	✓	✓	✓	✓	✓
0	表示該腳位輸出低電位。	✓	✓	✓	✓	✓	✓	✓
1	表示該腳位輸出高電位。	✓	✓	✓	✓	✓	✓	✓
A	表示該腳位輸出 Action(該功能僅對於播放整個 VIO 檔有效)。	✓	✓	✓	✓	✓	✓	✕

參數	說明	NY4	NY5	NY5+	NY6	NY7	NY9T	NX1
P	表示該腳位輸出 PWM-IO(該功能僅對 PlayPWM 播放 MPIOx 時有效)。	×	×	✓	×	×	×	×
FD	表示隨播放的聲音音量大小變化（Flash Dynamic）。	✓	✓	✓	✓	✓	×	✓
Q	表示該腳位元跟隨 *Quick-IO* 的 QIO 信號變化。	✓	✓	✓	×	×	×	✓

例. 若要將 PB 輸出 15，則下達 PB=[1 1 1 1]。

例. 若要將 PB 輸出 5，則下達 PB=[0 1 0 1]。

例. 若要將 PB 輸出 5，且不影響其它 bit 原本的輸出，則下達 PB=[X 1 0 1]。

例. 若要將 PB.3 輸出跟隨著音量大小閃動，則下達 PB=[FD X X X]。

例. 若要將 PB.3 輸出跟隨著 *Quick-IO* 的信號閃動，則下達 PB=[Q X X X]。

例. 使用 IO 指令設定來播放整個 VIO。

```
[Action File]
VIO0 = Action1.vio                    ; Action Label:A1 ~ A8。
VIO1 = Action2.vio                    ; Action Label:A1 ~ A10。
[Action]
Compression = Enable
FrameRate = 80
Multi_Pins = [PA.0, PA.1, PA.2, PA.3, PB.0, PB.1, PB.2, PB.3]
MVIO0 = [VIO0.A1, VIO0.A2, VIO0.A3, VIO0.A4, VIO0.A5, VIO0.A6, VIO0.A7, VIO0.A8]
; MVIO0 使用 PA.0 ~ PB.3 輸出 VIO0 的 A1 ~ A8。
MVIO1 = [VIO1.A3, VIO1.A4, VIO1.A5, VIO1.A6, VIO1.A7, VIO1.A8, VIO1.A9, VIO1.A10]
; MVIO1 使用 PA.0 ~ PB.3 輸出 VIO1 的 A3 ~ A10。
[Path]
PowerOn: PA=[A A A A], PB=[A A A A]   ; 設定 PA 及 PB 輸出 Action。
Path1: PlayA(Ch4, $MVIO0)             ; 播放 MVIO0 於通道 4。
Path2: PlayA(Ch4, $MVIO1)             ; 播放 MVIO1 於通道 4。
```

5.3.38 [Px.n …] = [1 0 X Q]

[NX1]

針對個別腳位的輸出控制。各腳位可選擇的輸出方式如下：

參數	說明
X	表示該腳位維持原來的狀態。
0	表示該腳位輸出低電位。

參數	說明
1	表示該腳位輸出高電位。
FD	表示隨播放的聲音音量大小變化（Flash Dynamic）。
Q	表示該腳位元跟隨 *Quick-IO* 的 QIO 信號變化。

5.3.39 Px.n = 1KHz(time)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Px.n 輸出幾秒 1KHz 的方波。

Time：0~15，單位為秒。time=0，將為迴圈輸出。

注意：該指令被執行時，將不回應任何動作，直到執行完畢。

例. **PB.3=1KHz(5) or PB.3=1KHz(5s)** **; PB3 輸出 5 sec 1KHz 的方波。**

5.3.40 XiL = Px

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

將 Px 的值存到 Xi 的 Low-Byte。

例. **PB=0x5, X0L = PB** **; X0L = 0x5**

5.3.41 XiH = Px

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

將 Px 的值存到 Xi 的高位。

例. **PB=0x5, X0H = PB** **; X0H = 0x5**

5.3.42 XiL.n = Px.n

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

將 Px 的第 n 個 bit，輸出到 XiL 的第 n 個 bit。

例. **PB=0x5, X0=0x0, X0L.2 = PB.0** **; X0L = 0x4**

5.3.43 XiH.n = Px.n

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

將 Px 的第 n 個 bit，輸出到 XiH 的第 n 個 bit。

例. **PB=0x5, X0=0x0, X0H.2 = PB.0** **; XiH = 0x4**

5.3.44 Xi.n = Px.n

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Px 的第 n 個 bit，輸出到 Xi 的第 n 個 bit。

例. **PB=0x5, X0=0x0, X0.2 = PB.0 ; X0 = 0x4**

5.3.45 Xi = [Px, Py]

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

將 Px 的值存到 Xi 的 High-Nibble，將 Py 的值存到 Xi 到 Low-Nibble。

例. **X0 = [PB, PA] ; X0H = PB, X0L = PA**

5.3.46 Px = XiL

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

Px 輸出 Xi 的 Low-Byte。

例. **X0L=0x3, PB=X0L ; PB = 0x3**

5.3.47 Px = XiH

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

Px 輸出 Xi 的 High-Byte。

例. **X0H=0x3, PB=X0H ; PB = 0x3**

5.3.48 Px.n = XiL.n

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

將 XiL 的第 n 個 bit，輸出到 Px 的第 n 個 bit。

例. **PB=0x0, X0=0x5, PB.2=X0L.0 ; PB = 0x4**

5.3.49 Px.n = XiH.n

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

將 XiH 的第 n 個 bit，輸出到 Px 的第 n 個 bit。

例. **PB=0x0, X0=0x50, PB.2=X0H.0 ; PB = 0x4**

5.3.50 Px.n = Xi.n

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

將 Xi 的第 n 個 bit，輸出到 Px 的第 n 個 bit。

例. **PB=0x0, X0=0x5, PB.2=X0.0 ; PB = 0x4**

5.3.51 [Px, Py] = Xi

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

Px 輸出 Xi 的 High-Byte 高位值，Py 輸出 Xi 的 Low-Byte。

例. **X0H=0x3, X0L=0x1, [PB, PA]=X0 ; PB = 0x3, PA = 0x1**

5.4 路徑指令（Path Command）

Path Command				
ASM	C-Code	BG	BreakFG	StopFG
StopBG	StopBG1	StopBG2	StopBG3	StopBG4
StopBG5	Subroutine	Label(Pathname)	Macro	1ms_RET
4ms_RET	-	-	-	--

5.4.1 ASM

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

在 *Q-Code* 中使用者可以插入組合語言模組，在 **[ASM]** 段落中加入組合語言模組，並且在 *Q-Code* 中使用。

例. 使用 ASM 來控制 IC。

```
[ASM]
LED
{
mvla    b'0101'
mpg     pa_page
mvam    pa_state
mvam    pa
}
[Path]
TR1: LED                                    ; 呼叫 Assembly。
```

5.4.2 C-Code

[NX1]

在 *Q-Code* 中使用者可以插入 C 語言模組，在 **[C-Code]** 段落中加入 C 語言模組，並且在 *Q-Code* 中使

用。

例. 使用 C 來控制 IC。

[C-Code]

```
//#define USER_DEBUG_MODE 1
#ifdef USER_DEBUG_MODE
void debug_print(int data)
{
    // do something
}
#else
#define debug_print(x)
#endif
```

[Path]

PowerOn: debug_print(0)

若有開啟 XIP (Execute in place) 功能，C-Code 也可以指定部份程式存放於 XIP 區段。在函數原型宣告增加 __XIP 屬性即可。

例. 指定函數存放於 XIP 區段

[C-Code]

```
void func1(void) __XIP;
void func1(void)
{
    // do something
}
```

除了直接使用 __XIP 屬性，也可以藉由預先定義的 _SPI_XIP 巨集得知目前有無開啟 XIP (Execute in place) 功能，若有開啟 XIP 則 _SPI_XIP 巨集為 1，否則為 0。建議編寫 C-Code 的時候使用以下範例的方式參考 _SPI_XIP 巨集，並另行定義 XIP 屬性，如此一來當 *Q-Code* 的 XIP (Execute in place) 功能開啟/關閉狀態改變時，C-Code 不需要額外做修正。使用者自行定義的巨集請盡量避免以底線起頭，避免與 *Q-Code* 核心（Firmware kernel）功能衝突。

例. 參考 _SPI_XIP 巨集

[C-Code]

```
#if _SPI_XIP == 1
#   define XIP __XIP
#else
#   define XIP
#endif
void func1(void) XIP;
void func1(void)
{
    // do something
```

}

5.4.3 BG

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

背景呼叫指令，使用者可以在任一地方來呼叫背景。*Q-Code* 提供最多 3 層的的背景供 User 使用，在不同的層背景可以呼叫另一層的背景。

BG(BG1)

BG(BG1, BG2)

BG(BG1, BG2, BG3)

BG(BG1, BG2, BG3, BG4)

BG(BG1, BG2, BG3, BG4, BG5)

BG1：背景 1 路徑。

- **[Background1]**中定義的背景路徑名稱。
- **X**：不改變當前狀態。
- **OFF**：停止背景 1 路徑。

BG2：背景 2 路徑。

- **[Background2]**中定義的背景路徑名稱。
- **X**：不改變當前狀態。
- **OFF**：停止背景 2 路徑。

BG3：背景 3 路徑。

- **[Background3]**中定義的背景路徑名稱。
- **X**：不改變當前狀態。
- **OFF**：停止背景 3 路徑。

BG4：背景 4 路徑。

- **[Background4]**中定義的背景路徑名稱。
- **X**：不改變當前狀態。
- **OFF**：停止背景 4 路徑。

BG5：背景 5 路徑。

- **[Background5]**中定義的背景路徑名稱。
- **X**：不改變當前狀態。
- **OFF**：停止背景 5 路徑。

注意：

1. ***NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 僅支援 BG1 / BG2。***
2. ***NX1 支援 BG1 / BG2 / BG3 / BG4 / BG5。***
3. ***BG1 僅能呼叫 [Background1]背景的動作，不能呼叫 [Background2] 的動作。反之 BG2 / BG3 / BG4 / BG5 亦同。***

例. 利用背景指令來達成 3Hz 閃燈。

[Path]

PowerOn: BG(OUT1,OUT2), Delay(1), Stop ; 呼叫背景 1 與背景 2 後，會執行後面的指令。

[Background1]

OUT1: PB=[X X 1 1], Delay (0.166), PB=[X X 0 0], Delay (0.166), OUT1

[Background2]

OUT2: PB=[0 0 X X], Delay (0.166), PB=[1 1 X X], Delay (0.166), OUT2

5.4.4 BreakFG

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

該指令可以立即停止目前的前景的 PlayV、PlayM 或 Delay 等指令，並且執行前景指令後的下一個指令。

例: 檔案$V0 在播放 1 秒後停止，並立即執行後面的 StopV 指令。

[Path]

TR1: PlayV(Ch1, $V0)&[BG1], StopV

[Background1]

BG1: Delay(1), BreakFG

5.4.5 StopFG

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

可以停止目前當前前景的動作。

例: 利用 StopFG 指令來停止前景動作。

[Path]

PowerOn:

TR1: BG1, Label(Pathname), PB=[X X 1 1], Delay (0.166), PB=[X X 0 0], Delay (0.166), Pathname

[Background1]

BG1: Delay(10), StopFG ; 背景 1 延遲 10 秒後，即停止前景的動作。

5.4.6 StopBG

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

停止背景 1，背景 2 與背景 3 當前的動作。

例: 利用 StopBG 背景指令來停止背景動作。

[Path]

PowerOn: BG(OUT1,OUT2)

TR1: StopBG

[Background1]

OUT1: PB=[X X 1 1], Delay (0.166), PB=[X X 0 0], Delay (0.166), OUT1

[Background2]

OUT2: PB=[0 0 X X], Delay (0.166), PB=[1 1 X X], Delay (0.166), OUT2

5.4.7 StopBG1

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

停止背景 1 當前的動作。

例. 利用 StopBG1 背景指令來停止背景 1 動作，背景 2 依舊會動作。

[Path]

PowerOn: BG(OUT1,OUT2)

TR1: StopBG1

[Background1]

OUT1: PB=[X X 1 1], Delay (0.166), PB=[X X 0 0], Delay (0.166), OUT1

[Background2]

OUT2: PB=[0 0 X X], Delay (0.166), PB=[1 1 X X], Delay (0.166), OUT2

5.4.8 StopBG2

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

停止背景 2 當前的動作。

例. 利用 StopBG2 背景指令來停止背景 2 動作，背景 1 依舊會動作。

[Path]

PowerOn: BG(OUT1,OUT2)

TR1: StopBG2

[Background1]

OUT1: PB=[X X 1 1], Delay(0.166), PB=[X X 0 0], Delay(0.166), OUT1

[Background2]

OUT2: PB=[0 0 X X], Delay(0.166), PB=[1 1 X X], Delay(0.166), OUT2

5.4.9 StopBG3

[NX1]

停止背景 3 當前的動作。

例. 利用 StopBG3 背景指令來停止背景 3 動作，背景 1 與背景 2 依舊會動作。

[Path]

PowerOn: BG(OUT1,OUT2,OUT3)

TR1: StopBG3

[Background1]

OUT1: PB=[X X 1 1], Delay(0.166), PB=[X X 0 0], Delay(0.166), OUT1

[Background2]

OUT2: PB=[0 0 X X], Delay(0.333), PB=[1 1 X X], Delay(0.333), OUT2

[Background3]

OUT3: PB=[0 0 X X X X], Delay(0.5), PB=[1 1 X X X X], Delay(0.5), OUT3

5.4.10 StopBG4

[NX1]

停止背景 4 當前的動作。

5.4.11 StopBG5

[NX1]

停止背景 5 當前的動作。

5.4.12 Subroutine

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

在 *Q-Code* 中，使用者可以插入 *Q-Code* 副程式，在 **[Subroutine]** 段落中加入 *Q-Code* 模組，並且在 *Q-Code* 中使用。

例. 使在 *Q-Code* 中插入副程式。

[Subroutine]

SUB:PA=0xF, DELAY(0.5), PA=0x0, DELAY(0.5), PlayV(Ch1, $V1)

[Path]

TR1:SUB, PlayV(Ch1, $V2) **；執行完“SUB”後，會執行 PlayV(Ch1, $V2)。**

5.4.13 Label(Pathname)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

可以在程式中不需要使用“：”來設定路徑名稱，來減少程式分段。

例.

TR1: PlayV(Ch1, $V0), Label(Pathname), PlayV(Ch1, $V1) , Pathname

；播放完$V0 後，不停播放$V1。

等同以下程式

TR1: PlayV(Ch1, $V0), Label_Loop

Label_Loop : PlayV(Ch1, $V1) , Label_Loop **；播放完$V0 後，不停播放$V1。**

5.4.14 Macro

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

在 *Q-Code* 中，使用者可以插入巨集副程式，在 **[Macro]** 段落中加入巨集副程式，並在 *Q-Code* 使用。

例. 在 *Q-Code* 中插入巨集。

```
[Macro]
MAC:PA=0xF, Delay(0.166)

[Path]
TR1: MAC, PlayV(Ch1, $V1)                    ; 執行完 Macro “MAC” 後，會執行 PlayV(Ch1,$V1)。
```

5.4.15 1ms_RET

[NY9T]

1ms 路徑返回。

注意：

1. *進入 1ms 路徑，最後一定要執行 1ms_RET 指令，否則系統將會出現不能預期的錯誤。*
2. *若是 TimeBase=4ms，則 1ms 路徑不會被執行。*

例.

```
[Option]
TimeBase = 1ms

[Path]
PowerOn: Delay(10), PA=0
1ms: !PA, 1MS_RET                            ; 執行完!PA 後，執行 1MS_RET 時，會返回主程序。
```

5.4.16 4ms_RET

[NY9T]

4ms 路徑返回。

注意：進入 4ms 路徑，最後一定要執行 4ms_RET 指令，否則系統將會出現不能預期的錯誤。

例.

```
[Option]
TimeBase = 1ms or 4ms

[Path]
PowerOn: Delay(10), PA=0
4ms: !PA, 4MS_RET                            ; 執行完!PA 後，執行 4MS_RET 時，會返回主程序。
```

5.5 語音指令（Voice Command）

Voice Command				
PlayV	PlayVS	Voicename	WaitVN(Ch)	PauseV(Ch)
ResumeV(Ch)	StopV(Ch)	FreqCH = nK	V_Chx_Freq = nK	V_Chx_Vol = n
V_Chx_Vol = Xi	Xi = V_Chx_Vol	SBC_Loop_On	SBC_Loop_Off	ADPCM_Loop_On
ADPCM_Loop_Off	ADPCM_UpSampling		ADM_Loop_On	ADM_Loop_Off
ADM_UpSampling	PCM_Loop_On	PCM_Loop_Off	ReadFileCountV	-

5.5.1 PlayV / PlayVS

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

是 *Q-Code* 提供使用者簡易的指令來達成播放語音檔的方式。指令格式如下：

PlayV ({Ch,} Voice {, PlaySpeed})

PlayV ({Ch,} Voice {, Storage})

PlayVS ({Ch,} Voice {, PlaySpeed})

PlayVS ({Ch,} Voice {, Storage})

Voicename{@CH}

PlayV：待 Voice 檔案播放結束後，執行下一個指令。

PlayVS：Voice 檔案開始播放，立即執行下一個指令。

Ch：指定要播放的語音通道。

- NY4 不能指定播放通道。
- NY5 / NY5+支援的語音通道為 Ch0 ~ Ch3 或是 0 ~ 3。
- NY6 支援的語音通道為 Ch0 ~ Ch5 或是 0 ~ 5。
- NY7 支援的語音通道為 Ch0 ~ Ch7 或是 0 ~ 7。
- NX1 支援的語音通道為 Ch0 ~ Ch7 或是 0 ~ 7。

Voice：指定要播放的語音。

- Voice Label。
- Ri（可支援到 1~4 個 Ri）或是 Xi（可支援到 1~2 個 Xi），由 Ri / Xi 的值決定要播放的是第幾個 voice 檔案。若是 Ri / Xi 帶有"++"則在播放後將其遞增。
- 在 **[Record]** 中定義的錄音區段。

PlaySpeed：指定要播放的速度，若不指定則使用語音檔案的採樣頻率做為播放速度。可為 Frequency 或 Xi，支援 4K ~ 44.1K。

- NY4 / NY5 使用 Ri / Xi 指定播放速度，Ri / Xi 與播放速度對應表可參考 NY4 / NY5 頻率計算公式。
- NY5+ / NY6 / NY7 則是使用 Ri / Xi 的值做為播放速度，R0 = 0xA，則播放速度即為 10k。
- NX1 不支援指定播放速度。

Storage：播放的檔案所在的儲存空間，若不指定則為內部空間。

- NY4 / NY5 / NY5+ / NY6 / NY7 不支援指定儲存空間。
- NX1 支援 SPI0 / SPI1 / UDR。

注意：NY4 / NY5 中，當使用 Voicename{*n}指令時，系統會按照其之前一個 Voice 的採樣率來播放 Voice，其自身沒有預設的採樣率，所以，如果程式中的 Voice 使用了不同的採樣率，可能引起同一首 Voicename{*n}多次執行時使用不同的採樣率來播放。

例. 指定欲播放的檔案

```
P1: PlayV(Ch1,$V1,6k)/PlayVS(Ch1,$V1,6k) ; 使用 Voice Label 指定欲播放的檔案
P2: PlayV(ch0, X1:X0, 10k), PB.3=1       ; 播放被暫存器內容值所指到的音檔（若 X1=0x01，X0=0xF2，則播放編號 V498 音檔），於語音通道 0，播放頻率 10k，結束後執行 PB.3=1。
P3: PlayV(ch0, R3:R2:R1:R0, 10k), PB.3=1 ; 播放被暫存器內容值所指到的音檔（若 R3=0x0，R2=0x3，R1=0XF，R0=0x1，則播放編號 V1009 音檔），於語音通道 0，播放頻率 10k，結束後執行 PB.3=1。
P4: PlayV(ch0, X1:X0++,10k), PB.3=1      ; 播放被暫存器內容值所指到的音檔（若 X1=0x01，X0=0xF2，則播放編號 V498 音檔，X0 自動加 1，X0=0xF3），於語音通道 0，播放頻率 10k，結束後執行 PB.3=1
P5: PlayVS(ch0, X1:X0, 10k), PB.3=1      ; 播放被暫存器內容值所指到的音檔（若 X1=0x01，X0=0xF2，則播放編號 V498 音檔），於語音通道 0，播放頻率 10k，播放同時執行 PB.3=1。
P6: PlayVS(ch0, R3:R2:R1:R0, 10k), PB.3=1; 播放被暫存器內容值所指到的音檔（若 R3=0x0，R2=0x3，R1=0XF，R0=0x1，則播放編號 V1009 音檔），於語音通道 0，播放頻率 10k，播放同時執行 PB.3=1。
P7: PlayVS(ch0, X1:X0++,10k), PB.3=1     ; 播放被暫存器內容值所指到的音檔（若 X1=0x01，X0=0xF2，則播放編號 V498 音檔，X0 自動加 1，X0=0xF3），於語音通道 0，播放頻率 10k，播放同時執行 PB.3=1。
```

例. 指定播放速度

```
P1: PlayV(Ch1, $V0,6.35k) / PlayVS(Ch1, $V0,6.35k)   ; 直接設定播放速度
P2: PlayV(Ch1, $V0) / PlayVS(Ch1, $V1)               ; 不寫播放速度使用該檔案的採樣頻率為播放速度
P3: PlayV(Ch1, $V0, R1:R0)                           ; 使用 2 個 Ri 指定播放速度
P4: PlayV(Ch1, $V0,X0) / PlayVS(Ch1,$V0,X0)          ; 使用 Xi 指定播放速度
```

例. 重複播放

```
P1: PlayV(Ch1, $V0) *3                               ; 重覆播放 3 次
```

例. 播放語音同時執行背景路徑

```
PlayV(Ch1, $V0)&[BG1,BG2] / PlayVS(Ch1, $V1)&[BG1,BG2]  ; 播放語音同時執行 BG1 與 BG2
```

PlayV(Ch1, $V0)&[BG1,X] / PlayVS(Ch1, $V0)&[BG1,X] ; 播放語音同時執行 BG1，Background2 保持不變。

PlayV(Ch1, $V0)&[OFF,X] / PlayVS(Ch1, $V0)&[OFF,X] ; 播放語音並關閉 Background1，Background2 保持不變。

例. 根據 [QIO] 設定播放語音並輸出 QIO 訊號。.

```
[Voice File]
V1 = Qio.nyq                                    ; 加入 voice 檔案。
[Output State]
Output_0: Q    Q    Q    Q    Q    Q    Q    Q    ; 使用 PA 及 PB 輸出 QIO 訊號。
[QIO]
MQIO = [PA.0:Q9, PA.1:Q10, PA.2:Q11, PA.3:Q12, PB.0:Q13, PB.1:Q14, PB.2:Q15, PB.3:Q16]
                                                ; PA 及 PB 輸出 Q9 ~ Q16。
[Path]
Path1: Output_0, PlayV(Ch0, $V1)                ; 播放 V1 並根據 MQIO 設定輸出 QIO 訊號。
```

例. 播放 SPI Flash 上的語音。

```
[Variable]
Var8: R0
[Memory]
SPI0_File = Spi.spiprj
[Path]
PowerOn: R0 = 0
Path1: PlayV(Ch0, R0++, SPI0)                   ; 播放 SPI Flash 上由 R0 所指定的語音。
```

5.5.1.1 PlayV / PlayVS with Table

使用讀表的方式來播放 Voice。

例.

```
[Table]
Tab:
{
[0x001, 0x003, 0x005, 0x007, 0x009],
[0x004, 0x005, 0x3F6, 0x008, 0x010],
[0x000, 0x001, 0x002, 0x003, 0x0F4]
}
[Path]
TR1: TableL(Tab,1,1,R0), PlayV(ch0, R0), PB.3=1
; 將 Table 指定位置值存放至 R0，播放 R0 指定音檔（R0=0x5，播放編號 V5 音檔），於語音通道 0，結束後執行 PB.3=1。
```

TR2: TableL(Tab,1,1,X0), PlayV(ch0, X0), PB.3=1

；將 Table 指定位置值存放至 X0，播放 X0 指定音檔（X0=0x05，播放編號 V5 音檔），於語音通道 0，結束後執行 PB.3=1。

TR3: TableL(Tab,1,1,X0), PlayV(ch0, $V0, X0), PB.3=1

；將 Table 指定位置值存放至 X0，播放編號 V0 音檔，於語音通道 0，播放頻率為 X0（X0=0x05，播放頻率為 5K），結束後執行 PB.3=1。

TR4: TableL(Tab,1,1,R0), PlayVS(ch0, R0), PB.3=1

；將 Table 指定位置值存放至 R0，播放 R0 指定音檔（R0=0x5，播放編號 V5 音檔），於語音通道 0，播放同時執行 PB.3=1。

TR5: TableL(Tab,1,1,X0), PlayVS(ch0, X0), PB.3=1

；將 Table 指定位置值存放至 X0，播放 X0 指定音檔（X0=0x05，播放編號 V5 音檔），於語音通道 0，播放同時執行 PB.3=1。

TR6: TableL(Tab,1,1,X0), PlayVS(ch0, $V0, X0), PB.3=1

；將 Table 指定位置值存放至 X0，播放編號 V0 音檔，於語音通道 0，播放頻率為 X0（X0=0x05，播放頻率為 5K），播放同時執行 PB.3=1。

5.5.2 WaitVN(Ch)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

指定通道的 Voice 正在播放中，則 Voice 播放完畢後，再執行下一個指令。沒有 Voice 播放則立即執行下一個指令。

Ch：選擇等待播放結束的語音通道，不帶此參數，則表示等待全部語音通道的 Voice 播放結束。

- NY4 不能指定通道。
- NY5 / NY5+支援的語音通道為 Ch0 ~ Ch3。
- NY6 支援的語音通道為 Ch0 ~ Ch5。
- NY7 支援的語音通道為 Ch0 ~ Ch7。
- NX1 支援的語音通道為 Ch0 ~ Ch3。

注意：PlayV=PlayVS+WaitVN。

例. NY7 使用 WaitVN 指令

[Path]

TR1: PlayV(CH1,$V0)

TR2: PlayVS(CH1, $V0), WaitVN(1)　　；與 TR1 的執行結果相同。

5.5.3 PauseV(Ch)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

用來暫停指定通道的 Voice 檔案播放。

Ch：指定要暫停播放的語音通道。不指定，則暫停全部語音通道的播放。

- NY4 不能指定通道。

- NY5 / NY5+支援的語音通道為 Ch0 ~ Ch3。
- NY6 支援的語音通道為 Ch0 ~ Ch5。
- NY7 支援的語音通道為 Ch0 ~ Ch7。
- NX1 支援的語音通道為 Ch0 ~ Ch3。

例.

```
PowerOn: InputState
TR1: PlayV(Ch0,$V0), PlayV(Ch1,$V1), PlayV(Ch2,$V2), PlayV(Ch3, $V3)
TR2: PauseV                    ; 暫停所有的 Voice 播放。
TR3: ResumeV                   ; 恢復所有的 Voice 播放。
TR4: PauseV(0)                 ; 暫停語音通道 0 的 Voice 播放。
TR5: ResumeV(0)                ; 恢復語音通道 0 的 Voice 播放。
TR6: PauseV(1)                 ; 暫停語音通道 1 的 Voice 播放。
TR7: ResumeV(1)                ; 恢復語音通道 1 的 Voice 播放。
TR8: PauseV(2)                 ; 暫停語音通道 2 的 Voice 播放。
TR9: ResumeV(2)                ; 恢復語音通道 2 的 Voice 播放。
TR10: PauseV(3)                ; 暫停語音通道 3 的 Voice 播放。
TR11: ResumeV(3)               ; 恢復語音通道 3 的 Voice 播放。
```

5.5.4 ResumeV(Ch)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

用來恢復指定通道的 Voice 檔案播放。

Ch：指定要恢復語音播放的通道。不指定，則恢復全部語音通道的播放。

- NY4 不能指定通道。
- NY5 / NY5+支援的語音通道為 Ch0 ~ Ch3。
- NY6 支援的語音通道為 Ch0 ~ Ch5。
- NY7 支援的語音通道為 Ch0 ~ Ch7。
- NX1 支援的語音通道為 Ch0 ~ Ch3。

以下情況指令不再有效：

1. Voice 檔案已經播放結束。
2. 有執行過 StopV 或 Stop 指令。

例.

```
PowerOn: InputState
TR1: PlayV(ch0,$V0), PlayV(ch1,$V1), PlayV(ch2,$V2), PlayV(ch3, $V3)
TR2: PauseV                    ; 暫停所有的 Voice 播放。
TR3: ResumeV                   ; 恢復所有的 Voice 播放。
TR4: PauseV(0)                 ; 暫停語音通道 0 的 Voice 播放。
TR5: ResumeV(0)                ; 恢復語音通道 0 的 Voice 播放。
```

```
TR6: PauseV(1)                    ; 暫停語音通道 1 的 Voice 播放。
TR7: ResumeV(1)                   ; 恢復語音通道 1 的 Voice 播放。
TR8: PauseV(2)                    ; 暫停語音通道 2 的 Voice 播放。
TR9: ResumeV(2)                   ; 恢復語音通道 2 的 Voice 播放。
TR10: PauseV(3)                   ; 暫停語音通道 3 的 Voice 播放。
TR11: ResumeV(3)                  ; 恢復語音通道 3 的 Voice 播放。
```

5.5.5 StopV(Ch)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

用來停止指定通道的 Voice 檔案播放。

Ch：指定要停止播放的語音通道。省略不寫，則停止全部語音通道的播放。

- NY4 不能指定通道。
- NY5 / NY5+支援的語音通道為 Ch0 ~ Ch3。
- NY6 支援的語音通道為 Ch0 ~ Ch5。
- NY7 支援的語音通道為 Ch0 ~ Ch7。
- NX1 支援的語音通道為 Ch0 ~ Ch3。

例.

```
PowerOn: InputState
TR1: PlayV(ch0, $V0), PlayV(ch1, $V1), PlayV(ch2, $V2), PlayV(ch3, $V3)
TR3: StopV(0)                     ; 停止通道 0 的 Voice 的播放。
TR4: StopV(1)                     ; 停止通道 1 的 Voice 的播放。
TR5: StopV(2)                     ; 停止通道 2 的 Voice 的播放。
TR6: StopV(3)                     ; 停止通道 3 的 Voice 的播放。
TR7: StopV                        ; 停止所有通道的 Voice 的播放。
```

5.5.6 FreqCH = nK

[NY4 / NY5]

只有用 Voice Label 的代號或檔案名來直接播放語音時才能使用該指令，而不能用於 PlayV 的指令下。要使用該指令，需要在 play 前先下達一次 Freq=nK 播放速度的指令，之後的 play 動作會延用上次指定的播放速度。

CH：指定播放的語音通道。

- NY4 不能指定播放通道。
- NY5 支援的語音通道為 Ch0 ~ Ch3。

n：播放速度，n=4K~44.1K（頻率對照表請參考頻率計算公式）。

例. **Freq=10k, $V0**

5.5.7 V_Chx_Freq = nK

[NY5+]

修改正在播放語音的通道 S.R.。設定的 S.R.數值範圍為 4K~44.1K。此設定不會影響下一次播放語音的 S.R.。

Chx：指定播放的語音通道。

- NY5+支援的語音通道為 Ch0~Ch3。

注意：當指定的語音通道沒有在播放語音，此指令無效。

例. **V_Ch0_Freq = 10K** ；**將語音通道 0 的 S.R.設為 10K。**

5.5.8 V_Chx_Vol = n / V_Chx_Vol = Xi

[NY5+ / NY6 / NY7]

修改正在播放語音的通道音量。設定的音量數值範圍為 0 ~ 255，可為立即值或用 Xi 來控制；設為 0 時表示該語音通道為靜音，設為 255 時表示將該語音通道音量維持在最大音量。

Chx：指定播放的語音通道。

- NY5+支援的語音通道為 Ch0 ~ Ch3。
- NY6 支援的語音通道為 Ch0 ~ Ch5。
- NY7 支援的語音通道為 Ch0 ~ Ch7。

注意：當指定的語音通道沒有在播放語音，此指令無效。

例. **V_Ch0_Vol = 200** ；**將語音通道 0 的音量設為 200，約原本音量的 78%。**

例. **V_Ch1_Vol = X0** ；**取 X0 內的數值當成通道 1 的音量。**

5.5.9 Xi = V_Chx_Vol

[NY5+ / NY6 / NY7]

讀取正在播放語音的通道音量。

Chx：指定播放的語音通道。

- NY5+支援的語音通道為 Ch0 ~ Ch3。
- NY6 支援的語音通道為 Ch0 ~ Ch5。
- NY7 支援的語音通道為 Ch0 ~ Ch7。

例. **X0 = V_Ch0_Vol** ；**讀取語音通道 0 的音量，並將結果存放 X0。**

5.5.10 SBC_Loop_On

[NX1]

SBC 循環播放。

注意：

1. 當使用循環播放功能時，SBC-1 和 SBC-2 分別僅提供單一通道可循環播放。

2. 自 Q-Code 8.20 起，此指令將不再繼續維護，建議以 Audio_Loop_On 指令替代。

例.

```
PowerOn: InputState
TR1: PlayV(ch0, $V0)                    ; SBC 音檔播放。
TR2: SBC_Loop_On                        ; 開啟循環播放。
TR3: SBC_Loop_Off                       ; 停止循環播放。
```

5.5.11 SBC_Loop_Off

[NX1]

停止 SBC 循環播放。停止循環播放後，仍會把音檔完整播放完畢。

Note：自 Q-Code 8.20 起，此指令將不再繼續維護，建議以 Audio_Loop_Off 指令替代。

5.5.12 ADPCM_Loop_On

[NX1]

ADPCM 循環播放。

注意：

1. 當使用循環播放功能時，僅提供單一通道可循環播放。

2. 自 Q-Code 8.20 起，此指令將不再繼續維護，建議以 Audio_Loop_On 指令替代。

例.

```
PowerOn: InputState
TR1: PlayV(ch0, $V0)                    ; ADPCM 音檔播放。
TR2: ADPCM_Loop_On                      ; 開啟循環播放。
TR3: ADPCM_Loop_Off                     ; 停止循環播放。
```

5.5.13 ADPCM_Loop_Off

[NX1]

停止 ADPCM 循環播放。停止循環播放後，仍會把音檔完整播放完畢。

Note：自 Q-Code 8.20 起，此指令將不再繼續維護，建議以 Audio_Loop_Off 指令替代。

5.5.14 ADPCM_UpSampling

[NX1]

ADPCM 設定升採樣。

ADPCM_UpSampling(Factor)

Factor：升採樣倍數，可設定 1~8。

注意：

1. *當使用升採樣功能時，僅提供單一通道可升採樣。*
2. *升採樣後不可大於 64 KHz。*

例.

```
PowerOn: ADPCM_UpSampling(2)          ; ADPCM 使用 2 倍升採樣。
TR1: PlayV(ch0, $V0)                  ; ADPCM 音檔播放。
```

5.5.15 ADM_Loop_On

[NX1]

ADM 循環播放。

注意：

1. *當使用循環播放功能時，僅提供單一通道可循環播放。*
2. *自 Q-Code 8.20 起，此指令將不再繼續維護，建議以 Audio_Loop_On 指令替代。*

例.

```
PowerOn: InputState
TR1: PlayV(Ch0, $V0)                  ; ADM 音檔播放。
TR2: ADM_Loop_On                      ; 開啟循環播放。
TR3: ADM_Loop_Off                     ; 停止循環播放。
```

5.5.16 ADM_Loop_Off

[NX1]

停止 ADM 循環播放。停止循環播放後，仍會把音檔完整播放完畢。

Note：自 Q-Code 8.20 起，此指令將不再繼續維護，建議以 Audio_Loop_Off 指令替代。

5.5.17 ADM_UpSampling

[NX1]

ADM 設定升採樣。

ADM_UpSampling(Factor)

Factor：升採樣倍數，可設定 1~8。

注意：

1. *當使用升採樣功能時，僅提供單一通道可升採樣。*
2. *升採樣後不可大於 64 KHz。*

例.

```
PowerOn: ADM_UpSampling(2)            ; ADM 使用 2 倍升採樣。
TR1: PlayV(Ch0, $V0)                  ; ADM 音檔播放。
```

5.5.18 PCM_Loop_On

[NX1]

PCM 循環播放。

注意：

1. 當使用循環播放功能時，僅提供單一通道可循環播放。

2. 自 Q-Code 8.20 起，此指令將不再繼續維護，建議以 Audio_Loop_On 指令替代。

例.

```
PowerOn: InputState
TR1: PlayV(ch0, $V0)                    ; PCM 音檔播放。
TR2: PCM_Loop_On                        ; 開啟循環播放。
TR3: PCM_Loop_Off                       ; 停止循環播放。
```

5.5.19 PCM_Loop_Off

[NX1]

停止 PCM 循環播放。停止循環播放後，仍會把音檔完整播放完畢。

Note：自 Q-Code 8.20 起，此指令將不再繼續維護，建議以 Audio_Loop_Off 指令替代。

5.5.20 ReadFileCountV

[NX1]

取得 Voice 播放清單的檔案數量。

ReadFileCountV(Ri)

ReadFileCountV(Ri, Storage)

Ri：用於取得數量的變數，取得的資料為 0~65535。

Storage：用於指定特定儲存空間的檔案數量，可使用 UDR / SPI0 / SPI1，不指定時則為 **[Voice File]** 內的數量。

例.

```
[Variable]
Var16: voice_max=0, voice_idx=0
[Path]
PowerOn: ReadFileCountV(voice_max)          ; 取得 Voice File 數量。
TR1:     voice_idx++,                       ; voice_idx+1。
         voice_idx>=voice_max?{voice_idx=0}, ; index 超過邊界，重置為 0。
         PlayV(ch0, voice_idx)              ; 播放 voice_idx 曲目。
```

5.6 錄音指令（Record Command）

Record Command				
Record	RecordS	WaitRN	StopR	EraseR
EraseRS	WaitEN	-	-	-

5.6.1 Record / RecordS

[NX1]

進行錄音。錄音完成後，可以透過 PlayV 播放。

指令格式如下：

Record(Label {,Time})

RecordS (Label {, Time})

Record：待錄音結束後，執行下一個指令。

RecordS：開始錄音後，立即執行下一個指令。

Label：**[Record]** 中定義的錄音區段名稱。

Time：錄音的長度，未定則為該錄音區段的長度。

例.

```
[Path]
TR1: Record($Rec0)                    ; 進行錄音，使用 Rec0 區段。
TR2: PlayV(ch0, $Rec0)                ; 使用 PlayV 播放錄音結果
```

5.6.2 WaitRN

[NX1]

若錄音正在進行中，則在錄音完畢後，再執行下一個指令。沒有在錄音則立即執行下一個指令。

注意：Record=RecordS+WaitRN。

例.

```
[Path]
TR1: Record($Rec0)
TR2: RecordS($Rec0), WaitRN           ; 與 TR1 的執行結果相同。
```

5.6.3 StopR

[NX1]

用來停止錄音。

例.

```
PowerOn: InputState
TR1R: Record($Rec0)
```

```
TR2F: StopR                                    ; 停止當前的錄音。
```

5.6.4 EraseR / EraseRS

[NX1]

對 SPI Flash 上的錄音區段進行擦除動作。

指令格式如下：

EraseR(Label)

EraseRS (Label)

EraseR：待擦除結束後，執行下一個指令。

EraseRS：開始擦除後，立即執行下一個指令。

Label：**[Record]** 中定義的錄音區段名稱。

例.

```
EraseR($Rec0)
```

5.6.5 WaitEN

[NX1]

若 SPI Flash 擦除正在進行中，則在擦除完畢後，再執行下一個指令。沒有在擦除則立即執行下一個指令。

注意：

1. ***EraseR=EraseRS+WaitEN。***
2. ***開啟即擦即錄功能時，此指令不可使用。***

例.

```
[Path]
TR1: EraseR($Rec0)
TR2: EraseRS($Rec0), WaitEN                    ; 與 TR1 的執行結果相同。
```

5.7 句子指令（Sentence Command）

Sentence Command				
PlayS	WaitSN(n)	PauseS(n)	ResumeS(n)	StopS(n)

5.7.1 PlayS(Parameter)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

是 *Q-Code* 提供使用者簡易的指令來達成播放組合句子的方式。指令格式如下：

PlayS(Parameter)

PlayS：待組合句子播放結束後，執行下一個指令。

Parameter：設定欲播放的組合句子代號 *例.* **PlayS($S1)**

注意：NY5+ / NY6 / NY7 / NX1 中。句子功能主要用來將多個資源檔案（Melody、Speech 及 Action 等）組合成一個句子使用，因此 Sentence 段落中只允許 Play 及 Delay 等指令，其餘指令不可使用在 Sentence 段落內。

例.

```
[Sentence]
S1: PlayV(CH2,$V0), DELAY(0.1), PlayV(CH2,$V1), DELAY(0.2), PlayV(CH2,$V2)
[Path]
PowerOn: KEY1
TR1: PlayS($S1)
[Background1]
BG1: PlayS($S3),PlayS($S4)
[Background2]
BG3: PlayS($S3),PlayS($S4)
```

5.7.2 WaitSN(n)

[NY6 / NY7 / NX1]

若有句子正在播放，則句子播放完畢後，再執行下一個指令。沒有句子播放中則立即執行下一個指令。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，若有句子播放中，則等待句子播放完畢後返回，執行下一個指令。

例. NY7 使用 WaitSN 指令

```
[Path]
TR1: PlayS($S1)
TR2: WaitSN(0), PB.0=1                    ; 待句子播放完畢，將 PB.0 設為 1。
```

5.7.3 PauseS(n)

[NY6 / NY7 / NX1]

可以暫停指定的句子播放。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，則暫停所有的句子播放。

例.

```
PowerOn: InputState
TR1: PlayS($S1)
TR2: PauseS                               ; 暫停所有的句子播放。
TR3: ResumeS                              ; 恢復所有的句子播放。
TR4: PauseS(0)                            ; 暫停前景的句子播放。
```

```
TR5: ResumeS(0)                    ; 恢復前景的句子播放。
```

5.7.4 ResumeS(n)

[NY6 / NY7 / NX1]

可以恢復指定的句子播放。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，則恢復所有的句子播放。

以下情況指令不再有效：

1. 句子已經播放結束。
2. 有執行過 StopS 或 Stop 指令。

例.

```
PowerOn: InputState
TR1: PlayS($S1)
TR2: PauseS                        ; 暫停所有的句子播放。
TR3: ResumeS                       ; 恢復所有的句子播放。
TR4: PauseS(0)                     ; 暫停前景的句子播放。
TR5: ResumeS(0)                    ; 恢復前景的句子播放。
```

5.7.5 StopS(n)

[NY6 / NY7 / NX1]

用來停止句子播放。可以用 n 參數指定要停止的前景或背景。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。若 n 不指定，則停止所有的句子播放。

例.

```
PowerOn: InputState
TR1: PlayS($S1)
TR2: StopS                         ; 停止所有的句子播放。
TR3: StopS(0)                      ; 停止前景的句子播放。
```

5.8 SPI Play 指令（SPIPlay Command）

SPIPlay Command				
SPIPlay	SPIPlayS	SPIWaitN	SPIStop	SPIPause
SPIResume	SPIVol = n	SPIVol = Ri	Ri = SPIVol	-
SPIGetIndex(Result, SPIGroup)		-		

注意：在對 SPI Flash 下任何指令前，請確定 SPI Flash 在穩定可讀寫狀態。依 SPI Flash 型號的不同，上電後至穩定可讀寫的時間也不同，約 200us ~ 15ms 不等。PCB 板上的電容也會影響上電後 SPI Flash 到達可工作電壓的時間，以及下電後放電時間。因此，若 SPI Flash 下電後重新上電應檢查

SPI Flash 狀態或適當地延時後再作讀寫。

5.8.1 SPIPlay / SPIPlayS

[NY6B / NY6C / NY7 / NX1]

Q-Code 提供使用者利用簡易的指令來達成播放存在於 SPI Flash 的檔案，支援 2 組 SPI 介面的設定，但這 2 組 SPI 介面無法同時播放。指令格式如下：

SPIPlay ({Ch, } Index {++}{, SPIGroup})

SPIPlayS ({Ch, } Index {++}{, SPIGroup})

SPIPlay ({Ch, } Index {++}, Px.n)

SPIPlayS ({Ch, } Index {++}, Px.n)

SPIPlay：待 SPI 檔案播放結束後，執行下一個指令。

SPIPlayS：SPI 檔案開始播放，立即執行下一個指令。

Ch：指定要使用的輸出通道 (***NX1 Only***)。

Index：播放檔案的索引值，可為立即值或變數。

- 使用立即值索引
- 使用 Ri/Xi 索引，可使用 1 ~ 4 個 Ri 或是 1 ~ 2 個 Xi 的組合 (NY6/NY7) 或是單一變數 (NX1)。

SPIGroup：選擇播放的 SPI 介面。

- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

Px.n：當 SPI 檔案為 action 時輸出的腳位。

NY7 的 SPIPlay 功能所提供的語音轉檔資料格式和所支援的採樣率如下：

Sample Rate	PCM8	PCM12	ADPCM6
7.8K	✓	✓	✓
15.6K	✓	✓	✓

注意：

1. *語音檔需使用 SPI_Encoder 轉成.bin 檔後，再燒錄到 SPI Flash 才可透過 Q-Code 播放。*
2. *Encode 後的檔案，其 Header 會包含檔案型態（句子或語音），使用者只需要給予 Index 即可播放。*
3. *播放的 Index 如超過 SPI Flash 內的檔案數，則會改播放 Index 0 的檔案。*
4. *NY6 / NY7 中，SPI 檔案無法跟語音(PlayV/PlayVS)或 melody(PlayM/PlayMS)同時播放，兩者會互相打斷。*
5. *NY6 / NY7 使用 SPIPlay 進行 action 播放時，固定使用 Ch1。*
6. *使用 NX1 時，有指定 Px.n 參數時，表示用於 action 播放，Ch 參數表示 action 通道，可以使用 Ch1 ~ Ch32，若選擇播放的檔案不為 action 則此指令無作用。未指定 Px.n 參數時，表示用於 Voice / MIDI 播放，Ch 參數表示聲音輸出通道，僅可使用 Ch0 ~ Ch7，此時若選擇播放的檔案不為 Voice / MIDI 時則此指令無作用。*

7. NY6 同時播放 SPI flash 中的 voice 與 action 時，若語音壓縮率為 1 ~ 9 且採樣率超過 16kHz，或是語音壓縮率為 10 ~ PCM 且採樣率超過 10kHz，則 action 不會輸出。

例. NY7 指定欲播放的檔案索引值。

```
SPIPlay(1, SPI1) / SPIPlayS(2, SPI1)          ; 使用立即值索引
SPIPlay(R0, SPI1) / SPIPlayS(R2:R1:R0, SPI1)  ; 使用 Ri 索引
SPIPlay(X0, SPI1) / SPIPlayS(X1:X0, SPI1)     ; 使用 Xi 索引
```

例. NY7 指定播放的 SPI 介面。

```
SPIPlay(1, SPI1) / SPIPlay(1, SPI2)           ; 指定 SPI 介面
SPIPlay(R1:R0)                                ; 不指定 SPI 介面，使用 SPI1
```

例. NY6 播放 SPI Flash 中的 action。

```
SPIPlay(0, PA.3)                              ; 播放 Index 0，將 action 輸出至 PA.3。
```

例. NX1 播放 SPI Flash 中的 action。

```
SPIPlay(Ch1, 0, PA.3, SPI0)                   ; 播放 Index 0，使用 Ch1 將 action 輸出至 PA.3。
```

例.

```
P1: SPIPlay(5), PB.3=1                        ; 播放 Index 5，結束後執行 PB.3=1。
P2: SPIPlayS(5), PB.3=1                       ; 播放 Index 5 後，馬上執行 PB.3=1。
P3: SPIPlay(R1:R0, SPI1)                      ; 使用 R1:R0 為索引值來播放檔案，R1 為高四位元，
                                              R0 為低四位元。
P4: SPIPlay(X0++)                             ; 使用 X0 為索引值來播放檔案，播放後 X0 自動加 1。
```

5.8.2 SPIWaitN(Ch)

[NY6B / NY6C / NY7 / NX1]

等待 SPIPlay 播放完畢後，再執行下一個指令，沒有 SPIPlay 正在執行則立即執行下一個指令。

Ch : 選擇等待播放結束的通道。不帶此參數，則表示全部通道的播放結束。此參數作用於聲音輸出通道，而非 action 通道。有指定此參數時，僅作用於播放中的語音。未指定此參數則作用於所有經由 SPIPlay 播放的媒體。

- NY6 / NY7 不支援指定通道。
- NX1 支援 Ch0 ~ Ch7。

5.8.3 SPIStop(Ch)

[NY6B / NY6C / NY7 / NX1]

用來停止 SPI 播放。

Ch : 選擇停止播放的通道。若省略不寫則相當於全部停止播放。此參數作用於聲音輸出通道，而非 action 通道。有指定此參數時，僅作用於播放中的語音。未指定此參數則作用於所有經由 SPIPlay 播放的媒體。

- NY6 / NY7 不支援指定通道。

- NX1 支援 Ch0 ~ Ch7。

5.8.4 SPIPause(Ch)

[NY6B / NY6C / NY7 / NX1]

用來暫停 SPI 播放。

Ch : 選擇暫停播放的通道。若省略不寫則全部暫停播放。此參數作用於聲音輸出通道，而非 action 通道。有指定此參數時，僅作用於播放中的語音。未指定此參數則作用於所有經由 SPIPlay 播放的媒體。

- NY6 / NY7 不支援指定通道。
- NX1 支援 Ch0 ~ Ch7。

注意：如果在沒有 SPIPlay 的情況下，此動作會被忽略。

5.8.5 SPIResume(Ch)

[NY6B / NY6C / NY7 / NX1]

用來恢復 SPI 播放。此指令不帶參數。

Ch : 選擇恢復播放的通道。若省略不寫則全部恢復播放。此參數作用於聲音輸出通道，而非 action 通道。有指定此參數時，僅作用於播放中的語音。未指定此參數則作用於所有經由 SPIPlay 播放的媒體。

- NY6 / NY7 不支援指定通道。
- NX1 支援 Ch0 ~ Ch7。

5.8.6 SPIVol = n / SPIVol = Ri

[NY6B / NY6C / NY7]

設定 SPIPlay 播放的語音音量。

n : 音量，可為立即值或用 Ri 來控制。

- NY6 數值範圍為 0~15；n=0 時表示為靜音，n=15 時表示將音量維持在最大音量。
- NY7 數值範圍為 0~4；n=0 時表示為靜音，n=4 時表示將音量維持在最大音量。

```
例. SPIVol = 3          ; 將 SPI 語音音量設為 3。
例. SPIVol = R0         ; 取 R0 內的數值當成 SPI 語音的音量。
```

5.8.7 Ri = SPIVol

[NY6B / NY6C / NY7]

讀取 SPIPlay 播放的語音音量。

```
例. R0 = SPIVol         ; 讀取 SPI 語音音量，並存於 R0。
```

5.8.8 SPIGetIndex

[NY6B / NY6C / NY7 / NX1]

讀取 SPI Flash 內的檔案索引數目。

SPIGetIndex(Result{, SPIGroup})

Result：檔案索引數目

- NY6 / NY7 支援 RI:Rk:Rj:Ri 或 Xi:Xj 的組合，其中 Ri=bit0 ~ 3，Rj=bit4 ~ 7，Rk=bit8 ~ 11，RI=bit12 ~ 15 / Xi=bit0 ~ 7，Xj=bit8 ~ 15。
- NX1 僅支援一個變數。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

例. **SPIGetIndex(R1:R0)** ；讀取第一組 SPI 的檔案索引值數目。
例. **SPIGetIndex(R2:R1:R0, SPI2)** ；讀取第二組 SPI 的檔案索引值數目。
例. **SPIGetIndex(X0)** ；讀取第一組 SPI 的檔案索引值數目。
例. **SPIGetIndex(X1:X0, SPI2)** ；讀取第二組 SPI 的檔案索引值數目。

5.9 SPI Flash 指令（SPI Flash Command）

SPI Flash Command				
SPI_WREN	SPI_WRDIS	SPI_RDID	SPI_CE	SPI_SE
SPI_BE	SPI_DP	SPI_RDP	SPI_WRSR	SPI_RDSR
SPI_WRD	SPI_RDD	SPI_GetAddr	-	-

注意：

1. *在對 SPI Flash 下任何指令前，請確定 SPI Flash 在穩定可讀寫狀態。依 SPI Flash 型號的不同，上電後至穩定可讀寫的時間也不同，約 200us ~ 15ms 不等。PCB 板上的電容也會影響上電後 SPI Flash 到達可工作電壓的時間，以及下電後放電時間。因此，若 SPI Flash 下電後重新上電應檢查 SPI Flash 狀態或適當地延時後再作讀寫。*
2. *不同的 SPI Flash 型號對於 Qual Mode 的設定不同，需留意 QE bit 是否已適當地設置。*

5.9.1 SPI_WREN

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Write Enable 指令，將 status register 的 WEL 位元（bit1）設為 1，使 SPI Flash 處於可寫入狀態。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. *此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0x06。*
2. *指令執行後，status register 的 WEL 位元（bit1）會被設為 1，此時 SPI Flash 的寫入動作才有效，用者可透過 SPI_RDSR 指令確認位元狀態。*

例.

```
SPI_WREN                        ; 傳送 Write Enable 指令。
```

例. NY7 / NX1

```
SPI_WREN(SPI1)                  ; 透過 SPI1 傳送 Write Enable 指令。
```

例. NY7

```
SPI_WREN(SPI2)                  ; 透過 SPI2 傳送 Write Enable 指令。
```

5.9.2 SPI_WRDIS

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Write Disable 指令，將 status register 的 WEL 位元（bit1）清為 0，使 SPI Flash 處於不可寫入的狀態。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. *此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0x04。*
2. *指令執行後，status register 的 WEL 位元（bit1）會被清為 0，此時 SPI Flash 的寫入動作無效，使用者可透過 SPI_RDSR 指令確認位元狀態。*

例.

```
SPI_WRDIS                       ; 傳送 Write Disable 指令。
```

例. NY7 / NX1

```
SPI_WRDIS(SPI1)                 ; 透過 SPI1 傳送 Write Disable 指令。
```

例. NY7

```
SPI_WRDIS(SPI2)                 ; 透過 SPI2 傳送 Write Disable 指令。
```

5.9.3 SPI_RDID(Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Read Identification 指令，讀取 SPI Flash 的 Manufacturer ID、Memory Type 及 Capacity 等資訊。

Result：欲存放讀回資訊的變數，可為 Ri 或 Xi 的組合，需要 24-bit；高 8 位元為 Manufacturer ID，中 8 位元 Memory Type，低 8 位元為 Capacity。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0x9F。

例. NY6B / NY6C / NY7

[Path]

TR1: SPI_RDID(R5:R4:R3:R2:R1:R0)

; 讀取 SPI Flash 資訊，將 Manufacturer ID 存在[R5:R4]，Memory Type 存在[R3:R2]，Capacity 存在[R1:R0]。

例. NY7

[Path]

TR1: SPI_RDID(X2:X1:X0, SPI2)

; 透過 SPI2 讀取 SPI Flash 資訊，將 Manufacturer ID 存在 X2，Memory Type 存在 X1，Capacity 存在 X0。

例. NX1

[Variable]

Var32: ID

[Path]

TR1: SPI_RDID(ID, SPI1)

; 透過 SPI1 讀取 SPI Flash 資訊，將 Manufacturer ID 存在 ID[8:11]，Memory Type 存在 ID[4:7]，Capacity 存在 ID[0:3]。

5.9.4 SPI_CE

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Chip Erase 指令，執行全記憶體抹除動作。

抹除動作執行時，系統不會進入睡眠模式，待抹除動作完成後，會自動執行"SPI_EraseEnd"路徑。使用者亦可利用 SPI_RDSR 指令讀取 status register 的 WIP 位元（bit0）狀態，當抹除進行中，此位元為 1，抹除完成，此位元為 0。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。

- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. *此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0xC7。*
2. *此指令會自動將 WEL 位元設為 1，使用者不需額外執行 SPI_WREN 指令。*
3. *根據不同大小的 SPI Flash，抹除動作可能需耗時數秒至數十秒，此期間除 SPI_RDSR 指令外，其餘指令無效。*
4. *抹除動作完成後，會自動執行 "SPI_EraseEnd" 路徑，使用者可透過此路徑知道抹除動作完成時間點，以便執行後續動作。*

例. NY6B / NY6C / NY7 / NX1

[Path]

TR1: SPI_CE

; 對 SPI Flash 傳送 Chip Erase 指令，執行全記憶體抹除動作。

例. NY7

[Path]

TR1: SPI_CE(SPI2)

; 透過 SPI2 對 SPI Flash 傳送 Chip Erase 指令，執行全記憶體抹除動作。

例. NY6B / NY6C / NY7 / NX1

[Path]

Erase: SPI_CE, R0=1

SPI_EraseEnd: R0=0

; 對 SPI Flash 傳送 Chip Erase 指令，抹除時將 R0 設為 1；抹除完成後，將 R0 設為 0。

5.9.5 SPI_SE(Addr, Count, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Sector Erase 指令，執行 sector 抹除動作。一個 sector 的大小為 4KB，使用者可指定一次要抹除的 sector 數量。

抹除動作執行時，系統不會進入睡眠模式，待抹除動作完成後，會自動執行"SPI_EraseEnd"路徑。使用者亦可利用 SPI_RDSR 指令讀取 status register 的 WIP 位元（bit0）狀態，當抹除進行中，此位元為 1，抹除完成，此位元為 0。

Addr：指定要抹除的位址，可為立即值或由變數指定；一個 sector 大小為 4KB，使用立即值定址時，需填入 24-bit 位址，但其中 Bit[11:0]為無效位元；若使用變數定址時，則僅需填入有效位元 Bit[23:12]。

Count：要進行抹除的 sector 數量，範圍 1 ~ 16，不填則預設為 1。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. *此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0x20。*
2. *此指令會自動將 WEL 位元設為 1，使用者不需額外執行 SPI_WREN 指令。*
3. *根據抹除的 sector 數量，抹除動作可能需耗時數十 ms 到數秒，此期間除 SPI_RDSR 指令外，其餘指令無效。*
4. *抹除動作完成後，會自動執行 "SPI_EraseEnd" 路徑，使用者可透過此路徑知道抹除動作完成時間點，以便執行後續動作。*

例. NY6B / NY6C / NY7

[Path]

TR1: SPI_SE(0x4321, 3)

; 將 SPI Flash 的 0x4000 ~ 0x6FFF 位址資料抹除。

TR2: R0=0x2, R1=0x1, SPI_SE(R1:R0, 2)

; 將 SPI Flash 的 0x12000 ~ 0x13FFF 位址資料抹除。

例. NY7

[Path]

Erase: X0=0x23, R2=0x1, SPI_SE(R2:X0, SPI1), R3=1

SPI_EraseEnd: R3=0

; 透過 SPI1 將 SPI Flash 的 0x123000 ~ 0x123FFF 位址資料抹除，抹除時將 R3 設為 1；抹除完成後，將 R3 設為 0。

例. NX1

[Variable]

Var32: Addr

Var8: R3

[Path]

TR1: SPI_SE(0x4321, 3)

; 透過 SPI0 將 SPI Flash 的 0x4000 ~ 0x6FFF 位址資料抹除。

TR2: Addr = 0x12, SPI_SE(Addr, 2, SPI1)

; 透過 SPI1 將 SPI Flash 的 0x12000 ~ 0x13FFF 位址資料抹除。

Erase: Addr = 0x123, SPI_SE(Addr, SPI0), R3=1

SPI_EraseEnd: R3=0

; 透過 SPI0 將 SPI Flash 的 0x123000 ~ 0x123FFF 位址資料抹除，抹除時將 R3 設為 1；抹除完成後，將 R3 設為 0。

5.9.6 SPI_BE(Addr, Count, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Block Erase 指令，執行 block 抹除動作。一個 block 的大小為 64KB，使用者可指定一次要抹除的 block 數量。

抹除動作執行時，系統不會進入睡眠模式，待抹除動作完成後，會自動執行"SPI_EraseEnd"路徑。使用者亦可利用 SPI_RDSR 指令讀取 status register 的 WIP 位元（bit0）狀態，當抹除進行中，此位元為 1，抹除完成，此位元為 0。

Addr：指定要抹除的位址，可為立即值或由變數指定；一個 block 大小為 64KB，使用立即值定址時，需填入 24-bit 位址，但其中的 Bit[15:0]為無效位元；若使用變數定址時，則僅需填入有效位元 Bit[23:16]。

Count：要進行抹除的 block 數量，範圍 1 ~ 16，不填則預設為 1。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. ***此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0xD8。***
2. ***此指令會自動將 WEL 位元設為 1，使用者不需額外執行 SPI_WREN 指令。***
3. ***根據抹除的 block 數量，抹除動作可能需耗時數秒，此期間除 SPI_RDSR 指令外，其餘指令無效。***
4. ***抹除動作完成後，會自動執行 "SPI_EraseEnd"路徑，使用者可透過此路徑知道抹除動作完成時間點，以便執行後續動作。***

例. NY6B / NY6C / NY7 / NX1

[Path]

TR1: SPI_BE(0x14321, 2)

; 將 SPI Flash 的 0x10000 ~ 0x2FFFF 位址資料抹除。

TR2: R0=0x3, SPI_BE(R0, 3)

; 將 SPI Flash 的 0x30000 ~ 0x5FFFF 位址資料抹除。

例. NY7

[Path]

Erase: X0=0x17, SPI_BE(X0, SPI1), R3=1

SPI_EraseEnd: R3=0

; 透過 SPI1 將 SPI Flash 的 0x170000 ~ 0x17FFFF 位址資料抹除，抹除時將 R3 設為 1；抹除完成後，將 R3 設為 0。

例. NX1

[Variable]

Var32: Addr

[Path]

TR1: SPI_BE(0x14321, 2)

; 透過 SPI0 將 SPI Flash 的 0x10000 ~ 0x2FFFF 位址資料抹除。

```
TR2: Addr=0x3, SPI_BE(Addr, 3, SPI1)
; 透過 SPI1 將 SPI Flash 的 0x30000 ~ 0x5FFFF 位址資料抹除。
Erase: Addr=0x17, SPI_BE(Addr, SPI1), R3=1
SPI_EraseEnd: R3=0
; 透過 SPI1 將 SPI Flash 的 0x170000 ~ 0x17FFFF 位址資料抹除，抹除時將 R3 設為 1；抹除完成後，將 R3 設為 0。
```

5.9.7 SPI_DP(SPIGroup)

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Deep Power-down 指令，使之進入 deep power-down 模式，**建議系統進入睡眠模式前，執行此指令，以節省 SPI Flash 耗電。**

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. 此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0xB9。

2. SPI Flash 進入 deep power-down 後，除 SPI_RDP 喚醒指令外，其餘指令無效。

例. NY6B / NY6C / NY7 / NX1

```
SPI_DP                          ; 傳送 Deep Power-down 指令。
```

例. NY7 / NX1

```
SPI_DP(SPI1)                    ; 透過 SPI1 傳送 Deep Power-down 指令。
```

例. NY7

```
SPI_DP(SPI2)                    ; 透過 SPI2 傳送 Deep Power-down 指令。
```

5.9.8 SPI_RDP(Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Release Deep Power-down 指令，將其從 deep power-down 模式中喚醒，並讀回 8-bit 的 Device ID。

Result：儲存讀回的 Device ID，可以為 Ri 或是 Xi 的組合，不填則表示不讀取 ID。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0xAB。

例. NY7

[Path]

TR1: SPI_RDP(R1:R0)

; 透過 SPI1 將 SPI Flash 從 deep power-down 模式喚醒，並將 Device ID 儲存在[R1:R0]。

TR2: SPI_RDP(SPI1)

; 透過 SPI1 將 SPI Flash 從 deep power-down 模式喚醒，不讀取 Device ID。

TR3: SPI_RDP(X0, SPI2)

; 透過 SPI2 將 SPI Flash 從 deep power-down 模式喚醒，並將 Device ID 儲存在 X0。

例. NX1

[Variable]

Var8: DevID

[Path]

TR1: SPI_RDP(DevID)

; 透過 SPI0 將 SPI Flash 從 deep power-down 模式喚醒，並將 Device ID 儲存在 DevID。

TR2: SPI_RDP(SPI0)

; 透過 SPI0 將 SPI Flash 從 deep power-down 模式喚醒，不讀取 Device ID。

TR3: SPI_RDP(DevID, SPI1)

; 透過 SPI1 將 SPI Flash 從 deep power-down 模式喚醒，並將 Device ID 儲存在 DevID。

5.9.9 SPI_WRSR(Data, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Write Status Register 指令，用來修改 Status Register 的資料。

Data：欲寫入 Status Register 的資料，可為立即值，或 Ri / Xi 的組合。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. 此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0x01。

2. 此指令會自動將 WEL 位元設為 1，使用者不需額外執行 SPI_WREN 指令。

例. NY7

[Path]

TR1: SPI_WRSR(0x18)

; 透過 SPI1 寫入 0x18 到 Status Register。

TR2: R0=0x8, R1=0x1, SPI_WRSR(R1:R0, SPI1)

; 透過 SPI1 寫入 0x18 到 Status Register。

TR3: X0=0x18, SPI_WRSR(X0, SPI2)

; 透過 SPI2 寫入 0x18 到 Status Register。

例. NX1

[Variable]

Var8: Data

[Path]

TR1: SPI_WRSR(0x18)

; 透過 SPI0 寫入 0x18 到 Status Register。

TR2: Data = 0x18, SPI_WRSR(Data, SPI0)

; 透過 SPI0 寫入 0x18 到 Status Register。

TR3: Data=0x18, SPI_WRSR(Data, SPI1)

; 透過 SPI1 寫入 0x18 到 Status Register。

5.9.10 SPI_RDSR(Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Read Status Register 指令，用來讀取 Status Register 的資料。

Result：儲存讀回的 Status Register 資料，可為 Ri / Xi 的組合。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0x05。

例. NY7

[Path]

TR1: SPI_RDSR(R1:R0)

; 透過 SPI1 讀取 Status Register 資料，並儲存到[R1:R0]。

TR2: SPI_RDSR(X0, SPI1)

; 透過 SPI1 讀取 Status Register 資料，並儲存到 X0。

TR3: SPI_RDSR(X0, SPI2)

; 透過 SPI2 讀取 Status Register 資料，並儲存到 X0。

例. NX1

[Variable]

Var8: SR

[Path]

TR1: SPI_RDSR(SR)

; 透過 SPI0 讀取 Status Register 資料，並儲存到 SR。

TR2: SPI_RDSR(SR, SPI0)

; 透過 SPI0 讀取 Status Register 資料，並儲存到 SR。

TR3: SPI_RDSR(SR, SPI1)

; 透過 SPI1 讀取 Status Register 資料，並儲存到 SR。

5.9.11 SPI_WRD(Addr, Data, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Page Program 指令，用來寫入資料到指定位址，使用者可搭配 SPI_TX 指令一次寫入多個位元組資料。當欲寫入資料傳送完畢後，需下達 SPI_CS_Off 指令，以執行寫入動作。寫入動作執行時，使用者需利用 SPI_RDSR 指令讀取 status register 的 WIP 位元(bit0)狀態，當寫入進行中，此位元為 1，寫入完成，此位元為 0。

Addr：要寫入資料的 24-bit 位址，可為立即值或是由變數指定；由變數指定時，高位元的變數可省略，省略的位元預設為 0。

Data：要寫入的資料，可為立即值或是由變數指定。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. ***此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0x02。***
2. ***指令會自動將 WEL 位元設為 1，使用者不需額外執行 SPI_WREN 指令。***
3. ***指令執行時會自動將 CS 腳位設為低準位，當欲寫入資料傳送完畢後，需下達 SPI_CS_Off 指令，以執行寫入動作。***
4. ***當 SPI Flash 執行寫入時，status register 的 WIP 位元(bit0)會被設為 1，使用者可透過 SPI_RDSR 指令讀取位元狀態，當狀態為 0，則表示寫入完成。***
5. ***Page Program 指令一次最多能寫入 256 個位元組資料，每傳送一位元組資料，寫入位址會自動加 1，但僅針對低 8 位元，假設當前寫入位址為 0x1FF，加 1 後會變 0x100。***
6. ***根據寫入的資料數，可能需耗時數百 ms，此期間除 SPI_RDSR 指令外，其餘指令無效。***

例. NY7

[Path]

TR1: SPI_WRD(0x4321, 0x32), SPI_CS_Off

; 透過 SPI1 將 0x32 寫入到 0x4321 位址。

TR2: X0=0xFF, R2=0x2, R3=0x1, X2=0x34, SPI_WRD(R3:R2:X0, X2, SPI2),

SPI_TX(0x56, SPI2), SPI_TX(0x78, SPI2), SPI_CS_Off(SPI2)

; 透過 SPI2 將 0x34 寫入到 0x12FF 位址，0x56 寫入到 0x1200 位址，0x78 寫入到 0x1201 位址。

TR3: SPI_WRD(0x4321, 0xAB, SPI1), R0=0xD, R1=0xC, SPI_TX(R1:R0, SPI1),

X0=0xEF, SPI_TX(X0, SPI1), SPI_CS_Off(SPI1)

; 透過 SPI1 將 0xAB 寫入到 0x4321 位址，0xCD 寫入到 0x4322 位址，0xEF 寫入到 0x4323 位址。

例. NX1

[Variable]

Var32: Addr

Var8: Data

[Path]

TR1: SPI_WRD(0x4321, 0x32), SPI_CS_Off

; 透過 SPI0 將 0x32 寫入到 0x4321 位址。

TR2: Addr = 0x12FF, Data=0x34, SPI_WRD(Addr, Data, SPI1),

SPI_TX(0x56, SPI1), SPI_TX(0x78, SPI1), SPI_CS_Off(SPI1)

; 透過 SPI1 將 0x34 寫入到 0x12FF 位址，0x56 寫入到 0x1200 位址，0x78 寫入到 0x1201 位址。

TR3: SPI_WRD(0x4321, 0xAB), Data = 0xCD, SPI_TX(Data),

Data=0xEF, SPI_TX(Data), SPI_CS_Off

; 透過 SPI0 將 0xAB 寫入到 0x4321 位址，0xCD 寫入到 0x4322 位址，0xEF 寫入到 0x4323 位址。

5.9.12 SPI_RDD(Addr, Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

對 SPI Flash 傳送 Read Data 指令，從指定位址讀取資料，使用者可搭配 SPI_RX 指令一次讀取多個位元組資料。資料讀取完畢後，需下達 SPI_CS_Off 指令，以結束讀取動作。

Addr：指定要讀取資料的 24-bit 位址，可為立即值或由變數指定；由變數指定時，高位元的變數可省略，省略的位元預設為 0。

Result：欲存放讀取資料的變數。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. ***此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0x03。***
2. ***指令執行時會自動將 CS 腳位設為低準位，當資料讀取完畢後，需下達 SPI_CS_Off 指令，以結束讀取動作。***
3. ***Read 指令每讀取一位元組資料，讀取位址會自動加 1。***

例. NY7

[Path]

TR1: SPI_RDD(0x4321, R1:R0), SPI_CS_Off

; 透過 SPI1 將 0x4321 位址內資料儲存到[R1:R0]。

TR2: R0=0x3, X1=0x12, SPI_RDD(X1:R0, X2, SPI2), SPI_CS_Off(SPI2)

; 透過 SPI2 將 0x123 位址內資料儲存到 X2。

TR3: SPI_RDD(0x1234, X0, SPI1), SPI_RX(R3:R2, SPI1), SPI_RX(X2, SPI1), SPI_CS_Off(SPI1)

; 透過 SPI1 將 0x1234 位址內資料儲存到 X0，0x1235 位址內資料儲存到[R3:R2]，0x1236 位址內資料儲存到 X2。

例. NX1

[Variable]

Var32: Addr

Var8: Data, Data0, Data1, Data2

[Path]

TR1: SPI_RDD(0x4321, Data), SPI_CS_Off

; 透過 SPI0 將 0x4321 位址內資料儲存到 Data。

TR2: Addr=0x123, SPI_RDD(Addr, Data, SPI1), SPI_CS_Off(SPI1)

; 透過 SPI1 將 0x123 位址內資料儲存到 Data。

TR3: SPI_RDD(0x1234, Data0, SPI1), SPI_RX(Data1, SPI1) , SPI_RX(Data2 , SPI1) , SPI_CS_Off(SPI1)

; 透過 SPI1 將 0x1234 位址內資料儲存到 Data0，0x1235 位址內資料儲存到 Data1，0x1236 位址內資料儲存到 Data2。

5.9.13 SPI_GetAddr(Index, Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

使用者可透過 SPI_Encoder 的 User Defined Sections 頁面加入自定義資訊檔或保留區塊，再利用此指令取得各區塊的位址，以執行讀寫或抹除動作。

當使用者加入檔案或定義保留區時，加入的順序(Sec 欄位)即為索引值，將欲取得區塊位址的索引值帶入指令內參數，即可獲得對應的位址；SPI_Encoder 詳細的操作方式請參考 SPI_Encoder 使用手冊。

Index：指定要取得區塊位址的索引值，可為立即值或由變數指定，共 16-bit，有效範圍為 0 ~ 65535；由變數指定時，高位元的變數可省略，省略的位元預設為 0。

Result：指定儲存區塊開始位址的變數，共 24-bit，高位元的變數可省略不儲存。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：此指令相容於九齊 N25Q 系列 SPI Flash，指令碼為 0x03。

例. NY7

[Path]

TR1: SPI_GetAddr(0x8, X2:X1:R1:R0)

; 透過 SPI1 讀取索引值 8 的區塊位址，並將位址的 Bit[23:16]存在 X2，Bit[15:8]存在 X1，Bit[7:4]存在 R1，Bit[3:0]存在 R0。

TR2: R0=0x3, R1=0x2, SPI_GetAddr(R1:R0, X2:X1, SPI2)

; 透過 SPI2 讀取索引值 0x23 的區塊位址，並將位址的 Bit[15:8]存在 X2，Bit[7:0]存在 X1。

TR3: X0=0x12, SPI_GetAddr(X0, R7:R6:R5:R4:R3:R2, SPI1)

; 透過 SPI1 讀取索引值 0x12 的區塊位址，並將位址存在[R7:R2]。

例. NX1

[Variable]

Var32: Addr

Var16: Index

[Path]

TR1: SPI_GetAddr(0x8, Addr)

; 透過 SPI0 讀取索引值 8 的區塊位址，並將位址存在 Addr。

TR2: Index=0x23, SPI_GetAddr(Index, Addr, SPI1)

; 透過 SPI1 讀取索引值 0x23 的區塊位址，並將位址存在 Addr。

5.10 SPI 指令（SPI Command）

SPI Command				
SPI_CS_On	SPI_CS_Off	SPI_TX	SPI_RX	SPI_CLKDIV
SPI_CPOL	SPI_CPHA	-	-	-

5.10.1 SPI_CS_On(SpiGroup)

[NY6B / NY6C / NY7 / NX1]

將 CS 腳位設為低準位。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：當任一 CS 腳位設為低準位時，系統將無法進入睡眠模式。

例. NY7

```
SPI_CS_On                    ; 將 SPI1 的 CS 腳位設為低準位。
SPI_ CS_On(SPI1)             ; 將 SPI1 的 CS 腳位設為低準位。
SPI_ CS_On(SPI2)             ; 將 SPI2 的 CS 腳位設為低準位。
```

5.10.2 SPI_CS_Off(SpiGroup)

[NY6B / NY6C / NY7 / NX1]

將 CS 腳位設為高準位。

SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

例. NY7

```
SPI_CS_Off                    ; 將 SPI1 的 CS 腳位設為高準位。
SPI_ CS_Off(SPI1)             ; 將 SPI1 的 CS 腳位設為高準位。
SPI_ CS_Off(SPI2)             ; 將 SPI2 的 CS 腳位設為高準位。
```

5.10.3 SPI_TX(Data, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

透過 SCK 及 MOSI 腳位傳送 8-bit 資料到 SPI Flash；此指令可以用來傳送指令或搭配 SPI_WRD 指令寫入資料等，亦可用此指令組合出 *Q-Code* 不支援的指令。

Data：欲傳送的 8-bit 資料，可為立即值或是由變數指定。
SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. ***此指令僅透過 SCK 及 MOSI 腳位送出 8-bit 資料，不影響 CS 腳位狀態，因此使用者需自行控制 CS 腳準位。***
2. ***此指令需搭配 SPI_WREN 及 SPI_WRD 指令才能將資料寫入到 SPI Flash 內，寫入方式請參考 SPI_WRD 指令說明。***

例. NY7

[Path]

```
TR1: SPI_CS_On, SPI_TX(0xB9), SPI_CS_Off
; 透過 SPI1 傳送 0xB9 指令到 SPI Flash。
TR2: R0=0xB, R1=0xA, SPI_CS_On(SPI2), SPI_TX(R1:R0, SPI2), SPI_CS_Off(SPI2)
; 透過 SPI2 傳送 0xAB 指令到 SPI Flash。
TR3: X0=0x06, SPI_CS_On(SPI1), SPI_TX(X0, SPI1), SPI_CS_Off(SPI1)
; 透過 SPI1 傳送 0x06 指令到 SPI Flash。
```

例. NX1

[Variable]

Var8: Data

[Path]

TR1: SPI_CS_On, SPI_TX(0xB9), SPI_CS_Off

; 透過 SPI0 傳送 0xB9 指令到 SPI Flash。

TR2: Data=0xAB, SPI_CS_On(SPI1), SPI_TX(Data, SPI1), SPI_CS_Off(SPI1)

; 透過 SPI1 傳送 0xAB 指令到 SPI Flash。

TR3: Data=0x06, SPI_CS_On(SPI0), SPI_TX(Data, SPI0), SPI_CS_Off(SPI0)

; 透過 SPI0 傳送 0x06 指令到 SPI Flash。

5.10.4 SPI_RX(Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

透過 SCK 及 MISO 腳位從 SPI Flash 接收 8-bit 資料；此指令可用來讀取 ID 或搭配 SPI_RDD 指令接收 SPI Flash 內資料等，亦可用此指令組合出 Q-Code 不支援的指令。

Result：讀取的 8-bit 資料，可為 Ri / Xi 的組合。
SPIGroup：選擇讀取的 SPI 介面。

- NY6 不支援指定 SPIGroup。
- NY7 可為 SPI1 或 SPI2，不填則預設為 SPI1。
- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

注意：

1. *此指令僅透過 SCK 及 MISO 腳位接收 8-bit 資料，不影響 CS 腳位狀態，因此使用者需自行控制 CS 腳準位。*
2. *此指令需搭配 SPI_RDD 指令才能讀取 SPI Flash 內資料，讀取方式請參考 SPI_RDD 指令說明。*

例. NY7

[Path]

TR1: SPI_RDD(0x0, X0), SPI_RX(R5:R4), SPI_CS_Off

; 透過 SPI1 讀取 SPI Flash 的 0x0~0x1 位址內資料，並將 0x0 位址內資料存在 X0，0x1 位址內資料存在[R5:R4]。

TR2: SPI_RDD(0x105, X0, SPI2), SPI_RX(X1, SPI2), SPI_CS_Off(SPI2)

; 透過 SPI2 讀取 SPI Flash 的 0x105~0x106 位址內資料，並將 0x105 位址內資料存在 X0，0x106 位址內資料存在 X1。

例. NX1

[Variable]

Var8: Data0, Data1

[Path]

TR1: SPI_RDD(0x0, Data0), SPI_RX(Data1), SPI_CS_Off

; 透過 SPI0 讀取 SPI Flash 的 0x0~0x1 位址內資料，並將 0x0 位址內資料存在 Data0，0x1 位址內資料存在 Data1。

TR2: SPI_RDD(0x105, Data0, SPI1), SPI_RX(Data1, SPI1), SPI_CS_Off(SPI1)

；透過 SPI1 讀取 SPI Flash 的 0x105~0x106 位址內資料，並將 0x105 位址內資料存在 Data0，0x106 位址內資料存在 Data1。

5.10.5 SPI_CLKDIV(Divisor, SPIGroup)

[NX1]

調整 SCK 透過 IHRC 除頻的倍率。

Divisor：除頻倍率，可設定 1 / 2 / 4 / 8 / 16 / 32 / 64 / 128，當 IHRC>32MHz 時預設值為 2，其餘預設值為 1。

SPIGroup：選擇讀取的 SPI 介面。

- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

例.

```
SPI_CLKDIV(16, SPI1)                    ；將 SPI1 的 SCK 的頻率設定為 IHRC 的 1/16。
```

5.10.6 SPI_CPOL(Polarity, SPIGroup)

[NX1]

調整 SCK 閒置時的極性。

Polarity：0 (Low)、1 (High)，預設值為 0。

SPIGroup：選擇讀取的 SPI 介面。

- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

例.

```
SPI_CPOL(0, SPI1)                       ；將 SPI1 的 CLK 閒置極性設為 Low。
```

5.10.7 SPI_CPHA(Phase, SPIGroup)

[NX1]

調整 SCK 的資料採樣時機。

Phase：0 (1st Edge)、1 (2nd Edge)，預設值為 0。

SPIGroup：選擇讀取的 SPI 介面。

- NX1 可為 SPI0 或 SPI1，不填則預設為 SPI0。

例.

```
SPI_CPHA(0, SPI1)                       ；將 SPI1 的 CLK 採樣設置為 First Edge。
```

5.11 Embedded Flash 指令（Embedded Flash Command）

Embedded Flash Command				
EF_SE	EF_WRD	EF_RDD	EF_GetAddr	-

5.11.1 EF_SE

[NX1 EF]

對 Embedded Flash 傳送 Sector Erase 指令，執行 sector 抹除動作。一個 sector 的大小為 512B，使用者可指定一次要抹除的 sector 數量。

抹除動作執行時，系統不會進入睡眠模式。

EF_SE(Addr)

EF_SE(Addr, Count)

Addr：指定要抹除的位址，可為立即值或由變數指定；一個 sector 大小為 512B，使用立即值定址時，需填入 20-bit 位址，但其中 Bit[8:0]為無效位元；若使用變數定址時，則僅需填入有效位元 Bit[20:9]。

Count：要進行抹除的 sector 數量，範圍 1 ~ 16，不填則預設為 1。

注意：根據抹除的 sector 數量，抹除動作可能需耗時數十 ms 到數秒，此期間其餘指令無效。

例.

[Variable]

Var32: Addr

[Path]

TR1: EF_SE(0x4321, 3)

; 將 Embedded Flash 的 0x4200 ~ 0x47FF 位址資料抹除。

TR2: Addr=0x12, EF_SE(Addr, 2)

; 將 Embedded Flash 的 0x2400 ~ 0x27FF 位址資料抹除。

5.11.2 EF_WRD

[NX1 EF]

對 Embedded Flash 傳送 Program 指令，用來寫入資料到指定位址。

EF_WRD(Addr, Data)

Addr：要寫入資料的 20-bit 位址，可為立即值或是由變數指定；由變數指定時，高位元的變數可省略，省略的位元預設為 0。

Data：要寫入的資料，可為立即值或是由變數指定。

注意：根據寫入的資料數，可能需耗時數百 ms，此期間其餘指令無效。

例.

[Variable]

Var32: Addr

Var8: Data

[Path]

TR1: EF_WRD(0x4321, 0x32)

; 將 0x32 寫入到 0x4321 位址。

TR2: Addr=0x12FF, Data=0x34, EF_WRD(Addr, Data)

; 將 0x34 寫入到 0x12FF 位址。

5.11.3 EF_RDD

[NX1 EF]

對 Embedded Flash 傳送 Read Data 指令，從指定位址讀取資料。

EF_RDD(Addr, Result)

Addr：指定要讀取資料的 20-bit 位址，可為立即值或由變數指定；由變數指定時，高位元的變數可省略，省略的位元預設為 0。

Result：欲存放讀取資料的變數。

例.

[Variable]

Var32: Addr

Var8: Data

[Path]

TR1: EF_RDD(0x4321, Data)

; 將 0x4321 位址內資料儲存到 Data。

TR2: Addr=0x123, EF_RDD(Addr, Data)

; 將 0x123 位址內資料儲存到 Data。

5.11.4 EF_GetAddr

[NX1 EF]

使用者可透過 Q-Code 的 User Defined Sections 頁面加入自定義資訊檔或保留區塊，再利用此指令取得各區塊的位址，以執行讀寫或抹除動作。

當使用者加入檔案或定義保留區時，加入的順序(Sec 欄位)即為索引值，將欲取得區塊位址的索引值帶入指令內參數，即可獲得對應的位址。

EF_GetAddr(Index, Result)

Index：指定要取得區塊位址的索引值，可為立即值或由變數指定，共 16-bit，有效範圍為 0 ~ 65535；由變數指定時，高位元的變數可省略，省略的位元預設為 0。

Result：指定儲存區塊開始位址的變數，共 20-bit，高位元的變數可省略不儲存。

例.

[Variable]

Var32: Index, Addr

[Path]

TR1: EF_GetAddr(0x8, Addr)

; 讀取索引值 8 的區塊位址，並將位址存在 Addr。

TR2: Index=0x23, EF_GetAddr(Index, Addr)

; 讀取索引值 0x23 的區塊位址，並將位址存在 Addr。

5.12 Storage 指令（Storage Command）

Storage Command				
Storage_Save	-	-	-	-

5.12.1 Storage_Save

[NX1]

將 **[Storage Variable]** 中所定義的變數儲存至 SPI Flash / Embedded Fash 中。

5.13 比較器指令（Comparator Command）

Comparator Command				
CMP_ON	CMP_ON(Source)	CMP_OFF	CMP_Read(Ri:Rj)	CMP_Read(Xi)
CMP_CNT_ON(Count)		CMP_CNT_OFF	-	-

5.13.1 CMP_ON / CMP_ON(Source)

[NY6B / NY6C]

用來啟動 Comparator 功能；若帶有計時源頻率，則可以將兩次比較成立的時間擷取下來，即 Capture 功能，計時源頻率分別有 62.5K、250K、1M、4M。

使用者開啟 Comparator 或 Capture 功能，當比較器成立發生中斷時，將會立即執行相對應的系統路徑。

注意：

1. 第一次啟動 Capture 時，Comparator 成立擷取的時間，會是錯誤的。

2. Capture 占用到 Timer 資源，無法再使用 Timer 計時功能。

例. 開啟 Comparator 功能，成立時執行 INT_CMP 系統路徑。

```
TR1:CMP_ON                    ; 開啟 Comparator 功能。
INT_CMP:!PB                   ; Comparator 成立時，立即將 PB 輸出反向。
```

例. 開啟 Capture 功能，Comparator 成立時執行 INT_CMP 系統路徑，若 Comparator 持續沒有成立，則執行 INT_TMR 系統路徑。

```
TR1:CMP_ON(1M)                  ; 開啟 Capture 功能。
INT_CMP:CMP_Read(X0)            ; Comparator 成立時，立即讀取擷取的時間。
INT_TMR:!PB                     ; Comparator 持續不成立時，立即將 PB 輸出反向。
```

5.13.2 CMP_OFF

[NY6B / NY6C]

關閉 Comparator 功能。

例.
```
TR2:CMP_OFF                     ; 關閉 Comparator 功能。
```

5.13.3 CMP_Read(Rj:Ri) / CMP_Read(Xi)

[NY6B / NY6C]

讀取 Capture 的計數值。

例. 將 Capture 的計數值存放到 X0。
```
INT_CMP: CMP_Read(X0)           ; 讀取 Capture 的計時值，並存放到 X0 暫存器。
```

例. 將 Capture 的計數值存放到 R0,R1。
```
INT_CMP: CMP_Read(R1:R0)        ; 讀取 Capture 的計時值，並存放到 R0、R1 暫存器。
```

5.13.4 CMP_CNT_ON(Count)

[NY6B / NY6C]

用來啟動外部輸入計數功能，需帶入計數次數，當次數達到將執行 INT_TMR 系統路徑。

注意：Counter 占用到 Timer 資源，無法再使用 Timer 功能。

例. 開啟 Counter 功能，設定計數 10 次 PB 輸出反向。
```
TR1: CMP_CNT_ON(10)             ; 開啟 Counter 功能，並設定計數 10 次。
INT_TMR:!PB                     ; 計數達到 10 次，立即將 PB 輸出反向。
```

5.13.5 CMP_CNT_OFF

[NY6B / NY6C]

關閉 Counter 功能。

例.
```
TR2:CMP_CNT_OFF                 ; 關閉 Counter 功能。
```

5.14 計時器指令（Timer Command）

Timer Command				
TMR_On	TMR_OFF	TMR_Read(Rj:Ri)	TMR_Read(Xi)	-

5.14.1 TMR_ON

[NY5+ / NY6 / NX1]

用來啟動 Timer 計時功能，語法如下：

TMR_On(Time)

TMR_On(Timer, Time)

Timer：計時器。

- NY5+支援 TMR0 / TMR1 / TMR2 / TMR3。
- NX1 支援 TMR0 / TMR1 / TMR2 / TMR3 / PWMA / PWMB。

Time：計時時間。

- NY5+支援 64 ~ 256us。
- NY6 支援 1.00ms ~ 4.00ms。
- NX1 OTP 計時時間隨系統頻率而有不同：
  - High Clock Frequency 為 12MHz 時，為 64 ~ 5461us。
  - High Clock Frequency 為 16MHz 時，為 64 ~ 4096us。
  - High Clock Frequency 為 24MHz 時，為 64 ~ 2730us。
  - High Clock Frequency 為 32MHz 時，為 64 ~ 2048us。
- NX1 EF 支援 64 ~ 1365us。

注意：

1. NY6 不支援 Timer 參數。

2. NY6 計時時間位於 1.00ms ~ 2.00ms 之間時，每次計時最小單位為 64us：當位於 2.01ms ~ 4.00ms 每次計時最小單位為 256us。若最小單位時間無法整除所設定的時間時，每次中斷時間會有誤差。

3. NY5+同時只能開啟 1 個計時功能，最後啟動的計時器會將之前的關閉。

例. 開啟 Timer 計時，每 1.5ms 中斷一次，中斷發生時執行 INT_TMR 系統路徑。

```
TR1: TMR_ON(1.5ms)                ; 開啟 Timer 計時功能，每 1.5ms 中斷一次。
INT_TMR:!PA                       ; 每次 Timer 中斷，立即將 PA 輸出反向。
```

5.14.2 TMR_OFF

[NY5+ / NY6 / NX1]

關閉 Timer 功能。語法如下：

TMR_Off

TMR_Off(Timer)

Timer：計時器。

- NX1 支援 TMR0 / TMR1 / TMR2 / TMR3 / PWMA / PWMB。

注意：NY5+ / NY6 不支援 Timer 參數。

例.

```
TR2: TMR_OFF                      ; 關閉 Timer 功能。
```

5.14.3 TMR_Read(Rj:Ri) / TMR_Read(Xi)

[NY6]

讀取 Timer 的計數值。

例. 將 Timer 的計數值存放到 X0。

```
TR1: TMR_Read(X0)                ; 讀取 Timer 的計時值，並存放到 X0 暫存器。
```

例. 將 Timer 的計數值存放到 R0,R1。

```
TR1: TMR_Read(R1:R0)             ; 讀取 Timer 的計時值，並存放到 R0、R1 暫存器。
```

5.15 MIDI 指令（MIDI Command）

MIDI Command				
PlayM	PlayMS	WaitMN	PauseM	ResumeM
StopM	Instrument(Ch, i)	M_Chx_Vol = n	M_Chx_Vol = Ri	Ri = M_Chx_Vol
Tempo + n	Tempo - n	Tempo++	Tempo--	TempoRst
Tempo(Rj:Ri)	Tempo(Xi)	ReadTempo(Rj:Ri)	ReadTempo(Xi)	Mute_On(Ch)
Mute_Off(Ch)	OKON_On	OKON_Off	OKON_Play	OKON_SustainON
OKON_SustainOff	OKON_SustainEnd	DynamicOn	DynamicOff	StopMNote
MIDI_Pitch	Mask_On(Ch)	Mask_Off(Ch)	ReadFileCountM	MIDI_Loop_On
MIDI_Loop_Off	-	-	-	-

5.15.1 PlayM / PlayMS

[NY5 / NY5+ / NY6 / NY7 / NX1]

是 *Q-Code* 提供使用者簡易的指令來達成播放音樂檔的方式。

指令格式如下：

PlayM ({Ch,) Parameter {++} {, Storage})

PlayMS ({Ch, } Parameter {++} {, Storage})

PlayM：待 Melody 檔案播放結束後，執行下一個指令。

PlayMS：Melody 檔案開始播放，立即執行下一個指令。

Ch：所要使用的聲音輸出通道。

- NY5 / NY5+ / NY6 / NY7 不支援指定通道。
- NX1 支援 0 ~ 3 或是 Ch0 ~ Ch3。

Parameter：欲播放的音樂檔。

- Melody file 檔案代號。
- Ri（可支援到 1~4 個 Ri）或是 Xi（可支援到 1~2 個 Xi），由 Ri / Xi 的值決定要播放的是第幾個 melody 檔案。若是 Ri / Xi 帶有"++"則在播放後將其遞增。

Storage：播放的檔案所在的儲存空間，若不指定則為內部空間。

- NY5 / NY5+ / NY6 / NY7 不支援指定儲存空間。
- NX1 支援 SPI0 / SPI1。

例.

```
P1: PlayM($M0),PB.3=1                    ; 播放 Melody 音樂檔，結束後執行 PB.3=1。
P2: PlayM(R3:R2:R1:R0),PB.3=1            ; 播放被暫存器內容值所指到的音樂檔（若 R3=0x0，
                                         R2=0x3，R1=0xF，R0=0x1，則播放編號 M1009 音樂
                                         檔），結束後執行 PB.3=1。
P3: PlayM(X1:X0),PB.3=1                  ; 播放被暫存器內容值所指到的音樂檔（若 X1=0x01，
                                         X0=0xF2，則播放編號 M498 音樂檔），結束後執行
                                         PB.3=1。
P4: PlayM(X1:X0++),PB.3=1                ; 播放被暫存器內容值所指到的音樂檔（若 X1=0x01，
                                         X0=0xF2，則播放編號 M498 音樂檔，且 X0 自動加 1，
                                         X0=0xF3），結束後執行 PB.3=1。
P5: PlayMS($M0),PB.3=1                   ; 播放 Melody，播放同時執行 PB.3=1。
P6: PlayMS(R3:R2:R1:R0),PB.3=1           ; 播放被暫存器內容值所指到的音樂檔（若 R3=0x0，
                                         R2=0x3，R1=0xF，R0=0x1，則播放編號 M1009 音樂
                                         檔），播放同時執行 PB.3=1。
P7: PlayMS(X1:X0),PB.3=1                 ; 播放被暫存器內容值所指到的音樂檔（若 X1=0x01，
                                         X0=0xF2，則播放編號 M498 音樂檔），播放同時執行
                                         PB.3=1。
P8: PlayMS(X1:X0++),PB.3=1               ; 播放被暫存器內容值所指到的音樂檔（若 X1=0x01，
                                         X0=0xF2，則播放編號 M498 音樂檔，且 X0 自動加 1，
                                         X0=0xF3），播放同時執行 PB.3=1。
```

例. 播放 SPI Flash 上的 MIDI。

```
[Variable]
Var8: R0
[Memory]
SPI0_File = Spi.spiprj
[Path]
PowerOn: R0 = 0
Path1: PlayM(Ch0, R0++, SPI0)            ; 播放 SPI Flash 上由 R0 所指定的 MIDI。
```

5.15.1.1 PlayM / PlayMS with Table

使用讀表的方式來播放 Melody。

例.

```
[Table]
Tab:
{
```

[0x001, 0x003, 0x005, 0x007, 0x009],

[0x004, 0x005, 0x3F6, 0x008, 0x010],

[0x000, 0x001, 0x002, 0x003, 0x0F4] }

[Path]

TR1: TableL(Tab,1,1,R0), PlayM(R0), PB.3=1

; 將 Table 指定位置值存放至 R0，播放 R0 指定音樂檔（R0=0x5，播放編號 M5 音樂檔），結束後執行 PB.3=1。

TR2: TableL(Tab,1,1,X0), PlayM(X0), PB.3=1

; 將 Table 指定位置值存放至 X0，播放 X0 指定音樂檔（X0=0x05，播放編號 M5 音樂檔），結束後執行 PB.3=1。

TR4: TableL(Tab,1,1,R0), PlayMS(R0), PB.3=1

; 將 Table 指定位置值存放至 R0，播放 R0 指定音樂檔（R0=0x5，播放編號 M5 音樂檔），播放同時執行 PB.3=1。

TR5: TableL(Tab,1,1,X0), PlayMS(X0), PB.3=1

; 將 Table 指定位置值存放至 X0，播放 X0 指定音樂檔（X0=0x05，播放編號 M5 音樂檔），播放同時執行 PB.3=1。

動作	指令	說明
代號播放	PlayM($M0)	播放 Label “M0” 這個音樂檔。
RAM 播放	PlayM(X0)	播放 X0 內容值所指到音樂檔。
重複播放的次數	PlayM($M0)*n	Label “M0”這個音樂檔會被重複播放 n 次。
在播放聲音檔時一併呼叫背景動作	PlayM($M0)&[BG1,BG2]	在播放 Label “M0” 時會一併呼叫背景 1 與背景 2 的動作。
	PlayM($M0)&[X,BG2]	不改變 BG1 當前的動作。
	PlayM($M0)&[BG1,X]	不改變 BG2 當前的動作。
	PlayM($M0)&[OFF,BG2]	停止 BG1 當前的動作。
	PlayM($M0)&[BG1,OFF]	停止 BG2 當前的動作。

例.

[Melody Database]

D:\QCODE\TEST\ADSR.qmd

; M0 = Music.mid, 4 channel

; M1 = Start.mid, 4 channel

[Path]

TR1: PlayM($M0) **; 播放編號 M0 的 Melody 檔案。**

TR2: PlayM($M0)*3 **; 播放編號 M0 的 Melody 檔案三次。**

TR3: PlayM($M0)&[BG1, BG2] **; 播放編號 M0 檔案，並且同時呼叫背景 1 與背景 2 的動**

作。

[Background1]

BG1: ……

[Background2]

BG2: ……

5.15.2 WaitMN

[NY5 / NY5+ / NY6 / NY7 / NX1]

若有 Melody 正在播放中，則 Melody 播放完畢後，再執行下一個指令。沒有 Melody 播放則立即執行下一個指令。

注意：PlayM=PlayMS+WaitMN。

例.

[Path]

TR1: PlayM($M0)

TR2: PlayMS($M0), WaitMN ; 與 TR1 的執行結果相同。

5.15.3 PauseM

[NY5 / NY5+ / NY6 / NY7 / NX1]

PauseM 暫停當前 MIDI 音樂檔的播放。

例.

PowerOn: PlayM($M0)

TR1: PauseM ; 暫停 Melody 播放。

5.15.4 ResumeM

[NY5 / NY5+ / NY6 / NY7 / NX1]

ResumeM 指令可用來恢復 MIDI 音樂檔案的播放，以下情況指令不再有效：

1. Melody 檔案已經播放結束。
2. 有執行過 StopM 或 Stop 指令。

例.

PowerOn: PlayM($M0)

TR1: PauseM ; 暫停 Melody 播放。

TR2: ResumeM ; 恢復 Melody 播放。

5.15.5 StopM

[NY5 / NY5+ / NY6 / NY7 / NX1]

StopM 指令是立即停止當前播放的 Melody 檔案。

例.

```
PowerOn: PlayM($M0)
TR1: StopM                    ; 停止 Melody 播放。
```

5.15.6 Instrument(Ch, i)

[NY5+ / NY6 / NY7 / NX1]

Instrument 指令是用來改變正在播放的 melody 音色。

Ch：為要改變音色的 MIDI 通道，

- NY5+支援 1 ~ 4、10。
- NY6 支援 1 ~ 6、10。
- NY7 支援 1 ~ 16。
- NX1 支援 1 ~ 16。

i：為 GM 音色編號，可用立即值或用變數控制，範圍為 0 ~ 127。

注意：

1. *此指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。*
2. *使用變數指定 GM 音色時，如果該音色不存在，有可能會造成音色切換錯誤或播放異常的現象。*

```
例. Instrument(Ch2, 1)          ; 將 MIDI 通道 2 的音色強制改為 1 號 GM 音色。
例. Instrument(Ch1, R1:R0)      ; 將 MIDI 通道 1 的音色強制改為 R1:R0 的 GM 音色編
                                  號，R1 為高 3 位元，R0 為低 4 位元。
例. Instrument(Ch3, X0)         ; 將 MIDI 通道 3 的音色強制改為 X0 的 GM 音色編號。
```

5.15.7 M_Chx_Vol = n / M_Chx_Vol = Ri

[NY5+ / NY6 / NY7 / NX1]

修改正在播放 Melody 的個別通道音量。設定的音量可為立即值或用 Ri 來控制。

Chx：指定 MIDI 通道。

- NY5+支援 1 ~ 4、10。
- NY6 支援 1 ~ 6、10。
- NY7 支援 1 ~ 16。
- NX1 支援 1 ~ 16。

n：通道音量。

- NY5+支援 0 ~ 7。
- NY6 支援 0 ~ 7。
- NY7 支援 0 ~ 15。
- NX1 支援 0 ~ 15。

注意：當 melody 開始播放時，會將每個 MIDI 通道的音量預設為 15，因此在播放前使用此指令無效。

例. **M_Ch1_Vol = 10** ；將 MIDI 通道 1 的音量設為 10，約原本音量的 67%。

例. **M_Ch1_Vol = R0** ；取 R0 內的數值當成 MIDI 通道 1 的音量。

5.15.8 Ri = M_Chx_Vol

[NY5+ / NY6 / NY7 / NX1]

讀取正在播放 Melody 的個別通道音量。

Chx：指定 MIDI 通道，

- NY5+支援 1 ~ 4、10。
- NY6 支援 1 ~ 6、10。
- NY7 支援 1 ~ 16。
- NX1 支援 1 ~ 16。

注意：當 melody 開始播放時，會將每個通道的音量預設為 15。

例. **R0 = M_Ch1_Vol** ；讀取 MIDI 通道 1 的音量，並存於 R0。

5.15.9 Tempo + n

[NY5+ / NY6 / NY7 / NX1]

Tempo + n 指令可以增加正在播放的 melody 速度。n 為增加的百分比，以 5%為單位，當增加比例不為 5 的整數倍時會取四捨五入。

注意：此指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。

例. **Tempo=100**

[Path]

PowerOn: PlayMS($M0), Tempo + 10 ；拍子速度增加 10%，變成 110。

5.15.10 Tempo - n

[NY5+ / NY6 / NY7 / NX1]

Tempo - n 指令可以降低正在播放的 melody 速度。n 為降低的百分比，以 5%為單位，當降低比例不為 5 的整數倍時會取四捨五入。

注意：此指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。

例. **Tempo=100**

[Path]

PowerOn: PlayMS($M0), Tempo - 5 ；拍子速度降低 5%，變成 95%。

5.15.11 Tempo++

[NY5 / NY5+ / NY6 / NY7 / NX1]

Tempo++ 指令。依據 Melody 原始的 Tempo 值加快一階。

注意：

1. *NY5 tempo 增量請參考 Tempo=n 指令說明。*
2. *NY5+ / NY6 / NY7 / NX1 每一階加快播放速度 5%，此指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。*

例.

```
PowerOn: PlayMS($M0)
TR1: Tempo++                    ; 拍子速度變成 105%。
```

5.15.12 Tempo--

[NY5 / NY5+ / NY6 / NY7 / NX1]

Tempo-- 指令。依據 Melody 原始的 Tempo 值減慢一階。

注意：

1. *NY5 tempo 增量請參考 Tempo=n 指令說明。*
2. *NY5+ / NY6 / NY7 / NX1 每一階加快播放速度 5%，此指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。*

例.

```
PowerOn: PlayMS($M0)
TR1: Tempo--                    ; 拍子速度變成 95%。
```

5.15.13 TempoRst

[NY5+ / NY6 / NY7 / NX1]

TempoRst 指令。可以恢復 Melody 檔原本的播放速度。

注意：此指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。

例.

```
Tempo=100
PowerOn: PlayMS($M0), Tempo - 5
TR1: TempoRst                   ; 拍子速度恢復成 100。
```

5.15.14 Tempo = n

[NY5]

Tempo=n 指令可以直接控制播放 Melody 的速度。n=34 ~ 500。

Tempo List			
34	51	76	150

Tempo List			
36	53	86	170
38	55	93	200
41	58	100	235
45	64	110	300
47	68	120	500
49	72	135	

例. **Tempo=100**

[Path]

PowerOn: Tempo=100 ; **拍子速度變成 100。**

5.15.15 Tempo(Rj:Ri) / Tempo(Xi)

[NY5]

根據目前 RAM 的值來設定 Tempo 值。

Tempo 值的計算公式如下：

Time base = 2.048ms

Value = 60000(ms) / Tempo / 24 / Timebase

例. **Tempo=100**

PowerOn: X0=12, Tempo(X0) ; **設定拍子速度為 100。**

5.15.16 ReadTempo(Rj:Ri) / ReadTempo(Xi)

[NY5 / NY5+ / NY6 / NY7 / NX1]

將目前的 Tempo 設定值，存放到 RAM。

例.

[Path]

TR1: PlayM($M0) ; **播放 Melody。**

TR2: ReadTempo(X0), [PD,PC]=X0 ; **將 Tempo 的設定值輸出到 PD、PC。**

5.15.17 Mute_On(Ch)

[NY5 / NY5+ / NY6 / NY7 / NX1]

開啟靜音功能，當 melody 播放時，Mute_on 指令可以把指定的 MIDI 通道作靜音。

Ch：要靜音的 MIDI 通道，不指定則為全部 MIDI 通道。

- NY5 支援 0 ~ 3。
- NY5+支援 1 ~ 4、10。
- NY6 支援 1 ~ 6、10。

- NY7 支援 1 ~ 16。
- NX1 支援 1 ~ 16。

注意：

1. NY5+ / NY6 / NY7 使用此指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。

2. NX1 於 Q-Code 6.50 起，Mute_Off 指令將不受 melody 播放限制。

例.

[Path]

PowerOn : PlayM($M0)

TR1: Mute_on(1) **; Melody 播放時，開啟 MIDI 通道 1 為靜音功能。**

TR2: Mute_on(3) **; Melody 播放時，開啟 MIDI 通道 3 為靜音功能。**

5.15.18 Mute_Off(Ch)

[NY5 / NY5+ / NY6 / NY7 / NX1]

關閉靜音功能，當靜音模式被開啟，可以利用 Mute_off 指令把指定的 MIDI 通道還原。

Ch：關閉靜音功能的通道。不指定則為全部 MIDI 通道。

- NY5 支援 0 ~ 3。
- NY5+支援 1 ~ 4、10。
- NY6 支援 1 ~ 6、10。
- NY7 支援 1 ~ 16。
- NX1 支援 1 ~ 16。

注意：NY5+ / NY6 / NY7 使用此指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。

例.

[Path]

PowerOn : PlayM($M0)

TR1: Mute_on(1) **; Melody 播放時，開啟 MIDI 通道 1 為靜音功能。**

TR2: Mute_on(3) **; Melody 播放時，開啟 MIDI 通道 3 為靜音功能。**

TR3: Mute_off(1) **; 取消 MIDI 通道 1 的靜音。**

TR4: Mute_off(3) **; 取消 MIDI 通道 3 的靜音。**

5.15.19 OKON_On / SingleNote_On

[NY5 / NY5+ / NY6 / NY7 / NX1]

One-Key-One-Note 功能開啟，使用者可以使用 OKON_On 指令開啟此功能。

注意：

1. NY5 / NY5+通道固定在 MIDI 通道 0。

2. NY6 / NY7 通道固定在 MIDI 通道 1，指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。

3. NX1 通道固定在 MIDI 通道 1。

例.

[Path]

PowerOn:

TR1: PlayM($M0) ; 播放音樂檔。

TR2: OKON_On ; 開啟 One-Key-One-Note 功能，播放通道 1 的音符。

5.15.20 OKON_Off / SingleNote_Off

[NY5 / NY5+ / NY6 / NY7 / NX1]

One-Key-One-Note 功能關閉，使用者可以使用 OKON_Off 指令關閉此功能。

注意：NY5+ / NY6 / NY7 此指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。

例.

[Path]

PowerOn:

TR1: PlayM($M0) ; 播放音樂檔。

TR2: OKON_On ; 開啟 One-Key-One-Note 功能，播放 MIDI 通道 1 的音符。

TR3: OKON_Off ; 關閉 One-Key-One-Note 功能。

5.15.21 OKON_Play / SinglePlay

[NY5 / NY5+ / NY6 / NY7 / NX1]

使用者可以使用 OKON_Play 指令開始進行 One-Key-One-Note 的播放動作。

注意：

1. NY5 / NY5+通道固定在 MIDI 通道 0。

2. NY6 / NY7 通道固定在 MIDI 通道 1，指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。

3. NX1 通道固定在 MIDI 通道 1。

例.

[Path]

PowerOn:

TR1: PlayM($M0) ; 播放音樂檔。

TR2: OKON_On ; 開啟 One-Key-One-Note 功能，播放通道 1 的音符。

TR3: OKON_Play ; 播放單音。

5.15.22 OKON_SustainOn

[NY5+]

開啟 OKON 的延音功能。當開啟時，OKON_Play 執行時，若 midi 音符有 sustain 段，則該音符持續輸

出不間斷，直到下達 OKON_SustainEnd 指令為止。

例.

```
[Input State]
Input_0: TR1 TR2 TR3 TR4/TR4_R
[Path]
PowerOn: Input_0
TR1: PlayMS($M0), OKON_On          ; 播放音樂檔，並開啟 OKON 功能。
TR2: OKON_SustainOn                ; 開啟 OKON 的延音功能。
TR3: OKON_SustainOff               ; 關閉 OKON 的延音功能。
TR4: OKON_Play                     ; 按下按鍵，開始播放音符，當 OKON 延音開啟時，持續播放
                                   ; sustain 段。
TR4_R: OKON_SustainEnd             ; 放開按鍵，結束音符的播放。
```

5.15.23 OKON_SustainOff

[NY5+]

關閉 OKON 的延音功能。當關閉時，OKON_Play 執行時，若 midi 音符有 sustain 段，則依照 Q-MIDI 中設定的時間長度播放。

5.15.24 OKON_SustainEnd

[NY5+]

結束 OKON 播放中音符的延音效果，若延音效果的時間小於 Q-MIDI 中設定的時間，則依據 Q-MIDI 中設定的時間長度播放，若延音效果的時間大於 Q-MIDI 中設定的時間，則音符直接收音結束。

5.15.25 DynamicOn

[NY5+ / NY6 / NY7]

開啟通道動態分配功能，功能開啟後，播放 melody 的 note 時，程式會依照當時 synthesizer 通道使用狀況，分配一個適合的通道來播放，因此譜曲時，將 note 譜在通道 1 ~ 4、10 (NY5+) / 1 ~ 6、10 (NY6) / 1 ~ 16 (NY7)皆可正常播放。但是仍須注意同時間 note 數不能超過 synthesizer 通道數。程式預設為 DynamicOn。

注意：指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。

例.

```
PowerOn: PlayMS($M0)
TR1: DynamicOn                     ; 開啟動態分配功能。
```

5.15.26 DynamicOff

[NY5+ / NY6 / NY7]

關閉通道動態分配功能，功能關閉後，播放 melody 的 note 時，程式會使用與 note 通道相同的 synthesizer 通道來播放，因此譜曲時，只能將 note 譜在通道 0 ~ 3、7 (NY5+) / 1 ~ 6、10 (NY6) / 1 ~ 8 (NY7)，且同一個通道同時間只能有一個 note，否則前面的 note 會被後面的 note 取代。另外，當 MIDI 通道正在播放語音或單音時，該通道所有的 note 會被忽略，直到語音或單音播放完畢才會繼續播放。

注意：

1. ***指令必須在 melody 播放時呼叫才有效，在 melody 播放前呼叫無效。***
2. ***當動態分配關閉，Percussion 的 note 會被指定到 hardware 通道 3 (NY5+) / 5 (NY6) / 7 (NY7)播放。***

例.

```
PowerOn: PlayMS($M0)
TR1: DynamicOff                    ; 關閉動態分配功能。
```

5.15.27 StopMNote

[NY7]

停止當前正在播放的 MIDI 音符，但不會停止 MIDI 本身的播放，也就是說，當前音符被停止後，後續新的音符仍會正常播放。除此之外，指令也不會影響鍵盤單音音符的播放，即利用 InstNoteOn 及 DrumNoteOn 指令播放的音符。

此指令可搭配 PauseM 指令使用，使 MIDI 暫停播放時，馬上將音符結束，避免暫停時因為尾音過長造成效果不協調。

例.

```
PauseM, StopMNote                  ; 暫停 MIDI 播放，並停止當前正在播放的 MIDI 音符。
```

5.15.28 MIDI_Pitch(Semitone)

[NX1]

此指令用於升降 MIDI 的播放音調。

Semitone：設定範圍由 Q-MIDI 限制，最大±12 個半音，預設值為 0。

例.

```
MIDI_Pitch(-3), PlayMS(Ch0, $M0)   ; 音調降低 3 個半音並播放 MIDI。
```

5.15.29 Mask_On(Ch)

[NX1]

遮蔽 MIDI 通道功能，當 melody 播放時，Mask_On 指令可以把指定的 MIDI 通道遮蔽，使該 MIDI 通道除能以降低 CPU 負載。

Ch：要遮蔽的 MIDI 通道，不指定則為全部 MIDI 通道。

- NX1 支援 1 ~ 16。

例.

```
[Path]
PowerOn : PlayM(Ch0, $M0)
TR1: Mask_on(1)                    ; 開啟 MIDI 通道 1 遮蔽功能。
TR2: Mask_on(3)                    ; 開啟 MIDI 通道 3 遮蔽功能。
```

5.15.30 Mask_Off(Ch)

[NX1]

取消遮蔽 MIDI 通道功能，當 MIDI 通道遮蔽被開啟，可以利用 Mask_Off 指令把指定的 MIDI 通道還原。

Ch：取消遮蔽功能的通道。不指定則為全部 MIDI 通道。

- NX1 支援 1 ~ 16。

例.

```
[Path]
PowerOn : PlayM(Ch0, $M0)
TR1: Mask_on(1)                    ; 開啟 MIDI 通道 1 遮蔽功能。
TR2: Mask_on(3)                    ; 開啟 MIDI 通道 3 遮蔽功能。
TR3: Mask_off(1)                   ; 取消 MIDI 通道 1 的遮蔽功能。
TR4: Mask_off(3)                   ; 取消 MIDI 通道 3 的遮蔽功能。
```

5.15.31 ReadFileCountM

[NX1]

取得 Midi 曲目清單的檔案數量。

ReadFileCountM(Ri)

ReadFileCountM(Ri, Storage)

Ri：用於取得數量的變數，取得的資料為 0~65535。

Storage：指定儲存空間，可使用 SPI0 / SPI1，不指定時則為 **[Melody Database]** 內的曲目數量。

例.

```
[Variable]
Var16: midi_max=0, midi_idx=0
[Path]
PowerOn: ReadFileCountM(midi_max)  ; 取得 midi 曲目數量。
TR1:     midi_idx++,               ; midi_idx+1。
         midi_idx>=midi_max?{midi_idx=0}, ; index 超過邊界，重置為 0。
         PlayM(ch0, midi_idx)      ; 播放 midi_idx 曲目。
```

5.15.32 MIDI_Loop_On

[NX1]

MIDI 循環播放。

Note：自 Q-Code 8.20 起，此指令將不再繼續維護，建議以 Audio_Loop_On 指令替代。

例.

```
PowerOn: InputState
TR1: PlayM(ch0, 0, SPI0)          ; 播放 SPI0 上的第 1 首 MIDI。
TR2: MIDI_Loop_On                 ; 開啟循環播放。
TR3: MIDI_Loop_Off                ; 停止循環播放。
```

5.15.33 MIDI_Loop_Off

[NX1]

停止 MIDI 循環播放。停止循環播放後，仍會把音檔完整播放完畢。

Note：自 Q-Code 8.20 起，此指令將不再繼續維護，建議以 Audio_Loop_Off 指令替代。

5.16 鍵盤指令（Keyboard Command）

Instrument Command				
InstNoteOn	InstNoteOff	InstNoteAllOff	DrumNoteOn	DrumNoteOff
NoteVibrato	Gliss	MaxSingleNote	LongInst_HoldTime	ShortInst_HoldTime
KRecord	KRecordS	WatiKRN	StopKR	PlayK
PlayKS	WaitKN	StopK	-	-

5.16.1 InstNoteOn(Index, Note , Vol) & InstNoteOff(Note)

[NY5+ / NY6 / NY7 / NX1]

InstNoteOn 指令和 InstNoteOff 指令可用作一般樂器的單音鍵盤功能，但兩個指令必須搭配使用。當按下一個鍵時，InstNoteOn 指令會播放一個音符並佔用一個通道，直到釋放按鍵或播放完畢為止。當釋放按鍵時，InstNoteOff 指令將停止播放並釋放佔用的通道。

Index：指定 GM 樂器音色編號，可用立即值或變數指定。

Note：{“C1”, “C#1”, “D1”, “D#1”,”E1”, “F1”, “F#1”, “G1”, “G#1”, “A2”, “A#2”, “B2”…. “B8”}。

Vol：單音的 velocity，範圍為 0~127，可用立即值或變數指定。

例.

```
Path1: InstNoteOn (0, C1, 127)    ; Falling Edge。播放 0 號一般音色，音高 C1，力度 127
Path2: InstNoteOff (C1)           ; Rising Edge。釋放 C1 音高單音。
```

例.

```
Path1: InstNoteOn (X0, C3, 127)   ; Falling Edge。使用 X0 指定音色，音高 C3，力度 127
```

Path2: InstNoteOff (C3) ; **Rising Edge。釋放 C3 音高單音。**

例.

Path1: InstNoteOn (0, C4, X1) ; **Falling Edge。播放 0 號一般音色，音高 C4，使用 X1 指定力度。**

Path2: InstNoteOff (C4) ; **Rising Edge。釋放 C4 音高單音。**

例.

Path1: InstNoteOn (X0, C4, X1) ; **Falling Edge。使用 X0 指定音色，音高 C4，使用 X1 指定力度。**

Path2: InstNoteOff (C4) ; **Rising Edge。釋放 C4 音高單音。**

5.16.2 InstNoteAllOff

[NX1]

InstNoteAllOff 指令可用做一般樂器與打擊樂器的鍵盤功能，InstNoteAllOff 將釋放當前所有播放中的一般樂器與打擊樂器的播放音符。

例.

Path1: InstNoteAllOff ; **釋放當前所有播音的音符**

5.16.3 DrumNoteOn(Index, Vol) & DrumNoteOff(Index)

[NY5+ / NY6 / NY7 / NX1]

DrumNoteOn 指令和 DrumNoteOff 指令可用作打擊樂器的單音鍵盤功能，但兩個指令必須搭配使用。當按下一個鍵時，DrumNoteOn 指令會播放一個音符並佔用一個通道，直到釋放按鍵或播放完畢為止。當釋放按鍵時，DrumNoteOff 指令將停止播放並釋放佔用的通道。

Index：指定 GM 打擊樂器音色編號，可用立即值或變數指定。

Vol：單音的 velocity，範圍為 0~127，可用立即值或變數指定。

例.

Path1: DrumNoteOn (35, 127) ; **Falling Edge。播放 35 號打擊音色，力度 127。**

Path2: DrumNoteOff (35) ; **Rising Edge。釋放 35 號打擊音色單音。**

例.

Path1: DrumNoteOn (X0, 127) ; **Falling Edge。使用 X0 指定打擊音色，力度 127。**

Path2: DrumNoteOff (X0) ; **Rising Edge。釋放 X0 指定打擊音色單音。**

例.

Path1: DrumNoteOn (35, X1) ; **Falling Edge。播放 35 號打擊音色，使用 X1 指定力度。**

Path2: DrumNoteOff (35) ; **Rising Edge。釋放 35 號打擊音色單音。**

例.

Path1: DrumNoteOn (X0, X1) ; **Falling Edge。使用 X0 指定打擊音色，使用 X1 指定力度。**

Path2: DrumNoteOff (X0) ; **Rising Edge。釋放 X0 指定打擊音色單音。**

5.16.4 NoteVibrato = n / NoteVibrato = Ri

[NY7 / NX1]

當音色透過 *Q-MIDI* 設定 vibrato 效果時，使用者可利用此指令來開啟單音的 vibrato 效果。Vibrato 控制參數，可為立即值或用 Ri 來控制，數值範圍為 0~8，0 表示關閉，1~8 可控制效果的 depth，數值越大，表示 depth 越大。

注意：Percussion 音色不支援 vibrato 功能。

例.

```
Path1: NoteVibrato = 8, InstNoteOn (0, C1, 127)   ; 開啟 vibrato 效果，播放 0 號一般音色，音高 C1，力度 127。
Path2: NoteVibrato = 0                            ; 關閉 vibrato 效果。
```

5.16.5 Gliss(Note, Semitone, Time)

[NY7 / NX1]

當單音播放時，使用者可利用此指令達到 glissando 的效果。

Note：套用此效果的單音音高。

Semitone：指定變頻的半音數，單次變頻範圍為+/-12 個半音。

Time：指定變頻的時間。

- NY7 可支援的時間為 100ms、200ms、300ms、400ms、500ms、600ms、700ms、800ms、900ms、1000ms、1200ms、1400ms、1600ms、1800ms 及 2000ms。
- NX1 支援 100ms ~ 2000ms。

注意：

1. ***Percussion 音色不支援 glissando 效果。***
2. ***單音在播放時間內且符合 note 音高時，指令才能有效執行；根據不同條件，變頻時間約有+/-10%的誤差。***
3. ***NY7 要使用 gliss，須以 Q-MIDI 製作音色時須加入含有 Wheel 控制碼的.mid 檔案，以定義變頻的範圍，否則 Gliss 指令將無效。***
4. ***針對 NY7，此指令的最大變頻範圍為原本的 sub-patch 範圍加上 pitch bend 的變頻範圍。例如，原本 sub-patch 為 C3~D4，pitch bend 範圍為+/-4 個半音，則指令有效變頻範圍則為 G#2~F#4。***

例.

```
Path1: InstNoteOn(0, F4, 127), Gliss(F4, -2, 300ms), Delay(300ms), Gliss(F4, -4, 100ms)
Path2: InstNoteOff(F4)
; 播放 F4 單音後，先用 300ms 降低兩個半音為 D#4，再用 100ms 降低 4 個半音為 B3。
```

5.16.6 MaxSingleNote = n / MaxSingleNote = Ri

[NY5+ / NY6 / NY7]

此指令允許使用者任意調整最大琴鍵音數目，以方便在電子琴應用時，根據不同功能，調整琴鍵音佔用的通道數，確保有足夠的通道提供給 MIDI 播放使用。

例.

Path1: MaxSingleNote = 3 ；將最大琴鍵音數目設定為 3 個。

Path2: R0=5, MaxSingleNote = R0 ；透過 R0 將最大琴鍵音數目設定為 5 個。

注意：此指令的設定值必須小於或等於 Melody_MaxSingleNote 這個 option 的設定值，否則無效。

5.16.7 LongInst_HoldTime(Time)

[NX1]

此指令用於設定有含 Sustain 的長音樂器 NoteOn 後，音符最長播放時間。

Time：指定長音樂器最長播放時間，可支援時間為 1ms ~ 131068ms，設定為 0 時為無限長度播音，預設值為 0。

例.

Path1: LongInst_HoldTime(16000) ；設定長音樂器最長播放 16 秒後進行收音。

Path2: LongInst_HoldTime(0) ；設定長音樂器無限長度播放。

5.16.8 ShortInst_HoldTime(Time)

[NX1]

此指令用於設定沒有 Sustain 的短音樂器 NoteOff 後，要延遲多少時間才銜接 Envelope Release 段。

Time：指定短音樂器延遲時間銜接 Envelope Release，可支援時間為 0ms~1020ms，預設值為 0。

例.

Path1: ShortInst_HoldTime(300) ；設定短音樂器 NoteOff 後延遲 300ms 銜接 Envelope Release 段。

5.16.9 KRecord / KRecordS

[NX1]

進行琴鍵錄音，用於記錄 InstNoteOn / InstNoteOff 的 Note 與 Vol。錄音完成後，可以透過 PlayK 播放。

指令格式如下：

KRecord(Label)

KRecordS(Label)

KRecord：待錄音結束後(包含 Timeout)，執行下一個指令。

KRecordS：開始錄音後，立即執行下一個指令。

Label：**[Record]** 中定義的錄音區段名稱。

注意：

1. ***當 Krecord / KRecordS 被執行時，必須要觸發 InstNoteOn 或 InstNoteOff 一次，才會正式進入琴鍵錄音模式，Timeout 計時功能始啟動。***

2. *當 Krecord / KRecordS 被執行時，任何一次 InstNoteOn / InstNoteOff 都會重置 Timeout 記時。*
3. *KRecord / KRecordS 不支援 Realtime Erasing，需搭配 EraseR / EraseRS 擦除錄音空間。*
4. *錄音空間抵達上限、發生 Timeout、使用 PlayKS 時，將視為 KRecord 正常結束，KRecord 會繼續執行下一個指令。*

例.

```
[Path]
TR1: ERASER($Rec0),KRecord($Rec0)        ; 擦除 Rec0 後使用 KRecord 進行琴鍵錄音。
TR2: PlayK($Rec0)                        ; 使用 PlayK 回放琴鍵錄音結果。
TR3: InstNoteOn (0, C4, 127)             ; Falling Edge。播放 0 號音色，音高 C4，力度 127
TR4: InstNoteOff (C4)                    ; Rising Edge。釋放 C4 音高單音。
TR5: InstNoteOn (0, D4, 127)             ; Falling Edge。播放 0 號音色，音高 D4，力度 127
TR6: InstNoteOff (D4)                    ; Rising Edge。釋放 D4 音高單音。
TR7: InstNoteOn (0, E4, 127)             ; Falling Edge。播放 0 號音色，音高 E4，力度 127
TR8: InstNoteOff (E4)                    ; Rising Edge。釋放 E4 音高單音。
```

5.16.10 WaitKRN

[NX1]

若有琴鍵錄音正在執行，則琴鍵錄音完畢後，再執行下一個指令。沒有琴鍵錄音則立即執行下一個指令。

注意：KRecord=KRecordS+WaitKRN。

例.

```
[Path]
TR1: KRecord($Rec0)
TR2: KRecordS($Rec0), WaitKRN            ; 與 TR1 的執行結果相同。
```

5.16.11 StopKR

[NX1]

StopKR 指令是立即停止當前琴鍵錄音。

例.

```
TR1: KRecordS($Rec0)
TR2: StopKR                              ; 停止琴鍵錄音。
```

5.16.12 PlayK / PlayKS

[NX1]

進行琴鍵回放指令。錄音完成後，可以透過 PlayK 播放。

指令格式如下：

PlayK(Label, Instrument)

PlayKS(Label, Instrument)

PlayK：琴鍵回放結束後，執行下一個指令。

PlayKS：開始琴鍵回放後，立即執行下一個指令。

Label：**[Record]**中定義的錄音區段名稱。

Instrument：使用指定的樂器播放琴鍵錄音，支援 0 ~ 127。

例.

[Path]

TR1: ERASER($Rec0),KRecord($Rec0) ; 擦除 Rec0 後使用 KRecord 進行琴鍵錄音。

TR2: PlayK($Rec0, 0) ; 使用 PlayK 回放琴鍵錄音結果，使用樂器編號 0。

5.16.13 WaitKN

[NX1]

若有琴鍵回放正在執行，則琴鍵回放完畢後，再執行下一個指令。沒有琴鍵回放則立即執行下一個指令。

注意：PlayK=PlayKS+WaitKN。

例.

[Path]

TR1: PlayK($Rec0)

TR2: PlayKS($Rec0), WaitKN ; 與 TR1 的執行結果相同。

5.16.14 StopK

[NX1]

StopK 指令是立即停止當前琴鍵回放。

例.

TR1: PlayKS($Rec0)

TR2: StopK ; 停止琴鍵回放。

5.17 音量指令（Volume Command）

Volume Command				
Vol_Max	Vol_Min	Vol = n	Vol = Ri	Vol++
Vol--	Ri = Vol	Px = Vol	Vol += n	Vol -= n
VolX1	VolX2	CHx_Vol = n	PP_Gain = n	PP_Gain = Ri
PP_Gain++	PP_Gain--	Ri = PP_Gain	PGA_Gain = n	PGA_Gain = Ri
Ri = PGA_Gain	Ri = MixCtrl	Px = MixCtrl	MixCtrl	-

5.17.1 Vol_Max

[NY5 / NY5+ / NY6 / NY7 / NX1]

Vol_Max 可將輸出音量調至最大（Vol=15）。

例. **PowerOn: Vol_Max** ；音量設定成最大。

5.17.2 Vol_Min

[NY5 / NY5+ / NY6 / NY7 / NX1]

Vol_Min 可將輸出音量調至最小。(Vol=0)

注意：NY5 / NX1 音量=0，並非為靜音。

例. **PowerOn: Vol_Min** ；音量設定成最小。

5.17.3 Vol = n

[NY5 / NY5+ / NY6 / NY7 / NX1]

使用者可以藉由 Vol command 來調整適合的音量。

n：欲設定的音量。

- NY5 / NY5+ / NY6 / NY7 / NX1，n = 0 ~ 15.

注意：NY5 / NX1 音量=0，並非為靜音。

例. **Vol=7** ；將 **Volume** 設置成第八階。

5.17.4 Vol = Ri

[NY5 / NY5+ / NY6 / NY7 / NX1]

音量可以藉由 Ri 來控制。

注意：NY5 / NX1 音量=0，並非為靜音。

例. **R1=3, Vol=R1** ；由 **R1** 來控制 **Volume**。

5.17.5 Vol++

[NY5 / NY5+ / NY6 / NY7 / NX1]

Vol++為目前的音量設定遞加一階（最大為 15）。

例. **Vol=8, Vol++** **；則音量=9。**

5.17.6 Vol--

[NY5 / NY5+ / NY6 / NY7 / NX1]

Vol--為目前的音量設定遞減一階，Vol 最小為 0。

注意：NY5 / NX1 音量=0，並非為靜音。

例. **Vol=8, Vol--** **；則音量=7。**

5.17.7 Ri = Vol

[NY5 / NY5+ / NY6 / NY7 / NX1]

Ri=Vol 可以把目前的音量設定取出，存入 Ri。

例. **R1=Vol** **；將目前 Volume 的階數取出後放在 R1。**

5.17.8 Px = Vol

[NY5 / NY5+ / NY6 / NY7 / NX1]

可以把目前的音量輸出到 Port。

例. **PB=Vol** **；將目前 Volume 的階數輸出到 PB。**

5.17.9 Vol += n

[NX1]

Vol += n 可將目前的音量增加 n 階，若目前音量增加 n 超過 15，則設定為 15。

例. **Vol += 2**

5.17.10 Vol -= n

[NX1]

Vol -= n 可將目前的音量減少 n 階，若目前音量增加 n 小於 0，則設定為 0。

注意：NX1 音量=0，並非為靜音。

例. **Vol -= 2**

5.17.11 VolX1

[NY6]

音量的階數為原本 Vol 的階數。

例. **Vol=8, VolX1** ; 音量=8。

5.17.12 VolX2

[NY6]

音量的階數為原本 Vol 的階數乘 2。

注意：使用 VolX2 可將音量增大，但有可能造成爆音，需要注意使用。

例. **Vol=0x8, VolX2** ; 音量=16。

5.17.13 CHx_VOL = n

[NX1]

CHx_VOL = n 控制指定聲音輸出通道的音量。n 介於 0~15 之間。代表音量大小。

CHx 代表 channel，對應到 PlayV / PlayVS、SPIPlay / SPIPlayS、PlayM / PlayMS 等指令所使用的聲音輸出通道。

注意：音量=0，為靜音。

例.

SPIPlay(Ch0,0),Ch0_vol=5 ; 設定 Ch0 音量為 5

5.17.14 PP_Gain = n

[NX1 OTP]

使用者可以藉由 PP_Gain 指令來調整適合的 Push-Pull 增益，n=0~16。PP_Gain=0 表示靜音。PP_Gain=16，輸出音量最大。PP_Gain 預設值為 9。PP_Gain 設定值對應 NY7 Push-Pull Volume 設定如下：

NX1 PP_Gain	Push-Pull Volume
16	130%
15	122%
14	114%
13	107%
12	100%
11	94%
10	88%
9	83%
8	77%

NX1 PP_Gain	Push-Pull Volume
7	71%
6	66%
5	60%
4	55%
3	50%
2	45%
1	40%
0	靜音

例. **PP_Gain=9** ; 將 PP 增益設置成第 9 階。

5.17.15 PP_Gain = Ri

[NX1 OTP]

使用者可以藉由 PP_Gain 指令來調整適合的 Push-Pull 增益，Ri=0~16。

例. **R1=3, PP_Gain=R1** ; 由 R1 來控制 PP 增益。

5.17.16 PP_Gain++

[NX1 OTP]

PP_Gain++為目前的 Push-Pull 增益設定遞加一階，PP_Gain 最大為 16。

例. **PP_Gain=8, PP_Gain++** ; 則 PP 增益=9。

5.17.17 PP_Gain--

[NX1 OTP]

PP_Gain--為目前的 Push-Pull 增益設定遞減一階，PP_Gain 最小為 0。

例. **PP_Gain=8, PP_Gain--** ; 則 PP 增益=7。

5.17.18 Ri = PP_Gain

[NX1 OTP]

Ri=PP_Gain 可以把目前的 Push-Pull 增益設定取出，存入 Ri。

例. **R1=PP_Gain** ; 將目前 PP 增益設定取出後放在 R1。

5.17.19 PGA_Gain = n

[NX1]

使用者可以藉由 PGA_Gain 指令來調整適合的 PGA 增益，可以對麥克風收錄的訊號做放大。n=0~31，0 為最小增益（仍然有訊號），31 為最大。預設值為 12。

PGA_Gain 的設定值對輸入訊號的增益倍率請參考以下列表。使用者可以依應用來調整 PGA_Gain，以語音辨識而言，倍率建議為 60 ~ 70 之間，放大倍率若過大會造成飽和波形失真不利於辨識的判斷。

請留意 NX1_FDB Ver.A 2016/11/21 上的 Ropi (R10) 為 5.1KΩ，NX1_FDB Ver.B 2017/10/5 (非正式的藍色板) 的 Ropi (R5)為 2KΩ，NX1_FDB Ver.B 2018/8/22 (正式的藍色板) 的 Ropi 為 0Ω 且不可更換。

PGA_Gain	倍率@Ropi = 5.1KΩ	倍率@Ropi = 2KΩ	倍率@Ropi = 0Ω
31	74.5	149.9	387.1
30	73.5	144.6	356.0
29	72.4	139.3	326.9
28	71.2	134.0	300.6
27	69.9	128.7	276.2
26	68.5	123.1	253.2
25	67.0	117.7	232.5
24	65.4	112.4	213.6
23	63.7	107.0	195.9
22	61.8	101.6	179.3
21	59.8	96.5	164.8
20	57.8	90.7	149.4
19	55.6	85.0	135.3
18	53.3	79.6	122.6
17	50.9	74.2	110.9
16	48.5	68.1	98.4
15	46.0	63.2	88.9
14	43.3	58.6	80.5
13	40.8	53.8	72.1
12	38.2	49.0	64.2
11	35.6	44.5	57.1
10	33.0	40.5	50.9
9	30.4	36.7	45.3
8	27.9	33.1	40.3
7	25.5	29.7	35.5
6	23.1	26.5	31.2
5	20.8	23.6	27.4
4	18.7	20.9	24.0
3	16.6	18.3	20.8
2	14.6	15.9	17.9
1	12.8	13.8	15.3
0	11.0	11.8	13.0

例. **PGA_Gain=10** **；將 PGA Gain 設定為 10。**

5.17.20 PGA_Gain = Ri

[NX1]

藉由暫存器設定 PGA_Gain 值。

例. **R1=10, PGA_Gain=R1** ; 將 PGA Gain 設定為 10。

5.17.21 Ri = PGA_Gain

[NX1]

將目前 PGA_Gain 值讀到暫存器。

例. **R1 = PGA_Gain** ; 將 PGA Gain 的值存到 R1。

5.17.22 Ri = MixCtrl

[NY5]

將目前的 MixCtrl 設定值，存放到 RAM。

例. **R0=MixCtrl** ; 將目前的 MixCtrl 設定存到 R0。

5.17.23 Px = MixCtrl

[NY5]

將目前的 MixCtrl 設定值，輸出到 Port。

例. **PB=MixCtrl** ; 將目前的 MixCtrl 設定輸出到 PB。

5.17.24 MixCtrl

[NY5 / NX1]

控制各頻道的混音輸出。

NY5 系列可以使用 CH0，CH1，CH2，CH3。可用設定如下：

MixCtrl Command	說明	設定值
MixCtrl(2, 0, 2, 0)	CH0/CH2 各一半音量。	0x3
MixCtrl(2, 0, 1, 1)	CH0 50%音量，CH2/CH3 各 25%音量。	0x1
MixCtrl(1, 1, 2, 0)	CH0/CH1 各 25%音量，CH2 50%音量。	0x2
MixCtrl(1, 1, 1, 1)	各 channel 平均分配音量。	0x0
MixCtrl(4, 0, 0, 0)	CH0 最大音量，關閉其餘 channel。	0x9
MixCtrl(2, 2, 0, 0)	CH0/CH1 各一半音量。	0x8
MixCtrl(0, 0, 4, 0)	CH2 最大音量，關閉其餘 channel。	0xE
MixCtrl(0, 0, 2, 2)	CH2/CH3 各一半音量。	0xC

NX1 控制各頻道的混音輸出，須填入百分比，共有 0%，25%，50%，75%，100%幾種選項。

例. **MixCtrl(50%,25%,75%,100%)** ; 將目前的 **MixCtrl** 設定輸出到 **PB**。

注意：

1. ***NY5 中，若使用者未寫 MixCtrl 時，系統預設值為 MixCtrl(1, 1, 1, 1)。***
2. ***NX1 中，若使用者未寫 MixCtrl 時，系統預設值為 MixCtrl(100%, 100%, 100%, 100%)。***

5.18 觸摸鍵指令（TouchKey Command）

TouchKey Command			
TouchKey_ON	TouchKey_OFF	TouchKey_CLR	TouchKey_Scan_Slow
TouchKey_Scan_Normal	TouchKey_Sensitivity	Calibrate_ON	Calibrate_OFF
AutoJudge_Calibrate	Enforce_Calibrate_Normal	Enforce_Calibrate_Sleep	Ri = TouchKey(Px)
Touchkey_Count	Touchkey_BGCount	-	-

5.18.1 TouchKey_ON

[NY9T]

開啟觸摸鍵掃描功能。若是開啟該功能，則觸摸鍵會持續做掃描。

注意：

1. ***系統預設值為 TouchKey_ON，PowerOn 後無須另外再下達一次 TouchKey_ON 指令。***
2. ***PowerOn 後，觸摸鍵掃描速度預設為 TouchKey_Scan_Normal。***
3. ***TouchKey_ON 後，會保持上一次設定的掃描狀態。***

5.18.2 TouchKey_OFF

[NY9T]

關閉觸摸鍵掃描功能。若是關閉該功能，則觸摸鍵不做掃描。

例.

[Input State]

KEY1:

[Path]

PowerOn: TouchKey_OFF, Delay(0.3), KEY1, TouchKey_ON

; PowerOn 時，將觸摸鍵掃描功能關閉，待延遲 0.3 秒後，才開啟觸摸鍵掃描。

5.18.3 TouchKey_CLR

[NY9T]

用於清除當前觸摸鍵狀態，當按鍵狀態被清除後，觸摸鍵會重新掃描。

例.

[Input State]

KEY1: TR1

[Path]

PowerOn: KEY1 **; TouchKey State 設置成 KEY1 狀態。**

TR1: PlayA(Ch1,$VIO0), TouchKey_CLR

; 按下 TR1 後，執行 PlayA(Ch1,$VIO0)。執行完 PlayA(Ch1,$VIO0)後，執行 TouchKey_CLR 指令，即清除當前的觸摸鍵的狀態，重新掃描觸摸鍵狀態。當 VIO0 播放完了之後，如果 TR1 有被按住時，程式會繼續執行 PlayA(Ch1,$VIO0)，如果 TR1 沒有被按住時，則程式進入 Sleep 狀態。

5.18.4 TouchKey_Scan_Slow

[NY9T]

開啟觸摸鍵慢速掃描功能，則觸摸鍵會間歇性對按鍵做掃描，可以達到較省電的效能。

注意：

1. *進入睡眠前，建議切換成 TouchKey_Scan_Slow 以達到省電。*
2. *當觸摸鍵被偵測到時，會自動切換至一般掃描模式。*
3. *若按鍵有被按住時，則不會進入慢速掃描模式。*

例.

[Path]

PowerOn: TouchKey_ON

Sleep: TouchKey_Scan_Slow **; 將觸摸鍵掃描切換成慢速掃描模式。**

5.18.5 TouchKey_Scan_Normal

[NY9T]

開啟觸摸鍵快速掃描功能，則觸摸鍵會回復為一般掃描方式。(預設值為 TouchKey_Scan_Normal)

注意：當 TouchKey_Scan_Normal 時，觸摸鍵的反應速度會較 TouchKey_Scan_Slow 快。

例.

[Path]

PowerOn: TouchKey_ON

Sleep: TouchKey_Scan_Slow **; 將觸摸鍵掃描切換成慢速掃描模式。**

TR1: TouchKey_Scan_Normal **; 將觸摸鍵掃描切換成一般掃描模式。**

5.18.6 TouchKey_Sensitivity(Level)

[NY9T / NX1]

切換觸摸鍵的靈敏度，數值越小越靈敏，NY9T 共有八段可調整，預設值為 4。NX1 共有四段可調整，

預設值為 0。

Level：設定觸摸鍵的靈敏度。NY9T 可直接指定靈敏度的階數或是由暫存器或外部電阻來調整，範圍 0 ~ 7。Radj 階數表請參考 ReadRadj(Ri)章節中說明。NX1 僅提供直接設定，範圍 0 ~ 3。

- 直接設定靈敏度階數 *例.* **TouchKey_Sensitivity(4)**
- Ri（支援 Ri） *例.* **TouchKey_Sensitivity(R0)**
- Radj（支援外部電阻調整靈敏度） *例.* **TouchKey_Sensitivity(Radj)**

例. 如何利用 bonding option，選擇不同的觸摸鍵靈敏度。

[Path]

```
PowerOn: Switch(PE[x x d d])= [Sens0, Sens1, Sens2, Sens3]    ; PE[1:0]為 bonding option。
Sens0: TouchKey_Sensitivity(0)                ; PE[1:0]為 00 時，則靈敏度最高。
Sens1: TouchKey_Sensitivity(2)                ; PE[1:0]為 01 時，則靈敏度介於最高和一般之間。
Sens2: TouchKey_Sensitivity(4)                ; PE[1:0]為 10 時，則靈敏度一般。
Sens3: TouchKey_Sensitivity(7)                ; PE[1:0]為 11 時，則靈敏度最低。
```

例. 如何利用暫存器因應不同模式下的行為，動態調整觸摸鍵靈敏度。

[Path]

```
PowerOn:
Main: R0=0, R_Mode_Count>=8?SetSens, R0=4, SetSens
SetSens: TouchKey_Sensitivity(R0)             ; 則靈敏度依據 R0 的數值設定。
```

例. 如何利用外部電阻上電時來選擇不同的觸摸鍵靈敏度。

[Path]

```
PowerOn: TouchKey_Sensitivity(Radj)           ; 根據 IC 偵測到外部電阻的階數，自動調整靈敏度。
```

注意：當外部電阻未連接或是電阻值超過範圍時，則該指令將會維持原本的靈敏度。

5.18.7 Calibrate_ON

[NY9T]

開啟觸摸鍵的靈敏度校正功能（預設值為開啟）。

注意：NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支援此指令。

5.18.8 Calibrate_OFF

[NY9T]

關閉觸摸鍵的靈敏度校正功能。

5.18.9 AutoJudge_Calibrate

[NY9T]

自動觸摸鍵的靈敏度校正功能。開啟後會自動校正環境變數，但若有觸摸鍵被按住，則會終止校正程序。

例. 每 4 秒進行觸摸鍵校正一次

[Path]

4Sec: AutoJudge_Calibrate ; 每隔 4 秒鐘，則觸摸鍵會自動校正一次，但觸摸鍵校正功能若被關閉的話，將不會進行校正。

TR1R: Calibrate_ON ; 開啟觸摸鍵校正功能。

TR2R: Calibrate_OFF ; 關閉觸摸鍵校正功能。

5.18.10 Enforce_Calibrate_Normal

[NY9T / NX1]

強制觸摸鍵的靈敏度校正功能。開啟後會自動校正環境變數，若當前有觸摸鍵被按住，仍會強迫進行校正。

注意：

1. *當 Calibrate_OFF 時，則執行到 Enforce_Calibrate_Normal 指令時，觸摸鍵將不會進行校正，並且會執行接續 Enforce_Calibrate_Normal 後面的指令。*
2. *強烈建議每間隔一段時間就進行觸摸鍵校正，以避免因環境改變而造成觸摸鍵可能會反應不佳的狀況。*
3. *若是按鍵被長按超過 40 秒，則建議強制校正一次。*
4. *Enforce_Calibrate_Normal 執行完畢後，若有程式中含有"Enforce_Calibrate"路徑時，則會先執行該路徑。*
5. *Enforce_Calibrate_Normal 執行完畢後，是否會執行按鍵放開路徑將與使用者是否有清除觸摸鍵狀態有關。若不清除觸摸鍵狀態，則會自動執行按鍵放開的路徑。*

例. 按鍵長按 40 秒，則強迫觸摸鍵校正一次

[Symbol]

KeyPress = R0

Count = R1

[Path]

4Sec: KeyPress=1?Check40sec, AutoJudge_Calibrate

Check40sec: Count++, Count=10?Timeout_40sec

Timeout_40sec: Count=0, Enforce_Calibrate_Normal

; 每隔 4 秒鐘，則觸摸鍵會自動校正一次。且當按鍵持續被按下 40 秒後，會強迫進行校正一次。

TR1R: Calibrate_ON ; 開啟觸摸鍵校正功能。

TR2R: Calibrate_OFF ; 關閉觸摸鍵校正功能。

TR3R: KeyPress=1, Count=0, Play ; 當 KEY3 被按下後，則 KeyPress=1（表示按鍵被按下）。

TR3F: KeyPress=0, Count=0 ; 當 KEY3 被放開後，則 KeyPress=0。（表示按鍵被放開）。

Enforce_Calibrate: SW_Reset

; 執行完 Enforce_Calibrate 後，若不需記憶原本的狀態，可直接進行軟體重置，來回復成開機時的狀態。

若沒有進行軟體重置的話，將會執行按鍵放開的動作。

5.18.11 Enforce_Calibrate_Sleep

[NY9T]

強制觸摸鍵的靈敏度校正功能。開啟後會自動校正環境變數，若當前有觸摸鍵被按住，仍會強迫進行校正。

注意:

1. *當 Calibrate_OFF 時，則執行到 Enforce_Calibrate_Sleep 指令時，觸摸鍵將不會進行校正，並且會執行接續 Enforce_Calibrate_Sleep 後面的指令。*
2. *強烈建議每間隔一段時間就進行觸摸鍵校正，以避免因環境改變而造成觸摸鍵可能會反應不佳的狀況。*
3. *Enforce_Calibrate_Sleep 執行完畢後，即使程式中含有"Enforce_Calibrate"路徑亦不會執行該路徑。*
4. *Enforce_Calibrate_Sleep 執行後，不需等待靈敏度校正完成，即可進入 Sleep。*

例. 每 0.5 秒，則強迫觸摸鍵校正一次

[Path]

```
500ms: Enforce_Calibrate_Sleep          ; 每隔 500 毫秒，則觸摸鍵會自動校正一次。
```

5.18.12 Ri = TouchKey(Px)

[NY9T]

將觸摸鍵狀態存到 Ri，以 Port 為單位，每次讀取指定 Port 的 4 支腳位狀態。NY9T001A/004A series：x = PA，NY9T008A series：x = PA / PB，NY9T016A series：x = PA ~ PD。

```
例. R0=TouchKey(PA)                     ; 將觸摸鍵 PA.0 ~ PA.3 當前的狀態存到 R0。
```

5.18.13 Var = Touchkey_Count(TouchKey)

[NX1]

此指令將指定的 TouchKey 掃描 Count 數值回傳至變數，用於調整 TouchKey 設定。

TouchKey：欲回傳 Count 數值的 TouchKey，依照使用數量輸入範圍為 0 ~ (n - 1)。

```
例. Var32_Count=TouchKey_Count(0)       ; 將 TouchKey0 的 Count 數值傳至變數
                                        Var32_Count。
```

5.18.14 Var = Touchkey_BGCount(TouchKey)

[NX1]

此指令將指定的 TouchKey 背景 Count 數值回傳至變數，用於調整 TouchKey 設定。

TouchKey：欲回傳 Count 數值的 TouchKey，依照使用數量輸入範圍為 0 ~ (n - 1)。

```
例. Var32_BGCount=TouchKey_BGCount(0)   ; 將 TouchKey0 的 Count 數值傳至變數
                                        Var32_BGCount。
```

5.19 查表指令（Table Command）

Table Command	
TableL(TableName, Rx/Xx, Ry/Xy, Ri)	TableM(TableName, Rx/Xx, Ry/Xy, Ri)
TableH(TableName, Rx/Xx, Ry/Xy, Ri)	Table(TableName, Rx/Xx, Ry/Xy, Rh, Rm, Rl)
TableL(TableName, X, Y, Ri)	TableM(TableName, X, Y, Ri)
TableH(TableName, X, Y, Ri)	Table(TableName, X, Y, Rh, Rm, Rl)
TableL(TableName, Rx/Xx, Ry/Xy, Xi)	TableH(TableName, Rx/Xx, Ry/Xy, Xi)
Table(TableName, Rx/Xx, Ry/Xy, Xh, Xl)	TableL(TableName, X, Y, Xi)
TableH(TableName, X, Y, Xi)	Table(TableName, X, Y, Xh, Xl)
Table(TableName, VarX, VarY, VarR)	-

5.19.1 TableL(TableName, Rx/Xx, Ry/Xy, Ri)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

TableL(TableName, Rx/Xx, Ry/Xy, Ri) 指令為間接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 Low-Nibble。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

Rx/Xx：Rx/Xx 內容值代表 X 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Ry/Xy：Ry/Xy 內容值代表 Y 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Ri：將 Table 取到的資料放入 Ri 內。

5.19.2 TableM(TableName, Rx/Xx, Ry/Xy, Ri)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

TableM(TableName, Rx/Xx, Ry/Xy, Ri) 指令為間接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 Middle-Nibble。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

Rx/Xx：Rx/Xx 內容值代表 X 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Ry/Xy：Ry/Xy 內容值代表 Y 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Ri：將 Table 取到的資料放入 Ri 內。

5.19.3 TableH(TableName, Rx/Xx, Ry/Xy, Ri)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

TableH(TableName, Rx/Xx, Ry/Xy, Ri) 指令為間接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 High-Nibble。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

Rx/Xx：Rx/Xx 內容值代表 X 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Ry/Xy：Ry/Xy 內容值代表 Y 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Ri：將 Table 取到的資料放入 Ri 內。

5.19.4 Table(TableName, Rx/Xx, Ry/Xy, Rh, Rm, Rl)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

間接定址指令，可用來讀取在 **[Table]** 段落中定義數值。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

Rx/Xx：Rx/Xx 內容值代表 X 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Ry/Xy：Ry/Xy 內容值代表 Y 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Rh：將 Table 取到的 High-Nibble 資料放入 Rh 內。

Rm：將 Table 取到的 Middle-Nibble 資料放入 Rm 內。

Rl：將 Table 取到的 Low-Nibble 資料放入 Rl 內。

例.

```
[Table]
Trans:
{
[0x001, 0x003, 0x005, 0x007, 0x009],
[0x004, 0x005, 0x3F6, 0x008, 0x010],
[0x000, 0x001, 0x002, 0x003, 0x0F4]
}
[Path]
P1: R0=2, R1=1, TableL(trans,R0,R1,R2)          ; R2=0x6。
P2: R0=2, R1=1, TableM(trans,R0,R1,R2)          ; R2=0xF。
P3: R0=2, R1=1, TableH(trans,R0,R1,R2)          ; R2=0x3。
P4: R0=2, R1=1, Table(trans,R0,R1,R4,R3,R2)     ; R2=0x6，R3=0xF，R4=0x3。
```

5.19.5 TableL(TableName, X, Y, Ri)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

TableL(TableName, X, Y, Ri) 指令為直接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 Low-Nibble。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

X：代表 X 軸的位址。

Y：代表 Y 軸的位址。

Ri：將 Table 取到的資料放入 Ri 內。

5.19.6 TableM(TableName, X, Y, Ri)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

TableM(TableName, X, Y, Ri) 指令為直接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 Middle-Nibble。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

X：代表 X 軸的位址。

Y：代表 Y 軸的位址。

Ri：將 Table 取到的資料放入 Ri 內。

5.19.7 TableH(TableName, X, Y, Ri)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

TableH(TableName, X, Y, Ri) 指令為直接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 High-Nibble。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

X：代表 X 軸的位址。

Y：代表 Y 軸的位址。

Ri：將 Table 取到的資料放入 Ri 內。

5.19.8 Table(TableName, X, Y, Rh, Rm, Rl)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

Table(TableName, X, Y, Rh, Rm, Rl) 指令為直接定址指令，可用來讀取在 **[Table]** 段落中定義數值。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

X：代表 X 軸的位址。

Y：代表 Y 軸的位址。

Rh：將 Table 取到的 High-Nibble 資料放入 Rh 內。

Rm：將 Table 取到的 Middle-Nibble 資料放入 Rm 內。

Rl：將 Table 取到的 Low-Nibble 資料放入 Rl 內。

例.

```
[Table]
Trans:
{
[0x001, 0x003, 0x005, 0x007, 0x009],
[0x004, 0x005, 0x3F6, 0x008, 0x010],
[0x000, 0x001, 0x002, 0x003, 0x0F4 ]
}

[Path]
P1: TableL(trans, 2, 1,R2)              ; R2=0x6。
P2: TableM(trans, 2, 1,R2)              ; R2=0xF。
```

P3: TableH(trans, 2, 1,R2) **; R2=0x3。**

P4: Table(trans, 2, 1,R4,R3,R2) **; R2=0x6，R3=0xF，R4=0x3。**

5.19.9 TableL(TableName, Rx/Xx, Ry/Xy, Xi)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

TableL(TableName, Rx/Xx, Ry/Xy, Xi) 指令為間接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 Low-Byte。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

Rx/Xx：Rx/Xx 內容值代表 X 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Ry/Xy：Ry/Xy 內容值代表 Y 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Xi：Table 取到的資料放入 Xi 內。

5.19.10 TableH(TableName, Rx/Xx, Ry/Xy, Xi)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

TableH(TableName, Rx/Xx, Ry/Xy, Xi) 指令為間接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 High-Byte。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

Rx/Xx：Rx/Xx 內容值代表 X 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Ry/Xy：Ry/Xy 內容值代表 Y 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Xi：將 Table 取到的資料放入 Xi 內。

5.19.11 Table(TableName, Rx/Xx, Ry/Xy, Xh, Xl)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

Table(TableName, Rx/Xx, Ry/Xy, Xh, Xl) 指令為間接定址指令，可用來讀取在**[Table]** 段落中定義數值的 High-Byte 和 Low-Byte。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

Rx/Xx：Rx/Xx 內容值代表 X 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Ry/Xy：Ry/Xy 內容值代表 Y 軸的位址，可選擇用 4-bit 或 8-bit 變數定址。

Xh：將 Table 取到的高位值放入 Xh 內。

Xl：將 Table 取到的低位值放入 Xl 內。

例.

```
[Table]
Trans:
{
[0x001, 0x003, 0x005, 0x007, 0x009],
[0x004, 0x005, 0x3F6, 0x008, 0x010],
```

[0x000, 0x001, 0x002, 0x003, 0x0F4]

}

[Path]

P1 : X0=2, X1=1, TableL(trans,X0,X1,X2)　　; X2=0xF6。

P2 : X0=2, X1=1, TableH(trans,X0,X1,X2)　　; X2=0x03。

P3 : X0=2, X1=1, Table(trans,X0,X1,X2,X3)　; X2=0x03，X3=0xF6。

5.19.12 TableL(TableName, X, Y, Xi)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

TableL(TableName, X, Y, Xi) 指令為直接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 Low-Byte。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

X：代表 X 軸的地址。

Y：代表 Y 軸的地址。

Xi：將 Table 取到的資料放入 Xi 內。

5.19.13 TableH(TableName, X, Y, Xi)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

TableH(TableName, X, Y, Xi) 指令為直接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 High-Byte。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

X：代表 X 軸的地址。

Y：代表 Y 軸的地址。

Xi：將 Table 取到的資料放入 Xi 內。

5.19.14 Table(TableName, X, Y, Xh, Xl)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

Table(TableName, X, Y, Xh, Xl) 指令為直接定址指令，可用來讀取在 **[Table]** 段落中定義數值的 High-Byte 和 Low-Byte。

TableName：是一個在 **[Table]** 段落中以字元定義的 table 名稱。

X：代表 X 軸的地址。

Y：代表 Y 軸的地址。

Xh：將 Table 取到的高位值放入 Xh 內。

Xl：將 Table 取到的低位值放入 Xl 內。

例.

```
[Table]
Trans:
{
[0x001, 0x003, 0x005, 0x007, 0x009],
[0x004, 0x005, 0x3F6, 0x008, 0x010],
[0x000, 0x001, 0x002, 0x003, 0x0F4]
}

[Path]
P1 : TableL(trans,2,1,X2)              ; X2=0xF6。
P2 : TableH(trans,2,1,X2)              ; X2=0x03。
P3 : Table(trans,2,1,X2,X3)            ; X2=0x03，X3=0xF6。
```

5.19.15 Table(TableName, VarX, VarY, VarR)

[NX1]

間接定址指令，可用來讀取在 **[Table]** 段落中定義數值。

Table(TableName, VarX, VarY, VarR)

VarR = TableName[VarX][VarY]

TableName：是一個在**[Table]** 段落中以字元定義的 table 名稱。

VarX：X 軸的位址。

- 立即值。
- **[Variable]** 中定義的變數。

VarY：Y 軸的位址。

- 立即值。
- **[Variable]** 中定義的變數。

VarR：將 Table 取到的資料放入 VarR 內。

例.

```
[Table]
Trans:
{
[0x001, 0x003, 0x005, 0x007, 0x009],
[0x004, 0x005, 0x3F6, 0x008, 0x010],
[0x000, 0x001, 0x002, 0x003, 0x0F4]
}

[Variable]
Var8: R0, R1, R2

[Path]
P1: R0=2, R1=1, Table(trans,R0,R1,R2)      ; R2=0x3F6。
```

P2: R0=2, R1=1, R2=trans[R0][R1]　　; R2=0x3F6。

5.20 紅外線指令（IR Command）

IR Command				
IR_TX=data	IR_TX(Rl:Rk:Rj:Ri)	IR_TX(Xj:Xi)	IR_TX_WaitN	IR_RX_ON
IR_RX_OFF	[Rl,Rk,Rj,Ri] = IR_RX	[Xj,Xi] = IR_RX	IR_RX != data?Path	
IR_TX_Busy?Path	IR_Carrier_On	IR_Carrier_Off	-	-

5.20.1 IR_TX=data

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

直接 IR 代碼發射。

例. **IR_TX=0xF or IR_TX=15　　; IR 發射代碼 0x0F。**

例. **IR_TX=0xFFF or IR_TX=4095　　; IR 發射代碼 0xFFF。**

例. **IR_TX=0xFFFF or IR_TX=65535　　; IR 發射代碼 0xFFFF。**

5.20.2 IR_TX(Rl:Rk:Rj:Ri)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

提供使用 RAM 的方式來發射代碼。Ri=bit0 ~ 3，Rj=bit4 ~ 7，Rk=bit8 ~ 11，Rl=bit12 ~ 15。

例. 4-bit mode 時，R0=0xF。

TR1: IR_TX(R0)　　; 紅外線發射代碼 0xF。

例. 8-bit mode 時，R0=0xF，R1=0x3。

TR1: IR_TX(R1:R0)　　; 紅外線發射代碼 0x3F。

例. 12-bit mode 時，R0=0xF，R1=0xF，R2=0x1。

TR1: IR_TX(R2:R1:R0)　　; 紅外線發射代碼 0x13F。

例. 16-bit mode 時，R0=0xF，R1=0x3，R2=0x01，R3=0x0。

TR1: IR_TX(R3:R2:R1:R0)　　; 紅外線發射代碼 0x013F。

5.20.3 IR_TX(Xj:Xi)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

提供使用 RAM 的方式來發射代碼。Xj：為 High-Byte；Xi：為 Low-Byte。

例. 4-bit mode 時，X0=0x3F。

TR1: IR_TX(X0)　　; 紅外線發射代碼 0x0F。

例. 8-bit mode 時，X0=0x3F。

```
TR1: IR_TX(X0)                          ; 紅外線發射代碼 0x3F。
```

例. 12-bit mode 時，X0=0x3F，X1=0x01。

```
TR1: IR_TX(X1, X0)                      ; 紅外線發射代碼 0x13F。
```

例. 16-bit mode 時，X0=0x3F，X1=0x01。

```
TR1: IR_TX(X1:X0)                       ; 紅外線發射代碼 0x013F。
```

5.20.4 IR_TX_WaitN

[NY5+ / NX1]

若有 IR 資料正在傳送中，則等待至資料傳送完畢後，再執行下一個指令。IR 資料沒有傳送則立即執行下一個指令。

5.20.5 IR_RX_ON

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

開啟紅外線接收功能。若是沒有開啟該功能，則紅外線無法接收。

```
IR_RX_ON                                ; 開啟紅外線接收功能。
```

5.20.6 IR_RX_OFF

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

關閉紅外線接收功能。

```
IR_RX_OFF                               ; 關閉紅外線接收功能。
```

5.20.7 [Rl, Rk, Rj, Ri] = IR_RX

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

使用者可透過指令將 RX 收到的資料存在指定的 RAM，Ri=bit0~3，Rj=bit4~7，Rk=bit8~11，Rl=bit12~15。

注意：在未收到 RX 資料前，使用此指令會讀取到未知資料。

例. 假設 IR 設定為 12-bit 模式

```
[R0] = IR_RX                            ; 將 RX 的 Bit[3:0]存到 R0。
[R1, R0] = IR_RX                        ; 將 RX 的 Bit[3:0]存到 R0，Bit[7:4]存到 R1。
[R2, R1, R0] = IR_RX                    ; 將 RX 的資料由低到高依序存入 R0、R1 及 R2。
```

5.20.8 [Xj, Xi] = IR_RX

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

使用者可透過指令將 RX 收到的資料存在指定的 RAM，Xi=bit0~7，Xj=bit8~15。

注意：在未收到 RX 資料前，使用此指令會讀取到未知資料。

例. 假設 IR 設定為 12-bit 模式

```
[X0] = IR_RX                ; 將 RX 的 Bit[7:0]存到 X0。
[X1, X0] = IR_RX            ; 將 RX 的 Bit[7:0]存到 X0，Bit[11:8]存到 X1，X1[7:4]
                              會清為 0。
```

5.20.9 IR_RX = data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

當接收到的 IR 代碼等於 data 值時，跳到 Path。

```
例. IR_RX = 0? Path         ; 當接收到的代碼為 0 時，跳到 Path。
```

5.20.10 IR_RX != data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

當接收到的 IR 代碼不等於 data 值時，跳到 Path。

```
例. IR_RX != 0? Path        ; 當接收到的代碼不為 0 時，跳到 Path。
```

5.20.11 IR_TX_Busy?Path

[**NX1**]

當 IR 正在傳送資料時，跳到 Path。

5.20.12 IR_Carrier_On

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

啟動 IR 載波輸出，提供手動操作的控制指令。

5.20.13 IR_Carrier_Off

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

關閉 IR 載波輸出，提供手動操作的控制指令。

5.21 串列控制傳送指令（Serial Control TX Command）

Serial Control TX Command		
SC_TX(Mode, data)	SC_TX(Mode, Ri:Rj:Rl:Rk)	SC_TX(Mode, Xi:Xj)

5.21.1 SC_TX (Mode, Parameter)

[NY9T008A / NY9T016A]

可連接一般微控制器，將 NY9T 當成微控制器的按鍵。提供 3 種不同的傳輸協定，分別是 SPI_Like、NY3 Serial_Trigger、IR_Trigger。僅支援 16-bit 模式。

SPI_Like：使用 2 根腳位，達成類似 SPI 傳輸協定。

NY3 Serial_Trigger：使用 2 個腳位來連接 NY3 系列，可將 NY9T 當成 NY3 的按鍵。詳細的連接方式，請參閱 NY3C/3D 規格書。

IR_Trigger：使用 1 個腳位來模擬 IR 的接收訊號，可直接使用目前的 NY4/5/7 系列的 IR 通訊。

Mode：選擇 Serial Control 訊號的通訊協定或是依照腳位來做自動選擇通訊協定的模式。

- 自動選擇模式 — ***例.* SC_TX(Auto, 0xFFFF)**
- SPI_Like — ***例.* SC_TX(SPI_Like, 0xFFFF)**
- NY3 — ***例.* SC_TX(NY3, 0xFFFF)**
- IR_Trigger — ***例.* SC_TX(IR_Trigger, 0xFFFF)**

Parameter：傳輸數據。

- data（支援 16-bit） — ***例.* SC_TX(Auto, 0xFFFF)**
- Ri（支援 1 ~ 4 個 Ri） — ***例.* SC_TX(Auto, R3:R2:R1:R0)**
- Xi（支援 1 ~ 2 個 Xi） — ***例.* SC_TX(Auto, X1:X0)**

注意：

1. 使用該功能前，需先在 Option 中設定傳輸協定。

2. 左邊的暫存器為最高位，訊號由最高位元開始傳送，傳輸的數據固定為 16-bit。

5.21.2 SC_TX(Mode, Rl:Rk:Rj:Ri)

[NY9T008A / NY9T016A]

提供使用 RAM 的方式來傳輸串列訊號。Ri=bit0~3，Rj=bit4~7，Rl=bit8~11，Rk=bit12~15。

例. Auto_Detect 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(Auto, R3:R2:R1:R0)

; 由 Serial Control 腳位，自動偵測使用的傳輸協定，輸出 [R3:R2:R1:R0] 暫存器內容值的訊號。

例. SPI_Like 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(SPI_Like, R3:R2:R1:R0)

***;* 由 Serial Control 腳位，使用 SPI_Like 傳輸協定，輸出 [R3:R2:R1:R0] 暫存器內容值的訊號。**

例. NY3 Serial_Trigger 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(NY3, R3:R2:R1:R0)

***;* 由 Serial Control 腳位，使用 NY3 Serial_Trigger 傳輸協定，輸出 [R3:R2:R1:R0] 暫存器內容值的訊號。若暫存器內容值為皆為 0 時，僅輸出 Reset 訊號。**

例. IR_Trigger 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(IR_Trigger, R3:R2:R1:R0)

; 由 Serial Control 腳位，使用 IR_Trigger 傳輸協定，輸出 [R3:R2:R1:R0] 暫存器內容值的訊號。

5.21.3 SC_TX(Mode, Xj:Xi)

[NY9T008A / NY9T016A]

提供使用 RAM 的方式來傳輸串列訊號。Xi=bit0~7，Xj=bit8~15。

例. Auto_Detect 模式

X1=0x0, X0=0, SC_TX(Auto, X1:X0)

; 由 Serial Control 腳位，自動偵測使用的傳輸協定，輸出 [X1:X0] 暫存器內容值的訊號。

例. SPI_Like 模式

X1=0x0, X0=0, SC_TX(SPI_Like, X1:X0)

; 由 Serial Control 腳位，使用 SPI_Like 傳輸協定，輸出 [X1:X0] 暫存器內容值的訊號。

例. NY3 Serial_Trigger 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(NY3, X1:X0)

; 由 Serial Control 腳位，使用 NY3 Serial_Trigger 傳輸協定，輸出 [X1:X0] 暫存器內容值的訊號。

若暫存器內容值為皆為 0 時，僅輸出 Reset 訊號。

例. IR_Trigger 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(IR_Trigger, X1:X0)

; 由 Serial Control 腳位，使用 IR_Trigger 傳輸協定，輸出 [X1:X0] 暫存器內容值的訊號。

5.22 串列資料接收指令（Serial Control RX Command）

Serial Control RX Command				
SC_RX_ON	SC_RX_OFF	[Rl,Rk,Rj,Ri] = SC_RX	[Xj,Xi] = SC_RX	SC_RX = data?Path
SC_RX != data?Path		-	-	--

5.22.1 SC_RX_ON

[NY4 / NY5 / NY5+ / NY6 / NY7]

開啟串列資料接收功能。若是沒有開啟該功能，則串列資料無法接收。

***例.* SC_RX_ON** **; 開啟串列資料接收功能。**

5.22.2 SC_RX_OFF

[NY4 / NY5 / NY5+ / NY6 / NY7]

關閉串列資料接收功能。

***例.* SC_RX_OFF** **; 關閉串列資料接收功能。**

5.22.3 [Rl, Rk, Rj, Ri] = SC_RX

[NY4 / NY5 / NY5+ / NY6 / NY7]

使用者可透過指令將接收到的串列資料存在指定的 RAM，Ri=bit0~3，Rj=bit4~7，Rk=bit8~11，Rl=bit12~15。

注意：在未收到串列資料前，使用此指令會讀取到未知資料。

例.

```
[R0] = SC_RX                    ; 將接收到資料的 Bit[3:0]存到 R0。
[R1, R0] = SC_RX                ; 將接收到資料的 Bit[3:0]存到 R0，Bit[7:4]存到 R1。
[R2, R1, R0] = SC_RX            ; 將接收到的資料由低到高依序存入 R0、R1 及 R2。
```

5.22.4 [Xj, Xi] = SC_RX

[NY4 / NY5 / NY5+ / NY6 / NY7]

使用者可透過指令將接收到的串列資料存在指定的 RAM，Xi=bit0~7，Xj=bit8~15。

注意：在未收到串列資料前，使用此指令會讀取到未知資料。

例.

```
[X0] = SC_RX                    ; 將接收到資料的 Bit[7:0]存到 X0。
[X1, X0] = SC_RX                ; 將接收到資料的 Bit[7:0]存到 X0，Bit[15:8]存到 X1。
```

5.22.5 SC_RX = data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7]

當接收到的串列資料等於 data 值時，跳到 Path。

例. **SC_RX = 0? Path** ; 當接收到的代碼為 0 時，跳到 Path。

5.22.6 SC_RX != data?Path

[NY4 / NY5 / NY5+ / NY6 / NY7]

當接收到的串列資料不等於 data 值時，跳到 Path。

例. **SC_RX != 0? Path** ; 當接收到的資料不為 0 時，跳到 Path。

5.23 I2C 指令（I2C Command）

I2C Command				
I2C_TX	I2C_RX	I2C_RX = data?Path		
I2C_RX != data?Path		I2C_Ack?Path	I2C_Start	I2C_Stop
I2C_MReadAck	I2C_MReadNAck	I2C_Reset	-	-

5.23.1 I2C_TX

[NY5+ / NX1]

傳輸 I2C 訊號。

I2C_TX(Data)
I2C_TX(Rj:Ri)
I2C_TX(Xi)

Data：欲傳送的立即值。
Rj:Ri：使用 RAM 的方式來傳輸 I2C 訊號。Ri=bit0 ~ 3，Rj=bit4 ~ 7。
Xi：使用 RAM 的方式來傳輸 I2C 訊號。Xi=bit0 ~ 7。

5.23.2 I2C_RX

[NY5+ / NX1]

使用者可透過指令將接收到的 I2C 資料存在指定的 RAM。

[Rj, Ri] = I2C_RX
[Xi] = I2C_RX
Var = I2C_RX
Rj, Ri：將接收的 I2C 資料存放至 RAM，Ri=bit0 ~ 3，Rj=bit4 ~ 7。
Xi：將接收的 I2C 資料存放至 RAM，Xi=bit0 ~ 7。
Var：將接收的 I2C 資料存放至 RAM。

5.23.3 I2C_RX = data?Path

[NY5+ / NX1]

當接收到的 I2C 資料等於 data 值時，跳到 Path。

5.23.4 I2C_RX != data?Path

[NY5+ / NX1]

當接收到的 I2C 資料不等於 data 值時，跳到 Path。

5.23.5 I2C_Ack?Path

[NY5+ / NX1]

當主控時，資料傳送完畢，判斷接收端是否有回應 ACK 訊號，若有則跳至 Path。

注意：只有在 I2C 為 Master 模式才可使用此指示。

5.23.6 I2C_Start

[NY5+ / NX1]

當主控時，傳送裝置位址，並開始由被動裝置讀資料或寫資料至被動裝置。

I2C_Start

I2C_Start(Address, Mode)

I2C_Start(Address, Bits, Mode)

Address：裝置位址。

- data（支援 0 ~ 0x3FF）
- Ri（支援 1 ~ 2 個 Ri）
- Xi（支援 1 個 Xi）

Bits：可為 7 / 10，預設為 7。

Mode：傳輸模式。可為 R / W。

注意：只有在 I2C 為 Master 模式才可使用此指示。

5.23.7 I2C_Stop

[NY5+ / NX1]

當主控時，傳送結束訊號，通知裝置通訊結束。

注意：只有在 I2C 為 Master 模式才可使用此指示。

5.23.8 I2C_MReadAck

[NY5+ / NX1]

主控輸出 clock 以接收裝置回傳資料，接收完成回應 ACK 訊號。

注意：只有在 I2C 為 Master 模式才可使用此指示。

5.23.9 I2C_MReadNAck

[NY5+ / NX1]

主控輸出 clock 以接收裝置回傳資料，接收完成不回應 ACK 訊號。

注意：只有在 I2C 為 Master 模式才可使用此指示。

5.23.10 I2C_Reset

[NX1]

重設 I2C。

注意：只有在 I2C 為 Master 模式才可使用此指示。

5.24 UART 指令（UART Command）

UART Command			
UART_TX	UART_TX_WaitN	UART_RX	UART_RX=data?Path
UART_RX != data?Path		UART_TX_Busy?Path	-

5.24.1 UART_TX

[NY5+ / NX1]

傳輸 UART 訊號。

UART_TX(Data)
UART_TX(Rj:Ri)
UART_TX(Xi)

Data：欲傳送的立即值。
Rj:Ri：使用 RAM 的方式來傳輸 UART 訊號。Ri=bit0 ~ 3，Rj=bit4 ~ 7。
Xi：使用 RAM 的方式來傳輸 UART 訊號。Xi=bit0 ~ 7。

5.24.2 UART_TX_WaitN

[NY5+ / NX1 EF]

若有 UART 資料正在傳送中，則等待至資料傳送完畢後，再執行下一個指令。UART 資料沒有傳送則立即執行下一個指令。

5.24.3 UART_RX

[NY5+ / NX1]

[Rj, Ri] = UART_RX
Xi = UART_RX
使用者可透過指令將接收到的 UART 資料存在指定的 RAM，Ri=bit0 ~ 3，Rj=bit4 ~ 7，Xi=bit0 ~ 7。

5.24.4 UART_RX = data?Path

[NY5+ / NX1]

當接收到的 UART 資料等於 Data 值時，跳到 Path。

5.24.5 UART_RX != data?Path

[NY5+ / NX1]

當接收到的 UART 資料不等於 Data 值時，跳到 Path。

5.24.6 UART_TX_Busy?Path

[NX1 EF]

當 UART TX 正在進行中，則跳到 Path。

5.25 脈衝調變 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	PlayPWM	PlayPWMS	WaitPN
StopPWM	PausePWM	ResumePWM	HoldPWM	PWMCtrl
PWMDuty	-	-	-	-

5.25.1 PWMOut / PWMOutS

5.25.1.1 PWMOut / PWMOutS (Pin, Duty, Time, Type)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

Q-Code 提供使用者簡易的指令來達成 PWM-IO 輸出的方式。指令格式如下：

PWMOut：待 PWM-IO 輸出的時間結束後，才執行下一個指令。

PWMOutS：PWM-IO 輸出後，立即執行下一個指令。

Pin：可指定任一輸出腳。

- **Px.n**：IC 腳位。
- **EXPx_Py.n**：I/O 擴展晶片的腳位。

Duty：可使用立即值指定輸出百分比及階數或使用變數指定階數。

- 使用立即值輸出階數 — *例.* **PWMOut(PC.0, 12, 1s)**
- 使用立即值輸出百分比 — *例.* **PWMOut(PC.0, 30%, 1s)**
- 使用 Ri 輸出階數(固定使用 2 個 Ri) — *例.* **PWMOut(PC.0, R1:R0, 1s)**
- 使用 Xi 輸出階數(固定使用 1 個 Xi) — *例.* **PWMOut(PC.0, X0, 1s)**

Time：可以設定 PWM-IO 輸出的時間，不填寫的話預設為 16ms，若指定 loop 則持續輸出不停止。（時間=16ms~30sec）

Type：選擇使用軟體(0)或硬體(1) PWM 輸出功能，預設為軟體(0)。

注意：NY4 / NY5 / NY6 / NY7 / NX1 不支援 Type 參數。

5.25.1.2 PWMOut / PWMOutS (Para8, Para7, Para6, Para5, Para4, Para3, Para2, Para1)

[NY9T]

PWMOut 指令是用來設定全部 PWM-IO 腳位的 Duty cycle，不需要經過 PlayPWM 指令即可直接輸出。

Para1~8：代表各個 pin 編號，Para1=PE.0，Para2=PE.1，依此類推。（只有設定成 PWM-IO 的腳位才能設定）。

- 直接設定輸出的百分比（0%~100%） *例.* **PWMOut(30%, 50%, 70%, 5%)**
- 直接設定輸出的階數（0~255） *例.* **PWMOut(30, 50, 70, 5)**

Time：可以設定 PWM-IO 輸出的時間，不填寫的話預設為 16ms。（時間=16ms~30sec）

注意：NY9T 使用兩個 PWM-IO 輸出指令，必須要間隔 16ms 以上才不會發生動作不正確的情況。

5.25.1.3 PWMOut / PWMOutS (@PWMEn, Percentage, Time)

[NY9T]

PWMEn：腳位及檔案對應的方式是依照 **[I/O]** 段落中所設定的 PWM-IO 輸出腳位順序及 Action 檔案中的 Label “A1~A8” 的順序，最多對應至 PE.0~PF.3，共 8 個 pin。PWMEn=1 or 0，1=該腳位在播放 Action 檔案時，開啟 PWM-IO 輸出功能；0=不啟動。（未填的部份會自動補 0，高位在最左邊）

- 直接設定要輸出的腳位 *例.* **PWMOut(@1111, 30%)**
- Ri（支援到 2 個 Ri） *例.* **PWMOut(R1:R0, 30%)**
- Xi（支援到 1 個 Xi） *例.* **PWMOut(X0, 30%)**
- 填寫 X 的話，會維持上一次的設定。 *例.* **PWMOut(X, 30%)**

Percentage：設定輸出的百分比或階數。

- 直接設定輸出的百分比（0%~100%） *例.* **PWMOut(@1111, 30%)**
- 直接設定輸出的階數（0~255） *例.* **PWMOut(@1111, 127)**
- Ri（支援到 2 個 Ri） *例.* **PWMOut(@1111, Rj:Ri)**
- Xi（支援到 1 個 Xi） *例.* **PWMOut(@1111, Xi)**

Time：可以設定 PWM-IO 輸出的時間，不填寫的話預設為 16ms。（時間=16ms~30sec）

例.

[Path]

```
TR1: PWMOut(@1111, 30%)                    ; PE.0~PE.3 輸出 30%的亮度。
TR2: PWMOut(@1111, 30)                     ; PE.0~PE.3 輸出 30/256 的亮度。
TR3: R1=0x1, R0=0xE, PWMOut(@1111, R1:R0) ; PE.0~PE.3 輸出 30/256 的亮度。
TR4: X0=0x1E, PWMOut(@1111,X0)             ; PE.0~PE.3 輸出 30/256 的亮度。
TR5: PWMOut(@1111, 30%, 16ms)              ; PE.0~PE.3 輸出 30%亮度及播放 16 毫秒的時間。
TR6: PWMOut(@1111, 30, 2s)                 ; PE.0~PE.3 輸出 30/256 的亮度及 2 秒的時間。
```

注意：

1. ***NY9T 使用兩個 PWM-IO 輸出指令，必須要間隔 16ms 以上才不會發生動作不正確的情況。***
2. ***若 Percentage 參數使用暫存器來決定的話，將會佔用 DATA ROM 來存放 PWM-IO 訊號 (DATA Size 計算方式 = (pin number+1)*256)。***

5.25.2 PlayPWM / PlayPWMS

5.25.2.1 PlayPWM($PIO) / PlayPWMS($PIO)

[NY5+]

使用者可在 **[PWM-IO]** 段落中設定要使用 PWM-IO 輸出的訊號組合，透過此指令來將信號播放出來。

PlayPWM({Px.n, }Ch, $PIO)

PlayPWM({Px.n, }Ch, $PIO, Loop)

PlayPWMS({Px.n, }Ch, $PIO)

PlayPWMS({Px.n, }Ch, $PIO, Loop)

PlayPWM：待 Action 檔案播放結束後，執行下一個指令。

PlayPWMS：Action 檔案開始播放，立即執行下一個指令。

Px.n：輸出腳位。

- NY5+支援 PB.0 ~ PB.3 及 PD.0 ~ PD.3。

Ch：指定要播放的 PWM-IO 通道。

- NY5+支援 Ch1 ~ Ch4。

PIO：**[PWM-IO]** 中指定的輸出組態。

- 直接指定代號(SPIOx 及 MPIOx)。
- Ri（可支援到 1~3 個 Ri）或是 Xi（可支援到 1~2 個 Xi），由 Ri / Xi 的值決定要播放的是第幾個 PWM-IO 檔案。若是 Ri / Xi 帶有"++"則在播放後將其遞增。若使用 2 個 Xi，則高 4 位元無效。

Loop：讓此 Action 訊號循環播放，直到有其他的指令來停止它，如 StopPWM(若不使用此功能則不需寫出此參數)。

注意：

1. NY9T 中，兩個 PWM-IO 輸出指令，必須要間隔 16ms 以上才不會發生動作不正確的情況。

2. NY5+中，當指定 Px.n 時，僅能使用 [PWM-IO] 中的單一訊號輸出；未指定 Px.n 時，則只能使用 [PWM-IO] 中的多訊號輸出。

例. NY5+

```
[Action File]
VIO0 = Action1.vio                    ; Action 檔中，含有 ActionLabel "A1~A6" 的訊號。
[PWM-IO]
Multi_Pins = [PB.0, PB.1, PB.2, PB.3, PD.0, PD.1, PD.2, PD.3]
MPIO1 = [VIO0.A1, VIO0.A2, VIO0.A3, VIO0.A4]
MPIO2 = [X, X, X, X, VIO0.A4, VIO0.A3, VIO0.A2, VIO0.A1]
SPIO1 = [VIO0.A6]
[Path]
Path1: PB=[P P P P], PlayPWM(Ch1, $MPIO1) ; PB.0 輸出 VIO0.A1，PB.1 輸出 VIO0.A2，以此類推。
```

Path2: PD=[P P P P], PlayPWM(Ch1, $MPIO2) ; PD.0 輸出 VIO0.A4，PD.1 輸出 VIO0.A3，以此類推。

Path3: PlayPWM(PB.0, Ch1, $SPIO1) ; PB.0 輸出 VIO0.A6。

5.25.2.2 PlayPWM(@PWMEn) / PlayPWMS(@PWMEn)

[NY9T]

使用者可由 *Q-Visio* 來繪製要輸出的 Action 信號，在 *Q-Code* 中可使用 PWM-IO 指令來將信號播放出來。

PlayPWM(@PWMEn, $VIOLabel)

PlayPWM(@PWMEn, $VIOLabel, Extension)

PlayPWMS(@PWMEn, $VIOLabel)

PlayPWMS(@PWMEn, $VIOLabel, Extension)

PlayPWM：待 Action 檔案播放結束後，執行下一個指令。

PlayPWMS：Action 檔案開始播放，立即執行下一個指令。

PWMEn：腳位及檔案對應的方式是依照 **[I/O]** 段落中所設定的 PWM-IO 輸出腳位順序及 Action 檔案中的 Label "A1 ~ A8" 的順序，最多對應至 PE.0 ~ PF.3，共 8 個 pin。PWMEn=1 or 0，1=該腳位在播放 Action 檔案時，開啟 PWM-IO 輸出功能；0=不啟動。(未填的部份會自動補 0，高位在最左邊)

- 直接設定要輸出的腳位 *例.* **PlayPWM(@1111, $VIO0, 4)**
- Ri（支援 1 ~ 2 個 Ri） *例.* **PlayPWM(R1:R0, $VIO0, 4)**
- Xi（支援到 1 個 Xi） *例.* **PlayPWM(X0, $VIO0, 4)**
- 填寫 X 的話，會維持上一次的設定 *例.* **PlayPWM(X, $VIO0, 4)**

VIOLabel：加入多個 VIO 檔時，須指定所要引用的 Action File 中的哪一組。

- 直接設定要輸出的檔案 *例.* **PlayPWM(@1111, $VIO0)**
- Ri（支援 1 ~ 4 個 Ri） *例.* **PlayPWM(@1111, R3:R2:R1:R0)**
- Xi（支援到 1 ~ 2 個 Xi） *例.* **PlayPWM(@1111, X1:X0)**

Extension：可將該訊號整個依照倍數來延長，以節省 ROM Size。Extension=1/2/4/8，1 倍可省略不寫。

- 直接賦值。 *例.* **PlayPWM(@1111, $VIO0, 4)**
- Ri（支援到 1 個 Ri） *例.* **PlayPWM(@1111, $VIO0, R0)**
- 填寫 X 的話，會維持上一次的設定 *例.* **PlayPWM(@1111, $VIO0, X)**

注意：

1. ***兩個 PWM-IO 輸出指令，必須要間隔 16ms 以上才不會發生動作不正確的情況。***
2. ***PWMEn 參數的最左邊對應有設定成 PWM-IO 輸出腳位的最高位；Action 檔案則是 A1 對應到最低位。***
3. ***NY9T 中，Action 訊號會依照 PWM-IO 開啟的數量來對應，A1 會以 PE.0 為優先對應腳位，若 PE.0 未被設定成 PWM-IO，則對應至 PE.1，依此類推。***
4. ***若是播放 VIO 檔，且同時輸出多個訊號時，須注意各訊號的輸出時間長度必須相等，若時間長度不***

相等時，則會以輸出時間最長的訊號為基準，並且將其餘訊號輸出時間不足的部份強制輸出 0%。

5. 若使用 Ri 指定 Extension 參數時，Ri 的內容值必須為 1/2/4/8，非這四個數值時，皆會變成 1 倍。

例. NY9T

```
[Action File]
VIO0 = Action1.vio                    ; Action 檔中，含有 ActionLabel “A1~A6”的訊號。
VIO1 = Action2.vio                    ; Action 檔中，含有 ActionLabel “A1~A6”的訊號。

[Path]
Path1: PlayPWM(@1100, $VIO0)          ; 播放“VIO0”，輸出到 PE.2, PE.3。
Path2: PlayPWM(@0011, $VIO0)          ; 播放“VIO0”，輸出到 PE.0, P.E1。
Path3: PlayPWM(@1111, $VIO1)          ; 播放“VIO1”，輸出到 PE.0, PE.1, PE.2, PE.3。
Path4: PlayPWM(@1111, $VIO1, 4)       ; VIO1 訊號延長 4 倍來播放，輸出到 PE.0, PE.1, PE.2,
                                      PE.3。
Path5: PlayPWM(@1111, $VIO1), PlayA…  ; 播放“VIO1”完，立即執行 PlayA 指令。
Path6: PlayPWMS(@1111, $VIO1), PlayA… ; 播放“VIO1”後，立即執行 PlayA 指令。
```

5.25.3 WaitPN

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

等待 PWM-IO 輸出完畢後，再執行下一個指令，沒有 PWM-IO 正在執行則立即執行下一個指令。

WaitPN

WaitPN(Pin)

WaitPN(Ch)

WaitPN(PIO)

Pin：指示要等待 PWMOut / PWMOutS 輸出的腳位。

- **Px.n**：IC 腳位。
- **EXPx_Py.n**：I/O 擴展晶片的腳位。

Ch：PlayPWM / PlayPWMS 所輸出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的輸出狀態。

注意：

1. NY4 / NY5 / NY6 / NY7 / NY9T 不支援 Pin / Ch / PIO 參數。

2. NX1 不支援 Ch / PIO 參數。

5.25.4 StopPWM

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

用來停止 PWM-IO 輸出。

StopPWM

StopPWM(Pin)

StopPWM(Ch)

StopPWM(PIO)

Pin：指定要停止 PWMOut / PWMOutS 輸出的腳位。

- **Px.n**：IC 腳位。
- **EXPx_Py.n**：I/O 擴展晶片的腳位。

Ch：PlayPWM / PlayPWMS 所輸出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的輸出狀態。

注意：

1. NY4 / NY5 / NY6 / NY7 / NY9T 不支援 Pin / Ch / PIO 參數。

2. NX1 不支援 Ch / PIO 參數。

5.25.5 PausePWM

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

暫停目前的 PWM-IO 輸出。

- NY9T，PausePWM 對所有的 PWM-IO 都是暫停輸出，IC 可進入睡眠。
- NY4 / NY5 / NY6 / NY7，不支援 PlayPWM，PausePWM 會暫停 PWM 計時，維持當下 duty 持續輸出，IC 不進入睡眠。
- NY5+
  - PausePWM 對 PWMOut 的輸出，會暫停 PWM 計時，維持當下 duty 持續輸出，IC 不進入睡眠。
  - PausePWM 對 PlayPWM 的輸出，暫停輸出，IC 可進入睡眠。

PausePWM

PausePWM(Pin)

PausePWM(Ch)

PausePWM(PIO)

Pin：指定要暫停 PWMOut / PWMOutS 輸出的腳位。

- **Px.n**：IC 腳位。
- **EXPx_Py.n**：I/O 擴展晶片的腳位。

Ch：PlayPWM / PlayPWMS 所輸出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的輸出狀態。

注意：

1. 如果在沒有 PWM-IO 輸出的情況下，此動作會被忽略。

2. NY4 / NY5 / NY6 / NY7 / NY9T 不支援 Pin / Ch / PIO 參數。

3. NX1 不支援 Ch / PIO 參數。

5.25.6 ResumePWM

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

用來恢復 PWM-IO 輸出，繼續計數時間。

ResumePWM

ResumePWM(Pin)

ResumePWM(Ch)

ResumePWM(PIO)

Pin：指定要恢復 PWMOut / PWMOutS 輸出的腳位。

- **Px.n**：IC 腳位。
- **EXPx_Py.n**：I/O 擴展晶片的腳位。

Ch：PlayPWM / PlayPWMS 所輸出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的輸出狀態。

注意：

1. NY4 / NY5 / NY6 / NY7 / NY9T 不支援 Pin / Ch / PIO 參數。

2. NX1 不支援 Ch / PIO 參數。

5.25.7 HoldPWM

[NY5+ / NY9T]

暫停目前的 PWM-IO 輸出，會維持目前的 duty 輸出，但不進入 Sleep 模式。

- NY9T，HoldPWM 可暫停 PWMOut / PlayPWM 的 PWM-IO 輸出。
- NY5+，HoldPWM 只可暫停 PlayPWM 的 PWM-IO 輸出，對 PWMOut 無作用。

HoldPWM

HoldPWM(Ch)

HoldPWM(PIO)

Ch：PlayPWM / PlayPWMS 所輸出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的輸出狀態。

注意：

1. NY9T 不支援 Ch 及 PIO 參數。

2. NY5+的 HoldPWM 對 PWMOut 無效

5.25.8 PWMCtrl (@PWMEn, Extension)

[NY9T]

可以開關在 **[I/O]** 段落中已被設定的 PWM-IO 腳位。PWMEn 的使用方式同 PlayPWM 指令，請參考 5.25.2。

例. 在播放 PWM-IO（PE.0~PE.3）的過程中，可以關閉或打開 PWM-IO 的輸出。

[Path]

```
PowerOn: PWMCtrl(@1111,1)          ; PWMIO 預設為 1111，Extension=1。
TR1: PlayPWM(X, $VIO0)             ; 播放"VIO0"，輸出到 PE。
TR2: PWMCtrl(@0011,X)              ; 關閉 PE.2, PE.3 的輸出。
```

TR3: PWMCtrl(X, 4) ；改變 Extension 為 4 倍。

5.25.9 PWMDuty

[NY5+ / NX1]

修改腳位上的 PWM-IO 輸出。

PWMDuty(Pin, Duty)

Pin：指定要修改 PWM-IO 輸出的腳位。

- **Px.n**：IC 腳位。
- **EXPx_Py.n**：I/O 擴展晶片的腳位。

Duty：可使用立即值指定輸出百分比及階數或使用變數指定階數。

- 使用立即值輸出階數
- 使用立即值輸出百分比
- 使用 Ri 輸出階數(固定使用 2 個 Ri)
- 使用 Xi 輸出階數(固定使用 1 個 Xi)

注意：PWMDuty 只有在 PWM-IO 進行中才有效。

例.

PWMDuty(PB.0, 50%)

PWMDuty(PB.0, 8)

PWMDuty(PB.0, R1:R0)

PWMDuty(PB.0, X0)

PWMDuty(EXP0_P0.3, X0)

5.26 中斷指令（Interrupt Command）

Interrupt Command				
INT_ON	INT_OFF	INT_RET	INT = n	-

5.26.1 INT_ON

[NY5 / NY5+ / NX1]

開啟時間中斷。

注意：

1. 即便是中斷打開後，若是無播放指令或延遲指令在執行，則系統會自動進入睡眠模式。

2. NX1 于 PowerOn 時，預設為 INT_ON，使用者無需再呼叫。

3. NX1 的 INT_ON 與 INT_OFF 必須配對使用。

例. NY5 / NY5+

```
[Path]
PowerOn: KEY1, INT_ON                    ; 開啟時間中斷。
```

例. NX1

```
[Variable]
Var8: count500ms = 0
[Path]
PowerOn: KEY1, INT_ON
500ms: INT_OFF, count500ms++, INT_ON     ; 修改 count500ms 前，將中斷關閉，避免同時修改變
                                         數造成錯誤。
[Interrupt]
RTC_2Hz: count500ms++
```

5.26.2 INT_OFF

[NY5 / NY5+ / NX1]

關閉時間中斷。

注意：

1. NX1 的 INT_OFF 與 INT_ON 必須配對使用。

2. NX1 處理器中斷程式請勿關閉過久，避免影響系統層級的運作。

例.

```
[Path]
PowerOn: KEY1, INT_OFF                   ; 關閉時間中斷。
```

5.26.3 INT_RET

[NY5 / NY5+]

返回中斷向量。

注意：有使用跳躍指令後，一定要使用 INT_RET 來做返回中斷向量的動作，否則將會造成動作不正常。

例.

```
[Path]
PowerOn: KEY1, INT_ON, INT=1.024
Interrupt: R0=0xF?Change, PA=0x0,        ; R0 的值若是等於 0xF，且會跳躍到“Change”，
           INT_RET                       ; 否則執行下一個命令。
Change: PA=0xF, INT_RET                  ; 執行完 PA=0xF 後，當執行 INT_RET 時，會返回中
                                         ; 斷向量。
```

5.26.4 INT = n

[NY5 / NY5+]

設定時間中斷間距。n = 0.256 / 0.512 / 1.024，單位 ms。

例.

[Path]

PowerOn: KEY1, INT_ON, INT=1.024

Interrupt: !PA, INT_RET **；每 1ms 發生時間中斷一次，且將 PA 反相輸出。**

5.27 時間延遲指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

5.27.1 Delay(time)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

在 *Q-Code* 提供 3 組的 delay，在前景與兩個背景分別提供一組。

time 單位：ms（毫秒）或 sec（秒）（預設為秒），最小值=4ms。未滿 4ms，無條件進位。

- NY4 / NY5 / NY6 / NY7 / NY9T 最大值=15s。
- NY5+ / NX1 最大可至 0x7FFFFFFF 毫秒。

注意：

1. NX1 於 Q-Code 6.51(含)以前存在-2.34%計時誤差，源自於 RTC_1024Hz 與 1ms 的時間差異，(0.9765625ms - 1ms) / 1ms ≒ -2.34%。Q-Code 6.60(含)之後透過軟體進行補償修正。

2. NX1 以 RTC 進行計時模擬，存在 I_LRC 的±1.5%計時誤差範圍。

3. NX1 採用分時多工系統架構，注意 1,注意 2 計時誤差不包含多工系統的任務切換耗時。

例. 要執行一個 0.1 sec 的延遲。

[Path]

PowerOn: Delay(0.1) **；延遲 0.1 秒。**

或

PowerOn: Delay(100ms) **；延遲 100 毫秒。**

5.27.2 Delay(Rk:Rj:Ri)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

根據 RAM 的內容值來設定 Delay 時間。Ri=低位元，Rj=中位元，Rk=高位元。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 時間是以 4.096ms 為最小單位。
- NX1 時間最小單位為 1ms。

例. NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 中，時間延遲 1 sec，如何計算 Ri，Rj，Rk 要填入的值。

1000 ms / 4.096 ms = 244.1（無條件進位），將 245 轉換成 Hex，則等於 0xF5。(計時器的最小單位為 4.096ms）

```
TR1: R2=0x0, R1=0xF, R0=0x5, Delay(R2:R1:R0)  ; 延遲 1 sec。
```

例. NX1 中，時間延遲 1 sec，直接填入 1000 即可。

```
[Variable]
Var16: Delay_Time
[Path]
TR1: Delay_Time=1000, Delay(Delay_Time)        ; 延遲 1 sec。
```

5.27.3 WaitDN(n)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

若有 Delay 正在執行中，則 Delay 執行完畢後，再執行下一個指令。沒有 Delay 執行中則立即執行下一個指令。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，則等待所有的 delay 執行完畢。

例.

```
[Path]
TR1: Delay(5)
TR2: WaitDN(1), Out                    ; 若背景 1 的正在執行時間延遲，則時間延遲執行
                                       ; 完後返回執行 Out 指令。

[Background1]
BG1: Delay(5)                          ; BG1 執行 5 秒時間延遲。
```

5.27.4 StopD(n)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

停止 Delay。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，則停止所有的 Delay。

例.

```
[Path]
TR1: BG1                               ; 呼叫 BG1。
TR2: StopD(1)                          ; 停止 BG1 的時間延遲。

[Background1]
BG1: Delay(5)                          ; BG1 執行 5 秒時間延遲。
```

例.

```
[Path]
PowerOn: BG1, Delay(3)                 ; 前景延遲 3 秒。
```

```
TR1: StopD                          ; 停止前景和 BG1 的時間延遲。

[Background1]
BG1: Delay(5)                       ; BG1 執行 5 秒時間延遲。
```

5.27.5 PauseD(n)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

可以暫停指定的 Delay。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，則暫停所有的 Delay。

```
[Path]
TR1: BG1                            ; 呼叫 BG1。
TR2: PauseD(1)                      ; 暫停 BG1 的時間延遲。
TR3: ResumeD(1)                     ; 恢復 BG1 的時間延遲。

[Background1]
BG1: Delay(5)                       ; BG1 執行 5 秒時間延遲。
```

5.27.6 ResumeD(n)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

可以恢復指定的 Delay。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，則恢復所有的 Delay。

```
[Path]
TR1: BG1                            ; 呼叫 BG1。
TR2: PauseD(1)                      ; 暫停 BG1 的時間延遲。
TR3: ResumeD(1)                     ; 恢復 BG1 的時間延遲。

[Background1]
BG1: Delay(5)                       ; BG1 執行 5 秒時間延遲。
```

5.27.7 SDelay(time)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

SDelay 可用來設定短暫的延遲（Short Delay）。time 單位：毫秒或微秒（預設為毫秒）。

注意：

1. ***在延遲過程中無法處理其它的動作，僅能在延遲結束後執行下一個步驟。***
2. ***NY4 / NY5 / NY5+ / NY9T 中的時間範圍為 50us ~ 12500us，每次增加 50us。***
3. ***NY6 / NY7 中的時間範圍為 10us ~ 4000us，每次增加 10us。***
4. ***NX1 中的時間範圍為 1us ~ 4000us，每次增加 1us。***

例. 要執行一個 250us 的延遲。

```
DELAY_TEST: SDelay(250us)                    ; 延遲 250us。
```

5.28 動作指令（Action Command）

Action Command				
PlayA	PlayAS	WaitAN(Ch)	PauseA(Ch)	ResumeA(Ch)
HoldA(Ch)	StopA(Ch)	-	-	-

5.28.1 PlayA / PlayAS

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

使用者可由 *Q-Visio* 來繪製要輸出的 Action 信號，在 *Q-Code* 中可使用 Action 指令來將信號播放出來。

PlayA (Px.n, Ch, {VIOLabel.)Label{, Extension}{, Loop})

PlayAS (Px.n, Ch, {VIOLabel.)Label{, Extension}{, Loop})

PlayA(Ch, VIOLabel{, Extension}{, Loop})

PlayAS(Ch, VIOLabel{, Extension}{, Loop})

PlayA：待 Action 文件播放結束後，執行下一個指令。

PlayAS：Action 文件開始播放，立即執行下一個指令。

Px.n：可指定任一輸出腳。

Ch：Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支援 Ch1 ~ Ch8。
- NX1 支援 Ch1 ~ Ch32。

VIOLabel：指定欲播放的.vio 文件。

- 直接指定 Action File。 *例.* **PlayA(PE.0, Ch1, $VIO, 4)**
- 指定 **[Action]** 中所定義的多訊號組態。*例.* **PlayA(Ch1, $MVIO0, R0)**
- 指定 **[Action]** 中所定義的單訊號組態。*例.* **PlayA(PE.0, Ch1, $SVIO0, R0)**

Label：指定欲播放.vio 檔的單一訊號。

Extension：可將該訊號整個依照倍數來延長，以節省 ROM Size。Extension 的範圍為 1~15。

- 直接賦值。 *例.* **PlayA(PE.0, Ch1, $VIO.A1, 4)**
- Ri（支援到 1 個 Ri） *例.* **PlayA(PE.0, Ch1, $VIO.A1, R0)**

Loop：讓此 Action 訊號循環播放，直到有其他的指令來停止它，如 StopA(若不使用此功能則不需寫出此參數)。

注意：

1. NX1 不支援播放單一 VIO 檔案，僅能播放 VIO 檔案中的單一訊號。

2. *NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支援 Extension 參數。*
3. *若是播放 VIO 檔，且同時輸出多個訊號時，須注意各訊號的輸出時間長度必須相等，若時間長度不相等時，則會以輸出時間最長的訊號為基準，並且將其餘訊號輸出時間不足的部份強制輸出 0%。*
4. *由於 NY9T Action 計時器的解析度僅有 1ms，因此變化的範圍較小，所以不建議用來輸出 Ascend / Descend 訊號。*
5. *開啟 Loop 功能，且同時輸出多個訊號時，使用者需注意所有輸出的訊號時間長度皆為相同，這樣才能無接縫地輸出各訊號，避免造成訊號不連續的問題。*

例.

```
Path1: PlayA(PB.0, Ch4, $A1)                ; 播放 Label “A1” 於通道 4，輸出到 PB.0。
Path2: PlayAS(PB.1, Ch3, $EyeMotion)        ; 播放 Label “EyeMotion” 於通道 3，輸出到 PB.1。
Path3: PlayAS(PB.2, Ch2, $A3)               ; 播放 Label “A3” 於通道 2，輸出到 PB.2。
Path4: PlayA(PB.2, Ch2, $VIO1.A3)           ; 播放 VIO1 中的 A3 訊號於通道 2，輸出到 PB.2。
```

例. 使用 **[Action]** 輸出多筆 action 訊號。

```
[Action File]
VIO0 = Action1.vio                          ; 加入 action 檔案。
VIO1 = Action2.vio
[Output State]
Output_0: A   A   A   A   A   A   A   A ; 使用 PA 及 PB 作為 action 訊號輸出。
[Action]
Compression = Enable
FrameRate = 80
Multi_Pins = [PA.0, PA.1, PA.2, PA.3, PB.0, PB.1, PB.2, PB.3]      ; 使用 PA 及 PB 作為輸出腳位。
MVIO0 = [VIO0.A1, VIO0.A2, VIO0.A3, VIO0.A4, VIO0.A5, VIO0.A6, VIO0.A7, VIO0.A8]
                                            ; MVIO0 使用 VIO0 的 A1 ~ A8。
MVIO1 = [VIO1.A8, VIO1.A7, VIO1.A6, VIO1.A5, VIO1.A4, VIO1.A3, VIO1.A2, VIO1.A1]
                                            ; MVIO1 使用 VIO1 的 A8 ~ A1。
[Path]
TR1: Output_0, PlayA(Ch1, $MVIO0)           ; 使用 MVIO0 輸出 action 訊號至 PA 及 PB。
TR2: Output_0, PlayA(Ch1, $MVIO1)           ; 使用 MVIO0 輸出 action 訊號至 PA 及 PB。
```

5.28.2 WaitAN(Ch)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

若指定通道的 Action 正在播放中，等待 Action 播放完畢後，再執行下一個指令。沒有 Action 播放，則立即執行下一個指令。

Ch：選擇 Action 通道。不帶此參數則表示全部的 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支援 Ch1 ~ Ch8。
- NX1 支援 Ch1 ~ Ch32。

5.28.3 PauseA(Ch)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

暫停正在播放中的指定通道的 Action。

Ch：選擇 Action 通道。不帶此參數則表示全部 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支援 Ch1 ~ Ch8。
- NX1 支援 Ch1 ~ Ch32。

5.28.4 ResumeA(Ch)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

恢復播放指定通道的 Action。

Ch：選擇 Action 通道。不帶此參數則表示全部 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支援 Ch1 ~ Ch8。
- NX1 支援 Ch1 ~ Ch32。

5.28.5 HoldA(Ch)

[NY9T]

暫停正在播放中的 Action，並維持目前的輸出狀態。

Ch：選擇 Action 通道，Ch=Ch1 ~ Ch8。不帶此參數則表示全部 Action 通道。

5.28.6 StopA(Ch)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

停止選擇的 Action 通道。

Ch：選擇 Action 通道。不帶此參數則表示全部 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支援 Ch1 ~ Ch8。
- NX1 支援 Ch1 ~ Ch32。

例.

[Path]

```
TR1: PlayA(PB.0, Ch1, $A0)          ; 播放$A0 於 Action 通道 1，且輸出到 PB.0。
TR2: PauseA(Ch1)                    ; 暫停執行 Action 通道 1 的動作。
TR3: Resume(Ch1)                    ; 恢復執行 Action 通道 1 的動作。
TR4: StopA(Ch1)                     ; 停止執行 Action 通道 1 的動作。
TR5: WaitAN(Ch1), Out               ; 若 Action 通道 1 正在執行，Action 通道 1 執行
                                    ; 完後返回執行 Out 指令。
```

5.29 LED Strip 指令（LED Strip Command）

LED Strip Command				
LEDStr_Play	LEDStr_PlayS	LEDStr_Stop	LEDStr_Clear	LEDStr_Brightness
LEDSync_Play	LEDSync_PlayS	LEDSync_Stop	LEDSync_Clear	LEDSync_Brightness
LEDText_Play	LEDText_PlayS	LESText_Stop	LEDText_Clear	LEDText_Brightness

5.29.1 LEDStr_Play / LEDStr_PlayS

[NX1]

Single-String 燈串播放指令，播放燈串檔案。

LEDStr_Play(Px.n, LEDLabel)

LEDStr_Play(Px.n, index)

LEDStr_Play(Px.n, index, Storage)

LEDStr_PlayS(Px.n, LEDLabel)

LEDStr_PlayS(Px.n, index)

LEDStr_PlayS(Px.n, index, Storage)

LEDStr_Play：待燈串播放結束後，執行下一個指令。

LEDStr_PlayS：燈串開始播放，立即執行下一個指令。

Px.n：輸出腳位。

LEDLabel：指定要播放的 Single-String LED 文件。

Index：播放檔案的索引值，可為立即值或變數。

Storage：LED 文件位置。支援 SPI0 / SPI1，不指定則為內部 ROM。

例.

```
[LED Strip]
Timer = PWMA
Period = 50ms
Single_LED_Order = RGB
Single_Data0 = H300_L900
Single_Data1 = H900_L300
Single_LED_Count = [PA.2: 100]
LED0 = Strip.csv /Color_Order=RGB /Format=Single ; 加入 LED Strip 檔案(.csv)。
[Path]
TR1: LEDStr_PlayS(PA.2, $LED0)                ; 播放 LED0 的資料，輸出至 PA.2。
TR2: SpiPlayS(ch,0), LEDStr_Play(PA.2, 0)     ; 播放第 0 首 Speech 和第 0 個燈串。
```

5.29.2 LEDStr_Stop

[NX1]

燈串播放停止指令，停止播放中的 Single-String 燈串。

LEDStr_Stop

LEDStr_Stop(Px.n)

Px.n：指定欲停止 Single-String 燈串輸出的腳位，不指定則停止所有播放中的 Single-String 燈串。

例.

```
TR1: LEDStr_Stop                    ; 停止播放燈串。
```

5.29.3 LEDStr_Clear

[NX1]

燈串清除指令，點滅 Single-String 燈串上所有 LED。

LEDStr_Clear

LEDStr_Clear(Px.n)

Px.n：指定欲清除 Single-String 燈串輸出的腳位，不指定則清除所有 Single-String 燈串。

例.

```
TR1: LEDStr_Clear                   ; 點滅燈串。
```

5.29.4 LEDStr_Brightness

[NX1]

燈串亮度設定指令，調整 Single-String 燈串亮度，n=0 ~ 15。

例.

```
TR1: LEDStr_Brightness(15)          ; 燈串設定最亮。
```

5.29.5 LEDSync_Play / LEDSync_PlayS

[NX1]

Sync-Play 燈串播放指令，播放燈串檔案。

LEDSync_Play(LEDLabel)

LEDSync_Play(index)

LEDSync_Play(index, Storage)

LEDSync_PlayS(LEDLabel)

LEDSync_PlayS(index)

LEDSync_PlayS(index, Storage)

LEDSync_Play：待燈串播放結束後，執行下一個指令。

LEDSync_PlayS：燈串開始播放，立即執行下一個指令。

LEDLabel：指定要播放的 Sync-Play LED 文件。

Index：播放檔案的索引值，可為立即值或變數。

Storage：LED 文件位置。支援 SPI0 / SPI1，不指定則為內部 ROM。

例.

```
[LED Strip]
Timer = PWMB
Period = 50ms
Sync_LED_Order = RGB
Sync_Data0 = H300_L900
Sync_Data1 = H900_L300
Sync_LEDCount = [PA.0:7, PA.1:10]
LED0 = Sync.csv /Color_Order=RGB /Format=Sync     ; 加入 LED Strip 檔案(.csv)。
[Path]
TR1: LEDSync_PlayS($LED0)                         ; 播放 LED0 的資料，輸出至 PA.2。
TR2: SpiPlayS(ch,0), LEDSync_Play(0)              ; 播放第 0 首 Speech 和第 0 個燈串。
```

5.29.6 LEDSync_Stop

[NX1]

燈串播放停止指令，停止播放中的 Sync-Play 燈串。

LEDSync_Stop

例.

```
TR1: LedStr_Stop                                  ; 停止播放燈串。
```

5.29.7 LEDSync_Clear

[NX1]

燈串清除指令，點滅 Sync-Play 燈串上所有 LED。

LEDSync_Clear

例.

```
TR1: LEDSync_Clear                                ; 點滅燈串。
```

5.29.8 LEDSync_Brightness

[NX1]

Sync-Play 燈串亮度設定指令，調整燈串亮度，n=0 ~ 15。

例.

```
TR1: LEDSync_Brightness(15)                       ; 燈串設定最亮。
```

5.29.9 LEDText_Play / LEDText_PlayS

[NX1]

Scrolling-Text 燈串播放指令，播放燈串檔案。

LEDText_Play(LEDLabel)

LEDText_Play(index)

LEDText_Play(index, Storage)

LEDText_PlayS(LEDLabel)

LEDText_PlayS(index)

LEDText_PlayS(index, Storage)

LEDText_Play：待燈串播放結束後，執行下一個指令。

LEDText_PlayS：燈串開始播放，立即執行下一個指令。

LEDLabel：指定要播放的 Scrolling-Text LED 文件。

Index：播放檔案的索引值，可為立即值或變數。

Storage：LED 文件位置。支援 SPI0 / SPI1，不指定則為內部 ROM。

例.

```
[LED Strip]
Timer = PWMB
Period = 50ms
Text_LED_Order = RGB
Text_Data0 = H300_L900
Text_Data1 = H900_L300
Text_MatrixUnit = 8x8
Text_MatrixType = Type1
Text_Pins_Matrix = 4x2
Text_Pins = [PA.0, PA.1, PA.2, PA.3, PA.4, PA.5, PA.6, PA.7]
LED0 = Text1.csv /Color_order=RGB /Format=Text ; 加入 LED Strip 檔案(.csv)。
[Path]
TR1: LEDText_PlayS($LED0)                    ; 播放 LED0 的資料，輸出至 PA.2。
TR2: SpiPlayS(ch,0), LEDText_Play(0)         ; 播放第 0 首 Speech 和第 0 個燈串。
```

5.29.10 LEDText_Stop

[NX1]

燈串播放停止指令，停止播放中的 Scrolling-Text 燈串。

LEDText_Stop

例.

```
TR1: LEDText_Stop                            ; 停止播放燈串。
```

5.29.11 LEDText_Clear

[NX1]

燈串清除指令，點滅 Scrolling-Text 燈串上所有 LED。

LEDText_Clear

例.

```
TR1: LEDText_Clear                    ; 點滅燈串。
```

5.29.12 LEDText_Brightness

[NX1]

燈串亮度設定指令，調整 Scrolling-Text 燈串亮度，n=0 ~ 15。

例.

```
TR1: LEDText_Brightness(15)           ; 燈串設定最亮。
```

5.30 QFID 指令（QFID Command）

QFID Command				
QFID_GroupID	QFID_TagId	QFID_TagInput	QFID_On	QFID_Off
QFID_SlowOn	QFID_SlowOff	-	-	-

5.30.1 QFID_GroupID(Ri)

[NX1]

掃描所有 QFID Group 的狀態。Ri[7:0]為 Group 的狀態，對應的 bit 為 1 表示有掃描到該 Group 的 Tag，若為 0 則無該 Group 的 Tag。

5.30.2 QFID_TagId (GroupID, RI:Rk:Rj:Ri)

[NY5+ / NY7 / NX1]

掃描所有 Tag 的狀態。

GroupID：為指定的 QFID Group ID，可為立即值或用變數來指定。

Ri：為 Tag0 ~ Tag3 的狀態。對應的 bit 為 1 表示有偵測到對應的 Tag，若為 0 則無。

Rj：為 Tag4 ~ Tag7 的狀態。對應的 bit 為 1 表示有偵測到對應的 Tag，若為 0 則無。

Rk：為 Tag8 ~ Tag11 的狀態。對應的 bit 為 1 表示有偵測到對應的 Tag，若為 0 則無。

RI：為 Tag12 ~ Tag15 的狀態。對應的 bit 為 1 表示有偵測到對應的 Tag，若為 0 則無。

注意：NY5+ / NY7 不可指定 GroupID。

5.30.3 QFID_TagInput (GroupID, ID, Ri)

[NY5+ / NY7 / NX1]

讀回 Tag 上三支 I/O 的輸入狀態。

GroupID：為指定的 QFID Group ID。
ID：為指定的 Tag ID，範圍為 0 ~ 15，可為立即值或用變數來指定。
Ri：為讀回的 I/O 狀態，Ri[0]表示 I/O1 狀態，Ri[1]表示 I/O2，Ri[2]表示 I/O3。

注意：NY5+ / NY7 不可指定 Group ID。

5.30.4 QFID_On

[NY5+ / NY7 / NX1]

開啟 QFID 掃描功能，開機後 QFID 功能預設為關閉。

5.30.5 QFID_Off

[NY5+ / NY7 / NX1]

關閉 QFID 掃描功能。

5.30.6 QFID_SlowOn

[NY5+ / NY7]

開啟 QFID 的低速掃描模式，模式開啟後，QFID 會以掃描一次休息數次的模式動作，休息的次數由 QFID Option 內的設定值決定，詳細說明請參考 QFID 段落小節；開啟此模式可以降低消耗電流，但反應速度也會同時降低。

注意：

1. ***此指令與一般指令的 Slow 指令不能同時執行，否則會造成休息時間長度錯誤。***
2. ***使用此指令前必須在 QFID Option 內選擇 slow 腳，並在腳位上並聯一顆電阻及一顆電容，否則動作會不正常；電阻及電容值請參考 Option 內的 QFID Reader Application Circuit。***

5.30.7 QFID_SlowOff

[NY5+ / NY7]

關閉 QFID 的低速掃描模式。

5.31 WaveID 指令（WaveID Command）

WaveID Command				
WaveID_TX	WaveID_RX_ON	WaveID_RX_OFF	WaveID_RX	

5.31.1 WaveID_TX(Ri)

[NX1]

代碼發射。

Ri：為指定的代碼，可為立即值或用變數來指定。

例.

```
[Option]
WaveID = TX                              ; 開啟 WaveID TX 功能。
WaveID_CommandCount = 8
[Path]
TR1: WaveID_TX(5)                        ; 發射代碼 5。
TR2: WaveID_TX(R0)                       ; 發射代碼 R0。
```

5.31.2 WaveID_RX_ON

[NX1]

開啟 WaveID 接收功能。

例.

```
[Path]
TR1: WaveID_RX_ON                        ; 開啟接收。
```

5.31.3 WaveID_RX_OFF

[NX1]

關閉 WaveID 接收功能。

例.

```
[Path]
TR1: WaveID_RX_OFF                       ; 關閉接收。
```

5.31.4 WaveID_RX

[NX1]

使用者可透過指令將接收到的 WaveID 代碼存在指定的變數。

例.

```
[Option]
WaveID = RX                              ; 開啟 WaveID RX 功能。
```

```
WaveID_CommandCount = 8
[Variable]
Var8: WaveId_Value
[Path]
TR1: WaveID_RX_On                          ; 啟用 WavdID RX。
WaveID_RX: WaveId_Value = WaveID_RX        ; 將接收到代碼存到 Var0。
```

5.32 聽聲辨位指令（Sound Localization Command）

Sound Localization Command				
SL_On	SL_Off	SL_RX	-	-

5.32.1 SL_On

[NX1]

開啟聽聲辨位掃描功能，開機後預設為關閉。

例.

```
[Option]
SoundLoc = Enable                          ; 開啟聽聲辨位功能。
SoundLoc_Amp_Pin = PA.15
SoundLoc_Amp_Shutdown_Control = Active_Low
[Variable]
Var8: R_Angle
[Path]
TR1: SL_On                                 ; 啟用聽聲辨位。
SL_RX: R_Angle=SL_RX, R_Angle=30?SL_30, R_Angle=60?SL_60
SL_30: PlayV(Ch0, $V0)                     ; 當 R_Angle 為 30，播放語音$V0。
SL_60: PlayV(Ch0, $V1)                     ; 當 R_Angle 為 60，播放語音$V1。
```

5.32.2 SL_Off

[NX1]

關閉聽聲辨位掃描功能。

5.32.3 Var=SL_RX

[NX1]

讀取聽聲辨位偵測到的角度，將數值存放於 Var，讀回的角度可為 0 / 30 / 60 / 90 / 120 / 150 / 180 / 210 / 240 / 270 / 300 / 330 共 12 個方位值，建議搭配 SL_RX 路徑使用。

5.33 聲音偵測指令（Sound Detect Command）

Sound Detect Command				
SoundDetect_On	SoundDetect_Off	-	-	-

5.33.1 SoundDetect_On

[NX1]

啟用聲音偵測功能。

例.

[Option]

SoundDetect = Enable

SoundDetect_High_Threshold = 300

SoundDetect_Low_Threshold = 300

SoundDetect_Active_Time = 80ms

SoundDetect_Mute_Time = 650ms

[Path]

TR1: SoundDetect_On ；啟用聲音偵測。

Sound_Detected: PA.0 = 1 ；偵測到聲音時，PA.0 輸出 1。

Sound_Muted: PA.0 = 0 ；當不再偵測到聲音超過 150ms 時，PA.0 輸出 0。

5.33.2 SoundDetect_Off

[NX1]

停用聲音偵測功能。

5.34 音高偵測指令（Pitch Detect Command）

Pitch Detect Command				
PitchDetect_On	PitchDetect_Off	ReadPitch	-	-

5.34.1 PitchDetect_On

[NX1]

啟用音高偵測功能。

例.

[Option]

PitchDetect = Tone ；偵測 Pure-Tone。

PitchDetect_Range = Middle ；偵測範圍 1000 ~ 3000Hz。

```
[Variable]
Var16: Pitch_Value
[Path]
TR1: PitchDetect_On                    ; 開啟音高偵測。
TR2: PitchDetect_Off                   ; 關閉音高偵測。
Pitch_Detected: ReadPitch(Pitch_Value) ; 偵測到 1000 ~ 3000Hz 內的 pure-tone，
                                       ; 將偵測到的音高儲存至 Pitch_Value。
```

5.34.2 PitchDetect_Off

[NX1]

停用音高偵測功能。

5.34.3 ReadPitch

[NX1]

讀取偵測到的音高。

ReadPitch(Var)

Var：將偵測到的音高儲存至 Var，範圍 31 ~ 4000。

5.35 RFC 指令（RFC Command）

RFC Command				
RFC_On	RFC_Off	RFC_Level(Ri)	-	-

5.35.1 RFC_On

[NY5+ / NY6 / NY7 / NX1]

開啟 RFC 掃描功能，開機後 RFC 功能預設為關閉。

5.35.2 RFC_Off

[NY5+ / NY6 / NY7 / NX1]

關閉 RFC 掃描功能。

5.35.3 RFC_Level(Ri)

[NY5+ / NY6 / NY7 / NX1]

讀取偵測到的 RFC 數值，將數值存放於 Ri。

注意：

1. *開機後需使用 RFC_On 指令將 RFC 功能開啟，並等待初始時間約 30ms 後才能利用此指令取到正確的 RFC 數值，否則可能會取到未知數值。*
2. *使用此指令前必須在 RFC 腳位接上一個電位器及一顆電容，電路接法請參考 RFC Option 內的 RFC Application Circuit。*

5.36 ADC 輸入指令（ADC Input Command）

ADC Input Command				
ADC_Input	-	-	-	-

5.36.1 ADC_Input

[NX1]

讀取類比數位轉換的值。

ADC_Input(Px.n, Var)

Px.n：由指定腳位讀取類比數位轉換結果。

Var：將讀取的類比數位轉換結果存放於 Var，結果範圍為 0 ~ 0xFFF0。

例.

```
[Option]
ADC_Input_Pins = PA.1, PA.2
[Variable]
Var16: Result
[Path]
TR: ADC_Input(PA.2, Result)          ; 讀取 PA.2 類比數位轉換的值，
                                     ; 並將結果存放於 Result。
```

5.37 語音辨識指令（VR Command）

VR Command				
VR State	VR_ON	VR_OFF	VR_VAD=n	VR_VAD_On
VR_VAD_Off	VRGC_Timeout_CLR		Ri = VR_HitScore	Ri = VR_HitID
Ri = VR_Loading	VT_Training	VT_Delete	VT_DeleteAll	VT_TrainingNum

5.37.1 VR State

[NX1]

VR State 用於定義語音指令的狀態，VR 狀態是指當語音指令被辨識後所應該作出的反應，使用者可以設定不同的 VR State，在不同的情況下語音指令被辨識後所作出的反應也可以不同。語音指令可以分成多個群組，但同時只有一個指令群組能被辨識。

例.

```
[VR]
VRG1_0: VR1   VR2   VR3                    ; 語音辨識指令群組 1 狀態 0
VRG2_0: VR4   VR5                          ; 語音辨識指令群組 2 狀態 0

[Path]
PowerOn: VRG1_0                            ; VR State 設置成 VRG1_0 狀態，只有第一組指令
                                           可以被辨識
```

5.37.2 VR_ON

[NX1]

啟用語音辨識功能。啟用語音辨識功能後，*Q-Code* 會不斷進行辨識是否有語音指令。請注意當播放聲音時，語音辨識功能會自動暫停，直到聲音播放結束，語音辨識功能才會自動回復，繼續進行語音指令的辨識。啟用語音辨識功能之前要先設定 VR State，在語音辨識功能開啟時可以隨時切換 VR State。

例.

```
[VR]
VRG1_0: VR1   VR2   VR3

[Path]
PowerOn: VRG1_0, VR_ON                     ; 開啟語音辨識功能
```

5.37.3 VR_OFF

[NX1]

關閉語音辨識功能。

例.

```
[Input State]
KEY1: TR1   TR2   X /TR4
[VR]
VRG1_0: VR1   VR2   VR3
[Path]
PowerOn: KEY1,VRG1_0,VR_ON
TR1: VR_OFF                                ; 關閉語音辨識功能
```

5.37.4 VR_VAD=n

[NX1]

設定語音辨識時輸入音量的閥值。語音啟動檢測 VAD（Voice Activity Detection）技術用於檢測語音訊號是否存在，設定其閥值也就是在某個音量以上才進行辨識。閥值愈大可以濾除愈大的背景雜訊，但也可能將小音量的語音指令濾除；而閥值太小可能無法將背景雜訊排除以防止誤動作，因此使用者需以使用環境來考量閥值。當未設定時，預設值為 4。

此 VAD 閥值相當於 CSMT（Cyberon CSpotter Modeling Tool）中的 energy threshold，惟兩者閥值的定義不相同，使用者可擇一使用，若 energy threshold > 0，則可設定 VR_VAD_Off；若 VR_VAD_On 則 energy threshold 可設為 0。閥值設定後還需以 VR_VAD_On 指令啟動。預設是 VR_VAD_Off。

n：閥值，可為立即值或變數，n= 0 ~ 16。若需要設定此表以外之值可利用 C Code 段落來直接呼叫底層的 API。

閥值	平均音量（%）	平均音量（Sample）	說明
0	0.00%	0	不濾除背景雜訊，即不啟用 VAD
1	0.46%	150	適用於非常安靜的封閉房間
2	0.61%	200	適用於有輕微背景噪音但沒有說話的封閉房間
3	0.76%	250	適用於有背景噪音或音樂但沒有說話的房間
4	0.92%	300	適用於 3 公尺外有說話的房間
5	1.07%	350	適用於 1 公尺外有說話的房間
6	1.22%	400	適用於 1 公尺內近距離有輕微活動的房間
7	1.53%	500	適用於 1 公尺內近距離有活動的房間
8	1.83%	600	適用於 1 公尺內近距離有說話聲的房間
9	2.14%	700	適用於較安靜的開放空間
10	2.44%	800	適用於有些微背景噪音的開放空間
11	3.66%	1000	適用於有比較大的背景噪音的開放空間
12	3.66%	1200	適用於有較大的背景噪音或近距離有說話聲的開放空間
13	4.58%	1500	適用於近距離 1 公尺內有活動，或有說話聲的開放空間
14	6.10%	2000	適用於近距離 1 公尺內有活動，或有較大說話聲的開放空間
15	9.16%	3000	適用於較吵雜的週邊環境，需要大音量才能辨識
16	13.73%	4500	適用於非常吵雜的週邊環境如展場，需要更大音量才能辨識

注意：若 energy threshold > 0，RAM 會多耗費約 300 bytes；而 VAD 僅多耗費 12 bytes。

例.

```
VR_VAD=1                    ; 設定 VAD 閥值為 1
```

5.37.5 VR_VAD_On

[NX1]

開啟語音辨識的 VAD 功能。當啟動 VAD，語音指令的音量須大於閥值，才會被判讀和辨識。

例.

```
VR_VAD_On                    ; 開啟 VAD
```

5.37.6 VR_VAD_Off

[NX1]

關閉語音辨識的 VAD 功能。

例.

```
VR_VAD_Off                   ; 關閉 VAD
```

5.37.7 VRGC_Timeout_CLR

[NX1]

清除 VRGC state 的 Timeout 計時器。當 VRGC 任何群組中有指令被辨識成功時，Timeout 計時器即被啟動，當語音指令組合已經全部的辨識完成，可使用此指令清除 Timeout 計時器。

例.

```
VRGC_Timeout_CLR             ; 清除 Timeout 計時器
```

5.37.8 Ri = VR_HitScore

[NX1]

取得目前已成功辨識的語音指令的分數，分數愈高表示愈符合語音指令。必須在語音指令成功辨識之後才可使用此指令。

例.

```
R0 = VR_HitScore             ; 將語音指令辨識分數儲存至變數 R0
```

5.37.9 Ri = VR_HitID

[NX1]

取得目前已成功辨識的語音指令的索引值。必須在語音指令成功辨識之後才可使用此指令。

例.

```
R0 = VR_HitID                ; 將語音指令索引值儲存至變數 R0
```

5.37.10 VR_Loading

[NX1]

取得語音辨識執行時，即時的 CPU 負載，單位為百分比

例.

```
R0 = VR_Loading                    ; 將 CPU 負載儲存至變數 R0
```

5.37.11 VT_Training

[NX1]

訓練語音指令，訓練成功的指令，可於語音識別時使用。

例.

```
VT_Training(1)                     ; 開始訓練第一組語音指令。
```

5.37.12 VT_Delete

[NX1]

刪除已訓練成功的語音指令。

例.

```
VT_Delete(1)                       ; 刪除第一組語音指令。
```

5.37.13 VT_DeleteAll

[NX1]

刪除所有訓練成功的語音指令。

例.

```
VT_DeleteAll                       ; 刪除所有語音指令。
```

5.37.14 VT_TrainingNum

[NX1]

設定訓練時錄音的次數，最多 3 次，最少 1 次，次數越多會越容易識別。Voice Tag 預設 3 次，Voice Password 預設 1 次。

例.

```
VT_TrainingNum=3                   ; 錄音次數設定 3 次。
```

5.38 變音效果指令（Sound Effect Command）

Sound Effect Command				
PitchChange	PitchChange_Off	SpeedChange	SpeedChange_Off	Robot1
Robot1_Off	Robot2	Robot2_Off	Robot3	Robot3_Off

Sound Effect Command				
Robot4	Robot4_Off	Echo	Echo_Off	Reverb
Reverb_Off	Darth	Darth_Off	AnimalRoar	AnimalRoar_Off

5.38.1 PitchChange

[NX1]

改變聲音的音高。聲音音高最高可調整為正常音高的兩倍，聲音音高最低可調整為正常音高的一半。

PitchChange(Channel, Pitch)

Channel：要改變的通道

Pitch：常數或變數，範圍必須落在-12 ~ +12 之間。

數值	說明
-12	0.5X
-11	0.53X
-10	0.56X
-9	0.59X
-8	0.63X
-7	0.67X
-6	0.72X
-5	0.75X
-4	0.79X
-3	0.84X
-2	0.89X
-1	0.94X
0	關閉 PitchChange
1	1.06X
2	1.12X
3	1.19X
4	1.26X
5	1.33X
6	1.41X
7	1.50X
8	1.59X
9	1.68X
10	1.78X
11	1.89X
12	2X

注意：

1. *Pitch = 0 等同於 PitchChange_Off。*
2. *若程序使用兩個通道，PitchChange 同時只能使用一通道，不允許同時兩通道 PitchChange。*
3. *PitchChange 所指定的通道與 SpeedChange 所指定的通道同時操作必須使用相同通道。*
4. *此指令只對 Voice 檔案有作用，MIDI 檔案播放不受此效果影響。*
5. *此指令只對取樣頻率小於或等於 16KHz 的 Voice 檔案有作用。*
6. *此指令只對 ADPCM / PCM / SBC-1 / CELP 有作用。*

例. **PitchChange(ch1, 1)**

5.38.2 PitchChange_Off

[NX1]

關閉 PitchChange 效果。

注意：必須有使用 PitchChange 方可使用 PitchChange_Off。

5.38.3 SpeedChange

[NX1]

改變聲音播放速度，聲音播放速度最高可調整為正常播放速度的兩倍。聲音播放速度最低可調整為正常播放速度的一半。

SpeedChange(channel, number)

Channel：作用通道

Number：常數或變數，範圍必須落在-12 ~ +12 之間。

數值	說明
-12	0.5X
-11	0.54X
-10	0.58X
-9	0.62X
-8	0.67X
-7	0.71X
-6	0.75X
-5	0.79X
-4	0.83X
-3	0.87X
-2	0.92X
-1	0.96X
0	關閉 SpeedChange
1	1.08X

數值	說明
2	1.17X
3	1.25X
4	1.33X
5	1.42X
6	1.45X
7	1.58X
8	1.67X
9	1.75X
10	1.83X
11	1.92X
12	2X

注意：

1. ***Number = 0 等同於 SpeedChange_Off。***
2. ***若程序使用兩個通道，SpeedChange 同時只能使用一通道，不允許同時兩通道 SpeedChange。***
3. ***PitchChange 所指定的通道與 SpeedChange 所指定的通道同時操作必須使用相同通道。***
4. ***此指令只對 Voice 檔案有作用，MIDI 檔案播放不受此效果影響。***
5. ***此指令只對取樣頻率小於或等於 16K 的 Voice 檔案有作用。***
6. ***此指令只對 ADPCM / PCM / SBC-1 / CELP 有作用。***

例. **SpeedChange(ch1, 1)**

5.38.4 SpeedChange_Off

[NX1]

關閉 SpeedChange 效果。

注意：必須有使用 SpeedChange 方可使用 SpeedChange_Off。

5.38.5 Robot1

[NX1]

用來產生機器人講話的效果。

Robot1(Channel, number)

Channel：通道

Number：範圍 0 ~ 15

注意：

1. ***此指令只對 Voice 檔案有作用，MIDI 檔案播放不受此效果影響。***
2. ***此指令只對 ADPCM / PCM / SBC-1 / CELP 有作用。***

5.38.6 Robot1_Off

[NX1]

關閉 Robot1 效果。

注意：必須有使用 Robot1 方可使用 Robot1_Off。

5.38.7 Robot2

[NX1]

用來產生機器人講話的效果。

Robot2(Channel, Type,Thres)

Channel：語音通道

Type：範圍 0 ~ 2，表示不同音階

Thres：範圍 0 ~ 7，噪音門檻值，一般語音播放因為沒有外界雜訊，可設成 0。

注意：

1. 若程序使用兩個通道，Robot2 同時只能使用一通道，不允許同時兩通道 Robot2。

2. 此指令只對 Voice 檔案有作用，MIDI 檔案播放不受此效果影響。

3. 此指令只對 ADPCM / PCM / SBC-1 / CELP 有作用。

例. **Robot2(ch0, 2,0)**

5.38.8 Robot2_Off

[NX1]

關閉 Robot2 效果。

注意：必須有使用 Robot2 方可使用 Robot2_Off。

5.38.9 Robot3

[NX1]

用來產生機器人講話的效果。

Robot3(Channel, Type,Thres)

Channel：語音通道

Type：範圍 0 ~ 2，表示不同音階

Thres：範圍 0~7，噪音門檻值，一般語音播放因為沒有外界雜訊，可設成 0

注意：

1. 若程序使用兩個通道，Robot3 同時只能使用一通道，不允許同時兩通道 Robot3。

2. 此指令只對 Voice 檔案有作用，MIDI 檔案播放不受此效果影響。

3. 此指令只對 ADPCM / PCM / SBC-1 / CELP 有作用。

例. **Robot3(ch0, 2,0)**

5.38.10 Robot3_Off

[NX1]

關閉 Robot3 效果。

注意：必須有使用 Robot3 方可使用 Robot3_Off。

5.38.11 Robot4

[NX1]

用來產生機器人講話的效果。

Robot4(Channel)

Channel：語音通道，僅支援 Ch0。

注意：

1. 此指令僅支援 ADPCM。

2. 此指令只對 Voice 檔案有作用，MIDI 檔案播放不受此效果影響。

例. **Robot4(Ch0)**

5.38.12 Robot4_Off

[NX1]

關閉 Robot4 效果。

注意：必須有使用 Robot4 方可使用 Robot4_Off。

5.38.13 Echo

[NX1]

用來產生迴音的效果。

Echo(Channel, Level)

Channel：語音通道

Level：範圍 0 ~ 7，表示不同迴音程度，7 是最長，此參數僅對 SBC/ADPCM Chann1/CELP/PCM 有效

Option ADPCM_ECHO_DELAY：short 表示短的迴音效果，long 表示長的迴音效果，此 option 僅對 Channel0 且聲音壓縮為 ADPCM 才有效

注意：

1. 若程序使用兩個通道，Echo 同時只能使用一通道，不允許同時兩通道 Echo。

2. ADPCM Channel0 的 Echo 效果較佳，建議使用 Echo 音效時，選擇 ADPCM 並用 Channel0 播放。

3. 此指令只對 Voice 檔案有作用，MIDI 檔案播放不受此效果影響。

4. 此指令只對 ADPCM / PCM / SBC-1 / CELP 演算法有作用。

例.

```
[Option]
ADPCM_ECHO_DELAY = Short                    ; Echo 迴音 delay 選擇短的
[Path]
PowerOn: PlayV(ch0,$V0), Echo(ch0,0)        ; 播放 Echo 音效
```

5.38.14 Echo_Off

[NX1]

關閉 Echo 效果。

注意：必須有使用 Echo 方可使用 Echo_Off。

5.38.15 Reverb

[NX1]

用來產生迴響的效果。

Reverb(Channel, Level)

Channel：語音通道

Level：範圍 0 ~ 7，表示不同迴響程度，7 程度最高

注意：

1. 若程序使用兩個通道，Reverb 同時只能使用一通道，不允許同時兩通道 Reverb。

2. 此指令只對 Voice 檔案有作用，MIDI 檔案播放不受此效果影響。

3. 此指令只對 ADPCM / PCM / SBC-1 / CELP 演算法有作用。

5.38.16 Reverb_Off

[NX1]

關閉 Reverb 效果。

注意：必須有使用 Reverb 方可使用 Reverb_Off。

5.38.17 Darth

[NX1]

用來產生黑武士的效果。

Darth(Channel, Pitch, Type)

Channel：語音通道，僅支援 Ch0。

Pitch：範圍-12 ~ 12，音高設定，設定同 **PitchChange**。

Type：範圍 0 ~ 3，設定黑武士類別，金屬音設定 0，金屬音+沙啞音設定 1/2，沙啞音設定 3。

注意：

1. 此指令僅支援 ADPCM。

2. 此指令只對 Voice 檔案有作用，MIDI 檔案播放不受此效果影響。

例. **Darth(Ch0,-4,2)**

5.38.18 Darth_Off

[NX1]

關閉 Darth 效果。

注意：必須有使用 Darth 方可使用 Darth_Off。

5.38.19 AnimalRoar

[NX1]

用來產生 AnimalRoar 的效果。

AnimalRoar(Channel, Speed, Pitch, Reverb)

Channel：語音通道，僅支援 Ch0。

Speed：範圍-12 ~ 12，速度設定，設定同 **SpeedChange** 的 Number 參數。

Pitch：範圍-12 ~ 12，音高設定，設定同 **PitchChange**。

Reverb：範圍 0 ~ 3。

注意：

1. 此指令僅支援 ADPCM。

2. 此指令只對 Voice 檔案有作用，MIDI 檔案播放不受此效果影響。

3. 使用 AnimalRoar 音效，Ch0 禁止其他音效的使用。

例. **AnimalRoar(Ch0,-4, -4, 2)**

5.38.20 AnimalRoar_Off

[NX1]

關閉 AnimalRoar 效果。

注意：必須有使用 AnimalRoar 方可使用 AnimalRoar_Off。

5.39 即時播放指令（Real Time Play Command）

Real Time Play Command				
RT_Play	RT_Play_Off	RT_PitchChange	RT_PitchChange_Off	RT_Robot1
RT_Robot1_Off	RT_Robot2	RT_Robot2_Off	RT_Robot3	RT_Robot3_Off
RT_Robot4	RT_Robot4_Off	RT_Echo	RT_Echo_Off	RT_Reverb
RT_Reverb_Off	RT_Ghost	RT_Ghost_Off	RT_Darth	RT_Darth_Off
RT_Chorus	RT_Chorus_Off	RT_Vol = n	RT_Vol = Var	Var = RT_Vol

5.39.1 RT_Play

[NX1]

即時播放(大聲公)指令。

注意：使用該指令時，語音通道最多支援兩通道。

例.

```
[Option]
RT_Sampling_Rate = 8000          ; 即時播放的取樣頻率 8k，亦可設定 12k 或 16k
[Path]
PowerOn: RT_Play                 ; 即時播放
```

5.39.2 RT_Play_Off

[NX1]

關閉即時播放。

注意：必須有使用 RT_Play 方可使用 RT_Play_Off。

5.39.3 RT_PitchChange

[NX1]

即時播放的音高設定。聲音音高最高可調整為正常音高的兩倍，聲音音高最低可調整為正常音高的一半。

RT_PitchChange(Pitch)

Pitch：常數或變數，範圍必須落在-12 ~ +12 之間。

數值	說明
-12	0.5X
-11	0.53X
-10	0.56X
-9	0.59X

數值	說明
-8	0.63X
-7	0.67X
-6	0.72X
-5	0.75X
-4	0.79X
-3	0.84X
-2	0.89X
-1	0.94X
0	關閉 PitchChange
1	1.06X
2	1.12X
3	1.19X
4	1.26X
5	1.33X
6	1.41X
7	1.50X
8	1.59X
9	1.68X
10	1.78X
11	1.89X
12	2X

*注意：**Pitch = 0 等同於 RT_PitchChange_Off**。*

例. **RT_PitchChange(1)**

5.39.4 RT_PitchChange_Off

[NX1]

關閉即時播放音高變化。

*注意：必須有使用 **RT_PitchChange** 方可使用 **RT_PitchChange_Off**。*

5.39.5 RT_Robot1

[NX1]

用來產生機械音的效果。

RT_Robot1(Number)

Number：範圍 0 ~ 15。

例. **RT_Robot1(1)**

5.39.6 RT_Robot1_Off

[NX1]

關閉即時播放機械音。

注意：必須有使用 RT_Robot1 方可使用 RT_Robot1_Off。

5.39.7 RT_Robot2

[NX1]

用來產生機械音的效果。

RT_Robot2(Type,Thres)

Type：範圍 0 ~ 2，表示不同音階。

Thres：範圍 0 ~ 7，噪音門檻值，數值越高門檻值越高，適合較吵雜的環境。

例. **RT_Robot2(1,3)**

5.39.8 RT_Robot2_Off

[NX1]

關閉即時播放機械音。

注意：必須有使用 RT_Robot2 方可使用 RT_Robot2_Off。

5.39.9 RT_Robot3

[NX1]

用來產生機械音的效果。

RT_Robot3(Type,Thres)

Type：範圍 0 ~ 2，表示不同音階。

Thres：範圍 0 ~ 7，噪音門檻值，數值越高門檻值越高，適合較吵雜的環境。

例. **RT_Robot3(1,3)**

5.39.10 RT_Robot3_Off

[NX1]

關閉即時播放機械音。

注意：必須有使用 RT_Robot3 方可使用 RT_Robot3_Off。

5.39.11 RT_Robot4

[NX1]

用來產生機械音的效果。

例. **RT_Robot4**

5.39.12 RT_Robot4_Off

[NX1]

關閉即時播放機械音。

注意：必須有使用 RT_Robot4 方可使用 RT_Robot4_Off。

5.39.13 RT_Echo

[NX1]

用來產生迴音的效果。

Option RT_ECHO_DELAY：short 表示短的迴音效果，long 表示長的迴音效果。

例.

```
[Option]
RT_ECHO_DELAY = Short                    ; Echo 迴音 delay 選擇短的
[Path]
PowerOn: RT_Play, RT_Echo                ; 即時播放使用 Echo 音效
```

5.39.14 RT_Echo_Off

[NX1]

關閉即時播放 Echo 功能。

注意：必須有使用 RT_Echo 方可使用 RT_Echo_Off。

5.39.15 RT_Reverb

[NX1]

用來產生迴響的效果。

RT_Reverb(Level)

Level：範圍 0 ~ 7，表示不同迴響程度，7 程度最高

例. **RT_Reverb(7)**

5.39.16 RT_Reverb_Off

[NX1]

關閉即時播放 Reverb 功能。

注意：必須有使用 RT_Reverb 方可使用 RT_Reverb_Off。

5.39.17 RT_Ghost

[NX1]

用來產生鬼聲的效果。

RT_Ghost(Pitch,Type)

Pitch：範圍-12 ~ 12，音高設定，女生建議設定 1 ~ 12，男生建議設定-1 ~ -12。

Type：範圍 0 ~ 2，模擬鬼講話建議設定 0 ~ 1，模擬鬼叫聲建議設定 2。

例. **RT_Ghost(12,2)**

5.39.18 RT_Ghost_Off

[NX1]

關閉即時播放鬼聲功能。

注意：必須有使用 RT_Ghost 方可使用 RT_Ghost_Off。

5.39.19 RT_Darth

[NX1]

用來產生黑武士的效果。

RT_Darth(Pitch,Type)

Pitch：範圍-12 ~ 12，音高設定，設定同 **RT_PitchChange**。

Type：範圍 0 ~ 3，設定黑武士類別，金屬音設定 0，金屬音+沙啞音設定 1/2，沙啞音設定 3。

例. **RT_Darth(-4,2)**

5.39.20 RT_Darth_Off

[NX1]

關閉即時播放黑武士功能。

注意：必須有使用 RT_Darth 方可使用 RT_Darth_Off。

5.39.21 RT_Chorus

[NX1]

用來產生合音的效果。

RT_Chorus(Pitch, Vol_1, Vol_2, Vol_3, Gain)

Pitch：範圍-12 ~ 12，音高設定，設定同 **RT_PitchChange**，合音音軌 1 與合音音軌 3 是可隨著 Pitch 調整音調，合音音軌 2 維持原音。

Vol_1：範圍 0 ~ 100，設定合音音軌 1 的音量。

Vol_2：範圍 0 ~ 100，設定合音音軌 2 的音量。

Vol_3：範圍 0 ~ 100，設定合音音軌 3 的音量。

Gain：範圍 0 ~ 10，設定整體音量增益。

Value	Gain
0	1.0
1	1.1
2	1.2
3	1.3
4	1.4
5	1.5
6	1.6
7	1.7
8	1.8
9	1.9
10	2.0

例. **RT_Chorus(-8, 80, 80, 80, 6)**

5.39.22 RT_Chorus_Off

[NX1]

關閉即時播放合音功能。

注意：必須有使用 RT_Chorus 方可使用 RT_Chorus_Off。

5.39.23 RT_Vol = n / RT_Vol = Var

[NX1]

修改 RT 的音量。設定的音量可為立即值或用變數控制。

n：通道音量，支援 0 ~ 15。

注意：必須有使用 RT 相關指令才能使用 RT_Vol 指令。

例. **RT_Vol = 10** ; 將 RT 的音量設為 10，約原本音量的 67%。

例. **RT_Vol = R0** ; 取 R0 內的數值當成 RT 的音量。

5.39.24 Var = RT_Vol

[NX1]

讀取 RT 的音量。

注意：必須有使用 RT 相關指令才能使用 RT_Vol 指令。

例. **R0 = RT_Vol** ; 讀取 RT 的音量，並存於 R0。

5.40 動物變音指令（Animaltalks Command）

Animaltalks Command				
Animaltalks_On	Animaltalks_Off	Animaltalks_Record	Animaltalks_RecordS	
Animaltalks_StopR	Animaltalks_Play	Animaltalks_PlayS	Animaltalks_Stop	
Animaltalks_SetVoice		Animaltalks_LongSound_On		-
Animaltalks_LongSound_Off		-	-	-

5.40.1 Animaltalks_On

[NX1]

開啟動物變音功能。

*注意：使用動物變音前必須叫用 **Animaltalks_On** 以配置系統資源。*

例.

[Path]

TR1: AnimalTalks_On ；開啟動物變音功能。

5.40.2 Animaltalks_Off

[NX1]

關閉動物變音功能。

*注意：動物變音使用完畢後必須叫用 **Animaltalks_Off** 以回收系統資源。*

例.

[Path]

TR1: AnimalTalks_Off ；關閉動物變音功能。

5.40.3 Animaltalks_Record / Animaltalks_RecordS

[NX1]

進行動物變音的錄音。錄音完成後，可以透過 AnimalTalks_Play / Animaltalks_PlayS 播放。

Animaltalks_Record(Label {,Time})

Animaltalks_RecordS (Label {, Time})

Animaltalks_Record：待錄音結束後，執行下一個指令。

Animaltalks_RecordS：開始錄音後，立即執行下一個指令。

Label：**[Record]** 中定義的錄音區段名稱。

Time：錄音的長度，未定則為該錄音區段的長度。

例.

[Path]

```
TR1: Animaltalks_Record($Rec0)          ; 進行動物變音的錄音，使用 Rec0 區段。
TR2: Animaltalks_Play                   ; 使用 Animaltalks_Play 播放錄音結果。
```

5.40.4 Animaltalks_StopR

[NX1]

用來停止動物變音的錄音。

例.

```
[Path]
TR1: Animaltalks_Record($Rec0)
TR2: Animaltalks_StopR                  ; 停止當前的錄音。
```

5.40.5 Animaltalks_Play / Animaltalks_PlayS

[NX1]

進行動物變音的播音。

Animaltalks_Play：待播放結束後，執行下一個指令。

Animaltalks_PlayS：開始播放後，立即執行下一個指令。

注意：動物變音的播放會直接使用有音效的 Audio 通道，原播放通道將會直接停止。

例.

```
[Path]
TR1: Animaltalks_Play                   ; 使用 Animaltalks_Play 播放錄音結果。
```

5.40.6 Animaltalks_Stop

[NX1]

用來停止動物變音的播音。

例.

```
[Path]
TR1: Animaltalks_Play
TR2: Animaltalks_Stop                   ; 停止當前的播音。
```

5.40.7 Animaltalks_SetVoice

[NX1]

設定 AnimalTalks_Play / Animaltalks_PlayS 時使用的動物原音音檔。

Animaltalks_SetVoice(Index {, Storage})

Index：動物音檔索引。

Storage：播放的動物音檔所在的儲存空間，若不指定則為內部空間，可使用 udr、spi0、spi1。

例.

[Path]

TR1: Animaltalks_SetVoice(2, spi0) ；設定動物變音音檔，使用 SPI0 的索引 2 音檔。

TR2: Animaltalks_Play ；使用 Animaltalks_Play 播放錄音結果。

5.40.8 Animaltalks_LongSound_On

[NX1]

設定長動物音模式，會略過靜音段的播放。

Animaltalks_LongSound_On(Factor)

Factor：播放段落持續時間係數，設定範圍 1% ~ 65535%。

例.

[Path]

TR1: Animaltalks_SetVoice(2, spi0) ；設定動物變音音檔，使用 SPI0 的索引 2 音檔。

TR2: Animaltalks_LongSound_On(300%) ；設定長音音檔播放時間係數 300%。

TR3: Animaltalks_Play ；使用 Animaltalks_Play 播放錄音結果。

5.40.9 Animaltalks_LongSound_Off

[NX1]

關閉長動物音模式。

例.

[Path]

TR1: Animaltalks_SetVoice(2, spi0) ；設定動物變音音檔，使用 SPI0 的索引 2 音檔。

TR2: Animaltalks_LongSound_Off ；關閉長音音檔模式。

TR3: Animaltalks_Play ；使用 Animaltalks_Play 播放錄音結果。

5.41 動物唱歌指令（Animalsings Command）

Animalsings Command			
Animalsings_On	Animalsings_Off	Animalsings_Record	Animalsings_RecordS
Animalsings_StopR	Animalsings_Play	Animalsings_PlayS	Animalsings_Stop
Animalsings_SetVoice		Animalsings_LongSound_On	
Animalsings_LongSound_Off		Animalsings_NC_On	Animalsings_NC_Off

Animalsings_NC_Auto		

5.41.1 Animalsings_On

[NX1]

開啟動物唱歌功能。

*注意：使用動物唱歌前必須叫用 **Animalsings_On** 以配置系統資源。*

例.

[Path]

TR1: AnimalSings_On ；開啟動物唱歌功能。

5.41.2 Animalsings_Off

[NX1]

關閉動物唱歌功能。

*注意：動物唱歌使用完畢後必須叫用 **Animalsings_Off** 以回收系統資源。*

例.

[Path]

TR1: AnimalSings_Off ；關閉動物唱歌功能。

5.41.3 Animalsings_Record / Animalsings_RecordS

[NX1]

進行動物唱歌的錄音。錄音完成後，可以透過 AnimalSings_Play / AnimalSings_PlayS 播放。

Animalsings_Record(Label {,Time})

Animalsings_RecordS (Label {, Time})

Animalsings_Record：待錄音結束後，執行下一個指令。

Animalsings_RecordS：開始錄音後，立即執行下一個指令。

Label：**[Record]** 中定義的錄音區段名稱。

Time：錄音的長度，未定則為該錄音區段的長度。

例.

[Path]

TR1: Animasings_Record($Rec0) ；進行動物唱歌的錄音，使用 **Rec0** 區段。

TR2: Animalsings_Play ；使用 **Animalsings_Play** 播放錄音結果。

5.41.4 Animalsings_StopR

[NX1]

用來停止動物唱歌的錄音。

例.

[Path]

TR1: Animalsings_Record($Rec0)

TR2: Animalsings_StopR ; 停止當前的錄音。

5.41.5 Animalsings_Play / Animalsings_PlayS

[NX1]

進行動物唱歌的播音。

Animalsings_Play：待播放結束後，執行下一個指令。

Animalsings_PlayS：開始播放後，立即執行下一個指令。

*注意：動物唱歌的播放會直接使用有音效的 **Audio** 通道，原播放通道將會直接停止。*

例.

[Path]

TR1: Animalsings_Play ; 使用 **Animalsings_Play** 播放錄音結果。

5.41.6 Animalsings_Stop

[NX1]

用來停止動物唱歌的播音。

例.

[Path]

TR1: Animalsings_Play

TR2: Animalsings_Stop ; 停止當前的播音。

5.41.7 Animalsings_SetVoice

[NX1]

設定 AnimalSings_Play / AnimalSings_PlayS 時使用的動物原音音檔。

Animalsings_SetVoice(Index)

Animalsings_SetVoice(Index, Storage)

Index：動物音檔索引。

Storage：播放的動物音檔所在的儲存空間，若不指定則為內部空間，可使用 udr、spi0、spi1。

例.

[Path]

TR1: Animalsings_SetVoice(2, spi0) ; 設定動物唱歌音檔，使用 **SPI0** 的索引 **2** 音檔。

```
TR2: Animalsings_Play                ; 使用 Animalsings_Play 播放錄音結果。
```

5.41.8 Animalsings_LongSound_On

[NX1]

設定長動物音模式，會略過靜音段的播放。

例.

```
[Path]
TR1: Animalsings_SetVoice(2, spi0)   ; 設定動物變音音檔，使用 SPI0 的索引 2 音檔。
TR2: Animalsings_LongSound_On        ; 設定長音音檔模式。
TR3: Animalsings_Play                ; 使用 Animalsings_Play 播放錄音結果。
```

5.41.9 Animalsings_LongSound_Off

[NX1]

關閉長動物音模式。

例.

```
[Path]
TR1: Animalsings_SetVoice(2, spi0)   ; 設定動物變音音檔，使用 SPI0 的索引 2 音檔。
TR2: Animalsings_LongSound_Off       ; 關閉長音音檔模式。
TR3: Animalsings_Play                ; 使用 Animalsings_Play 播放錄音結果。
```

5.41.10 Animalsings_NC_On

[NX1]

啟動動物音降噪功能。

例.

```
[Path]
TR1: Animalsings_NC_On               ; 設定動物音降噪功能。
```

5.41.11 Animalsings_NC_Off

[NX1]

關閉動物音降噪功能。

例.

```
[Path]
TR1: Animalsings_NC_Off              ; 關閉動物音降噪功能。
```

5.41.12 Animalsings_NC_Auto

[NX1]

設定動物音自動判斷降噪功能。

例.

[Path]

TR1: Animalsings_NC_Auto ; 設定動物音自動判斷降噪功能。

5.42 I/O 擴展晶片指令（I/O Expander Command）

I/O Expander Command				
IoExp Input State	IoExp Output State	IoExp_Key_CLR	IoExp_Key_ON	IoExp_Key_OFF
IoExp_Sleep	IoExp_ErrId	IoExp_ReadInfo	IoExp_ReadSN	-
IoExp_SetInputPullHigh		IoExp_SetInputPullLow		
IoExp_SetInputFloating		IoExp_SetOutputHigh		
IoExp_SetOutputLow		-		

5.42.1 IoExp Input State

[NY5+ / NX1]

用於改變現在 I/O 擴展晶片上的按鍵對應的型態。使用者可以在 **[Input State Exp0] / [Input State Exp1] / [Input State Exp2] / [Input State Exp3]** 中設定不同的 Input State，藉以控制不同按鍵狀態的需求。

例.

[Input State Exp0]

IoExp0_KEY1: TR1 TR2 X /TR4

[Path]

PowerOn: IoExp0_KEY1 ; I/O 擴展晶片的按鍵設置成 IoExp0_KEY1 狀態。

5.42.2 IoExp Output State

[NY5+ / NX1]

用於改變現在 I/O 擴展晶片輸出腳位的狀態。使用者可以在 **[Output State Exp0] / [Output State Exp1] / [Output State Exp2] / [Output State Exp3]** 中設定不同的 Output State，藉以控制各輸出腳位的狀態。

例.

[Output State Exp0]

IoExp0_Output_0: 0 1 0 1

[Path]

PowerOn: IoExp0_Output_0 **; I/O 擴展晶片輸出腳位設置成 IoExp0_Output_0 狀態。**

5.42.3 IoExp_Key_CLR

[NY5+ / NX1]

用於清除當前 I/O 擴展晶片的按鍵狀態，當按鍵狀態被清除後，I/O 擴展晶片會重新掃描按鍵。

IoExp_Key_CLR

IoExp_Key_CLR(ExpId)

ExpId：I/O 擴展晶片 ID。

5.42.4 IoExp_Key_ON

[NY5+ / NX1]

開啟 I/O 擴展晶片按鍵掃描功能。若是開啟該功能，則 I/O 擴展晶片會持續做掃描按鍵。

IoExp_Key_ON

IoExp_Key_ON(ExpId)

ExpId：I/O 擴展晶片 ID。

注意：系統預設值為 IoExp_Key_ON，PowerOn 後無須另外再下達一次 IoExp_Key_ON 指令。

5.42.5 IoExp_Key_OFF

[NY5+ / NX1]

關閉 I/O 擴展晶片按鍵掃描功能。若是關閉該功能，則 I/O 擴展晶片不做按鍵掃描。

IoExp_Key_OFF

IoExp_Key_Off(ExpId)

ExpId：I/O 擴展晶片 ID。

5.42.6 IoExp_Sleep

[NY5+ / NX1]

讓所有的 I/O 擴展晶片進入休眠。

注意：喚醒後須等待 1ms 才能對 I/O 擴展晶片下達指令。

5.42.7 IoExp_ErrId

[NY5+ / NX1]

當與 I/O 擴展晶片通訊失敗時，可使用此指令取得 I/O 擴展晶片 ID。

Var = IoExp_ErrId

Var：取得的 I/O 擴展晶片 ID。

5.42.8 IoExp_ReadInfo

[NY5+ / NX1]

讀取 I/O 擴展晶片的資訊。

IoExp_ReadInfo(ExpId, Body, Func, FwVer)

ExpId：I/O 擴展晶片 ID。
Body：I/O 擴展晶片的 Body 代碼。
Func：I/O 擴展晶片的功能代碼。
FwVer：I/O 擴展晶片的軔體版本。

5.42.9 IoExp_ReadSN

[NY5+ / NX1]

每次針對 I/O 擴展晶片執行指令時，都會增加 SN 的計數，此指令用來讀取 I/O 擴展晶片的 SN。

IoExp_ReadSN(ExpId, SN)

ExpID：I/O 擴展晶片 ID。
SN：I/O 擴展晶片的 SN。

5.42.10 IoExp_SetInputPullHigh

[NY5+ / NX1]

將 I/O 擴展晶片上的腳位設定為帶上拉電阻的輸入模式。

IoExp_SetInputPullHigh(Port, Value)

Port：I/O 擴展晶片上的 Port，例如：Exp0_P0 為編號 0 的 I/O 擴展晶片上的 P0。
Value：可為數字或變數，要設定腳位為帶上拉電阻的輸入模式，需將此參數的對應 bit 設為 1。

5.42.11 IoExp_SetInputPullLow

[NY5+ / NX1]

將 I/O 擴展晶片上的腳位設定為帶下拉電阻的輸入模式。

IoExp_SetInputPullLow(Port, Value)

Port：I/O 擴展晶片上的 Port，例如：Exp0_P0 為編號 0 的 I/O 擴展晶片上的 P0。

Value：可為數字或變數，要設定腳位為帶下拉電阻的輸入模式，需將此參數的對應 bit 設為 1。

5.42.12 IoExp_SetInputFloating

[NY5+ / NX1]

將 I/O 擴展晶片上的腳位設定為不帶上拉電阻的輸入模式。

IoExp_SetInputFloating(Port, Value)

Port：I/O 擴展晶片上的 Port，例如：Exp0_P0 為編號 0 的 I/O 擴展晶片上的 P0。

Value：可為數字或變數，要設定腳位為不帶上拉電阻的輸入模式，需將此參數的對應 bit 設為 1。

5.42.13 IoExp_SetOutputHigh

[NY5+ / NX1]

將 I/O 擴展晶片上的腳位設定為輸出 High。

IoExp_SetOutputHigh(Port, Value)

Port：I/O 擴展晶片上的 Port，例如：Exp0_P0 為編號 0 的 I/O 擴展晶片上的 P0。

Value：可為數字或變數，要設定腳位為輸出 High，需將此參數的對應 bit 設為 1。

5.42.14 IoExp_SetOutputLow

[NY5+ / NX1]

將 I/O 擴展晶片上的腳位設定為輸出 Low。

IoExp_SetOutputLow(Port, Value)

Port：I/O 擴展晶片上的 Port，例如：Exp0_P0 為編號 0 的 I/O 擴展晶片上的 P0。

Value：可為數字或變數，要設定腳位為輸出 Low，需將此參數的對應 bit 設為 1。

5.43 一般指令（MISC Command）

MISC Command				
Input State	Output State	Action Mark State		-
Wave Mark State	Melody Mark State		Note On State	-
PWM-IO Mark State		QFID State	Key_CLR	Key_ON
Key_OFF	Stop	Pause(n)	Resume(n)	ReadChannel
PauseDown	ResumeUp	Audio_Loop_On	Audio_Loop_Off	Audio_ON

Audio_OFF	AudioMode	NoiseFilter_ON	NoiseFilter_OFF	EQ_Filter
EQ_Filter_Off	AGC_On	AGC_Off	RampUp	RampDown
AutoSleep_On	AutoSleep_Off	Sleep	WDT_CLR	Repeat
Background	Slow	SlowOff	END	ChMode(n)
ReadRollingCode	ReadRadj(Ri)	SW_Reset	Debounce(Time)	-
Direct_Debounce(Time)		Matrix_Debounce(Time)		Ri=LVD
Enforce_Wakeup	GetMicVol	Millis	Srand	-

5.43.1 Input State

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

用於改變現在按鍵對應的型態。使用者可以在 **[Input State]** 中設定不同的 Input State，藉以控制不同按鍵狀態的需求。

例.

[Input State]

KEY1: TR1 TR2 X /TR4

[Path]

PowerOn: KEY1 **; Input State 設置成 KEY1 狀態。**

5.43.2 Output State

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

用於改變現在輸出腳位的狀態。使用者可以在 **[Output State]** 中設定不同的 Output State，藉以控制各輸出腳位的狀態。

例.

[Output State]

Output_0: 0 1 0 1

[Path]

PowerOn: Output_0 **; Output State 設置成 Output_0 狀態。**

5.43.3 Action Mark State

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

Q-Visio 編輯檔案時，可以在 action 檔案中插入 Mark 標記。當 *Q-Code* 讀到標記時，會依照當前的 Action Mark 狀態自動執行對應的路徑。使用者可以在[Action Mark]段落中定義 Action Mark 的狀態。

Action Mark 的標記編號最多可有 255 個，從 A1 至 A255。在 Q-Code 中，使用者僅需要加入.vio 檔及定義好標記對應的路徑即可。最多可以定義 255 組設定。

使用者在播放 Action 文件時，可以用 *Q-Visio* 插入 Mark 來標記 Action Mark。標記可由 A1 至 A255。當播放 Action 檔時，可以藉由 Action Mark State 來切換目前要對應的 Action Mark 的路徑。最多可定義 255 個 Action Mark State。

例.

```
[Action Mark]
;                A1      A2      A3      A4
ActionMark1:     AM1     AM2     AM3     AM4

[Path]
PowerOn: ActionMark1, PlayA(PF.3, CH1, $VIO1.A1)

[Background1]
AM1: PB=0x1                          ; 當讀取到 Action Mark 編號 1，即會執行 AM1。
AM2: PB=0x2                          ; 當讀取到 Action Mark 編號 2，即會執行 AM2。
AM3: PB=0x3                          ; 當讀取到 Action Mark 編號 3，即會執行 AM3。
AM4: PB=0x0                          ; 當讀取到 Action Mark 編號 4，即會執行 AM4。
```

5.43.4 Wave Mark State

[NY4 / NY5 / NY5+ / NX1]

使用者在播放 Voice 文件時，可以由 *Quick-IO* 來插入 Mark 來標記 Wave Mark。標記最多可有[M01]至[M15]。當播放 Voice 檔時，可以藉由 Wave Mark State 來切換目前要對應的 Wave Mark 的路徑。最多可定義 255 個 Wave Mark State。

```
[Wave Mark]
;                M1      M2      M3      M4
WaveMark1:       WM1     WM2     WM3     WM4

[Path]
PowerOn: WaveMark1, PlayV($V0)

[Background1]
WM1: PB=0x1                          ; 當 QIO 發生，且 Mark 編號為 1，即會執行 WM1。
WM2: PB=0x2                          ; 當 QIO 發生，且 Mark 編號為 2，即會執行 WM2。
WM3: PB=0x3                          ; 當 QIO 發生，且 Mark 編號為 3，即會執行 WM3。
WM4: PB=0x0                          ; 當 QIO 發生，且 Mark 編號為 4，即會執行 WM4。
```

5.43.5 Melody Mark State

[NY5 / NY5+ / NY6 / NY7 / NX1]

使用者在播放 Melody 文件時，可以由 *Cakewalk* 或 *Q-MIDI* 來插入 Mark 來標記 Melody Mark。Melody Mark 標記最多可有 M1 至 M255。當播放 Melody 時，可以藉由 Melody Mark State 來切換目前要對應的 Melody Mark 的路徑。最多可定義 255 個 Melody Mark State。

```
[Melody Mark]
;                M1        M2        M3        M4
MelodyMark1:     MM1       MM2       MM3       MM4

[Path]
PowerOn: MelodyMark1, PlayM($M0)

[Background1]
MM1: PB=0x1                    ; 當讀取到 Melody Mark 時，且 Mark 編號為 1，即會執行 MM1。
MM2: PB=0x2                    ; 當讀取到 Melody Mark 時，且 Mark 編號為 2，即會執行 MM2。
MM3: PB=0x3                    ; 當讀取到 Melody Mark 時，且 Mark 編號為 3，即會執行 MM3。
MM4: PB=0x0                    ; 當讀取到 Melody Mark 時，且 Mark 編號為 4，即會執行 MM4。
```

5.43.6 Note On State

[NY5 / NY5+ / NY6 / NY7 / NX1]

使用者在播放 Melody 文件時，若不使用 Cakewalk 或 *Q-MIDI* 來插碼。可以透過 **[Note On]** 來執行每個音符被讀取時，會執行的動作。藉由 NoteOn State 來切換目前要對應的路徑，最多可定義 255 個 NoteOn State。

- NY5 針對 Ch0 ~ Ch3。
- NY5+針對 Ch1 ~ Ch4、Ch10。
- NY6 針對 MIDI 通道 Ch1 ~ Ch6、Ch10。
- NY7 針對 MIDI 通道 Ch1 ~ Ch16。
- NX1 針對 MIDI 通道 Ch1 ~ Ch16。

例.

```
[Note On]
;    CH1          CH2          CH3          CH4
CH1: CH1_ON       X            X            X
CH2: X            CH2_ON       X            X

[Path]
TR1: PlayM($M0)                          ; 播放 Melody。
TR2: SWITCH(R0)=[NO1, NO2], R0=0, TR2    ; 切換 Note On 狀態。
NO1: CH1, R0=1                           ; 設定 Note On State=CH1。
NO2: CH2, R0=0                           ; 設定 Note On State=CH2。
CH1_ON: BG1                              ; 執行 CH1 的 Note On 時，呼叫 BG1。
CH2_ON: BG2                              ; 執行 CH2 的 Note On 時，呼叫 BG2。

[Background1]
BG1:PB=[ X X X 1], DELAY(0.08), PB=[ X X X 0]

[Background2]
BG2:PB=[ X X 1 X], DELAY(0.08), PB=[ X X 0 X]
```

5.43.7 PWM-IO Mark State

[NY5+]

當使用者使用 Q-Visio 編輯檔案時，可以隨意在 action 檔案中插入 Mark 標記。當 Q-Code 使用 PlayPWM 播放時讀到標記，會依照當前的 PWM-IO Mark 狀態自動執行對應的路徑。使用者可以在 **[PWM-IO Mark]** 段落中定義 PMW-IO Mark 的狀態。

PWM-IO Mark 的標記編號最多可有 255 個，從 A1 至 A255。最多可以定義 255 組設定。

例. 在 Q-Visio 中，插入 A1~A4 的碼。

```
[PWM-IO]
SPIO0 = [VIO0.A1]
[PWM-IO Mark]
;                A1      A2      A3      A4
PWMIOMark_0:  PM1     PM2     PM3     PM4
[Path]
PowerOn: PWMIOMark_0, PlayPWM(PA.0, Ch1, $SPIO0)
[Background1]
PM1: PB=0x1                    ; 當讀取到 PWM-IO Mark，且 Mark 編號為 1，即會執行 PM1。
PM2: PB=0x2                    ; 當讀取到 PWM-IO Mark，且 Mark 編號為 2，即會執行 PM2。
PM3: PB=0x3                    ; 當讀取到 PWM-IO Mark，且 Mark 編號為 3，即會執行 PM3。
PM4: PB=0x0                    ; 當讀取到 PWM-IO Mark，且 Mark 編號為 4，即會執行 PM4。
```

5.43.8 QFID State

[NY5+ / NY7 / NX1]

用於改變現在 QFID 反應的狀態。使用者可以在 **[QFID]** 中設定不同的 QFID State，藉以控制不同 QFID Tag 偵測的需求。

例.

```
[QFID]
QFIDGroup0_0: TR1    TR2    X /TR4
[Path]
PowerOn: QFIDGroup0_0                    ; QFID State 設置成 QFIDGroup0_0 狀態。
```

5.43.9 Key_CLR

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

用於清除當前按鍵狀態，當按鍵狀態被清除後，按鍵會重新掃描。

例.

[Input_State]

KEY1：TR1　TR2

[Path]

PowerOn: KEY1　　; Input State 設置成 KEY1 狀態。

TR1: PlayV(Ch0, $V0), KEY_CLR

; 按下 TR1 後，執行 PlayV(Ch0, $V0)。執行完 PlayV(Ch0, $V0)後，執行 Key_CLR 指令，即清除當前的按鍵的狀態，重新掃描按鍵狀態。當 V0 播放完了之後，如果 TR1 有被按住時，程式會繼續執行 PlayV(ch0, $V0)，如果 TR1 沒有被按住時，則程式進入 Sleep 狀態。

5.43.10 Key_ON

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

開啟按鍵掃描功能。若是開啟該功能，則按鍵會持續做掃描。

注意：系統預設值為 Key_ON，PowerOn 後無須另外再下達一次 Key_ON 指令。

5.43.11 Key_OFF

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

關閉按鍵掃描功能。若是關閉該功能，則按鍵不做掃描。

例.

[Input_State]

KEY1: TR1

[Path]

PowerOn: KEY_OFF, DELAY(0.3), KEY1, KEY_ON

; PowerOn 時，將按鍵掃描功能關閉，待延遲 0.3 秒後，才開啟按鍵掃描。

5.43.12 Stop

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

停止當前所有動作包含 PlayV、PlayM、Delay、PlayA、PlayPWM、PWMOut 及所有背景動作。

注意：NY5 中使用 Stop 指令後，中斷亦會被關閉。

例.

[Path]

PowerOn: PlayV(ch0,$V0) & [BG1, BG2]

TR1: Stop　　; 停止全部的播放，Delay 及背景動作。

[Background1]

BG1: .....

```
[Background2]
BG2: …..
```

5.43.13 Pause(n)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

可以暫停指定的前景、背景 1、背景 2 或背景 3 的所有動作，包含 PlayV、PlayA、PlayM、Delay、PlayPWM / PlayPWMS、PWMOut / PWMOutS。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，則暫停所有的執行。

例.

```
[Path]
TR1: Pause(1)                          ; 暫停背景 1 的所有動作。
[Background1]
BG1: PlayV(Ch0, $V0), Delay(5)         ; BG1 執行 PlayV(Ch0, $V0)及 5 秒時間延遲。
```

5.43.14 Resume(n)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

可以恢復指定的前景、背景 1、背景 2 或背景 3.的所有動作，包含 PlayV、PlayA、PlayM、Delay、PlayPWM / PlayPWMS、PWMOut / PWMOutS。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，則恢復所有的執行。

例.

```
[Path]
TR1: Pause(1)                          ; 暫停背景 1 的所有動作。
TR2: Resume(1)                         ; 恢復背景 1 的所有動作。
[Background1]
BG1: PlayV(ch0, $V0), Delay(5)         ; BG1 PlayV(Ch0, $V0)及 5 秒時間延遲。
```

5.43.15 ReadChannel(Rj:Ri) / ReadChannel(Xi)

[NY5 / NY5+ / NY6 / NY7]

將目前的通道的使用狀況，存放到 RAM。

例.

```
[Path]
TR1: PlayV(CH2, $V0)                   ; 播放 Voice。
TR2: ReadChannel(X0), [PC,PB]=X0       ; 將 Channel 的使用狀況輸出到 PC 與 PB。
```

5.43.16 PauseDown

[NY5]

在執行完 PauseDown 指令後，會立即做 Ramp Down 動作。執行完 PauseDown 指令後，僅能使用 ResumeUp 指令來恢復播放。(僅對 Channel2 有效)

注意：

1. ***PauseDown 指令僅對於 Channel2 有效，僅有 Channel0 / Channel1 播放的話無法暫停播放。若下達 PauseDown 指令時，Channel0 / Channel1 正在播放的話會一併暫停。***
2. ***使用 PauseDown 指令，可以進入 Sleep。***

例.

[Path]

```
TR1: PlayV(ch2, $V0)          ; 播放 Voice。
TR2: PauseDown                ; 暫停播放，且 IC 會進入 Sleep。
TR3: ResumeUp                 ; 恢復播放。
```

5.43.17 ResumeUp

[NY5]

恢復 PauseDown 指令所暫停的播放動作。ResumeUp 指令僅能恢復使用 PauseDown 指令所暫停的播放。

例.

[Path]

```
TR1: PlayV(ch2, $V0)          ; 播放 Voice。
TR2: PauseDown                ; 暫停播放，且 IC 會進入 Sleep。
TR3: ResumeUp                 ; 恢復播放。
```

5.43.18 Audio_Loop_On

[NX1]

Audio_Loop_On 指令用於開啟指定語音通道循環播放功能。

Audio_Loop_On(Ch)

Ch：指定要開啟循環播放的語音通道。

- NX1 支援的語音通道為 Ch0 ~ Ch7。

5.43.19 Audio_Loop_Off

[NX1]

Audio_Loop_Off 指令用於關閉指定語音通道循環播放功能。

Audio_Loop_Off(Ch)

Ch：指定要關閉循環播放的語音通道。

- NX1 支援的語音通道為 Ch0 ~ Ch7。

5.43.20 Audio_ON

[NY5 / NY5+ / NY6 / NY7]

開啟 Audio 輸出。

注意：

1. 系統預設值為 Audio_ON，PowerOn 後無須另外再下達一次 Audio_ON 指令。

2. 若使用 DAC 或 Push-Pull 輸出，則需注意在恢復 Audio 輸出時，可能會有“ bo”聲。

5.43.21 Audio_OFF

[NY5 / NY5+ / NY6 / NY7]

關閉 Audio 輸出。

注意：

1. 若將為 Audio 設置為 OFF，將會一直維持直到再下達一次 Audio_ON 指令。播放音檔才會有聲音輸出。

2. 若使用 DAC 或 Push-Pull 輸出，則需注意在關閉 Audio 輸出時，可能會有“ bo”聲。

3. 當使用 PWM / DAC 輸出時，呼叫 Audio_OFF 會使 PWM / DAC 輸出腳位成為 floating 狀態。

例.

```
[Path]
TR1: PlayV(Ch0,$V0)          ; 播放 Voice。
TR2: Audio_ON                ; 開啟聲音輸出。
TR3: Audio_OFF               ; 關閉聲音輸出。
```

5.43.22 AudioMode=n

[NY5+ / NY6 / NY7]

選擇聲音輸出的模式。

n：聲音輸出的模式，可使用 DAC / PWM / PP。

- NY5+僅能提供 DAC 與 PWM 輸出。
- NY6A 僅能提供 PWM 輸出。
- NY6B / NY6C 提供 DAC 與 PWM 輸出。
- NY7A 提供 DAC 與 PWM 輸出。
- NY7B / NY7C 提供 DAC 與 Push-Pull 輸出。

例. NY7A

```
[Path]
TR1: AudioMode=DAC           ; 設定成 DAC 輸出。
TR2: AudioMode=PWM           ; 設定成 PWM 輸出。
TR3: PlayV(Ch0, $V0)         ; 播放音檔。
```

例: NY7B / NY7C

[Path]

```
TR1: AudioMode=DAC                    ; 設定成 DAC 輸出。
TR2: AudioMode=PP                     ; 設定成 Push-Pull 輸出。
TR3: PlayV(Ch0, $V0)                  ; 播放音檔。
```

5.43.23 NoiseFilter_ON(Ch)

[NY6 / NY7 / NX1]

開啟雜訊濾波功能。

NoiseFilter_On

NoiseFilter_On(Ch)

Ch：語音通道，若不指定則開啟所有語音通道的雜訊濾波功能。

- NY6 支援 0 ~ 5 或 Ch0 ~ Ch5。
- NY7 / NX1 不支援指定語音通道。

注意：

1. *該功能一經開啟後，直到關閉前會一直維持開啟的狀態。*
2. *此功能不允許在聲音播放中使用。*
3. *預設為開啟。*

5.43.24 NoiseFilter_OFF(Ch)

[NY6 / NY7 / NX1]

關閉雜訊濾波功能。

NoiseFilter_Off

NoiseFilter_Off(Ch)

Ch：語音通道，若不指定則關閉所有語音通道的雜訊濾波功能。

- NY6 支援 0 ~ 5 或 Ch0 ~ Ch5。
- NY7 / NX1 不支援指定語音通道。

注意：

1. *該功能一經開啟後，直到關閉前會一直維持開啟的狀態。*
2. *此功能不允許在聲音播放中使用。*

例: 若語音聽起來有背景雜音時，可以藉由開啟 Noise Filter 來進行降噪。

[Path]

```
TR1: PlayV(Ch0,$V0)                   ; 播放 Voice。
TR2: NoiseFilter_ON                   ; 開啟 Noise Filter。
TR3: NoiseFilter_OFF                  ; 關閉 Noise Filter。
```

5.43.25 EQ_Filter

[NX1]

開啟音頻濾波器濾波功能並指定濾波器參數。

EQ_Filter(Ch, Index)

Ch：語音通道。

- NX1 支援 Ch0 ~ Ch7。

Index：音頻濾波器參數索引。

- 0 ~ 濾波器索引最大值。

注意：

1. 請搭配音頻濾波器 (Audio Filter) 進行參數設置。

2. 此功能僅支援 SBC-1 / ADPCM 使用。

例. 若希望語音使用音頻濾波器，可以藉由設定 EQ_Filter 來實現。

```
[Option]
Custom_Filter = Filter.fnx                ; 加入 Audio Filter.
[Path]
TR1: PlayV(Ch0, $V0)                      ; 播放 Voice。
TR2: EQ_Filter(Ch0, 1)                    ; 套用 index=1 的濾波器參數。
```

5.43.26 EQ_Filter_Off

[NX1]

關閉音頻濾波器的濾波功能。

EQ_Filter_Off(Ch)

Ch：語音通道。

- NX1 支援 Ch0 ~ Ch7。

注意：必須有使用 EQ_Filter 方可使用 EQ_Filter_Off。

5.43.27 AGC_On

[NX1]

開啟 AGC 功能。在 *Q-Code* 使用 VR 或 Record 指令時，AGC(Auto Gain Control)可自動調節麥克風增益，避免音量過大造成訊號飽和的情形，可提升 VR 辨識率與減少 Record 雜音的狀況。

注意：若專案包含 VR，且 IC Body 不為 NX12P44，預設為開啟。

5.43.28 AGC_Off

[NX1]

關閉 AGC 功能。在 *Q-Code* 使用 VR 或 Record 指令時，AGC(Auto Gain Control)可自動調節麥克風增益，避免音量過大造成訊號飽和的情形，可提升 VR 辨識率與減少 Record 雜音的狀況。

注意：若專案包含 VR，且 IC Body 不為 NX12P44，預設為開啟。

例. 若錄音在大音量時聽起來有雜音，可藉由開啟 AGC 降低雜音發生。

[Path]

```
TR1: Record($Rec0,5s)                ; 開始錄音
TR2: PlayV(CH0,$Rec0)                ; 播放錄音內容
TR3: AGC_On                          ; 開啟 AGC
TR4: AGC_Off                         ; 關閉 AGC
```

5.43.29 RampUp

[NY5 / NY5+ / NY6 / NY7 / NX1]

在 *Q-Code* 要播放 Voice 檔或是 Melody 檔時，系統會自動幫 User 做 RampUp 的動作。但若是 User 有特殊的需求，而必須先做 RampUp 這個動作，在這邊我們也提供了 RampUp 的指令以供 User 來使用。

注意：

1. *若已經執行過 RampUp 指令後，則在 IC 未進入睡眠前或執行 RampDn 指令前，系統不會再執行 RampUp 動作。*
2. *NY5 / NY5+ / NY6 / NY7 在延遲過程中無法處理其它的動作，僅能在延遲結束後執行下一個步驟。*
3. *NX1 在執行 RampUp 後，會立即執行下一道指令，RampUp 執行時間由選項 Ramp Up Time 決定。*

例. **RampUp, Delay(1), PlayV…..**

; 執行過 RampUp 後，PlayV/PlayM 不會再執行 RampUp 動作。

例. 有 OP Amp 應用時，且有控制腳位。

[Path]

```
TR1: RampUp, PB.3=0, PlayV…..
; 執行完 RampUp 後，將 OP Amp 致能。播放音檔時就不會有“bo”聲。
```

5.43.30 RampDown

[NY5 / NY5+ / NY6 / NY7 / NX1]

在 *Q-Code* 系統中，當聲音或者是 Melody 檔播放完畢後會直接幫 User 做 RampDown 的動作。但若是 User 有特殊的需求，仍可使用指令來達成。

注意：

1. *若是在進 Sleep 前，會有一段延遲時間的話，則會造成耗電或者是有雜音的問題。在這邊我們提供了 RampDown 指令讓 User 可以自行關閉 Audio Output，以解決耗電及雜音的問題。*
2. *NY5 / NY5+ / NY6 / NY7 在延遲過程中無法處理其它的動作，僅能在延遲結束後執行下一個步驟。*
3. *NX1 在執行 RampDown 後，會立即執行下一道指令，RampDown 執行時間由選項 Ramp Down Time 決定。*

*例.***PlayV….. , RampDown, Delay(15)**

; 播放完畢後仍會延遲 15 秒才會進入睡眠模式，此時必須要先做 RampDown 動作，來關閉聲音輸出以

避免“bo”聲及耗電。

5.43.31 AutoSleep_On

[NX1]

Q-Code 在 IC 閒置時會自動進入睡眠模式。

注意：系統預設值為 AutoSleep_ON，PowerOn 後無須另外再下達一次 AutoSleep_ON 指令。

5.43.32 AutoSleep_Off

[NX1]

Q-Code 在 IC 閒置時不進入睡眠模式。

注意：若將 AutoSleep 設置為 OFF，IC 將不會進入睡眠，直到再下達一次 AutoSleep_ON 指令。

5.43.33 Sleep

[NX1]

設定當系統不動作時，IC 的睡眠模式。

Sleep(Mode)

Mode：IC 睡眠模式。

- NX1 支援 Halt / Standby。

5.43.34 WDT_CLR

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

WDT_CLR（Watch Dog Clear）是為了要清除 Watch Dog Counter 用。若是在 *Q-Code* 裡面執行過多的算數邏輯指令或是其它指令，而其間都沒有執行過 Play 或者是延遲命令，則系統可能會由於執行時間過長，會將 IC 當成是異常執行，因而 Reset IC。這時就可以透過 WDT_CLR 指令來強制 Reset Counter。

例. **WDT_CLR ; Watch Dog Clear。**

5.43.35 Repeat

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

提供 repeat function 讓 user 可以藉由 repeat function 來達到省 Code size 的目的。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 總共提供 3 組，分別對於前景及兩個背景都可獨立使用而不互相干擾。
- NX1 總共提供 4 組，分別對於前景及三個背景都可獨立使用而不互相干擾。

{…} *n

n: 重覆次數

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支援 1 ~ 15。
- NX1 支援 1 ~ 255。

例. 單一個 section repeat 時，利用 LOOP counter 來達到 repeat 功能。

```
{ PlayV($V0), PlayV($V1), PlayV($v2) } * 3     ; “{“…. “}” 號內的程式會執行 3 次。
```

5.43.36 Background

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

執行指令時一併啟動背景路徑。

… & [BG1]
… & [BG1, BG2]
… & [BG1, BG2, BG3]
… & [BG1, BG2, BG3, BG4]
… & [BG1, BG2, BG3, BG4, BG5]

BG1：呼叫 Background1 的程式。

- BG1 可以呼叫 [**Background1**] 底下的動作。
- X：不改變當前 Background1 path 的動作。
- OFF：停止當前 Background1 path 的動作。

BG2：呼叫 Background2 的程式。

- BG2 可以呼叫 [**Background2**] 底下的動作。
- X：不改變當前 Background2 path 的動作。
- OFF：停止當前 Background2 path 的動作。

BG3：呼叫 Background3 的程式（NX1 Only）。

- BG3 可以呼叫 [**Background3**] 底下的動作。
- X：不改變當前 Background3 path 的動作。
- OFF：停止當前 Background3 path 的動作。

BG4：呼叫 Background4 的程式（NX1 Only）。

- BG3 可以呼叫 [**Background4**] 底下的動作。
- X：不改變當前 Background4 path 的動作。
- OFF：停止當前 Background4 path 的動作。

BG5：呼叫 Background5 的程式（NX1 Only）。

- BG3 可以呼叫 [**Background5**] 底下的動作。
- X：不改變當前 Background5 path 的動作。
- OFF：停止當前 Background5 path 的動作。

例.

```
PlayV($V0) & [BG1, BG2]                    ; 播放 V0 時一併執行 BG1 及 BG2。
```

5.43.37 Slow

[NY5+ / NY6 / NY7]

切換 Sleep 模式為 Slow 模式，此模式可用於長時間的計時或是需要固定時間檢查訊號的應用。當使用者執行此指令後，原本的 Sleep 模式將會由 slow 模式來取代，系統固定時間會自動喚醒一次，喚醒後會執行 WakeUp 路徑一次，而在進入 Slow 模式前也會執行 Sleep 路徑一次。使用者可使用 SlowOff 指令來關閉 Slow 模式，恢復成 sleep 模式。Slow 模式喚醒的時間為中斷時間的 16 倍，即中斷 256us 會在 4ms 喚醒一次，中斷 1ms 會在 16ms 喚醒一次。

例. 低速模式切換。

[Path]

```
TR1: Slow                    ; 切換成低速模式。
```

5.43.38 SlowOff

[NY5+ / NY6 / NY7]

用來關閉 Slow 模式，恢復成 Sleep 模式。

例. 關閉低速模式。

[Path]

```
TR1: SlowOff                 ; 關閉低速模式。
```

5.43.39 END

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1]

在此的 END 指令並非強制停止目前所有正在執行的動作。若碰到 END 指令後，後面的指令將不會被執行。當所有的程序全部皆執行完畢的話，系統才會允許進入睡眠模式。若是使用者需要完全停止目前正在執行的所有動作，並且進入睡眠模式，可以使用 Stop 指令來停止有目前正在執行的動作。

注意：系統編譯完後會自動在每行結束加上"END"指令，故可以不需要在每行結束時下達"END"指令。

例. **PlayV….. , RampDown, Delay(15), END**

5.43.40 ReadRollingCode

[NY4 / NY5 / NY5+ / NY6 / NY7 / NX1]

滾動碼（Rolling Code）為 20-bit 流水號，每次在寫入 OTP 時會自動增加，可賦予每一個 OTP 獨特的 ID。使用此指令可取得寫入時賦予的流水號，並存放到暫存器。暫存器存放的內容為：Ri=bit0~3、Rj=bit4~7、Rk=bit8~11、Rl=bit12~15、Rm=bit16~19。

ReadRollingCode(Rm:Rl:Rk:Rj:Ri)

例. NY4 / NY5 / NY5+ / NY6 / NY7

[Path]

TR1: ReadRollingCode(R4:R3:R2:R1:R0) ；將讀出的 rolling code 存到 R0~R4。

例. NX1

[Variable]

Var32: Code

[Path]

TR1: ReadRollingCode(Code) ；將讀出的 rolling code 存到 Code。

5.43.41 ChMode(n)

[NY6 / NY7]

設定通道數量。

n：通道數量。

- NY6 支援 2 / 4 / 6（預設為 6）
- NY7 支援 2 / 4 / 6 / 8。（預設為 8）

注意：

1. 當語音或 melody 播放中，切換通道數會造成採樣率錯誤，因而播放異常。請避免在播放中切換通道數。

2. NY6 使用 6 通道，語音採樣率最高僅支援 41.6KHz，其他通道數皆可達 44.1KHz。

例.

ChMode(4), PlayV(Ch1,$V1) ；設定使用 4 通道來播放 V1。

音質	可用通道編號
高品質	0 ~ 1
優良	0 ~ 3
中等	0 ~ 5
一般	0 ~ 7

5.43.42 ReadRadj(Ri)

[NY9T]

讀取外部電阻值，指令會回傳 0x0~0x7 數值，若沒有接外部電阻值時，則回傳 0xF，此讀取行為並不會直接影響觸摸鍵的靈敏度，使用者可以根據讀取的數值，自行定義如何對應到 8 階的靈敏度設定。

建議電阻值（±1%）	Level	Ri
750K	0	0x0
360K	1	0x1
180K	2	0x2
120K	3	0x3
82K	4	0x4
51K	5	0x5
33K	6	0x6

建議電阻值（±1%）	Level	Ri
16K	7	0x7
Other	-	0xF

*注意：若不使用外部電阻調整靈敏度功能時，需將 **Radj** 腳位接到 **VDD**。*

例.

[Path]

PowerOn: ReadRadj(R0), Switch(R0)=[Sens0, Sens1, Sens2, Sens3] ；將外部電阻的階數讀出。

Sens0: TouchKey_Sensitivity(0) ；設定靈敏度最高。

Sens1: TouchKey_Sensitivity(2) ；設定靈敏度介於一般和最高之間。

Sens2: TouchKey_Sensitivity(4) ；設定靈敏度一般。

Sens3: TouchKey_Sensitivity(7) ；設定靈敏度最低。

5.43.43 SW_Reset

[NY9T]

SW_Reset（Software Reset）可恢復成上電時的初始狀態。與硬體 Reset 不同處在於 IC 不會重新讀取硬體的初始設定，僅是重置 RAM 及 IO 狀態。該指令被執行後，程式會回到啟始進入點，SW_Reset 後面的指令將不會被執行。

*注意：在執行 **Enforce_Calibrate** 後，若想要重置整個系統可利用 **SW_Reset** 指令或是在 **Option** 中選用 **SW_Reset = Enable**。*

例. **Enforce_Calibrate: SW_Reset** ；**Software Reset**。

5.43.44 Debounce(Time)

[NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T]

Debounce 指令提供給使用者一種方法同時改變 Direct 及 Matrix 按鍵掃描的 Debounce 時間。Time 單位：ms（毫秒）或 sec（秒）（預設為秒），最小值=0ms，最大值=1s。

注意：

1. ***Debounce(Time)指令會同時影響 Direct 及 Matrix 按鍵。***
2. ***NY9T 中，Debounce option 與 Debounce(Time)指令，只能擇一使用，且時間須為 4ms 的倍數。***

例.

[Path]

PowerOn: KEY1, KEY_ON

TR1: Debounce(16ms) ；設定 **Direct** 及 **Matrix** 按鍵的 **Debounce** 時間為 **16ms**。

TR2: Debounce(0.06) ；設定 **Direct** 及 **Matrix** 按鍵的 **Debounce** 時間為 **60ms**。

5.43.45 Direct_Debounce(Time)

[NY5+ / NY6 / NY7]

Direct_Debounce 指令提供給使用者一種方法來改變 Direct 按鍵掃描的 Debounce 時間。Time 單位：ms（毫秒）或 sec（秒）（預設為秒），最小值=0ms，最大值=1s。

注意：Direct_Debounce(Time)指令僅對 Direct 按鍵有效，不影響 Matrix 按鍵。

例.

[Path]

PowerOn: KEY1, KEY_ON

TR1: Direct_Debounce(16ms) ; 設定 Direct 按鍵的 Debounce 時間為 16ms。

TR2: Direct_Debounce(0.06) ; 設定 Direct 按鍵的 Debounce 時間為 60ms。

5.43.46 Matrix_Debounce(Time)

[NY5+ / NY6 / NY7]

Matrix_Debounce 指令提供給使用者一種方法來改變 Matrix 按鍵掃描的 Debounce 時間。Time 單位：ms（毫秒）或 sec（秒）（預設為秒），最小值=0ms，最大值=1s。

注意：Matrix_Debounce(Time)指令僅對 Matrix 按鍵有效，不影響 Direct 按鍵。

例.

[Path]

PowerOn: KEY1, KEY_ON

TR1: Matrix_Debounce(16ms) ; 設定 Matrix 按鍵的 Debounce 時間為 16ms。

TR2: Matrix_Debounce(0.06) ; 設定 Matrix 按鍵的 Debounce 時間為 60ms。

5.43.47 Ri = LVD

[NY5+ / NY6B / NY6C / NX1]

讀取 LVD 的階數，並將 LVD 的階數存入指定的 RAM Ri。

階數的對應如下：

階數	NY4PxxxC / NY5+	NY6P025A LVD1	NY6P025A LVD2	NY6B / NY6C	NX1 OTP	NX1 EF
1	<2.0V	<2.4V	<2.0V	<2.4V	<2.2V	<2.0V
2	2.0V ~ 2.2V	2.4V ~ 2.8V	2.0V ~ 2.2V	2.4V ~ 2.7V	2.2V ~ 2.4V	2.0V ~ 2.2V
3	2.2V ~ 2.4V	2.8V ~ 3.6V	2.2V ~ 3.0V	2.7V ~ 3.6V	2.4V ~ 2.6V	2.2V ~ 2.4V
4	2.4V ~ 2.8V	3.6V ~ 4.1V	3.0V ~ 3.2V	3.6V ~ 4.1V	2.6V ~ 3.2V	2.4V ~ 2.8V
5	2.8V ~ 3.0V	>4.1V	>3.2V	>4.1V	3.2V ~ 3.4V	2.8V ~ 3.0V
6	3.0V ~ 3.3V				3.4V ~ 3.6V	3.0V ~ 3.2V
7	3.3V ~ 3.6V				> 3.6V	3.2V ~ 3.4V
8	>3.6V					3.4V ~ 3.6V
9						> 3.6V

例. 將 LVD 的階數存入 R0。

TR1: R0 = LVD ; 將 LVD 的階數存到 R0。

5.43.48 Enforce_Wakeup

[NX1]

Enforce_Wakeup 指令提供給使用者於 Standby Mode 時，從 RTC_2Hz 路徑喚醒系統的方法。

注意：Enforce_Wakeup 指令僅於 Standby Mode 條件下 RTC_2Hz 路徑內有效。

例.

```
[Variable]
Var8: event_trig=0, count_500ms=0
[Path]
PowerOn:
main: event_trig=1?TR1                          ; 確認喚醒事件進行播音。
TR1: event_trig=0, PlayVS(CH0,$V0)
[Interrupt]
RTC_2Hz: count_500ms++, count_500ms>=10?Per5Sec
Per5Sec: count_500ms=0, event_trig=1, Enforce_Wakeup  ; 每 5 秒喚醒一次。
```

5.43.49 GetMicVol

[NX1]

GetMicVol 指令透過擷取麥克風 ADC 讀值，進行估測音量資訊提供。

GetMicVol(Var)

Var：將讀取的結果存放於 Var，結果範圍為 0~65535。

注意：

1. 僅於麥克風運行期間能取得數值，麥克風未運行時會得到 0。

2. 依當前使用環境與麥克風靈敏度等因素，會有 5%不等的底噪，此音量資訊僅供粗略的趨勢參考。

例.

```
[Variable]
Var16: var_mic=0
[Path]
PowerOn: RT_Play                                ; 透過 RT 啟動麥克風。
8ms: GetMicVol(var_mic),                        ; 每 8ms 讀取一次麥克風估測音量。
     var_mic>6000?{PA.1=0}:{PA.1=1}             ; 音量大於 6000 設置 PA.1=0，反之 PA.1=1。
```

5.43.50 Millis

[NX1]

Millis 指令透過讀取 NX1 計數資訊取得初始化後的運行時間 (毫秒)，資料長度為 Var32。

Millis(Var)

Var：將讀取的結果存放於 Var。

注意：睡眠時將不會繼續計數。

例.

[Variable]

Var32: millis_var

[Path]

PowerOn: Millis(millis_var) **；取得運行時間。**

5.43.51 Srand

[NX1]

Srand 會設定用於產生一系列亂數的起始點。

Srand(Seed)

Seed：亂數種子。

5.44 偵錯指令（Debug Command）

Debug Command				
Printf	-	-	-	-

5.44.1 Printf

[NX1]

用於透過 UART TX 硬體輸出除錯訊息，使用方式如 C 語言的 printf。

Printf(Format_str, Args)

Format_str：訊息字串。

Args：變數資訊，數量可隨需求增減，可不使用。

注意：

1. 需開啟 UART TX 功能才可使用此指令。

2. Format_str 中支援的變數輸出型別有%o、%d、%u、%x 以及%X。

3. Q-Code 不會檢查 Format_str 中使用的變數的型別與數量，使用者需自行注意。

例.

[Variable]

Var32: cnt = 0

[Path]

PowerOn: Printf("Hello\r\n") **；輸出 Hello。**

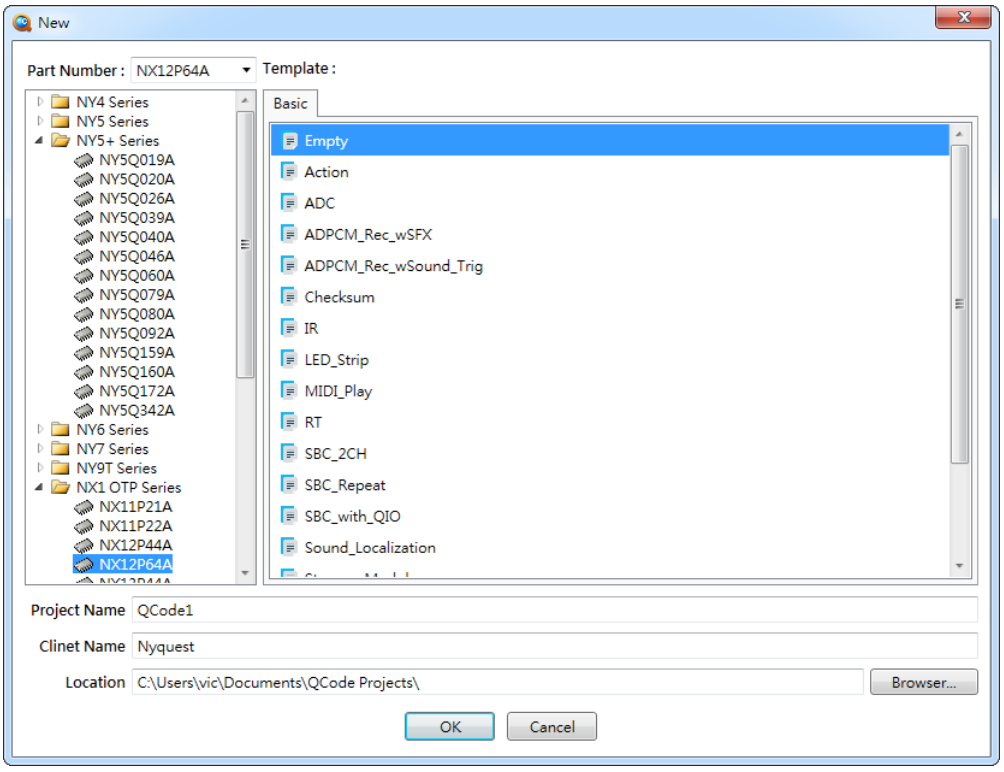
8ms: cnt++ **；每 8ms 累加一次 cnt。**

500ms: Printf("cnt%d\r\n", cnt) **；每 500ms 輸出當前 cnt 的值。**

6 附錄

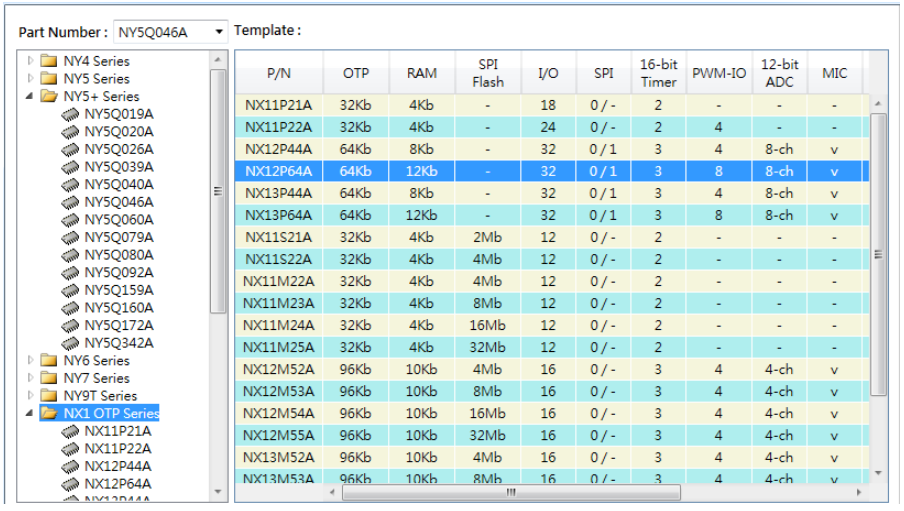
6.1 新增檔案視窗

[檔案] → [新增檔案]



Part Number：選擇 IC 的型號。

Template：選擇要使用的功能樣本，建立檔案的內容。點擊 IC 的型號，則右方顯示樣板選單如上圖。若點擊系列目錄，則右方會顯示系列產品資訊如下圖。



Project Name：輸入專案名稱。

Client Name：輸入客戶名稱，以保護客戶的權益。

Location：檔案的目錄。

6.2 相關工具介紹

6.2.1 相關軟體工具

在使用 *Q-Code* 編寫程式時，我們不僅僅只要用到這一個工具，我們還需要用到幾個相關的工具。例如，當我們在編寫完程式時，就需要編譯產生一個.bin 和.htm 檔（這是投 Code 的重要文件），此時，我們就需要用到 *NYASM*，它應該在安裝 *Q-Code* 時同步安裝。

在使用 *Q-Code* 時還可能用到以下一些相關工具：

Q-MIDI：此軟體是用來編輯 MIDI 與音色庫，產生的檔案為.qmd，可在 NY5 / NY5+ / NY6 / NY7 / NX1 系列中使用。

Quick-IO：此軟體是用來畫出輸出腳的信號，需結合.wav 檔使用，所產生的檔案為.nyq。

Q-Visio：此軟體是用來畫出輸出腳的信號，所產生的檔案為.vio。

Q-Touch：此軟件是用來掃描觸摸鍵的靈敏度，所產生的文件為.t9x。

Q-Writer：此軟體是用來將.bin 檔案燒錄在 Flash Demo Board、Romter 或 OTP 上以供驗證。

注意：在安裝 Q-Code 時，最好是將用到的相關工具都同步安裝，以上工具的使用請參考相關的使用手冊。

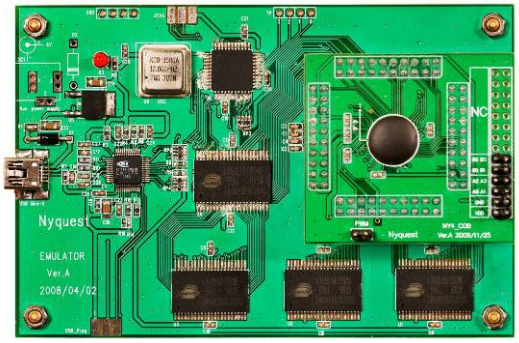
6.2.2 相關硬體工具

當程式編寫完成並編譯後，使用者可以將程式下載至 ICE 上進行驗證，或者通過燒錄器（FDB_Writer 或 Q-FDB_Writer）燒錄至 Demo 板上進行演示。

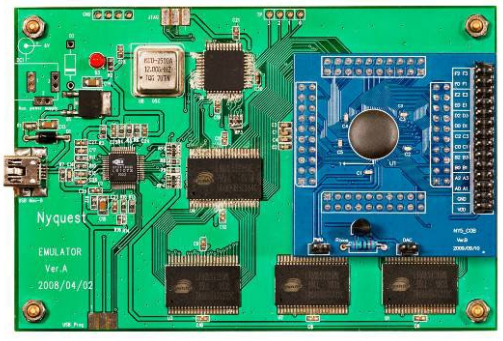
何謂 ICE? ICE 是 In Circuit Emulator 的縮寫，中文叫實體模擬器或模擬器。只需要將 ICE 通過 USB 連接至個人電腦，便可以將編譯好的程式下載至 ICE 進行驗證。（有關 ICE 硬體安裝請參考 *NYIDE* 使用手冊。）

注意：

1. ***ICE 有分為 V1 / V2（下圖中 NY4 / NY5 使用的是 ICE V1，NY7 則是 ICE V2）。***
2. ***ICE V1 僅支援 NY4 / NY5。***
3. ***ICE V2 支援 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T。***



NY4_COB on ICE V1



NY5_COB on ICE V1

NY7_COB on ICE V2

何謂 *Q-Writer*? *Q-Writer* 是一個圖形介面的燒錄系統，讓使用者能夠快速的將程式產生的.bin 檔燒錄至 FDB（Flash Demo Board）演示板中，或者下載到 Romter 驗證，也可直接透過 *Q-Writer* 直接燒錄至 OTP 來進行驗證。使用者可以用 FDB_Writer 燒錄，也可以用 *Q-FDB_Writer* 燒錄（有關 *Q-Writer* 的軟件和硬體安裝及使用請參考 *Q-Writer* 使用手冊）。

FDB_Writer：如下圖所示。

Q-FDB_Writer：如下圖所示。

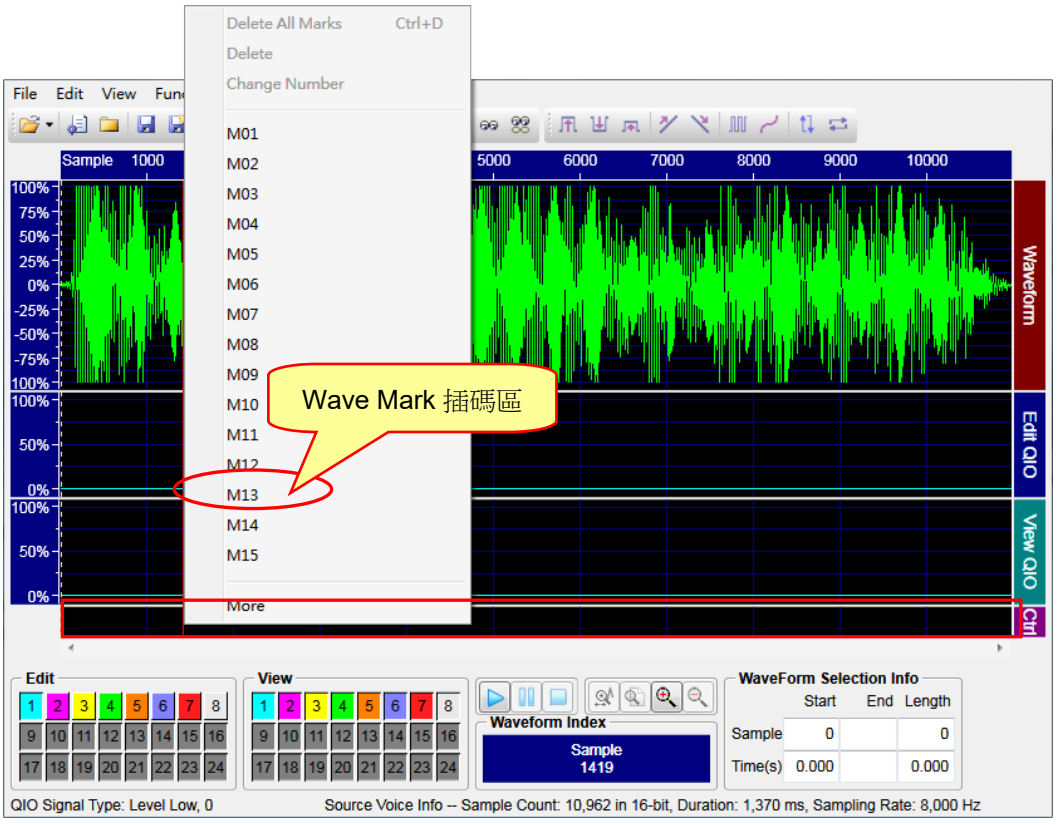
Romter：如下圖所示。

NX_Programmer：如下圖所示。

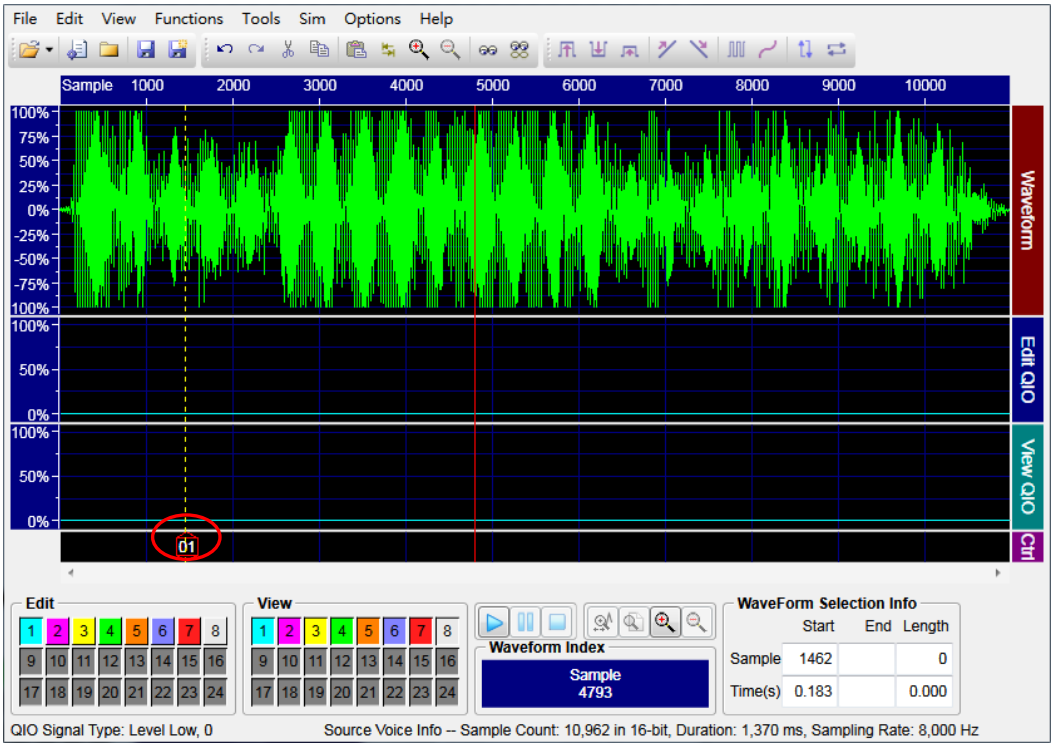
6.3 使用 *Quick-IO* 在.wav 檔中插入控制碼

在 *Quick-IO* 中，有提供插入控制碼的功能。讓使用者可以在任一個位置插入控制碼，提供 *Q-Code* 判讀。當 *Q-Code* 讀取到 Wavemark 時，會執行 **[WaveMark]** 段落中相對應的路徑。

當開啟一個.wav 文件後，將滑鼠遊標移到 Ctrl 欄，點擊右鍵，即會出現 Mark Number。



選擇好要插入的 Mark Number 後，在控制欄中會出現插入的 Mark，此時就完成一個插碼的動作。如下圖：



6.4 在 *Q-Code* 程式中使用 *Q-Sound*

在 *Q-Code* 使用 *Q-Sound* 的 Split 功能，可依據下列的步驟進行操作：

[第一步：如何開啟 ***Q-Sound*** 視窗。]

[第二步：如何開啟 **Split** 視窗。]

[第三步：設定 **Noise Gate**。]

[第四步：設定 **Minimum Mute Length**。]

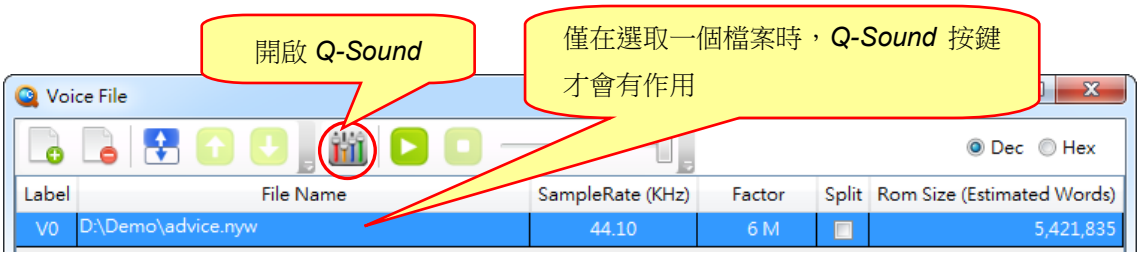
[第五步：進行 **Auto Mark**。]

[第六步：調整 **Mark**。]

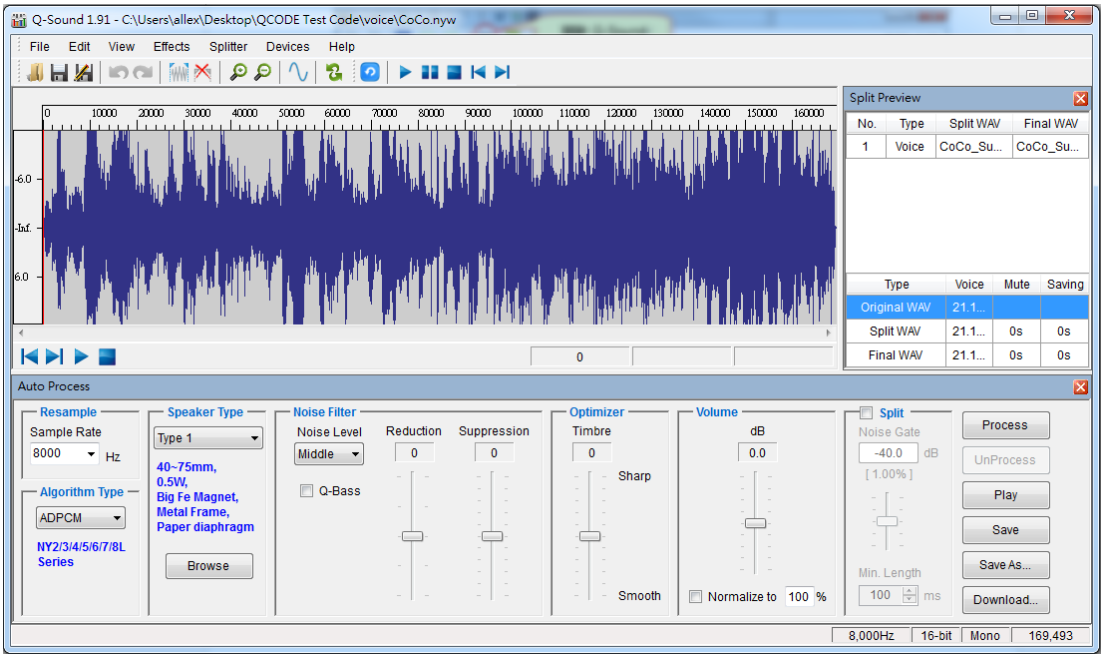
[第七步：在 ***Q-Code*** 的程式中使用 **NYW** 檔。]

第一步：如何開啟 *Q-Sound* 視窗。

在 *Q-Code* 的段落功能表中單擊 Voice File 段落中的 Add File，開啟 Voice File 對話盒。



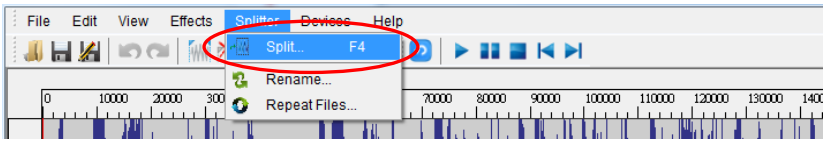
單擊  開啟 *Q-Sound*。



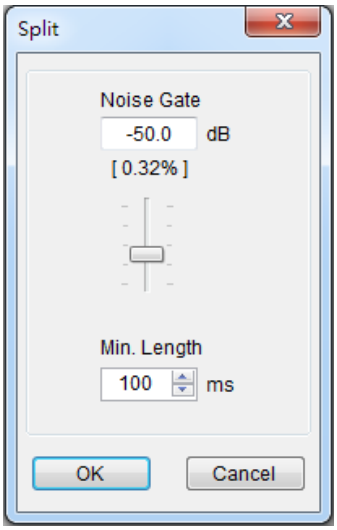
注意：支援.wav 和.nyw 二種文件格式。

第二步：如何開啟 Split 視窗。

按下功能表 [Splitter]。

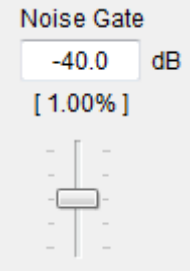


點擊 Split 開啟 Split 視窗。



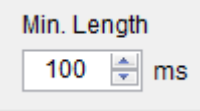
第三步：設定 Noise Gate。

調整 Noise Gate 的值，設定靜音區段音量的範圍。



第四步：設定 Minimum Mute Length。

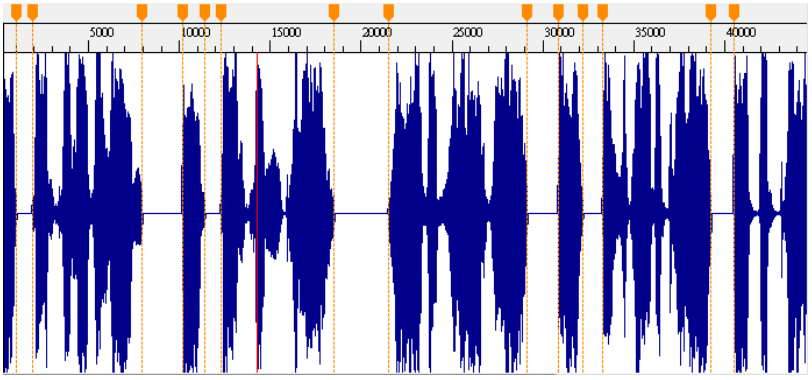
調整 Minimum Mute Length 的值，來設定靜音區段的最小長度。大於這個長度的靜音段，*Q-Sound* 才會進行處理。



第五步：進行 Auto Mark。

當設定完成後，*Q-Sound* 會依照 Noise Gate 及 Minimum Mute Length 的設定，自動插入標記。

完成後，會在以下對話框中顯示出靜音段的數量，並於 Split Preview 視窗產生預計切割的狀態。



Split Preview

No.	Type	Split WAV	Final WAV
1	Voice	Vocal_bin_dec_Sub1(134ms)	Vocal_bin_dec_Sub1(134ms)
2	Mute	112ms	112ms
3	Voice	Vocal_bin_dec_Sub2(753ms)	Vocal_bin_dec_Sub2(753ms)
4	Mute	282ms	282ms
5	Voice	Vocal_bin_dec_Sub3(147ms)	Vocal_bin_dec_Sub3(147ms)
6	Mute	110ms	110ms
7	Voice	Vocal_bin_dec_Sub4(780ms)	Vocal_bin_dec_Sub4(780ms)

若是文件的切割不符合預期，請回到第二步繼續操作。

第六步：調整 Mark。

若是想要手動調整標記，可將游標移至標記上壓住左鍵即可進行微調。

完成後，檢視 Split Preview 視窗中的文件切割狀態是否符合預期。若是不符合預期，請重複第二步和第三步的操作，直至符合預期。確認無誤後，按下儲存並關閉 *Q-Sound*。

第七步：在 *Q-Code* 的程式中使用.nyw 檔。

關閉 *Q-Sound* 回到 Voice File 對話盒，進行功能設定。（如果未勾選選 Split，即使已經開啟 *Q-Sound* 完成插碼並儲存，仍然不會進行切割。）

按 OK 關閉 Voice File 對話盒後，會在 Voice File 區塊自動產生程式碼。

```
[Voice File]
V0 = D:\Demo\advice.nyw /5 /fs /s
```

播放聲音檔"V0"的方式，與一般未使用 *Q-Sound* 處理過的音檔，採用相同的播放指令，使用者可以當成播放單一音檔的方式來處理。

```
[Path]
PowerOn:playV($v0), input_0
p1: playV($v0)
p2: playV($v0)
```

當語音文件包含相當程度的靜音，編譯（Build）使用 Split 功能的 *Q-Code* 程式，會發現佔用的 ROM Size 變小了。下圖是同一個 *Q-Code* 程式，未使用與使用 Split 功能編譯後，在 Status bar 上的 ROM Size 資訊。

注意：更詳細的 Q-Sound 功能，請參閱 Q-Sound 使用手冊。

6.5 *Q-Code* 指令表

6.5.1 NY4 *Q-Code* 指令表

6.5.1.1 算數邏輯指令（Arithmetic Command）

Arithmetic Logic Command			
Var1 = Var2	Var1 = Var2 + Var3	Var1 = Var2 – Var3	Var1 = Var2 * Var3
Var1 = Var2 / Var3	Var1 = Var2 % Var3	Var++	Var--
Var1 = Var2 & Var3	Var1 = Var2 | Var3	Var1 = Var2 ^ Var3	Var1 = Var2 << Var3
Var1 = Var2 >> Var3	!Var	Var=RandomL	Var=RandomH
Var=Random	-	-	-

6.5.1.2 流程控制指令（Flow Control Command）

Flow Control Command				
Var = Var ? Path	Var != Var ? Path	Var >= Var ? Path	Var <= Var ? Path	Var > Var ? Path
Var < Var ? Path	Px = data?Path	Px != data?Path	Px.n = 0?Path	Px.n != 0?Path
Px.n = 1?Path	Px.n != 1?Path	-	-	--
RandomL=data?Path	RandomH=data?Path	RandomL!=data?Path	RandomH!=data?Path	
Random=data?Path	Random!=data?Path	-	-	
Px[1 X 0 X]?Path		Voice?Path	PaueV?Path	Delay? Path
Action?Path	PWMIO?Path	PausePWM?Path	CheckSum?Path	
!Px[1 X 0 X]?Path		!Voice?Path	!PaueV?Path	!Delay? Path
!Action?Path	!PWMIO?Path	!PausePWM?Path		-
!CheckSum?Path		If-Else	-	-
Switch(Ri) = [Path0, Path1, Path2,..Path15]		Switch(Px) = [Path0, Path1, Path2,..Path15]		
Switch(RandomL) = [Path0, Path1,..Path15]		Switch(RandomH) = [Path0, Path1,..Path15]		
Switch(Px[d x d x]) = [Path0, Path1, Path2….Path15]				
Switch(Xi)=[Path0, Path1, Path2,..Path255]				
Switch(Random)=[Path0, Path1,Path2,..Path255]				
While	Do-While	For	-	-

6.5.1.3 I/O 指令（I/O Command）

4-bit I/O Command				
Ri = Px	Ri = PxM	Ri.n = Px.n	PxM = data	PxM = Ri
PxM.n = 0	PxM.n = 1	Px = data	Px = Ri	Px = Py
!Px	!Px.n	Px.n = 0	Px.n = 1	Px.n = Ri.n
Px.n = Py.n	Px = Py + Ri	Px = Py – Ri	Px = Py | Ri	Px = Py ^ Ri
Px = Py & Ri	Ri = Px + Rj	Ri = Px – Rj	Ri = Px | Rj	Ri = Px ^ Rj
Ri = Px & Rj	Px = Py + data	Px = Py – data	Px = Py | data	Px = Py ^ data
Px = Py & data	Ri = Px + data	Ri = Px – data	Ri = Px | data	Ri = Px ^ data
Ri = Px & data	Px = [1 x 0 FD An]	Px.n= 1KHz(time)	-	-
8-bit I/O Command				
XiL = Px	XiH = Px	XiL.n = Px.n	XiH.n = Px.n	Xi.n = Px.n
Xi = [Px, Py]	Px = XiL	Px = XiH	Px.n = XiL.n	Px.n = XiH.n
Px.n = Xi.n	[Px, Py] = Xi	-	-	-

6.5.1.4 路徑指令（Path Command）

Path Command				
ASM	BG	BreakFG	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro

6.5.1.5 語音指令（Voice Command）

Voice Command				
PlayV	PlayVS	Voicename	WaitVN(Ch)	PauseV(Ch)
ResumeV(Ch)	StopV(Ch)	FreqCH=nK-	-	-

6.5.1.6 句子指令（Sentence Command）

Sentence Command				
PlayS	-	-	-	-

6.5.1.7 查表指令（Table Command）

Table Command	
TableL(TableName, Rx/Xx, Ry/Xy, Ri)	TableM(TableName, Rx/Xx, Ry/Xy, Ri)
TableH(TableName, Rx/Xx, Ry/Xy, Ri)	Table(TableName, Rx/Xx, Ry/Xy, Rh, Rm, Rl)
TableL(TableName, X, Y, Ri)	TableM(TableName, X, Y, Ri)

Table Command	
TableH(TableName, X, Y, Ri)	Table(TableName, X, Y, Rh, Rm, Rl)
TableL(TableName, Rx/Xx, Ry/Xy, Xi)	TableH(TableName, Rx/Xx, Ry/Xy, Xi)
Table(TableName, Rx/Xx, Ry/Xy, Xh, Xl)	TableL(TableName, X, Y, Xi)
TableH(TableName, X, Y, Xi)	Table(TableName, X, Y, Xh, Xl)

6.5.1.8 紅外線指令（IR Command）

IR Command				
IR_TX=data	IR_TX(Rl:Rk:Rj:Ri)	IR_TX(Xj:Xi)	IR_RX_ON	IR_RX_OFF
[Rl,Rk,Rj,Ri] = IR_RX	[Xj,Xi] = IR_RX	IR_RX = data?Path	IR_RX != data?Path	
IR_Carrier_On	IR_Carrier_Off	-	-	

6.5.1.9 串列資料接收指令（Serial Control Command）

Serial Control Command			
SC_RX_ON	SC_RX_OFF	[Rl,Rk,Rj,Ri] = SC_RX	[Xj,Xi] = SC_RX
SC_RX = data?Path		SC_RX != data?Path	

6.5.1.10 脈衝調變 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	WaitPN	StopPWM	PausePWM
ResumePWM	-	-	-	-

6.5.1.11 時間延遲指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.1.12 動作指令（Action Command）

Action Command				
PlayA	PlayAS	WaitAN(Ch)	PauseA(Ch)	ResumeA(Ch)
StopA(Ch)	-	-	-	-

6.5.1.13 一般指令（MISC Command）

MISC Command				
Input State	Output State	Action Mark State	Wave Mark State	Key_CLR
Key_ON	Key_OFF	Stop	Pause(n)	Resume(n)
WDT_CLR	Repeat	Background	END	Debounce(Time)
ReadRollingCode	Ri=LVD	-	-	-

6.5.2 NY5 *Q-Code* 指令表

6.5.2.1 算數邏輯指令（Arithmetic Command）

Arithmetic Logic Command			
Var1 = Var2	Var1 = Var2 + Var3	Var1 = Var2 – Var3	Var1 = Var2 * Var3
Var1 = Var2 / Var3	Var1 = Var2 % Var3	Var++	Var--
Var1 = Var2 & Var3	Var1 = Var2 | Var3	Var1 = Var2 ^ Var3	Var1 = Var2 << Var3
Var1 = Var2 >> Var3	!Var	Var=RandomL	Var=RandomH
Var=Random	-	-	-

6.5.2.2 流程控制指令（Flow Control Command）

Flow Control Command				
Var = Var ? Path	Var != Var ? Path	Var >= Var ? Path	Var <= Var ? Path	Var > Var ? Path
Var < Var ? Path	Px = data?Path	Px != data?Path	Px.n = 0?Path	Px.n != 0?Path
Px.n = 1?Path	Px.n != 1?Path	Vol = n?Path	Vol !=n ?Path	-
MixCtrl = data?Path	MixCtrl != data ?Path	RandomL=data?Path	RandomH=data?Path	
RandomL!=data?Path	RandomH!=data?Path	Random=data?Path	Random!=data?Path	
Px[1 X 0 X]?Path	Voice?Path	PaueV?Path	Melody?Path	PauseM?Path
Delay? Path	Action?Path	PWMIO?Path	PausePWM?Path	
CheckSum?Path	!Px[1 X 0 X]?Path		!Voice?Path	!PaueV?Path
!Melody?Path	!PauseM?Path	!Delay? Path	!Action?Path	!PWMIO?Path
!PausePWM?Path		!CheckSum?Path	If-Else	
Switch(Ri) = [Path0, Path1, Path2,..Path15]		Switch(Px) = [Path0, Path1, Path2,..Path15]		
Switch(RandomL) = [Path0, Path1,..Path15]		Switch(RandomH) = [Path0, Path1,..Path15]		
Switch(Px[d x d x]) = [Path0, Path1, Path2….Path15]				
Switch(Xi)=[Path0, Path1, Path2,..Path255]				
Switch(Random)=[Path0, Path1,Path2,..Path255]				

Flow Control Command				
While	Do-While	For	-	-

6.5.2.3 I/O 指令（I/O Command）

4-bit I/O Command				
Ri = Px	Ri.n = Px.n	Px = data	Px = Ri	Px = Py
!Px	!Px.n	Px.n = 0	Px.n = 1	Px.n = Ri.n
Px.n = Py.n	Px = Py + Ri	Px = Py – Ri	Px = Py | Ri	Px = Py ^ Ri
Px = Py & Ri	Ri = Px + Rj	Ri = Px – Rj	Ri = Px | Rj	Ri = Px ^ Rj
Ri = Px & Rj	Px = Py + data	Px = Py – data	Px = Py | data	Px = Py ^ data
Px = Py & data	Ri = Px + data	Ri = Px – data	Ri = Px | data	Ri = Px ^ data
Ri = Px & data	Px = [1 x 0 FD An]	Px.n= 1KHz(time)	-	-
8-bit I/O Command				
XiL = Px	XiH = Px	XiL.n = Px.n	XiH.n = Px.n	Xi.n = Px.n
Xi = [Px, Py]	Px = XiL	Px = XiH	Px.n = XiL.n	Px.n = XiH.n
Px.n = Xi.n	[Px, Py] = Xi	-	-	-

6.5.2.4 路徑指令（Path Command）

Path Command				
ASM	BG	BreakFG	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro

6.5.2.5 語音指令（Voice Command）

Voice Command				
PlayV	PlayVS	Voicename	WaitVN(Ch)	PauseV(Ch)
ResumeV(Ch)	StopV(Ch)	FreqCH=nK	-	-

6.5.2.6 句子指令（Sentence Command）

Sentence Command				
PlayS	-	-	-	-

6.5.2.7 MIDI 指令（MIDI Command）

Melody Command				
PlayM	PlayMS	WaitMN	PauseM	ResumeM

Melody Command				
StopM	Tempo = n	Tempo++	Tempo--	Tempo(Rj:Ri)
Tempo(Xi)	ReadTempo	Mute_On(Ch)	Mute_Off(Ch)	OKON_On
OKON_Off	OKON_Play	-	-	-

6.5.2.8 音量指令（Volume Command）

Volume Command				
Vol_Max	Vol_Min	Vol = n	Vol = Ri	Vol++
Vol--	Ri = Vol	Px = Vol	Ri = MixCtrl	Px = MixCtrl
MixCtrl	-	-	-	-

6.5.2.9 查表指令（Table Command）

Table Command	
TableL(TableName, Rx/Xx, Ry/Xy, Ri)	TableM(TableName, Rx/Xx, Ry/Xy, Ri)
TableH(TableName, Rx/Xx, Ry/Xy, Ri)	Table(TableName, Rx/Xx, Ry/Xy, Rh, Rm, Rl)
TableL(TableName, X, Y, Ri)	TableM(TableName, X, Y, Ri)
TableH(TableName, X, Y, Ri)	Table(TableName, X, Y, Rh, Rm, Rl)
TableL(TableName, Rx/Xx, Ry/Xy, Xi)	TableH(TableName, Rx/Xx, Ry/Xy, Xi)
Table(TableName, Rx/Xx, Ry/Xy, Xh, Xl)	TableL(TableName, X, Y, Xi)
TableH(TableName, X, Y, Xi)	Table(TableName, X, Y, Xh, Xl)

6.5.2.10 紅外線指令（IR Command）

IR Command				
IR_TX=data	IR_TX(Rl:Rk:Rj:Ri)	IR_TX(Xj:Xi)	IR_RX_ON	IR_RX_OFF
[Rl,Rk,Rj,Ri] = IR_RX	[Xj,Xi] = IR_RX	IR_RX = data?Path	IR_RX != data?Path	
IR_Carrier_On	IR_Carrier_Off	-	-	

6.5.2.11 串列資料接收指令（Serial Control Command）

Serial Control Command			
SC_RX_ON	SC_RX_OFF	[Rl,Rk,Rj,Ri] = SC_RX	[Xj,Xi] = SC_RX
SC_RX = data?Path		SC_RX != data?Path	

6.5.2.12 脈衝調變 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	WaitPN	StopPWM	PausePWM
ResumePWM	-	-	-	-

6.5.2.13 中斷指令（Interrupt Command）

Interrupt Command				
INT_ON	INT_OFF	INT_RET	INT = n	-

6.5.2.14 時間延遲指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.2.15 動作指令（Action Command）

Action Command				
PlayA	PlayAS	WaitAN(Ch)	PauseA(Ch)	ResumeA(Ch)
StopA(Ch)	-	-	-	-

6.5.2.16 一般指令（MISC Command）

MISC Command				
Input State	Output State	Action Mark State	Wave Mark State	Melody Mark State
Note On State	Key_CLR	Key_ON	Key_OFF	Stop
Pause(n)	Resume(n)	ReadChannel	PauseDown	ResumeUp
Audio_ON	Audio_OFF	RampUp	RampDown	WDT_CLR
Repeat	Background	END	ReadRollingCode	Debounce(Time)

6.5.3 NY5+ Q-Code 指令表

6.5.3.1 算數邏輯指令（Arithmetic Command）

Arithmetic Logic Command			
Var1 = Var2	Var1 = Var2 + Var3	Var1 = Var2 – Var3	Var1 = Var2 * Var3
Var1 = Var2 / Var3	Var1 = Var2 % Var3	Var++	Var--
Var1 = Var2 & Var3	Var1 = Var2 | Var3	Var1 = Var2 ^ Var3	Var1 = Var2 << Var3
Var1 = Var2 >> Var3	!Var	Var=RandomL	Var=RandomH
Var=Random	-	-	-

6.5.3.2 流程控制指令（Flow Control Command）

Flow Control Command				
Var = Var ? Path	Var != Var ? Path	Var >= Var ? Path	Var <= Var ? Path	Var > Var ? Path
Var < Var ? Path	Px = data?Path	Px != data?Path	Px.n = 0?Path	Px.n != 0?Path
Px.n = 1?Path	Px.n != 1?Path	Vol = n?Path	Vol !=n ?Path	-
RandomL=data?Path	RandomH=data?Path	RandomL!=data?Path	RandomH!=data?Path	
Random=data?Path	Random!=data?Path	Px[1 X 0 X]?Path	ChUsed?Path	
Voice?Path	PaueV?Path	Melody?Path	PauseM?Path	Delay? Path
Action?Path	PauseA?Path	PWMIO?Path	PausePWM?Path	-
IoExp_Exists?Path		CheckSum?Path	!Px[1 X 0 X]?Path	!ChUsed?Path
!Voice?Path	!PaueV?Path	!Melody?Path	!PauseM?Path	!Delay? Path
!Action?Path	!PauseA?Path	!PWMIO?Path	!PausePWM?Path	
!IoExp_Exists?Path		!CheckSum?Path	If-Else	-
Switch(Ri) = [Path0, Path1, Path2,..Path15]		Switch(Px) = [Path0, Path1, Path2,..Path15]		
Switch(RandomL) = [Path0, Path1,..Path15]		Switch(RandomH) = [Path0, Path1,..Path15]		
Switch(Px[d x d x]) = [Path0, Path1, Path2….Path15]				
Switch(Xi)=[Path0, Path1, Path2,..Path255]				
Switch(Random)=[Path0, Path1,Path2,..Path255]				
While	Do-While	For	-	-

6.5.3.3 I/O 指令（I/O Command）

4-bit I/O Command				
Ri = Px	Ri = PxM	Ri.n = Px.n	PxM = data	PxM = Ri
PxM.n = 0	PxM.n = 1	Px = data	Px = Ri	Px = Py
!Px	!Px.n	Px.n = 0	Px.n = 1	Px.n = Ri.n

Px.n = Py.n	Px = Py + Ri	Px = Py – Ri	Px = Py | Ri	Px = Py ^ Ri
Px = Py & Ri	Ri = Px + Rj	Ri = Px – Rj	Ri = Px | Rj	Ri = Px ^ Rj
Ri = Px & Rj	Px = Py + data	Px = Py – data	Px = Py | data	Px = Py ^ data
Px = Py & data	Ri = Px + data	Ri = Px – data	Ri = Px | data	Ri = Px ^ data
Ri = Px & data	Px = [1 x 0 FD An]	Px.n= 1KHz(time)	-	-
8-bit I/O Command				
XiL = Px	XiH = Px	XiL.n = Px.n	XiH.n = Px.n	Xi.n = Px.n
Xi = [Px, Py]	Px = XiL	Px = XiH	Px.n = XiL.n	Px.n = XiH.n
Px.n = Xi.n	[Px, Py] = Xi	-	-	-

6.5.3.4 路徑指令（Path Command）

Path Command				
ASM	BG	Break	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro

6.5.3.5 語音指令（Voice Command）

Voice Command				
PlayV	PlayVS	WaitVN(Ch)	PauseV(Ch)	ResumeV(Ch)
StopV(Ch)	V_Chx_Freq=nK	V_Chx_Vol = n	V_Chx_Vol = Xi	Xi = V_Chx_Vol

6.5.3.6 句子指令（Sentence Command）

Sentence Command				
PlayS	-	-	-	-

6.5.3.7 計時器指令（Timer Command）

Timer Command				
TMR_ON	TMR_OFF	-	-	-

6.5.3.8 MIDI 指令（MIDI Command）

Melody Command				
PlayM	PlayMS	WaitMN	PauseM	ResumeM
StopM	Instrument(Ch, i)	M_Chx_Vol = n	M_Chx_Vol = Ri	Ri = M_Chx_Vol
Tempo + n	Tempo - n	Tempo++	Tempo--	TempoRst
ReadTempo(Ri:Rj)	ReadTempo(Xi)	Mute_On(Ch)	Mute_Off(Ch)	OKON_On

Melody Command				
OKON_Off	OKON_Play	OKON_SustainOn	OKON_SustainOff	OKON_SustainEnd
DynamicOn	DynamicOff	-	-	-

6.5.3.9 鍵盤指令（Keyboard Command）

Instrument Command				
InstNoteOn	InstNoteOff	DrumNoteOn	DrumNoteOff	MaxSingleNote

6.5.3.10 音量指令（Volume Command）

Volume Command				
Vol_Max	Vol_Min	Vol = n	Vol = Ri	Vol++
Vol--	Ri = Vol	Px = Vol	-	-

6.5.3.11 查表指令（Table Command）

Table Command	
TableL(TableName, Rx/Xx, Ry/Xy, Ri)	TableM(TableName, Rx/Xx, Ry/Xy, Ri)
TableH(TableName, Rx/Xx, Ry/Xy, Ri)	Table(TableName, Rx/Xx, Ry/Xy, Rh, Rm, Rl)
TableL(TableName, X, Y, Ri)	TableM(TableName, X, Y, Ri)
TableH(TableName, X, Y, Ri)	Table(TableName, X, Y, Rh, Rm, Rl)
TableL(TableName, Rx/Xx, Ry/Xy, Xi)	TableH(TableName, Rx/Xx, Ry/Xy, Xi)
Table(TableName, Rx/Xx, Ry/Xy, Xh, Xl)	TableL(TableName, X, Y, Xi)
TableH(TableName, X, Y, Xi)	Table(TableName, X, Y, Xh, Xl)

6.5.3.12 紅外線指令（IR Command）

IR Command				
IR_TX=data	IR_TX(Rl:Rk:Rj:Ri)	IR_TX(Xj:Xi)	IR_TX_WaitN	IR_RX_ON
IR_RX_OFF	[Rl,Rk,Rj,Ri] = IR_RX	[Xj,Xi] = IR_RX	IR_RX = data?Path	
IR_RX != data?Path		IR_Carrier_On	IR_Carrier_Off	-

6.5.3.13 串列資料接收指令（Serial Control Command）

Serial Control Command			
SC_RX_ON	SC_RX_OFF	[Rl,Rk,Rj,Ri] = SC_RX	[Xj,Xi] = SC_RX
SC_RX = data?Path		SC_RX != data?Path	-

6.5.3.14 I2C 指令（I2C Command）

I2C Command				
I2C_TX	I2C_RX	I2C_RX = data?Path		
I2C_RX != data?Path		I2C_Ack?Path	I2C_Start	I2C_Stop
I2C_MReadAck	I2C_MReadNAck	-	-	-

6.5.3.15 UART 指令（UART Command）

UART Command				
UART_TX	UART_TX_WaitN	UART_RX	UART_RX=data?Path	
UART_RX != data?Path		-	-	-

6.5.3.16 脈衝調變 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	PlayPWM	PlayPWMS	WaitPN
StopPWM	PausePWM	ResumePWM	-	-

6.5.3.17 中斷指令（Interrupt Command）

Interrupt Command				
INT_ON	INT_OFF	INT_RET	INT = n	-

6.5.3.18 時間延遲指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.3.19 動作指令（Action Command）

Action Command				
PlayA	PlayAS	WaitAN(Ch)	PauseA(Ch)	ResumeA(Ch)
StopA(Ch)	-	-	-	-

6.5.3.20 QFID 指令（QFID Command）

QFID Command				
QFID_TagId	QFID_TagInput	QFID_On	QFID_Off	QFID_SlowOn
QFID_SlowOff	-	-	-	-

6.5.3.21 RFC 指令（RFC Command）

RFC Command				
RFC_On	RFC_Off	RFC_Level(Ri)	-	-

6.5.3.22 I/O 擴展晶片指令（I/O Expander Command）

I/O Expander Command				
IoExp Input State	IoExp Output State	IoExp_Key_On	IoExp_Key_Off	IoExp_Key_Clr
IoExp_Sleep	IoExp_ErrId	IoExp_ReadInfo	IoExp_ReadSN	
IoExp_SetInputPullHigh		IoExp_SetInputPullLow		
IoExp_SetInputFloating		IoExp_SetOutputHigh		
IoExp_SetOutputLow		-		

6.5.3.23 一般指令（MISC Command）

MISC Command				
Input State	Output State	Action Mark State	Wave Mark State	Melody Mark State
Note On State	PWM-IO Mark State		QFID State	Key_CLR
Key_ON	Key_OFF	Stop	Pause(n)	Resume(n)
ReadChannel	PauseDown	ResumeUp	Audio_ON	Audio_OFF
AudioMode	RampUp	RampDown	WDT_CLR	Repeat
Background	Slow	SlowOff	END	Debounce(Time)
Direct_Debounce(Time)		Matrix_Debounce(Time)		ReadRollingCode
Ri = LVD	-	-	-	-

6.5.4 NY6 Q-Code 指令表

6.5.4.1 算數邏輯指令（Arithmetic Command）

Arithmetic Logic Command			
Var1 = Var2	Var1 = Var2 + Var3	Var1 = Var2 – Var3	Var1 = Var2 * Var3
Var1 = Var2 / Var3	Var1 = Var2 % Var3	Var++	Var--
Var1 = Var2 & Var3	Var1 = Var2 | Var3	Var1 = Var2 ^ Var3	Var1 = Var2 << Var3
Var1 = Var2 >> Var3	!Var	Var=RandomL	Var=RandomH
Var=Random	-	-	-

6.5.4.2 流程控制指令（Flow Control Command）

Flow Control Command				
Var = Var ? Path	Var != Var ? Path	Var >= Var ? Path	Var <= Var ? Path	Var > Var ? Path
Var < Var ? Path	Px = data?Path	Px != data?Path	Px.n = 0?Path	Px.n != 0?Path
Px.n = 1?Path	Px.n != 1?Path	Vol = n?Path	Vol !=n ?Path	-
RandomL=data?Path	RandomH=data?Path	RandomL!=data?Path	RandomH!=data?Path	
Random=data?Path	Random!=data?Path	Px[1 X 0 X]?Path	ChUsed?Path	
Voice?Path	PaueV?Path	Melody?Path	PauseM?Path	Delay? Path
Action?Path	PauseA?Path	PWMIO?Path	PausePWM?Path	
CheckSum?Path		!Px[1 X 0 X]?Path	!ChUsed?Path	!Voice?Path
!PaueV?Path	!Melody?Path	!PauseM?Path	!Delay? Path	!Action?Path
!PauseA?Path	!PWMIO?Path	!PausePWM?Path		!CheckSum?Path
If-Else	-	-	-	-
Switch(Ri) = [Path0, Path1, Path2,..Path15]		Switch(Px) = [Path0, Path1, Path2,..Path15]		
Switch(RandomL) = [Path0, Path1,..Path15]		Switch(RandomH) = [Path0, Path1,..Path15]		
Switch(Px[d x d x]) = [Path0, Path1, Path2….Path15]				
Switch(Xi)=[Path0, Path1, Path2,..Path255]				
Switch(Random)=[Path0, Path1,Path2,..Path255]				
While	Do-While	For	-	-

6.5.4.3 I/O 指令（I/O Command）

4-bit I/O Command				
Ri = Px	Ri = PxM	Ri.n = Px.n	PxM = data	PxM = Ri
PxM.n = 0	PxM.n = 1	Px = data	Px = Ri	Px = Py
!Px	!Px.n	Px.n = 0	Px.n = 1	Px.n = Ri.n

Px.n = Py.n	Px = Py + Ri	Px = Py – Ri	Px = Py | Ri	Px = Py ^ Ri
Px = Py & Ri	Ri = Px + Rj	Ri = Px – Rj	Ri = Px | Rj	Ri = Px ^ Rj
Ri = Px & Rj	Px = Py + data	Px = Py – data	Px = Py | data	Px = Py ^ data
Px = Py & data	Ri = Px + data	Ri = Px – data	Ri = Px | data	Ri = Px ^ data
Ri = Px & data	Px = [1 x 0 FD An]	Px.n= 1KHz(time)	-	-
8-bit I/O Command				
XiL = Px	XiH = Px	XiL.n = Px.n	XiH.n = Px.n	Xi.n = Px.n
Xi = [Px, Py]	Px = XiL	Px = XiH	Px.n = XiL.n	Px.n = XiH.n
Px.n = Xi.n	[Px, Py] = Xi	-	-	-

6.5.4.4 路徑指令（Path Command）

Path Command				
ASM	BG	BreakFG	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro

6.5.4.5 語音指令（Voice Command）

Voice Command				
PlayV	PlayVS	WaitVN(Ch)	PauseV(Ch)	ResumeV(Ch)
StopV(Ch)	V_Chx_Vol = n	V_Chx_Vol = Xi	Xi = V_Chx_Vol	-

6.5.4.6 句子指令（Sentence Command）

Sentence Command				
PlayS	WaitSN(n)	PauseS(n)	ResumeS(n)	StopS(n)

6.5.4.7 SPI Play 指令（SPI Play Command）

SPIPlay Command				
SPIPlay	SPIPlayS	SPIWaitN	SPIStop	SPIPause
SPIResume	SPIVol = n	SPIVol = Ri	Ri = SPIVol	-
SPIGetIndex(Result, SPIGroup)		-	-	-

6.5.4.8 SPI Flash 指令（SPI Flash Command）

SPI Flash Command				
SPI_WREN	SPI_WRDIS	SPI_RDID	SPI_CE	SPI_SE

SPI Flash Command				
SPI_BE	SPI_DP	SPI_RDP	SPI_WRSR	SPI_RDSR
SPI_WRD	SPI_RDD	SPI_GetAddr		

6.5.4.9 SPI 指令（SPI Command）

SPI Command				
SPI_CS_On	SPI_CS_Off	SPI_TX	SPI_RX	-

6.5.4.10 比較器指令（Comparator Command）

Comparator Command				
CMP_ON	CMP_ON(Source)	CMP_OFF	CMP_Read(Ri:Rj)	CMP_Read(Xi)
CMP_CNT_ON(Count)		CMP_CNT_OFF	-	-

6.5.4.11 計時器指令（Timer Command）

Counter Command				
TMR_ON(Time)	TMR_OFF	TMR_Read(Rj:Ri)	TMR_Read(Xi)	-

6.5.4.12 MIDI 指令（MIDI Command）

Melody Command				
PlayM	PlayMS	WaitMN	PauseM	ResumeM
StopM	Instrument(Ch, i)	M_Chx_Vol = n	M_Chx_Vol = Ri	Ri = M_Chx_Vol
Tempo + n	Tempo - n	Tempo++	Tempo--	TempoRst
ReadTempo(Ri:Rj)	ReadTempo(Xi)	Mute_On(Ch)	Mute_Off(Ch)	OKON_On
OKON_Off	OKON_Play	DynamicOn	DynamicOff	-

6.5.4.13 鍵盤指令（Keyboard Command）

Instrument Command				
InstNoteOn	InstNoteOff	DrumNoteOn	DrumNoteOff	MaxSingleNote

6.5.4.14 音量指令（Volume Command）

Volume Command				
Vol_Max	Vol_Min	Vol = n	Vol = Ri	Vol++
Vol--	Ri = Vol	Px = Vol	VolX1	VolX2

6.5.4.15 查表指令（Table Command）

Table Command	
TableL(TableName, Rx/Xx, Ry/Xy, Ri)	TableM(TableName, Rx/Xx, Ry/Xy, Ri)
TableH(TableName, Rx/Xx, Ry/Xy, Ri)	Table(TableName, Rx/Xx, Ry/Xy, Rh, Rm, Rl)
TableL(TableName, X, Y, Ri)	TableM(TableName, X, Y, Ri)
TableH(TableName, X, Y, Ri)	Table(TableName, X, Y, Rh, Rm, Rl)
TableL(TableName, Rx/Xx, Ry/Xy, Xi)	TableH(TableName, Rx/Xx, Ry/Xy, Xi)
Table(TableName, Rx/Xx, Ry/Xy, Xh, Xl)	TableL(TableName, X, Y, Xi)
TableH(TableName, X, Y, Xi)	Table(TableName, X, Y, Xh, Xl)

6.5.4.16 紅外線指令（IR Command）

IR Command				
IR_TX=data	IR_TX(Rl:Rk:Rj:Ri)	IR_TX(Xj:Xi)	IR_RX_ON	IR_RX_OFF
[Rl,Rk,Rj,Ri] = IR_RX	[Xj,Xi] = IR_RX	IR_RX = data?Path	IR_RX != data?Path	
IR_Carrier_On	IR_Carrier_Off	-	-	

6.5.4.17 串列資料接收指令（Serial Control Command）

Serial Control Command			
SC_RX_ON	SC_RX_OFF	[Rl,Rk,Rj,Ri] = SC_RX	[Xj,Xi] = SC_RX
SC_RX = data?Path		SC_RX != data?Path	-

6.5.4.18 脈衝調變 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	WaitPN	StopPWM	PausePWM
ResumePWM	-	-	-	-

6.5.4.19 時間延遲指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.4.20 動作指令（Action Command）

Action Command				
PlayA	PlayAS	WaitAN(Ch)	PauseA(Ch)	ResumeA(Ch)
StopA(Ch)	-	-	-	-

6.5.4.21 RFC 指令（RFC Command）

RFC Command				
RFC_On	RFC_Off	RFC_Level(Ri)	-	-

6.5.4.22 一般指令（MISC Command）

MISC Command				
Input State	Output State	Action Mark State	Melody Mark State	Note On State
Key_CLR	Key_ON	Key_OFF	Stop	Pause(n)
Resume(n)	ReadChannel	Audio_ON	Audio_OFF	AudioMode
NoiseFilter_ON	NoiseFilter_OFF	RampUp	RampDown	WDT_CLR
Repeat	Background	Slow	SlowOff	END
ReadRollingCode		ChMode(n)	Ri = LVD	Debounce(Time)
Direct_Debounce(Time)		Matrix_Debounce(Time)		-

6.5.5 *NY7 Q-Code* 指令表

6.5.5.1 算數邏輯指令（Arithmetic Command）

Arithmetic Logic Command			
Var1 = Var2	Var1 = Var2 + Var3	Var1 = Var2 – Var3	Var1 = Var2 * Var3
Var1 = Var2 / Var3	Var1 = Var2 % Var3	Var++	Var--
Var1 = Var2 & Var3	Var1 = Var2 | Var3	Var1 = Var2 ^ Var3	Var1 = Var2 << Var3
Var1 = Var2 >> Var3	!Var	Var=RandomL	Var=RandomH
Var=Random	-	-	-

6.5.5.2 流程控制指令（Flow Control Command）

Flow Control Command				
Var = Var ? Path	Var != Var ? Path	Var >= Var ? Path	Var <= Var ? Path	Var > Var ? Path
Var < Var ? Path	Px = data?Path	Px != data?Path	Px.n = 0?Path	Px.n != 0?Path
Px.n = 1?Path	Px.n != 1?Path	Vol = n?Path	Vol !=n ?Path	-
RandomL=data?Path	RandomH=data?Path	RandomL!=data?Path	RandomH!=data?Path	

Flow Control Command				
Random=data?Path	Random!=data?Path	Px[1 X 0 X]?Path	ChUsed?Path	
Voice?Path	PaueV?Path	Melody?Path	PauseM?Path	Delay? Path
Action?Path	PauseA?Path	PWMIO?Path	PausePWM?Path	CheckSum?Path
!Px[1 X 0 X]?Path		!ChUsed?Path	!Voice?Path	!PaueV?Path
!Melody?Path	!PauseM?Path	!Delay? Path	!Action?Path	!PauseA?Path
!PWMIO?Path	!PausePWM?Path		!CheckSum?Path	If-Else
Switch(Ri) = [Path0, Path1, Path2,..Path15]		Switch(Px) = [Path0, Path1, Path2,..Path15]		
Switch(RandomL) = [Path0, Path1,..Path15]		Switch(RandomH) = [Path0, Path1,..Path15]		
Switch(Px[d x d x]) = [Path0, Path1, Path2….Path15]				
Switch(Xi)=[Path0, Path1, Path2,..Path255]				
Switch(Random)=[Path0, Path1,Path2,..Path255]				
While	Do-While	For	-	-

6.5.5.3 I/O 指令（I/O Command）

4-bit I/O Command				
Ri = Px	Ri = PxM	Ri.n = Px.n	PxM = data	PxM = Ri
PxM.n = 0	PxM.n = 1	Px = data	Px = Ri	Px = Py
!Px	!Px.n	Px.n = 0	Px.n = 1	Px.n = Ri.n
Px.n = Py.n	Px = Py + Ri	Px = Py – Ri	Px = Py | Ri	Px = Py ^ Ri
Px = Py & Ri	Ri = Px + Rj	Ri = Px – Rj	Ri = Px | Rj	Ri = Px ^ Rj
Ri = Px & Rj	Px = Py + data	Px = Py – data	Px = Py | data	Px = Py ^ data
Px = Py & data	Ri = Px + data	Ri = Px – data	Ri = Px | data	Ri = Px ^ data
Ri = Px & data	Px = [1 x 0 FD An]	Px.n= 1KHz(time)	-	-
8-bit I/O Command				
XiL = Px	XiH = Px	XiL.n = Px.n	XiH.n = Px.n	Xi.n = Px.n
Xi = [Px, Py]	Px = XiL	Px = XiH	Px.n = XiL.n	Px.n = XiH.n
Px.n = Xi.n	[Px, Py] = Xi	-	-	-

6.5.5.4 路徑指令（Path Command）

Path Command				
ASM	BG	BreakFG	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro

6.5.5.5 語音指令（Voice Command）

Voice Command				
PlayV	PlayVS	WaitVN(Ch)	PauseV(Ch)	ResumeV(Ch)
StopV(Ch)	V_Chx_Vol = n	V_Chx_Vol = Xi	Xi = V_Chx_Vol	-

6.5.5.6 句子指令（Sentence Command）

Sentence Command				
PlayS	WaitSN(n)	PauseS(n)	ResumeS(n)	StopS(n)

6.5.5.7 SPI Play 指令（SPI Play Command）

SPIPlay Command				
SPIPlay	SPIPlayS	SPIWaitN	SPIStop	SPIPause
SPIResume	SPIVol = n	SPIVol = Ri	Ri = SPIVol	-
SPIGetIndex(Result, SPIGroup)		-		

6.5.5.8 SPI Flash 指令（SPI Flash Command）

SPI Flash Command				
SPI_WREN	SPI_WRDIS	SPI_RDID	SPI_CE	SPI_SE
SPI_BE	SPI_DP	SPI_RDP	SPI_WRSR	SPI_RDSR
SPI_WRD	SPI_RDD	SPI_GetAddr	-	-

6.5.5.9 SPI 指令（SPI Command）

SPI Command				
SPI_CS_On	SPI_CS_Off	SPI_TX	SPI_RX	-

6.5.5.10 MIDI 指令（MIDI Command）

Melody Command				
PlayM	PlayMS	WaitMN	PauseM	ResumeM
StopM	Instrument(Ch, i)	M_Chx_Vol = n	M_Chx_Vol = Ri	Ri = M_Chx_Vol
Tempo + n	Tempo - n	Tempo++	Tempo--	TempoRst
Tempo(Ri:Rj)	Tempo(Xi)	ReadTempo(Ri:Rj)	ReadTempo(Xi)	Mute_On(Ch)
Mute_Off(Ch)	OKON_On	OKON_Off	OKON_Play	DynamicOn
DynamicOff	StopMNote	-	-	-

6.5.5.11 鍵盤指令（Keyboard Command）

Instrument Command				
InstNoteOn	InstNoteOff	DrumNoteOn	DrumNoteOff	NoteVibrato
Gliss	MaxSingleNote	-	-	-

6.5.5.12 音量指令（Volume Command）

Volume Command				
Vol_Max	Vol_Min	Vol = n	Vol = Ri	Vol++
Vol--	Ri = Vol	Px = Vol	-	-

6.5.5.13 查表指令（Table Command）

Table Command	
TableL(TableName, Rx/Xx, Ry/Xy, Ri)	TableM(TableName, Rx/Xx, Ry/Xy, Ri)
TableH(TableName, Rx/Xx, Ry/Xy, Ri)	Table(TableName, Rx/Xx, Ry/Xy, Rh, Rm, Rl)
TableL(TableName, X, Y, Ri)	TableM(TableName, X, Y, Ri)
TableH(TableName, X, Y, Ri)	Table(TableName, X, Y, Rh, Rm, Rl)
TableL(TableName, Rx/Xx, Ry/Xy, Xi)	TableH(TableName, Rx/Xx, Ry/Xy, Xi)
Table(TableName, Rx/Xx, Ry/Xy, Xh, Xl)	TableL(TableName, X, Y, Xi)
TableH(TableName, X, Y, Xi)	Table(TableName, X, Y, Xh, Xl)

6.5.5.14 紅外線指令（IR Command）

IR Command				
IR_TX=data	IR_TX(Rl:Rk:Rj:Ri)	IR_TX(Xj:Xi)	IR_RX_ON	IR_RX_OFF
[Rl,Rk,Rj,Ri] = IR_RX	[Xj,Xi] = IR_RX	IR_RX = data?Path	IR_RX != data?Path	
IR_Carrier_On	IR_Carrier_Off	-	-	

6.5.5.15 串列資料接收指令（Serial Control Command）

Serial Control Command			
SC_RX_ON	SC_RX_OFF	[Rl,Rk,Rj,Ri] = SC_RX	[Xj,Xi] = SC_RX
SC_RX = data?Path		SC_RX != data?Path	-

6.5.5.16 脈衝調變 IO 指令（PWMIO Command）

PWMIO Command

PWMOut	PWMOutS	WaitPN	StopPWM	PausePWM
ResumePWM	-	-	-	-

6.5.5.17 時間延遲指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.5.18 動作指令（Action Command）

Action Command				
PlayA	PlayAS	WaitAN(Ch)	PauseA(Ch)	ResumeA(Ch)
StopA(Ch)	-	-	-	-

6.5.5.19 QFID 指令（QFID Command）

QFID Command				
QFID_TagId	QFID_TagInput	QFID_On	QFID_Off	QFID_SlowOn
QFID_SlowOff	-	-	-	-

6.5.5.20 RFC 指令（RFC Command）

RFC Command				
RFC_On	RFC_Off	RFC_Level(Ri)	-	-

6.5.5.21 一般指令（MISC Command）

MISC Command				
Input State	Output State	Action Mark State	Melody Mark State	Note On State
QFID State	Key_CLR	Key_ON	Key_OFF	Stop
Pause(n)	Resume(n)	ReadChannel	Audio_ON	Audio_OFF
AudioMode	NoiseFilter_ON	NoiseFilter_OFF	RampUp	RampDown
WDT_CLR	Repeat	Background	Slow	SlowOff
END	ChMode(n)	ReadRollingCode	Debounce(Time)	-
Direct_Debounce(Time)		Matrix_Debounce(Time)		-

6.5.6 NY9T *Q-Code* 指令表

6.5.6.1 算數邏輯指令（Arithmetic Command）

Arithmetic Logic Command			
Var1 = Var2	Var1 = Var2 + Var3	Var1 = Var2 – Var3	Var1 = Var2 * Var3
Var1 = Var2 / Var3	Var1 = Var2 % Var3	Var++	Var--
Var1 = Var2 & Var3	Var1 = Var2 | Var3	Var1 = Var2 ^ Var3	Var1 = Var2 << Var3
Var1 = Var2 >> Var3	!Var	Var=RandomL	Var=RandomH
Var=Random	-	-	-

6.5.6.2 流程控制指令（Flow Control Command）

Flow Control Command				
Var = Var ? Path	Var != Var ? Path	Var >= Var ? Path	Var <= Var ? Path	Var > Var ? Path
Var < Var ? Path	Px = data?Path	Px != data?Path	Px.n = 0?Path	Px.n != 0?Path
Px.n = 1?Path	Px.n != 1?Path	Vol = n?Path	Vol !=n ?Path	-
RandomL=data?Path	RandomH=data?Path	RandomL!=data?Path	RandomH!=data?Path	
Random=data?Path	Random!=data?Path	Px[1 X 0 X]?Path	Delay? Path	
PauseD?Path	Action?Path	PauseA?Path	PWMIO?Path	-
PausePWM?Path		HoldPWM?Path	Pause?Path	Calibrate?Path
CheckSum?Path		!Px[1 X 0 X]?Path		!Delay? Path
!PauseD?Path	!Action?Path	!PauseA?Path	!PWMIO?Path	-
!PausePWM?Path		!HoldPWM?Path	!Pause?Path	!Calibrate?Path
!CheckSum?Path		If-Else	-	-
Switch(Ri) = [Path0, Path1, Path2,..Path15]		Switch(Px) = [Path0, Path1, Path2,..Path15]		
Switch(RandomL) = [Path0, Path1,..Path15]		Switch(RandomH) = [Path0, Path1,..Path15]		
Switch(Px[d x d x]) = [Path0, Path1, Path2….Path15]				
Switch(Xi)=[Path0, Path1, Path2,..Path255]				
Switch(Random)=[Path0, Path1,Path2,..Path255]				
While	Do-While	For	-	-

6.5.6.3 I/O 指令（I/O Command）

4-bit I/O Command				
Ri = Px	Ri = PxM	Ri.n = Px.n	PxM = data	PxM = Ri
PxM.n = 0	PxM.n = 1	Px = data	Px = Ri	Px = Py
!Px	!Px.n	Px.n = 0	Px.n = 1	Px.n = Ri.n
Px.n = Py.n	Px = Py + Ri	Px = Py – Ri	Px = Py | Ri	Px = Py ^ Ri
Px = Py & Ri	Ri = Px + Rj	Ri = Px – Rj	Ri = Px | Rj	Ri = Px ^ Rj
Ri = Px & Rj	Px = Py + data	Px = Py – data	Px = Py | data	Px = Py ^ data
Px = Py & data	Ri = Px + data	Ri = Px – data	Ri = Px | data	Ri = Px ^ data
Ri = Px & data	Px = [1 x 0 FD An]	Px.n= 1KHz(time)	-	-
8-bit I/O Command				
XiL = Px	XiH = Px	XiL.n = Px.n	XiH.n = Px.n	Xi.n = Px.n
Xi = [Px, Py]	Px = XiL	Px = XiH	Px.n = XiL.n	Px.n = XiH.n
Px.n = Xi.n	[Px, Py] = Xi	-	-	-

6.5.6.4 路徑指令（Path Command）

Path Command				
ASM	BG	BreakFG	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro
1ms_RET	4ms_RET	-	-	-

6.5.6.5 句子指令（Sentence Command）

Sentence Command				
PlayS	-	-	-	-

6.5.6.6 觸摸鍵指令（TouchKey Command）

TouchKey Command			
TouchKey_ON	TouchKey_OFF	TouchKey_CLR	TouchKey_Scan_Slow
TouchKey_Scan_Normal	TouchKey_Sensitivity	Calibrate_ON	Calibrate_OFF
AutoJudge_Calibrate	Enforce_Calibrate_Normal	Enforce_Calibrate_Sleep	
Ri = TouchKey(Px)	-	-	

6.5.6.7 查表指令（Table Command）

Table Command	
TableL(TableName, Rx/Xx, Ry/Xy, Ri)	TableM(TableName, Rx/Xx, Ry/Xy, Ri)
TableH(TableName, Rx/Xx, Ry/Xy, Ri)	Table(TableName, Rx/Xx, Ry/Xy, Rh, Rm, Rl)
TableL(TableName, X, Y, Ri)	TableM(TableName, X, Y, Ri)
TableH(TableName, X, Y, Ri)	Table(TableName, X, Y, Rh, Rm, Rl)
TableL(TableName, Rx/Xx, Ry/Xy, Xi)	TableH(TableName, Rx/Xx, Ry/Xy, Xi)
Table(TableName, Rx/Xx, Ry/Xy, Xh, Xl)	TableL(TableName, X, Y, Xi)
TableH(TableName, X, Y, Xi)	Table(TableName, X, Y, Xh, Xl)

6.5.6.8 串列控制傳送指令（Serial Control TX Command）

Serial Control TX Command		
SC_TX(Mode, data)	SC_TX(Mode, Ri:Rj:Rl:Rk)	SC_TX(Mode, Xi:Xj)

6.5.6.9 脈衝調變 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	PlayPWM	PlayPWMS	WaitPN
StopPWM	PausePWM	ResumePWM	HoldPWM	PWMCtrl

6.5.6.10 時間延遲指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.6.11 動作指令（Action Command）

Action Command				
PlayA	PlayAS	WaitAN(Ch)	PauseA(Ch)	ResumeA(Ch)
HoldA(Ch)	StopA(Ch)	-	-	-

6.5.6.12 一般指令（MISC Command）

MISC Command				
Input State	Output State	Action Mark State	Key_CLR	Key_ON
Key_OFF	Stop	Pause(n)	Resume(n)	WDT_CLR
Repeat	Background	END	ReadRadj(Ri)	SW_Reset
Debounce(Time)	-	-	-	-

6.5.7 NX1 OTP Q-Code 指令表

6.5.7.1 算數邏輯指令（Arithmetic Command）

Arithmetic Logic Command			
Var1 = Var2	Var1 = Var2 + Var3	Var1 = Var2 – Var3	Var1 = Var2 * Var3
Var1 = Var2 / Var3	Var1 = Var2 % Var3	Var++	Var--
Var1 = Var2 & Var3	Var1 = Var2 | Var3	Var1 = Var2 ^ Var3	Var1 = Var2 << Var3
Var1 = Var2 >> Var3	!Var	Var=RandomL	Var=RandomH
Var=Random	-	-	-

6.5.7.2 流程控制指令（Flow Control Command）

Flow Control Command				
Var = Var ? Path	Var != Var ? Path	Var >= Var ? Path	Var <= Var ? Path	Var > Var ? Path
Var < Var ? Path	Px = data?Path	Px != data?Path	Px.n = 0?Path	Px.n != 0?Path
Px.n = 1?Path	Px.n != 1?Path	Vol = n?Path	Vol !=n ?Path	-
Random = data?Path		Random != data?Path		Px[1 X 0 X]?Path
ChUsed?Path	Voice?Path	PaueV?Path	Melody?Path	PauseM?Path
Delay? Path	Action?Path	PauseA?Path	SPIPlay?Path	SPIPause?Path
Record?Path	KRecord?Path	Recorded?Path	PlayK?Path	EraseR?Path
VR_VAD?Path	LEDStr?Path	LEDSync?Path	LEDText?Path	-
Animaltalks_Record?Path		Animaltalks_Play?Path		-
IoExp_Exists?Path		CheckSum?Path	SPI_CheckSum?Path	
!Px[1 X 0 X]?Path		!ChUsed?Path	!Voice?Path	!PaueV?Path
!Melody?Path	!PauseM?Path	!Delay? Path	!Action?Path	!PauseA?Path
!SPIPlay?Path	!SPIPause?Path	!Record?Path	!KRecord?Path	!Recorded?Path
!PlayK?Path	!EraseR?Path	!VR_VAD?Path	!LEDStr?Path	!LEDSync?Path
!LEDText?Path	!Animaltalks_Record?Path		!Animaltalks_Play?Path	
!IoExp_Exists?Path		!CheckSum?Path	!SPI_CheckSum?Path	
If-Else	-	-	-	-
Switch(Ri)=[Path0, Path1, Path2, ... Path15]		Switch(Px) = [Path0, Path1, Path2,..Path15]		
Switch(Px[d x d x]) = [Path0, Path1, Path2….Path15]				
Switch(Xi)=[Path0, Path1, Path2,..Path255]		Switch(Random)=[Path0, Path1,Path2,..Path255]		
While	Do-While	For	-	-

6.5.7.3 I/O 指令（I/O Command）

I/O Command				
Ri = Px	Ri = PxM	Ri.n = Px.n	PxM = data	PxM = Ri
PxM.n = 0	PxM.n = 1	Px = data	Px = Ri	Px = Py
!Px	!Px.n	Px.n = 0	Px.n = 1	Px.n = Ri.n
Px.n = Py.n	Px = Py + Ri	Px = Py – Ri	Px = Py | Ri	Px = Py ^ Ri
Px = Py & Ri	Ri = Px + Rj	Ri = Px – Rj	Ri = Px | Rj	Ri = Px ^ Rj
Ri = Px & Rj	Px = Py + data	Px = Py – data	Px = Py | data	Px = Py ^ data
Px = Py & data	Ri = Px + data	Ri = Px – data	Ri = Px | data	Ri = Px ^ data
Ri = Px & data	Px = [1 x 0]		-	-

6.5.7.4 路徑指令（Path Command）

Path Command				
C-Code	BG	StopFG	StopBG	StopBG1
StopBG2	StopBG3	StopBG4	StopBG5	Subroutine
Label(Pathname)	Macro	-	-	-

6.5.7.5 語音指令（Voice Command）

Voice Command				
PlayV	PlayVS	WaitVN(Ch)	PauseV(Ch)	ResumeV(Ch)
StopV(Ch)	SBC_Loop_On	SBC_Loop_Off	ADPCM_Loop_On	ADPCM_Loop_Off
ADPCM_UpSampling		ADM_Loop_On	ADM_Loop_Off	ADM_UpSampling
PCM_Loop_On	PCM_Loop_Off	ReadFileCountV	-	-

6.5.7.6 錄音指令（Record Command）

Record Command				
Record	RecordS	WaitRN	StopR	EraseR
EraseRS	WaitEN	-	-	-

6.5.7.7 句子指令（Sentence Command）

Sentence Command				
PlayS	WaitSN	PauseS(n)	ResumeS(n)	StopS(n)

6.5.7.8 SPI Play 指令（SPI Play Command）

SPIPlay Command				
SPIPlay	SPIPlayS	SPIWaitN	SPIStop	SPIPause
SPIResume	SPIGetIndex(Result, SPIGroup)		-	-

6.5.7.9 SPI Flash 指令（SPI Flash Command）

SPI Flash Command				
SPI_WREN	SPI_WRDIS	SPI_RDID	SPI_CE	SPI_SE
SPI_BE	SPI_DP	SPI_RDP	SPI_WRSR	SPI_RDSR
SPI_WRD	SPI_RDD	SPI_GetAddr		

6.5.7.10 SPI 指令（SPI Command）

SPI Command				
SPI_CS_On	SPI_CS_Off	SPI_TX	SPI_RX	SPI_CLKDIV
SPI_CPOL	SPI_CPHA	-	-	-

6.5.7.11 Storage 指令（Storage Command）

Storage Command				
Storage_Save	-	-	-	-

6.5.7.12 計時器指令（Timer Command）

Timer Command				
TMR_On	TMR_OFF	-	-	-

6.5.7.13 MIDI 指令（MIDI Command）

Melody Command				
PlayM	PlayMS	WaitMN	PauseM	ResumeM
StopM	Instrument	M_Chx_Vol = n	M_Chx_Vol = Ri	Ri = M_Chx_Vol
Tempo + n	Tempo - n	Tempo++	Tempo--	TempoRst
ReadTempo(Ri)	Mute_On(Ch)	Mute_Off(Ch)	OKON_On	OKON_Off
OKON_Play	MIDI_Pitch	Mask_On	Mask_Off	ReadFileCountM
MIDI_Loop_On	MIDI_Loop_Off	-	-	-

6.5.7.14 鍵盤指令（Keyboard Command）

Instrument Command				
InstNoteOn	InstNoteOff	InstNoteAllOff	DrumNoteOn	DrumNoteOff
NoteVibrato	Gliss	LongInst_HoldTime		-
ShortInst_HoldTime		KRecord	KRecordS	WaitKRN
StopKR	PlayK	PlayKS	WaitKN	StopK

6.5.7.15 音量指令（Volume Command）

Volume Command				
Vol_Max	Vol_Min	Vol = n	Vol = Ri	Vol++
Vol--	Ri = Vol	Px = Vol	Vol += n	Vol -= n
CHx_Vol = n	PP_Gain = n	PP_Gain = Ri	PP_Gain++	PP_Gain--
Ri = PP_Gain	PGA_Gain = n	PGA_Gain = Ri	Ri = PGA_Gain	MixCtrl

6.5.7.16 觸摸鍵指令（TouchKey Command）

TouchKey Command			
TouchKey_Sensitivity	Enforce_Calibrate_Normal		Touchkey_Count
Touchkey_BGCount	-	-	-

6.5.7.17 查表指令（Table Command）

Table Command	
Table(TableName, VarX, VarY, VarR)	-

6.5.7.18 紅外線指令（IR Command）

IR Command				
IR_TX=data	IR_TX(Ri)	IR_TX_WaitN	IR_RX_ON	IR_RX_OFF
Ri = IR_RX	IR_RX = data?Path	IR_RX != data?Path	IR_TX_Busy?Path	
IR_Carrier_On		IR_Carrier_Off	-	

6.5.7.19 I2C 指令（I2C Command）

I2C Command				
I2C_TX	I2C_RX	I2C_RX = data?Path		
I2C_RX != data?Path		I2C_Ack?Path	I2C_Start	I2C_Stop
I2C_MReadAck	I2C_MReadNAck	I2C_Reset	-	-

6.5.7.20 UART 指令（UART Command）

UART Command			
UART_TX	UART_RX	UART_RX=data?Path	-
UART_RX != data?Path		-	-	-

6.5.7.21 中斷指令（Interrupt Command）

Interrupt Command				
INT_ON	INT_OFF	-	-	--

6.5.7.22 時間延遲指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.7.23 動作指令（Action Command）

Action Command				
PlayA	PlayAS	WaitAN(Ch)	PauseA(Ch)	ResumeA(Ch)
StopA(Ch)	-	-	-	-

6.5.7.24 LED Strip 指令（LED Strip Command）

LED Strip Command				
LEDStr_Play	LEDStr_PlayS	LEDStr_Stop	LEDStr_Clear	LEDStr_Brightness
LEDSync_Play	LEDSync_PlayS	LEDSync_Stop	LEDSync_Clear	LEDSync_Brightness
LEDText_Play	LEDText_PlayS	LESText_Stop	LEDText_Clear	LEDText_Brightness

6.5.7.25 QFID 指令（QFID Command）

QFID Command				
QFID_GroupID	QFID_TagId	QFID_TagInput	QFID_On	QFID_Off

6.5.7.26 WaveID 指令（WaveID Command）

WaveID Command				
WaveID_TX	WaveID_RX_ON	WaveID_RX_OFF	WaveID_RX	-

6.5.7.27 聽聲辨位指令（Sound Localization Command）

Sound Localization Command

SL_On	SL_Off	SL_RX	-	-

6.5.7.28 聲音偵測指令（Sound Detect Command）

Sound Detect Command				
SoundDetect_On	SoundDetect_Off	-	-	-

6.5.7.29 音高偵測指令（Pitch Detect Command）

Pitch Detect Command				
PitchDetect_On	PitchDetect_Off	ReadPitch	-	-

6.5.7.30 RFC 指令（RFC Command）

RFC Command				
RFC_On	RFC_Off	RFC_Level(Ri)	-	-

6.5.7.31 ADC 輸入指令（ADC Input Command）

ADC Input Command				
ADC_Input	-	-	-	-

6.5.7.32 語音辨識指令（VR Command）

VR Command				
VR State	VR_ON	VR_OFF	VR_VAD=n	VR_VAD_On
VR_VAD_Off	VRGC_Timeout_CLR		Ri = VR_HitScore	Ri = VR_HitID
VR_Loading	VT_Training	VT_Delete	VT_DeleteAll	VT_TrainingNum

6.5.7.33 變音效果指令（Sound Effect Command）

Sound Effect Command				
PitchChange	PitchChange_Off	SpeedChange	SpeedChange_Off	Robot1
Robot1_Off	Robot2	Robot2_Off	Robot3	Robot3_Off
Robot4	Robot4_Off	Echo	Echo_Off	Reverb
Reverb_Off	Darth	Darth_Off	AnimalRoar	AnimalRoar_Off

6.5.7.34 即時播放指令（Real Time Play Command）

Real Time Play Command

Real Time Play Command				
RT_Play	RT_Play_Off	RT_PitchChange	RT_PitchChange_Off	
RT_Robot1	RT_Robot1_Off	RT_Robot2	RT_Robot2_Off	RT_Robot3
RT_Robot3_Off	RT_Robot4	RT_Robot4_Off	RT_Echo	RT_Echo_Off
RT_Reverb	RT_Reverb_Off	RT_Ghost	RT_Ghost_Off	RT_Darth
RT_Darth_Off	RT_Chorus	RT_Chorus_Off	**RT_Vol = n**	**RT_Vol = Var**
Var = RT_Vol	-	-	-	-

6.5.7.35 動物變音指令（Animaltalks Command）

Animaltalks Command				
Animaltalks_On	Animaltalks_Off	Animaltalks_Record	Animaltalks_RecordS	
Animaltalks_StopR	Animaltalks_Play	Animaltalks_PlayS	Animaltalks_Stop	
Animaltalks_SetVoice		Animaltalks_LongSound_On		-
Animaltalks_LongSound_Off		-	-	-

6.5.7.36 動物唱歌指令（Animalsings Command）

Animalsings Command			
Animalsings_On	Animalsings_Off	Animalsings_Record	Animalsings_RecordS
Animalsings_StopR	Animalsings_Play	Animalsings_PlayS	Animalsings_Stop
Animalsings_SetVoice		Animalsings_LongSound_On	
Animalsings_LongSound_Off		Animalsings_NC_On	Animalsings_NC_Off
Animalsings_NC_Auto			

6.5.7.37 I/O 擴展晶片指令（I/O Expander Command）

I/O Expander Command				
IoExp Input State	IoExp Output State	IoExp_Key_Clr	IoExp_Key_On	
IoExp_Key_Off	IoExp_Sleep	IoExp_ErrId	IoExp_ReadInfo	IoExp_ReadSN
IoExp_SetInputPullHigh		IoExp_SetInputPullLow		
IoExp_SetInputFloating		IoExp_SetOutputHigh		
IoExp_SetOutputLow		-		

6.5.7.38 一般指令（MISC Command）

MISC Command				
Input State	Output State	Action Mark State	Wave Mark State	Melody Mark State
Note On State	QFID State	Key_CLR	Key_ON	Key_OFF
Stop	Pause(n)	Resume(n)	Audio_Loop_On	Audio_Loop_Off
NoiseFilter_ON		NoiseFilter_OFF	EQ_Filter	EQ_Filter_Off
AGC_On	AGC_Off	AutoSleep_On	AutoSleep_Off	Sleep
WDT_CLR	Repeat	Background	END	-
ReadRollingCode		Ri = LVD	Enforce_Wakeup	GetMicVol
Millis	Srand	-	-	-

6.5.7.39 偵錯指令（Debug Command）

Debug Command				
Printf	-	-	-	-

6.5.8 NX1 EF Q-Code 指令表

6.5.8.1 算數邏輯指令（Arithmetic Command）

Arithmetic Logic Command			
Var1 = Var2	Var1 = Var2 + Var3	Var1 = Var2 – Var3	Var1 = Var2 * Var3
Var1 = Var2 / Var3	Var1 = Var2 % Var3	Var++	Var--
Var1 = Var2 & Var3	Var1 = Var2 | Var3	Var1 = Var2 ^ Var3	Var1 = Var2 << Var3
Var1 = Var2 >> Var3	!Var	Var=RandomL	Var=RandomH
Var=Random	-	-	-

6.5.8.2 流程控制指令（Flow Control Command）

Flow Control Command				
Var = Var ? Path	Var != Var ? Path	Var >= Var ? Path	Var <= Var ? Path	Var > Var ? Path
Var < Var ? Path	Px = data?Path	Px != data?Path	Px.n = 0?Path	Px.n != 0?Path
Px.n = 1?Path	Px.n != 1?Path	Vol = n?Path	Vol !=n ?Path	-
Random=data?Path	Random!=data?Path	Px[1 X 0 X]?Path	ChUsed?Path	
Voice?Path	PaueV?Path	Melody?Path	PauseM?Path	Delay? Path
Action?Path	PauseA?Path	SPIPlay?Path	SPIPause?Path	Record?Path
KRecord?Path	Recorded?Path	PlayK?Path	EraseR?Path	VR_VAD?Path

Flow Control Command				
LEDStr?Path	LEDSync?Path	LEDText?Path	Animaltalks_Record?Path	
Animaltalks_Play?Path		IoExp_Exists?Path		CheckSum?Path
SPI_Checksum?Path		!Px[1 X 0 X]?Path		!ChUsed?Path
!Voice?Path	!PaueV?Path	!Melody?Path	!PauseM?Path	!Delay? Path
!Action?Path	!PauseA?Path	!SPIPlay?Path	!SPIPause?Path	!Record?Path
!KRecord?Path	!Recorded?Path	!PlayK?Path	!EraseR?Path	!VR_VAD?Path
!LEDStr?Path	!LEDSync?Path	!LEDText?Path	!Animaltalks_Record?Path	
!Animaltalks_Play?Path		!IoExp_Exists?Path		!CheckSum?Path
!SPI_Checksum?Path		If-Else	-	-
Switch(Ri) = [Path0, Path1, Path2,..Path15]		Switch(Px) = [Path0, Path1, Path2,..Path15]		
Switch(Px[d x d x]) = [Path0, Path1, Path2….Path15]				
Switch(Xi)=[Path0, Path1, Path2,..Path255]		Switch(Random)=[Path0, Path1,Path2,..Path255]		
While	Do-While	For	-	-

6.5.8.3 I/O 指令（I/O Command）

I/O Command				
Ri = Px	Ri = PxM	Ri.n = Px.n	PxM = data	PxM = Ri
PxM.n = 0	PxM.n = 1	Px = data	Px = Ri	Px = Py
!Px	!Px.n	Px.n = 0	Px.n = 1	Px.n = Ri.n
Px.n = Py.n	Px = Py + Ri	Px = Py – Ri	Px = Py | Ri	Px = Py ^ Ri
Px = Py & Ri	Ri = Px + Rj	Ri = Px – Rj	Ri = Px | Rj	Ri = Px ^ Rj
Ri = Px & Rj	Px = Py + data	Px = Py – data	Px = Py | data	Px = Py ^ data
Px = Py & data	Ri = Px + data	Ri = Px – data	Ri = Px | data	Ri = Px ^ data
Ri = Px & data	Px = [1 x 0]		-	-

6.5.8.4 路徑指令（Path Command）

Path Command				
C-Code	BG	StopFG	StopBG	StopBG1
StopBG2	StopBG3	StopBG4	StopBG5	Subroutine
Label(Pathname)	Macro	-	-	-

6.5.8.5 語音指令（Voice Command）

Voice Command				
PlayV	PlayVS	WaitVN(Ch)	PauseV(Ch)	ResumeV(Ch)
StopV(Ch)	SBC_Loop_On	SBC_Loop_Off	ADPCM_Loop_On	ADPCM_Loop_Off
ADPCM_UpSampling		ADM_Loop_On	ADM_Loop_Off	ADM_UpSampling
PCM_Loop_On	PCM_Loop_Off	ReadFileCountV	-	-

6.5.8.6 錄音指令（Record Command）

Record Command				
Record	RecordS	WaitRN	StopR	EraseR
EraseRS	WaitEN	-	-	-

6.5.8.7 句子指令（Sentence Command）

Sentence Command				
PlayS	WaitSN	PauseS(n)	ResumeS(n)	StopS(n)

6.5.8.8 SPI Play 指令（SPI Play Command）

SPIPlay Command				
SPIPlay	SPIPlayS	SPIWaitN	SPIStop	SPIPause
SPIResume	SPIGetIndex(Result, SPIGroup)		-	-

6.5.8.9 SPI Flash 指令（SPI Flash Command）

SPI Flash Command				
SPI_WREN	SPI_WRDIS	SPI_RDID	SPI_CE	SPI_SE
SPI_BE	SPI_DP	SPI_RDP	SPI_WRSR	SPI_RDSR
SPI_WRD	SPI_RDD	SPI_GetAddr		

6.5.8.10 SPI 指令（SPI Command）

SPI Command				
SPI_CS_On	SPI_CS_Off	SPI_TX	SPI_RX	SPI_CLKDIV-
SPI_CPOL	SPI_CPHA	-	-	-

6.5.8.11 Embedded Flash 指令（Embedded Flash Command）

Embedded Flash Command				
EF_SE	EF_WRD	EF_RDD	EF_GetAddr	-

6.5.8.12 Storage 指令（Storage Command）

Storage Command				
Storage_Save	-	-	-	-

6.5.8.13 計時器指令（Timer Command）

Timer Command				
TMR_On	TMR_OFF	-	-	-

6.5.8.14 MIDI 指令（MIDI Command）

Melody Command				
PlayM	PlayMS	WaitMN	PauseM	ResumeM
StopM	Instrument	M_Chx_Vol = n	M_Chx_Vol = Ri	Ri = M_Chx_Vol
Tempo + n	Tempo - n	Tempo++	Tempo--	TempoRst
ReadTempo(Ri)	Mute_On(Ch)	Mute_Off(Ch)	OKON_On	OKON_Off
OKON_Play	MIDI_Pitch	Mask_On	Mask_Off	ReadFileCountM
MIDI_Loop_On	MIDI_Loop_Off	-	-	-

6.5.8.15 鍵盤指令（Keyboard Command）

Instrument Command				
InstNoteOn	InstNoteOff	InstNoteAllOff	DrumNoteOn	DrumNoteOff
NoteVibrato	Gliss	LongInst_HoldTime		-
ShortInst_HoldTime		KRecord	KRecordS	WaitKRN
StopKR	PlayK	PlayKS	WaitKN	StopK

6.5.8.16 音量指令（Volume Command）

Volume Command				
Vol_Max	Vol_Min	Vol = n	Vol = Ri	Vol++
Vol--	Ri = Vol	Px = Vol	Vol += n	Vol -= n
CHx_Vol = n	PP_Gain = n	PP_Gain = Ri	PP_Gain++	PP_Gain--
Ri = PP_Gain	PGA_Gain = n	PGA_Gain = Ri	Ri = PGA_Gain	MixCtrl

6.5.8.17 觸摸鍵指令（TouchKey Command）

TouchKey Command			
TouchKey_Sensitivity	Enforce_Calibrate_Normal		Touchkey_Count
Touchkey_BGCount	-	-	-

6.5.8.18 查表指令（Table Command）

Table Command	
Table(TableName, VarX, VarY, VarR)	-

6.5.8.19 紅外線指令（IR Command）

IR Command				
IR_TX=data	IR_TX(Ri)	IR_TX_WaitN	IR_RX_ON	IR_RX_OFF
Ri = IR_RX	IR_RX = data?Path	IR_RX != data?Path	IR_TX_Busy?Path	
IR_Carrier_On	IR_Carrier_Off	-	-	

6.5.8.20 I2C 指令（I2C Command）

I2C Command				
I2C_TX	I2C_RX	I2C_RX = data?Path		
I2C_RX != data?Path		I2C_Ack?Path	I2C_Start	I2C_Stop
I2C_MReadAck	I2C_MReadNAck	I2C_Reset	-	-

6.5.8.21 UART 指令（UART Command）

UART Command			
UART_TX	UART_TX_WaitN	UART_RX	UART_RX=data?Path
UART_RX != data?Path		UART_TX_Busy?Path	-

6.5.8.22 中斷指令（Interrupt Command）

Interrupt Command				
INT_ON	INT_OFF	-	-	--

6.5.8.23 時間延遲指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.8.24 動作指令（Action Command）

Action Command				
PlayA	PlayAS	WaitAN(Ch)	PauseA(Ch)	ResumeA(Ch)
StopA(Ch)	-	-	-	-

6.5.8.25 LED Strip 指令（LED Strip Command）

LED Strip Command				
LEDStr_Play	LEDStr_PlayS	LEDStr_Stop	LEDStr_Clear	LEDStr_Brightness
LEDSync_Play	LEDSync_PlayS	LEDSync_Stop	LEDSync_Clear	LEDSync_Brightness
LEDText_Play	LEDText_PlayS	LESText_Stop	LEDText_Clear	LEDText_Brightness

6.5.8.26 QFID 指令（QFID Command）

QFID Command				
QFID_GroupID	QFID_TagId	QFID_TagInput	QFID_On	QFID_Off

6.5.8.27 WaveID 指令（WaveID Command）

WaveID Command				
WaveID_TX	WaveID_RX_ON	WaveID_RX_OFF	WaveID_RX	-

6.5.8.28 聽聲辨位指令（Sound Localization Command）

Sound Localization Command				
SL_On	SL_Off	SL_RX	-	-

6.5.8.29 聲音偵測指令（Sound Detect Command）

Sound Detect Command				
SoundDetect_On	SoundDetect_Off	-	-	-

6.5.8.30 音高偵測指令（Pitch Detect Command）

Pitch Detect Command				
PitchDetect_On	PitchDetect_Off	ReadPitch	-	-

6.5.8.31 RFC 指令（RFC Command）

RFC Command

RFC_On	RFC_Off	RFC_Level(Ri)	-	-

6.5.8.32 ADC 輸入指令（ADC Input Command）

ADC Input Command				
ADC_Input	-	-	-	-

6.5.8.33 語音辨識指令（VR Command）

VR Command				
VR State	VR_ON	VR_OFF	VR_VAD=n	VR_VAD_On
VR_VAD_Off	VRGC_Timeout_CLR		Ri = VR_HitScore	Ri = VR_HitID
VR_Loading	VT_Training	VT_Delete	VT_DeleteAll	VT_TrainingNum

6.5.8.34 變音效果指令（Sound Effect Command）

Sound Effect Command				
PitchChange	PitchChange_Off	SpeedChange	SpeedChange_Off	Robot1
Robot1_Off	Robot2	Robot2_Off	Robot3	Robot3_Off
Robot4	Robot4_Off	Echo	Echo_Off	Reverb
Reverb_Off	Darth	Darth_Off	AnimalRoar	AnimalRoar_Off

6.5.8.35 即時播放指令（Real Time Play Command）

Real Time Play Command				
RT_Play	RT_Play_Off	RT_PitchChange	RT_PitchChange_Off	
RT_Robot1	RT_Robot1_Off	RT_Robot2	RT_Robot2_Off	RT_Robot3
RT_Robot3_Off	RT_Robot4	RT_Robot4_Off	RT_Echo	RT_Echo_Off
RT_Reverb	RT_Reverb_Off	RT_Ghost	RT_Ghost_Off	RT_Darth
RT_Darth_Off	RT_Chorus	RT_Chorus_Off	RT_Vol = n	RT_Vol = Var
Var = RT_Vol	-	-	-	-

6.5.8.36 動物變音指令（Animaltalks Command）

Animaltalks Command			
Animaltalks_On	Animaltalks_Off	Animaltalks_Record	Animaltalks_RecordS
Animaltalks_StopR	Animaltalks_Play	Animaltalks_PlayS	Animaltalks_Stop
Animaltalks_SetVoice		Animaltalks_LongSound_On	-
Animaltalks_LongSound_Off		-	-

6.5.8.37 动物唱歌指令（Animalsings Command）

Animalsings Command			
Animalsings_On	Animalsings_Off	Animalsings_Record	Animalsings_RecordS
Animalsings_StopR	Animalsings_Play	Animalsings_PlayS	Animalsings_Stop
Animalsings_SetVoice		Animalsings_LongSound_On	
Animalsings_LongSound_Off		Animalsings_NC_On	Animalsings_NC_Off
Animalsings_NC_Auto			

6.5.8.38 I/O 擴展晶片指令（I/O Expander Command）

I/O Expander Command				
IoExp Input State	IoExp Output State	IoExp_Key_On	IoExp_Key_Off	IoExp_Key_Clr
IoExp_Sleep	IoExp_ErrId	IoExp_ReadInfo	IoExp_ReadSN	
IoExp_SetInputPullHigh		IoExp_SetInputPullLow		
IoExp_SetInputFloating		IoExp_SetOutputHigh		
IoExp_SetOutputLow		-		

6.5.8.39 一般指令（MISC Command）

MISC Command				
Input State	Output State	Action Mark State		-
Wave Mark State	Melody Mark State		Note On State	QFID State
Key_CLR	Key_ON	Key_OFF	Stop	Pause(n)
Resume(n)	Audio_Loop_On		Audio_Loop_Off	NoiseFilter_ON
NoiseFilter_OFF	EQ_Filter	EQ_Filter_Off	AGC_On	AGC_Off
AutoSleep_On	AutoSleep_Off	Sleep	WDT_CLR	Repeat
Background	END	ReadRollingCode		Ri = LVD
Enforce_Wakeup	GetMicVol	Millis	Srand	-

6.5.8.40 偵錯指令（Debug Command）

Debug Command				
Printf	-	-	-	-

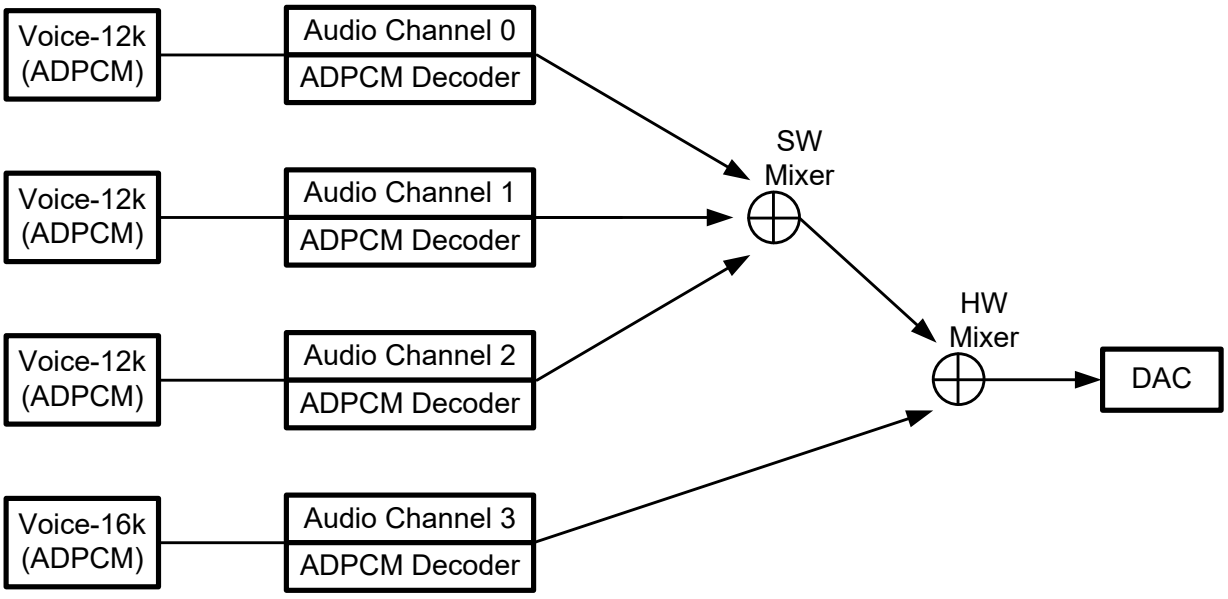
6.6 NX1 聲音輸出通道

NX1(底層為 C_MODULE)支援八個語音同時透過軟體通道輸出，軟體通道分別為：Audio Channel 0 ~ Audio Channel 7，底層程式會根據音樂的取樣頻率，自動對應到硬體通道的 HW_CH0 與 HW_CH1。同時只能播放兩種不同的取樣頻率，若同時播放三種不同的取樣頻率，則會造成播音異常。若有使用語音錄音或是音效(SpeedChange / PitchChange / RobotX / Echo / Reverb / Darth)，軟體通道同時可以支援的通道數會降為兩通道。

語音格式上，最多支援兩組 SBC-1、兩組 SBC-2、八組 ADPCM、八組 ADM、三組 PCM、一組 CELP 和一組 MIDI 解碼演算法，使用上須注意解碼演算法的組數，如果同一時間需要播放的語音超過支援的組數，語音會無法成功播放。多通道的應用需要設定演算法的使用狀況，設定方法請參考 RAM Usage。

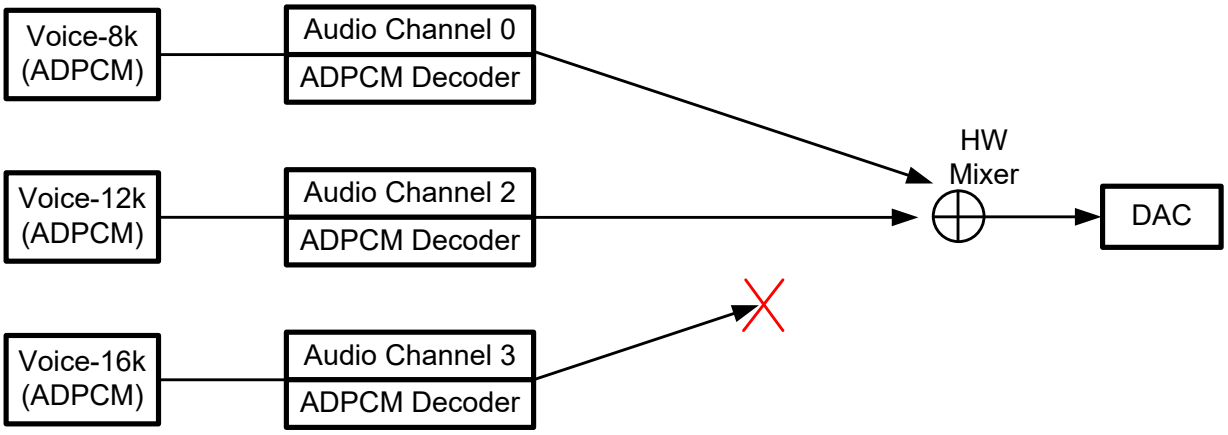
例. 四通道 ADPCM 同時播放，總共兩種 sample rate

TR1: PlayVS(Ch0, $v0), PlayVS(Ch1, $v1) , PlayVS(Ch2, $v2) , PlayV(Ch3, $v3)



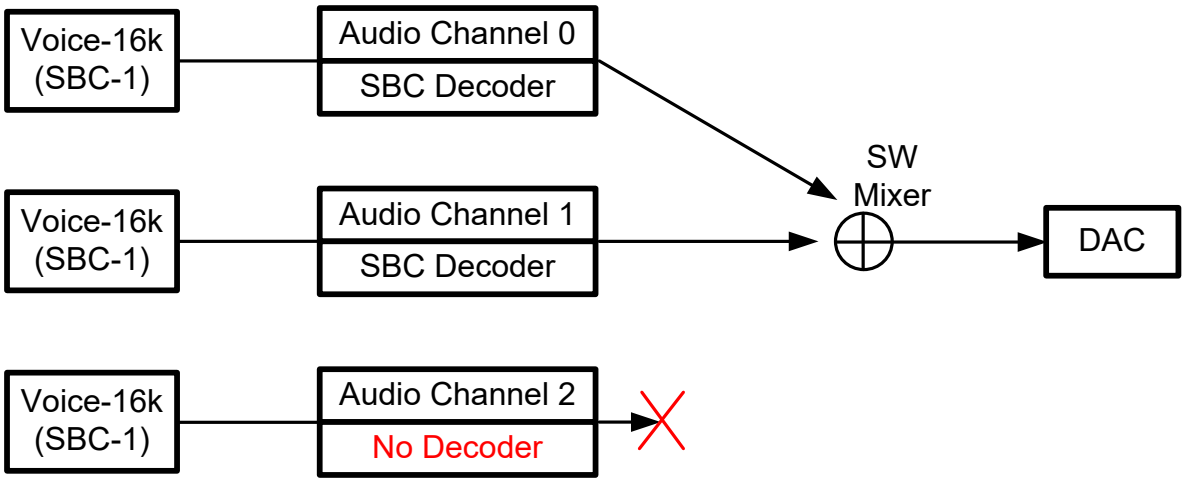
例. 三通道 ADPCM 總共三種 sample rate，無法同時播放

TR1: PlayVS(Ch0, $v0), PlayVS(Ch1, $v1) , PlayV(Ch2, $v2)



例. 三通道 SBC 無法同時播放，

TR1: PlayVS(Ch0, $v0), PlayVS(Ch1, $v1) , PlayV(Ch2, $v2)



6.7 RAM 資源使用說明

在 *Q-Code* 中，使用到特定的功能或是指令時，會佔用到 RAM。根據使用不同的功能或是指令，佔用的 RAM 數量皆不同。系統佔用的規則為從編號後面的 RAM 開始往前佔用。佔用 RAM 的功能與數量關係由以下各章節說明。

6.7.1 NY4 RAM 資源使用說明

共有兩個 Page 分別為 Page0、Page1。每個 Page 共有 48 個 RAM，總共有 96 的 RAM 可以使用。其中的 76 個 RAM 開放給 User 使用，其餘為系統佔用。

注意：實際的 RAM 配置位址，系統會依照使用的功能隨機做配置。

***Q-Code* 功能**	**說明**	**使用 RAM 數**
系統佔用	程式流程控制。NY4A 與 NY4B 分別佔用不同數量。	20 / 22
Action	使用 PlayA / PlayAS 時，每一 Action 通道佔用 9 個 RAM。 若開啟壓縮功能，則每一 Action 通道會多使用 1 個 RAM。	(9+n) * ch
Action Mark State	超過 15 個 action mark state 時，會佔用 2 個 RAM。	1~2
	路徑超過 15 個 Path 時，會再佔用 1 個 RAM。	1~2
BackGround	使用背景 1 路徑時。超過 15 個 Path 時，會再佔用 1 個 RAM。	1~2
	使用背景 2 路徑時。超過 15 個 Path 時，會再佔用 1 個 RAM。	1~2
Delay	使用 Delay 時，每一層使用 3 個 RAM。系統佔用 2 個 RAM。	3*n+2
FD	使用 FD 時，有使用的 Port 會佔用 1 個 RAM。	1~2
Input State	超過 15 個 Input State 時，會佔用 2 個 RAM。	1~2
IR	使用 IR TX 時。DATA Length 每增加 4-bit 就多佔用 1 個 RAM。	1~4
	使用 IR RX 時。DATA Length 每增加 4-bit 就多佔用 1 個 RAM。	1~4
Key Function	Matrix 每一根輸出 pin 對應一組輸入 port 會佔用 3 個 RAM。 Direct 每一 port 會佔用 3 個 RAM。 Debounce 超過 60ms 時，會佔用 2 個 RAM。 系統佔用 2 個 RAM。	(Key / 4)*3 + 3~4
Random	使用 Random 時。當 Random 超過 16 個，會再佔用 1 個 RAM。	3~4
Repeat	在前景使用 Repeat 功能時。	1
	在背景 1 使用 Repeat 功能時。	1
	在背景 2 使用 Repeat 功能時。	1
Subroutine	在 Subroutine 段落中，提供巢狀堆疊功能。 每呼叫第一層就需要佔用 4 個 RAM，呼叫第二層時會再佔用 4 個 RAM。	4 / 8
PauseD	使用 PauseD 時，每一層佔用 3 個 RAM。	3*n
PlayVS	使用 PlayVS 時。	1
QIO	使用 QIO 時。每使用 1 個 Port，則會佔用一個 RAM。系統	1*n+5

Q-Code 功能	說明	使用 RAM 數
	佔用 5 個 RAM。	
Wait	使用 WaitVN 時，佔用 1 個 RAM，系統佔用 1 個 RAM。	1+1
	使用 WaitDN 時，佔用 1 個 RAM，系統佔用 1 個 RAM。	1+1
	使用 WaitAN 時，佔用 2 個 RAM，系統佔用 1 個 RAM。	2+1
Wave Mark	使用 Wave Mark 時。當 Wave Mark State 超過 15 個，會再佔用 1 個 RAM。	2~3

6.7.2 NY5 RAM 資源使用說明

NY5 共有四個 Page 分別為 Page0、Page1、Page2、Page3。每個 Page 共有 56 個 RAM，總共有 224 的 RAM 可以使用。其中的 188 個 RAM 開放給 User 使用，其餘為系統佔用。

注意：實際的 RAM 配置位址，系統會依照使用的功能隨機做配置。

Q-Code 功能	說明	使用 RAM 數量
系統佔用	程式流程控制。(NY5A / NY5B / NY5C*20 I/O; NY5C*24 I/O)	36 / 36 / 36 / 38
Action	使用 PlayA/PlayAS 時，每一 Action 通道佔用 10 個 RAM。 若開啟壓縮功能，則每一 Action 通道會多使用 1 個 RAM。	(10+n)*ch
	使用 PlayA/PlayAS 時，且設定 InterruptService = Action，則會多佔用 7 個 RAM。	(10+n)*ch+7
Action Mark State	超過 15 個 action mark state 時，會佔用 2 個 RAM。	1~2
	路徑超過 15 個 Path 時，會再佔用 1 個 RAM。	1~2
BackGround	使用背景 1 路徑時。超過 15 個 Path 時，會再佔用 1 個 RAM。	1~2
	使用背景 2 路徑時。超過 15 個 Path 時，會再佔用 1 個 RAM。	1~2
Delay	使用 Delay 時，每一層使用 3 個 RAM。系統佔用 2 個 RAM。	3*n+2
FD	使用 FD 時，有使用的 Port 會佔用 1 個 RAM。	1~6
Input State	超過 16 個 Input State 時，會佔用 2 個 RAM。	1~2
IR	使用 IR TX 時。DATA Length 每增加 4-bit 就多佔用 1 個 RAM。	1~4
	使用 IR RX 時。DATA Length 每增加 4-bit 就多佔用 1 個 RAM。	1~4
Interrupt Path	使用 Interrupt Path 時。	8~12
Key Function	Matirx 每一根輸出 pin 對應一組輸入 port 會佔用 3 個 RAM。 Direct 每一 port 會佔用 3 個 RAM。 Debounce 超過 60ms 時，會佔用 2 個 RAM。 系統佔用 2 個 RAM。	(Key / 4)*3 + 3~4
Melody Mark	使用 Melody Mark 時。當 Melody Mark 編號超過 15 時，會再佔用 1 個；當 Melody Mark State 超過 15 個，會再佔用 1 個 RAM。	2~4
Note On	使用 Note On 時。當 Note On State 超過 15 個，會再佔用 1 個 RAM。	2~3

Q-Code 功能	說明	使用 RAM 數量
PauseD	使用 PauseD 時，每一層佔用 3 個 RAM。	3*n
PauseDown / ResumeUp	使 PauseDown/ResumeUp 時，則佔用 4 個 RAM。	4
PauseV	使用 PauseV 時。	1
PlayM	使用 PlayM + 4 通道 PCT Mode。	99
	使用 PlayM + 4 通道 ADSR Mode（自動開啟 InterruptService = Melody）。	117
	使用 PlayM 時，且設定 InterruptService = Melody，則會多佔用 7 個 RAM。	7
PlayMS	使用 PlayMS 功能時，除了原本 PlayM 外，需多佔用 1 個。	1
PlayVS	在前景使用 PlayVS 時，佔用 1 個 RAM。	1
	在背景 1 使用 PlayVS 時，佔用 1 個 RAM。	1
	在背景 2 使用 PlayVS 時，佔用 1 個 RAM。	1
QIO	使用 QIO 時，每使用 1 個 Port，則會佔用一個 RAM。系統佔用 6 個 RAM。	1*n+6
	使用 QIO 時，且設定 InterruptService = QIO，則會多佔用 10 個 RAM。	(1*n+6)+10
Random	使用 Random 時。當 Random 超過 16 個，會再佔用 1 個 RAM。	3~4
Repeat	在前景使用 Repeat 功能時。	1
	在背景 1 使用 Repeat 功能時。	1
	在背景 2 使用 Repeat 功能時。	1
Single Play	使用 Single Play 時。	2
Subroutine	在 Subroutine 段落中，提供巢狀堆疊功能。 每呼叫第一層就需要佔用 4 個 RAM，呼叫第二層時會再佔用 4 個 RAM。	4 / 8
Wait	使用 WaitVN 時，佔用 2 個 RAM，系統佔用 1 個 RAM。	2+1
	使用 WaitDN 時，佔用 1 個 RAM，系統佔用 1 個 RAM。	1+1
	使用 WaitAN 時，佔用 2 個 RAM，系統佔用 1 個 RAM。	2+1
	使用 WaitMN 時，佔用 1 個 RAM，系統佔用 1 個 RAM。	1+1
Wave Mark	使用 Wave Mark 時。當 Wave Mark State 超過 15 個，會再佔用 1 個 RAM。	2~3

6.7.3 NY5+ RAM 資源使用說明

總共有 215 的 RAM 可以使用。系統會依照使用的功能多寡，動態決定所有功能所需的 RAM 數量；因此將不必要的功能關閉以增加可使用的 RAM 數量。可使用的 RAM 數量會在專案執行完 Build 動作後列在 Build Message 視窗內。

6.7.4 NY6 RAM 資源使用說明

共有六個 Page：Page0 ~ Page5。每個 Page 共有 56 個 RAM，總共有 336 的 RAM 可以使用。系統會依照使用的功能多寡，動態決定所有功能所需的 RAM 數量；因此將不必要的功能關閉以增加可使用的 RAM 數量。可使用的 RAM 數量會在專案執行完 Build 動作後列在 Build Message 視窗內。

6.7.5 NY7 RAM 資源使用說明

共有兩個 Page 分別為 Page0，Page1。每個 Page 共有 224 個 RAM，總共有 448 的 RAM 可以使用。系統會依照使用的功能多寡，動態決定所有功能所需的 RAM 數量；因此將不必要的功能關閉以增加可使用的 RAM 數量。可使用的 RAM 數量會在專案執行完 Build 動作後列在 Build Message 視窗內。

6.7.6 NY9T RAM 資源使用說明

在 *Q-Code* 中，使用到特定的功能或是指令時，會佔用到 RAM。根據使用不同的功能或是指令，佔用的 RAM 數量皆不同。系統佔用的規則為從編號後面的 RAM 開始往前佔用。佔用 RAM 的功能與數量關係表，如下表所示。

NY9T001A / NY9T004A RAM 共有 1 個 Page 為 Page0，總共有 48 的 RAM 可以使用。其中的 41 個 RAM 開放給 User 使用，其餘為系統佔用。

注意：實際的 RAM 配置位址，系統會依照使用的功能隨機做配置。

***Q-Code* 功能**	**說明**	**使用 RAM 數量**
Reserved	程式流程控制。	7
AutoJudge_Calibrate	使用觸摸鍵自動校正時，佔用 1 個 RAM。	1
Enforce_Calibrate	使用觸摸鍵強制自動校正時，佔用 1 個 RAM。	1
Process	使用前景指令時，佔用 4 個 RAM。	4
	使用背景 1 的指令時，佔用 4 個 RAM。	4
	使用背景 2 的指令時，佔用 4 個 RAM。	4
IO	根據不同 Body 的輸出埠數量，所佔用的數量有所不同。	1*n
Action	使用 PlayA / PlayAS 時，每一通道佔用 9 個 RAM。	9
	使用 Period 功能時，每一通道佔用 1 個 RAM。	1
	使用 Extension 功能時，每一通道佔用 6 個 RAM。	6
	使用 Loop 功能時，通道 1~4 會佔用 1 個 RAM，通道 5~8	1~2

Q-Code 功能	說明	使用 RAM 數量
	會佔用 1 個 RAM。	
	播放整組 VIO 時，會根據定義的輸出埠數量佔用 RAM。	1*n
	使用 PlayA / PlayAS 時，且設定 Action = INT，則會多佔用 6~7 個 RAM。	8*ch+(6~7)
PWM-IO	使用 PlayPWM / PlayPWMS / PWMOut 時，佔用 1 個 RAM。	1
BackGround	使用背景 1 路徑時。超過 15 個 Path 時，會再佔用 1 個 RAM。	1~2
	使用背景 2 路徑時。超過 15 個 Path 時，會再佔用 1 個 RAM。	1~2
Delay	使用 Delay 時，每一層使用 3 個 RAM。系統佔用 2 個 RAM。	3*n+2
Input State	超過 15 個 Input State 時，會佔用 2 個 RAM。	1~2
Key	每個 Input Port 最多對應 4 個鍵，原則上每 4 個鍵，會佔用 1 個 RAM。	(Key/4)+2
	Debounce 超過 60ms 時，會佔用 2 個 RAM。	1~2
Touch-Key	每個 Input Port 最多對應 4 個鍵，原則上每 4 個鍵，會佔用 1 個 RAM。	(Key/4)+3
Random	使用 Random 時。當 Random 超過 16 個，會再佔用 1 個 RAM。	1~2
Repeat	在前景使用 Repeat 功能時。	1
	在背景 1 使用 Repeat 功能時。	1
	在背景 2 使用 Repeat 功能時。	1
Subroutine	使用 Subroutine 功能時，佔用 4 個 RAM。	4
Wait	使用 WaitDN 時，佔用 1 個 RAM，系統佔用 1 個 RAM。	1+1
	使用 WaitAN 時，佔用 1 個 RAM，系統佔用 1 個 RAM。	1+1
	使用 WaitPN 時，佔用 1 個 RAM，系統佔用 1 個 RAM。	1+1
Special Path	使用 4ms 路徑時，佔用 1 個 RAM。	1
	使用 Enforce_Calibrate 路徑時，佔用 1 個 RAM。	
Radj	使用 Radj 相關指令時，佔用 1 個 RAM。	1

NY9T008A / NY9T016A RAM 共有兩個 Page 分別為 Page0、Page1。每個 Page 共有 48 個 RAM，總共有 96 的 RAM 可以使用。其中的 89 個 RAM 開放給 User 使用，其餘為系統佔用。

注意：實際的 RAM 配置位址，系統會依照使用的功能隨機做配置。

Q-Code 功能	說明	使用 RAM 數量
Reserved	程式流程控制。	7
AutoJudge_Calibrate	使用觸摸鍵自動校正時，佔用 1 個 RAM。	1

Q-Code 功能	說明	使用 RAM 數量
Enforce_Calibrate	使用觸摸鍵強制自動校正時，佔用 1 個 RAM。	1
Process	使用前景指令時，佔用 4 個 RAM。	4
	使用背景 1 的指令時，佔用 4 個 RAM。	4
	使用背景 2 的指令時，佔用 4 個 RAM。	4
IO	根據不同 Body 的輸出埠數量，所佔用的數量有所不同。	1*n
Arithmetic Logic	使用乘除法時，會佔用 2 個 RAM。	2
Action	使用 PlayA / PlayAS 時，每一通道佔用 9 個 RAM。	9
	使用 Period 功能時，每一通道佔用 1 個 RAM。	1
	使用 Extension 功能時，每一通道佔用 6 個 RAM。	6
	使用 Loop 功能時，通道 1~4 會佔用 1 個 RAM，通道 5~8 會佔用 1 個 RAM。	1~2
	播放整組 VIO 時，會根據定義的輸出埠數量佔用 RAM。	1*n
	使用 PlayA / PlayAS 時，且設定 Action = INT，則會多佔用 7~8 個 RAM。	9*ch+(7~8)
PWM-IO	使用 PlayPWM / PlayPWMS / PWMOut 時，佔用 1 個 RAM。	1
BackGround	使用背景 1 路徑時。超過 15 個 Path 時，會再佔用 1 個 RAM。	1~2
	使用背景 2 路徑時。超過 15 個 Path 時，會再佔用 1 個 RAM。	1~2
Delay	使用 Delay 時，每一層使用 3 個 RAM。系統佔用 2 個 RAM。	3*n+2
Input State	超過 15 個 Input State 時，會佔用 2 個 RAM。	1~2
Key	每個 Input Port 最多對應 4 個鍵，原則上每 4 個鍵，會佔用 1 個 RAM。	(Key/4)+2
	Debounce 超過 60ms 時，會佔用 2 個 RAM。	1~2
Touch-Key	每個 Input Port 最多對應 4 個鍵，原則上每 4 個鍵，會佔用 1 個 RAM。	(Key/4)+3
Random	使用 Random 時。當 Random 超過 16 個，會再佔用 1 個 RAM。	1~2
Repeat	在前景使用 Repeat 功能時。	1
	在背景 1 使用 Repeat 功能時。	1
	在背景 2 使用 Repeat 功能時。	1
Subroutine	使用 Subroutine 功能時，佔用 4 個 RAM。	4
Wait	使用 WaitDN 時，佔用 1 個 RAM，系統佔用 1 個 RAM。	1+1
	使用 WaitAN 時，佔用 2 個 RAM，系統佔用 1 個 RAM。	2+1
	使用 WaitPN 時，佔用 1 個 RAM，系統佔用 1 個 RAM。	1+1

Q-Code 功能	說明	使用 **RAM** 數量
Special Path	使用 4ms 路徑時，佔用 1 個 RAM。	1
	使用 Enforce_Calibrate 路徑時，佔用 1 個 RAM。	
Serial Control	使用 Serial Control 功能時，佔用 4 個 RAM。	4
Radj	使用 Radj 相關指令時，佔用 1 個 RAM。	1

6.8 Ri 與 Xi 對應表

Ri（4-bit RAM）與 Xi（8-bit RAM）對應表。如下表：

Ri	Xi	Ri	Xi	Ri	Xi	Ri	Xi	Ri	Xi	Ri	Xi	Ri	Xi	Ri	Xi
R0	X0L	R1	X0H	R2	X1L	R3	X1H	R4	X2L	R5	X2H	R6	X3L	R7	X3H
R8	X4L	R9	X4H	R10	X5L	R11	X5H	R12	X6L	R13	X6H	R14	X7L	R15	X7H
R16	X8L	R17	X8H	R18	X9L	R19	X9H	R20	X10L	R21	X10H	R22	X11L	R23	X11H
R24	X12L	R25	X12H	R26	X13L	R27	X13H	R28	X14L	R29	X14H	R30	X15L	R31	X15H
R32	X16L	R33	X16H	R34	X17L	R35	X17H	R36	X18L	R37	X18H	R38	X19L	R39	X19H
R40	X20L	R41	X20H	R42	X21L	R43	X21H	R44	X22L	R45	X22H	R46	X23L	R47	X23H
R48	X24L	R49	X24H	R50	X25L	R51	X25H	R52	X26L	R53	X26H	R54	X27L	R55	X27H
R56	X28L	R57	X28H	R58	X29L	R59	X29H	R60	X30L	R61	X30H	R62	X31L	R63	X31H
R64	X32L	R65	X32H	R66	X33L	R67	X33H	R68	X34L	R69	X34H	R70	X35L	R71	X35H
R72	X36L	R73	X36H	R74	X37L	R75	X37H	R76	X38L	R77	X38H	R78	X39L	R79	X39H
R80	X40L	R81	X40H	R82	X41L	R83	X41H	R84	X42L	R85	X42H	R86	X43L	R87	X43H
R88	X44L	R89	X44H	R90	X45L	R91	X45H	R92	X46L	R93	X46H	R94	X47L	R95	X47H
R96	X48L	R97	X48H	R98	X49L	R99	X49H	R100	X50L	R101	X50H	R102	X51L	R103	X51H
R104	X52L	R105	X52H	R106	X53L	R107	X53H	R108	X54L	R109	X54H	R110	X55L	R111	X55H
R112	X56L	R113	X56H	R114	X57L	R115	X57H	R116	X58L	R117	X58H	R118	X59L	R119	X59H
R120	X60L	R121	X60H	R122	X61L	R123	X61H	R124	X62L	R125	X62H	R126	X63L	R127	X63H
R128	X64L	R129	X64H	R130	X65L	R131	X65H	R132	X66L	R133	X66H	R134	X67L	R135	X67H
R136	X68L	R137	X68H	R138	X69L	R139	X69H	R140	X70L	R141	X70H	R142	X71L	R143	X71H
R144	X72L	R145	X72H	R146	X73L	R147	X73H	R148	X74L	R149	X74H	R150	X75L	R151	X75H
R152	X76L	R153	X76H	R154	X77L	R155	X77H	R156	X78L	R157	X78H	R158	X79L	R159	X79H
R160	X80L	R161	X80H	R162	X81L	R163	X81H	R164	X82L	R165	X82H	R166	X83L	R167	X83H
R168	X84L	R169	X84H	R170	X85L	R171	X85H	R172	X86L	R173	X86H	R174	X87L	R175	X87H
R176	X88L	R177	X88H	R178	X89L	R179	X89H	R180	X90L	R181	X90H	R182	X91L	R183	X91H
R184	X92L	R185	X92H	R186	X93L	R187	X93H	R188	X94L	R189	X94H	R190	X95L	R191	X95H
R192	X96L	R193	X96H	R194	X97L	R195	X97H	R196	X98L	R197	X98H	R198	X99L	R199	X99H
R200	X100L	R201	X100H	R202	X101L	R203	X101H	R204	X102L	R205	X102H	R206	X103L	R207	X103H
R208	X104L	R209	X104H	R210	X105L	R211	X105H	R212	X106L	R213	X106H	R214	X107L	R215	X107H
R216	X108L	R217	X108H	R218	X109L	R219	X109H	R220	X110L	R221	X110H	R222	X111L	R223	X111H
R224	X112L	R225	X112H	R226	X113L	R227	X113H	R228	X114L	R229	X114H	R230	X115L	R231	X115H
R232	X116L	R233	X116H	R234	X117L	R235	X117H	R236	X118L	R237	X118H	R238	X119L	R239	X119H
R240	X120L	R241	X120H	R242	X121L	R243	X121H	R244	X122L	R245	X122H	R246	X123L	R247	X123H
R248	X124L	R249	X124H	R250	X125L	R251	X125H	R252	X126L	R253	X126H	R254	X127L	R255	X127H
R256	X128L	R257	X128H	R258	X129L	R259	X129H	R260	X130L	R261	X130H	R262	X131L	R263	X131H
R264	X132L	R265	X132H	R266	X133L	R267	X133H	R268	X134L	R269	X134H	R270	X135L	R271	X135H
R272	X136L	R273	X136H	R274	X137L	R275	X137H	R276	X138L	R277	X138H	R278	X139L	R279	X139H
R280	X140L	R281	X140H	R282	X141L	R283	X141H	R284	X142L	R285	X142H	R286	X143L	R287	X143H
R288	X144L	R289	X144H	R290	X145L	R291	X145H	R292	X146L	R293	X146H	R294	X147L	R295	X147H
R296	X148L	R297	X148H	R298	X149L	R299	X149H	R300	X150L	R301	X150H	R302	X151L	R303	X151H

Ri	Xi	Ri	Xi	Ri	Xi	Ri	Xi	Ri	Xi	Ri	Xi	Ri	Xi	Ri	Xi
R304	X152L	R305	X152H	R306	X153L	R307	X153H	R308	X154L	R309	X154H	R310	X155L	R311	X155H
R312	X156L	R313	X156H	R314	X157L	R315	X157H	R316	X158L	R317	X158H	R318	X159L	R319	X159H
R320	X160L	R321	X160H	R322	X161L	R323	X161H	R324	X162L	R325	X162H	R326	X163L	R327	X163H
R328	X164L	R329	X164H	R330	X165L	R331	X165H	R332	X166L	R333	X166H	R334	X167L	R335	X167H
R336	X168L	R337	X168H	R338	X169L	R339	X169H	R340	X170L	R341	X170H	R342	X171L	R343	X171H
R344	X172L	R345	X172H	R346	X173L	R347	X173H	R348	X174L	R349	X174H	R350	X175L	R351	X175H
R352	X176L	R353	X176H	R354	X177L	R355	X177H	R356	X178L	R357	X178H	R358	X179L	R359	X179H
R360	X180L	R361	X180H	R362	X181L	R363	X181H	R364	X182L	R365	X182H	R366	X183L	R367	X183H
R368	X184L	R369	X184H	R370	X185L	R371	X185H	R372	X186L	R373	X186H	R374	X187L	R375	X187H
R376	X188L	R377	X188H	R378	X189L	R379	X189H	R380	X190L	R381	X190H	R382	X191L	R383	X191H
R384	X192L	R385	X192H	R386	X193L	R387	X193H	R388	X194L	R389	X194H	R390	X195L	R391	X195H
R392	X196L	R393	X196H	-	-	-	-	-	-	-	-	-	-	-	-

6.9 頻率計算公式

6.9.1 NY4 / NY5 頻率計算公式

當使用 PlayV 來播放.wav 檔時，計時器會影響播放的速度。播放速度的公式如下：

TM =（F_{TCS} / F_{SR}）－ 1

TM：計時器數值。

F_{TCS}：系統頻率（播放語音時為 1MHz）。

F_{SR}：播放速度。

播放頻率對照表：

TM(HEX)	SR(Hz)	TM(HEX)	SR(Hz)	TM(HEX)	SR(Hz)	TM(HEX)	SR(Hz)	TM(HEX)	SR(Hz)
F9	4000.0	CB	4901.9	9D	6329.1	6F	8928.5	41	15151.
F8	4016.0	CA	4926.1	9C	6369.4	6E	9009.0	40	15384.
F7	4032.2	C9	4950.5	9B	6410.2	6D	9090.9	3F	15625.
F6	4048.5	C8	4975.1	9A	6451.6	6C	9174.3	3E	15873.
F5	4065.0	C7	5000.0	99	6493.5	6B	9259.2	3D	16129.
F4	4081.6	C6	5025.1	98	6535.9	6A	9345.7	3C	16393.
F3	4098.3	C5	5050.5	97	6578.9	69	9433.9	3B	16666.
F2	4115.2	C4	5076.1	96	6622.5	68	9523.8	3A	16949.
F1	4132.2	C3	5102.0	95	6666.6	67	9615.3	39	17241.
F0	4149.3	C2	5128.2	94	6711.4	66	9708.7	38	17543.
EF	4166.6	C1	5154.6	93	6756.7	65	9803.9	37	17857.
EE	4184.1	C0	5181.3	92	6802.7	64	9900.9	36	18181.
ED	4201.6	BF	5208.3	91	6849.3	63	10000.	35	18518.
EC	4219.4	BE	5235.6	90	6896.5	62	10101.	34	18867.
EB	4237.2	BD	5263.1	8F	6944.4	61	10204.	33	19230.

TM(HEX)	SR(Hz)	TM(HEX)	SR(Hz)	TM(HEX)	SR(Hz)	TM(HEX)	SR(Hz)	TM(HEX)	SR(Hz)
EA	4255.3	BC	5291.0	8E	6993.0	60	10309.	32	19607.
E9	4273.5	BB	5319.1	8D	7042.2	5F	10416.	31	20000.
E8	4291.8	BA	5347.5	8C	7092.2	5E	10526.	30	20408.
E7	4310.3	B9	5376.3	8B	7142.8	5D	10638.	2F	20833.
E6	4329.0	B8	5405.4	8A	7194.2	5C	10752.	2E	21276.
E5	4347.8	B7	5434.7	89	7246.3	5B	10869.	2D	21739.
E4	4366.8	B6	5464.4	88	7299.2	5A	10989.	2C	22222.
E3	4385.9	B5	5494.5	87	7352.9	59	11111.1	2B	22727.
E2	4405.2	B4	5524.8	86	7407.4	58	11235.	2A	23255.
E1	4424.7	B3	5555.5	85	7462.6	57	11363.	29	23809.
E0	4444.4	B2	5586.5	84	7518.8	56	11494.	28	24390.
DF	4464.2	B1	5617.9	83	7575.7	55	11627.	27	25000.
DE	4484.3	B0	5649.7	82	7633.5	54	11764.	26	25641.
DD	4504.5	AF	5681.8	81	7692.3	53	11904.	25	26315.
DC	4524.8	AE	5714.2	80	7751.9	52	12048.	24	27027.
DB	4545.4	AD	5747.1	7F	7812.5	51	12195.	23	27777.
DA	4566.2	AC	5780.3	7E	7874.0	50	12345.	22	28571.
D9	4587.1	AB	5813.9	7D	7936.5	4F	12500.	21	29411.
D8	4608.2	AA	5847.9	7C	8000.0	4E	12658.	20	30303.
D7	4629.6	A9	5882.3	7B	8064.5	4D	12820.	1F	31250.
D6	4651.1	A8	5917.1	7A	8130.0	4C	12987.	1E	32258.
D5	4672.9	A7	5952.3	79	8196.7	4B	13157.	1D	33333.
D4	4694.8	A6	5988.0	78	8264.4	4A	13333.	1C	34482.
D3	4716.9	A5	6024.1	77	8333.3	49	13513.	1B	35714.
D2	4739.3	A4	6060.6	76	8403.3	48	13698.	1A	37037.
D1	4761.9	A3	6097.5	75	8474.5	47	13888.	19	38461.
D0	4784.6	A2	6134.9	74	8547.0	46	14084.	18	40000.
CF	4807.6	A1	6172.8	73	8620.6	45	14285.	17	41666.
CE	4830.9	A0	6211.1	72	8695.6	44	14492.	16	43478.
CD	4854.3	9F	6250.0	71	8771.9	43	14705.	15	45454.
CC	4878.0	9E	6289.3	70	8849.5	42	14925.	14	47619.

例. 利用 RAM 來設定播放速度時

TR1: X0=0x63, PlayV($V0,X0) ；播放速度為 **10KHz**

TR2: PlayV($V0,X0) ；播放速度為 **8KHz**

6.10 特殊路徑中不可使用的功能

特殊路徑 512us、1ms、2ms、4ms、8ms、500ms、1sec、4sec、Int_256us、Int_512us、Int_1ms、interrupt 不支援以下指令。

- **Arithmetic Logic 指令：**乘法指令、除法指令。
- **Condition Jump 指令：**CheckSum?Path。
- **IO 指令：**Px.n=1KHz(sec)。
- **Path 指令：**StopFG、BG(BG1,BG2)、StopBG、StopBG1、StopBG2、Macro、Subroutine、BreakFG。
- **Voice 指令：**所有 Voice 指令。
- **Sentence 指令：**所有 Sentence 指令。
- **SPI 指令：**所有 SPI 指令。
- **Melody 指令：**所有 Melody 指令。
- **Keyboard 指令：**所有 Keyboard 指令。
- **Volume 指令：**所有 Volume 指令。
- **Touchkey 指令：**
  - Ri=Touchkey 不受限制。
  - 其餘 touchkey 指令，除 500ms / 1sec / 4sec 路徑以外皆不可使用。
- **Table 指令：**所有 Table 指令。
- **IR 指令：**所有 IR 指令。
- **SC_RX 指令：**所有 SC_RX 指令。
- **SC_TX 指令：**所有 SC_TX 指令。
- **PWM 指令：**所有 PWM 指令，除 NY9T 500ms / 1sec / 4sec 路徑外皆不可使用。
- **Delay 指令：**所有 Delay 指令。
- **Action 指令：**所有 Action 指令。
- **QFID 指令：**所有 QFID 指令。
- **MISC 指令：**所有 MISC 指令。
  - SW_Reset: 除 NYT9T 500ms / 1sec / 4sec 路徑外皆不可使用。

6.11 串列訊號控制通訊協定（Serial Control Protocol）

可連接一般微控制器，將 NY9T 當成微控制器的按鍵。提供 3 種不同的傳輸協定，分別是 SPI_Like、NY3 Serial_Trigger、IR_Trigger。使用者可以設定成自動偵測或是特定的傳輸協定。

SPI_Like：使用 2 根腳位，達成類似 SPI 傳輸協定。

NY3 Serial_Trigger：使用 2 個腳位來連接 NY3 系列，可將 NY9T 當成 NY3 的按鍵。詳細的連接方式，請參閱 NY3C/3D 規格書。

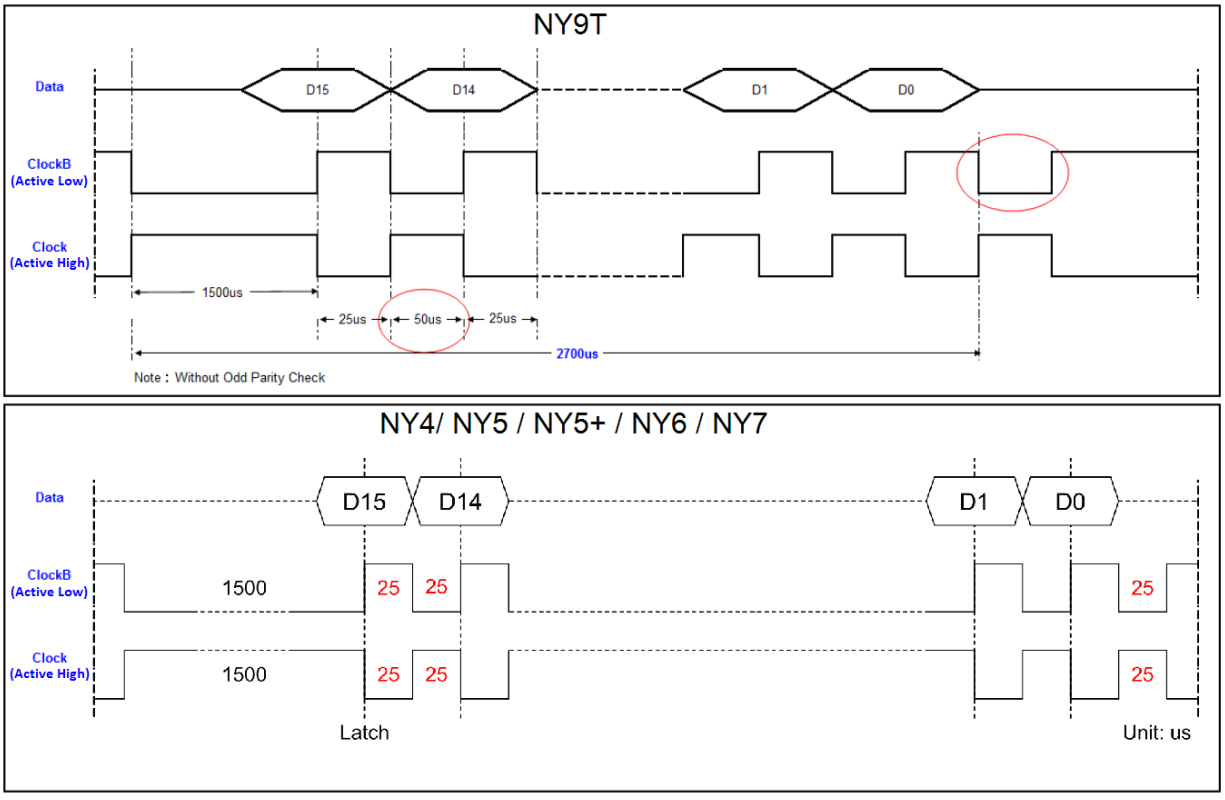
IR_Trigger：使用 1 個腳位來模擬 IR 的接收訊號，可直接使用目前的 NY4/5/7 系列的 IR 通訊。

自動偵測：可由 3 根腳位的設定方式來決定，NY9T 要使用何種傳輸協定。自動偵測的腳位設定方式如下表所示：

自動模式偵測			
Mode	**Pin-3**	**Pin-2**	**Pin-1**
IR_Trigger	X	VDD	X
NY3 Serial_Trigger	Pin-3 connected to Pin-1	X	Pin-3 connected to Pin-1
SPI_Like	No Connected	No Connected	No Connected

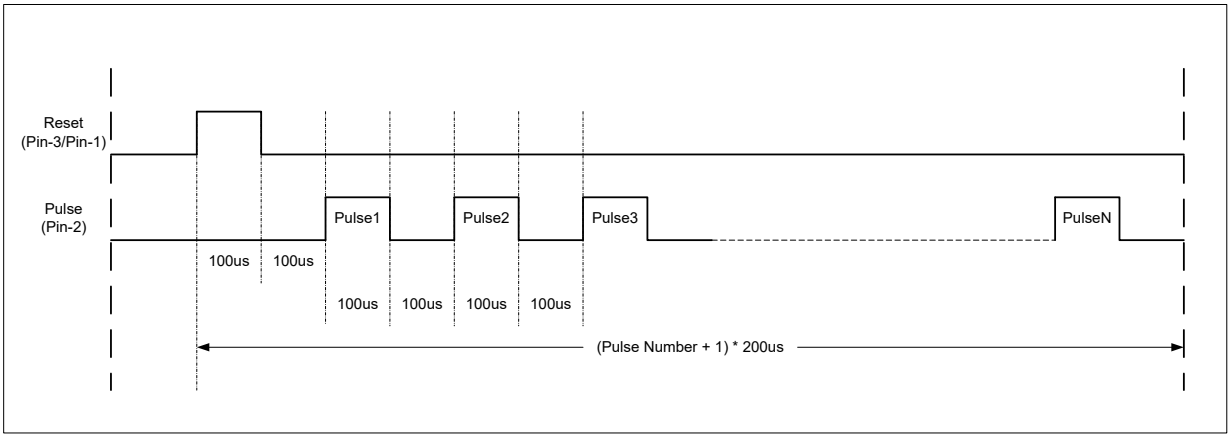
Serial Control 傳輸協定的時序圖如下所示：

SPI_Like Timing

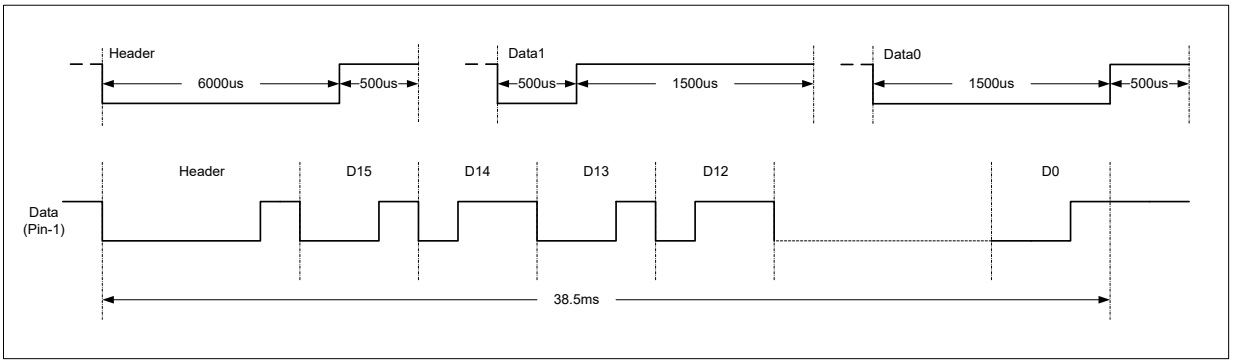


Note: 上圖中的 25us 為最小值，範圍為 25us ~ 100us。

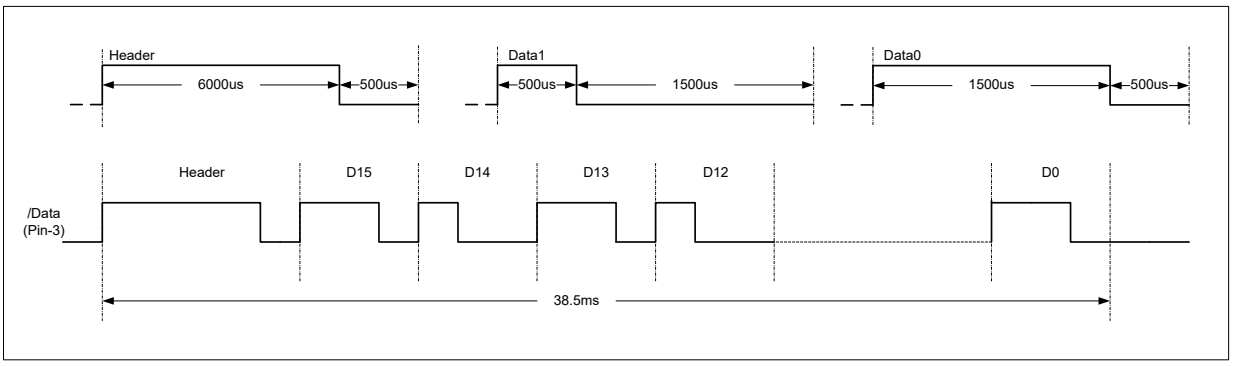
NY3 Serial_Trigger Timing



IR_Trigger(Active Low)



IR_Trigger(Active High)



6.12 *NY5 與 NY5+音量階數對映*

當轉換 NY5 的.qc 至 NY5+時，由於硬體上的差異，因此音量指令的階數與 NY5 輸出並不相同，以下表格列出在不同設定下，音量輸出相同的階數對映。

Voltage = 3.0V & PWM Current = Normal																
	Vol															
NY5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NY5+	2	5	7	8	9	10	10	11	12	12	12	13	13	13	14	15

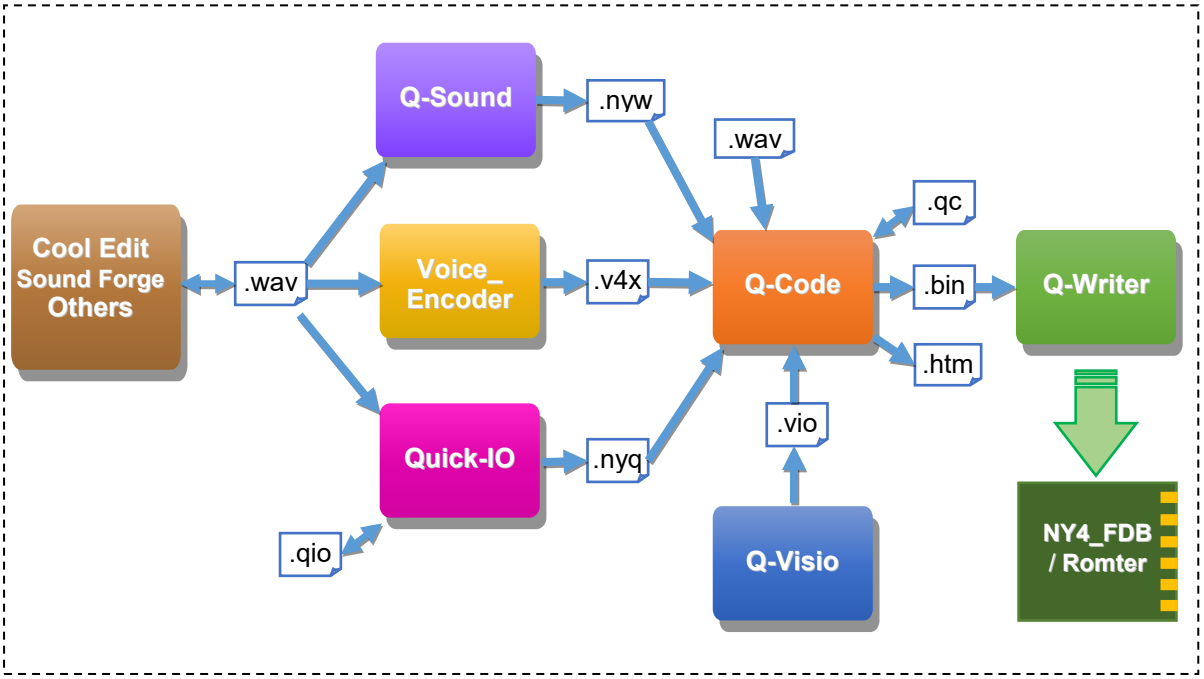
Voltage = 3.0V & PWM Current = Large																
	Vol															
NY5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NY5+	4	7	10	11	12	12	13	13	14	14	15	15	15	15	15	15

Voltage = 4.5V & PWM Current = Normal																
	Vol															
NY5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NY5+	3	6	6	7	8	9	9	10	10	10	10	11	12	12	13	13

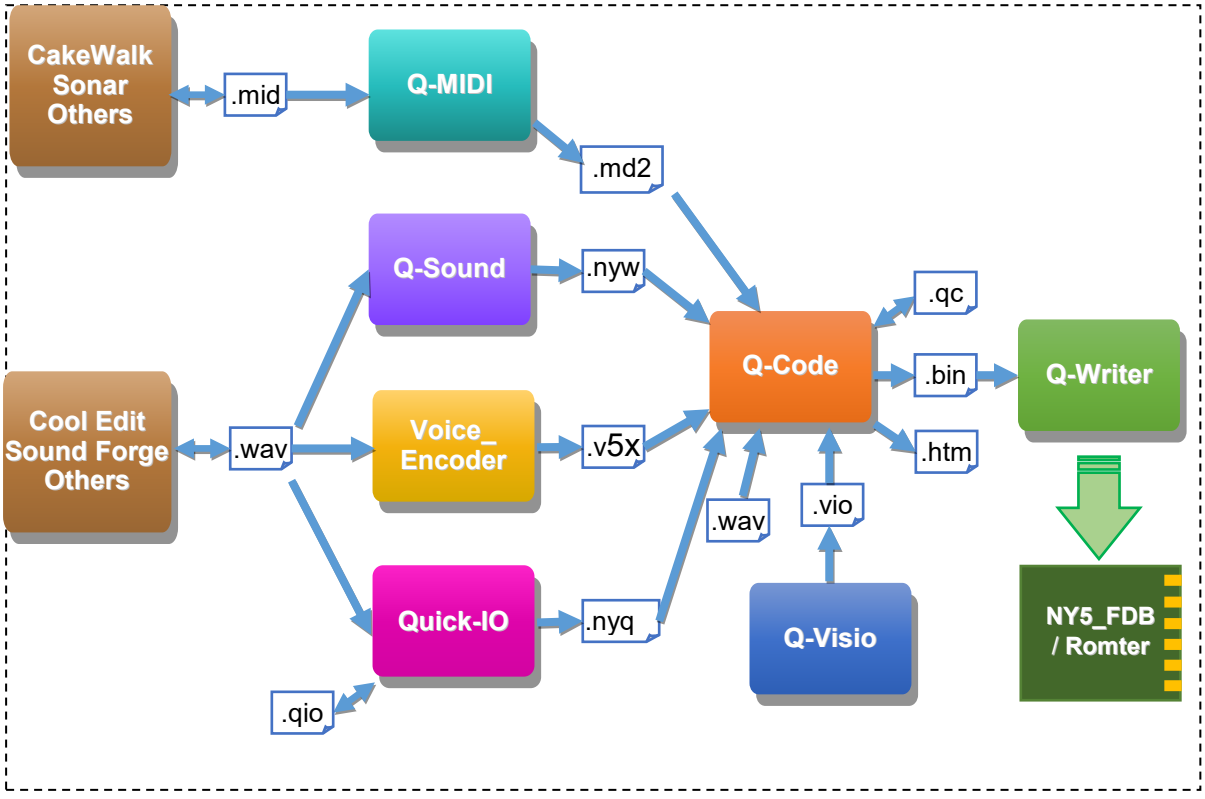
Voltage = 4.5V & PWM Current = Large																
	Vol															
NY5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NY5+	3	6	7	8	9	10	11	12	13	14	15	15	15	15	15	15

6.13 *Q-Code* 開發流程圖

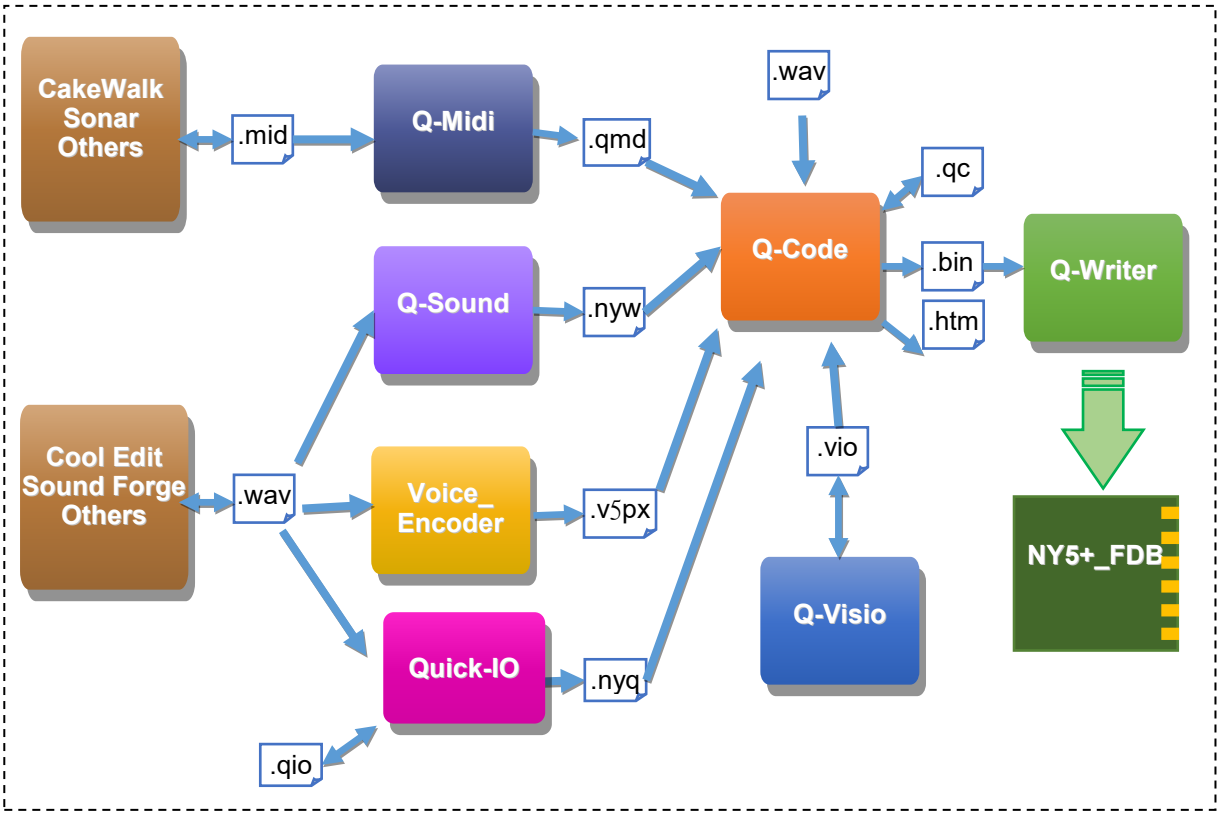
6.13.1 NY4 開發流程圖



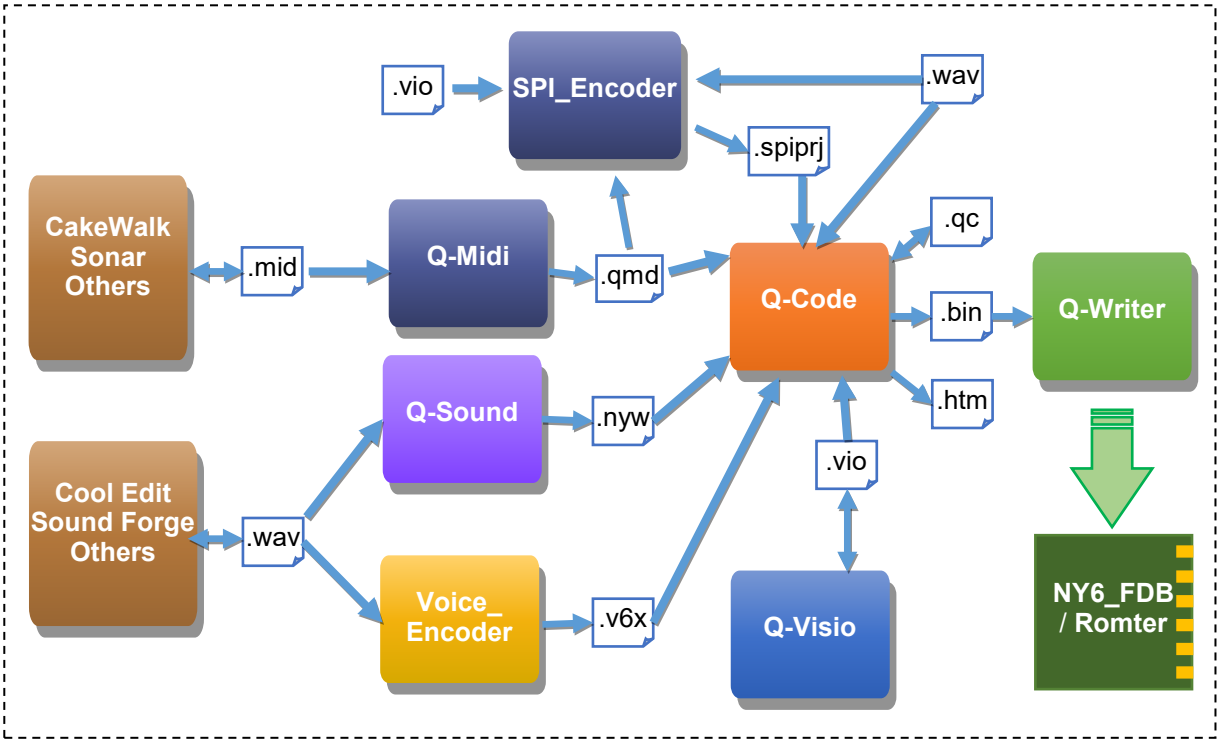
6.13.2 NY5 開發流程圖



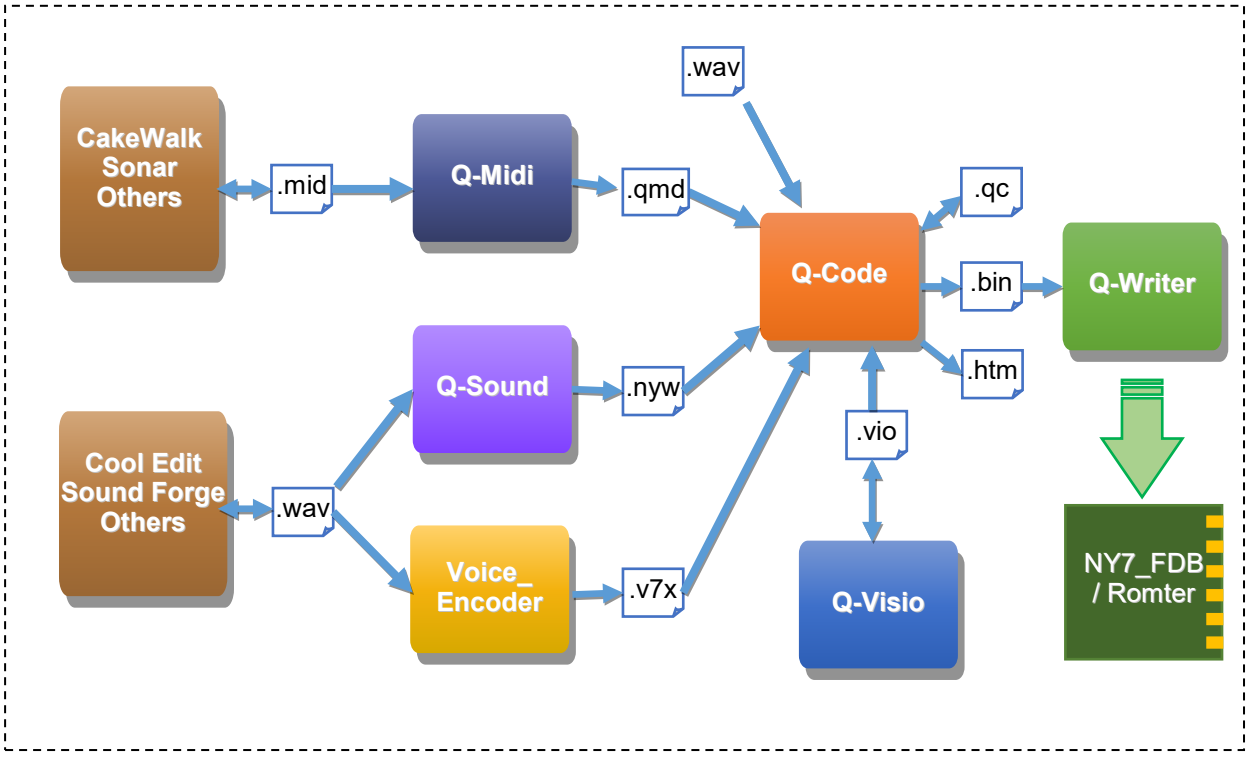
6.13.3 NY5+開發流程圖



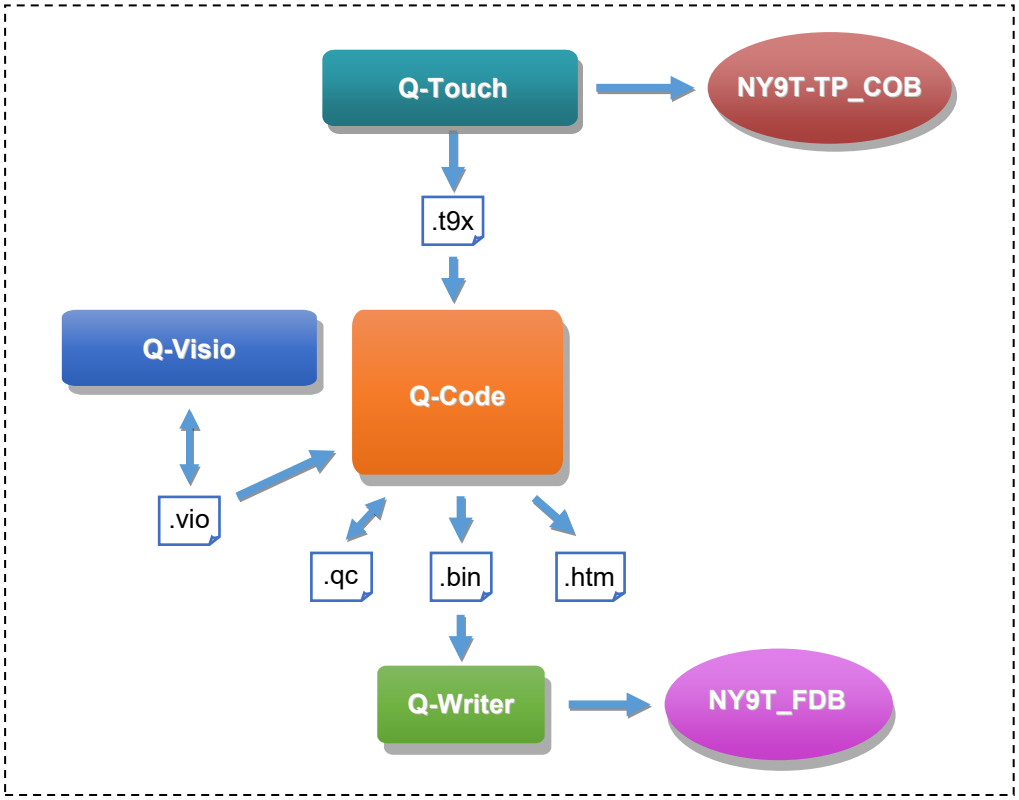
6.13.4 NY6 開發流程圖



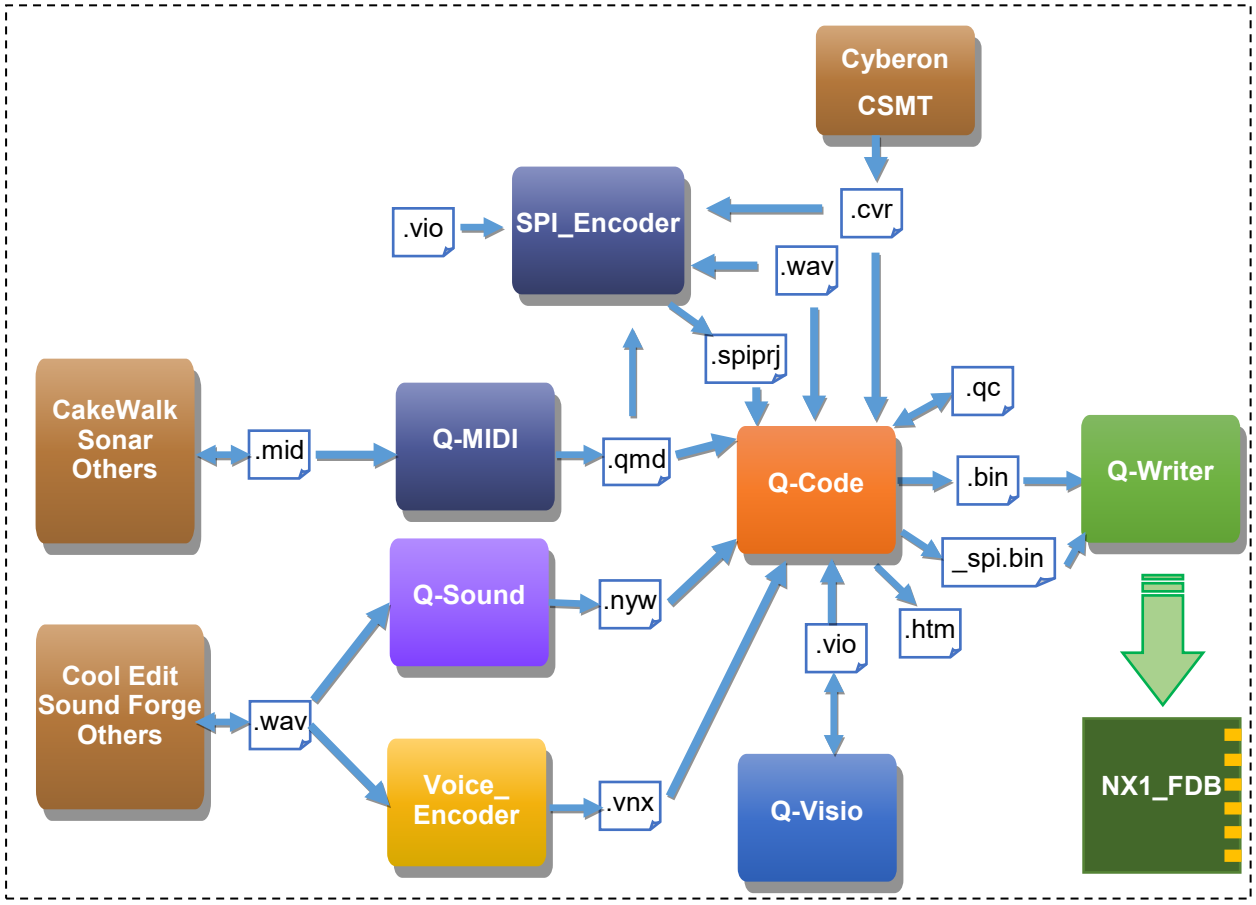
6.13.5 NY7 開發流程圖



6.13.6 NY9T 開發流程圖

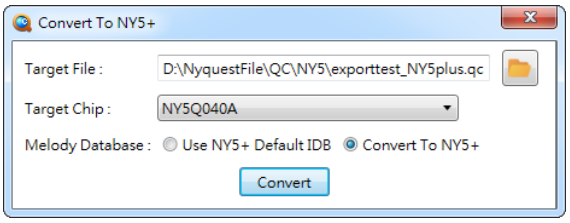


6.13.7 NX1 開發流程圖



6.14 Convert To N5+ 流程

按下 Converter To NY5+ 會出現下列視窗。



Target File（檔案路徑）：設定檔案路徑。

Target Chip（目標 IC）：設定轉換後的 IC。盡量選擇腳位數目和 ROM Size 相近的 IC，可以避免轉換後功能的誤差。

Melody Data Base：選擇 .md2 轉換成 .qmd 要使用哪一種 IDB，可直接使用目前 .md2 的 IDB 或選擇使用 NY5+預設的 IDB。

轉換成功後，系統會自動開啟 .qc 並用預設的瀏覽器顯示下圖的轉換報告。

Nyquest Convert Report

***Project Information:**

(1). Target Body: NY5Q040A
(2). File Name: convertqc2.qc
(3). Tool Version: Q-Code 6.60
(4). Date: 2021/06/01

***Convert Information:**

Location	Description
Line 2, Col 1	Option ICBody is changed to NY5Q040A.
Line 5, Col 1	NY5+ does not support Package_Test option, removed.

*注意：**NY5** 才支援此功能。*

7 改版記錄

版本	日期	內 容 描 述	修正頁
2.0	2012/01/02	1. 整合 NY4/NY5 *Q-Code* 使用手冊。 2. NY4 Option 部分 – 新增 Frame Rate。 – 新增 Power On Trigger。 – 新增 Short Debounce。 3. NY4 段落部分 – 新增 Action 段落。 – 修改 Wave Mark 段落。 – 新增 QIO Custom 段落。 4. NY4 指令部分 (1) 算數邏輯指令 – 新增 Ri.n = Rj.n 指令。 – 新增 Ri = data + Rj 指令。 – 新增 Ri = data - Rj 指令。 – 新增 Ri.n = Rj.n 指令。 – 新增[Ri, Rj] = Rk * Rl 指令。 – 新增[Ri, Rj] = Rk * data 指令。 – 新增[Ri, Rj] = data * Rk 指令。 – 新增[Ri, Rj] = Rk / Rl 指令。 – 新增[Ri, Rj] = Rk / data 指令。 – 新增[Ri, Rj] = data / Rk 指令。 – 新增 Ri.n = XjL.n 指令。 – 新增 Ri.n = XjH.n 指令。 – 新增 Ri.n = Xj.n 指令。 – 新增 XiL.n = Rj.n 指令。 – 新增 XiH.n = Rj.n 指令。 – 新增 Xi.n = Xj.n 指令。 – 新增 Xi = data + Xj 指令。 – 新增 Xi = data - Xj 指令。 – 新增[Xi, Xj] = Xk * Xl 指令。 – 新增[Xi, Xj] = Xk * data 指令。 – 新增[Xi, Xj] = data * Xk 指令。 – 新增[Xi, Xj] = Xk / Xl 指令。 – 新增[Xi, Xj] = Xk / data 指令。 – 新增[Xi, Xj] = data / Xk 指令。	

版本	日期	內 容 描 述	修正頁
		(2) 條件跳躍指令 – 新增 Delay(n)?Path 指令。 – 新增 Action(ch)?Path 指令。 – 新增 Switch(Px[d x d x]) 指令。 (3) I/O 指令 – 新增 Ri.n = Px.n 指令。 – 新增 Px.n = Ri.n 指令。 – 新增 Px.n = Py.n 指令。 – 新增 Px = Py + Ri 指令。 – 新增 Px = Py + data 指令。 – 新增 Px = Py - Ri 指令。 – 新增 Px = Py - data 指令。 – 新增 Px = Py | Ri 指令。 – 新增 Px = Py | data 指令。 – 新增 Px = Py ^ Ri 指令。 – 新增 Px = Py ^ Rata 指令。 – 新增 Px = Py & Ri 指令。 – 新增 Px = Py & data 指令。 – 新增 Ri = Py + data 指令。 – 新增 Ri = Py - data 指令。 – 新增 Ri = Py | Rj 指令。 – 新增 Ri = Py | data 指令。 – 新增 Ri = Py ^ Rj 指令。 – 新增 Ri = Py ^ data 指令。 – 新增 Ri = Py & Rj 指令。 – 新增 Ri = Py & data 指令。 – 新增 XiL.n = Px.n 指令。 – 新增 XiH.n = Px.n 指令。 – 新增 Xi.n = Px.n 指令。 (4) 路徑指令 – 新增 Break 指令。 – 新增 StopFG 指令。 – 新增 Label(Pathname)指令。 (5) 語音指令 – 新增 PlayVS 指令。 – 新增 WaitVN 指令。 (6) 紅外線指令	

版本	日期	內 容 描 述	修正頁
		– 新增 TX(Ri:Rj:Rk:Rl）指令。 (7) 時間延遲指令 – 新增 Delay(Ri:Rj:Rk)指令。 – 新增 WaitDN(n)指令。 – 新增 StopD(n)指令。 – 新增 PauseD(n)指令。 – 新增 ResumeD(n)指令。 (8) 動作指令 – 新增 PlayA 指令。 – 新增 PlayAS 指令。 – 新增 WaitAN(Ch) 指令。 – 新增 PauseA(Ch) 指令。 – 新增 ResumeA(Ch) 指令。 – 新增 StopA(Ch) 指令。 (9) 一般指令 – 新增 Wave Mark State 指令。 – 新增 Key_CLR 指令。 – 新增 Key_ON 指令。 – 新增 Key_OFF 指令。 – 新增 Pause(n) 指令。 – 新增 Resume(n) 指令。 5. NY5 Option 部分 – 新增 Frame Rate。 – 新增 Power On Trigger。 – 新增 Short Debounce。 – 新增 Interrupt Service。 6. NY5 段落部分 – 新增 Action 段落。 – 修改 Wave Mark 段落。 – 修改 Melody Mark 段落。 – 新增 Melody Database 段落。 – 修改 Note On 段落。 – 新增 QIO Custom 段落。 – 移除 RFC 段落。 7. NY5 指令部分 (1) 算數邏輯指令 – 新增 Ri.n = Rj.n 指令。	

版本	日期	內 容 描 述	修正頁
		– 新增 Ri = data + Rj 指令。 – 新增 Ri = data - Rj 指令。 – 新增 Ri.n = Rj.n 指令。 – 新增[Ri, Rj] = Rk * Rl 指令。 – 新增[Ri, Rj] = Rk * data 指令。 – 新增[Ri, Rj] = data * Rk 指令。 – 新增[Ri, Rj] = Rk / Rl 指令。 – 新增[Ri, Rj] = Rk / data 指令。 – 新增[Ri, Rj] = data / Rk 指令。 – 新增 Ri.n = XjL.n 指令。 – 新增 Ri.n = XjH.n 指令。 – 新增 Ri.n = Xj.n 指令。 – 新增 XiL.n = Rj.n 指令。 – 新增 XiH.n = Rj.n 指令。 – 新增 Xi.n = Xj.n 指令。 – 新增 Xi = data + Xj 指令。 – 新增 Xi = data - Xj 指令。 – 新增[Xi, Xj] = Xk * Xl 指令。 – 新增[Xi, Xj] = Xk * data 指令。 – 新增[Xi, Xj] = data * Xk 指令。 – 新增[Xi, Xj] = Xk / Xl 指令。 – 新增[Xi, Xj] = Xk / data 指令。 – 新增[Xi, Xj] = data / Xk 指令。 (2) 條件跳躍指令 – 新增 Voice(Ch)?Path 指令。 – 新增 PauseV(Ch)?Path 指令。 – 新增 Delay(n)?Path 指令。 – 新增 Action(Ch)?Path 指令。 – 新增 Mixctrl = data?Path 指令。 – 新增 Mixctrl != data?Path 指令。 – 新增 Switch(Px[d x d x]) 指令。 (3) I/O 指令 – 新增 Ri.n = Px.n 指令。 – 新增 Px.n = Ri.n 指令。 – 新增 Px.n = Py.n 指令。 – 新增 Px = Py + Ri 指令。 – 新增 Px = Py + data 指令。	

版本	日期	內 容 描 述	修正頁
		– 新增 Px = Py - Ri 指令。 – 新增 Px = Py - data 指令。 – 新增 Px = Py | Ri 指令。 – 新增 Px = Py | data 指令。 – 新增 Px = Py ^ Ri 指令。 – 新增 Px = Py ^ Rata 指令。 – 新增 Px = Py & Ri 指令。 – 新增 Px = Py & data 指令。 – 新增 Ri = Py + data 指令。 – 新增 Ri = Py - data 指令。 – 新增 Ri = Py | Rj 指令。 – 新增 Ri = Py | data 指令。 – 新增 Ri = Py ^ Rj 指令。 – 新增 Ri = Py ^ data 指令。 – 新增 Ri = Py & Rj 指令。 – 新增 Ri = Py & data 指令。 – 新增 XiL.n = Px.n 指令。 – 新增 XiH.n = Px.n 指令。 – 新增 Xi.n = Px.n 指令。 (4) 路徑指令 – 新增 Break 指令。 – 新增 StopFG 指令。 – 新增 Label(Pathname)指令。 (5) 語音指令 – 新增 PlayVS 指令。 – 新增 WaitVN(Ch)指令。 (6) 音樂指令 – 新增 PlayTS 指令。 – 新增 WaitTN 指令。 – 新增 PlayMS 指令。 – 新增 WaitMN 指令。 – 新增 Tempo(Ri:Rj) 指令。 (7) 紅外線指令 – 新增 TX(Ri:Rj:Rk:Rl）指令。 (8) 時間延遲指令 – 新增 Delay(Ri:Rj:Rk)指令。 – 新增 WaitDN(n)指令。	

版本	日期	內 容 描 述	修正頁
		– 新增 StopD(n)指令。 – 新增 PauseD(n)指令。 – 新增 ResumeD(n)指令。 (9) 動作指令 – 新增 PlayA 指令。 – 新增 PlayAS 指令。 – 新增 WaitAN(Ch) 指令。 – 新增 PauseA(Ch) 指令。 – 新增 ResumeA(Ch) 指令。 – 新增 StopA(Ch) 指令。 (10) 一般指令 – 新增 Wave Mark State 指令。 – 新增 Melody Mark State 指令。 – 新增 Note ON State 指令。 – 新增 Key_CLR 指令。 – 新增 Key_ON 指令。 – 新增 Key_OFF 指令。 – 新增 Pause(n) 指令。 – 新增 Resume(n) 指令。 – 新增 ReadTempo(Ri:Rj) 指令。 – 新增 ReadChannel(Ri) 指令。 – 新增 PauseDown 指令。 – 新增 ResumeUp 指令。 – 新增 Audio_ON 指令。 – 新增 Audio_OFF 指令。 8. 附錄 (1) 新增相關工具介紹。 (2) 新增 *Q-Code* 基本功能介紹。 (3) 新增 *Q-Code* 應用實例。 (4) 新增在 *Q-Code* 中，使用 *Wave_Splitter*。 (5) 修改 *Q-Code* 指令表。 (6) 修改 RAM 資源使用說明。 (7) 修改 Ri 與 Xi 對應表。 (8) 修改 *Q-Code* 開發流程圖。	

版本	日期	內 容 描 述	修正頁
2.1	2012/01/31	1. 增加 ASM 使用說明。 2. 修正 NY4 *Q-Code* RAM 資源使用說明。 3. 修正 NY5 *Q-Code* RAM 資源使用說明。	53, 131 258 260
2.2	2012/03/23	1. WDT_Clear 指令更名成 WDT_CLR。 2. 修改 Interrupt Path 中，所禁用的指令。 3. 修正 NY4 *Q-Code* RAM 資源使用說明。 4. 修正 NY5 *Q-Code* RAM 資源使用說明。	98, 198 137 258 260
2.3	2012/05/31	1. 新增暫存器 PlayVS 播放指令。 2. 新增暫存器 PlayTS 播放指令。 3. 新增暫存器 PlayMS 播放指令。 4. 增加 Table PlayV/PlayVS 範例說明。 5. 增加 Table PlayT/PlayTS 範例說明。 6. 增加 Table PlayM/PlayMS 範例說明。 7. 更換 IC Body 圖示。 8. 修改語音段落說明。 9. 修改在 *Q-Code* 中如何播放 *Q-Sound* 處理後的檔案說明。 10. 移除 Multi-channel PlayV 指令。 11. 增加 Audio_ON/Auiod_OFF 注意事項。 12. 修改在 *Q-Code* 程式中使用 *Q-Sound* 說明。	79, 160 167 170 80, 161 168 171 33, 102 35, 106 37, 108 160 197 248
2.4	2012/08/30	1. 修改執行（Run）選單內容。 2. 修改 ICE/Romter 連接狀態（ICE/Romter Connect Status）內容。 3. *Q-Sound* 取代 *Wave_Splitter* 在語音段落的功能。 4. 修改音樂檔段落加入/刪除檔案圖示。 5. 修改樂器段落加入/刪除檔案圖示。 6. 新增 Configure Download 功能。 7. 修改按鍵偵測時間選項。 8. 新增特殊路徑 – 主迴圈 9. 新增一般指令 – Debounce(Time)指令。 10. 修改 NY4 RAM 資源使用說明。 11. 修改 NY5 RAM 資源使用說明。 12. 修改 *Q-Code* 開發流程圖	26 31 35, 106 110 126 267 35, 105 57, 136 97, 138 258 260 268

版本	日期	內 容 描 述	修正頁
2.5	2012/11/01	1. 新增支援 NY5AxxxB、NY5BxxxB IC 系列。 2. 新增系統時脈可選 1 或 2 MHz 功能。 3. NY5AxxxB IC 新增支援 PWM Current 功能。 4. 變更 FrameRate 為 Action_FrameRate。	102 103 - 34, 105
2.6	2012/11/30	1. NY4 選項部分 (1) 新增 Action Compression 功能。 2. NY4 段落部分 (1) 修改 Action File 介面。 (2) 新增 Action 輸出參數設定。 3. NY4 指令部分 (1) IO 指令 – Px=[1 X 0 FD An Q] 指令新增 Action 參數。 (2) Action 指令 – 新增多通道訊號播放功能。 4. NY5 選項部分 (1) 新增 Action Compression 功能。 5. NY5 段落部分 (1) 更改 Action File 介面。 (2) 新增 Action 輸出參數設定。 6. NY5 部分 (1) IO 指令 – Px=[1 X 0 FD An Q] 指令新增 Action 參數。 (2) Action 指令 – 新增多通道訊號播放功能。 7. 修改 RAM 資源使用說明。	 34 38 46 73 93 105 108 120 154 189 258, 260
2.7	2013/02/27	1. 新增震盪器的 Time Base 設定。 2. 新增 Action Loop 功能。	103 93, 189
2.8	2014/05/29	新增 ReadRollingCode(Ri:Rj:Rk:Rl:Rm)。	99, 183
2.9	2014/08/20	NY5AxxxA 增加 Tone/ToneS 指令。	173
3.0	2015/02/10	1. 新增 Action 訊號輸出支援 Drive/Sink 設定。 2. 修改 Random 功能的 RAM 資源使用說明。	38, 111 259, 261, 263

版本	日期	內 容 描 述	修正頁
3.1	2015/05/21	1. 更新介面圖片。 2. 更新幫助（Help）功能表。 3. 更新 QIO Custom 範例程式。 4. 修改主迴圈、4 毫秒和中斷路徑上可使用的指令類型。 5. 新增 IR_RX 路徑。 6. 新增[Ri,Rj,Rk,Rl] = RX / [Xi, Xj] = RX 指令。	- 27 56, 133 57, 58, 136, 136 59, 138 90, 135
4.0	2015/08/28	1. 合併 NY7。 2. 新增串列資料接收（Serial Control）。 3. 移除 StopDelay。 4. 修改 RX / TX 指令為 IR_RX / IR_Tx。 5. 更新 SDelay 時間範圍。 6. 修改 ReadTempo 說明。	- 138, 163, 167, 171 - - 142 119
4.1	2015/11/25	1. 移除 Debounce(Time)指令。 2. 修改上電觸發說明。 3. 修改按鍵偵測時間說明。 4. 新增矩陣掃描設定時間說明。 5. 新增 Ri 與 Xi 對應說明。 6. 新增 SPIVol 指令說明。 7. 修改 Instrument(Ch, i)指令說明，支援變數控制 GM 音色。 8. 修改 InstNoteOn 及 DrumNoteOn 指令說明。 9. 新增 Gliss(Note, Semitone, Time)指令說明。 10. 修改 Ri 與 Xi 對應表。	- 39 40 40 77 110 114 124, 125 125 174
4.2	2016/02/02	1. 更新 Shortcut icon。 2. 更新 NY7 上拉電阻說明。 3. NY4 不支援 vol 指令。 4. 更新 Background2 描述。	24, 25 48 82, 126 86

版本	日期	內 容 描 述	修正頁
4.3	2016/05/31	1. 修改按鍵掃描間隔說明。 2. 新增最大琴鍵音數說明。 3. 更新 I/O 段落說明。 4. 新增 PauseA(Ch)?Path、PWMIO?Path 及 PausePWM?Path 指令。 5. 新增 V_Chx_Vol = n 及 V_Chx_Vol = Xi 指令。 6. 新增 SPIReadIndex(Rl:Rk:Rj:Ri)及 SPIReadIndex(Xj:Xi)指令。 7. 修改查表指令說明。 8. 新增脈衝調變 IO 指令說明。 9. 新 增 Direct_Debounce(Time) 、 Matrix_Debounce(Time) 及 Debounce(Time)指令。 10. 更新 NY7 Q-Code 指令表說明。 11. 更新 NY4 / NY5 RAM 資源使用說明。	46 48 52 98, 98 120 125 144 154 173 190 198, 199
4.4.	2016/08/30	1. 編譯(Complie)新增重新編譯(Rebuild)功能。 2. 更新輸入狀態段落(Input State)說明。	25 63
5.0	2016/11/30	1. 移除 NY5AA。 2. 新增 MaxSingleNote 指令。 3. 合併 NY9T。	- 137 194, 201, 207, 210
5.1	2017/02/24	1. 新增 QFID。 2. 新增 Action Mark。	52, 163 69, 164
5.2	2017/05/22	1. 新增 Export Project 功能說明。 2. 新增 RFC。 3. 修改變數運算說明。	23 48, 165 90
5.3	2017/08/21	1. 修改 Conditional Jump 指令說明。 2. 新增 Xi = V_Chx_Vol / Ri = SPIVol / Ri = M_Chx_Vol 指令。 3. 新增 Enforce_Calibrate_Normal / Enforce_Calibrate_Sleep 指令，並移除 Enforce_Calibrate 指令。	92 120, 124, 129 143, 143

版本	日期	內 容 描 述	修正頁
5.4	2017/11/27	1. 新增 IR CRC 說明。 2. 更新 PWMIO 說明。 3. 更新 RFC 說明。 4. 新增 QFID Slow。 5. 更新 MixCtrl 指令說明。 6. 更新 PWM 指令說明。 7. 新增指令 QFID_SlowOn / QFID_SlowOff。 8. 新增指令 RFC_On / RFC_Off。 9. 更新 Audio_OFF 指令說明。	42 48 49 54 141 156 166 167 173
5.5	2018/02/27	1. 更新 QFID 段落。 2. 新增抹除完成（SPI_EraseEnd）指令。 3. SPIReadIndex 指令改為 SPIGetIndex。 4. 新增 SPI Flash 指令。 5. 新增 StopMNote 指令。	55 91 127 128 147
5.6	2018/05/30	1. 更新 Scankey Interval / Matrix Scan Setup Time 說明。 2. 更新 Mute_On / Mute_Off 指令說明。 3. 新增 HoldA 指令。	48 144, 145 176
5.7	2018/08/30	1. 更新 Scankey Interval 選項說明。 2. 修改指令說明格式。	- -
5.8	2018/12/07	1. 合併 NY6、NX1。	-
5.9	2019/2/27	1. 更新操作畫面。 2. 更新 Action Mark State 指令說明。	- 213
6.0	2019/5/31	1. 新增 VR Timeout Extend 選項。 2. 新增 VR_VAD?Path / AGC_On / AGC_Off 指令。 3. 更新 PGA_Gain 指令說明。	75 115, 220, 220 177
6.1	2019/8/29	1. 更新 PGA_Gain 指令說明。 2. 更新 SDelay 指令說明。 3. 更新 Debounce(time)指令說明。	174 198 221

版本	日期	內 容 描 述	修正頁
6.2	2019/11/22	1. 更新 FD 說明。 2. 更新 Factor 說明。 3. 更新 WaitSN 說明。 4. 更新 M_CHx_Vol 說明。	44 50 139 161
6.3	2020/3/17	1. 修改編輯介面 2. 新增偵錯模式 3. 修改訊息視窗 4. 修改隨機產生器選項說明。 5. 修改 Debounce / PowerOnTrigger 選項說明。 6. 修改 Before_PowerOn 說明。 7. 新增 SBC_Loop_On / SBC_Loop_Off 指令。 8. 修改 MixCtrl 指令說明。 9. 新增 SFX 指令。 10. 新增 Real Time Play 指令。 11. 修改 NX1 聲音輸出通道說明。	21 22 22 46 47 - 139 178 209 214 259
6.4	2020/06/05	1. 修改 RAM_Usage 選項說明。 2. 新增 Variable_Compatible 選項說明。 3. 修改 Record 說明。 4. 修改 Output State 說明。 5. 修改 Px=[0 1 An Q FD…] 說明。 6. 新增 [Px.n Px.n…] = [0 1 Q…] 指令。 7. 新增 Ri=VR_Loading 指令。 8. 修改 ReadRollingCode 指令說明。	50 51 57 73 125 126 210 230
6.5	2020/08/31	1. 新增 LED String 段落。 2. 修改 Touchkey 說明。 3. 修改 SPI Flash 說明。 4. 修改 I/O 段落說明。 5. 修改語音辨識段落說明。 6. 新增 WaveID_RX / VT_BeforeRecord / VT_BeforeTraining / VT_AddTagFail 路徑。 7. 新增 MIDI_Pitch / LongInst_Holdtime / ShortInst_HoldTime 指令。 8. 修改鍵盤指令說明。	63 38 43 67 79 106, 109 178, 182 179, 180, 181

版本	日期	內 容 描 述	修正頁
		9. 新增 Touchkey_Count / Touchkey_BGCount 指令。	195
		10.新增 LED String 指令。	216
		11.新增 WaveID 指令。	218
		12.新增 VT_Training / VT_Delete / VT_DeleteAll / VT_TrainingNum 指令。	224
6.6	2020/11/30	1. 更新 UI 圖片。 2. 更新 Record 段落說明。 3. 新增 ADPCM_Loop_On / ADPCM_Loop_Off 指令。 4. 新增 Mask_On / Mask_Off 指令。 5. 新增琴鍵錄音功能。	 61 147 178, 179 182 - 184
6.7	2020/01/18	新增 Melody_OKON_BgNote 選項。	53
6.8	2021/5/20	新增 NY5+系列。	-
6.9	2021/09/30	1. NY5+移除 PauseS / ResumeS / StopS / WaitSN 指令支援。 2. 加入 NX1 EF series。 3. NY5+支援 **[QIO Custom]** 段落。	- - 101
7.0	2021/11/30	1. 更新 Voice Output 選項說明。 2. 更新 **[Record]** 段落說明。 3. 更新 **[I/O]** 段落說明。 4. 更新 LVD_mVn 路徑說明。 5. 更新 Ri = LVD 指令說明。 6. 新增 Robot4 / RT_Robot4。	41 70 76 117 272 251, 256
7.1	2022/2/25	1. 新增 UART / I2C 說明。 2. 新增**[PWM-IO]**段落說明。 3. 更新 LVD_mVn 路徑說明。 4. 移除 LVD_LVn 路徑說明。 5. 修改指令支援說明。 6. 新增 UART / I2C 指令。 7. 更新 PWM 指令說明。 8. 新增 RT_Darth / RT_Darth_Off。	55 87 117 - 222 225 258

版本	日期	內 容 描 述	修正頁
7.2	2022/5/31	1. 修改 FD 選項說明。	55
		2. 新增聽聲辨位功能。	61, 119, 242
		3. 更新 LVD_mVn / LVD_Max 系統路徑說明。	116
		4. 新增 I2C_Start / I2C_Stop / I2C_MReadAck / I2C_MReadNAck / I2C_Ack?Path 指令。	24
		5. 新增 Darth / Darth_Off。	253
		6. 新增 RT_Chorus / RT_Chorus_Off。	258
		7. 更新 RampUp / RampDown 指令說明。	267
		8. 更新 Ri=LVD 指令說明。	282
7.3	2022/8/31	1. 更新 Oscillator NX1 說明。	9
		2. 更新 SPI Flash 說明。	50
		3. 更新 I2C 說明。	56
		4. 新增 LVD 說明。	60
		5. 更新 **[Record]** 說明。	72
		6. 新增 Recorded?Path 指令。	139
		7. 更新 PausePWM / HoldPWM 指令說明。	234, 235
7.4	2022/11/30	1. 新增 Audio Filter。	68
		2. 新增 Common ROM Code。	69
		3. 更新 **[Record]** 說明。	76
		4. 新增 **[QIO]**。	94
		5. 新增 **[Interrupt]**。	117
		6. 新增 Q-Code 特殊路徑。	119
		7. 更新 PWMDuty 指令說明。	246
		8. 更新音效指令說明。	264
		9. 新增 EQ_Filter / EQ_Filter_Off 指令。	284
		10. 新增 Enforce_Wakeup 指令。	291
7.5	2023/02/24	1. 更新 Imformation 內容。	26
		2. 更新 Sound Localization 內容。	65
		3. 更新 Realtime Play 內容。	67
		4. 新增 Sound Effect。	68
		5. 更新 QIO 內容。	94
		6. 新增 V_Chx_Freq 指令。	174

版本	日期	內 容 描 述	修正頁
		7. NX1 新增支援 SPI Flash Command。	184
		8. 更新 Gliss 指令說明。	212
		9. 新增 AnimalRoar / AnimalRoar_Off 指令。	217
7.6	2023/5/31	1. 新增 **[Memory]**。	69
		2. 新增 **[LED Strip]**。	73
		3. 新增 Embedded Flash 指令。	195
		4. 新增 NY5 與 NY5+音量階數對映。	349
		5. I2C 新增支援 NX1。	49, 241
7.7	2023/8/31	1. 修改 SPI 相關指令範例。	
		2. 新增 RT_Vol 指令。	
		3. 更新 Oscillator。	34, 35
		4. 更新 Voice Output。	38, 39
		5. 更新 Touch Key。	45
		6. 更新 Reset。	46
		7. 更新 Interrupt Service。	61
		8. 更新 RAM Usage。	63
		9. 更新 Realtime Play。	67
		10.新增 ISP。	70
		11.新增 Storage Modual。	70
		12.更新 QIO。	96
		13.更新 Action。	100

版本	日期	內 容 描 述	修正頁
7.8	2023/11/30	1. 更新 Information Window。 2. 更新 IR。 3. 更新 RFC。 4. 新增 ADC Input。 5. 新增 Sync-Play。 6. 新增 Scrolling-Text。 7. 更新 Direct Key。 8. 更新 Pull-High Resistor。 9. 更新 Sentence。 10.新增 OKON_SustainOn / OKON_SustainOff / OKON_SustainEnd 指令。 11.新增 IR_TX_WaitN / UART_TX_WaitN 指令。	30 49 69 77 89 90 95 102 126 234 261, 268
8.0	2024/08/30	1. NY5+支援 QFID。 2. PlayV / PlayM 新增 Storage 參數。 3. NX1 支援 Animaltalks 功能。 4. 新增 ReadFileCountV / ReadFileCountM 指令。 5. 新增 PCM_Loop_On / PCM_Loop_Off / ADPCM_UpSampling 指令。 6. 新增 GetMicVol 指令。 7. NX1 支援 Enforce_Calibrate_Normal 指令。 8. 更新 OKON_SustainOn / OKON_SustainOff / OKON_SustainEnd。 9. 更新 MIDI_Pitch。	- 189, 227 314 198, 239 197 332 255 237 238
8.1	2024/11/30	1. 新增 ADM_Loop_On / ADM_Loop_Off / ADM_UpSampling 指令。 2. 新增 Sleep 指令。	202 336
8.2	2025/02/28	1. NY5+ / NX1 支援 I/O Expander。 2. 新增 MIDI_Loop_On / MIDI_Loop_Off。 3. 新增 Millis / Printf。	118, 349 268 370

版本	日期	內 容 描 述	修正頁
8.3	2025/05/29	1. 更新簡介。	24
		2. 更新 Touch Key。	50
		3. 更新 IR。	54, 55
		4. 更新 Voice File。	85
		5. 新增更多的流程控制指令，包括 While / Do-While。	167,194
		6. Break 指令更名為 BreakFG。	208
		7. 修改 Delay 指令內容。	304
		8. Action 指令修改 NX1 支援通道範圍。	307
		9. 新增 Srand 指令。	362
8.4	2025/08/29	1. 更新功能選單。	26
		2. 更新段落。	30
		3. 更新 Voice Output。	42
		4. 更新 IR。	54
		5. 更新 RAM Usage。	76,
		6. 更新 Voice File。	87
		7. 新增 Animalsings 功能。	96, 345
		8. 新增 For 迴圈。	199
		9. SBC / ADPCM / ADM / PCM / MIDI loop 指令標示為過期。	224 ~ 226, 268
		10. 新增 IR_Carrier_On / IR_Carrier_Off 指令。	293
		11. 新增 Audio_Loop 功能	359