九齐科技股份有限公司

Nyquest Technology Co., Ltd.

用户手册

Q-Code

Power-Speech Format Programmer

Version 8.4

Aug. 29, 2025

目　录

1 简介

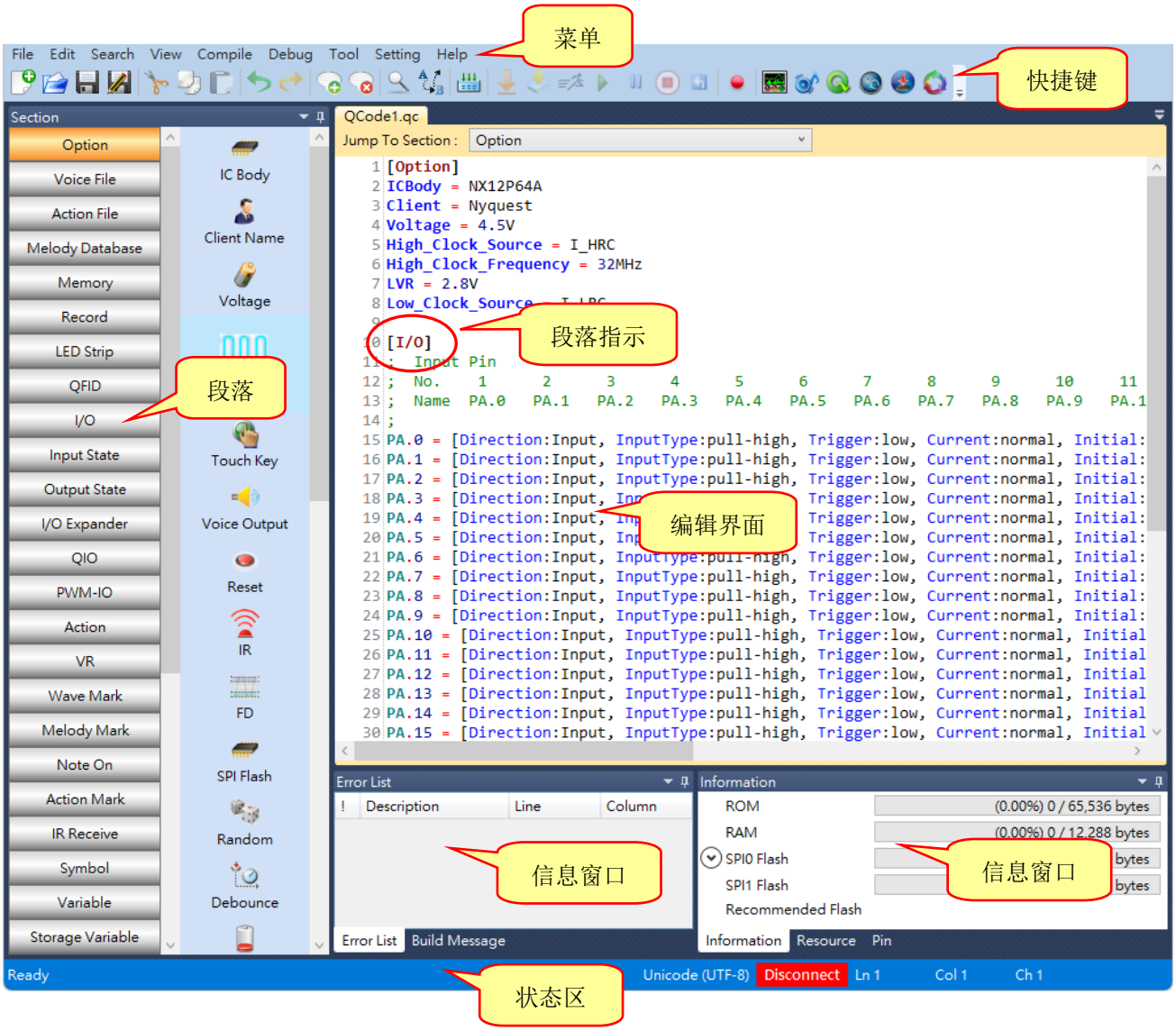
Q-Code 为九齐科技股份有限公司所提供，它是针对九齐科技的 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1 系列而开发的软件工具。它提供简易的图形处理界面来完成应用程序的开发。无需了解硬件结构和汇编语言的编写，工程师依然可以利用 *Q-Code* 强大的功能来完成应用程序的开发，简化了产品开发流程，提高了产品开发效率。对于初级工程师只需要进行简单的培训，在短时间内就可以完成某一产品的开发。

您可以联系九齐科技来获得 *Q-Code* 的安装程序文件，双击执行后进入安装程序向导，然后依照画面指示将可轻松完成安装流程。

系统需求：

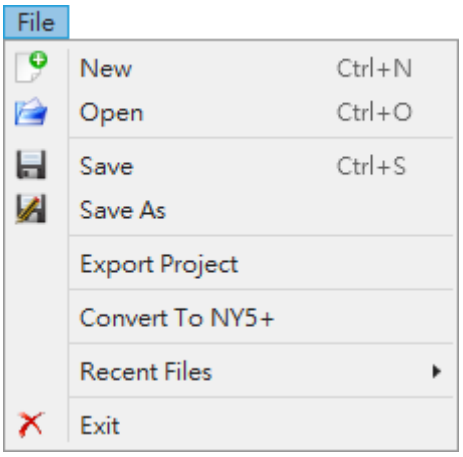
- Pentium 1.3GHz 或更高级处理器，Win7、Win8、Win10、Win11 操作系统。
- 至少 1GB SDRAM。
- 至少 2G 硬盘空间。
- 显示器和显示卡支持分辨率 1366x768 或更高。
- 需安装 .Net Framework 4.8。

2 Q-Code 界面外观


菜单
File Edit Search View Compile Debug Tool Setting Help
快捷键
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
段落
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]
段落指示
; Input Pin
; 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, Trigger:low, Current:normal, Initial:
PA.4 = [Direction:Input, Trigger:low, Current:normal, Initial:
PA.5 = [Direction:Input, 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
编辑界面
Error List
! Description Line Column
信息窗口
Error List Build Message
Information
ROM (0.00%) 0 / 65,536 bytes
RAM (0.00%) 0 / 12,288 bytes
SPI0 Flash bytes
SPI1 Flash bytes
Recommended Flash
信息窗口
Information Resource Pin
Ready
Unicode (UTF-8) Disconnect Ln 1 Col 1 Ch 1
状态区

2.1 *Q-Code* 功能菜单

2.1.1 文件（File）



新建（New）：新建一个.qc 文件。

打开（Open）：打开一个.qc 文件。

保存（Save）：保存一个.qc 文件。

另存为（Save As）：将一个.qc 文件另存为其它文件夹中。

导出（Export Project）：将.qc 以及其中所使用到的文件导出至另外的目录中。

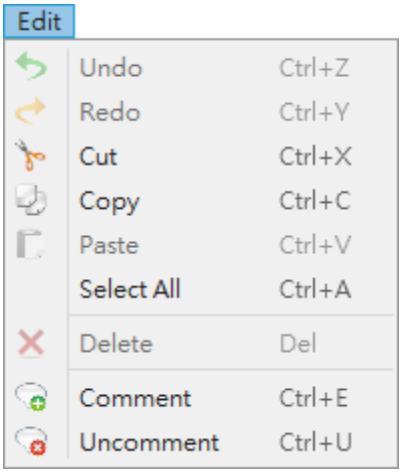
变换成 NY5+（Convert To NY5+）：将.qc 转换为 NY5+专案。详细说明请参考 6.13。

注意：仅 NY5 支持此功能。

打开最近的文件（Recent Files）：打开最近使用过的.qc 文件。当关闭一个文件后，该文件会自动添加到“Recent Files”目录中。

退出（Exit）：退出 *Q-Code*。

2.1.2 编辑（Edit）



撤销（Undo）：撤销最近一次编辑的动作。

恢复（Redo）：取消最近一次撤销的动作。

剪切（Cut）：剪切一选取区段文字内容。

复制（Copy）：复制一选取区段文字内容。

粘贴（Paste）：粘贴已剪切或复制的文字内容。

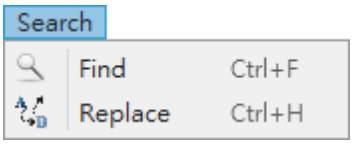
选取全部（Select All）：选取全部文字内容。

删除（Delete）：删除一选取区段文字内容。

批注（Comment Selection）：将光标所在位置或已选取多行进行批注。

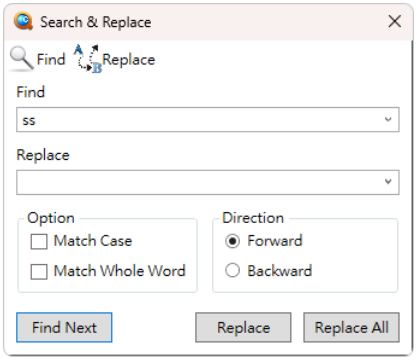
取消批注（Uncomment Selection）：将光标所在位置或已选取多行进行批注取消。

2.1.3 搜寻（Search）



查找（Find）：在对话框内输入欲寻找的字符串或数字。

替换（Replace）：把目前相符合的字符串替换。



符合大小写（Match Case）：在查找时区分大小写。

符合单字（Match Whole Word）：只查找单字。（当未选取这个选项时，可能会查找到更长的单字）

向下搜寻（Forward）：向下查找符合的字符串。

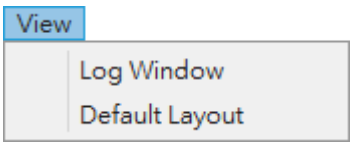
向上搜寻（Backward）：向上查找符合的字符串。

继续查找（Find Next）：查找下一个。

替换（Replace）：把目前相符合的字符串替换。

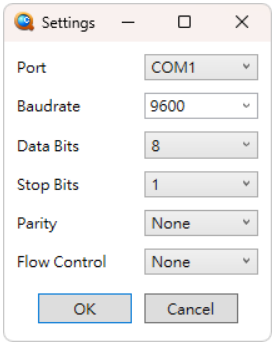
全部替换（Replace All）：点击该项可以把“相符合的字符串”全部替换。

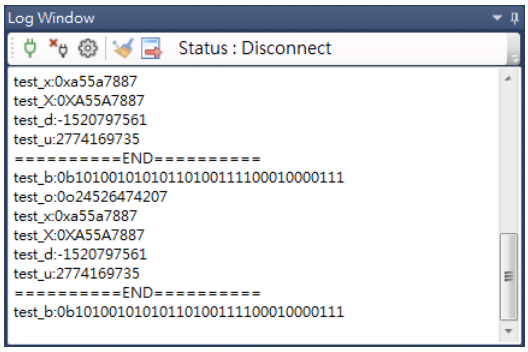
2.1.4 检视（View）



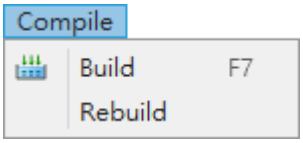
默认布局（Default Layout）:：恢复默认画面。

纪录窗口（Log Window）：接收由 UART 送出的信息。使用设定画面（如下图）设定串行端口（Port）/ 鲍率（Baudrate）/ 同位检察（Parity）等相关设定后，点击连接按钮即可开始进行数据接收。





2.1.5 编译（Compile）

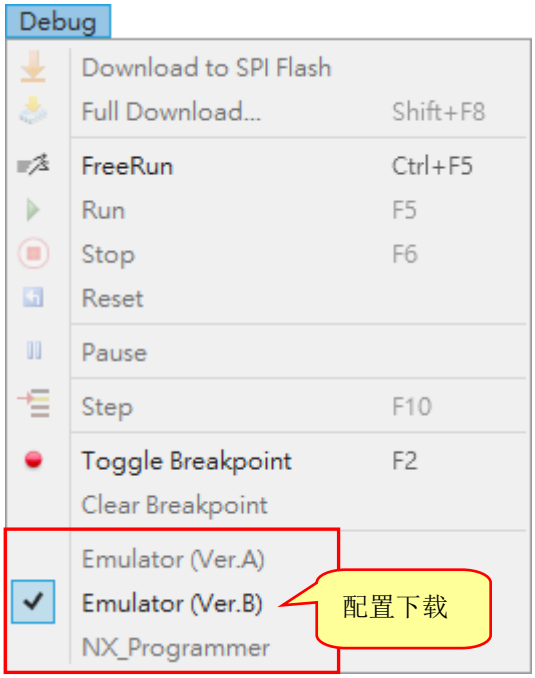


构建（Build）：将当前.qc 档编译成.bin 档。以便下载到 ICE 或是 Flash Demo Board 验证用。如果音源文件被编译过且未改变，就不会重新编译，以加速编译。编译完成后，Build 结果会显示在 Build Message 窗口，ROM 的使用情况会显示在信息窗口（Information）。

重新构建（Rebuild）：不论音源文件是否被编译过，其已编译数据都会被清除并重新编译成.bin 档。

2.1.6 调适（Debug）

此功能主要是当用户在 *Q-Code* 的程序中可能需要用 ICE 来进行除错。当使用此功能时，需先将 ICE 或 NX_Programmer 连接到计算机的 USB 端口（USB Port）。



下载到 SPI Flash（Download To SPI Flash）：下载程序到仿真器的 SPI Flash。

注意：仅 NY6 支持此功能。

完整下载（Full Download）：下载完整的程序和数据到仿真器和 SPI Flash。计算机需要连接 NX_Programmer。

注意：仅 NX1 支持此功能。

执行但不调适（Free Run）：ICE 执行程序，不可中断进行侦错。

执行（Run）：ICE 执行程序，可中断进行侦错。

注意：仅 NX1 支持此功能。

停止（Stop）：停止 ICE。

重置（Reset）：不关电源进行重置。

暂停（Pause）：暂停 ICE。

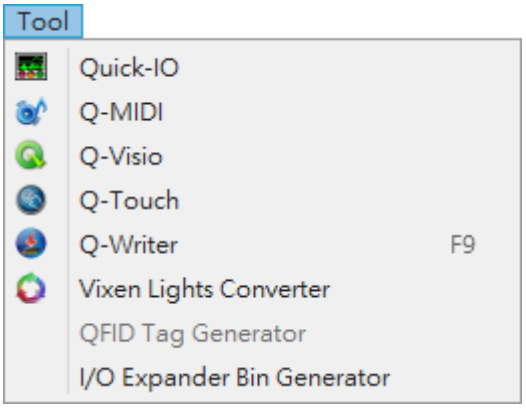
单步执行（Step）：逐行执行 ICE。

开关断点（Toggle Breakpoint）：断点开启或关闭。

清除所有断点（Clear Breakpoint）：清除所有断点。

配置下载：可通过此配置设定来选择欲下载试听的演示板。目前支持 Emulator Ver.A、Emulator Ver.B 和 NX_Programmer。在设定后每一次的下载都会依照此设定执行。

2.1.7 工具（Tool）



Quick-IO：此软件是用来画出输出脚的信号，需结合.wav 文件使用，所产生的文件为.nyq。

Q-MIDI：此软件是用来编辑 MIDI 与音色库，产生的文件为.qmd，可在 NY5 / NY5+ / NY6 / NY7 / NX1 使用。

Q-Visio：此软件是用来画出输出脚的信号，所产生的文件为.vio。

Q-Touch：此软件是用来扫描触摸键的灵敏度，所产生的文件为.t9x 和.tnx。

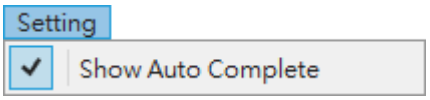
Q-Writer：此软件是用来将.bin 文件烧录在 Flash Demo Board、Romter 或 OTP 上以供验证。

Vixen Lights Converter：此软件是用来将.csv 文件转换成多个.csv 和.vio 文件。

QFID Tag Generator：此处用来产生 QFID 应用中，Tag 所使用的.bin 文件，仅可在 NY7 / NX1 使用。

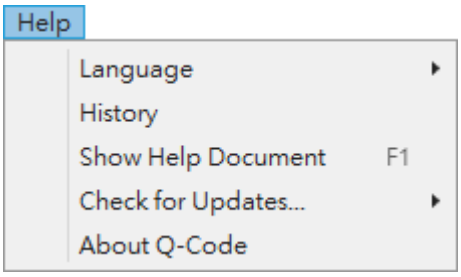
I/O Expander Bin Generator：此工具用来产生 I/O Expander 应用中 I/O Expander 所使用的 bin 檔。

2.1.8 设定（Setting）



自动完成（Show Auto Complete）：搜寻建议功能的开关。

2.1.9 帮助（Help）



语言（Language）：切换 *Q-Code* 显示语言。

改版信息（History）：查看 *Q-Code* 最新改版信息。

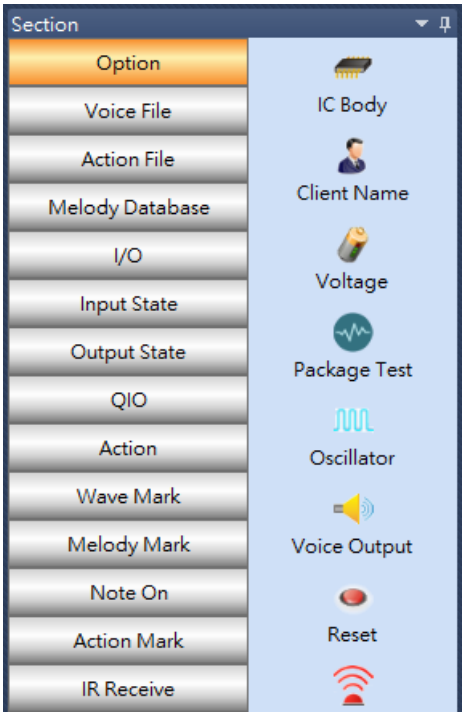
参考手册（Show Help Document）：显示 Q-Code Reference Manual。

检查更新（Check for Updates…）：检查是否有最新的 *Q-Code*、*NYASM*、*NYC_NX1* 版本，此功能需连上网络。

关于 *Q-Code*（About *Q-Code*）：显示目前所安装的 *Q-Code* 版本，以及技术支持的相关联系方式。

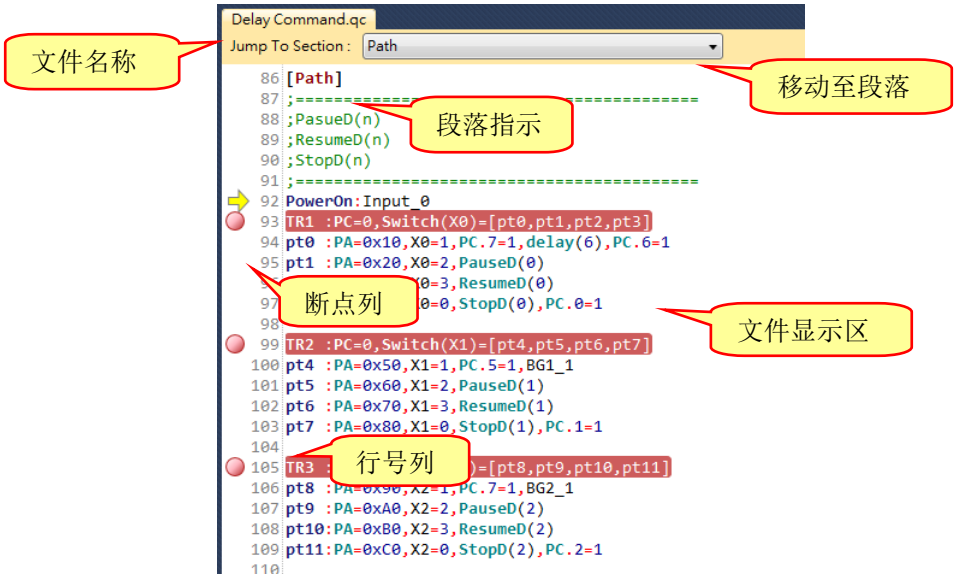
2.2 段落（Section）

段落提供多项简易的用户界面，让用户更容易完成 *Q-Code* 程序的开发。详细操作方式请见段落描述。



2.3 编辑界面（Text Edit Area）

用户编辑 *Q-Code* 程序的区块，与一般编辑软件的操作类似，编辑界面的左方有行号的显示，*Q-Code* 程序相关的指令及关键词也会进行变色处理，除了菜单及快捷键的编辑功能外，用户可以利用鼠标右键菜单使用各项编辑功能，也可以利用段落来编辑 *Q-Code* 程序，关于 *Q-Code* 程序开发的详细信息请参阅 *Q-Code* 架构。



文件名称：显示文件名称。

断点列：程序运行 / 断点等图标。

行号列：显示行号。

移动至段落（Jump To Section）：将光标移动至选择的段落。

文件显示区：文件内容。

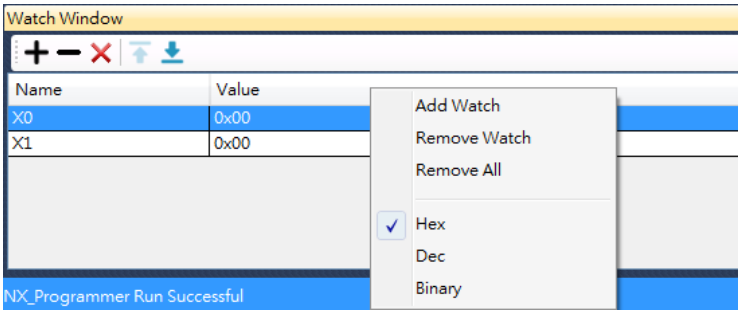
段落指示：显示目前的段落。

2.4 侦错模式（Debug Mode）

此功能仅支持 NX1。按下执行（Run）进入侦错模式，就会出现两个窗口：监视窗口（Watch Window）和寄存器窗口（Register Window）。

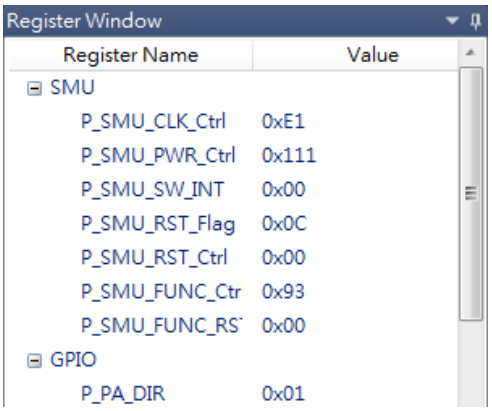
2.4.1 监视窗口（Watch Window）

用户可以自行加入欲观察的寄存器或是变量，方便追踪。



2.4.2 寄存器窗口（Register Window）

侦错过程中，显示寄存器的数值，只有在 Pause 状态下才会更新数值。

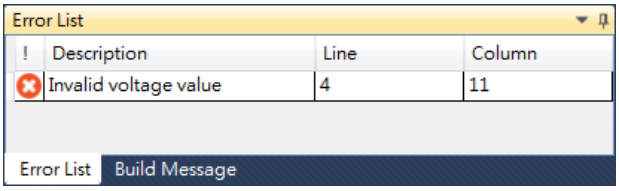


2.5 信息窗口（Message Window）

分为两个部分：错误列表（Error List）和建置信息（Build Message）。

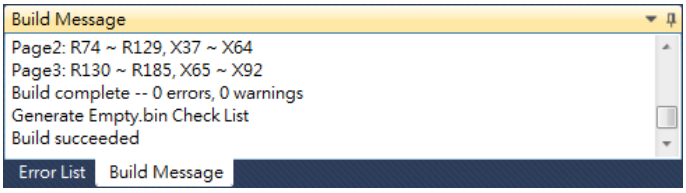
2.5.1 错误清单（Error List）

错误清单（Error List）负责显示所有编辑界面要传达给用户的信息，包括语法上的错误、错误信息等等。用户可以利用鼠标左键点击两次错误的信息，编辑界面就将可能产生错误的位置反白。



2.5.2 构建信息（Build Message）

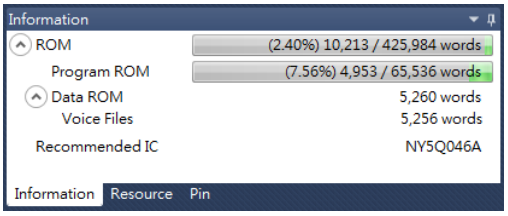
构建信息（Build Message）负责所有编译过程及下载到 ICE 过程需传达给用户的信息。



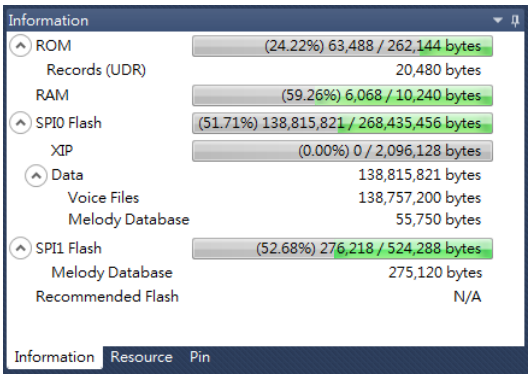
2.6 信息窗口（Information Window）

2.6.1 存储信息（Information）

编译完成后会将内存的使用状况显示于此窗口，若有超过内存容量的部分，会以红色字体显示。Recommended IC 栏位是依据程序使用的 ROM 大小，在同系列母体中提供选择参考。

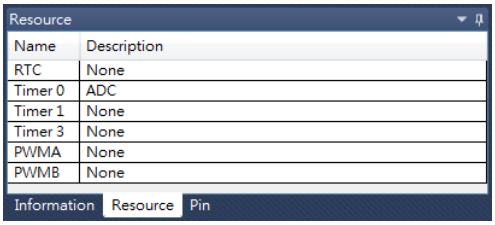


若为 NX1 系列母体，则显示会改为 Recommended Flash 栏位。此栏位依据 SPI_Encoder 项目中的设置或是使用的 SPI Flash 容量，显示适合的 N25Q 系列母体供选择参考。



2.6.2 资源信息（Resource）

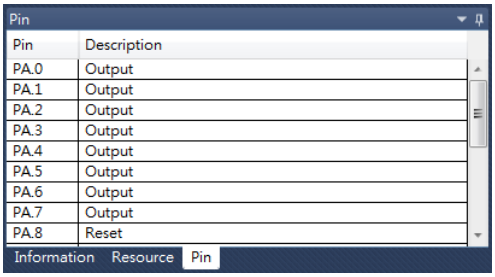
NX1 系列母体，会显示 Timer 的使用信息。



注意：RTC 以外的 Timer 分配，均采用独占处理，如有冲突请重新安排相关应用程序。

2.6.3 脚位信息（Pin）

会将目前脚位的状况显示于此窗口。

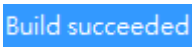


2.7 *Q-Code* 状态区（*Q-Code* Status Bar）

显示 *Q-Code* 目前的各项状态。包含 Build 结果（或 ICE 状态）、ICE 图示状态、编辑行列、字数、INS 等等信息。

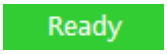
2.7.1 编译结果与 ICE 状态（Build Result and ICE Status）

下图所示为.qc 文件编译后 OK or Fail：



2.7.2 ICE 连接状态（ICE Connect Status）

下图所示为 ICE 已连接：



下图所示为 ICE 未连接：

2.7.3 行列字符显示（Row, Column, Characters）

显示光标目前所在的列、行及字符所在位置。

3 *Q-Code* 架构

Q-Code 由不同的段落组成，各个段落代表不同的功能描述，某些段落仅在某些系列支持。下表为所有段落的列表以及各系列的支持程度。

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

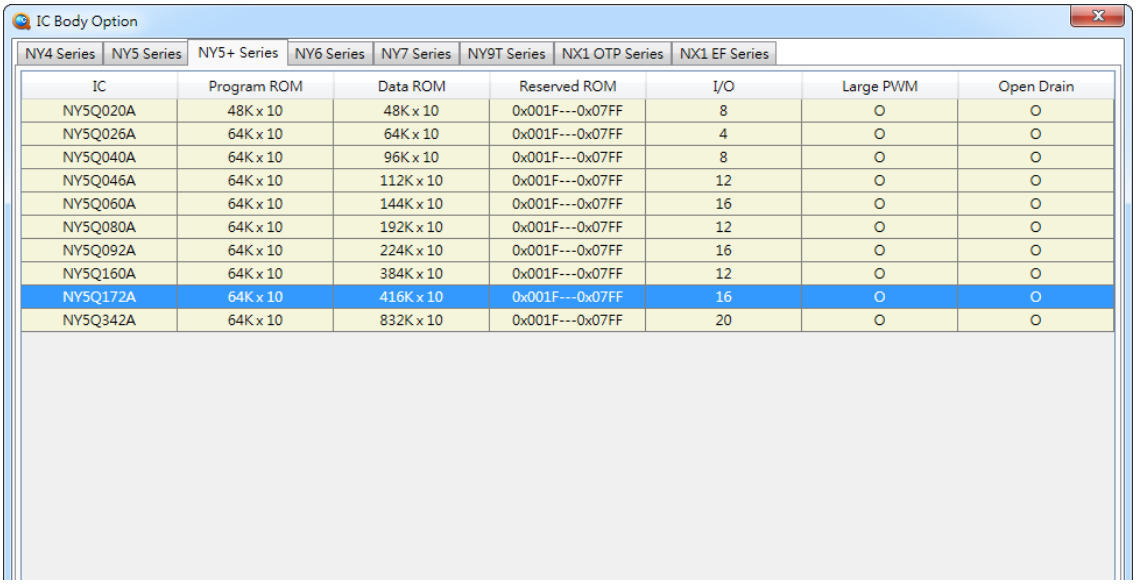
段落	NY4	NY5	NY5+	NY6	NY7	NY9T	NX1
C-Code	X	X	X	X	X	X	O
Macro	O	O	O	O	O	**O**	O
Sentence	O	O	O	O	O	**O**	O
Subroutine	O	O	O	O	O	**O**	O
QIO Custom	O	O	O	X	X	X	X
Path	O	O	O	O	O	O	O
Background1	O	O	O	O	O	O	O
Background2	O	O	O	O	O	O	O
Background3	X	X	X	X	X	X	O
Interrupt	X	X	X	X	X	X	O

3.1 Option

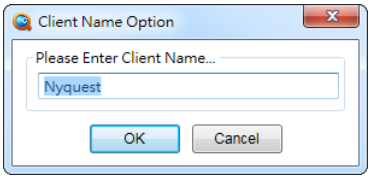
3.1.1 IC Body

选择 IC 的型号。

例. **ICBody** = NY5Q172A



3.1.2 Client

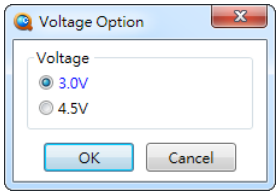


用户或是公司名称。必须填入客户名称，否则编译将不能通过。(此举是用来保护程序开发者的所有权益)

例. **Client** = Nyquest

3.1.3 Voltage

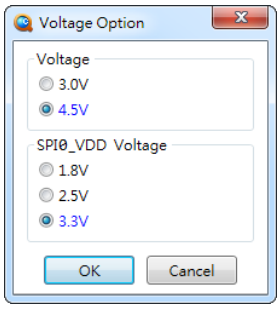
3.1.3.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1EF



Voltage：选择 IC 实际应用的工作电压。（IC 的工作频率在 3.0V 和 4.5V 会有点不同）

例. **Voltage** = 3.0V

3.1.3.2 NX1 OTP



Voltage：选择 IC 实际应用的工作电压。（IC 的工作频率在 3.0V 和 4.5V 会有点不同）

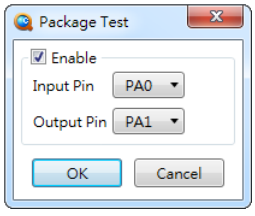
SPI0_VDD Voltage：选择通过 SPI0 接脚（PB.0 ~ PB.3）连接的外部 SPI flash 工作电压，SPI Flash 电源将由内部 LDO 提供。

例. **Voltage** = 4.5V

SPI0_VDD_Voltage = 4.5V

3.1.4 Package Test

3.1.4.1 NY4 / NY5 / NY7



当用户需要代客封装后的 IC 做基本功能检测时，可打开此功能，让封装后的 IC 能进行 Open / Short、Standby Current、OSC 频飘和 Checksum 的验证。

注意：

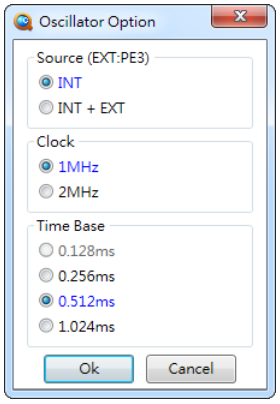
1. ***打开此功能后，IC 的上电初始化时间会增加额外的 80ms。***
2. ***打开此功能后，测试程序会占用一小部分的 Program ROM Size。***
3. ***只有使用九齐公版封装且有打开封装测试功能并用默认脚位出 Pin 的代客封装需求，才被视为标准流程，出货前才会进行封装测试。***

4. 由于 NY5 的脚位组态选定后就不能再变更，因此输入与输出脚位组态必需符合下表：

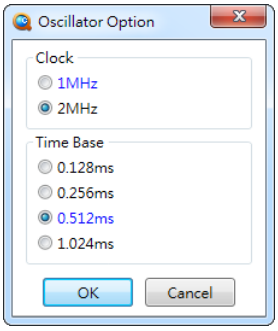
封装测试脚位	组态
指令输入脚位	Direct Input
	Direct Input IO
	Matrix Input
	Matrix Output
	Serial Control (SPI_Like) Clock
	Serial Control (SPI_Like) Data
	Serial Control (IR_Trigger) Data
	IR RX
信号输出脚位	Direct Input IO
	Direct Output
	Matrix Output

3.1.5 Oscillator

3.1.5.1 NY5



NY5



NY5PxxxB / NY5PxxxJ

Source：选择 INT 表示 OSC 使用内部振荡电阻，INT+EXT 则表示 OSC 既可使用内部电阻也可使用外部电阻。（默认 INT）。

注意：只有 NY5 支持 EXT 选项。

例. **Oscillator** = INT + EXT

Clock：设定系统频率。支持 1MHz / 2MHz（默认 1MHz）。

例. **Clock** = 1MHz

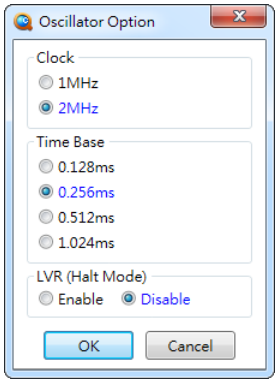
Time Base：*Q-Code* 运作的基础计时时间，所有的时间计数皆参照此定时器。用户可以自行设定所需的定时器。支持 0.128ms / 0.256ms / 0.512ms / 1.024ms（默认 0.512ms）。

例. **TimeBase** = 0.512ms

注意：

1. *NY5 因为性能问题，所以在 Clock=2MHz 下，才能使用 0.128ms。*
2. *定时器越快，则越占 CPU 性能。*
3. *定时器与 Action 定时器共享，所以改变定时器的基准时间会连带改变 Action 的定时基准。*
4. *Time Base 设定不会影响 Melody 的定时基准时间。*

3.1.5.2 NY5+



Clock：设定系统频率。支持 1MHz / 2MHz（默认 2MHz）。

例. **Clock** = 2MHz

Time Base：*Q-Code* 运作的基础计时时间，所有的时间计数皆参照此定时器。用户可以自行设定所需的定时器。支持 0.128ms / 0.256ms / 0.512ms / 1.024ms（默认 0.256ms）。

例. **TimeBase** = 0.512ms

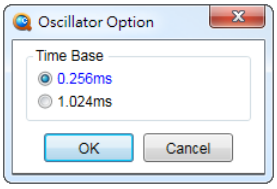
LVR (Halt Mode)：进 Halt Mode 时，低电压复位功能的开关选项。

例. **LVR_When_Halt** = Enable

注意：

1. *定时器越快，则越占 CPU 性能。*
2. *定时器与 Action 定时器共享，所以改变定时器的基准时间会连带改变 Action 的定时基准。*
3. *Time Base 设定不会影响 Melody 的定时基准时间。*
4. *LVR On 在 Halt Mode 将增加 Standby Current。*

3.1.5.3 NY6



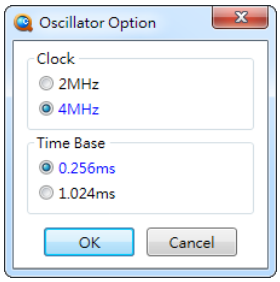
Time Base：*Q-Code* 运作的基础计时时间，所有的时间计数皆参照此定时器。用户可以自行设定所需的定时器。支援 0.256ms / 1.024ms（预设 0.256ms）。

例. **TimeBase** = 0.256ms

注意：

1. ***定时器越快，则越占 CPU 性能。***
2. ***定时器与 Action 定时器共享，所以改变定时器的基准时间会连带改变 Action 的定时基准。***
3. ***Time Base 设定不会影响 Melody 的定时基准时间。***

3.1.5.4 NY7



Clock：设定系统频率。支援 2MHz / 4MHz（预设 4MHz）。

例. **Clock** = 4MHz

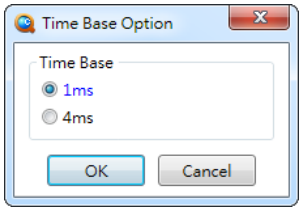
Time Base：*Q-Code* 运作的基础计时时间，所有的时间计数皆参照此定时器。用户可以自行设定所需的定时器。支援 0.128ms / 0.256ms / 1.024ms（预设 0.256ms）。

例. **TimeBase** = 0.256ms

注意：

1. ***定时器越快，则越占 CPU 效能。***
2. ***定时器与 Action 定时器共享，所以改变定时器的基准时间会连带改变 Action 的定时基准。***
3. ***Time Base 设定不会影响 Melody 的定时的基准时间。***

3.1.5.5 NY9T



Time Base：*Q-Code* 运作的基础计时时间，所有的时间计数皆参照此定时器。用户可以自行设定所需的定时器。支援 1ms / 4ms（预设 1ms）。

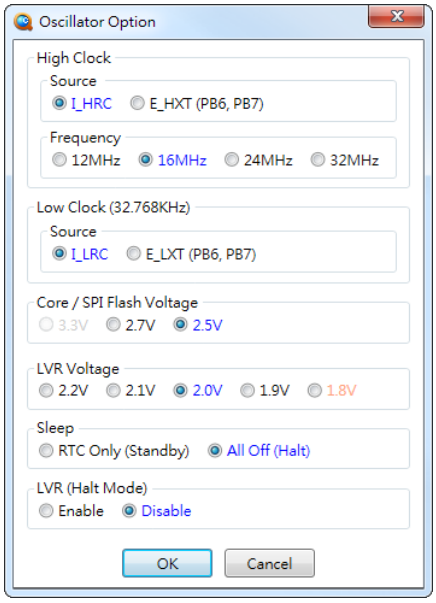
例. **TimeBase** = 1ms

注意：

1. ***定时器越快，则越占 CPU 效能。***
2. ***定时器与 Action 时间计数器共享，所以改变定时器的基准时间会连带改变 Action 的定时基准。***
3. ***Time Base 设定不会影响 Melody 的定时的基准时间。***

3.1.5.6 NX1 OTP

NX1 震荡器选项分为二组 High_Clock 和 Low_Clock。High_Clock 及 Low_Clock 皆可使用外部振荡器。然而 IC 上仅有一组外部震荡连接脚位，仅有 High_Clock 或 Low_Clock 其中之一可使用外部振荡器。



High_Clock_Source：正常模式使用的频率来源，可选择内部（I_HRC）或外部（E_HXT）。

例. **High_Clock_Source** = I_HRC

High_Clock_Frequency：正常模式系统运作频率。

例. **High_Clock_Frequency** = 24MHz

Low_Clock_Source：低速模式使用的频率来源，可选择内部（I_LRC）或外部（E_LXT）。频率固定为 32.768KHz。

例. **Low_Clock_Source** = I_LRC

Core / SPI Flash Voltage：核心暨外部 SPI Flash 工作电压，电源由 NX1 内部 LDO 提供。

注意：NX11P21A / NX11S / NX11M / NX12M / NX13M 支持 Core / SPI Flash Voltage 选项。

例. **SPI_Flash_Voltage** = 2.8V

LVR Voltage：低电压复位阀值，High Clock Frequency 的选择会影响可用的阀值。

例. **LVR** = 2.4V

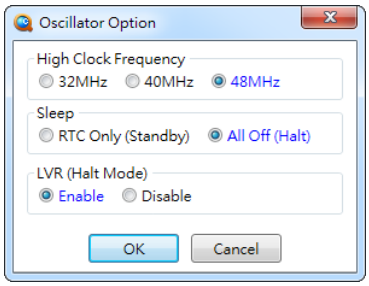
Sleep：IC 进入睡眠时，定时器运作的模式。设定为 RTC Only 时，会保留 RTC_2Hz 和 RTC_64Hz 运作。若是设定为 All Off 则所有的定时器都会停止，在此状态下，只有 IO 中断会唤醒 IC。

例. **Sleep** = All_Off

LVR (Halt Mode)：进 Halt Mode 时，低电压复位功能的开关选项。Sleep 必须选择 All Off 才能设定。

例. **LVR_When_Halt** = Enable

3.1.5.7 NX1 EF



High_Clock_Frequency：正常模式系统运作频率。

例. **High_Clock_Frequency** = 48MHz

Sleep：IC 进入睡眠时，定时器运作的模式。设定为 RTC Only 时，会保留 RTC_2Hz 和 RTC_64Hz 运作。若是设定为 All Off 则所有的定时器都会停止，在此状态下，只有 IO 中断会唤醒 IC。

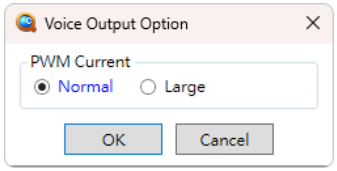
例. **Sleep** = All_Off

LVR (Halt Mode)：进 Halt Mode 时，低电压复位功能的开关选项。Sleep 必须选择 All Off 才能设定。

例. **LVR_When_Halt** = Enable

3.1.6 Voice Output

3.1.6.1 NY4

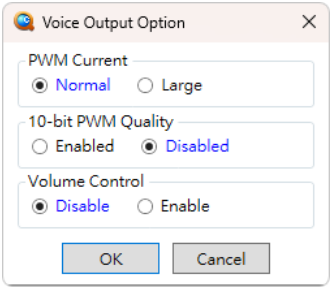


PWM Current：PWM 输出电流。支持 Normal / Large。

注意：NY4A 不支持 PWM Current 选项。

例. **PWM Current** = Normal

3.1.6.2 NY4PxxxC



PWM Current：PWM 输出电流。支持 Normal / Large。

例. **PWM Current** = **Normal**

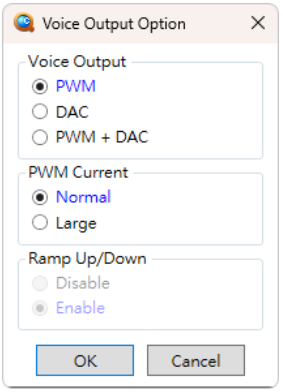
10-bit PWM Quality：设置 10-bit PWM 是否打开。

例. **PWM_10Bit** = **Enable**

Volume Control：设置音量控制是否打开。

例. **Volume_Control** = **Enable**

3.1.6.3 NY5



Voice Output：设定喇叭的驱动型态。PWM 输出可直接接上喇叭输出，DAC 输出需要另外加上晶体管及偏压电阻。PWM+DAC 为自动侦测外部组件是使用 PWM 或者是 DAC 输出。

例. **Voice Output** = PWM

PWM Current：PWM 输出电流。支持 Normal / Large。

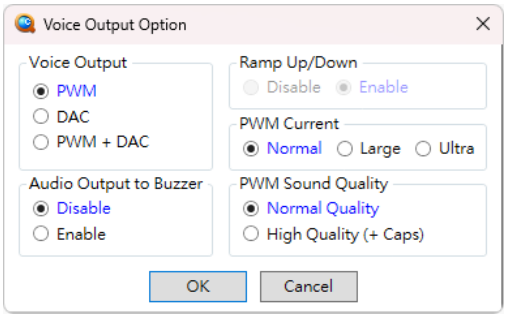
例. **PWM Current** = Normal

Ramp Up/Down：只在 DAC 模式下，可允许关闭自动 Ramp Up/Down 功能。

例. **RampUp/Down** = Disable

注意：并不是所有的应用都需要关闭 RampUp / RampDown 功能，该功能针对外接 OP Amp 输出时，要消除播音瞬间，因为 RampUp / RampDown 的动作所造成的"bo"声。

3.1.6.4 NY5+



Voice Output：设定喇叭的驱动型态。PWM 输出可直接接上喇叭输出，DAC 输出需要加上放大电路。PWM+DAC 为自动侦测外部组件是使用 PWM 或者是 DAC 输出。

例. **Voice Output** = PWM

PWM Current：PWM 输出电流。支持 Normal / Large / Ultra。

例. **PWM Current** = Normal

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

例. **PWM_Sound_Quality** = Normal

RampUp/Down：只在 DAC 模式下，可允许关闭自动 Ramp Up/Down 功能。

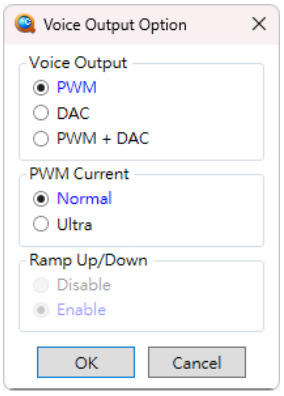
例. **RampUp/Down** = Disable

注意：并不是所有的应用都需要关闭 RampUp / RampDown 功能，该功能针对外接 OP Amp 输出时，要消除播音瞬间，因为 RampUp / RampDown 的动作所造成的"bo"声。

Audio Output to Buzzer：声音由蜂鸣片输出时，请启用此选项。

例. **Audio_Output_Buzzer** = Enable

3.1.6.5 NY6



Voice Output：设定喇叭的驱动型态。PWM 输出可直接接上喇叭输出，DAC 输出需要加上放大电路。PWM+DAC 为自动侦测外部组件是使用 PWM 或者是 DAC 输出。NY6A 系列仅提供 PWM，NY6B / NY6C 系列提供 PWM / DAC / PWM+DAC 选项。

注意：NY6 的 Voice Output 仅设定初始输出方式，若用户往后需要变更时，可用 AudioMode 指令来变更输出方式。

例. **Voice Output** = PWM

PWM Current：PWM 输出电流。NY6 系列支持 Normal / Ultra。

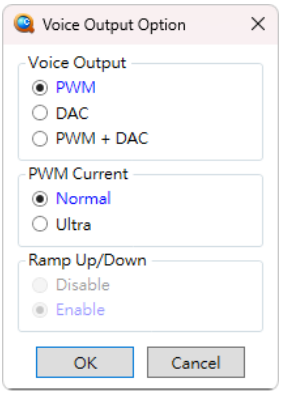
例. **PWM Current** = Ultra

Ramp Up/Down：只在 DAC 模式下，可允许关闭自动 Ramp Up/Down 功能。

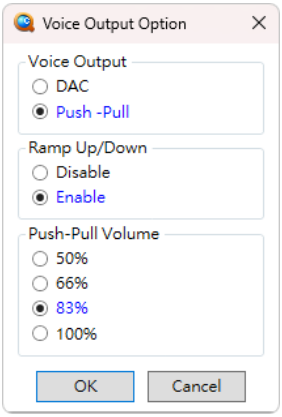
例. **RampUp/Down** = Disable

注意：并不是所有的应用都需要关闭 RampUp / RampDown 功能，该功能针对外接 OP Amp 输出时，要消除播音瞬间，因为 RampUp / RampDown 的动作所造成的"bo"声。

3.1.6.6 NY7



NY7A



NY7B / NY7C

Voice Output：设定喇叭的驱动型态。PWM 输出可直接接上喇叭输出，DAC 输出需要加上放大电路。PWM+DAC 为自动侦测外部组件是使用 PWM 或者是 DAC。Push-Pull 则是模拟推挽式输出。NY7A 系列提供 PWM / DAC / PWM+DAC 选项（默认值 PWM）。NY7B / NY7C 提供 DAC 与 Push-Pull 选项。同时间只能有一种输出方式（默认值 Push-Pull）。

注意：NY7 的 Voice Output 仅设定初始输出方式，若用户往后需要变更时，可用 AudioMode 指令来变更输出方式。

例. **Voice Output** = PWM

例. **Voice Output** = Push-Pull

PWM Current：PWM 输出电流。NY7A 系列支持 Normal / Ultra。

例. **PWM Current** = Ultra

注意：NY7B / NY7C 不支持 PWM Current 设置。

Ramp Up/Down：只在 DAC 模式下，可允许关闭自动 Ramp Up/Down 功能。

例. **RampUp/Down** = Disable

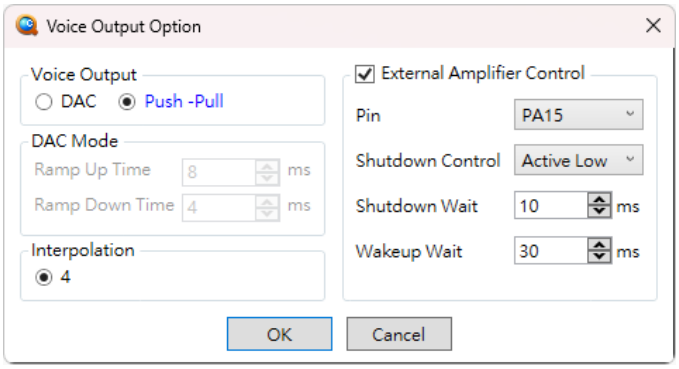
注意：并不是所有的应用都需要关闭 RampUp / RampDown 功能，该功能针对外接 OP Amp 输出时，要消除播音瞬间，因为 RampUp / RampDown 的动作所造成的"bo"声。

Push-Pull Volume：调整 Push-Pull 功放的 Analog 输出音量设定。提供 100%、83%、66%、50% 选项（默认值 83 %）。

注意：NY7A 不支持 Push-Pull Volume 设置。

例. **Gain** = 83%

3.1.6.7 NX1 OTP



Voice Output：设定喇叭的驱动型态。DAC 输出需要加上放大电路，Push-Pull 则是模拟推挽式输出。支持 DAC 与 Push-Pull 选项。

例. **Voice Output** = Push-Pull

例. **Voice Output** = DAC

Ramp Up Time：设定 Ramp Up 的时间。会在设定的时间内完成 Ramp Up 功能。

例. **RampUp_Time** = 8ms

Ramp Down Time：设定 Ramp Down 的时间。会在设定的时间内完成 Ramp Down 功能。

例. **RampDown_Time** = 4ms

Interpolation：设定（倍频）插点。NX1 OTP 固定为 4。

例. **Interpolation**= 4

External Amplifier Control：外部放大器控制功能。

例. **ExtAmp** = **Enable**

Pin：外部放大器控制脚位。

例. **ExtAmp_Pin** = **PA**.15

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

例. **ExtAmp_Shutdown_Control** = Active_Low

例. **ExtAmp_Shutdown_Control** = Active_High

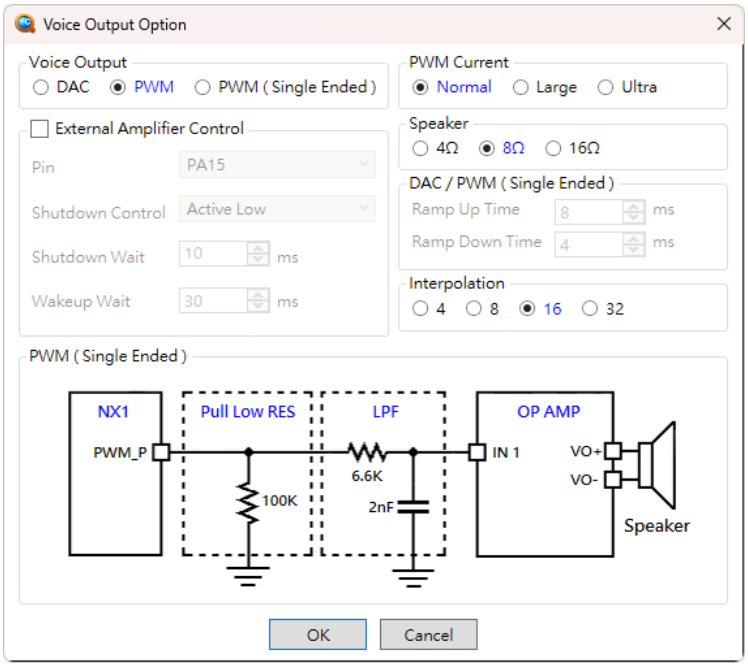
Shutdown Wait：关闭外部放大器等待时间。

例. **ExtAmp_Shutdown_Time** = 10ms

Wakeup Wait：唤醒外部放大器等待时间。

例. **ExtAmp_Wakeup_Time** = 30ms

3.1.6.8 NX1 EF



Voice Output：为设定喇叭的驱动型态。PWM 输出可直接接上喇叭输出，DAC 输出需要加上放大电路，PWM (Single Ended) 输出需要加上放大电路及另外的电阻及电容。

例. **Voice Output** = PWM

例. **Voice Output** = DAC

例. **Voice Output** = PWM_SE

Ramp Up Time：设定 Ramp Up 的时间。会在设定的时间内完成 Ramp Up 功能。

例. **RampUp_Time** = 8ms

Ramp Down Time：设定 Ramp Down 的时间。会在设定的时间内完成 Ramp Down 功能。

例. **RampDown_Time** = 4ms

Interpolation：设定（倍频）插点。分为 4、8、16 与 32 四种（倍频）插点供选择，数值越大插点越多，反之数值越小插点越少，默认值 16。

例. **Interpolation**= 16

External Amplifier Control：外部放大器控制功能。

例. **ExtAmp** = **Enable**

Pin：外部放大器控制脚位。

例. **ExtAmp_Pin** = **PA**.15

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

例. **ExtAmp_Shutdown_Control** = Active_Low

Shutdown Wait：关闭外部放大器等待时间。

例. **ExtAmp_Shutdown_Time** = 10ms

Wakeup Wait：唤醒外部放大器等待时间。

例. **ExtAmp_Wakeup_Time** = 30ms

PWM Current：PWM 输出电流。支援 Normal / Large / Ultra。

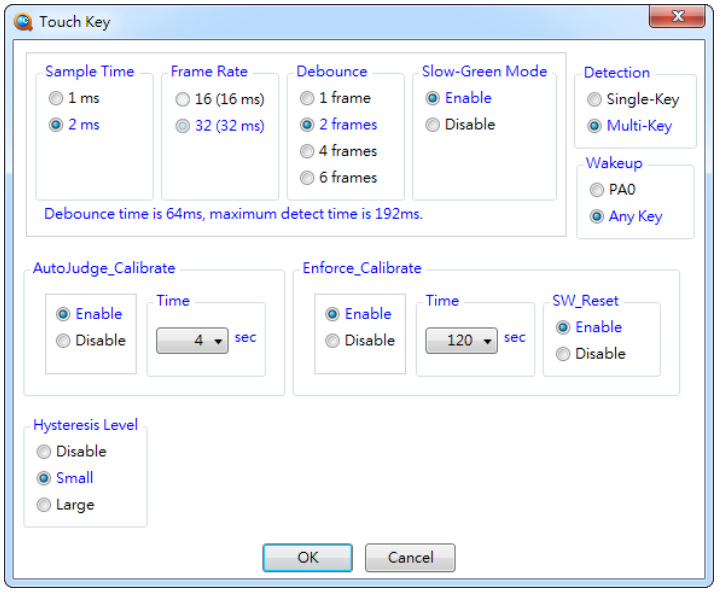
例. **PWM Current** = Large

Speaker：设定 Speaker 的阻抗值。

例. **Speaker** = 8ohm

3.1.7 Touch Key

3.1.7.1 NY9T



Sample Time：扫描一个触摸键的时间。较长的取样时间触摸更精确，但侦测到 TouchKey 被触摸的时间会较久。（默认 2ms）

Frame Rate：在一个 Frame 要取样的数量，即完成所有触摸键的扫描时间。取样的数量愈多则较不耗电，但侦测到 TouchKey 被触摸的时间会较久。8/16 samples（Sample Time 1ms），16/32 samples（Sample Time 2ms）。

Debounce：连续几个 Frame 被侦测到，按键才会成立。愈多 Frame 触摸更精确，但侦测到 TouchKey 被触摸的时间会愈久，对于较容易有外部干扰的环境建议选择多些 Frame。

Slow-Green Mode：只有在 TouchKey 设定使用超过 8 个键时（含 8 个），才能打开或关闭此项功能。若用户有下达“TouchKey_Scan_Slow”指令：则 Slow-Green Mode 打开时表示在进睡眠后会维持 Slow-Green Mode，反之关闭 Slow-Green Mode 时表示进睡眠后会维持 Slow Mode。Slow-Green Mode 比 Slow Mode 更省电，但侦测到 TouchKey 被触摸的时间会较久。（默认 Enable）

TouchKey_DebounceTime 为一般模式下侦测到 TouchKey 被触摸的时间；而 TouchKey_DetectTime 为进入睡眠后的最长侦测时间，实际上可能少于最长侦测时间。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. ***选用 NY9T016A 时，Frame Rate 固定为 16 samples。***
2. ***NY9T001A/NY9T004A 无 Slow-Green Mode 设定。***
3. ***Slow-Green Mode 只有在进入睡眠后，且切换到 TouchKey_Scan_Slow 才有效。***
4. ***在正常工作模式下，都维持 Normal Mode Scan，不管有没有下达"TouchKey_Scan_Slow"指令。***

例.

```
TouchKey_DebounceTime=64ms{SampleTime=2ms,FrameRate=32ms,Debounce=2,
Slow-Green_Mode=Enable}                    ; 一般模式侦测触摸的扫描时间为 64ms。
TouchKey_DetectTime = 192ms                ; 睡眠模式侦测触摸的最长扫描时间为 192ms。
```

注意：

1. ***Sample Time、Frame Rate、Debounce、Slow-Green Mode 和 Scan Count(Q-Touch Option) 皆会影响触摸键的反应时间及耗电流。***
2. ***触摸键的 Detect Time 时间，会因为 Q-Touch 选项中 Scan Count 的设置而使得扫描时间变为 1~4 倍，按键反应会比较慢，但按键准确率会提高。***

Detection：用户可以选择触摸键的侦测方式，共有 2 种侦测方式。Single-Key 为同时间只能接受单一触摸键被按下，Multi-Key 为可同时间多个触摸键被按下。（默认值 Multi-Key）

```
例. TouchKey_Detection = Multi-Key          ; 可同时接受多个按键被触摸。
例. TouchKey_Detection = Single-Key         ; 可同时接受一个按键被触摸。
```

Wakeup：用户可以选择只用 PA0 来唤醒或是全部的触摸键皆可唤醒。（默认值 AnyKey）

```
例. TouchKey_Wakeup = PA0                   ; 触摸键只有 PA0 可以唤醒 IC。
例. TouchKey_Wakeup = AnyKey                ; 所有的触摸键皆可唤醒 IC。
```

注意：

1. ***NY9T001A 无此设定。***
2. ***使用单键（PA0）唤醒可以获得更佳的省电效能。***
3. ***需要使用 TouchKey_Scan_Slow 指令打开触摸键慢速扫描功能，PA0 Wakeup 才会生效。***

AutoJudge_Calibrate：触摸键自动环境校正。使用触摸键时，建议经过一段时间后要进行触摸键校正。系统提供自动更正，并可设定多久作一次自动更正（时间 0.5 ~ 60 秒，默认 4 秒）。用户可以将自动更正功能关闭，并自行利用时间路径来计数更长的时间。

; 触摸键每 4 秒自动更正一次。

例. **AutoJudge_Calibrate** = 4s

; 关闭触摸键自动更正功能。

; 关闭 4 秒自动更正功能。（可使用程序计算来决定校正的时间，请参考 4.2.8）

例. **AutoJudge_Calibrate** = Disable

注意：触摸键功能被关闭后，则自动更正功能也会被关闭。

Enforce_Calibrate：强制进行触摸键环境校正。为了避免触摸键非人为按下，而导致 IC 非预期性的工作，建议经过一段时间后要进行强制触摸键校正，若触摸键被误触，则因强制校正将此误触视为环境参数，可避免 IC 无法进入睡眠而持续耗电。若不使用系统提供的设定时间可自行将强迫校正功能关闭，由用户利用时间路径来计时，以进行校正。（时间 4 ~ 120 秒，默认 120 秒）

例. **Enforce_Calibrate** = 60s ; 触摸键每 60 秒自动更正一次。

例. **Enforce_Calibrate** = Disable ; 关闭触摸键自动更正功能。

*注意：当触摸键被按住时，**Enforce_Calibrate** 定时器才会开始计数，按键放开则会重置定时器。*

SW_Reset：在执行触摸键的强制校正后，是否进行重置 RAM 及 IO 状态。（默认值 Enable）

例. **SW_Reset** = Enable ; 触摸键强制校正后，进行重置。

例. **SW_Reset** = Disable ; 触摸键强制校正后，不进行重置。

*注意：用户如需要记忆设定或操作模式，可将 **SW_Reset** 功能关闭，改成在 **Enforce_Calibrate** 路径中来进行手动重置。请参考 **4.6.1** 强制校正路径的范例。*

Hysteresis Level：触摸键迟滞功能。可以调整触摸键按下后的灵敏度，避免摸到触摸点边缘部分时，可能会造成触摸键处于不稳定状态。（默认值 Small）

例. **TouchKey_Hysteresis_Level** = Disable ; 触摸键触发后，灵敏度不自动调整。

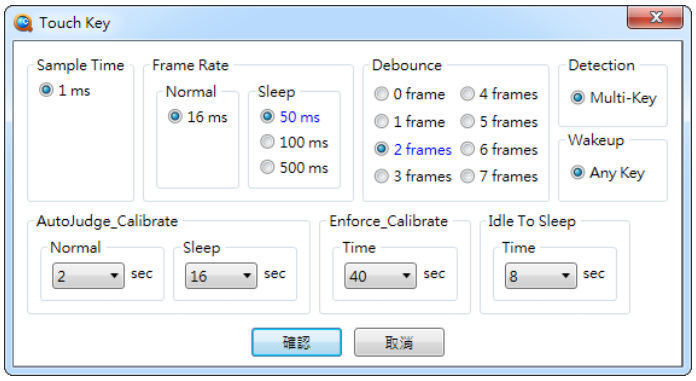
例. **TouchKey_Hysteresis_Level** = Small ; 触摸键触发后，该触摸键灵敏度提高一阶。

例. **TouchKey_Hysteresis_Level** = Large ; 触摸键触发后，该触摸键灵敏度增强 50%。

注意：

1. *通常触摸部位离 IC 的触摸脚位较远或是触摸键较为敏感时，会需要打开该功能。用户要实际测试产品，比较容易决定要使用哪一个设定。*
2. ***NY9T004A** 不支持 **Hysteresis Level** 功能。*

3.1.7.2 NX1



Sample Time：扫描一个触摸键的时间。较长的取样时间触摸更精确，但侦测到 TouchKey 被触摸的时间会较久。仅支援 1ms。

Frame Rate：在一个 Frame 要取样的数量，即完成所有触摸键的扫描时间。取样的数量愈多则较不耗电，但侦测到 TouchKey 被触摸的时间会较久。

例. **TouchKey_FrameRate_Sleep** = 50ms

Debounce：连续几个 Frame 被侦测到，按键才会成立。愈多 Frame 触摸更精确，但侦测到 TouchKey 被触摸的时间会愈久，对于较容易有外部干扰的环境建议选择多些 Frame。

例. **TouchKey_Debounce** = 2

Detection：择触摸键的侦测方式。Multi-Key 为可同时间多个触摸键被按下。

Wakeup：AnyKey 全部的触摸键皆可唤醒。

AutoJudge_Calibrate：触摸键自动环境校正。使用触摸键时，建议经过一段时间后要进行触摸键校正。系统提供自动更正，并可设定多久作一次自动更正。Normal Mode 可设置每隔 1 ~ 8 秒执行一次更正（默认值 2 秒）。Sleep Mode 可设定每 8 ~ 60 秒执行一次更正（默认值 16 秒）

例. **TouchKey_AutoJudge_Calibrate** = 4s ；**触摸键每 4 秒自动更正一次。**

例. **TouchKey_AutoJudge_Calibrate_Sleep** = 16s

注意：触摸键功能被关闭后，则自动更正功能也会被关闭。

Enforce_Calibrate：强制进行触摸键环境校正。为了避免触摸键非人为按下，而导致 IC 非预期性的工作，建议经过一段时间后要进行强制触摸键校正，若触摸键被误触，则因强制校正将此误触视为环境参数，可避免 IC 无法进入睡眠而持续耗电。若不使用系统提供的设定时间可自行将强迫校正功能关闭，由用户利用时间路径来计时，以进行校正。（时间 4 ~ 120 秒，默认值 40 秒）

例. **TouchKey_Enforce_Calibrate** = 60s ；**触摸键每 60 秒自动更正一次。**

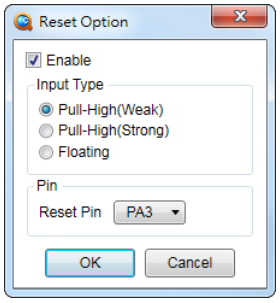
注意：当触摸键被按住时，Enforce_Calibrate 定时器才会开始计数，按键放开则会重置定时器。

Idle To Sleep：进入休眠模式的时间。可以调整持续多久没有被触发，会进入休眠模式。（时间 1 ~ 16 秒，默认值 8 秒）

例. **TouchKey_ IdleToSleep** = 7s ；**持续 7s 未触发，会进入休眠。**

3.1.8 Reset

3.1.8.1 NY4



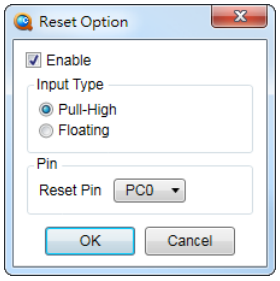
外部强制复位 IC 的脚位。Reset pin 可选择有 Pull-High 电阻 Weak 或是 Strong 或是 Floating。若用户选择 Floating，则需要外接 1M 电阻（默认值 Disable）。

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



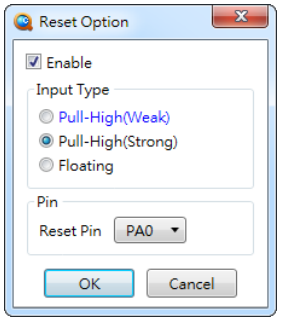
外部强制复位 IC 的脚位。Reset pin 可选择有 Pull-High 电阻或是 Floating。若用户选择 Floating，则需要外接 1M 电阻（默认值 Disable）。

例. **Reset = Pull-High**

NY5 可以选择 Reset pin 的脚位，NY6 / NY7 的 Reset pin 为固定脚位。

例. **Reset_Pin = PC.0**

3.1.8.3 NY5+

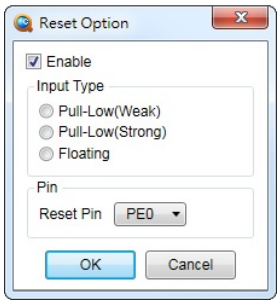


外部强制复位 IC 的脚位。Reset pin 可选择有 Pull-High 电阻 Weak 或是 Strong 或是 Floating。若用户选择 Floating，则需要外接 1M 电阻（默认值 Disable）。可以选择 Reset pin 的脚位。

例. **Reset = Floating**

例. **Reset_Pin = PA.0**

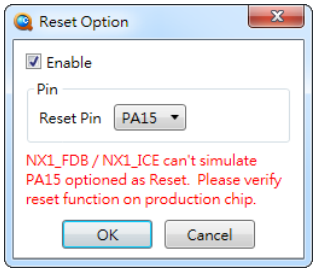
3.1.8.4 NY9T



外部强制复位 IC 的脚位。Reset pin 可选择有 Pull-Low 电阻或是 Floating。若用户选择 Floating，则需要外接 1M 电阻（预设 Disable）。

例. **Reset = F ; Floating**

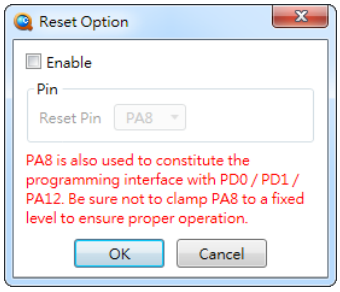
3.1.8.5 NX1 OTP



外部强制复位 IC 的脚位。

例. **Reset_Pin** = PA.15

3.1.8.6 NX1 EF



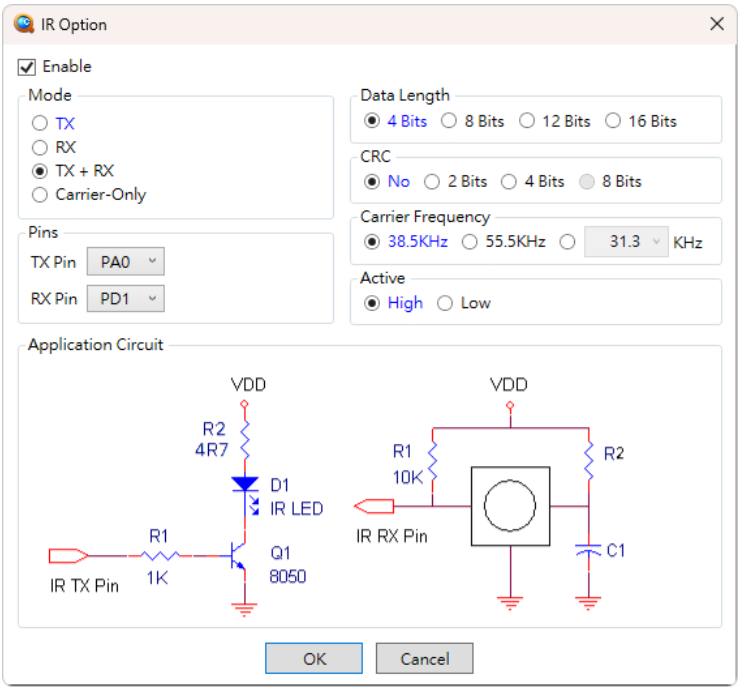
外部强制复位 IC 的脚位。

例. **Reset_Pin** = PA.15

3.1.9 IR

设定红外线发射及接收功能。系统已默认特定的 I / O 作为发射或接收脚，但需用户指定 IR 编码数据的长度与 IR 载波频率。

3.1.9.1 NY4 / NY5 / NY6 / NY7



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

例. **IR_Mode** = TX + RX

例. **IR_Mode** = Carrier_Only

TX Pin：选择 TX 脚位。只有 NY5 / NY5+ 可以选择脚位。

例. **IR_TX** = PA.1

RX Pin：选择 RX 脚位。

例. **IR_RX** = PA.2

Data Length：选择 IR 编码长度。

例. **IR_Bit** = 8

CRC：选择 IR 检查码的长度，默认为关闭检查码功能。

例. **IR_CRC** = 2

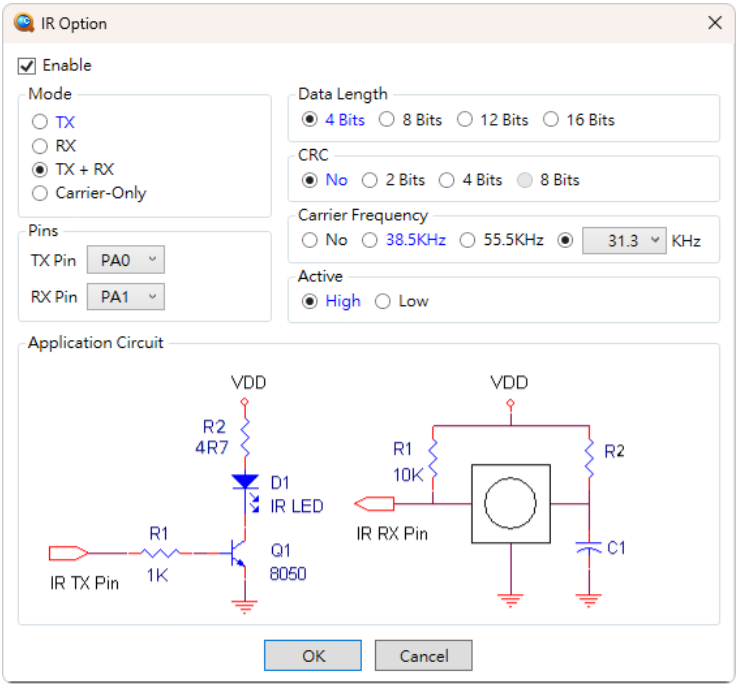
Carrier Frequency：选择 IR 载波频率。

例. **IR_Carrier_Freq** = 38.5K

Active：选择由 NPN 或者 PNP 晶体管来驱动 IR LED。

例. **IR_Active** = High

3.1.9.2 NY5+



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

例. **IR_Mode** = TX + RX

例. **IR_Mode** = TX + RX

例. **IR_Mode** = Carrier_Only

TX Pin：选择 TX 脚位。只有 NY5 / NY5+ 可以选择脚位。

例. **IR_TX** = PA.1

RX Pin：选择 RX 脚位。

例. **IR_RX** = PA.2

Data Length：选择 IR 编码长度。

例. **IR_Bit** = 8

CRC：选择 IR 检查码的长度，默认为关闭检查码功能。

例. **IR_CRC** = 2

Carrier Frequency：选择 IR 载波频率。

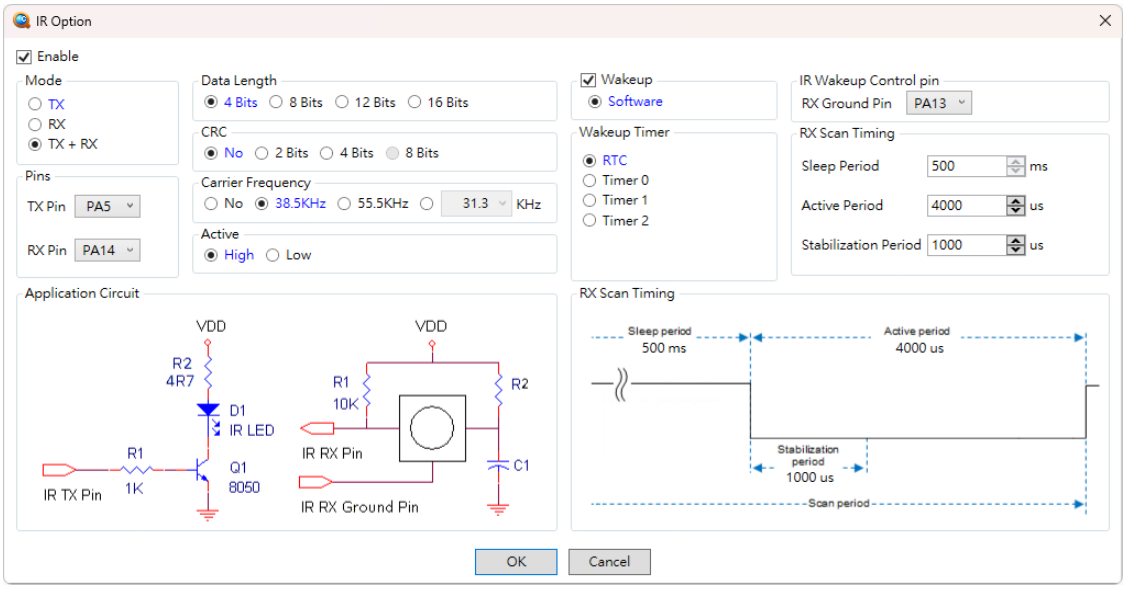
例. **IR_Carrier_Freq** = 38.5K

例. **IR_Carrier_Freq** = **Disable**

Active：选择由 NPN 或者 PNP 晶体管来驱动 IR LED。

例. **IR_Active** = High

3.1.9.3 NX1 OTP



Mode：选择打开 TX or RX 功能。

例. **IR_Mode** = TX + RX

例. **IR_Mode** = TX + RX

TX Pin：选择 TX 脚位。

例. **IR_TX** = PA.0

RX Pin：选择 RX 脚位。

例. **IR_RX** = PA.3

Data Length：选择 IR 编码长度。

例. **IR_Bit** = 8

CRC：选择 IR 检查码的长度，默认为关闭检查码功能。

例. **IR_CRC** = 2

Carrier Frequency：选择 IR 载波频率。

例. **IR_Carrier_Freq** = 38.5K

例. **IR_Carrier_Freq** = **Disable**

Active：选择由 NPN 或者 PNP 晶体管来驱动 IR LED。

例. **IR_Active** = High

Wakeup：选择打开 IR RX Wakeup 功能。

例. **IR_Wakeup** = Enable

IR_Wakeup_Type = SW

Wakeup Timer：选择 IR RX Wakeup 使用的 Timer。

例. **IR_Wakeup_Timer** = RTC

IR RX Wakeup Control Pin：选择 RX Ground 脚位。

例. **IR_Wakeup_Pin** = PA.3

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

例. **IR_Wakeup_Sleep_Period** = 500ms

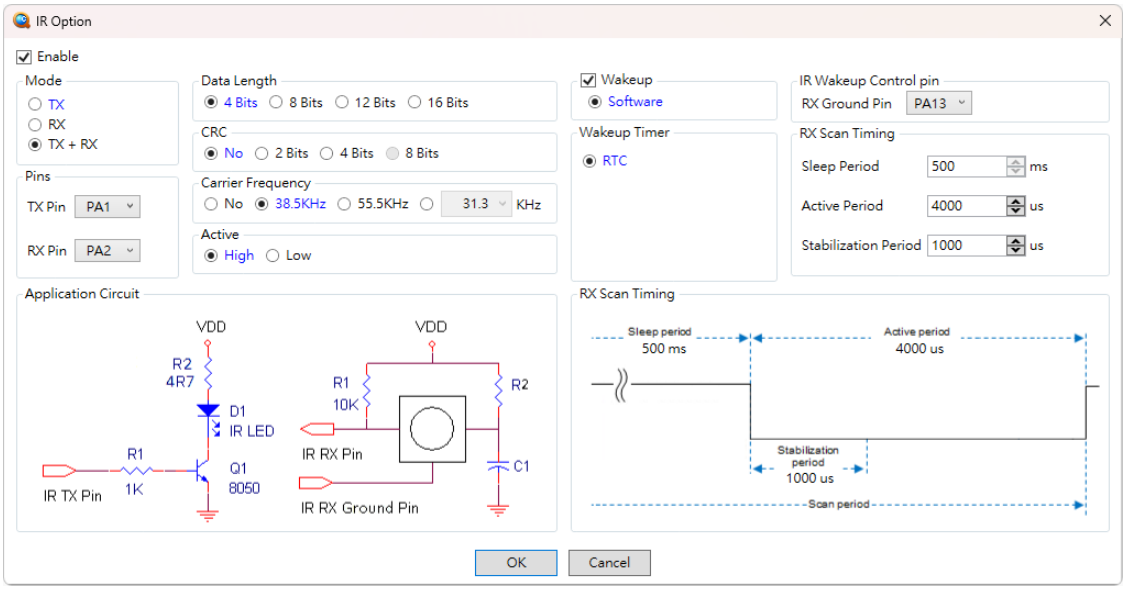
Active Period (RX Scan Timing)：选择 RX Active Period。

例. **IR_Wakeup_Active_Period** = 1000us

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

例. **IR_Wakeup_Stabilization_Period** = 4000us

3.1.9.4 NX1 EF



Mode：选择打开 TX or RX 功能。

例. **IR_Mode** = TX

例. **IR_Mode** = RX

例. **IR_Mode** = TX + RX

TX Pin：选择 TX 脚位。

例. **IR_TX** = PA.0

RX Pin：选择 RX 脚位。

例. **IR_RX** = PA.3

Data Length：选择 IR 编码长度。

例. **IR_Bit** = 8

CRC：选择 IR 检查码的长度，默认为关闭检查码功能。

例. **IR_CRC** = 2

Carrier Frequency：选择 IR 载波频率。

例. **IR_Carrier_Freq** = 38.5K

例. **IR_Carrier_Freq** = 52.6K

例. **IR_Carrier_Freq** = **Disable**

Active：选择由 NPN 或者 PNP 晶体管来驱动 IR LED。

例. **IR_Active** = High

例. **IR_Active** = Low

Wakeup：选择打开 SW / HW IR RX Wakeup 功能。

例. **IR_Wakeup** = **Enable**

IR_Wakeup_Type = HW

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

Wakeup Timer：选择 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：选择 RX Ground 脚位。

例. **IR_Wakeup_Pin** = PA.3

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

例. **IR_Wakeup_Sleep_Period** = 500ms

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

例. **IR_Wakeup_Active_Period** = 4000us

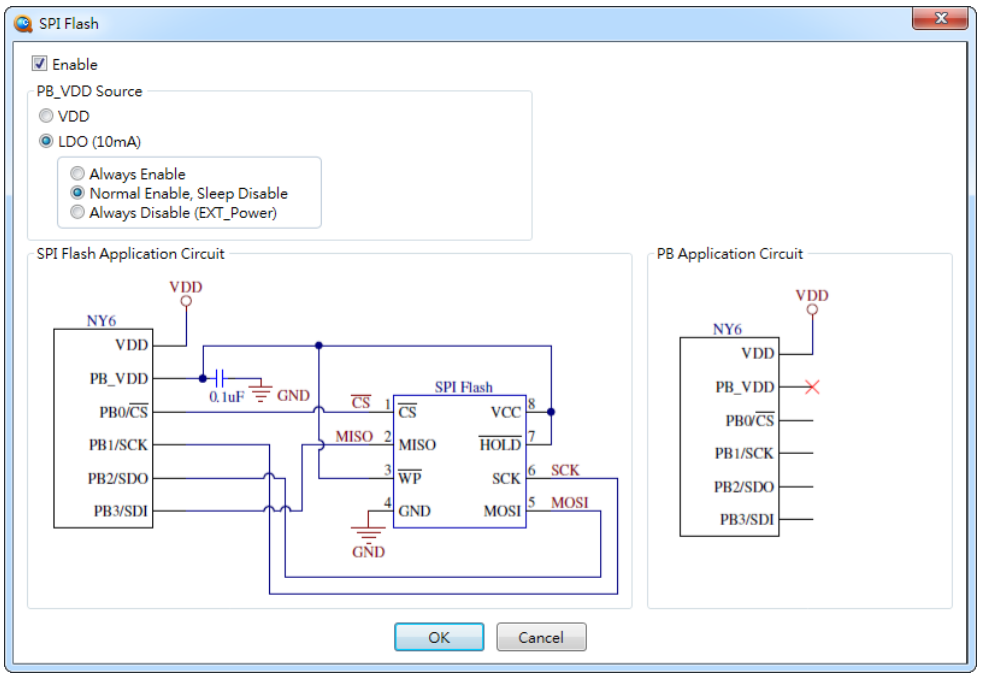
Stabilization Period (RX Scan Timing)：选择 RX Stabilization Period。

例. **IR_Wakeup_Stabilization_Period** = 1000us

3.1.10 SPI Flash

串行总线接口（Serial Peripheral Interface），通过此 SPI 窗口配合 Voice_Encoder V3.30 或之后的版本，进行播放储存于外面 SPI Flash 的 Voice。亦可通过 SPI 传输接口以 Master Mode 和其他 SPI 的装置进行通信。

3.1.10.1 NY6



PB_VDD Source：PB 电源之来源。

1. VDD：PB 电源由 VDD 提供。（Real chip 需自行将 PB_VDD 绑定到 VDD）
2. LDO Always Enable：PB 电源由 IC 内部提供。
3. LDO Normal Enable, Sleep Disable：PB 电源由 IC 内部提供，当 IC 进入休眠时，LDO 关闭。
4. LDO Always Disable (EXT_Power)：PB 电源不由 VDD 提供，也不由 LDO 提供，PB_VDD 可接其他外部电源，若外接电源大于 IC VDD，会有漏电的情形发生。

注意：

1. 当 PB 作为 SPI Flash 应用，并使用 IC 内部提供电源时，PB_VDD 需要外加一颗 0.1uF 电容。

2. 固定使用 PB 所有的脚位作为 SPI 沟通的脚位，各脚位对应如下：

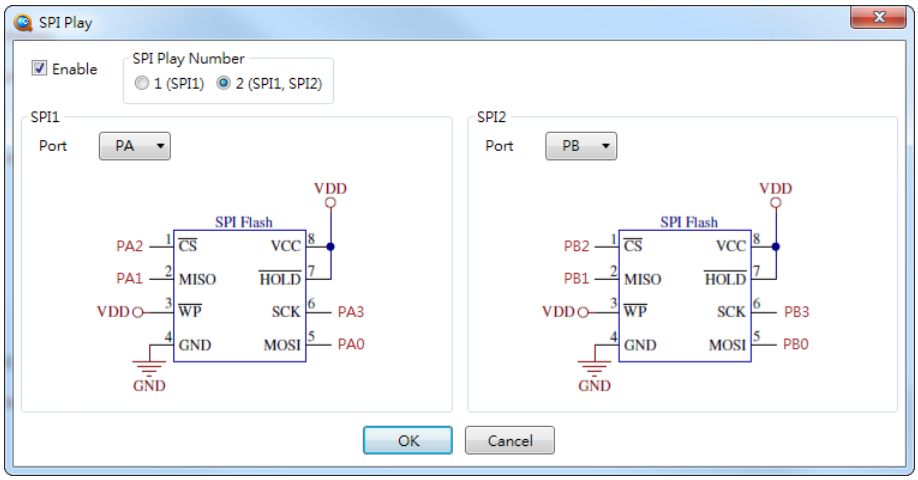
CSB：PB.0　　SCK：PB.1　　MOSI：PB.2　　MISO：PB.3

3. PB 当一般 IO 使用，不当 SPI 使用时，PB 的电源亦可选择。

例. **SPI** = Enable

PB_VDD = VDD

3.1.10.2 NY7



SPI Play Number：支持二组 SPI 设定。

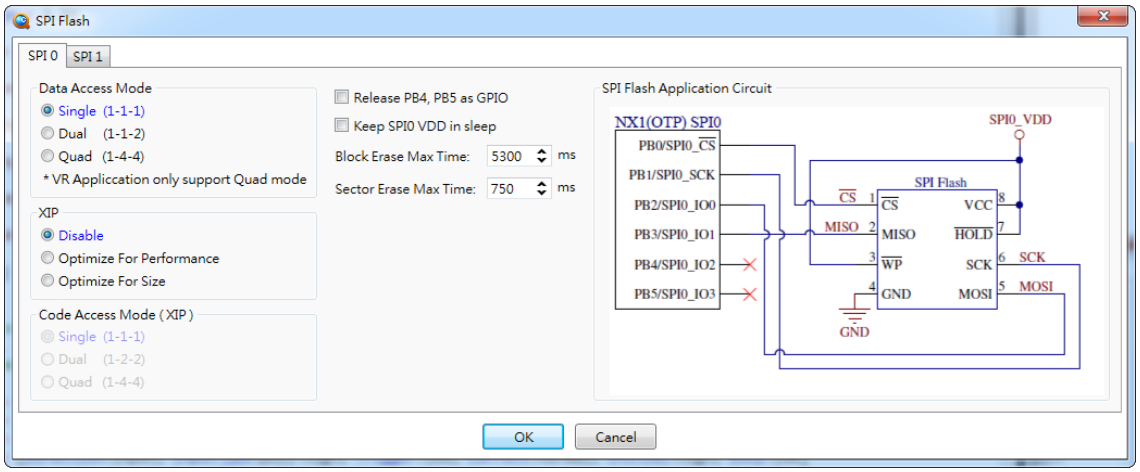
注意：目前只支持 SPI Dual Mode，固定使用完整的 port 作为 SPI 沟通的脚位，各脚位对应如下：

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：可选择 Single / Dual / Quad。

例. **SPI0_Data_Access_Mode** = Single

XIP：可选择 Disable / Optimze For Performance / Optimze For Size

1. Optimze For Performance 可将部份用户程序放到 SPI Flash
2. Optimze For Size 可将更多程序放到 SPI Flash，但 CPU 负载相对会上升。

例. **XIP** = Speed_Optimize

Code Access Mode：可选择 Disable / Single / Dual

例. **SPI0_Data_Access_Mode** = Single

注意：使用 XIP 建议选择 Quad，执行速度会比 Single 和 Dual 快一些，CPU 负载可相对下降一些。

Release PB4, PB5 as GPIO：当 Data Access Mode 与 Code Access Mode 不选用 Quad 时，将 PB.4 及 PB.5 开放为 GPIO 使用。

例. **Release_PB4_PB5** = Disable

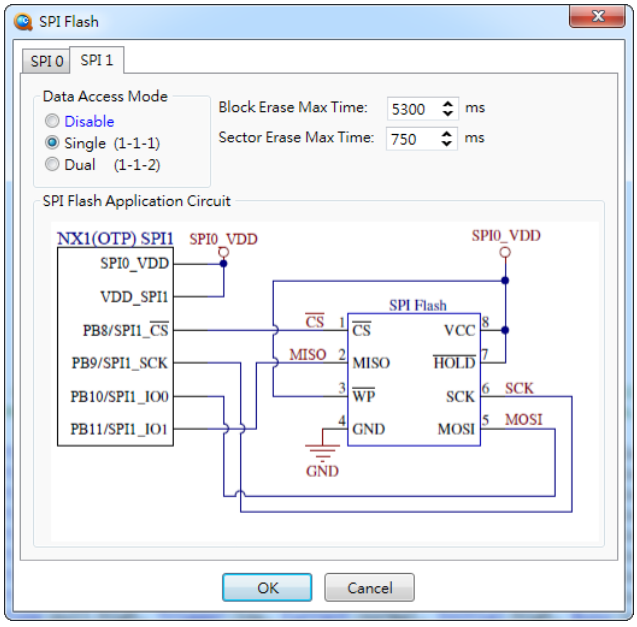
Block Erase Max Time：使用 Block Erase 指令发生 Flash 等待逾时，将停止等待并立刻返回，建议依照 Flash 规格进行合适的时间设定。

例. **SPI0_BE_MaxTime** = 5300ms

Sector Erase Max Time：使用擦除指令发生 Flash 等待逾时，将停止等待并立刻返回，建议依照 Flash 规格进行合适的时间设定。

例. **SPI0_SE_MaxTime** = 750ms

- **SPI1**



Data Access Mode：可选择 Disable / Single / Dual。

例. **SPI1_Data_Access_Mode** = Single

Block Erase Max Time：使用 Block Erase 指令发生 Flash 等待逾时，将停止等待并立刻返回，建议依照 Flash 规格进行合适的时间设定。

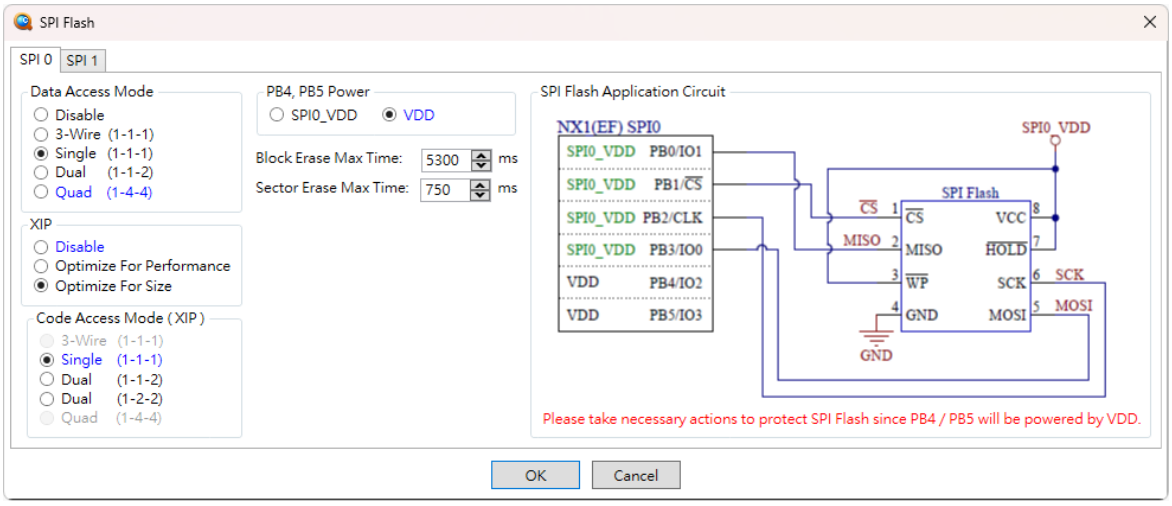
例. **SPI1_BE_MaxTime** = 5300ms

Sector Erase Max Time：使用擦除指令发生 Flash 等待逾时，将停止等待并立刻返回，建议依照 Flash 规格进行合适的时间设定。

例. **SPI1_SE_MaxTime** = 750ms

3.1.10.4 NX1 EF

- **SPI0**



Data Access Mode：可选择 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：可选择 Disable / Optimze For Performance / Optimze For Size。

1. Optimze For Performance 可将部份用户程序放到 SPI0 Flash。
2. Optimze For Size 可将更多程序放到 SPI Flash，但 CPU 负载相对会上升。

例. **XIP** = Speed_Optimize

Code Access Mode：可选择 Disable / 3-Wire / Single / Dual。

例. **SPI1_Data_Access_Mode = Single**

注意：使用 XIP 建议选择 Quad，执行速度会比 Single 和 Dual 快一些，CPU 负载可相对下降一些。

PB4, PB5 Power：当 Data Access Mode 与 Code Access Mode 选择 3-Wire / Single / Dual 时，将可以设定 PB4, PB5 的电源。

例. **PB45_Power_Source** = SPI0_VDD

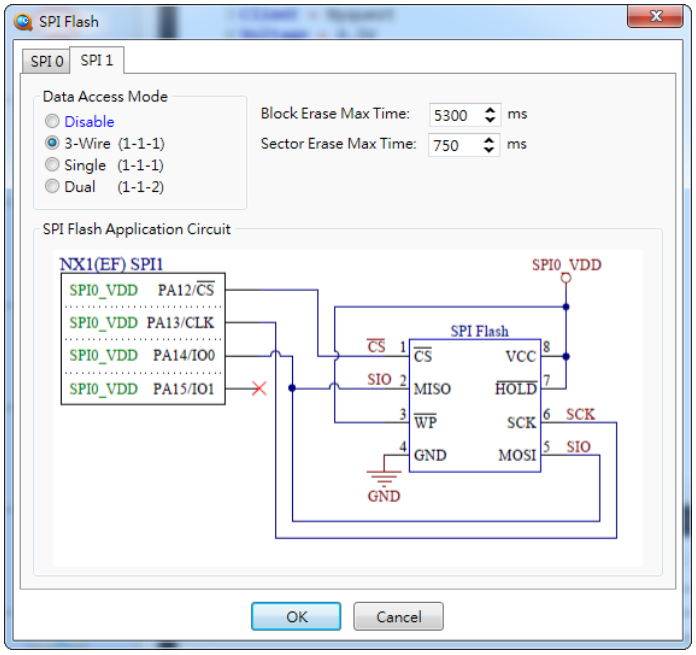
Block Erase Max Time：在使用 Block Erase 指令发生 Flash 等待逾时，将停止等待并立刻返回，建议依照 Flash 规格进行合适的时间设定。

例. **SPI0_BE_MaxTime** = 5300ms

Sector Erase Max Time：在使用擦除指令发生 Flash 等待逾时，将停止等待并立刻返回，建议依照 Flash 规格进行合适的时间设定。

例. **SPI0_SE_MaxTime** = 750ms

- **SPI 1**



Data Access Mode：可选择 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：在使用 Block Erase 指令发生 Flash 等待逾时，将停止等待并立刻返回，建议依照 Flash 规格进行合适的时间设定。

例. **SPI1_BE_MaxTime** = 5300ms

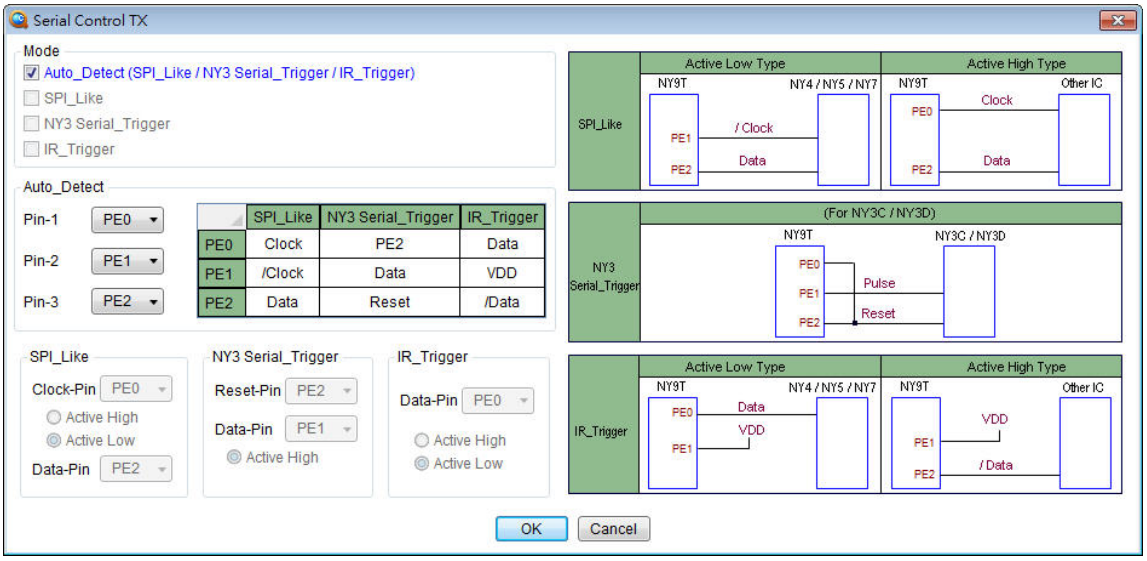
Sector Erase Max Time：在使用擦除指令发生 Flash 等待逾时，将停止等待并立刻返回，建议依照 Flash 规格进行合适的时间设定。

例. **SPI1_SE_MaxTime** = 750ms

3.1.11 Serial Control TX

3.1.11.1 NY9T

可连接一般微控制器，将 NY9T 当成微控制器的按键。提供 3 种不同的传输协议，分别是 SPI_Like、NY3 Serial_Trigger、IR_Trigger。用户可以设定成自动侦测或是特定的传输协议，通信时序图请参考串行信号控制通信协议。



Auto Detect：可由 3 根脚位的设定方式来决定，NY9T 要使用何种传输协议。自动侦测的脚位设定方式如下表所示：

自动模式侦测			
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 ；自动侦测传输协议。

注意：Auto 模式时，3 根脚位都必须在同一个埠。

SPI_Like：使用 2 根脚位，达成类似 SPI 传输协议。

例. **SerialControl** = SPI_Like

SerialControl_SPI_Clock = PF.2

SerialControl_SPI_Data = PF.3

SerialControl_SPI_ClockTrigger = High or Low ；传输协议为正缘触发或是负缘触发。

注意：SPI_Like 模式时，2 根脚位都必须在同一个埠。

NY3 Serial_Trigger：用户可以通过串行信号与 NY3 联机，将 NY9T 当成 NY3 的触发按键。

例. **SerialControl** = NY3

SerialControl_NY3_Reset = PF.3

SerialControl_NY3_Data = PF.2

IR_Trigger：使用 1 个脚位来仿真 IR 的接收信号，可直接使用目前的 NY4/5/7 系列的 IR 通信。

例. **SerialControl** = IR_Trigger

SerialControl_IR_Data = PF.3

SerialControl_IR_DataBusy = High or Low ；传输协议为正缘触发或是负缘触发。

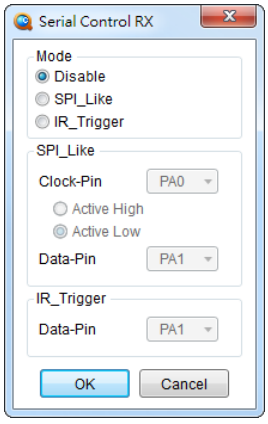
注意：

1. *同一时间只能有一种传输协议。*
2. *打开 Auto-Detect 功能时，PIN-1 会被设成 Input Floating，PIN-2/PIN-3 会被设成 Input Weak-Pull-Low。*
3. *打开 Auto-Detect 功能时，接收端的 IC，必须将连接脚设定成 Input floating。*
4. *NY9T001A/NY9T004A 不支持 Serial Control 功能。*

3.1.12 Serial Control RX

3.1.12.1 NY4 / NY5 / NY5+ / NY6 / NY7

接收由外部送出的串行数据。可使用 SPI_Like / IR_Trigger 两种传输协议，通信时序图请参考串行信号控制通信协议。



SPI_Like：使用 2 根脚位，类似 SPI 传输协议。

例. **Serial_Rx** = SPI_Like

Serial_Rx_Data = PA.1

Serial_Rx_Clock = PA.0

Serial_Rx_ClockTrigger = High

IR_Trigger：使用 1 根脚位，使用 IR 协议。

例. **Serial_Rx** = IR_Trigger

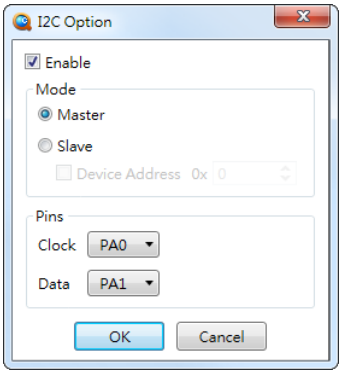
Serial_Rx_Data = PF.1

注意：

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

3.1.13 I2C

3.1.13.1 NY5+



Mode：可以选择 Master 模式或 Slave 模式。传输速度为 250Hz (传输一笔数据约 36ms)。

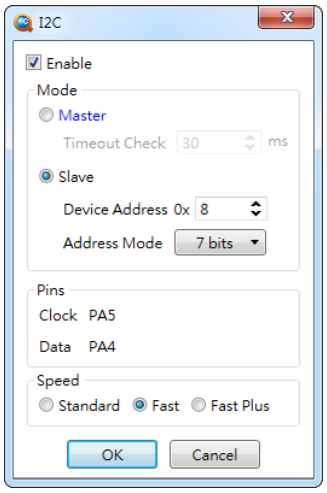
Device Address：做为 Slave 时的 Address，可为 0 ~ 127。

Clock：选择 Clock(SCL) 脚位。

Data：选择 Data(SDA) 脚位。

注意：Clock 与 Data 脚位必须为相同的 Port。

3.1.13.2 NX1



Mode：可以选择 Master 模式或 Slave 模式。传输速度可支持 Standard (100kHz) / Fast (400kHz) / Fast+ (1MHz)。

Timeout Check：作为 Master 端时，间隔多久无响应即视为逾时，1 ~ 30ms，设 0 则永不逾时。

Device Address：做为 Slave 时的 Address。

Device Address Bit：做为 Slave 端时，指定 Address 为 7 / 10 bits。

Clock：选择 Clock(SCL) 脚位。

Data：选择 Data(SDA) 脚位。

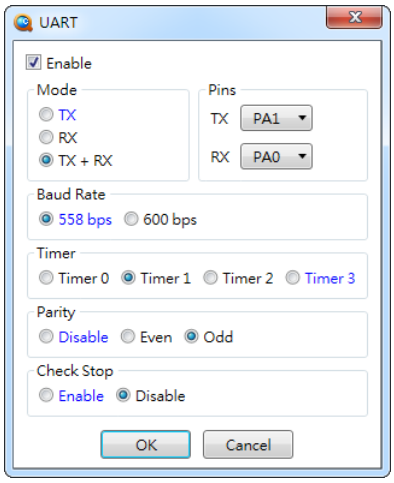
Speed：传输速度，支持 Standard / Fast / Fast Plus，默认为 Fast。

注意：当 NX1 作为 Slave 端时，Master 端需支持 Clock Stretching，否则行为会异常。

3.1.14 UART

Q-Code 支持 UART(Universal Asynchronous Receiver / Transmitter)通信协议。

3.1.14.1 NY5+



Mode：可选择 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：用来产生所选择 Baud Rate 的定时器。

例. **UART_ Timer_Channel** = 1

Parity：是否进行奇偶校验，可选择 Disable / Even / Odd。

例. **UART_ Parity** = Odd

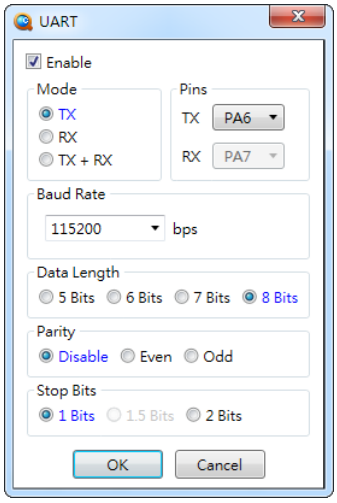
Check Stop：是否检查 StopBit。

例. **UART _Check_Stop** = **Disable**

注意：

1. ***TX 与 RX 脚位必须为相同的 Port。***
2. ***UART 使用时，所选择的定时器无法进行 Voice 播放或 Timer 应用。***

3.1.14.2 NX1



Mode：可选择 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 同时使用 TX 与 RX 时，必须为相同的 Port。***
2. ***NX1 EF 系列 TX 仅支持 PA.6 / PD.0，若要使用 PD.0 作为 TX，必须先在 I/O 段落将 PD.0 设置为 GPIO。***
3. ***NX1 EF 系列 RX 仅支持 PA.7/ PD.1，若要使用 PD.1 作为 RX，必须先在 I/O 段落将 PD.1 设置为 GPIO。***

Baud Rate：UART 的 Baud Rate。

例. **UART_BaudRate** = 558bps

Data Length：UART 的数据位长度。

例. **UART_ Bit** = 1

Parity：是否进行奇偶校验，可选择 Disable / Even / Odd。

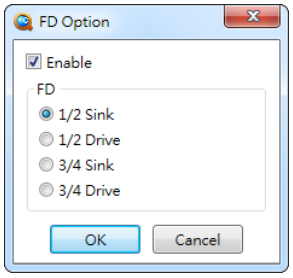
例. **UART_ Parity** = Odd

Stop Bits：UART 的 Stop Bit。

例. **UART _Stop _Bit** = 1

3.1.15 FD

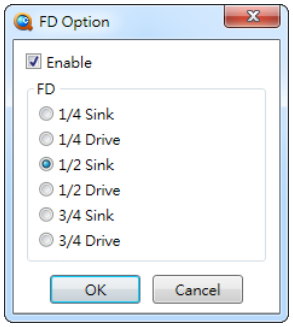
3.1.15.1 NY4 / NY5



Flash with Dynamic 随着音量的大小来闪动。可以选择大于 1 / 2 或者是 3 / 4 音量来闪动，并且可以设定输出型态为 Drive 或是 Sink 的方式。（D 为 Drive，S 为 Sink。默认 Disable）

例. **FD** = 1/2D

3.1.15.2 NY5+ / NY6 / NY7 / NX1

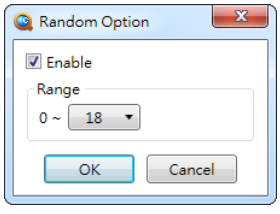


Flash with Dynamic 随着音量的大小来闪动。可以选择大于 1/4、1 / 2 或者是 3 / 4 音量来闪动，并且可以设定输出型态为 Drive 或是 Sink 的方式。（D 为 Drive，S 为 Sink。默认 Disable）

例. **FD** = 1/4D

3.1.16 Random

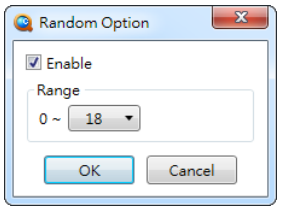
3.1.16.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T



Random 表示产生出来的随机数范围，默认值 Disable，可选 0~255 之间。

例. **Random** = 122

3.1.16.2 NX1

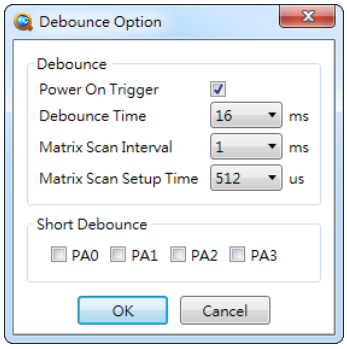


Random 表示产生出来的随机数范围，默认值 Disable，值可选 0 ~ 65535 之间。

例. **Random** = 822

3.1.17 Debounce

3.1.17.1 NY4 / NY5



Power On Trigger：选择是否要有 Power on Trigger 功能。当功能打开后，如果用户按着按键后开机，则开机后会马上去执行该按键的路径；反之，则不执行该按键路径。（默认 Enable）

注意：如果打开此功能，则 PowerOn 路径必须设定 input state，否则会因为找不到设定路径而无效。

例. **PowerOnTrigger** = Disable ；关闭所有按键的上电触发功能。

Debounce Time：用户可以选择按键的 debounce 时间，时间范围：1~1000ms。（默认值=16ms）

注意：Q-Code 实际上是采用轮询（Polling）的方式。原则上，实际按键的时间必须大于设定的 debounce 时间，如需要很精准的 debounce 时间或不同的 debounce 时间，需用汇编语言来编写。

例. **Debounce** = 16 ms ；全部按键的 **Debounce** 时间为 **16ms**。

Matrix Scankey Interval：用户可以选择 Matrix Key 的扫描的时间间隔。可支持 0 ~ 4ms（默认值 1ms）。

注意：Q-Code 实际上是采用轮询（Polling）的方式处理此时间，所以实际时间会根据当时的负载轻重而有所误差。

例. **ScankeyInterval** = 4ms ；按键的扫描时间间隔为 **4ms**。

Matrix Scan Setup Time：用户可以选择矩阵扫描的输出设定时间；假设此时间设定为 256us，则每

支输出脚设定为输出低电平 后，会延迟 256us 才抓取输入脚的状态，以避免输入脚在下拉过程中，尚未稳定的情况下抓取状态造成误判。支持 0 / 512 / 1024 / 1536us（默认 512us）。

注意：Q-Code 实际上是采用轮询（Polling）的方式处理此时间，所以实际时间会根据当时的负载轻重而有所误差。

例. **Matrix_Scan_SetupTime** = 256us ；**设定矩阵扫描设定时间为 256us。**

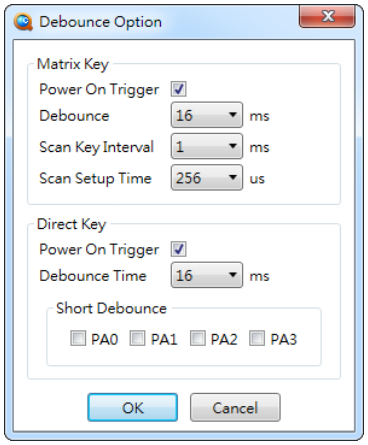
Short Debounce 用户可以选择哪一个独立按键的 Short debounce，时间范围：0~1ms。

注意：

1. ***此功能不支持 Matrix Key***
2. ***Q-Code 实际上是采用轮询（Polling）的方式，所以实际 debounce 时间会有所误差。如需要很精准的 debounce 时间或不同的 debounce 时间，需用汇编语言来编写。***

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

3.1.17.2 NY5+ / NY6 / NY7



Power On Trigger：选择是否要有 Power on Trigger 功能。当功能打开后，如果用户按着按键后开机，则开机后会马上去执行该按键的路径；反之，则不执行该按键路径。允许个别设定 Direct Key 及 Matrix Key 是否开启此功能。（默认值 Enable）

注意：如果打开此功能，则 PowerOn 路径必须设定 input state，否则会因为找不到设定路径而无效。

例. **Direct_PowerOnTrigger** = Disable ；**关闭 Direct Key 上电触发功能。**

例. **Matrix_PowerOnTrigger** = Disable ；**关闭 Matrix Key 上电触发功能。**

Debounce Time：用户可以选择按键的 debounce 时间，时间范围：1~1000ms。允许 Direct Key 及 Matrix Key 设定不同的 debounce 时间。（默认值=16ms）

注意：Q-Code 实际上是采用轮询（Polling）的方式。原则上，实际按键的时间必须大于设定的 debounce 时间，如需要很精准的 debounce 时间或不同的 debounce 时间，需用汇编语言来编写。

例. **Direct_Debounce** = 16 ms ; **Direct Key 的 Debounce 时间为 16ms。**

例. **Matrix_Debounce** = 16 ms ***; Matrix Key 的 Debounce 时间为 16ms。***

Scankey Interval：用户可以选择 Matrix Key 的扫描的时间间隔。可支持 0 ~ 4ms（默认值 1ms）。

***注意：*Q-Code 实际上是采用轮询（Polling）的方式处理此时间，所以实际时间会根据当时的负载轻重而有所误差。**

例. **ScankeyInterval** = 4ms ; **按键的扫描时间间隔为 4ms。**

Matrix Scan Setup Time：用户可以选择矩阵扫描的输出设定时间；假设此时间设定为 256us，则每支输出脚设定为输出低电平 后，会延迟 256us 才抓取输入脚的状态，以避免输入脚在下拉过程中，尚未稳定的情况下抓取状态造成误判。支持 0 / 512 / 1024 / 1536us（默认值=512us）。

***注意：*Q-Code 实际上是采用轮询（Polling）的方式处理此时间，所以实际时间会根据当时的负载轻重而有所误差。**

例. **Matrix_Scan_SetupTime** = 256us ; **设定矩阵扫描设定时间为 256us。**

Short Debounce 用户可以选择哪一个独立按键的 Short debounce，时间范围：0~1ms。

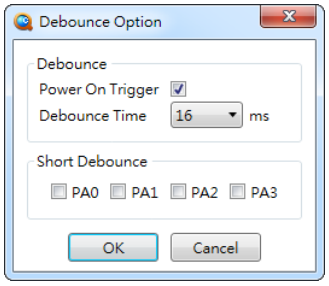
注意：

1. 此功能不支持 Matrix Key

2. Q-Code 实际上是采用轮询（Polling）的方式，所以实际 debounce 时间会有所误差。如需要很精准的 debounce 时间或不同的 debounce 时间，需用汇编语言来编写。

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

3.1.17.3 NY9T



Power On Trigger：选择是否要有 Power on Trigger 功能。当功能打开后，如果用户按着按键后开机，则开机后会马上去执行该按键的路径；反之，则不执行该按键路径。（默认值 Enable）

注意：

1. 上电触发功能仅对一般按键有效，触摸键不支持该功能。

2. 如果打开此功能，则 PowerOn 路径必须设定 input state，否则会因为找不到设定路径而无效。

例. **PowerOnTrigger** = Disable ; **关闭所有按键的上电触发功能。**

Debounce Time：用户可以选择按键的 debounce 时间，时间范围：1~1000ms。（默认值=16ms）

注意：Q-Code 实际上是采用轮询（Polling）的方式。原则上，实际按键的时间必须大于设定的 debounce 时间，如需要很精准的 debounce 时间或不同的 debounce 时间，需用汇编语言来编写。

例. **Debounce** = 16 ms ；全部按键的 **Debounce** 时间为 **16ms**。

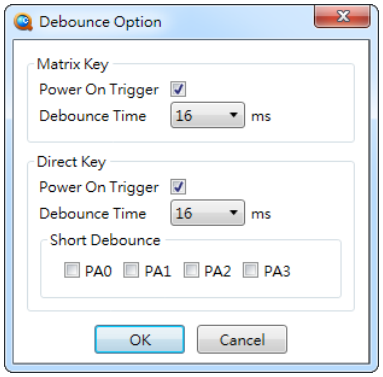
Short Debounce 用户可以选择哪一个独立按键的 Short debounce，时间范围：0~1ms。

注意：

1. *此功能不支持 Matrix Key*
2. *Q-Code 实际上是采用轮询（Polling）的方式，所以实际 debounce 时间会有所误差。如需要很精准的 debounce 时间或不同的 debounce 时间，需用汇编语言来编写。*

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

3.1.17.4 NX1



Power On Trigger：选择是否要有 Power on Trigger 功能。当功能打开后，如果用户按着按键后开机，则开机后会马上去执行该按键的路径；反之，则不执行该按键路径。允许个别设定 Direct Key 及 Matrix Key 是否开启此功能。（默认值 Enable）

注意：如果打开此功能，则 PowerOn 路径必须设定 input state，否则会因为找不到设定路径而无效。

例. **Direct_PowerOnTrigger** = Disable ；关闭 **Direct Key** 上电触发功能。

例. **Matrix_PowerOnTrigger** = Disable ；关闭 **Matrix Key** 上电触发功能。

Debounce Time：用户可以选择按键的 debounce 时间，时间范围：1~1000ms。允许 Direct Key 及 Matrix Key 设定不同的 debounce 时间。（默认值=16ms）

注意：Q-Code 实际上是采用轮询（Polling）的方式。实际按键的时间必须大于设定的 debounce 时间，如需要很精准的 debounce 时间或不同的 debounce 时间，需用汇编语言来编写。

例. **Direct_Debounce** = 16 ms ; Direct Key 的 Debounce 时间为 16ms。

例. **Matrix_Debounce** = 16 ms *; Matrix Key 的 Debounce 时间为 16ms。*

Short Debounce 用户可以选择哪一个独立按键的 Short debounce，时间范围：0~1ms。

注意：

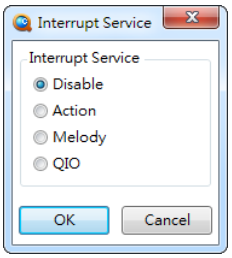
1. *此功能不支持 **Matrix Key***
2. *Q-Code 实际上是采用轮询（Polling）的方式，所以实际 debounce 时间会有所误差。如需要很精准的 debounce 时间或不同的 debounce 时间，需用汇编语言来编写。*

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

3.1.18 Interrupt Service

3.1.18.1 NY5

提供 3 种不同的功能中断服务，分别为 Action 与 QIO 及 Melody。若对于精确度有较高的要求，可以将该功能改由中断来处理。



注意：

1. ***NY5+ / NY6 / NY7 Q-Code 会固定将中断服务打开，以确保所有功能皆能达到较高的精确度，故不支持此选项。***
2. *中断服务同时仅能支持一种功能，设定后程序执行过程无法变更。*
3. *中断服务于功能启动时，系统自动致能，无须下达 **INT_ON** 指令。*
4. *使用中断服务会占用较多的 **RAM**。*
5. ***NY5** 的 **ADSR** 功能默认为打开中断，**PCT** 模式默认为关闭，其余功能默认皆为关闭。*
6. *与时间中断指令混用会导致无法预期的结果。*

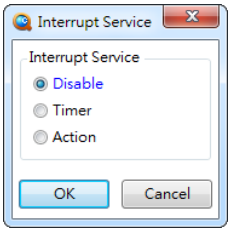
例. **InterruptService** = Action ; **Action 功能使用中断。**

例. **InterruptService** = QIO ; **QIO 功能使用中断。**

例. **InterruptService** = Melody ; **Melody 功能使用中断。**

3.1.18.2 NY9T

提供 2 种不同功能的中断服务，分别为 Action 与 Timer。若对于精确度有较高的要求，可以将该功能改由中断来处理。



注意：

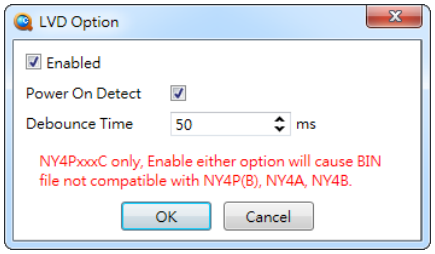
1. *中断服务同时仅能支持一种功能，设定后程序执行过程无法变更。*
2. *中断服务于功能启动时，系统自动致能，无须下达 **INT_ON** 指令。*
3. *使用中断服务会占用较多的 **RAM**。*
4. *与时间中断指令混用会导致无法预期的结果。*
5. ***NY9T** 使用 **Action** 中断服务会占用较多的 **RAM**，且会一并激活 **Timer** 中断服务。*

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

3.1.19 LVD

用户可以透过 IC 内建 LVD 功能对系统进行监控，避免电压不稳定导致系统出错，当系统电压有变动时，程序跳至对应的系统路径。

3.1.19.1 NY4PxxxC



Enabled：开启或关闭电压侦测系统。

例. **LVD = Enable**

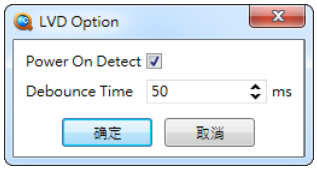
Power On Detect：上电随即侦测目前的系统电压，并跳至对应的系统路径执行程序。

例. **PowerOnLVD = Enable**

Debounce：电压变化需维持一段时间，才会判定电压确实有变化。

例. **LVD_Debounce** = 50ms

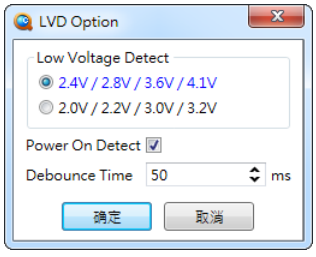
3.1.19.2 NY5+ / NY6B / NY6C / NX1



Power On Detect：上电随即侦测目前的系统电压，并跳至对应的系统路径执行程序。

Debounce：电压变化需维持一段时间，才会判定电压确实有变化。

3.1.19.3 NY6P025A



NY6P025A 支援 2 组电压侦测。

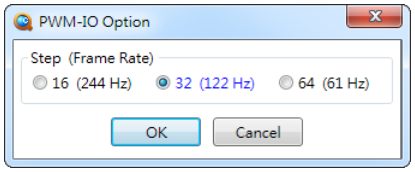
例. **LVD_Selection** = LVD2

Power On Detect：上电随即侦测目前的系统电压，并跳至对应的系统路径执行程序。

Debounce：电压变化需维持一段时间，才会判定电压确实有变化。

3.1.20 PWM-IO

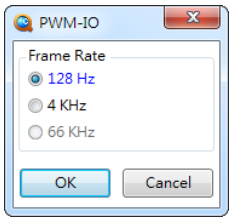
3.1.20.1 NY4 / NY5 / NY6 / NY7



PWMIO_Step：设定 PWMIO 的阶数，可为 16 / 32 / 64，分别对应的 frame rate 为 244Hz / 122Hz / 61Hz。

例. **PWMIO_Step** = 32 ; **PWM 设定输出阶数 32。**

3.1.20.2 NY9T



Frame Rate：PWM-IO 信号的更新率（NY9T001A 预设为 4KHz，其余系列默认为 128Hz）。

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

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

注意：

1. ***更新率越高，LED 越不容易闪烁。***
2. ***NY9T001A 设定 FrameRate=66KHz 时，PWM-IO 仅能提供一个 PWM-IO 输出，但能设定于 PE.0 / PE.1 / PE.2 中的任一脚位，其余脚位仅能设定成一般 IO。（仅有 NY9T001A 提供 66K 选项）***
3. ***NY9T001A / NY9T008A / NY9T016A 设定 FrameRate=4KHz 时，PWM-IO 仅能使用 PE.0 / PE.1 / PE.2 脚位，其余脚位仅能设定成一般 IO。（NY9T004A 仅提供 128Hz）***

NY9T 母体支持的 PWM 更新率和可设置的脚位:

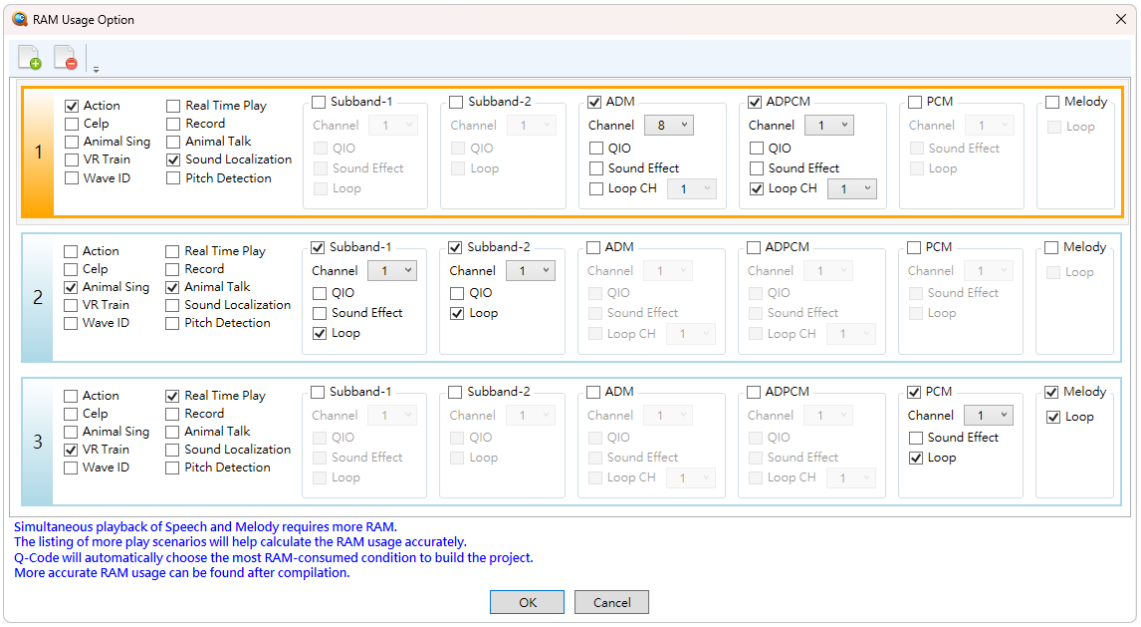
	NY9T001A		NY9T004A	NY9T008A		NY9T016A	
Frame Rate	4KHz	66KHz	128Hz	128Hz	4KHz	128Hz	4KHz
Pin	PE.0 ~ PE.2	one of PE.0 ~ PE.2 pin only	PE	PE ~ PF	PE.0 ~ PE.2	PE ~ PF	PE.0 ~ PE.2

3.1.21 RAM Usage

3.1.21.1 NX1

指定各种功能同时使用的使用条件，依据此设定可推估执行期间占用的内存空间。当有同时使用到两个 channel 时必须使用此选项，必须使用 RAM Usage 设定同时间使用的功能组合。每一种功能组合表示同一个时间，同时会执行的功能。

假设同时播放 SBC 20sec / ADPCM 10 sec，ADPCM 播放完后马上播放 PCM(SBC 还未结束)，这样要选择 SBC + ADPCM + PCM。



Action：打开动作信号输出。

Celp：使用 Celp 播放。

Animal Sing：打开动物唱歌功能。

VR Train：打开语音卷标训练功能。

WaveID：打开 WaveID 功能。

Real Time Play：打开实时播放(大声公)功能。

Record：打开录音功能。

Animal Talk：打开动物音功能。

Sound Localization：打开听声辨位功能。

Pitch Detection：打开音高侦测功能。

Subband-1：选择几个通道使用 SBC-1 播放，最多 2 通道。

Subband-2：选择几个通道使用 SBC-2 播放，最多 2 通道。

ADM：选择几个通道使用 ADM 播放，最多 8 通道。

ADPCM：选择几个通道使用 ADPCM 播放，最多 8 通道。

PCM：选择几个通道使用 PCM 播放，最多 3 通道。

Melody：播放 MIDI。

QIO：对应的压缩算法打开 QIO 功能。

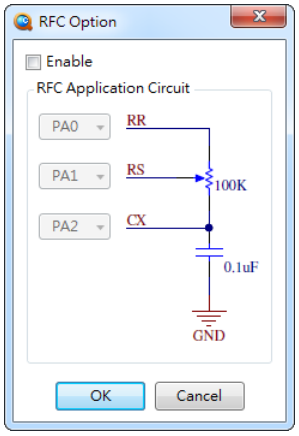
Sound Effect：对应的压缩算法打开音效功能。

Loop：对应的压缩算法或 MIDI 打开 Loop 功能。ADM / ADPCM 可以支持多个通道 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

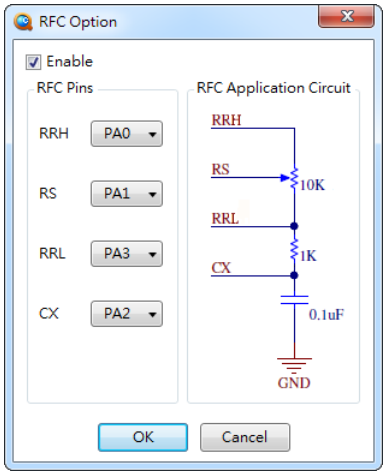
3.1.22.1 NY5+ / NY6 / NY7



RFC（Resistor to Frequency Converter）模块需外接一颗电容器及电位器，利用 I/O 对电容器充电的时间常数来侦测电位器旋钮的位置，分辨率为 16 阶，简而言之，就是电位器的旋钮会被分为 16 个区域，用户可通过 API 读取目前旋钮所在的区域来执行不同的控制，如音量大小及节奏快慢等。

注意：电容器及电位器的数值建议为 0.1uF 及 100KΩ，如用户有必要调整，需尽量维持相同的充电常数；例如，电位器数值降低为一半，则电容器数值就需增加一倍，以此类推。

3.1.22.2 NX1

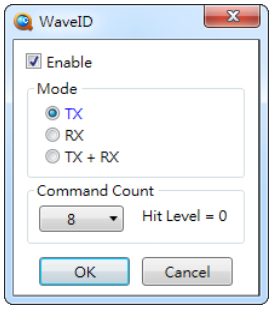


RFC（Resistor to Frequency Converter）模块需外接一颗电容器及电位器，利用 I/O 对电容器充电的时间常数来侦测电位器旋钮的位置，分辨率为 16 阶，简而言之，就是电位器的旋钮会被分为 16 个区域，用户可通过 API 读取目前旋钮所在的区域来执行不同的控制，如音量大小及节奏快慢等。

注意：电容器及电位器的数值建议为 0.1uF 及 100KΩ，如用户有必要调整，需尽量维持相同的充电常数；例如，电位器数值降低为一半，则电容器数值就需增加一倍，以此类推。

3.1.23 WaveID

3.1.23.1 NX1



Wave ID 使用喇叭传送 ID，透过麦克风接收 ID，达成 ID 的传送和接收。喇叭传送的频率为人耳不灵敏的频带(16K ~ 20KHz)，应用上可与一般语音一起由喇叭输出，搭配的语音只能由 Ch0 播放。

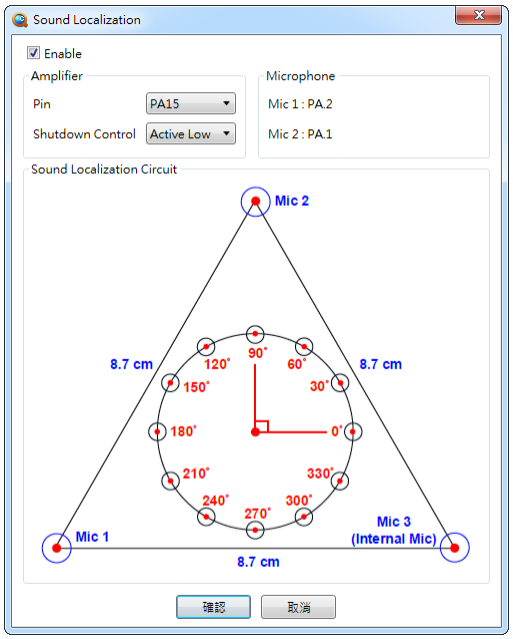
Mode：选择 TX or RX 功能。

Command Count：选择 ID 传送的数量，最多 38 个 ID。ID 越多，会降低接收的成功率。

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

3.1.24 Sound Localization

3.1.24.1 NX1 OTP

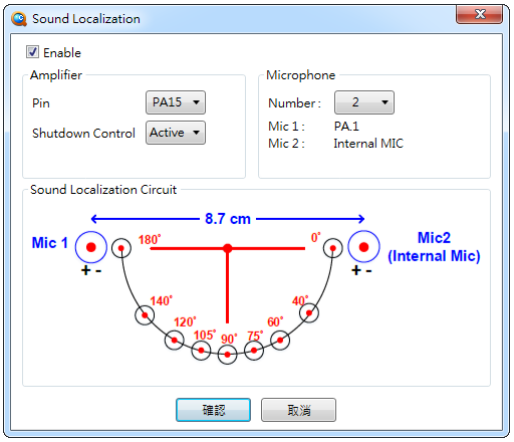


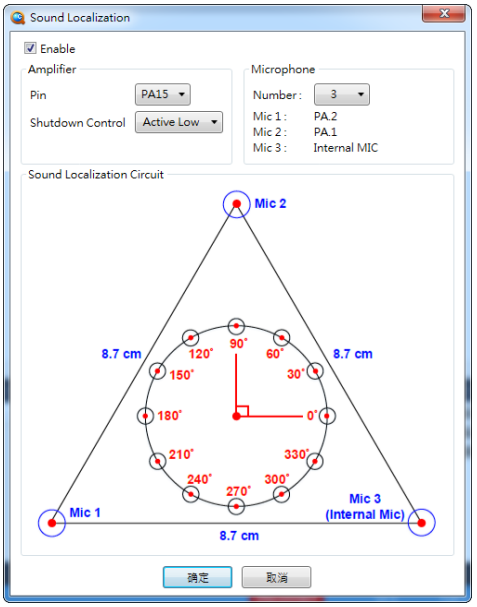
Amplifier Pin：外部放大器控制脚位。

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

3.1.24.2 NX1 EF

这功能可以选择使用 2 组或 3 组麦克风，透过外部放大器放大输入信号，再搭配内建的麦克风，可推算出声音来源的角度。





Amplifier Pin：外部放大器控制脚位。

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

Microphone Number：麦克风数量。

注意：2 组麦克风需占用 PA.1，3 组麦克风需占用 PA.1、PA.2。

例.

SoundLoc = Enable

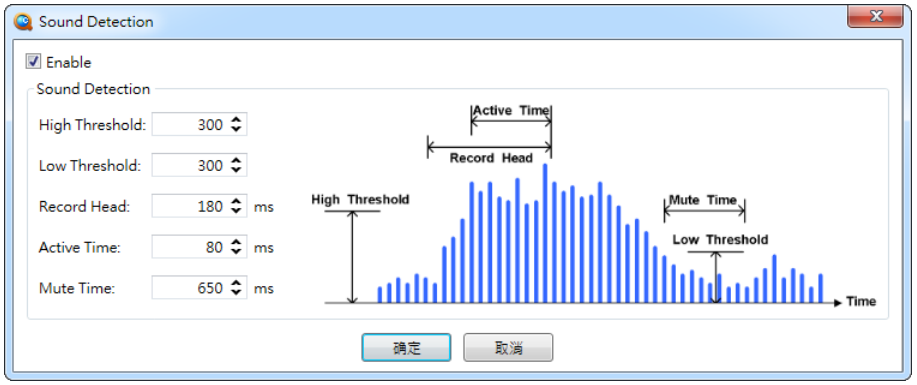
SoundLoc_Amp_Pin = PA.15

SoundLoc_Amp_Shutdown_Control = Active_Low

SoundLoc_MIC = 2

3.1.25 Sound Detection

3.1.25.1 NX1



High Threshold：侦测有语音的门坎值。高于此门坎且超过 Active Time，会被判定为有效的语音。

Low Threshold：侦测无语音的门坎值。低于此门坎且超过 Mute Time，会被判定为仅剩背景音，而停止录音。

Active Time：侦测有语音的时间。语音高于 High Threshold 且超过侦测时间，会被判定为有效的语音。

Mute Time：侦测无语音的时间。语音低于 Low Threshold 且超过侦测的时间，会被判定为仅剩背景音，而停止录音。

例.

SoundDetect = Enable

SoundDetect_High_Threshold = 300

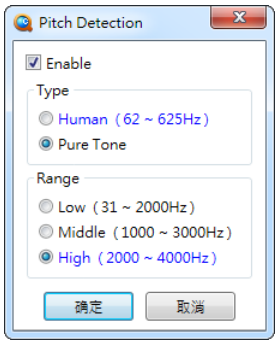
SoundDetect_Low_Threshold = 300

SoundDetect_Active_Time = 80ms

SoundDetect_Mute_Time = 650ms

3.1.26 Pitch Detection

3.1.26.1 NX1



Tyep：选择侦测人声（Human）或纯音频（Pure Tone）。

例. **PitchDetect** = Tone

Range：设定纯音频（Pure Tone）的侦测范围。

例. **PitchDetect_Range** = Low

3.1.27 Maximum Single Note

3.1.27.1 NY5+ / NY6 / NY7

用户可以选择最多可同时播放的琴键音数目，设定范围：1~8。（默认值=4）

注意：

1. ***NY5+最多四个通道，没有在播放琴键音或语音的通道才能用来播放 melody 使用。***
2. ***NY6 最多六个通道，没有在播放琴键音或语音的通道才能用来播放 melody 使用。***
3. ***NY7 最多八个通道，没有在播放琴键音或语音的通道才能用来播放 melody 使用。***

例. **Melody_MaxSingleNote** = 4 ；设定最大琴键音数目为 4 个。

3.1.28 Melody_OKON_BgNote

3.1.28.1 NY5+ / NY6 / NY7

当使用 One-Key-One-Note 功能时，默认仅会播放 MIDI 通道 1。用户可以打开此选项，一并播放其它通道。

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

3.1.29 Variable Compatible

3.1.29.1 NX1

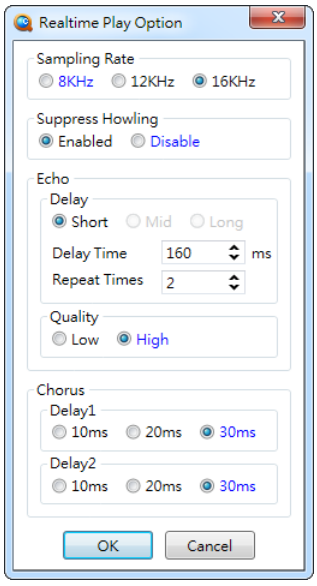
变量兼容性提供了 NX1 系列对传统 4bit 变量运算的兼容性，如果您在 NX1 系列仍然需要使用 4 位的数据型态，请设定在 NX1 系列，变量最小长度的变量是 8 位。如果需要 4 位的变量，请设定 Variable_Compatible。*Q-Code* 将会产生 R0、R1、X0、X0H、X0L 之类的变量，让数学运算的指令与 NY4 / 5 / 6 / 7 / 9T 一致。

例. **Variable_Compatible** = Enable ；打开变量运算兼容性。

3.1.30 Realtime Play

3.1.30.1 NX1

实时播放(大声公)功能。



Sampling Rate：选择采样率。

例. **RT_Sampling_Rate** = 8000

Suppress Howling：嚎叫声抑制。

例. **RT_SuppressHowling** = Enable

Echo Delay Time：回声的长度。

例. **RT_ECHO_DelayTime** = 200ms

Echo Repeat Times：回声的重复次数。

例. **RT_ECHO_Repeat** = 2

Echo Quality：回声的质量。

例. **RT_ECHO_Quality** = High

Chorus Delay1：合音频道 1 与频道 2 之间的延迟时间，支持 10/20/30ms。

例. **RT_Chorus_Delay1** = 30ms

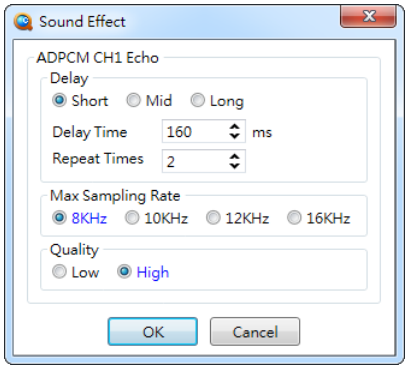
Chorus Delay2：合音频道 2 与频道 3 之间的延迟时间，支持 10/20/30ms。

例. **RT_Chorus_Delay2** = 30ms

3.1.31 Sound Effect

3.1.31.1 NX1

音效设定。



ADPCM CH1 Echo Delay Time：使用者自定义 ADPCM CH1 回声的长度。

例. **SFX_ADPCM_Echo_DelayTime** = 200ms

ADPCM CH1 Echo Repeat Times：使用者自定义 ADPCM CH1 回声的重复次数。

例. **SFX_ADPCM_Echo_Repeat** = 4

ADPCM CH1 Echo Max Sampling Rate：选择 ADPCM CH1 回声的最大采样率。

例. **SFX_ADPCM_Echo_Max_SR** = 8000

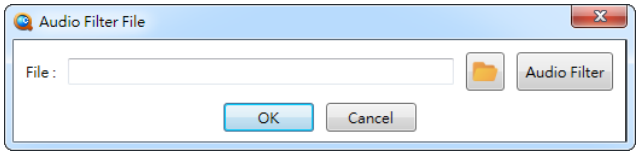
ADPCM CH1 Echo Advance Quality：选择高质量的 ADPCM CH1 回声。

例. **SFX_ADPCM_Echo_Quality** = High

3.1.32 Audio Filter

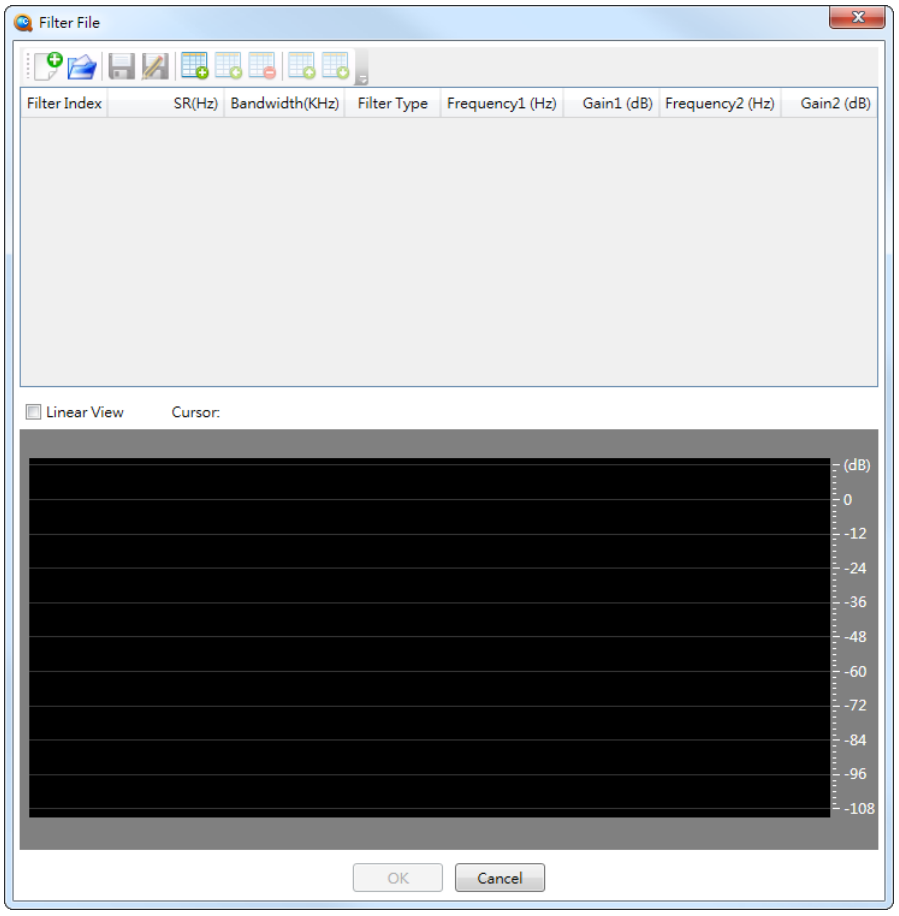
3.1.32.1 NX1 OTP / NX1 EF

设定画面如下图所示，可直接选择已编辑好的.fnx 文件，或是呼叫音频滤波器编辑窗口来产生.fnx 文件。



点击上图音频滤波器按钮会跳出编辑窗口画面如下图所示。可设定多组的滤波器，每组滤波器可设定取样频率、带宽以及滤波器类型，目前支持低通滤波器(LPF)、高通滤波器(HPF)、带通滤波器(BPF)以及均衡器(EQ)四种类型。选择类型后，再输入所需过滤的频率或频率范围与增益值即完成设定。设定完成可将此设定保存为扩展名为.fnx 的文件，方便之后重复使用。

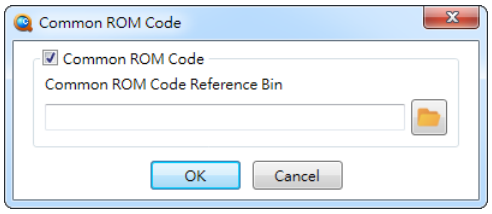
例. **Custom_Filter** = Filter.fnx



3.1.33 Common ROM Code

3.1.33.1 NX1

ROM 程序共享功能用于主要功能相同，仅需替换资源内容的应用。替换的内容放置于 SPI Flash 中，只需替换 SPI Flash 就可以有不同的效果。为了维持主要 ROM 内容一致，会弹出以下画面进行设定。



ROM 程序共享参考 bin 文件（Common ROM Code Reference Bin）则是选择之前建立的 bin 文件，*Q-Code* 会自动对目前与选择 bin 文件中的 ROM 数据进行比对，以确保 ROM 的数据完全一致。

基本开发 ROM 程序共享应用流程如下：

1. 首次开发时开启 ROM 程序共享功能。
2. 后续开发的程序，需开启 ROM 程序共享功能，并选择首次开发产生的 bin 档作为比对文件。

按照以上的操作，*Q-Code* 会检查产出的 ROM 数据皆为一致，否则会发出错误。

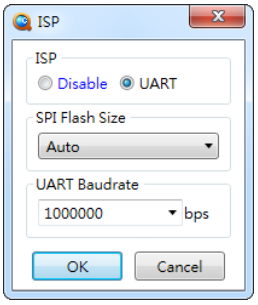
例.

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



设置 ISP 的通信类型。目前仅支持 UART，而 UART 通信内建使用 PD.0/1 这组脚位，更详细的 ISP 操作与说明，请参考 NYISP 使用手册目录。

例.

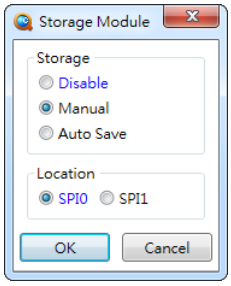
ISP = UART

ISP_UART_Baundrate = 1000000bps

ISP_FlashSize = 256Mbit

3.1.35 Storage Modual

3.1.35.1 NX1 OTP



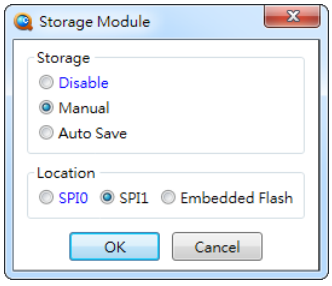
Storage：设置储存的模式。

例. **Storage** = Disable

Locaiton：设置储存的位置。

例. **Storage_Location** = SPI0

3.1.35.2 NX1 EF



Storage：设置储存的模式。

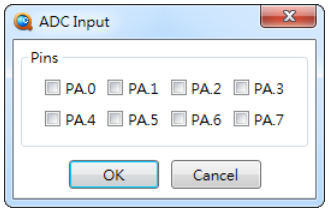
例. **Storage** = Manual

Locaiton：设置储存的位置。

例. **Storage_Location** = EF

3.1.36 ADC Input

3.1.36.1 NX1



设定模拟数字转换器的模拟信号输入脚。

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

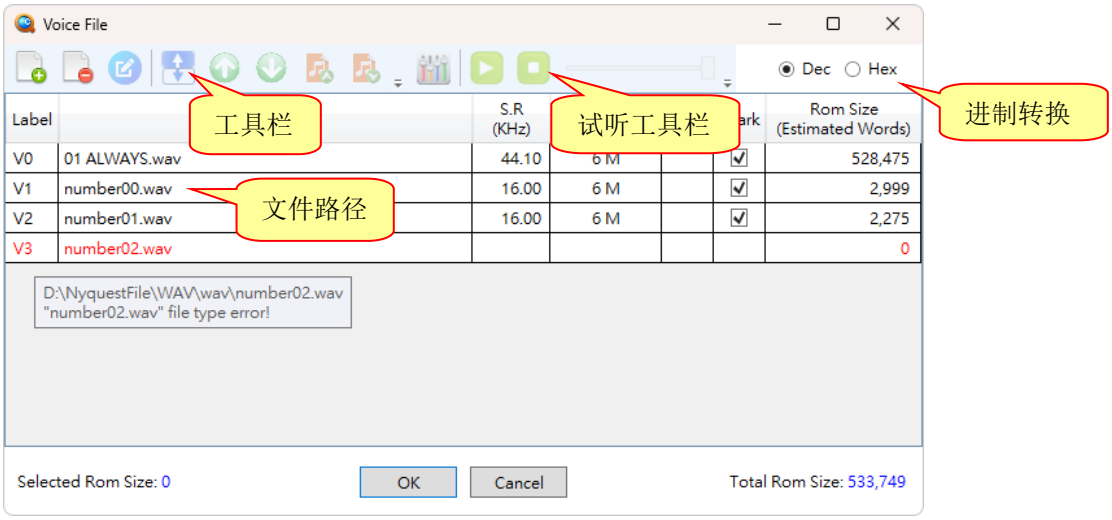
3.2 Voice File

播放用的声音文件，一般扩展名为.wav，也可以是经过编码后的.v4x / .v5x / .v5px / .v6x / .v7x / .vnx 文件，或是通过 *Q-Sound* 产生的.nyw 文件。若是有 *Quick-I/O* 功能（QIO 信号或 WaveMark 信号）的扩展名为.nyq。如果加进来的是.nyq 文件，将会在播放该文件时，启动 QIO 或 Wave Mark 功能。

单击 Voice File 段落菜单中的 Add File，打开 Voice File，可以向程序中插入 Voice 音源。如果想要删除一个或多个音源，则只需要先选中您想要删除的音源文件，点击 Delete Files 就可移除不需要的音源。

注意：Voice Label 用户可以自行定义。

点击 Add File 后会出现 Voice File 的对话框，可在 Voice File 的对话框中加入所需的 Voice 音源。不同系列的 Voice File 对话框会显示不同的栏位。



添加文件：新增音源档。

删除文件：删除选取的文件。

重命名工具：快速重命名多个 Label 的工具。

反相：将目前所选取的文件取消选取，并选取其余未选取的文件。

向上移动：选取的项目向上移动。

向下移动：选取的项目向下移动。

向上移动文件：向上移动选取的文件。

向下移动文件：向下移动选取的文件。

Q-Sound：是针对音源档切割而开发的软件工具。它提供简易的图形处理界面来达成切割文件的功能。使用时仅需在 Q-Sound 完成标记编辑后储存。

试听钮：可试听所选择的音源档，拖曳 indicator 可控制播放进度。

十进制 / 十六进制：将 ROM Size 的数值切换成十进制或十六进制。

Selected ROM size：选取的音源档预估 ROM Size 的总和。

Total ROM Size：所有音源档预估 ROM Size 的总和。

注意：

1. Q-Code 内建的 Q-Sound 功能，支持外部 Q-Sound 切好的.nyw 文件。

2. *所有插入的标记（Mark）须在保存后才会记录。*
3. *若用户添加的是 .v5px / .v6x / .v7x / .vnx 文件，或是 Voice_Encoder 2.20 或更新版本所编码的.v4x / .v5x，程序会自动使用原始音源的采样率。例如：PlayV(Ch0, $V0)。*
4. *若是用 Voice_Encoder 2.20 之前的版本编码的.v4x / .v5x，则需要手动写上原始音源的采样率。例如：PlayV(Ch0, $V0, 6K)。*

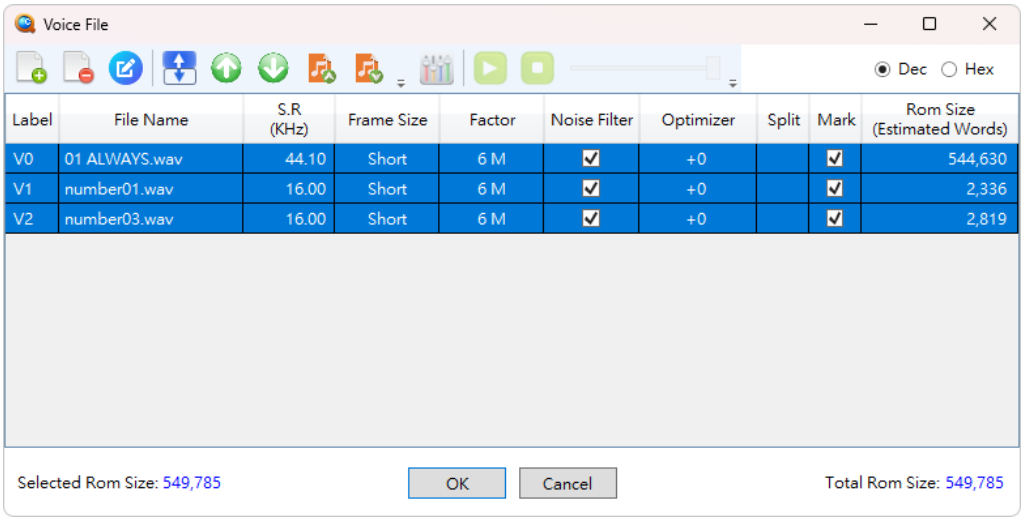
例.

[Voice File]

V0 = C:\MyQ-CodeProjects\Prj1\voice1.wav /6

V1 = Prj1\voice12.wav /PCM

3.2.1 NY4



S.R（Sample Rate）：显示加入的音源文件的采样率，采样率越高所占的 ROM Size 越大。

Frame Size：提供两种不同的 Frame Size，分别是 Short 与 Long。

1. Short Frame Size：音质较佳，但是占 ROM 较多的声音压缩方式。
2. Long Frame Size：音质一般，但是占 ROM 较少的声音压缩方式。

Factor：意指语音压缩率。提供 12 种不同的压缩率，其数字越大，则代表压缩比越低，所占的 ROM Size 也越高，则语音质量也随之提高。另外，还提供 PCM 模式，则是对语音不采取压缩的方式，其所占的 ROM Size 最大，声音质量也是最好。（声音的质量与压缩率乃是成正比，默认值为 6）

Noise Filter ：提供噪声滤波功能（降噪）。在某些情况下，声音的输出听起来会有细微的背景噪音混杂在音源里面，所以我们会使用 Noise Filter 来将杂音滤掉。

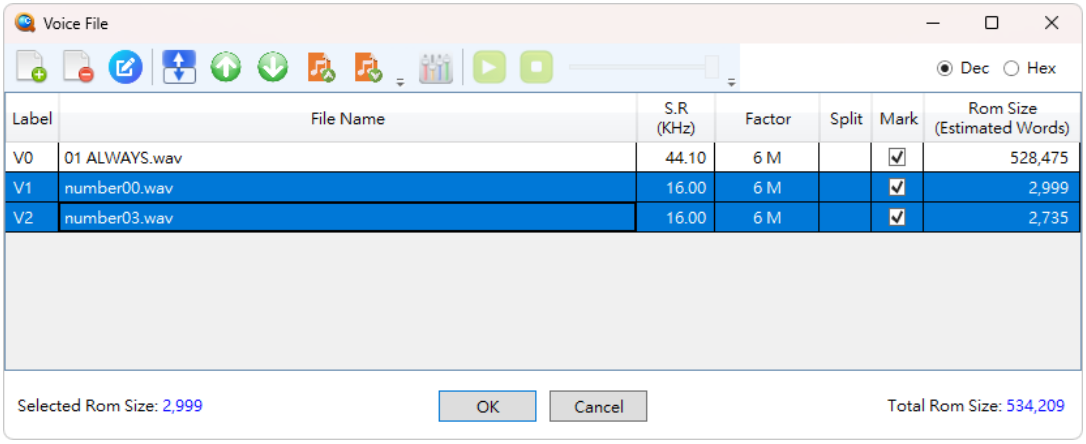
Optimizer：将声音依设定值进行优化。声音优化功能可对音乐信号进行音色调整，以增强对比度（sharp），或者让声音更柔和（smooth）。

Split：语音档切割功能。点击可针对单一文件开启或关闭。

Mark：开启.wav 文件的 Wave Mark 功能。

Estimated ROM Size：语音档被加入后，在 IC 中所占的 ROM Size。

3.2.2 NY5 / NY5+



S.R（Sample Rate）：显示加入的音源文件的采样率，采样率越高所占的 ROM Size 越大。

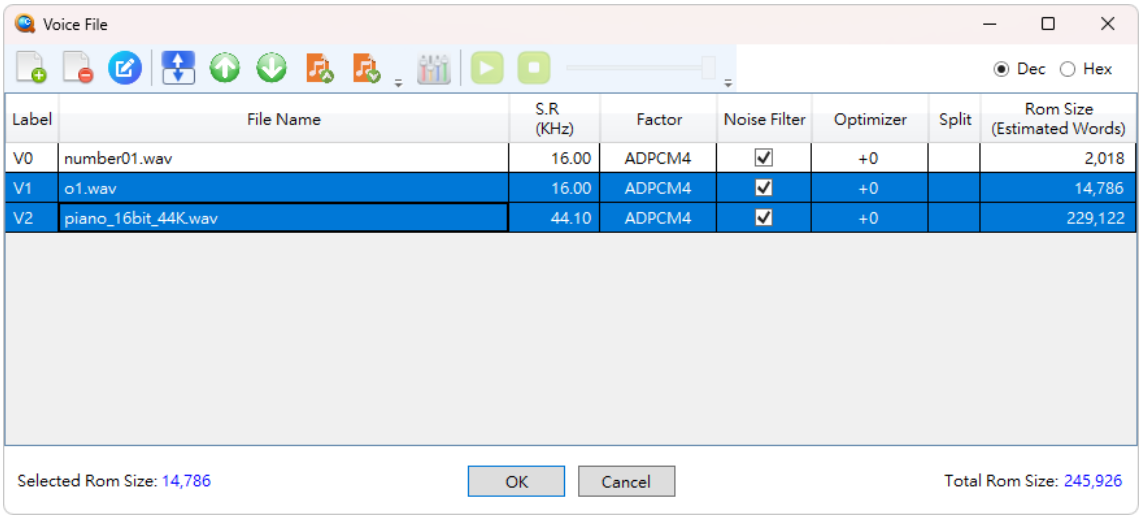
Factor：意指语音压缩率。提供 12 种不同的压缩率，其数字越大，则代表压缩比越低，所占的 ROM Size 也越高，则语音质量也随之提高。另外，还提供 PCM 模式，则是对语音不采取压缩的方式，其所占的 ROM Size 最大，声音质量也是最好。（声音的质量与压缩率乃是成正比，默认值为 6）

Split：语音档切割功能。点击可针对单一文件开启或关闭。

Mark：开启.wav 文件的 Wave Mark 功能。

Estimated ROM Size：语音档被加入后，在 IC 中所占的 ROM Size。

3.2.3 NY6



S.R（Sample Rate）：显示加入的音源文件的采样率，采样率越高所占的 ROM Size 越大。

Factor：意指语音压缩率。默认使用 ADPCM 模式。另外，还提供 ADPCM5 / PCM 模式。PCM 采取不压缩的方式，其所占的 ROM Size 最大，声音质量也是最好。

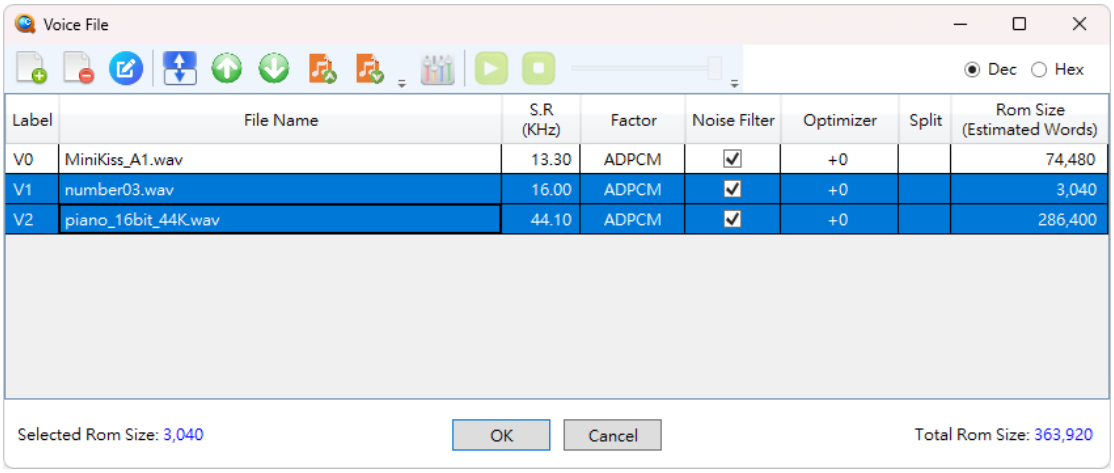
Noise Filter：提供噪声滤波功能（降噪）。在某些情况下，声音的输出听起来会有细微的背景噪音混杂在音源里面，所以我们会使用 Noise Filter 来将杂音滤掉。

Optimizer：将声音依设定值进行优化。声音优化功能可对音乐信号进行音色调整，以增强对比度（sharp），或者让声音更柔和（smooth）。

Split：语音档切割功能。点击可针对单一文件开启或关闭。

Estimated ROM Size：语音档被加入后，在 IC 中所占的 ROM Size。

3.2.4 NY7



S.R（Sample Rate）：显示加入的音源文件的采样率，采样率越高所占的 ROM Size 越大。

Factor：意指语音压缩率。默认使用 ADPCM 模式。另外，还提供 PCM 模式，则是对语音不采取压缩的方式，其所占的 ROM Size 最大，声音质量也是最好。

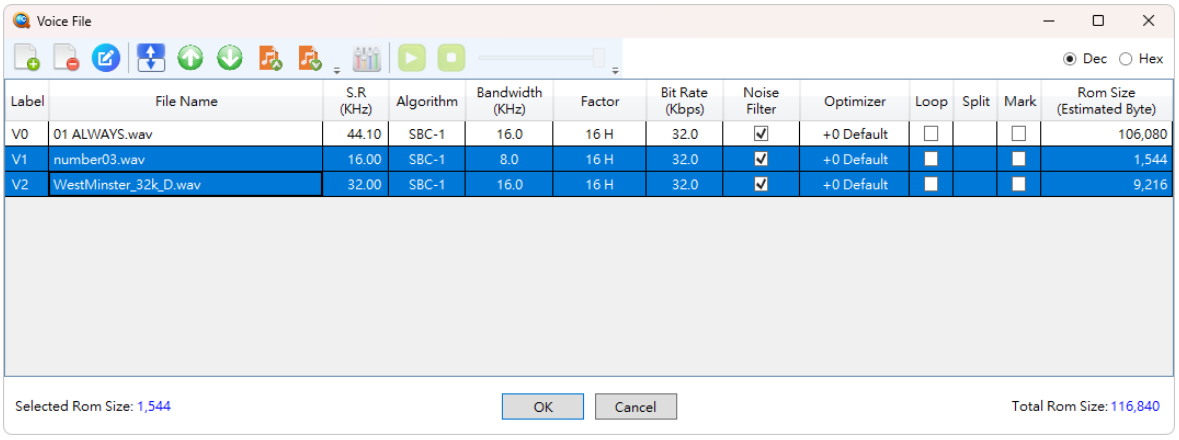
Noise Filter：提供噪声滤波功能（降噪）。在某些情况下，声音的输出听起来会有细微的背景噪音混杂在音源里面，所以我们会使用 Noise Filter 来将杂音滤掉。

Optimizer：将声音依设定值进行优化。声音优化功能可对音乐信号进行音色调整，以增强对比度（sharp），或者让声音更柔和（smooth）。

Split：语音档切割功能。点击可针对单一文件开启或关闭。

Estimated ROM Size：语音档被加入后，在 IC 中所占的 ROM Size。

3.2.5 NX1



S.R（Sample Rate）：显示加入的音源文件的采样率，采样率越高所占的 ROM Size 越大。

Algorithm：除了常用的 PCM 和 ADPCM 格式以外，NX1 也提供更高压缩率的 SBC-1 / SBC-2 / CELP 格式供客户选择。子带编码（Sub-Band Coding）是一种以信号频谱为依据的波形编码方法，

它首先用一组带通滤波器将输入信号按频谱分开，然后让每路子信号通过各自的自适应 PCM 编码器（ADPCM）编码，经过分接和译码再复合成原始信号。经由高效能的 NX1 以软件的方式来实现 Sub-Band Coding，能够达到更低的 Bit Rate 来节省内存的需求量，而且以较高的采样率、更宽的频带来重现高频的音域表现，尤其是高音域的器乐更是如此。SBC-2 比 SBC-1 降低了 ROM 空间、RAM 空间和 CPU 负载，但 SBC-2 音质会比 SBC-1 差一些。CELP 则是针对人声所特定发展出来的超低 Bit Rate 压缩算法，不过，CELP 并不适用于非人声的应用。

Bandwidth：单位为 KHz。

Factor：使用动态压缩的算法，依据选择的算法有不同的压缩率供选择，SBC-1 / SBC-2 共有 19 阶参数（-2~16）；ADPCM4 和 ADPCM5 共有 12 阶参数（1~12）；CELP 为固定压缩率。客户可以根据产品的需求，选定合适的算法，再依声音质量的要求，调整 Factor 的高低来达到优化内存的使用。

Bit Rate：单位时间播放压缩后语音的位数量，它相当于语音播放时的带宽消耗量。可以想象的是，较高的比特率可容纳更高的语音质量，单位为 Kbps。

Noise Filter：提供噪声滤波功能（降噪）。在某些情况下，声音的输出听起来会有细微的背景噪音混杂在音源里面，所以我们会使用 Noise Filter 来将杂音滤掉。

Optimizer：将声音依设定值进行优化。声音优化功能可对音乐信号进行音色调整，以增强对比度（sharp），或者让声音更柔和（smooth）。

Loop：选择是否开启循环播放功能，当开启功能时将会耗用较多的 ROM Size。

Split：语音档切割功能。点击可针对单一文件开启或关闭。

Mark (NY4 / NY5 / NY5+ / NX1) ：开启.wav 文件的 Wave Mark 功能。

Estimated ROM Size：语音档被加入后，在 IC 中所占的 ROM Size。

3.2.6 *Q*-Code 如何播放 *Q-Sound* 处理后的文件

Q-Code 提供简易的方式，让用户可以播放 *Q-Sound* 处理后的文件。

第一步：在 Voice File 对话框中，先将要播放的 Voice File 加入，加入后将 Split 功能打开。

第二步：开启 *Q-Sound* 并设定好要切开的音源。

第三步：在 *Q-Code* 中下达 PlayV 指令。

例.

[Voice File]

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

;在 Voice 档后面带了 /s 参数，表示有使用 Split 功能。

[Path]

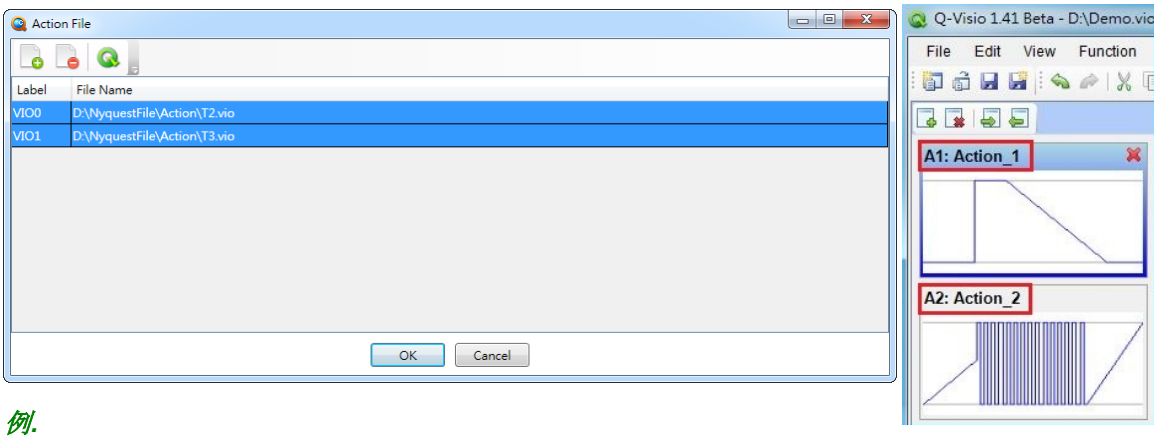
PowerOn: PlayV(Ch1, $V01)

;要播放 Label “$V01”音档，仅需要下 PlayV 指令，系统会自动将切完的文件展开，用户不需要另外处理。

3.3 Action File

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

播放用的信号文件，一般信号文件的扩展名为.vio。关于 Action 的编辑，请参考 *Q-Visio* 用户手册。



例.

[Action File]

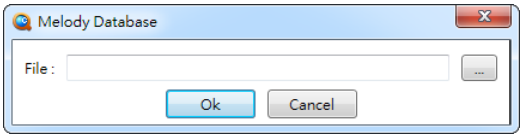
VIO0 = C:\Project.vio ； 加入 **Action** 文件。

[Path]

PowerOn: PlayA(PB.0, CH1, $A1) ；上电后立即播放 **Action** 文件，并将 **A1** 的信号输出至 **PB.0**。

3.4 Melody Database

播放 melody 时，需要音乐数据库，其中包含音色、包络、以及乐曲等信息。使用 *Q-Code* 写程序时需添加这个.qmd 文件来调音乐数据库中的数据，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

使用的音乐数据库为 *Q-MIDI* 产生的.md2 文件。

注意：用户需要通过 Q-MIDI 自行制作音色库，产生一个.idb 文件，之后用 Q-MIDI 将 melody 会用到的音色与音色库联结并产生一个.md2 音色文件，即可在 Q-Code 程序中添加这个音色文件（详情请见 Q-MIDI 用户手册）。

3.4.2 NY5+ / NY6 / NY7 / NX1

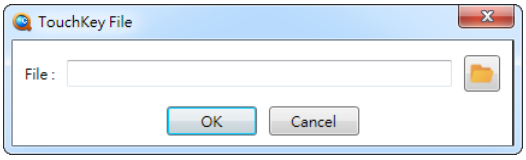
使用的音乐数据库为 *Q-MIDI* 产生的.qmd 文件。

注意：用户需要通过 Q-MIDI 的 Instrument Database 页面自行制作音色库，产生一个.idb 档，之后用 Q-MIDI 的 Music 页面添加需使用的.mid 档，Q-MIDI 会用到的音色与音色库联结并产生一个.qmd 音乐数据档，即可在 Q-Code 程序中添加这个音乐数据档（详情请见 Q-MIDI 用户手册）。

3.5 TouchKey File

3.5.1 NY9T

触摸键灵敏度配置文件，可替换文件来对应不同的灵敏度。扩展名为.t9x。



例.

[TouchKey File]

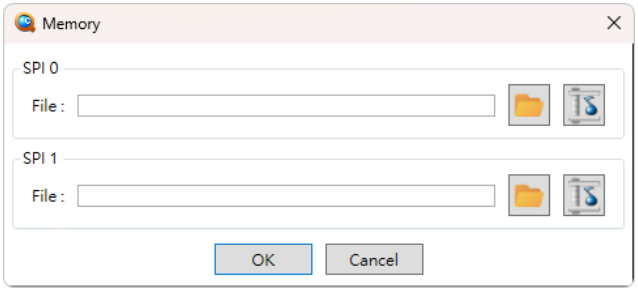
; TouchKey Number = 1

; Scane Mode = Preset。

C:\Normal.t9x

3.6 Memory

3.6.1 NX1 OTP



可将资源放在 SPI Flash 上，支持 SPI0 / SPI1。

将资源放在 SPI Flash，需加入 *SPI_Encoder* 使用的项目档，扩展名为.spiprj，建置后会产生 _SPI.bin 檔，可以经由 *Q-Writer* 或是 *Q-Code* 下载。

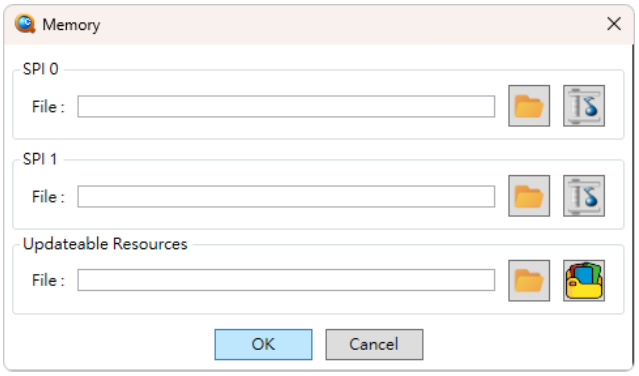
例.

[Memory]

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

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

3.6.2 NX1 EF



可将资源放在 SPI Flash 上，可支持 SPI0 / SPI1。另外支持 Updateable Resources。

将资源放在 SPI Flash，需加入 *SPI_Encoder* 使用的项目档，扩展名为.spiprj，建置后会产生 _SPI.bin 檔，可以经由 *Q-Writer* 或是 *Q-Code* 下载。

Updateable Resources 可按下图所示按纽编辑内容，并存为.udrprj 档案。

例.

[Memory]

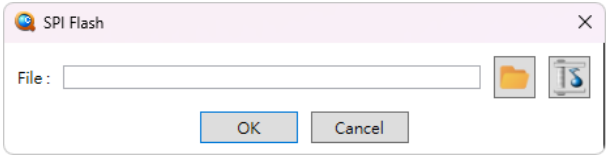
SPI0_File = C:\Users\Desktop\Untitled.spiprj

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

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

3.7 SPI Flash

3.7.1 NY6



可以加入 *SPI_Encoder* 使用的项目档，扩展名为.spiprj，建置后会产生 _SPI.bin 档，可以经由 *Q-Writer* 或是 *Q-Code* 下载。

关于 *SPI_Encoder* 的编辑，请参考 *SPI_Encoder* 使用手册。

例.

[SPI Flash]

C:\Users\Desktop\Untitled.spiprj

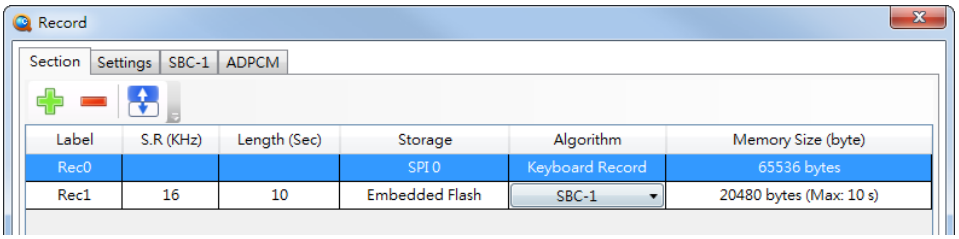
3.8 Record

3.8.1 NX1

NX1 提供了简单的方式进行录音。当使用 SPI Flash 做为录音区段的储存空间时，会将录音的结果写至 SPI Flash 上。为了进行录音，SPI Flash 上需要保留空间。所以在 Build 后会产生_SPI.bin。

3.8.1.1 Section

段落中亦需要定义录音的区段。每次进行录音或擦除时都是以区段为单位。



Length：语音录音的时间长度，区段的大小由所选定的录音算法与录音的时间长度所决定。而琴键录音区段的大小固定为 64kB。

Storage：录音区段的储存空间，可支持 SPI0 / SPI1 / Embedded Flash，琴键录音只支持 SPI0。

Algorithm：SBC-1 / ADPCM 是语音录音，Keyboard Record 则是做为琴键录音使用。

Memory Size：实际占用的 Size。

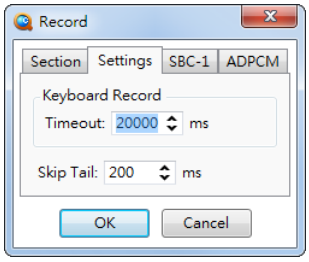
例.

[Record]

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

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

3.8.1.2 Seetings



Skip Tail：会删除录音的结尾，可以避免将结束的按键音一并录进去。

Keyboard Record Timeout: 当使用琴键录音功能时，设定多久未收到 InstNoteOn / InstNoteOff 即结束录音功能。

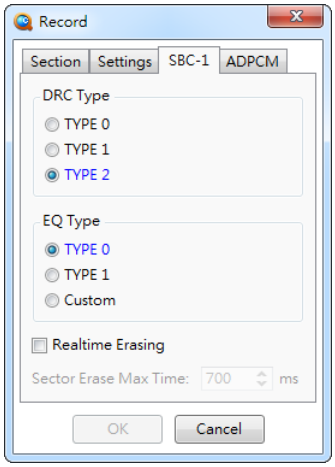
例.

[Record]

KRecord_Timeout = 20000ms

Skip_Tail = 200ms

3.8.1.3 SBC-1



DRC Type：针对录音播放，TYPE 0 不做任何处理，可以减少资源消耗。TYPE 1 会减少最大声与最小声之间的差距，使声音更加厚实、宏亮。TYPE 2 仅放大小音量部份，使小音量变更清楚。（默认值 TYPE 2）

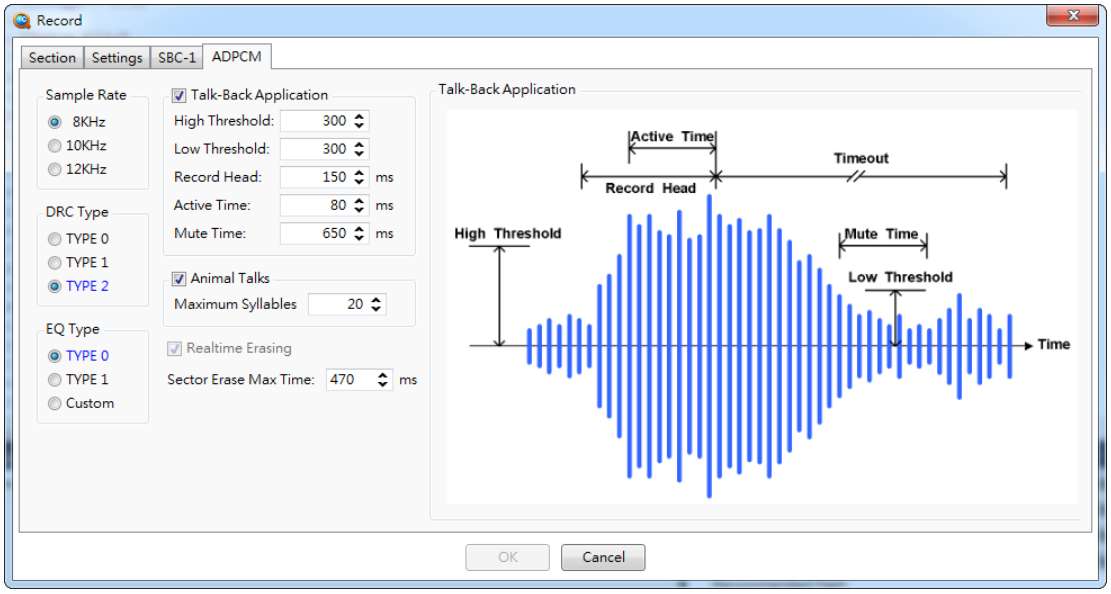
EQ Type：针对录音播放套用 EQ 效果。TYPE 0 会锐化音色特征，使声音较为明亮，适合用于乐曲类型；TYPE 1 加强低音，使声音较为厚实，使人声更为突出；Customize 则使用音频滤波器（Audio Filter）进行音信处理。（默认值 TYPE 0）

Realtime Erasing：是否开启随擦即录功能。未开启时，使用者要自行在录音前使用 EraseR / EraseRS 等指令将要录音的区段进行擦除。开启后 Q-Code 会在录音前视需要进行录音区段擦除的动作。开启此功能会耗费较多的 RAM。

Sector Erase Max Time：使用随擦即录功能时，于擦除阶段发生 Flash 等待逾时将停止等待并立刻返回，建议依照 Flash 规格进行合适的时间设定。

注意：SBC-1 录音仅支援 16k sample rate。

3.8.1.4 ADPCM



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

DRC Type：同 SBC-1 同名设定。

EQ Type：同 SBC-1 同名设定。

Talk-Back Application：开启自动应答功能。

High Threshold：侦测有语音的门坎值。高于此门坎且超过 Active Time，会被判定为有效的语音。

Low Threshold：侦测无语音的门坎值。低于此门坎且超过 Mute Time，会被判定为仅剩背景音，而停止录音。

Record Head：录音文件最前面的语音。当有效的语音被侦测，实际录音的起始点，会再往前 Record Head 的时间。

Active Time：侦测有语音的时间。语音高于 High Threshold 且超过侦测时间，会被判定为有效的语音。

Mute Time：侦测无语音的时间。语音低于 Low Threshold 且超过侦测的时间，会被判定为仅剩背景音，而停止录音。

Animal Talks：启用动物音。

Maximum Syllable：最大音节数。

Realtime Erasing：同 SBC-1 同名设定。

Sector Erase Max Time：同 SBC-1 同名设定。

注意：

1. ***录音功能需要搭配 SPI_Encoder 1.53 或更新的版本。***
2. ***录音不能与播音(Voice / MIDI)同时。***
3. ***当 Q-Code 正在处理 SPI Flash 的擦除动作时，XIP 的程序会暂停大约 30 ~ 100ms(开启随擦即录) / 100 ~ 200ms (未开启随擦即录)左右 (依 SPI Flash 而定)。***
4. ***当擦除正在进行中，无法对 SPI Flash 进行操作。***
5. ***播放 SPI Flash 上的内容 (voice / melody / action …)时，Q-Code 处理 SPI Flash 的擦除动作会造成播放不正常。***
6. ***开启随擦即录功能时，无法使用琴键录音功能。***

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

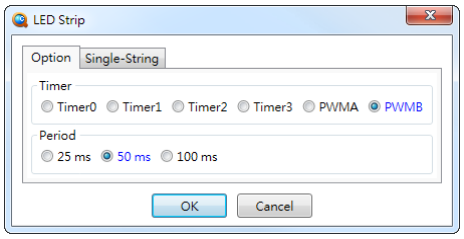
3.9 LED Strip

LED Strip 可透过输出脚位控制多颗 RGB LED 所组成的灯串，灯串数据由 Vixen 产生，扩展名为.csv，被控制的 RGB LED 须支持特定的通信格式。

注意：LED Strip 的最低 Reset Time 需大于 60us。

3.9.1 Option

3.9.1.1 NX1

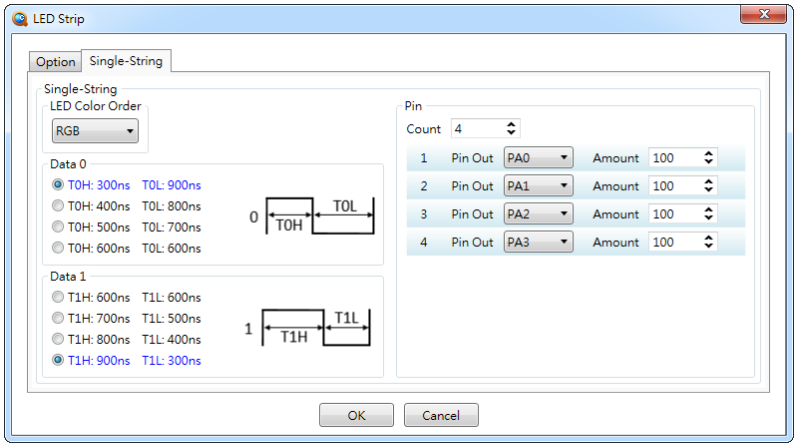


Timer：选择定时器来源，需避免其他功能已使用的定时器。

Period：选择灯串 LED 的更新周期。

3.9.2 Single-String

3.9.2.1 NX1



LED Color Order：灯串上 LED 的 RGB 排列顺序。

LED Data 0：选择灯串 Data 0 的通信格式。

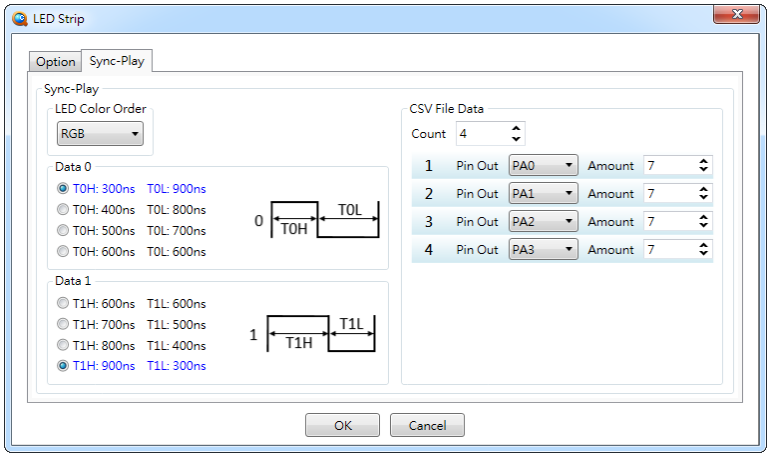
LDE Data 1：选择灯串 Data 1 的通信格式。

Pin：设定要使用 Single-String 功能的脚位，以及该脚位外接的灯串上的 LED 数量。所有脚位输出的 LED 数量总数，当 Period 为 25ms 最多支援 399 颗，Period 为 50ms 最多支持 799 颗，Period 为 100ms 最多支持 1599 颗，另外，输出脚位数量也会影响最大 LED 数量。

注意：LED 灯串的最低 Reset Time 需大于 60us。

3.9.3 Sync-Play

3.9.3.1 NX1



LED Color Order：灯串上 LED 的 RGB 排列顺序。

LED Data 0：选择灯串 Data 0 的通信格式。

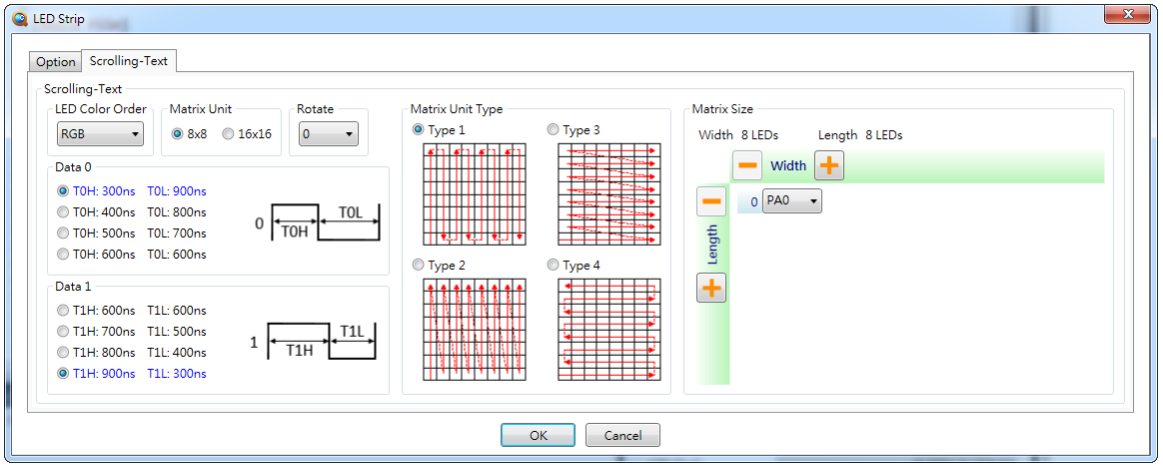
LDE Data 1：选择灯串 Data 1 的通信格式。

CSV File Data：设定要 CSV File 使用的脚位，以及该脚位灯串的 LED 数量。每个脚位最多支援 300 颗 LED 且必须使用同一个 Port 的脚位。

注意：LED 灯串的最低 Reset Time 需大于 60us。

3.9.4 Scrolling-Text

3.9.4.1 NX1



LED Color Order：灯串上 LED 的 RGB 排列顺序。

Matrix Unit：选择矩阵单元。

Rotate：矩阵单元的旋转角度。

LED Data 0：选择灯串 Data 0 的通信格式。

LDE Data 1：选择灯串 Data 1 的通信格式。

Matrxi Unit type：选择矩阵单元灯串排列的类型。

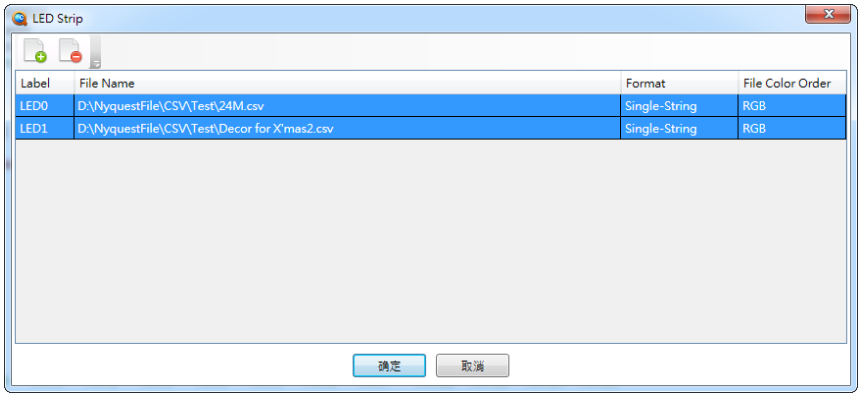
Matrix Size：选择矩阵的长度和宽度，并设定每个矩阵单元数据输出的脚位。最大的宽度为 256 个 LED 数量，最大的长度为 256 个 LED。必须使用同一个 Port 且连续的脚位。

注意：LED 灯串的最低 Reset Time 需大于 60us。

3.9.5 Add File

3.9.5.1 NX1

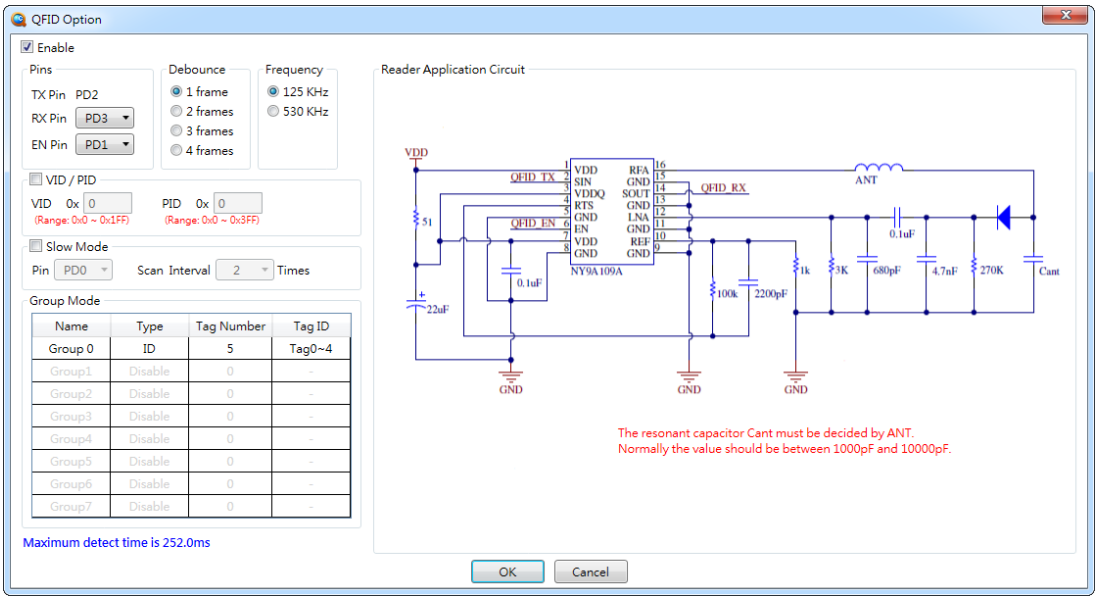
加入 .csv 和 .ledstr 文件，File Color Order 指定文件的 RGB 排列顺序。



3.10 QFID

QFID 为九齐为了 RFID 应用所推出的功能，搭配九齐所提供的应用电路以及对应的 Tag 使用，即可进行 RFID 相关应用的开发。

3.10.1 NY5+ / NY7 / NX1



TX Pin：用来产生 QFID 载波使用。

- NY5+可选用 PA.0 / PB.0 / PC.0 / PD.0 / PE.0。
- NY7A 固定为 PB.2，NY7B 固定为 PD.2，NY7C 固定为 PF.2。

- NX1 OTP 固定为 PA.5。
- NX1 EF 可选用 PA.1 / PA.2 / PC.0 / PC.1。

RX Pin：接收 tag 回传数据使用，用户仅需将指定的 I/O 连接到 NY9A109A 的 SOUT 脚位即可。

En Pin：用来开关 NY9A109A 的功能，*Q-Code* 在扫描时会自动控制，用户仅需将指定的 I/O 连接到 NY9A109A 的 EN 脚位即可。

Debounce：QFID 扫描时的 debounce 时间。NY5+ / NY7 可设为 1 / 2 / 3 / 4 frames，一并套用在 tag 辨识及移除。NX1 可分别设定辨识及移除时的 debounce 时间，支持 1 ~ 50ms。

Frequency：选择 QFID 的载波频率，可选为 125KHz 或 530KHz。当选择 530KHz 时，虽然需外加一颗 NY8A051D，但 Tag 可选择用 PCB 天线制作，并在项目建置后，*Q-Code* 自动产生额外的 CarrierGen-530KHz.bin 供烧录 NY8A051D 使用。

Timer：NX1 可选择 QFID 功能要使用的系统计数器。

Slow Pin：QFID 的低速扫描的休息时间是利用指定 I/O 的 RC 充放电来计数，使用低速扫描前，须在指定 I/O 上并接一颗 100KΩ 电阻及 0.1uF 电容。

Scan Interval：QFID 的低速扫描会以扫描一次休息数次的方式动作，休息的次数由 Scan Interval 的选项决定；例如，Scan Interval 为 2 则表示扫描一次，休息两次，假设扫描一次的时间为 200ms，则休息时间则约为 400ms，而扫描时间由设定的 Tag 数量决定，用户亦可变更电阻及电容值以调整 Interval 单位时间，变更后单位时间计算公式如下。

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

注意：休息时间为大约值，可能会有些许误差。

QFID_GroupX_Mode：QFID 模式，ID 为仅辨识 Tag，ID+Input 为辨识 Tag 加上读取 tag 上的 I/O 状态。

QFID_GroupX_Tags：可使用的 tag 数量。QFID_GroupX_Mode 为 ID 时可使用 1~16，对应的 tag 为 Tag0 ~ Tag15。QFID_GroupX_Mode 为 ID+Input 时可使用 1~8，对应的 tag 为 Tag0 ~ Tag7。

QFID_VID：设定产品的 Vender ID，范围为 0x0~0x1FF，如果 Tag 的 Vender ID 与 reader 的设定不同则该 tag 会无法辨识。

QFID_PID：设定产品的 Project ID，范围为 0x0~0x3FF，如果 Tag 的 Project ID 与 reader 的设定不同则该 tag 会无法辨识。

QFID_VID / QFID_PID 是用来避免不同产品间的 Tag 互相干扰，tag 与 reader 设定的 VID / PID 必须完全相同才能辨识，用户也可以选择关闭 VID / PID 功能。

注意：

1. NX1 不支持 Slow Pin 与 Scankey Interval 选项。

2. NY5+ / NY7 仅支持一组 QFID Group。

[QFID] 段落中亦需定义相对应的触发状态，NY7 只支持一组状态。Tag 触发型态是指当相对应的 Tag 被 reader 辨识到或从 reader 移除时所应该作出的反应，触发的数目不可超过 QFID_Group0_Tags 所设定的数量，且触发事项之间必须以空格（TAB 或 Space）来分开。

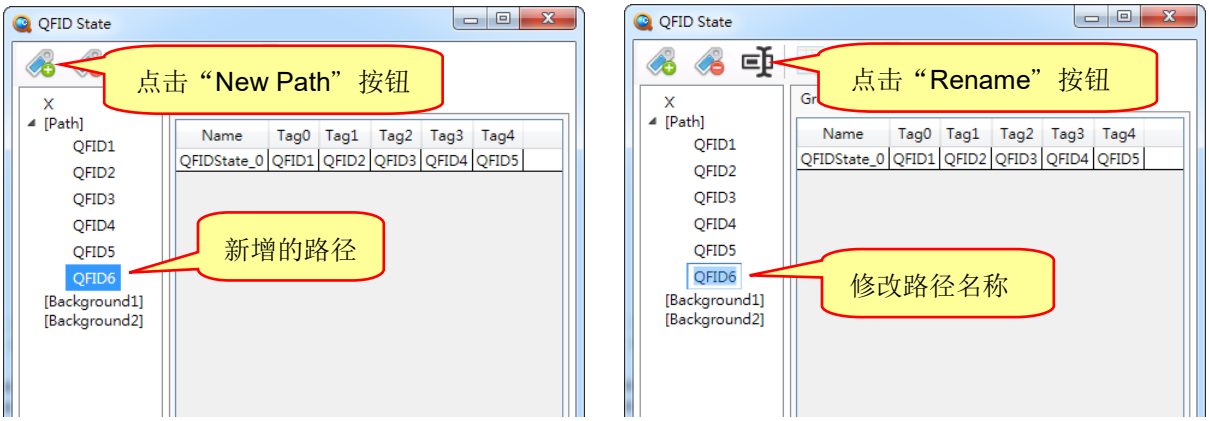
触发事项如下表所示：

格式	Tag 触发型态	注释
X	忽略	无动作。
Path1	辨识	当 Tag 辨识成立时，执行 Path1 路径。
/Path2	移除	当 Tag 移除时，执行 Path2 路径。
Path1/Path2	辨识&移除	当辨识到 Tag 时，执行 Path1 路径。 当 Tag 移除时，执行 Path2 路径。

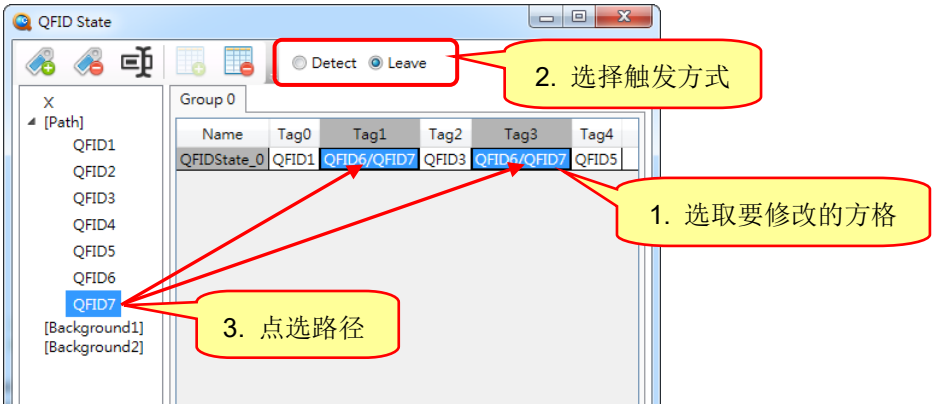
操作步骤：QFID→QFID State→Add State Name→Add PathName→OK，具体操作如下所示：

第一步：新增 **QFID State**，如下图所示：

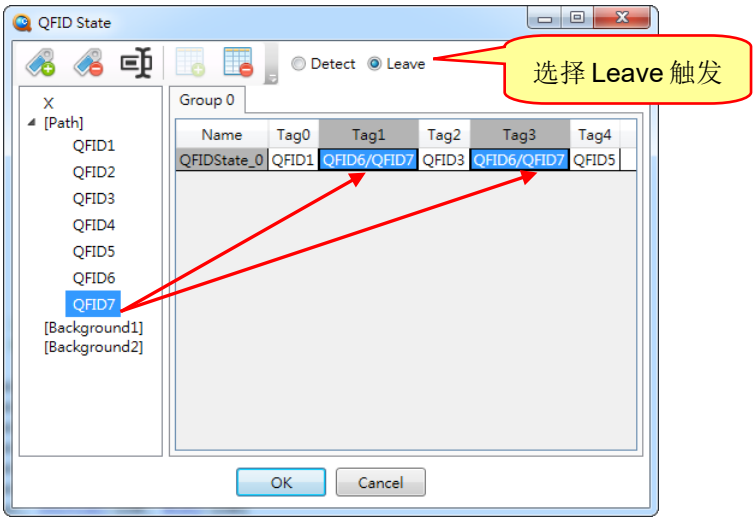
第二步：添加路径或修改路径名称，如下图：



第三步：修改对应路径。系统已经默认路径，用户也可以根据需求修改对应路径。如下图的步骤：



如果要设定移除时的触发方式，只需要在第 2 个动作的时候选择 Leave 按钮，如下图所示：



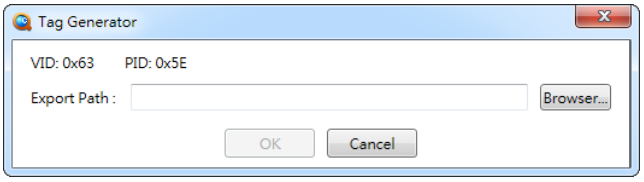
例.

[QFID]

Group0: TR1R X /TR3F TR4R/TR4F

QFID 功能仅能搭配 *Q-Code* 产生的 Tag .bin 档使用，*Q-Code* 提供 Tag Generator 工具产生 Tag .bin 文件，用户可自行通过 *Q-Writer* 烧录至 Tag IC。

使用方式如下：Tool→QFID Tag Generator



VID：对应的 QFID Vender ID。

PID：对应的 QFID Project ID。

Export Path：设定 Tag .bin 产生的路径。

输入完毕，按下 OK 按钮，即会在 Location 所指定的路径下产生 Tag .bin 文件案与 Tag 的应用电路图。

Tag_Schematic_ID.jpg。

档名格式：{VID_}{PID_}{TagID}.bin / Tag_Schematic_ID.jpg。

3.11 I/O

3.11.1 Matrix Key

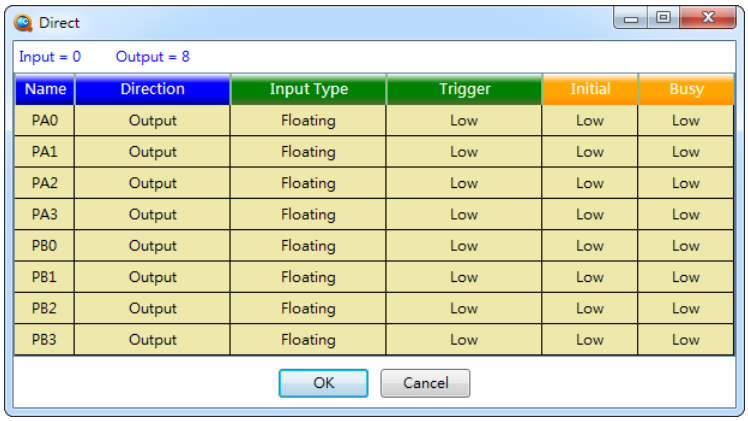
3.11.1.1 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T / NX1

点击 I / O 中的 Matrix，在弹出的窗口中点击+/-按键来设置 Matrix。点击 Available Pins 按键来设置脚位。如下图所示：

例. **Matrix =** [PA.0, PA.1] * [GND, PB.0, PA.2] ；使用 **GND** 来做 **Matrix** 按键扫描。

3.11.2 Direct Key

3.11.2.1 NY4



Direction：设定初始时 I/O 的输出入方向。Input 初始时为输入模式，Output 初始时为输出模式。

Input Type：设定一般 I/O 的输入模式中的上拉电阻。

- Pull-High 有上拉电阻的输入模式（Input mode with Pull-High Resistor）。
- Floating 无上拉电阻的输入模式（Input mode with Floating）。

Trigger：设定输入脚位的触发电平。

- Low 低电平触发（Low Trigger）。
- Hign 高电平触发（High Trigger）。

Initial：初始输出电压设定。

- Low 初始输出电压为低电平（Initial Low）。
- High 初始输出电压为高电平（Initial High）。

Busy：Action 信号的输出方式。

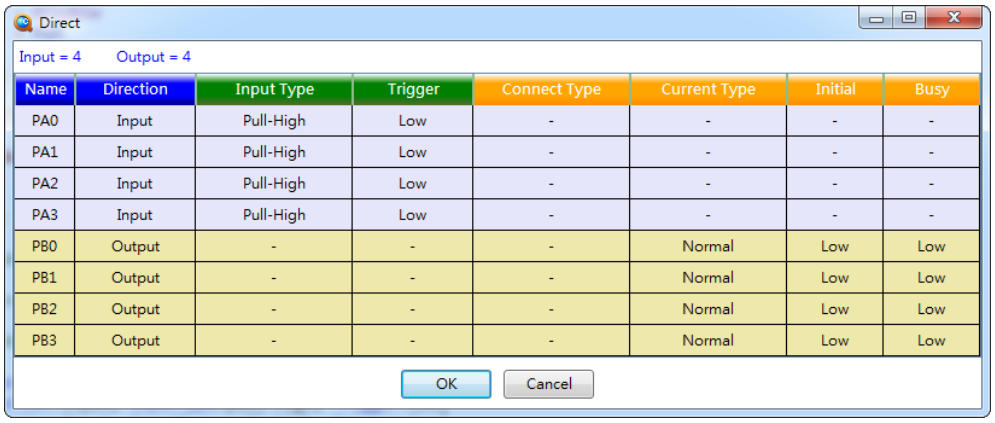
- High 设定 action 脚位的组态为 Busy High。
- Low 设定 action 脚位的组态为 Busy Low。

例.

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

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

3.11.2.2 NY5



Direction：设定初始时 I/O 的输出入方向。Input 为输入模式，Output 为输出模式。

Input Type：设定一般 I/O 的输入模式中的下拉电阻。

- Pull-High 有上拉电阻的输入模式（Input mode with Pull-High Resistor）。
- Floating 无上拉电阻的输入模式（Input mode with Floating）。
- I/O Pull-High 有上拉电阻的 I/O 模式（I/O mode with Pull-High resistor）。
- I/O Open-Drain 无上拉电阻的开汲闸 I/O 模式（Input mode without Pull-High resistor and with Open Drain）。

Trigger：设定输入脚位的触发电平。

- Low 低电平触发（Low Trigger）。
- Hign 高电平触发（High Trigger）。

Connect Type：设定外部组件的驱动方式。Sink 外部组件的驱动方式为 Active Low。

Current Type：设定电流输出方式。

- Normal：一般电流输出（Normal current output）。
- Large：大电流（Large Current output）。

Initial：初始输出电压设定。

- Low 初始输出电压为低电平（Initial Low）。
- High 初始输出电压为高电平（Initial High）。

Busy：Action 信号的输出方式。

- High 设定 action 脚位的组态为 Busy High。
- Low 设定 action 脚位的组态为 Busy Low。

注意：若 I/O 脚位要使用 input 模式，则只能将该 I/O 脚的 input mode 设成 Pull-High resistor。若是要使用 output 模式，则只能将该 I/O 脚的 output mode 设成 Normal current output 和 Initial Low，且 Normal current output 仅提供 Sink output。若 I/O 一直使用 input 模式而

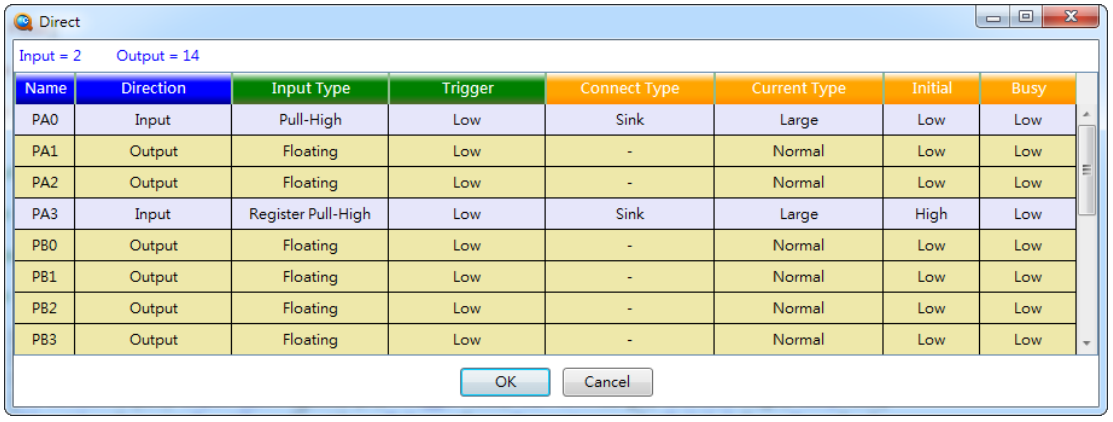
未切换成 output 模式，按键一直压住不放不能进入 Sleep mode。另外，I/O Pull-High 的默认值（Current:normal, Initial:low）的初始状态是 input 模式。这点请特别注意！

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：设定初始时 I/O 的输出入方向。Input 初始时为输入模式，Output 初始时为输出模式。

Input Type：设定一般 I/O 的输入模式中的下拉电阻。

- Pull-High 有上拉电阻的输入模式（Input mode with Pull-High Resistor）。
- Floating 无上拉电阻的输入模式（Input mode with Floating）。
- Register Pull-High 上拉电阻控制模式（Pull-High Resistor Control）。设定该模式后，上拉电阻可由 IO 寄存器来控制打开或是关闭。

例.

[Path]

TR1: PA=0xF ；打开 PA.0、PA.1 的上拉电阻。

TR2: PA=0x0 ；关闭 PA.0、PA.1 的上拉电阻。

Trigger：设定输入脚位的触发电平。

- Low 低电平触发（Low Trigger）。
- Hign 高电平触发（High Trigger）。

注意：InputType: Register_PH / Trigger: Low 模式下仅能接受低电平电压来唤醒 IC。

Connect Type：设定外部组件的驱动方式。

- Sink 外部组件的驱动方式为 Active Low。
- Drive 外部组件的驱动方式为 Active High。

Current Type：设定电流输出方式。

- Normal 一般电流输出（Normal current output）。
- Large 大电流（Large Current output）。

Initial：初始输出电压设定。

- Low 初始输出电压为低电平（Initial Low）。
- High 初始输出电压为高电平（Initial High）。

Busy：Action 信号的输出方式。

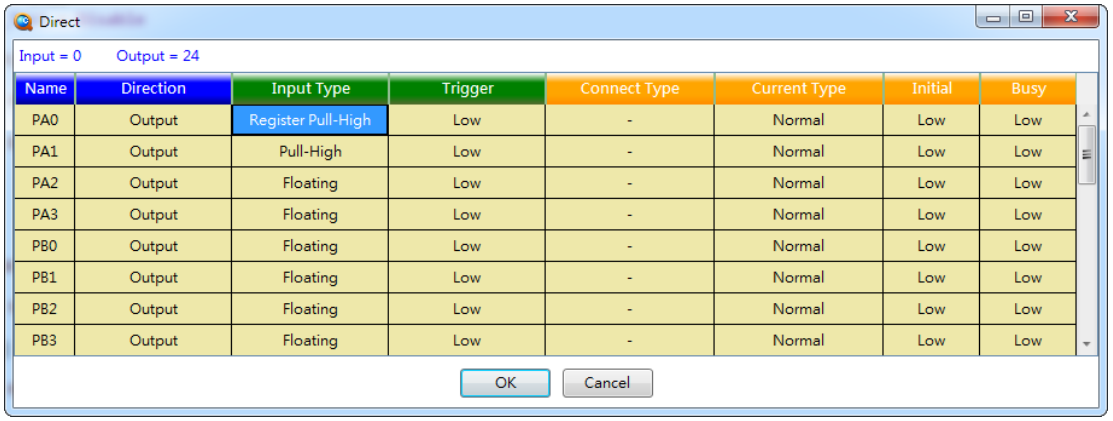
- High 设定 action 脚位的组态为 Busy High。
- Low 设定 action 脚位的组态为 Busy Low。

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：设定初始时 I/O 的输出入方向。Input 初始时为输入模式，Output 初始时为输出模式。

Input Type：设定一般 I/O 的输入模式中的下拉电阻。

- Pull-High：有上拉电阻的输入模式（Input mode with Pull-High Resistor）。
- Floating：无上拉电阻的输入模式（Input mode with Floating）。
- Register Pull-High (NY5+ / NY6 / NY7 Only)：上拉电阻控制模式（Pull-High Resistor Control）。设定该模式后，上拉电阻可由 IO 寄存器来控制打开或是关闭。

  例.

  [Path]

  TR1: PA=0xF ；**打开 PA.0、PA.1 的上拉电阻。**

  TR2: PA=0x0 ；**关闭 PA.0、PA.1 的上拉电阻。**

Trigger：设定输入脚位的触发电平。

- Low：低电平触发（Low Trigger）。
- Hign：高电平触发（High Trigger）。

注意：InputType: Register_PH / Trigger: Low 模式下仅能接受低电平电压来唤醒 IC。

Connect Type：**设定外部组件的驱动方式**。

- Sink 外部组件的驱动方式为 Active Low。

- Drive 外部组件的驱动方式为 Active High。

Current Type：设定电流输出方式。

- Normal 一般电流输出（Normal current output）。
- Large 大电流（Large Current output）。
- Constant：定电流（Constant Sink current output）。

Initial：初始输出电压设定。

- Low 初始输出电压为低电平（Initial Low）。
- High 初始输出电压为高电平（Initial High）。

Busy：Action 信号的输出方式。

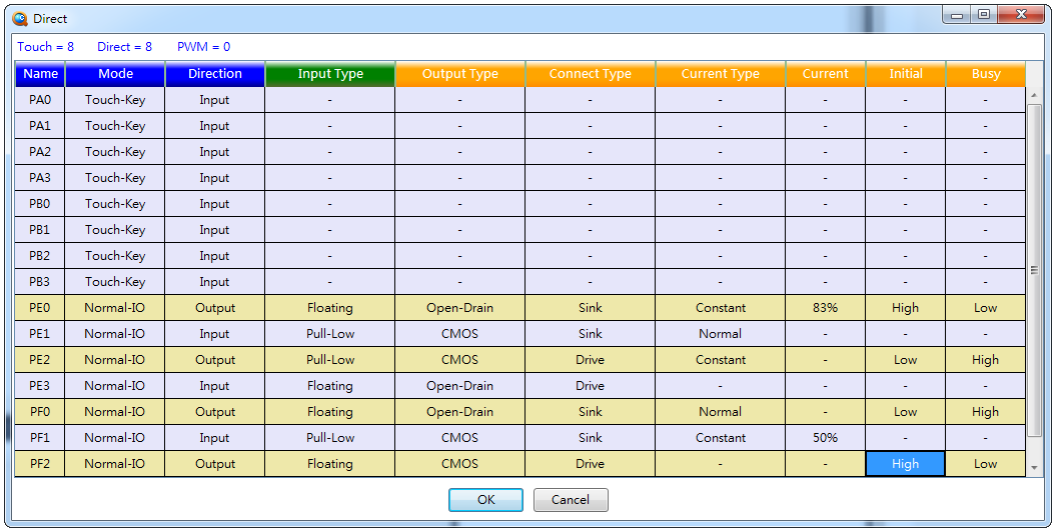
- High 设定 action 脚位的组态为 Busy High。
- Low 设定 action 脚位的组态为 Busy Low。

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：设定脚位 I/O 模式。

- Normal-IO 通用输出入脚位，由 Direction 参数决定该脚位为 Input 或是 Output。
- Touch-Key 触摸键，Direction 固定为 Input。
- PWM-IO 输出 PWM 脚位。

注意：PA.0 ~ PD.3 为触摸键与一般 IO 共享脚，PE.0 ~ PF.3 为一般 IO 与 PWM-IO 共享脚。Touch-Key 或是 PWM-IO 无法在程序中更改状态。Normal-IO 可在程序中变更输出入状态。

Direction：设定初始时 I/O 的输出入方向。Input 初始时为输入模式，Output 初始时为输出模式。

Input Type：设定一般 I/O 的输入模式中的下拉电阻。

- Pull-Low 有上拉电阻的输入模式（Input mode with Pull-Low Resistor）。

- Floating 无上拉电阻的输入模式（Input mode with Floating）。

Output Type：设定 Normal-IO 的输出状态。

- CMOS 为一般 CMOS 的 I/O 功能。
- Open-Drain 无上拉电阻的开漏极 I/O 模式。

Connect Type：设定 Normal-IO 外部组件的驱动方式。

- Sink 外部组件的驱动方式为 Active Low。
- Drive 外部组件的驱动方式为 Active High。

Current Type：设定电流输出方式。

- Normal 一般电流输出（Normal current output）。
- Large 大电流（Large Current output）。
- Constant 定电流（Constant Sink current output）。

Current：设定电流输出能力，提供四种不同的电流输出模式，仅在 Constant 与 Large 模式时有支持。

- 33% 提供 33%的电流输出。
- 50% 提供 50%的电流输出。
- 83% 提供 83%的电流输出。
- 100% 提供 100%的电流输出。

Initial：初始输出电压设定。

- Low 初始输出电压为低电平（Initial Low）。
- High 初始输出电压为高电平（Initial High）。

Busy：Action 信号的输出方式。

- High 设定 action 脚位的组态为 Busy High。
- Low 设定 action 脚位的组态为 Busy Low。

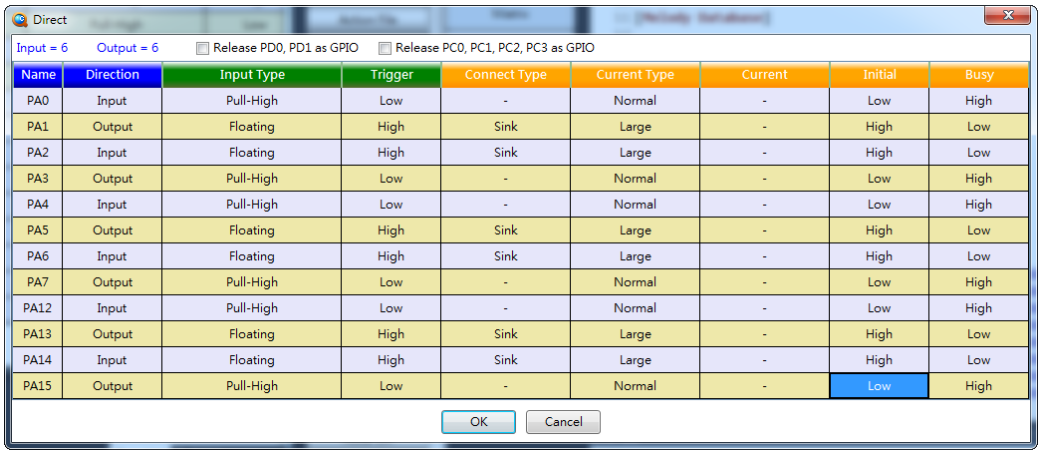
NY9T 脚位对于输出电流的支持如下表：

	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：设定初始时 I/O 的输出入方向。Input 初始时为输入模式，Output 初始时为输出模式。

Input Type：设定一般 I/O 的输入模式中的下拉电阻。

- Pull-High 有上拉电阻的输入模式（Input mode with Pull-High Resistor）。
- Floating 无上拉电阻的输入模式（Input mode with Floating）。

Trigger：设定输入脚位的触发电平。

- Low 低电平触发（Low Trigger）。
- Hign 高电平触发（High Trigger）。

Connect Type：设定外部组件的驱动方式。

- Sink 外部组件的驱动方式为 Active Low。
- Drive 外部组件的驱动方式为 Active High。

Current Type：设定电流输出方式。

- Normal 一般电流输出（Normal current output）。
- Large 大电流（Large Current output）。
- Constant 定电流（Constant Sink current output）。

Current：设定电流输出能力，提供四种不同的电流输出模式，仅在 Constant 与 Large 模式时有支持。

- 33% 提供 33%的电流输出。
- 50% 提供 50%的电流输出。
- 83% 提供 83%的电流输出。
- 100% 提供 100%的电流输出。

Initial：初始输出电压设定。

- Low 初始输出电压为低电平（Initial Low）。
- High 初始输出电压为高电平（Initial High）。

Busy：Action 信号的输出方式。

- High 设定 action 脚位的组态为 Busy High。
- Low 设定 action 脚位的组态为 Busy Low。

例.

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

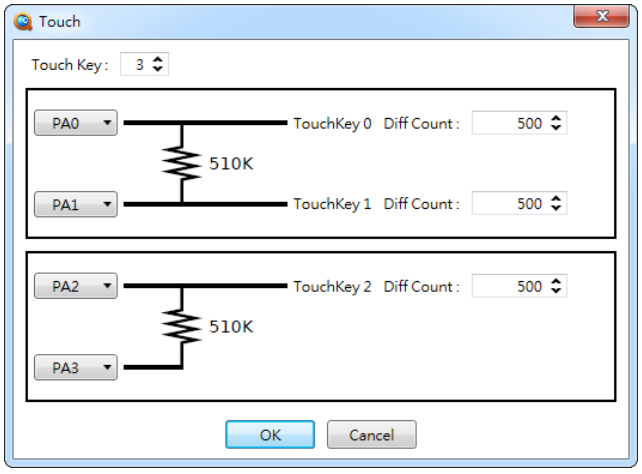
PA.1 = [Direction:Output, InputType:Floating, Trigger:High, Current:Large, Initial:Low, Busy:High]

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

3.11.3 Touch

3.11.3.1 NX1

点击 I/O 中的 Touch 设置 Touch Key 的数量和脚位。



Touch Key：设置 Touch Key 的数量。

Import .tnx：汇入 .tnx 文件。

注意：Touchkey 中，每组 Touch 需要 2 个脚位，可以使用两个 Touch Key。

例.

TouchKey = 3

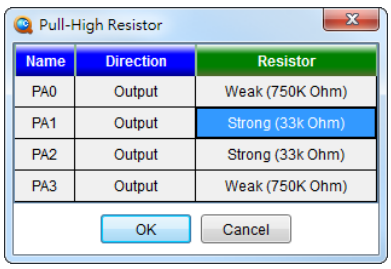
TouchGroup = [PIN0:PA.0, PIN1:PA.1, Diff0:500, Diff1:500] **; TouchKey0, TouchKey1**

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

3.11.4 Pull-High Resistor

3.11.4.1 NY4

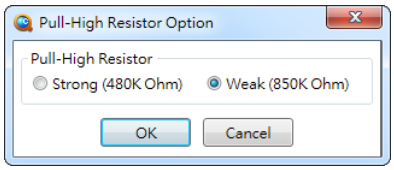
NY4 可针对每根脚位设定上拉电阻。Weak 为弱上拉 750k Ohm, Strong 为强上拉 33k Ohm。



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

3.11.4.2 NY5

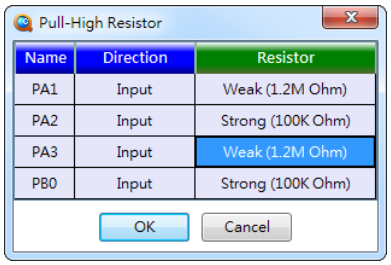
NY5 仅可使用全局的上拉电阻选项来设定电阻强度。Weak 所有脚位的上拉电阻皆为弱上拉 850k Ohm，Strong 所有脚位的上拉电阻皆为强上拉 480k Ohm。



例. **Input_Resistor** = strong

3.11.4.3 NY5+

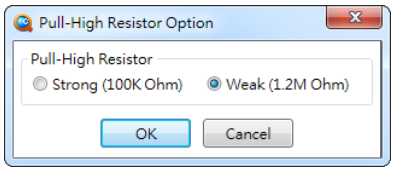
NY5+可针对每根脚位设定上拉电阻。Weak 为弱上拉 1.2M Ohm，Strong 为强上拉 100k Ohm。



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

3.11.4.4 NY6

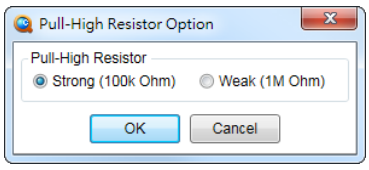
NY6 仅可使用全局的上拉电阻选项来设定电阻强度。Weak 所有脚位的上拉电组皆为弱上拉 1.2M Ohm，Strong 所有脚位的上拉电阻皆为强上拉 100k Ohm。



例. **Input_Resistor** = strong

3.11.4.5 NY7

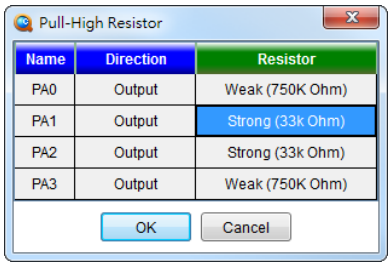
NY7 仅可使用全局的上拉电阻选项来设定电阻强度。Weak 所有脚位的上拉电阻皆为弱上拉 1M Ohm，Strong 所有脚位的上拉电阻皆为强上拉 100k Ohm。



例. **Input_Resistor** = strong

3.11.4.6 NY9T

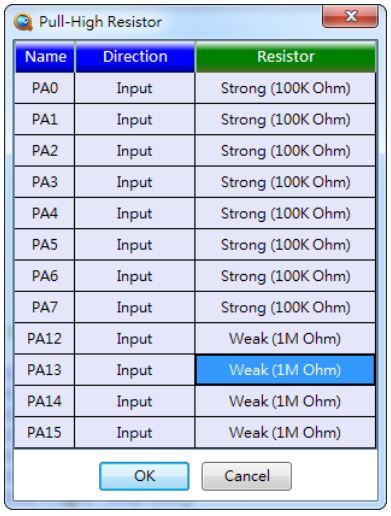
NY9T 可针对每根脚位设定上拉电阻。Weak 为弱上拉 1M Ohm， Strong 为强上拉 100k Ohm。



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

3.11.4.7 NX1

NX1 可设定脚位上拉电阻，但 PA / PB / PC 的 0 ~ 7 脚位必须相同，PA / PB 的 8 ~ 15 脚位亦须相同。Weak 为弱上拉 1M Ohm，Strong 为强上拉 100k Ohm。

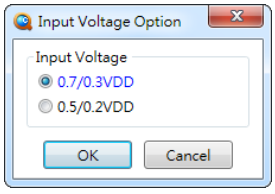


例. **Input_Resistor** = [PA.0:Strong, PA.1:Strong, PA.2:Strong, PA.3:Strong,
PA.4:Strong, PA.5:Strong, PA.6:Strong, PA.7:Strong,
PA.12:weak, PA.13:weak, PA.14:weak, PA.15:weak]

3.11.5 Input Voltage

3.11.5.1 NX1

设定输入电压电位。



例. **Input_Voltage** = VIH7_VIL3 ； **高电位 0.7VDD，低电位 0.3VDD**

3.12 Input State

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

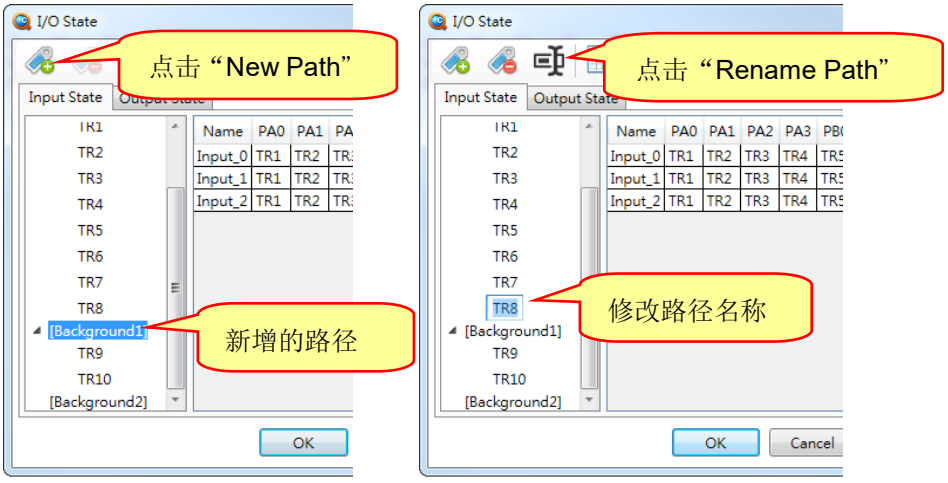
定义相对应的按键动作，最多可定义 255 组 input state。按键触发型态是指当相应的输入键被按下或释放时所应该作出的反应，触发的数目必须与输入脚数相同，且触发事项之间必须以空格（TAB 或 space）来分开。触发事项如下表所示：

格 式	按键触发型态	注释
X	不管	无动作。
Path1	下降沿	当输入脚从高电平到低电平时，执行 Path1 路径。
/Path2	上升沿	当输入脚从低电平到高电平时，执行 Path2 路径。
Path1/Path2	下降沿&上升沿	当输入脚从高电平到低电平时，执行 Path1 路径。 当输入脚从低电平到高电平时，执行 Path2 路径。

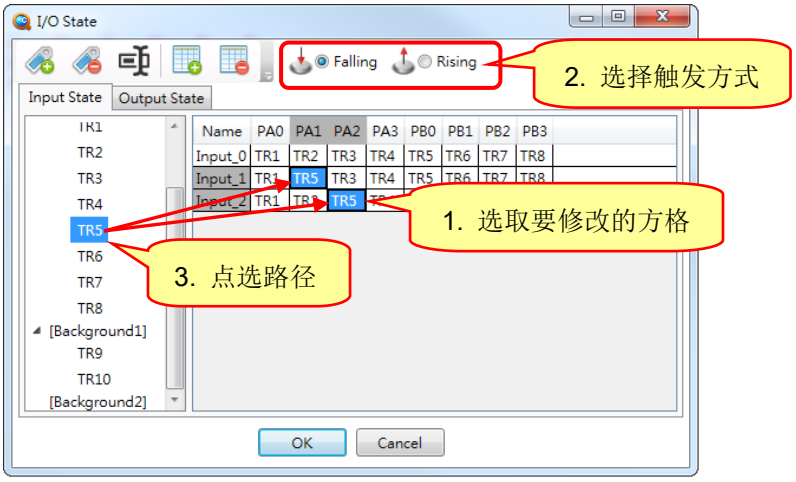
具体操作如下所示：

第一步：新增 **Input State**：

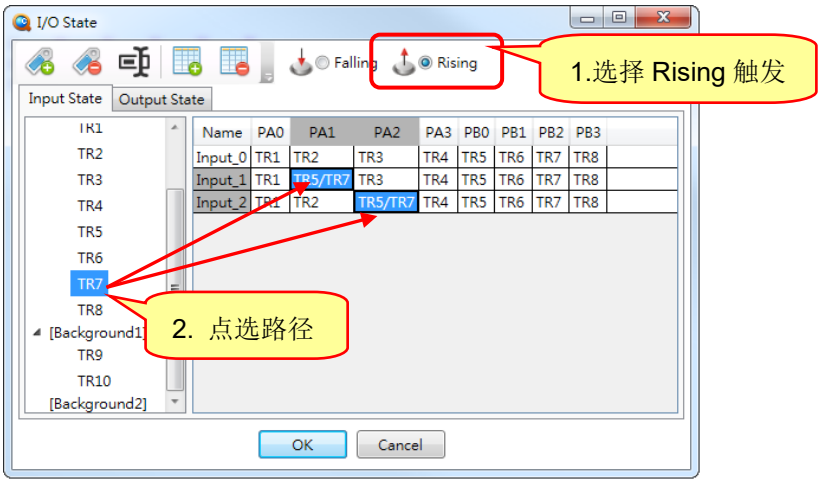
第二步：添加路径或修改路径名称，如下图：



第三步：修改按键路径。系统已经默认路径，用户也可以根据需求修改按键路径。如下图的步骤：



如果想使用 Rising 触发方式，只需要在第 2 个动作的时候选择 Rising 按钮，如下图所示：



例.

[Input State]

Input_0: TR1R X / TR3F TR4R / TR4F

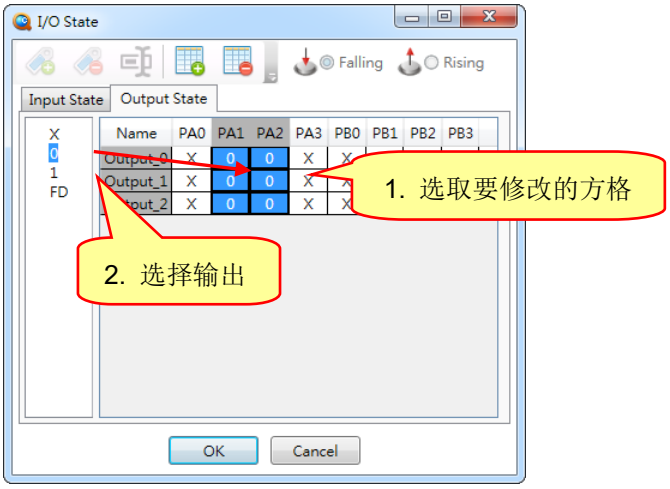
3.13 Output State

用户可以在 **[Output State]** 段落中定义输出的状态。最多可定义 255 组 output state。在 **[Path]** 与 **[Background]** 都可以调用 **[Output State]** 中所定义的 output state。每个 output state 都由一个 output state 名称，一个冒号":"及指定的输出值组成。输出值是指当该 output state 被调用时，每个输出脚应该输出的状态值。输出值的数目必须与输出脚数相同，且输出值之间必须以空格（TAB 或 space）分开。

具体操作步骤如下所示：

第一步：新增 **Output State**。

第二步：修改输出信号。



例.

[Output State]

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

3.13.1 NY4 / NY5

以下是 output state 的定义：

输出值	注释
X	表示该脚位维持原来的状态。
0	表示该脚位输出低电平。
1	表示该脚位输出高电平。
A	表示该脚位输出 Action。（该功能仅对于播放整个 VIO 文件有效）
FD	表示随播放的声音音量大小变化。（Flash with Dynamic）
Q	表示各脚位跟随 *Quick-IO* 的 QIO 信号变化。

3.13.2 NY5+

以下是 output state 的定义：

输出值	注释
X	表示该脚位维持原来的状态。
0	表示该脚位输出低电平。
1	表示该脚位输出高电平。
A	表示该脚位输出 Action。（该功能仅对于播放整个 VIO 文件有效）
P	表示该脚位输出 PWM-IO。（该功能仅对 PlayPWM 播放 MPIOx 时有效）
FD	表示随播放的声音音量大小变化。（Flash with Dynamic）
Q	表示各脚位跟随 *Quick-IO* 的 QIO 信号变化。

3.13.3 NY6 / NY7

以下是 output state 的定义：

输出值	注释
X	表示该脚位维持原来的状态。
0	表示该脚位输出低电平。
1	表示该脚位输出高电平。
A	表示该脚位输出 Action。（该功能仅对于播放整个 VIO 文件有效）
FD	表示随播放的声音音量大小变化。（Flash with Dynamic）

3.13.4 NY9T

以下是 output state 的定义：

输出值	注释
X	表示该脚位维持原来的状态。
0	表示该脚位输出低电平。
1	表示该脚位输出高电平。
A	表示该脚位输出 Action。（该功能仅对于播放整个 VIO 文件有效）

3.13.5 NX1

以下是 output state 的定义：

输出值	注释
X	表示该脚位维持原来的状态。
0	表示该脚位输出低电平。
1	表示该脚位输出高电平。
FD	表示随播放的声音音量大小变化。（Flash with Dynamic）
Q	表示各脚位跟随 *Quick-IO* 的 QIO 信号变化。

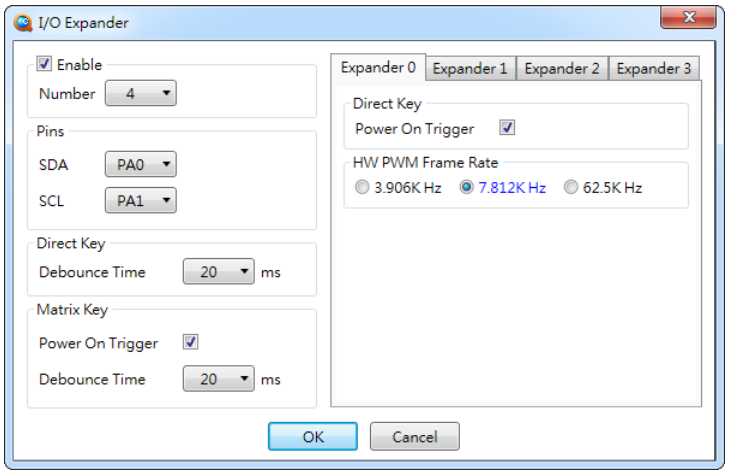
3.14 I/O Expander

3.14.1 I/O Expander

Q-Code 支持 I/O 扩展芯片功能，当 IC 脚位不足时，可透过 I/O 扩展芯片增加可用脚位。

3.14.1.1 NY5+ / NX1

可支持 4 组 I/O 扩展芯片。



Number：要使用的 I/O 扩展芯片数量。

SDA 脚位：连接至 I/O 扩展芯片的 SDA 脚位。

SCL 脚位：连接至 I/O 扩展芯片的 SCL 脚位。

Debounce Time：用户可以选择 Direct Key 和 Matrix Key 的 debounce 时间，范围：1~1000ms。

Expander Power On Trigger：选择 I/O 扩展芯片是否要开启 Direct Key 和 Matrix Key 的 Power on Trigger 功能。当功能开启后，如果用户按着 I/O 扩展芯片上的按键后开机，则开机后会马上去执行该按键对应的路径。可以独立选择每个 I/O 扩展芯片 Direct Key 是否要开启 Power on Trigger 功能。

Expander HW PWM Frame Rate：设定 I/O 扩展芯片脚位作为硬体 PWM 输出时的 Frame Rate。（预设 7.812kHz）

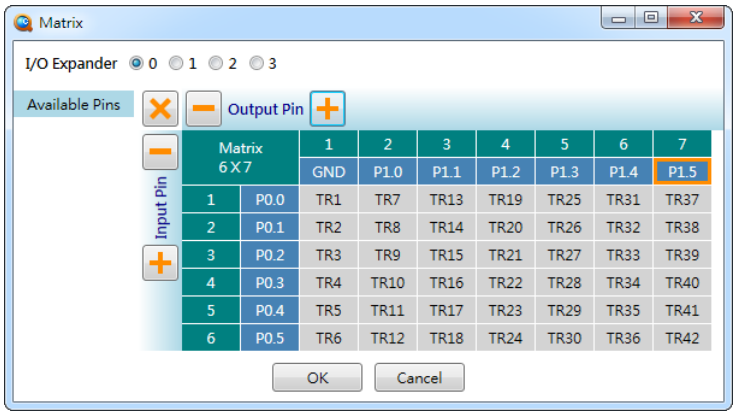
例.

```
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

设置 I/O 扩展芯片的 Matrix 功能，仅支援一组 I/O 扩展芯片设置 Matrix。

3.14.2.1 NY5+ / NX1



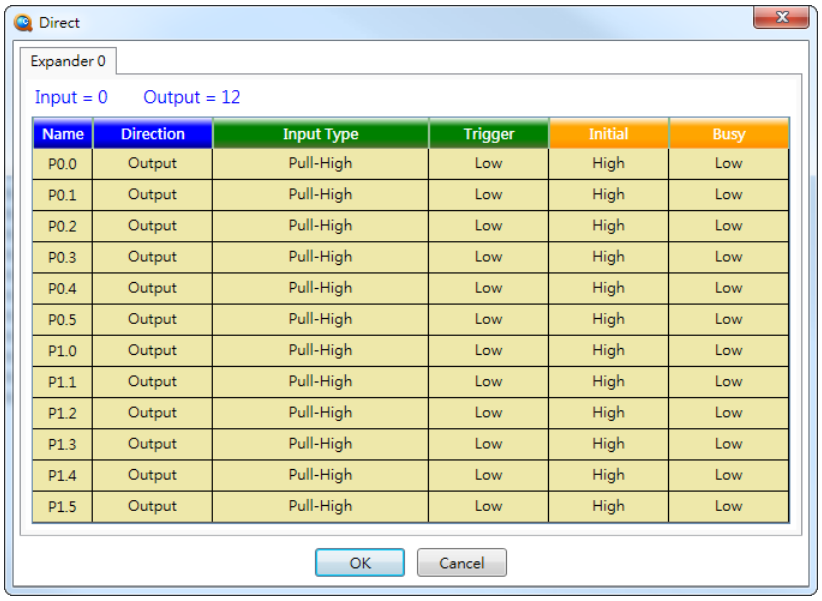
例.

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

设置可各个 I/O 扩展芯片的脚位。

3.14.3.1 NY5+ / NX1



Direction：设定初始时脚位的输出入方向。Input 初始时为输入模式，Output 初始时为输出模式。

Input Type：设定脚位的输入模式中的上下拉电阻。

- Pull-High 有上拉电阻的输入模式（Input mode with Pull-High Resistor）。
- Pull-Low 有下拉电阻的输入模式（Input mode with Pull-Low Resistor）。

- Floating 无上下拉电阻的输入模式（Input mode with Floating）。

Trigger：设定输入脚位的触发准位。

- Low 低准位触发（Low Trigger）。
- High 高准位触发（High Trigger）。

Initial：初始输出电压设定。

- Low 初始输出电压为低准位（Initial Low）。
- High 初始输出电压为高准位（Initial High）。

Busy：Action 信号的输出方式。

- High 设定 action 脚位的组态为 Busy High。
- Low 设定 action 脚位的组态为 Busy Low。

例.

EXP0_P0.0 = [Direction:Output, InputType:pull-high, Trigger:low, Initial:high, Busy:low]

3.14.4 Input State

开启 I/O 扩展芯片功能后，用户可以在 **[Input State ExpID]** 段落中定义各个 I/O 扩展芯片对应的按键动作。***Q-Code*** 会依据 I/O 扩展芯片使用的数量，自动产生相对应的段落。

按键触发型态是指当相应的输入键被按下或释放时所应该作出的反应，触发的数目必须与输入脚数相同，且触发事项之间必须以空格（TAB 或 Space）来分开。

3.14.4.1 NY5+ / NX1

NY5+最多可定义 255 组 input state，NX1 则无限制。UI 的操作方式同 Input State。触发事项如下表所示：

格式	按键触发型态	注释
X	忽略	无动作。
Path1	下降缘	当输入脚从高电位到低电位时，执行 Path1 路径。
/Path2	上升缘	当输入脚从低电位到高电位时，执行 Path2 路径。
Path1/Path2	下降缘&上升缘	当输入脚从高电位到低电位时，执行 Path1 路径。 当输入脚从低电位到高电位时，执行 Path2 路径。

例.

[Input State Exp0]

InputExp0_0: TR1 TR2 TR3 TR4 TR5 TR6

3.14.5 Output State

开启 I/O 扩展芯片功能后，用户可以在 **[Output State ExpID]** 段落中定义各个 I/O 扩展芯片输出脚的状态。***Q-Code*** 会依据 I/O 扩展芯片使用的数量，自动产生相对应的段落。

在 **[Path]** 与**[Background]** 都可以呼叫 **[Output State ExpID]** 中所定义的 output state。每个 output state 都由一个 output state 名称，一个冒号":"及指定的输出值组成。输出值是指当该 output state 被呼

叫时，每个输出脚应该输出的状态值。输出值的数目必须与输出脚数相同，且输出值之间必须以空格（TAB 或 space）分开。

3.14.5.1 NY5+ / NX1

UI 的操作方式同 Output State。下列表格是 I/O 扩展芯片的 output state 可用的输出状态。

输出值	注释
X	表示该脚位维持原来的状态。
0	表示该脚位输出低电位。
1	表示该脚位输出高电位。

例.

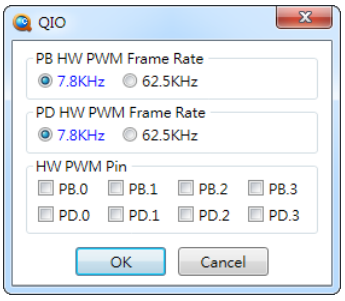
[Output State Exp0]

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 信号的更新率。

PD HW PWM Frame Rate：Port D Hardware PWM 信号的更新率。

HW PWM Pin：选择输出 Hardware PWM 信号的脚位。

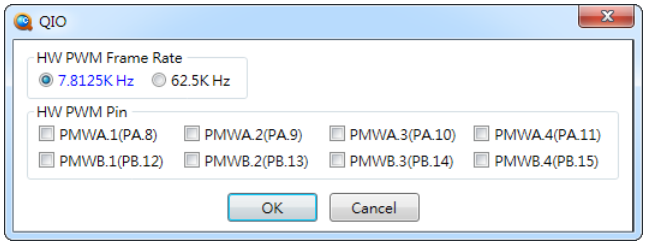
例.

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 信号的更新率。

HW PWM Pin：选择输出 Hardware PWM 信号的脚位。

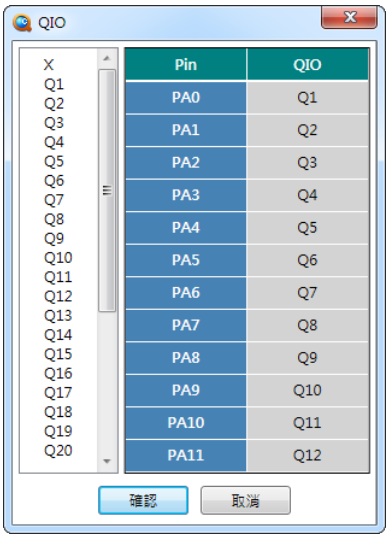
例.

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

HW_PWM_FrameRate = 7812

3.15.2 QIO Table

设定 QIO 信号与输出脚的对映组态。仅支持一组设定。项目中只要输出 QIO 即会套用此处的设定。



例.

[QIO]

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

3.15.2.1 NY4

当 **[QIO]** 未设定时，QIO 信号与脚位的对映关系如下：

	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

当 **[QIO]** 未设定时，QIO 信号与脚位的对映关系如下：

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+

当 **[QIO]** 未设定时，QIO 信号与脚位的对映关系如下：

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

当 **[QIO]** 未设定时，QIO 信号与脚位的对映关系如下：

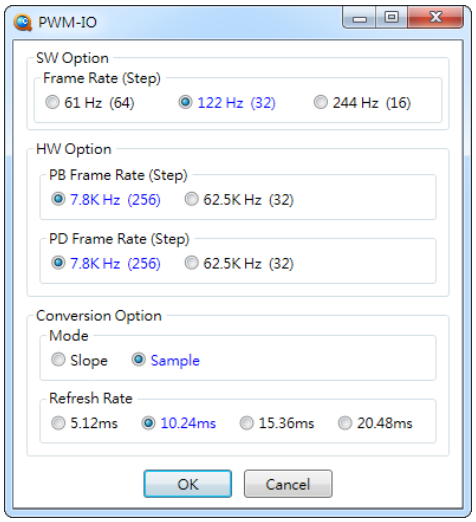
Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
PA.0	PA.1	PA.2	PA.3	PA.4	PA.5	PA.6	PA.7
Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16
PA.8	PA.9	PA.10	PA.11	PA.12	PA.13	PA.14	PA.15
Q17	Q18	Q19	Q20	Q21	Q22	Q23	Q24
PB.6	PB.7	PB.8	PB.9	PB.10	PB.11	PB.12	PB.13
Q25	Q26	Q27	Q28	Q29	Q30	Q31	Q32
PB.14	PB.15	PC.0	PC.1	PC.2	PC.3	PC.4	PC.5

3.16 PWM-IO

用于设定 PWM-IO 输出的组态。设定好的输出组态可以在 PlayPWM / PlayPWMS 及 PWMOut / PWMOutS 中使用。

3.16.1 Configure

3.16.1.1 NY5+



SW Frame Rate (Step)：当使用 PWMOut / PWMOutS 并设定非 PB 及 PD 接脚，才会依照此设定控制 PWM 周期。Frame Rate 越高则输出较不易闪烁，但可用阶数较少且系统的负载也高，Frame Rate 越低则有较高的阶数，但较易出现闪烁。

HW PB Frame Rate (Step)：当使用 PlayPWM / PlayPWMS 及 PWMOut / PWMOutS 并设定 PB 接脚，会依照此设定控制 PWM 周期。Frame Rate 越高则输出较不易闪烁但可用阶数较少，Frame Rate 越低则输出阶数较多但较易出现闪烁。

HW PD Frame Rate (Step)：当使用 PlayPWM / PlayPWMS 及 PWMOut / PWMOutS 并设定 PD 接脚，会依照此设定控制 PWM 周期。Frame Rate 越高则输出较不易闪烁但可用阶数较少，Frame Rate 越低则输出阶数较多但较易出现闪烁。

Conversion Option：为数据转换时的选项，*仅适用于 PlayPWM / PlayPWMS 指令*。

Mode：Slope 使用斜率方式做数据转换，Sample 则是以对准取样点的方式做数据转换。当使用 Sample 模式时，耗用的 ROM 较多，而使用 Slope 模式时，耗用的 RAM 较多。

Refresh Rate：数据转换时隔多久取样一次。*间隔越小，数据量越大。*

例.

SW_FrameRate = 122Hz

PB_FrameRate = 7.8KHz

PD_FrameRate = 7.8KHz

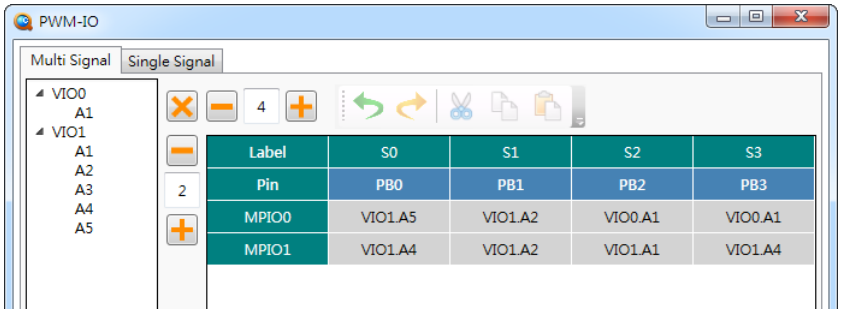
Conversion_Mode = Sample

Refresh_Rate = 10.24ms

3.16.2 Multi Signal

3.16.2.1 NY5+

可以设定多信号的输出组态。播放时会同时输出至对应的脚位。输出组态最少需包含 2 只脚位，上方 +/-可调整脚位数量。Multi Signal 当透过 PlayPWM 指令播放时，会依照此处的设定，将指定的信号输出至指定的脚位。



例.

[PWM-IO]

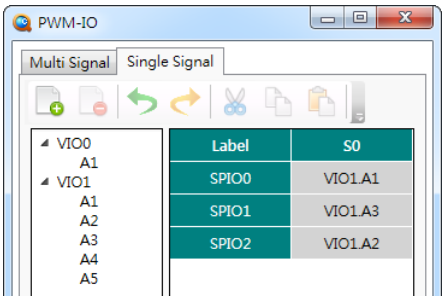
Multi_Pins = [PB.0, PB.1, PB.2, PB.3]

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

3.16.3 Single Signal

3.16.3.1 NY5+

可以设定单一信号的输出组态。Single Signal 不指定输出脚位，因此使用 PlayPWM 指令时，需要指定输出脚位。



例.

[PWM-IO]

SPIO0 = [VIO0.A1]

SPIO1 = [VIO1.A1]

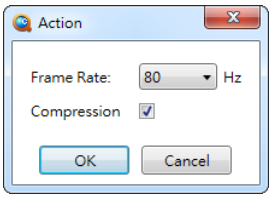
3.17 Action

设定 Action 输出的组态。设定好的输出组态可以在 PlayA / PlayAS 中使用。

3.17.1 Configure

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

Frame Rate：信号每秒的更新频率。Frame Rate 越高则输出较不易闪烁，但可用阶数较少且系统的负载也高。



Compression：开启 Action 信号压缩功能。（默认值 Enable）

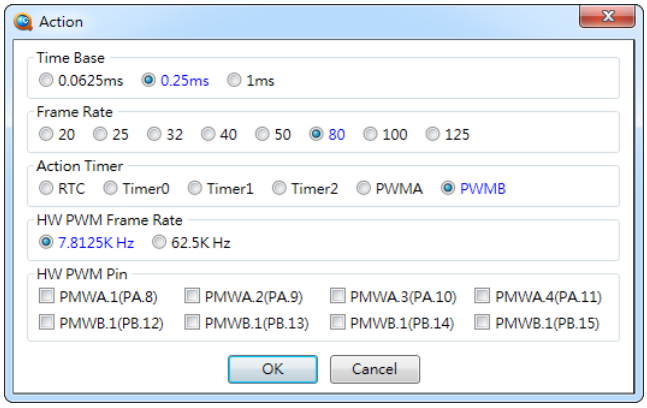
例.

[Action]

Compression = Enable

FrameRate = 80

3.17.1.2 NX1



Time Base：时间计数的最小单位，单位为毫秒(ms)。

Frame Rate：信号每秒的更新频率。Frame Rate 越高则输出较不易闪烁，但可用阶数较少且系统的负载也高。

Action Timer：设定 Action 使用的 Timer。

HW PWM Frame Rate：HW PWM 信号每秒的更新频率。

HW PWM Pin：设定使用 HW PWM 的脚位。

例.

[Action]

FrameRate = 80

TimeBase = 0.25ms

Timer = PWMB

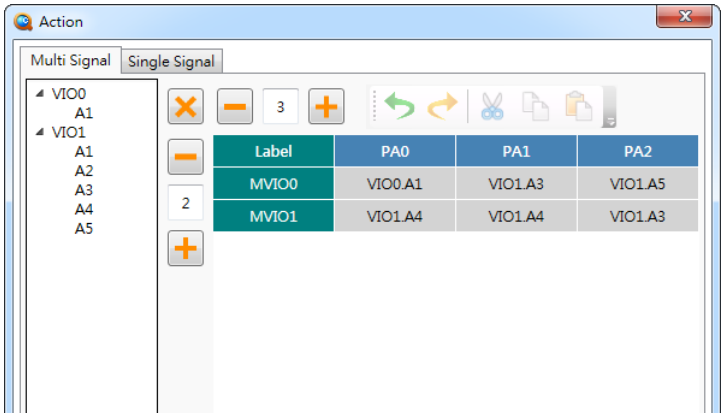
HW_PWM_FrameRate = 7812

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

3.17.2 Multi Signal

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

可以设定多信号的输出组态。播放时会同时输出至对应的脚位。输出组态最少需包含 2 只脚位，上方 +/-可调整脚位数量。Multi Signal 当透过 PlayA 指令播放时，会依照此处的设定，将指定的信号输出至指定的脚位。



例.

[Action]

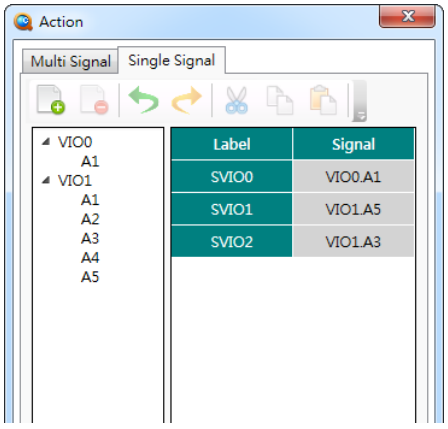
Multi_Pins = [PB.0, PB.1, PB.2, PB.3]

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

3.17.3 Single Signal

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

可以设定单一信号的输出组态。Single Signal 不指定输出脚位，因此使用 PlayA 指令时，需要指定输出脚位。



例.

[Action]

SVIO0 = [VIO0.A1]

SVIO1 = [VIO1.A1]

3.18 VR

语音识别段落（VR，Voice Recognition）可引用 *Cyberon CSpotter Modeling Tool* 所产生的 .cvr 文件（VR File），并设定语音识别状态（VR State）指定语音指令辨识成功所要执行的路径。

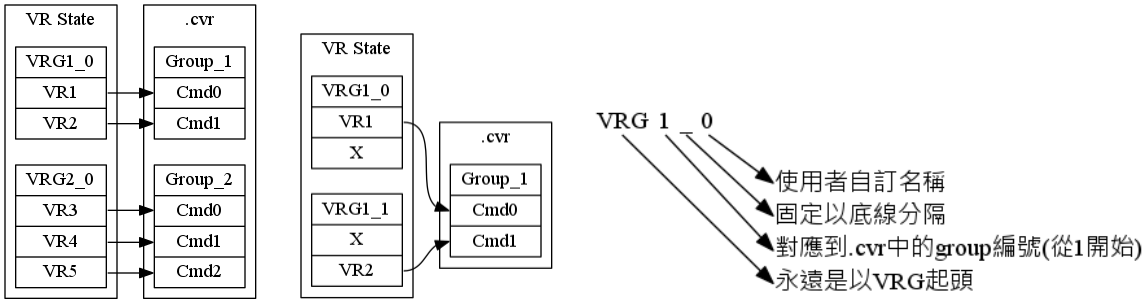
语音识别文件.cvr 文件可选择存放于 IC 内部的 ROM，亦可选择存放于 SPI Flash。用户可以在 VR File 选项中指定将.cvr 文件存放于 IC 内部的 ROM 或外部 SPI Flash，两者只能选择其一。

除了脱机建立语音指令，另外可以在 IC 运作时，在线训练出语音指令，Voice Tag 提供这样的功能。Voice Password 提供在线训练语音指令并记录声纹特征，训练出来的语音指令，须使用相似的声音才能成功辨识。

设定语音指令启用状态及其对应路径的方式有两种，VR State 与 VRGC state (VR Group Combo state)。*Cyberon CSpotter Modeling Tool* 项目中可以有多个 group，VR State 指定启用其中一个语音指令群组(VR Group)，程序中可设定多个 VR State，每一个 VR State 都可以指定对应到不同的语音指令群组。VRGC state 则是设定启用多个语音指令群组（最多三个群组），例如，如果设定结合 2 个 group，这两个 group 的所有指令都会被判断和辨识，VRGC 的所有群组的每个指令也都有对应的路径。

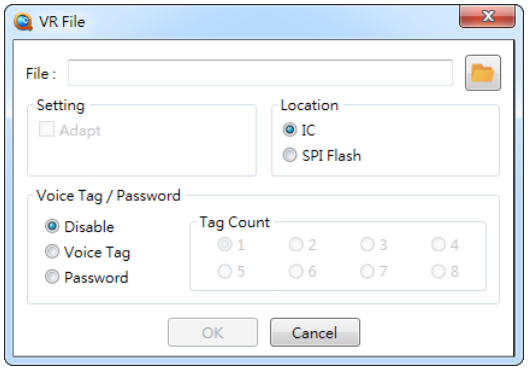
VR State 的名称必须与.cvr 文件的指令群组有所对应，例如 VRG1_0 的 1 代表第一个语音指令群组；VRG2_0 的 2 代表第二个语音指令群组。路径名称默认为 VR1、VR2、……依序编号，如果想关闭某条语音指令，可在 VR State 对应的路径写 X，代表不动作。VR State 名称必须符合规范，以 VRG 起头，跟着一数字代表指令群组编号，底线分隔后接着用户自定义名称。

当使用 VoiceTag / Password 模式时，可以使用语音指令群组 VTG，所对应的 VR State 名称为 VTG_0、VTG_1、…。VTG 语音群组亦可使用于 VRGC state 中。



3.18.1 VR File

3.18.1.1 NX1



File：.cvr 文件的路径。

Location：选择 cvr 存放的位置，选择放置于外部 SPI Flash 时，会复写 .spiprj 档的内容。（若 .spiprj 档不存在，需先选择要开启的 .spiprj 档）。选择放置于 IC 内部则产生范例内容。

Adapt：选择要不要使用 Adpat 功能。

Voice Tag / Password：选择使用 Voice Tag 或 Password。

Tag Count：当使用 Voice Tag / Password 时选择对应的语音指令群组所包含的指令数量。Voice Tag 最多 8 个，Password 最多 1 个。

例.

[VR]

VR File = C:\test.cvr

VR_Application = VoiceTag

VT_TagCount = 6

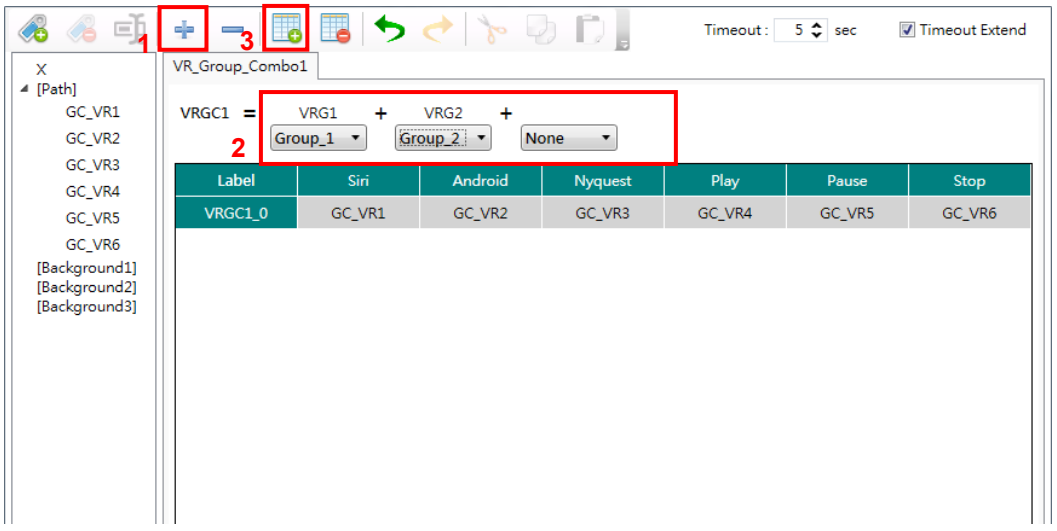
3.18.2 VR Combo

3.18.2.1 NX1

VRGC state 为多个 VR Group 的集合，必须先设定 VRGC 由哪些 VR Group 组成，然后指定所有群组的每一条语音指令所对应的路径。不论语音指令是属于哪一个群组，一旦被辨认成功都会立刻执行相对应的路径。

编辑 VRGC State 流程：

1. 按下新增将自动产生名称为 VRGC1 的 VR Group Combo。
2. 选择 VR Group Combo 要结合的群组（最多三个群组）。
3. 按下新增将自动产生名称为 VRGC1_0 的状态，例如以下范例中，第一个群组有三个语音指令分别对应到 G1_VR1、G1_VR2、G1_VR3 三条路径，第二个群组有三个语音指令分别对应到 G2_VR1、G2_VR2、G3_VR3 三条路径



注意：.cvr 档必须有两个以上的 Group 才能使用此功能。

VRGC Timeout 为计数逾时，分为两个 VR Group Combo 和三个 VR Group Combo 来说明。

两个 Group Combo：第一组 VR Group 中的指令成功辨识后，开始计数 timeout，直到指令 VRGC_Timeout_CLR 被调用后，停止计数。若计数时间到，仍未有呼叫 VRGC_Timeout_CLR，则跳到 timeout 路径。

三个 Group Combo：第一组 VR Group 中的指令成功辨识后，开始计数 timeout，若第二组 VR Group 中的指令成功辨识后，会再重新开始计时，直到指令 VRGC_Timeout_CLR 被调用后，停止计数。若第一组 VR Group 中的指令没有先成功辨识，反而是第二组 VR Group 中的指令先成功辨识，timeout 也会停止计数。

Timeout Extend 为延长 timeout，延长次数以一次为限。若 VRGC timeout 开始计数到时间结束期间，若有声音超过 VAD 门坎，会自动延长一次。

例.

```
VRGC1 = VRG1 + VRG2              ; VRGC1 结合 VRG1 和 VRG2 两个 group
VRGC1_0: G1VR1 G1VR2             ; VRG1 两条语音指令的路径。
         G2VR1 G2VR2             ; VRG2 两条语音指令的路径。
```

VRGC state 提供逾时机制。以上面的范例来说，若 VRG1(第一组 VR Group)之中的指令成功辨认，但并未在时限内辨认到 VRG2(第二组 VR Group)的指令，将会自动执行 VRGC_Timeout 路径，如果第二组 Group 之中的指令先成功辨认，则 Timeout 条件不会成立，动作上一定要第一组 VR Group 之中的指令先成功辨识，VRGC_Timeout 路径才会成立。逾时时间的长度单位可为秒，最长可设为 60 秒。Timeout Extend 功能打开后，当 VRG1(第一组 VR Group)之中的指令辨认成功后，若有声音大于 VAD 阀值，则自动延长逾时时间的长度。

```
例. VRGC_Timeout = 5s            ; 指定 timeout 时间 5 秒。
例. VR_Timeout_Extend = Enable   ; 打开自动延长逾时时间功能。
```

VRGC state 也可用来订定指令群组被辨识的顺序，当特定指令被辨识后才可触发真正指令。例如：“Hi Siri, what time is it?”，Hi Siri 可作为第一个群组指令，当被辨识后才接受第二个群组指令 what time is it?。程序撰写上可使用旗标变量记录第一个群组指令是否已被辨识的状态，如以下范例。

例.

```
[VR]
VRGC_Timeout = 5s                ; 指定 timeout 时间 5 秒。
VRGC1 = VRG1 + VRG2              ; VRGC1 结合 VRG1 和 VRG2 两个 group。
                                 ; VRG1 为前置词的 group。
                                 ; VRG2 为后续指令的 group。
VRGC2 = VRG1 + VRG3              ; VRGC2 结合 VRG1 和 VRG3 两个 group。
VRGC1_0: G1_VR1 G1_VR2           ; VRG1 的 state 中定义所有指令的 path。
         G2_VR1 G2_VR2           ; 第二个群组指令对应的 path。

[Variable]
```

VAR8: flag

[Path]

PowerOn: VRGC1_0 ; 启用 **VRGC1_0** 状态。

G1_VR1: flag = 1 ; 前置词出现，设定 **flag**。

G1_VR2: flag = 1 ; 前置词出现，设定 **flag**。

G2_VR1: flag = 1 ? G2_VR1_True ; 如果有前置词才执行路径。

G2_VR2: flag = 1 ? G2_VR2_True ; 如果有前置词才执行路径。

VRGC_Timeout: flag = 0 ; 如果讲了前置词，却没有后续指令，时间到会执行 ; **VRGC_Timeout** 路径让用户可以重设状态。

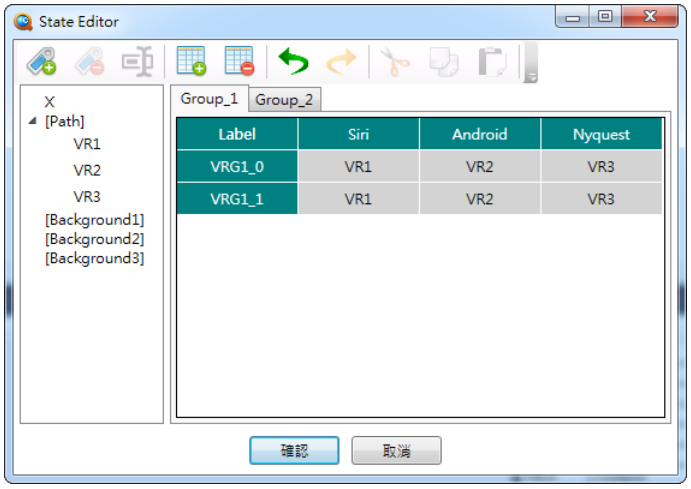
G2_VR1_True: flag = 0, VRGC_Timeout_CLR, PlayV(ch0, $V1) ; 当被辨识到就播放声音。

G2_VR2_True: flag = 0, VRGC_Timeout_CLR, PlayV(ch0, $V1) ; 当被辨识到就播放声音。

3.18.3 VR State

3.18.3.1 NX1

VR State 编辑窗口。



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)

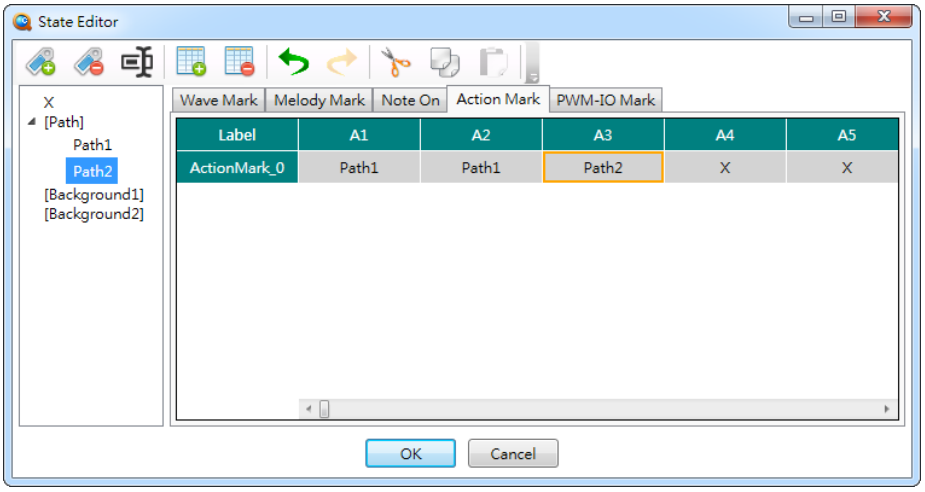
VR4: PlayV(ch0, $V3)

3.19 Action Mark

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

当用户使用 *Q-Visio* 编辑文件时，可以随意在 action 文件中插入 Mark 标记。当 *Q-Code* 读到标记时，会依照当前的 Action Mark 状态自动执行对应的路径。用户可以在 **[Action Mark]** 段落中定义 Action Mark 的状态。

Action Mark 的标记编号最多可有 255 个，从 A1 至 A255。在 *Q-Code* 中，用户仅需要加入.vio 文件及定义好标记对应的路径即可。最多可以定义 255 组设定。



注意：

1. Action Mark 状态的切换，等同 Input State 的用法。

2. 标记编号必须为 A1~A255。

例. 在 *Q-Visio* 中，插入 A1~A4 的码。

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

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

[Background1]
AM1: PB=0x1                  ; 当读取到 Action Mark，且 Mark 编号为 1，即会执行 AM1。
AM2: PB=0x2                  ; 当读取到 Action Mark，且 Mark 编号为 2，即会执行 AM2。
AM3: PB=0x3                  ; 当读取到 Action Mark，且 Mark 编号为 3，即会执行 AM3。
AM4: PB=0x0                  ; 当读取到 Action Mark，且 Mark 编号为 4，即会执行 AM4。
```

3.20 Wave Mark

3.20.1 NY4 / NY5 / NY5+ / NX1

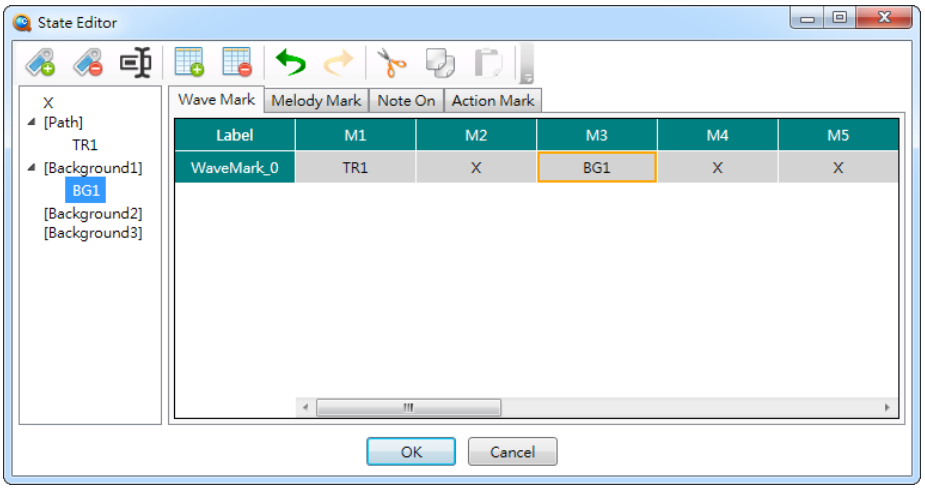
当用户使用 *Quick-IO* 来编辑文件时，用户可以随意在 Voice 文件的任何地方插入 Mark 标记。当 *Q-Code* 读到标记时，会依照当前的 Wave Mark 状态自动执行对应的路径。用户可以在 Wave Mark 段落中定义 WaveMark 的状态。

Wave Mark 的标记编号，如 M1；此标记编号最多可有 255 个，从 M1 至 M255。在 *Q-Code* 中，用户仅需要加入.nyq 档及定义好标记对应的路径即可。最多可定义 255 组设定。

注意：

1. Wave Mark 状态的切换，等同 Input State 的用法。

2. 标记编号必须为 M1~M255。

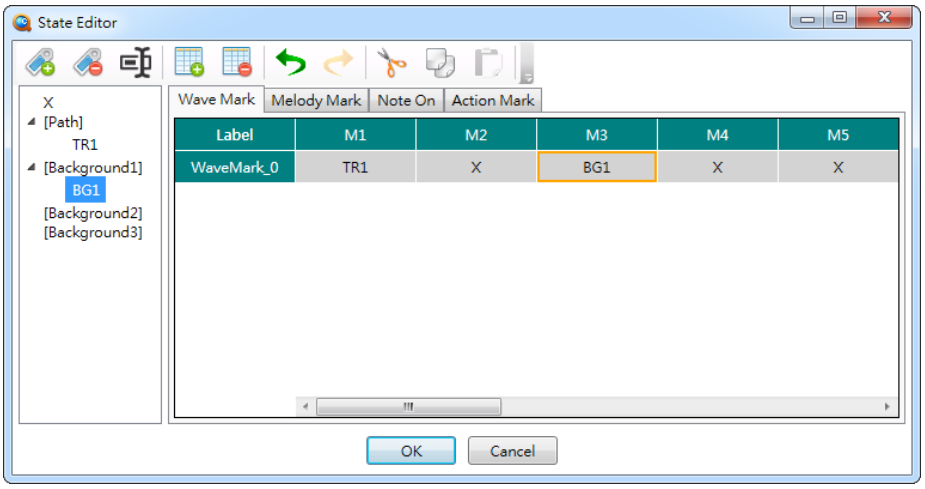


例. 使用 *Quick-IO* 在.wav 档中，插入 M1~M4 的码。

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

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

[Background1]
WM1: PB=0x1                    ; 当 QIO 发生，且 Mark 编号为 1，即会执行 WM1。
WM2: PB=0x2                    ; 当 QIO 发生，且 Mark 编号为 2，即会执行 WM2。
WM3: PB=0x3                    ; 当 QIO 发生，且 Mark 编号为 3，即会执行 WM3。
WM4: PB=0x0                    ; 当 QIO 发生，且 Mark 编号为 4，即会执行 WM4。
```

3.21 Melody Mark

3.21.1 NY5 / NY5+ / NY6 / NY7 / NX1



当用 *CakeWalk* 来编辑 MIDI 音乐文件时，用户可以随意在任何一个音符中插入控制码来控制输出脚的状态或执行其它操作。播放 Melody 文件时，按照 Melody 文件中的标记来执行相应的马达旋转或 LED 闪灯动作。

Melody Mark 动作的标记编号，如 M1；与 *CakeWalk* 或 *Q-MIDI* 中插入的控制码相对应；**[Melody Mark]** 页面中标记编号最多可有 255 个，从 M1 至 M255，每一标记编号有其相对应的路径。当定义了 MelodyMark 后，用户仅需要在 MIDI 文件中插入相应的背景控制码来启动 MelodyMark 功能。最多可以定义 255 组设定。

注意：

1. Melody Mark 状态的切换，等同 Input State 的用法。

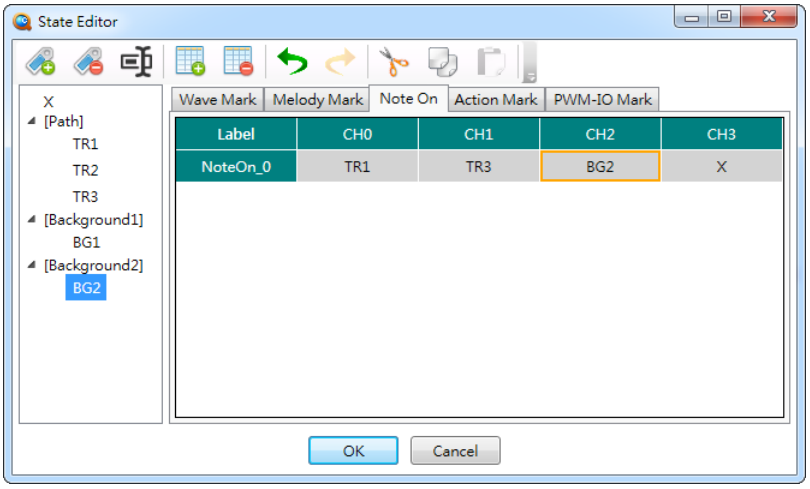
2. 标记编号必须为 M1~M255。

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

```
[MelodyMark]
;                       M1      M2      M3
NewMelodyMark1:         MM1     MM2     MM3
; 当读取到 Melody Mark 时，且 Mark 编号为 1，即会执行 MM1。
; 当读取到 Melody Mark 时，且 Mark 编号为 2，即会执行 MM2。
; 当读取到 Melody Mark 时，且 Mark 编号为 3，即会执行 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



当播放 Melody 文件时，可以不需要使用 *Q-MIDI* 来对每个 MIDI 通道的音符进行插码，来达到节省 ROM Size 的目的。使用 **[Note On]** 段落可直接在每个 MIDI 通道读取音符后，执行该 MIDI 通道所对应的动作。最多可以定义 255 组设定。

注意：

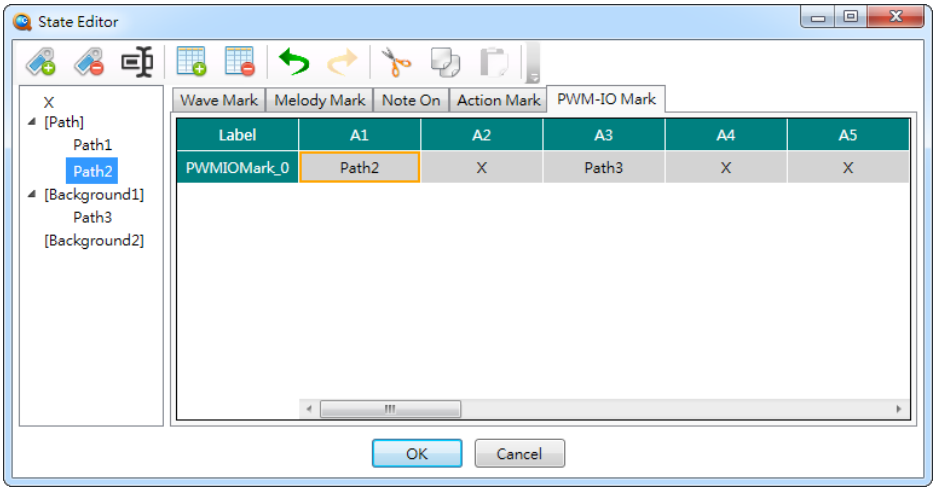
1. ***Note On 状态的切换，等同 Input State 的用法。***
2. ***NY5 系列支持 CH0 ~ CH3，NY5+系列支持 CH1 ~ CH4、CH10，NY6 系列支持 CH1 ~ CH6、CH10，NY7 / NX1 系列支持 CH1 ~ CH16。***
3. ***若是在 [Note On] 中，直接执行 play 程序，则有可能会中断当前的 Melody 播放。***
4. ***休止符不会动作。***

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        ; 切换 Note On 状态。
NO1: NoteOn1, R0=1                           ; 执行 NoteOn1。
NO2: NoteOn2, R0=0                           ; 执行 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+



当用户使用 *Q-Visio* 编辑文件时，可以随意在 action 文件中插入 Mark 标记。当 *Q-Code* 使用 PlayPWM 播放时读到标记，会依照当前的 PWM-IO Mark 状态自动执行对应的路径。用户可以在 **[PWM-IO Mark]** 段落中定义 PWM-IO Mark 的状态。

PWM-IO Mark 的标记编号最多可有 255 个，从 A1 至 A255。最多可以定义 255 组设定。

注意：

1. PWM-IO Mark 状态的切换，等同 Input State 的用法。

2. 标记编号必须为 A1~A255。

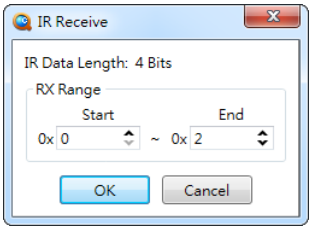
例. 在 *Q-Visio* 中，插入 A1~A4 的码。

```
[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                    ; 当读取到 PWM-IO Mark，且 Mark 编号为 1，即会执行 PM1。
PM2: PB=0x2                    ; 当读取到 PWM-IO Mark，且 Mark 编号为 2，即会执行 PM2。
PM3: PB=0x3                    ; 当读取到 PWM-IO Mark，且 Mark 编号为 3，即会执行 PM3。
PM4: PB=0x0                    ; 当读取到 PWM-IO Mark，且 Mark 编号为 4，即会执行 PM4。
```

3.24 IR Receive

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

编写 IR 接收指令。必须先在 Option 段落中选择需要的 IR Mode（红外线发射或接收功能，编码长度与载波频率）。



用户只能在 **[Option]** 段落中选择 IR 数据的编码长度，窗口会显示先前所选择的数据长度。

1. IR 数据为 4-bit 编码时，RX 的代码范围为：0x0~0xF。
2. IR 数据为 8-bit 编码时，RX 的代码范围为：0x00~0xFF。
3. IR 数据为 12-bit 编码时，RX 的代码范围为：0x000~0xFFF。
4. IR 数据为 16-bit 编码时，RX 的代码范围为：0x0000~0xFFFF。

例.

[IR Receive]

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

Code [0x0]: **PlayV**(CH0, $V1)

注意：要发射 IR，只要在 Option 选项中选择 Enable IR TX mode，然后在 Path 段落下达 TX=data。

3.25 Symbol

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

定义在条件语句和系统指令中使用的常数（Constant）或变量（Variable）。请尝试在程序中使用 Symbol，因为这样可以提高程序的可读性和灵活性。

例.

[Symbol]

On = 5

Off = 15

用户可以用自定的变量（Variable）名称来替代用户 RAM 的名称（Ri / Xi）。

例. Counter 为 8-bit 变数，CounterH 为 Counter 的高位 nibble，CounterL 为 Counter 的低位 nibble。

[Symbol]

Counter = X0

CounterL= R0

CounterH = R1

3.26 Variable

3.26.1 NX1

变量段落是用于定义变量名称及数据大小。可用的数据型态如下表：

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

可用变量名称为 A-Z 起头，第二字符开始可使用 A-Z、0-9 及底线，不可与保留字冲 0 突。在 NX1 所使用的变量皆须于此段落宣告，并定义其数据大小。

例. Buf0、Buf1 为 8-bit 变数，Buf2 为 16-bit 变数。

```
[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。

注意: 当使用 COC 功能时，第二版之后的程序所使用的 Variable 不允许进行变更（包括数量、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 段落可以定义以二维方式排列用于查找操作的 table。针对 NY4 / NY5 / NY5+ / NY6 / NY9T 每一个数值可以是 0x000 到 0x3FF 的任意整数，针对 NY7 每一个数值可以是 0x000 到 0xFFF 的任意整数，

针对 NX1 每一个数值可以是 0 ~ 0xFFFFFFFF (32-bit)。对于 4-bit 指令来说，排列方式不能够超过 16x16 数组，而对于 8-bit 指令来说，排列方式不能够超过 256x256 数组，十进制和十六进制的数值都可以在 *Q-Code* 中使用。

例.

```
[Table]
Trans:
{
    [0x1, 0x3, 0x005, 0x7, 0x09],
    [0x4, 0x5, 0x3F6, 0x8, 0x10],
    [0x0, 0x1, 0x002, 0x3, 0xF4]
}
[Path]
P1: R0=2, R1=1, TableL(trans,R0,R1,R2)     ; R2 等于 0x6。
P2: R0=2, R1=1, TableM(trans,R0,R1,R2)     ; R2 等于 0xF。
P3: R0=2, R1=1, TableH(trans,R0,R1,R2)     ; R2 等于 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 b9~b8。

Table_Name: {Table 名称}

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

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

Return_Reg: {RAM}

注意：

1. 数组必须要填满，该地址若是空的，则 Q-Code 会自动填 0。

2. 当所读取的地址超出数组大小时则会读到 0。

3.29 ASM

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

有时当用户开发某些项目时，由于 *Q-Code* 是一种简易的高级语言，其对于时序控制的准确性及效率等可能未必能满足用户的需要。此时，用户可考虑在 ASM 段落中直接编写一些汇编语言子程序来满足这个需求，而不用重新以汇编语言编写整个程序。在 ASM 段落内编写的汇编语言子程序，将可以在 Path，Background1 和 Background2 等段落中调用。

由于这些汇编语言子程序有别于一般的 *Q-Code* 语法，所以在编写这些代码时必须将汇编语言子程序的名称和“{”“}”以及每句汇编语言各自置于一个独立行中。

语法：

```
[ASM]
```

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

例.

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

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

注意：由于 Q-Code 的 User RAM 采取动态配置的方式，所以，当用户要在 Q-Code 中插入 ASM，则需要看 Q-Code 的 Build Message 中，该 RAM 被定义至哪一个 PAGE。用户在 Assembly Code 中需要自行切换 RAM PAGE。

3.30 C-Code

3.30.1 NX1

C 语言段落允许编写任意 C 语言程序代码，此段落内的文字将会完整的传递给 C 语言编译程序。Q-Code 仅解析其中的函数定义，在 **[Path]** 可以直接的调用 C 语言段落内定义的函数。

关于大小写的处理，C 语言是有大小写区分，而 Q-Code 无大小写分别。当 C 语言段落定义了两个同名、大小写不相符的函数，Q-Code 在处理 **[Path]** 内函数调用将永远配对到第一个出现的 C 语言函数。建议 C 语言段落不要定义大小写不同的同名函数。C 语言区段的程序可以直接存取变量段落（Variable），是用于定义变量名称及数据大小。可用的数据型态如下表：

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

可用变量名称为 A-Z 起头，第二字符开始可使用 A-Z、0-9 及底线，不可与保留字 0 冲突。在 NX1 所使用的变量皆须于此段落宣告，并定义其数据大小。

例. Buf0、Buf1 为 8-bit 变数，Buf2 为 16-bit 变数。

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

存取变量时，必须使用小写字母。

实际范例请参考 5.4.2。

3.31 Macro

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

当用户开发项目时，为了节省程序空间。可考虑在 Macro 段落中直接编写一些 *Q-Code* 程序来满足这个需求，而不用每次在程序段中写相同的程序来完成项目。在 Macro 段落内编写的 *Q-Code* 程序，将可以在 Path，Background1，Background2 和 Background3 等段落中直接使用。

每一个宏皆由 path 名称加上一个冒号":"及一个或数个步骤（steps）所组成。

注意：Macro 段落中的程序将会用宏的方式展开，每次展开皆会占用 ROM Size。

在以下范例中，Macro 段落中的程序可在前景或是背景中调用，且可调用相同的 Macro；也可以利用 Switch 指令来进行调用。

例.

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

[Path]
PowerOn: KEY1, R0=0
TR1: SWITCH(R0)=[SW1, SW2, SW3, SW4]  ; R0 的内容值来决定要跳转的 Label。
SW1: R0=1, MACRO1                    ; 调用 Macro1。
SW2: R0=2, BG1                       ; 调用 BG1。
SW3: R0=3, BG2                       ; 调用 BG2。
SW4: R0=0, PLAYV($V1)                ; 播放 PlayV。
TR2: VOICE? MACRO4, PB=0             ; Voice 播放中，则执行 Macro4 完后，继续执行接在
                                     ; 后面的指令。

[Background1]
BG1: MACRO2                          ; 调用 Marco2。

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

3.32 Sentence

当用户开发项目时，为了节省程序空间。可考虑在 Sentence 段落中直接编写一些组合子程序来满足此需求，而不用每次在程序段中写相同的程序来完成项目。在 Sentence 段落内编写的 *Q-Code* 程序，将可以使用 PlayS 指令在 Path，Background1，Background2 和 Background3 段落中直接使用。

3.32.1 NY4 / NY5

Sentence 组合句子皆由 path 名称加上一个冒号":"及一个或数个步骤（steps）所组成。

例.

```
[Sentence]
S1: PLAYV($V0), DELAY(0.1), PLAYV($V1), DELAY(0.2), PLAYV($V2)
S2: PLAYV($V3), PLAYV($V4), PLAYV($V5)
S3: PLAYV($V6), DELAY(0.1), PLAYV($V7)

[Path]
PowerOn: KEY1
TR1: PlayS($S1)                    ; 播放 Sentence "S1"。

[Background1]
BG1: PlayS($S1), PlayS($S3)        ; 播放 Sentence "S1"与 Sentence "S3"。

[Background2]
BG3: PlayS($S1), PlayS($S2)        ; 播放 Sentence "S1"与 Sentence "S2"。
```

3.32.2 NY5+ / NY6 / NY7 / NX1

Sentence 组合句子皆由 path 名称加上一个冒号":"及一个或数个步骤（steps）所组成。

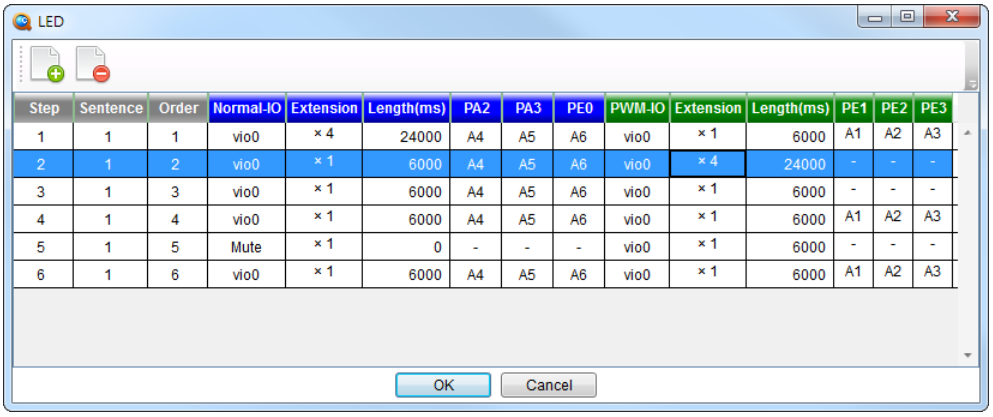
注意：句子功能主要用来将多个资源文件（Melody、Speech 及 Action 等）组合成一个句子使用，因此 Sentence 段落中只允许 Play 及 Delay 等指令，其余指令不可使用在 Sentence 段落内。

例.

```
[Sentence]
S1: PLAYV($V0), DELAY(0.1), PLAYV($V1), DELAY(0.2), PLAYV($V2)
S2: PLAYV($V3), PLAYV($V4), PLAYV($V5)
S3: PLAYV($V6), DELAY(0.1), PLAYV($V7)

[Path]
PowerOn: KEY1
TR1: PlayS($S1)                    ; 播放 Sentence "S1"。

[Background1]
BG1: PlayS($S1), PlayS($S3)        ; 播放 Sentence "S1"与 Sentence "S3"。

[Background2]
BG3: PlayS($S1), PlayS($S2)        ; 播放 Sentence "S1"与 Sentence "S2"。
```

3.32.3 NY9T

提供编排 Action 及 PWM 输出用的 LED Sentence 编辑工具。



鼠标右键菜单如下表所示：

菜单选项	功能描述	热键
New Step	在全部段落的最后新增一个 LED 动作格	＋
Insert Step	在选取的 LED 动作格上插入一个 LED 动作格	Insert
Delete Step	移除选取的 LED 动作格	－, Delete
Delete Sentence	移除选取的 LED 组合	-
Remove All	移除全部的 LED 组合	-

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”与 Sentence “L3”及”L4”。

[Background2]

BG2: PlayS($L3), PlayS($L2), L4 ; 播放 Sentence “L3”与 Sentence “L2”及”L4”。

3.33 Subroutine

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

当用户开发项目时，为了节省程序空间。可考虑在 Subroutine 段落中直接编写一些 *Q-Code* 子程序来满足这个需求，而不用每次在程序段里写相同的程序来完成项目。在 Subroutine 段落内编写的 *Q-Code* 程序，将可以在 Path，Background1，Background2 和 Background3 段落中直接使用。

Subroutine 子程序皆由 path 名称加上一个冒号":"及一个或数个步骤（steps）所组成。

在以下范例中，BG1 与 BG2 都是调用相同的 Subroutine。但是 *Q-Code* 会自动将 Program 展开成 Background1 与 Background2 的子程序。不需要用户自己写两种不同层级的子程序。

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+

提供让用户自行对 QIO / Wave Mark 来进行译码。当使用此功能时，用户必须使用汇编语言来编译 QIO 译码程序，原本 *Q-Code* 中的译码程序将会被替换。

注意：

1. 当使用 QIO Custom 时，会套用至整个项目，用户必须自己控制 QIO 的解码及输出流程。

2. 要使用该功能必须对 QIO 的格式及 Q-Code 架构有一定程度的了解，详情请联系九齐科技。

3.35 Path

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

段落中所包含的路径（paths）是用来描述 *Q-Code* 程序执行过程中的流程，每一个流程称之为一个路径。路径可分为两种类型，一种是前景路径（Path 或 Foreground Path），多数用来播放 Voice 或 Melody；另一种是背景路径（Background Path），用来配合前景路径做一些背景动作。

每一个前景路径（Foreground path）皆由 path 名称加上一个冒号":"及一个或数个步骤（steps）所组成。一个 path 的执行通常是由在其他段落中定义的外部触发事项或由其它 path 直接调用所引起的。

注意：路径段落中的每个路径在执行时，若是没有出现 PlayV，PlayS，Delay，Stop，StopV 指令，来停止当前的播放。其余指令执行完后，皆会返回原本执行 Play 或 Delay 指令的位置，来执行下一个指令。在 Q-Code 系统中，不需要额外的特殊路径来做返回动作。

例.

```
[Input State]
KEY1:   TR1R      TR2R

[Path]
PowerOn: KEY1
TR1R: PlayV($V0), Delay(1)        ; TR1 播放$V0 后，执行延迟 1 秒。
TR2R: R0=5                        ; 执行 R0=5 完毕后，若是 PlayV 或 Delay 仍在执行，则会返回。
```

3.36 Background1

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

背景路径，Background1 是用来定义第 1 个背景路径，每个背景路径（Background Path）皆由 path 名称加上一个冒号“：”及一个或数个步骤（steps）所组成。一个 path 的执行通常是由前景路径（Foreground Path）中的外部触发事项或由其它背景路径（Background Path）直接调用所引起的。

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

此段落与 Background1 用法相同，Background2 是用来定义第 2 个背景路径。

例.

```
[Path]
P0: R0=0, PlayV(Indian, 8K) & [X, BG2]
P1: BG(X, BG2)

[Background2]
BG2: PlayV($V0, 12k)
```

3.38 Background3

3.38.1 NX1

此段落与 Background1 用法相同，Background3 是用来定义第 3 个背景路径。

例.

[Path]

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

P1: BG(X, X, BG3)

[Background3]

BG3: PlayV($V0, 12k)

3.39 Background4

3.39.1 NX1

此段落与 Background1 用法相容，Background4 是用来定义第 4 个背景路径。

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

此段落与 Background1 用法相容，Background5 是用来定义第 5 个背景路径。

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

中断段落同路径段落，中断路径以及会使用到的路径需在这个段落中。

在 NX1 中，中断发生时的程序必须位于 ROM 中，当使用 XIP 或是 COC 时，*Q-Code* 会将路径放至 SPI

Flash，而当中断路径呼叫时有可能出现问题。

例. 中断呼叫有可能发生错误

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

[Path]
PowerOn: TMR_On(TMR0, 256us)        ; 开启 TMR0 中断。
Int_TMR0: Var0=1?Path1
Path1: …                            ; Q-Code 会将 Path1 放至 SPI Flash 中。
                                    ; 当 TMR0 中断发生时执行 Path1，程序可能发生错误。
```

因此将中断路径放以及相关路径放至中断段落中，Q-Code 会确保将其置于 ROM 中，避免程序的不正常行为。正确的写法如下：

例.

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

[Path]
PowerOn: TMR_On(TMR0, 256us)        ; 开启 TMR0 中断。

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

以下是适用的中断路径：

- 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 特殊路径（Q-Code Special Paths）

4.1 通用路径

4.1.1 开机（PowerOn）

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

“**PowerOn**”路径名称必须出现在 **[Path]** 段落中，并且是 *Q-Code* 程序开始执行的起始语句。当 IC 的电源打开后，“**PowerOn**”路径是 **[Path]** 段落中首先被执行的语句。

注意：PowerOn 为系统保留路径，用户不得设置相同的路径名称。

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 开机前（Before_PowerOn）

[NX1]

“**Before_PowerOn**”路径名称仅能出现在 **[Path]** 段落中，“**Before_PowerOn**”路径会在“**PowerOn**”之前执行。当 XIP(Execute in place)功能打开，“**PowerOn**”路径的程序将会存放于 SPI Flash，而“**Before_PowerOn**”路径则存放于 IC 内部 ROM。如果用户所选用的 SPI Flash 芯片需要以特殊指令初始化，可以在“**Before_PowerOn**”撰写初始化 SPI Flash 的指令。

一旦“**Before_PowerOn**” 执行结束后，系统会自动执行“**PowerOn**”。用户不须要在“**Before_PowerOn**”的结尾自行跳转到“**PowerOn**”。

例. 在系统启动前对 SPI Flash 进行初始化。

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

4.1.3 主循环（Main）

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

“**Main**”路径名称仅能出现在 **[Path]** 段落中，每次主循环执行完，系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多指令，执行过多容易让系统执行效率降低。***
2. ***由于系统需要执行其它的功能，且依照所使用的功能不同，所以该时间不会每次都相同。***

3. 不允许在此路径直接执行任何播放或 Delay 指令，但可执行条件判断指令。

例.

```
[Path]
PowerOn:
Main: !PA                                   ; 每此主循环执行完，PA 就反相输出。
```

4.1.4 休眠（Sleep）

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

“**Sleep**”路径名称仅能出现在 **[Path]** 段落中，每次 IC 进入休眠前，皆会处理该路径一次。

例.

```
[Path]
PowerOn: PB=0xF, DELAY(1)                   ; Power On 后，PB 输出 1 秒后，进入休眠。
SLEEP: PB=0                                 ; 进入休眠前将 PB 输出设为低电平。
```

4.1.5 唤醒（WakeUp）

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

“**WakeUp**”路径名称仅能出现在 **[Path]** 段落中，每次 IC 被唤醒后，皆会执行该路径一次。

注意：若有使用 I/O 扩展芯片，需延迟 1ms 才能对 I/O 扩展芯片进行操作。

例.

```
[Path]
PowerOn: KEY1                               ; Power On 后，立即进入休眠。
WakeUp: PB=0xF                              ; 每次 IC 被唤醒后，将 PB 设为高电平。
```

4.2 定时器路径

4.2.1 512 微秒（512us）

[NY7]

“**512us**”路径名称仅能出现在 **[Path]** 段落中，每次 512us 时间，系统会处理该路径一次。

注意：

1. 请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低。

2. 由于系统还需要执行其它的功能，所以该 512us 并不会非常准确。

3. 不可使用的指令请参考特殊路径中不可使用的功能。

例.

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

4.2.2 1 毫秒（1ms）

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

“**1ms**”路径名称仅能出现在 **[Path]** 段落中，每次 1ms 时间，系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低。***
2. ***由于系统还需要执行其它的功能，所以该 1ms 并不会非常准确。***
3. ***不可使用的指令请参考特殊路径中不可使用的功能。***
4. ***NY9T 使用 1ms 路径后，一定要执行 1ms_RET 指令，否则系统会出现不能预期的错误。***
5. ***NY9T 若 Timebase=4ms，则 1ms 路径不会被执行。***

例.

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

4.2.3 2 毫秒（2ms）

[NY5+ / NY6 / NY7 / NX1]

“**2ms**”路径名称仅能出现在 **[Path]** 段落中，每次 2ms 时间，系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低。***
2. ***由于系统还需要执行其它的功能，所以该 2ms 并不会非常准确。***
3. ***不可使用的指令请参考特殊路径中不可使用的功能。***

例.

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

4.2.4 4 毫秒（4ms）

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

“**4ms**”路径名称仅能出现在 **[Path]** 段落中，每次 4ms 时间，系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低。***
2. ***由于系统还需要执行其它的功能，所以该 4ms 并不会非常准确。***
3. ***不可使用的指令请参考特殊路径中不可使用的功能。***
4. ***在 NY9T 中使用 4ms 路径，一定要执行 4ms_RET 指令，否则系统将会出现不能预期的错误。***

例.

[Path]

PowerOn:

4ms: !PA ；每 4ms，PA 就反相输出。

4.2.5 8 毫秒（8ms）

[NY5+ / NY6 / NY7 / NX1]

“**8ms**”路径名称仅能出现在 **[Path]** 段落中，每次 8ms 时间，系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低。***
2. ***由于系统还需要执行其它的功能，所以该 8ms 并不会非常准确。***
3. ***不可使用的指令请参考特殊路径中不可使用的功能。***

例.

[Path]

PowerOn:

8ms: !PA ；每 8ms，PA 就反相输出。

4.2.6 500 毫秒（500ms）

[NY9T / NX1]

“**500ms**”路径名称仅能出现在 **[Path]** 段落中，每次 500 毫秒时间，系统会处理该路径一次。

例.

[Path]

PowerOn:

500ms: R0++, R0=2?Timer_1sec ；每 500 毫秒 R0 就计数一次，计数至 2 次表示 1 秒钟。

Timer_1sec: R0=0, User Program.....

注意：

1. ***用于长时间的计时，可利用内部 16KHz 的 RC 定时器获得较佳的省电效能。由于 16KHz 的 RC 定时器存在某种程度的误差，若是需要较为精确的计时，请勿使用 16KHz 的 RC 定时器功能来计时。***
2. ***不可使用的指令请参考特殊路径中不可使用的功能。***

4.2.7 1 秒（1sec）

[NY9T]

“**1sec**”路径名称仅能出现在 **[Path]** 段落中，每次 1 秒时间，系统会处理该路径一次。

例.

[Path]

```
PowerOn:
1sec: X0++, X0=60?Timer_1min                ; 每 1 秒 X0 就计数一次，计数至 60 次表示 1 分钟。
Timer_1min: X0=0, User Program.....
```

注意：

1. *用于长时间的计时，可利用内部 16KHz 的 RC 定时器获得较佳的省电效能。由于 16KHz 的 RC 定时器存在某种程度的误差，若需要较为精确的计时，请勿使用 16KHz 的 RC 定时器功能来计时。*
2. *不可使用的指令请参考特殊路径中不可使用的功能。*

4.2.8 4 秒（4sec）

[NY9T]

“**4sec**”路径名称仅能出现在 **[Path]** 段落中，每次 4 秒时间，系统会处理该路径一次。

例.

```
[Path]
PowerOn:
4sec: R0++, R0=15?Timer_1min                ; 每 4 秒 R0 就计数一次，计数至 15 次表示 1 分钟。
Timer_1min: R0=0, User Program.....
```

注意：

1. *用于长时间的计时，可利用内部 16KHz 的 RC 定时器获得较佳的省电效能。由于 16KHz 的 RC 定时器存在某种程度的误差，若是需要较为精确的计时，请勿使用 16KHz 的 RC 定时器功能来计时。*
2. *不可使用的指令请参考特殊路径中不可使用的功能。*

4.3 中断路径

4.3.1 中断（Interrupt）

[NY5 / NY5+ / NY7]

“**Interrupt**”路径名称仅能出现在 **[Path]** 段落中，每次 IC 进入中断，皆会处理该路径一次。

注意：

1. *由于 IC 仅提供 1 阶的中断，若选择使用中断来执行 PlayA / PlayAS、PlayM / PlayMS 及 QIO 时，一样会执行 Interrupt 路径中的指令，用户在使用时需要特别注意!!*
2. *NY5 中，若是执行 PlayA / PlayAS、PlayM / PlayMS 及 QIO 功能后，原本设定的中断时间将会被系统修改。*
3. *不可使用的指令请参考特殊路径中不可使用的功能。*

例.

```
[Path]
PowerOn: KEY1, INT_ON, INT=1.024
Interrupt: !PA                              ; 每 1ms 发生时间中断一次，且将 PA 反相输出。
```

4.3.2 256 微秒中断路径（Int_256us）

[NY6]

“**Int_256us**”路径名称仅能出现在 **[Path]** 段落中，当 256us 中断发生时，系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。***
2. ***不可使用的指令请参考特殊路径中不可使用的功能。***

例.

```
[Path]
PowerOn:
Int_256us: !PA                    ; 当 256us 中断发生时，PA 就反相输出。
```

4.3.3 512 微秒中断路径（Int_512us）

[NY6]

“**Int_512us**”路径名称仅能出现在 **[Path]** 段落中，当 512us 中断发生时，系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。***
2. ***不可使用的指令请参考特殊路径中不可使用的功能。***

例.

```
[Path]
PowerOn:
Int_512us: !PA                    ; 当 512us 中断发生时，PA 就反相输出。
```

4.3.4 1 毫秒中断路径（Int_1ms）

[NY6]

“**Int_1ms**”路径名称仅能出现在 **[Path]** 段落中，当 1ms 中断发生时，系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。***
2. ***不可使用的指令请参考特殊路径中不可使用的功能。***

例.

```
[Path]
PowerOn:
Int_1ms: !PA                      ; 当 1ms 中断发生时，PA 就反相输出。
```

4.3.5 16 毫秒中断路径（Int_16ms）

[NY6]

“**Int_16ms**”路径名称仅能出现在 **[Path]** 段落中，当 16ms 中断发生时，系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。***
2. ***不可使用的指令请参考特殊路径中不可使用的功能。***

例.

```
[Path]
PowerOn:
Int_16ms: !PA                    ; 当 16ms 中断发生时，PA 就反相输出。
```

4.3.6 比较器中断路径（Int_CMP）

[NY6B / NY6C]

“**Int_CMP**”路径名称仅能出现在**[Path]** 段落中，当 Comparator 中断发生时，系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。***
2. ***不允许在此路径直接执行任何播放或 Delay 指令，但可执行条件判断指令。***

例.

```
[Path]
PowerOn:
Int_CMP: !PA                     ; 当 CMP 中断发生时，PA 就反相输出。
```

4.3.7 定时器中断路径（Int_TMR）

[NY6]

“**Int_TMR**”路径名称仅能出现在 **[Path]** 段落中，当 Timer 中断发生时（Timer 时间可由程序设定），系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。***
2. ***不允许在此路径直接执行任何播放或 Delay 指令，但可执行条件判断指令。***

例.

```
[Path]
PowerOn:
Int_TMR: !PA                     ; 当 TMR 中断发生时，PA 就反相输出。
```

4.3.8 定时器 0 中断路径（Int_TMR0）

[NY5+ / NX1]

NY5+系列，“**Int_TMR0**”路径名称仅能出现在 **[Path]** 而在 NX1 系列仅能出现在 **[Interrupt]** 段落中。

当 Timer0 中断发生时（Timer 时间可由程序设定），系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。***
2. ***不允许在此路径直接执行任何播放或 Delay 指令，但可执行条件判断指令。***
3. ***NY5+指定的时间范围为 128us ~ 256us。***
4. ***NX1 OTP 计时时间随系统频率而有不同：***
   - ***High Clock Frequency 为 12MHz 时，为 64 ~ 5461us。***
   - ***High Clock Frequency 为 16MHz 时，为 64 ~ 4096us。***
   - ***High Clock Frequency 为 24MHz 时，为 64 ~ 2730us。***
   - ***High Clock Frequency 为 32MHz 时，为 64 ~ 2048us。***
5. ***NX1 EF 支援 64 ~ 1365us 。***

例.

```
[Path]
PowerOn:
TR0: TMR_ON(TMR0, 128us)
Int_TMR0: !PA.0                    ; 每 128us 触发一次，当 Int_TMR0 触发，PA.0 就反向输出。
```

4.3.9 定时器 1 中断路径（Int_TMR1）

[NY5+ / NX1]

NY5+系列，“**Int_TMR1**”路径名称仅能出现在 **[Path]** ，而在 NX1 系列仅能出现在 **[Interrupt]** 段落中。

当 Timer1 中断发生时（Timer 时间可由程序设定），系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。***
2. ***不允许在此路径直接执行任何播放或 Delay 指令，但可执行条件判断指令。***
3. ***NY5+指定的时间范围为 128us ~ 256us。***
4. ***NX1 OTP 计时时间随系统频率而有不同：***
   - ***High Clock Frequency 为 12MHz 时，为 64 ~ 5461us。***
   - ***High Clock Frequency 为 16MHz 时，为 64 ~ 4096us。***
   - ***High Clock Frequency 为 24MHz 时，为 64 ~ 2730us。***
   - ***High Clock Frequency 为 32MHz 时，为 64 ~ 2048us。***
5. ***NX1 EF 支援 64 ~ 1365us 。***

例.

[Path]

PowerOn:

TR0: TMR_ON(1, 128us)

Int_TMR1: !PA.1 ；每 128us 触发一次，当 Int_TMR1 触发，PA.1 就反向输出。

4.3.10 定时器 2 中断路径（Int_TMR2）

[NY5+ / NX1]

NY5+系列，“**Int_TMR2**”路径名称仅能出现在 **[Path]** ，而在 NX1 系列仅能出现在 **[Interrupt]** 段落中。

当 Timer2 中断发生时（Timer 时间可由程序设定），系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。***
2. ***不允许在此路径直接执行任何播放或 Delay 指令，但可执行条件判断指令。***
3. ***NY5+指定的时间范围为 128us ~ 256us。***
4. ***NX1 OTP 计时时间随系统频率而有不同：***
   - ***High Clock Frequency 为 12MHz 时，为 64 ~ 5461us。***
   - ***High Clock Frequency 为 16MHz 时，为 64 ~ 4096us。***
   - ***High Clock Frequency 为 24MHz 时，为 64 ~ 2730us。***
   - ***High Clock Frequency 为 32MHz 时，为 64 ~ 2048us。***
5. ***NX1 EF 支援 64 ~ 1365us 。***

例.

[Path]

PowerOn:

TR0: TMR_ON(TMR2, 128us)

Int_TMR2: !PA.2 ；每 128us 触发一次，当 Int_TMR2 触发，PA.2 就反向输出。

4.3.11 定时器 3 中断路径（Int_TMR3）

[NY5+ / NX1]

NY5+系列，“**Int_TMR3**”路径名称仅能出现在 **[Path]** ，而在 NX1 系列仅能出现在 **[Interrupt]** 段落中。

当 Timer3 中断发生时（Timer 时间可由程序设定），系统会处理该路径一次。

注意：

1. ***请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。***
2. ***不允许在此路径直接执行任何播放或 Delay 指令，但可执行条件判断指令。***
3. ***NY5+指定的时间范围为 128us ~ 256us。***
4. ***NX1 OTP 计时时间随系统频率而有不同：***
   - ***High Clock Frequency 为 12MHz 时，为 64 ~ 5461us。***

- *High Clock Frequency 为 16MHz 时，为 64 ~ 4096us。*
- *High Clock Frequency 为 24MHz 时，为 64 ~ 2730us。*
- *High Clock Frequency 为 32MHz 时，为 64 ~ 2048us。*

5. *NX1 EF 支援 64 ~ 1365us。*

例.

[Path]

PowerOn:

TR0: TMR_ON(TMR3, 128us)

Int_TMR3: !PA.3 ；每 128us 触发一次，当 Int_TMR3 触发，PA.3 就反向输出。

4.3.12 PWMA 中断路径（Int_PWMA）

[NX1]

“**Int_PWMA**”路径名称仅能出现在 **[Interrupt]** 段落中，当 PWMA 定时器中断发生时（Timer 时间可由程序设定），系统会处理该路径一次。

注意：

1. *请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。*
2. *不允许在此路径直接执行任何播放或 Delay 指令，但可执行条件判断指令。*
3. *NX1 OTP 计时时间随系统频率而有不同：*
   - *High Clock Frequency 为 12MHz 时，为 64 ~ 5461us。*
   - *High Clock Frequency 为 16MHz 时，为 64 ~ 4096us。*
   - *High Clock Frequency 为 24MHz 时，为 64 ~ 2730us。*
   - *High Clock Frequency 为 32MHz 时，为 64 ~ 2048us。*
4. *NX1 EF 支援 64 ~ 1365us。*

例.

[Path]

TR0: TMR_ON(PWMA, 128us)

[Interrupt]

Int_PWMA: !PA.3 ；当 PWMA 定时器中断每 128us 触发，PA.3 就反向输出。

4.3.13 PWMB 中断路径（Int_PWMB）

[NX1]

“**Int_PWMB**”路径名称仅能出现在 **[Interrupt]** 段落中，当 PWMB 定时器中断发生时（Timer 时间可由程序设定），系统会处理该路径一次。

注意：

1. *请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。*

2. *不允许在此路径直接执行任何播放或 Delay 指令，但可执行条件判断指令。*
3. *NX1 OTP 计时时间随系统频率而有不同：*
   - *High Clock Frequency 为 12MHz 时，为 64 ~ 5461us。*
   - *High Clock Frequency 为 16MHz 时，为 64 ~ 4096us。*
   - *High Clock Frequency 为 24MHz 时，为 64 ~ 2730us。*
   - *High Clock Frequency 为 32MHz 时，为 64 ~ 2048us。*
4. *NX1 EF 支援 64 ~ 1365us。*

例.

```
[Path]
TR0: TMR_ON(PWMB, 128us)
[Interrupt]
Int_PWMB: !PA.3                    ; 当 PWMB 定时器中断每 128us 触发，PA.3 就反向输出。
```

4.3.14 实时时钟中断路径（RTC_2Hz / RTC_64Hz / RTC_1024Hz / RTC_4096Hz / RTC_16384Hz）

[NX1]

“**RTC_2Hz / RTC_64Hz / RTC_1024Hz / RTC_4096Hz / RTC_16384Hz**”路径名称仅能出现在 **[Interrupt]** 段落中，当 RTC 中断发生时，系统会处理该路径一次。

注意：

1. *请勿在此路径底下执行过多的指令，执行过多容易让系统执行效率降低并影响其他中断的发生。*
2. *不允许在此路径直接执行任何播放或 Delay 指令，但可执行条件判断指令。*
3. *RTC 中断由系统初始化时自动启用，无需开关。*
4. *RTC_4096Hz 仅 NX12/3P44A 与 NX11P22A 支援。RTC_16384Hz 除了 NX12/3P44A 与 NX11P22A 以外，其他 NX1 可支持。*

例.

```
[Interrupt]
RTC_2Hz: !PA.6                     ; 每秒触发两次，当 RTC_2Hz 触发，PA.6 就反向输出。
```

4.4 电压侦测路径

4.4.1 特定电压侦测（LVD_mVn / LVD_Max）

[NY4PxxxC / NY5+ / NY6P025A / NY6B / NY6C / NX1]

“**LVD_mVn**” / “**LVD_Max**”路径名称仅能出现在 **[Path]** 段落中，当 VDD 变化至指定范围时，系统会处理该路径一次。

不同系列 IC 可支持的电压侦测系统路径及对应的电压范围如下：

电压范围	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. *若上电后即触发此路径，则 PowerOn 路径可能会执行不完全。*
2. *以 NY5+为例，当指定 LVD_2V8 及 LVD_2V4 时，LVD_2V4 也涵盖<2.0V ~ 2.4V 的变化。*

4.4.2 电压侦测（LVD）

[NY5+ / NY6B / NY6C / NX1]

“**LVD**”路径名称仅能出现在 **[Path]** 段落中，当 VDD 电压在不同 LVD 阶数变化时，系统会处理该路径一次。

例.

```
[Path]
PowerOn:
LVD: PA=1                    ; 当电压在不同 LVD 阶数间变化时，PA 输出 1。
```

4.5 資料傳輸路徑

4.5.1 红外线接收（IR_RX）

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

“**IR_RX**”路径名称仅能出现在 **[Path]** 段落中，每当 IR 完成接收动作，皆会执行该路径一次，用户可在此路径用指令读取 IR 内的数据。

例.

```
[Path]
```

```
IR_RX: [R1, R0] = IR_RX                    ; 在接收到 IR 数据时，将数据存到 R0 及 R1。
```

4.5.2 串行数据接收（SC_RX）

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

“**SC_RX**”路径名称仅能出现在 **[Path]** 段落中，每当 SC 完成接收动作，皆会执行该路径一次，用户可在此路径用指令读取 SC 内的数据。

例.

```
[Path]
SC_RX: [R1, R0] = SC_RX                    ; 在接收到 SC 数据时，将数据存到 R0 及 R1。
```

4.5.3 I2C 数据接收（I2C_RX）

[NY5+ / NX1]

“**I2C_RX**”路径名称仅能出现在 **[Path]** 段落中，每次 I2C 接收到一笔数据，皆会处理该路径一次，用户在此路径用指令读取 I2C 接收到的数据。

例.

```
[Path]
I2C_RX: [R1, R0] = I2C_RX                  ; 在接收到 I2C 数据时，将数据存到 R0 及 R1。
```

4.5.4 I2C 数据传输完成（I2C_TX_Ok）

[NY5+ / NX1]

“**I2C_TX_Ok**”路径名称仅能出现在 **[Path]** 段落中，每次 I2C 传送完成一笔资料，皆会处理该路径一次，用户在此路径可以判断是否有 ACK。

例.

```
[Path]
I2C_TX_OK: I2C_Ack?Path1                   ; I2C 数据传送完成，判断是否有响应 ACK 信号。
Path1:     I2C_TX(0x5A)                    ; 有 ACK 信号则传送 0x5A 的资料。
```

4.5.5 I2C_Error

[NX1]

“**I2C_Error**”路径名称仅能出现在 **[Path]** 段落中，当 I2C 因 Clock Stretching 发生 Timeout，或 I2C Bus arbitration 失败等状况发生时，皆会处理该路径一次。

例.

```
[Path]
I2C_Error: I2C_Reset                       ; I2C 数据传送失败，将 I2C 传输重设。
```

4.5.6 UART 数据接收（UART_RX）

[NY5+]

“**UART_RX**”路径名称仅能出现在 **[Path]** 段落中，每次 UART 接收到一笔数据，皆会处理该路径一次，用户在此路径用指令读取 UART 接收到的数据。

例.

[Path]

UART_RX: [R1, R0] = UART_RX ；在接收到 UART 数据时，将数据存到 R0 及 R1。

4.5.7 WaveID 数据接收（WaveID_RX）

[NX1]

“**WaveID_RX**”路径名称仅能出现在 **[Path]** 段落中，每次 WaveID 接收到一笔数据，皆会处理该路径一次，用户在此路径用指令读取 WaveID 的数据。

例.

[Path]

WaveID_RX: R0= WaveID_RX ；在接收到 WaveID 数据时，将数据存到 R0。

4.5.8 Sound Localization 数据接收（SL_RX）

[NX1]

“**SL_RX**”路径名称仅能出现在 **[Path]** 段落中，每次 Sound Localization 接收到一笔数据，皆会处理该路径一次，用户在此路径用指令读取声音来源的角度。

例.

[Path]

SL_RX: R0= SL_RX ；在接收到 Sound Localization 数据时，将侦测到的角度存到 R0。

4.6 触控路径

4.6.1 强制校正（Enforce_Calibrate）

[NY9T]

“**Enforce_Calibrate**”路径名称仅能出现在 **[Path]** 段落中，每次 Enforce_Calibrate 执行完毕后，皆会处理该路径一次。

注意：

1. ***当 SW_Reset 功能关闭后，强制校正路径方可执行。***
2. ***若用户需要针对不同的应用，而于执行完 Enforfce_Calibrate 后所需要的重置行为不同时，可将 SW_Reset 功能关闭，改由用户手动进行系统重置。***

例. 无需记忆设定或模式时，可用 SW_Reset 来进行重置。

```
[Path]
PowerOn: KEY1                          ; Power On 后，立即进入休眠。
TR1R: PE.0=1                           ; 触摸键按下时，PE.0 输出 1。
TR1F: PE.0=0                           ; 触摸键放开时，PE.0 输出 0。
Enforce_Calibrate: SW_Reset            ; 每次 Enforce_Calibrate 执行后，随即进行软件重置。
```

例. 需记忆设定或模式，且执行完 Enforce_Calibrate 后，持续维持原本状态。

```
[Path]
PowerOn: KEY1                          ; Power On 后，立即进入休眠。
TR1R: PE.0=1                           ; 触摸键按下时，PE.0 输出 1。
TR1F: PE.0=0                           ; 触摸键放开时，PE.0 输出 0。
TR2R: R0++                             ; 切换模式。
Enforce_Calibrate: TouchKey_CLR; 每次 Enforce_Calibrate 执行后，随即进行触摸键状态清除。
```

例. 需记忆设定或模式，且执行完 Enforce_Calibrate 后，只清除 IO 状态。若执行完 Enforce_Calibrate 后不清除触摸键，会自动依序执行触摸键按键放开路径。

```
[Path]
PowerOn: KEY1                          ; Power On 后，立即进入休眠。
TR1R: PE.0=1                           ; 触摸键按下时，PE.0 输出 1。
TR1F: PE.0=0                           ; 触摸键放开时，PE.0 输出 0。
TR2R: R0++                             ; 切换模式。
Enforce_Calibrate: PE=0                ; 每次 Enforce_Calibrate 执行后，随即进行 IO 状态清除。
```

例. 需记忆设定或模式，且执行完 Enforce_Calibrate 后，清除 IO 及按键状态，但不执行按键放开路径。

```
[Path]
PowerOn: KEY1                          ; Power On 后，立即进入休眠。
TR1R: PE.0=1                           ; 触摸键按下时，PE.0 输出 1。
TR1F: PE.0=0                           ; 触摸键放开时，PE.0 输出 0。
TR2R: R0++                             ; 切换模式。
Enforce_Calibrate: PE=0, TouchKey_CLR  ; 每次 Enforce_Calibrate 执行后，随即进行 IO 与触摸键状态清除。
```

4.7 SPI Flash 路径

4.7.1 擦除完成（SPI_EraseEnd）

[NY6B / NY6C / NY7 / NX1]

“**SPI_EraseEnd**”路径名称仅能出现在 **[Path]** 段落中。当 SPI Flash 擦除指令完成后，皆会处理该路径一次，以便用户知道擦除完成的时间点，进行后续动作。擦除指令包含 SPI_CE、SPI_SE 及 SPI_BE。

例.

```
[Path]
Erase: SPI_CE, R0=1
SPI_EraseEnd: R0=0    ；执行全内存擦除动作，并将 R0 设为 1，擦除完成将 R0 清为 0。
```

4.7.2 擦除逾时（SPI_EraseTimeout）

[NX1]

“**SPI_EraseTimeout**”路径名称仅能出现在 **[Path]** 段落中。当 SPI Flash 抹除等待超过 Option->SPI Flash 所设定的抹除时间，会立刻处理该路径一次，以便用户知道抹除发生逾时。此路径用于 SPI Flash 硬件异常的提示，不允许有跳转行为。

例.

```
[Path]
TR1: EraseR($Rec0)                    ；擦除位于 SPI Flash 的 Rec0 录音空间。
SPI_EraseTimeout: PA.0=1              ；发生抹除逾时，将 PA.0 设为 1。
```

4.8 语音识别路径

4.8.1 语音指令群组逾时路径（VRGC_Timeout）

[NX1]

“**VRGC_Timeout**”路径名称仅能出现在 **[Path]** 段落中，若设定语音指令群组为 VRGC1 = VRG1+VRG2，当 VRG1 之中的指令成功辨认，但并未在时限内辨认到 VRG2 的指令，将会自动执行 VRGC_Timeout 路径。

例.

```
[Path]
PowerOn:
VRGC_Timeout: PlayV(ch0, $V0)          ；在语音指令不明时，提示用户再说一次。
```

4.8.2 语音识别指令未成功（VR_Unknown）

[NX1]

“**VR_Unknown**”路径名称仅能出现在 **[Path]** 段落中，当语音识别有侦测到声音，但是却没有辨识成功时，会执行该路径一次。

注意：使用此功能，须先开启 VAD。

例.

```
[Path]
PowerOn:
VR_Unknown: PlayV(ch0, $V0)                    ; 在语音指令不明时，提示用户再说一次。
```

4.8.3 Voice Tag 录音前（VT_BeforeRecord）

[NX1]

“**VT_BeforeRecord**“路径名称仅能出现在 **[Path]** 段落中，Voice Tag 训练前会先录音数次，录音前会跳进此路径。

注意：

1. ***此路径不可再跳其它路径。***
2. ***此路径不可执行须等待的指令，除了 PlayV / PlayA / SpiPlay / SDelay。***
3. ***跳进此路径后，除了该路径的工作继续执行外，其他所有工作将暂停。***

例.

```
[Path]
VT_BeforeRecord: SpiPlay(ch0, 0)               ; Voice Tag 录音前，跳此 path。
```

4.8.4 Voice Tag 训练前（VT_BeforeTraining）

[NX1]

“**VT_BeforeTraining**“路径名称仅出现在 **[Path]** 段落中，当 Voice Tag 录音后训练前，会跳进此路径。

注意：

1. ***此路径不可再跳其它路径。***
2. ***此路径不可执行须等待的指令，除了 PlayV / PlayA / SpiPlay / SDelay。***
3. ***跳进此路径后，除了该路径的工作继续执行外，其他所有工作将暂停。***

例.

```
[Path]
VT_BeforeTraining: SpiPlay(ch0, 0)             ; Voice Tag 训练前，跳此 path。
```

4.8.5 Voice Tag 训练失败（VT_AddTagFail）

[NX1]

“**VT_AddTagFail**”路径名称仅能出现在 **[Path]** 段落中，Voice Tag 训练失败，会跳进此路径。

例.

[Path]

VT_AddTagFail: SpiPlay(ch0, 0) **; Voice Tag 训练失败，跳此 path。**

4.9 录音路径

4.9.1 琴键录音逾时（KRecord_Timeout）

[NX1]

“**KRecord_Timeout**”路径名称仅能出现在 **[Path]** 段落中，当琴键录音超过时间没有 InstNoteON / InstNoteOff 发生时，会跳进此路径。

例.

[Path]

KRecord_Timeout: PlayKS($Rec0, 0) **;发生琴键录音逾时时，使用音色 0 播放琴键录音。**

4.10 聲音偵測路徑

4.10.1 聲音偵測（Sound_Detected）

[NX1]

“**Sound_Detect**”路径名称仅能出现在 **[Path]** 段落中，当有侦测到声音时，会跳进此路径。

例.

[Option]

SoundDetect = Enable

SoundDetect_High_Threshold = 300

SoundDetect_Low_Threshold = 300

SoundDetect_Active_Time = 80ms

SoundDetect_Mute_Time = 650ms

[Path]

TR1: SoundDetect_On **; 启用声音侦测。**

Sound_Detect: PA.0 = 1 **; 侦测到声音时，PA.0 输出 1。**

Sound_Mute: PA.0 = 0 **; 当不再侦测到声音超过 150ms 时，PA.0 输出 0。**

4.10.2 聲音靜默（Sound_Muted）

[NX1]

“**Sound_Mute**”路径名称仅能出现在 **[Path]** 段落中，当不再侦测到声音时，会跳进此路径。

4.11 音高侦测路徑

4.11.1 音高侦测（Pitch_Detected）

[NX1]

“**Pitch_Detected**”路径名称仅能出现在 **[Path]** 段落中，当侦测到音高时，会跳进此路径。

例.

```
[Option]
PitchDetect = Tone                    ; 侦测 Pure-Tone。
PitchDetect_Range = Middle            ; 侦测范围 1000 ~ 3000Hz。
[Variable]
Var16: Pitch_Value
[Path]
TR1: PitchDetect_On                   ; 开启音高侦测。
TR2: PitchDetect_Off                  ; 关闭音高侦测。
Pitch_Detected: ReadPitch(Pitch_Value)  ; 侦测到 1000 ~ 3000Hz 内的 pure-tone，
                                      ; 将侦测到的音高储存至 Pitch_Value。
```

4.12 IO 扩展芯片路徑

4.12.1 IO 扩展芯片通信失败（IoExp_CommFail）

[NY5+ / NX1]

“IoExp_CommFail”路径名称仅能出现在 **[Path]** 段落中，当系统与 I/O 扩展芯片通信失败时，会跳进此路径。

例.

```
[Path]
IoExp_CommFail: R0 = IoExp_ErrId    ; 当系统与 I/O 扩展芯片通信失败时，读取通信失败的 I/O
                                    ; 扩展芯片的 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 为 **Ri**，8-bit 指令所使用的 RAM 为 **Xi**。

Xi 由两个 4-bit RAM 所组成，对应方式如下：

$$X(i) = [X(i)L, X(i)H] = [\ R(i*2), R(i*2+1)\ ]$$

例. **X10 = [R20, R21]**，其中 R20 / X10L 为 X10 低位的 nibble，R21 / X10H 为 X10 高位的 nibble。

完整的对应表请参考 Ri 与 Xi 对应表。

注意：

1. ***NY9T001A/004A 不提供 8-bit 乘除法指令。***
2. ***NX1 不预先定义 Ri / Xi 等变量，用户需自行于 [Variable]中定义所需的变量。***

5.1 算数逻辑指令（Arithmetic Logic Command）

Arithmetic Logic Command			
Var1 = Var2	Var1 = Var2 + Var3	Var1 = Var2 – Var3	Var1 = Var2 * Var3
Va1 = 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 变量运算（Variable Operation）

Q-Code 支持变量的赋值与运算操作，变量可以为：

- 1-bit 运算符：Ri.n / Xi.n / Pin
- 4-bit 运算符：Ri / Port
- 8-bit 运算符：Xi
- **[Variable]** 中定义的变量。
- 系统变量

变量之间可进行赋值及下表所列的运算操作。

No.	变数操作	注释	范例
1	Var1 = Var2	赋值操作	R0 = 0x4, R0 = R1, R0 = RandomL, R0 = RandomH
2	Var1 = Var2 + Var3	加法运算	R2 = R0 + 5, R2 = 5 + R0
3	Var1 = Var2 – Var3	减法运算	R2 = R0 – 3, R2 = 5 – R0

No.	变数操作	注释	范例
4	Var1 = Var2 * Var3 [Var1, Var2] = Var3 * Var4	乘法运算	X1 = R0 * R1, X1 = R0 * 5 [R0, R1] = R2 * 15
5	Var1 = Var2 / Var3 [Var1, Var2] = Var3 / Var4	除法运算 第二种除法运算的商数存于 Var1，余数存于 Var2	R2 = R0 / 5, X2 = X1 / R0, [R2, R1] = R0 / 5
6	Var1 = Var2 % Var3	余数运算	R2 = R1 % 15
7	Var++	递增运算	R0++
8	Var--	递减运算	PA--
9	Var1 = Var2 & Var3	AND 运算	R2 = R0 & R1
10	Var1 = Var2 | Var3	OR 运算	R2 = R0 | R1
11	Var1 = Var2 ^ Var3	XOR 运算	R2 = R0 ^ R1
12	Var1 = Var2 << Var3	左移运算	R2 = R0 << 2, R2 = R0 << R1
13	Var1 = Var2 >> Var3	右移运算	R2 = R0 >> 2, R2 = R0 >> R1
14	!Var	反向运算	!R0, !R0.3

注意：

1. 可用的系统变量如下表：

系统变量名称	说明	支援性	范例
Vol	音量	NY5 / NY5+ / NY6 / NY7 / NX1	Vol = R1
V_Chx_Vol	语音的通道音量	NY5+ / NY6 / NY7	V_Ch0_Vol = X0
M_Chx_Vol	Melody 的个别通道音量	NY5+ / NY6 / NY7 / NX1	M_Ch10_Vol = R0
SPIVol	SPIPlay 播放的语音音量	NY6B / NY6C / NY7	SPIVol = R0
Px	IO Port	全系列	PA = R0 ^ 0x9
PxM	IO Port 输出入状态	NY4 / NY5+ / NY6 / NY7 / NY9T / NX1	PAM = PFM | 0x1
PxM.n	IO 脚位输出入状态	NY4 / NY5+ / NY6 / NY7 / NY9T / NX1	PAM.0 = PFM.0
RandomL	随机数的低位 nibble 只读，不可使用于 = 左边。	全系列	R0 = RandomL ^ 0x5
RandomH	随机数的高位 nibble 只读，不可使用于 = 左边。	全系列	R0 = RandomH | 0x9
Random	随机数 只读，不可使用于 = 左边。	全系列	X0 = Random ^ 0x33

2. 1-bit / 4-bit / 8-bit 变量互相运算时若是将较高 bit 的变量或运算结果赋值于较低 bit 的变量时，多余的 bit 会被截去。

Ex. X0 = 0xFA, R3 = X0 ; R3 = 0xA

5.1.2 Var=RandomL

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

产生随机数，并将结果的 low-nibble 存入 Var。随机数范围由 Random 选项决定。

5.1.3 Var=RandomH

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

产生随机数，并将结果的 high-nibble 存入 Var。随机数范围由 Random 选项决定。

5.1.4 Var=Random

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

产生随机数，并将结果存入 Var。当未指定 Min / Max 时，随机数范围为 0 ~ Random 选项，若指定 Min / Max 时，随机数范围为 Min ~ Max - 1。

Var=Random

Var=Random({Min, }Max)

Var：用来储存随机数的变量。

Min：随机数最小值。若不指定则为 0。

Max：最大接受 0xFFFFFFFF。

注意：

1. NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 不支持设定 Min / Max 参数。

2. NX1 当指定 Min / Max 时，产生的随机数范围不受 Random 选项影响。

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	-	-

语法：

R op T ? True_Branch { : False_Branch }

R：RAM 或变数（Variable）。

Op：运算符，见下列表格。

T：R 或常数（Constant）。常数可以接受 2 进制，10 进制与 16 进制的数值。

?：判断符号。

True_Branch { : False_Branch }：可以是 Path 名称或 ASM。

{ }：中括号里面的 False 子句可以被省略。

条件跳转指令是用条件式的 True 或 False 值来选择应执行成立分支或不成立分支，若不成立分支省略时且条件式之值为 False，则会跳到目前路径的下一个步骤。

分支路径除了使用一般路径名称，亦可使用以大括号包住的一连串指令集合。指令集合会被自动展开为匿名路径。底下的范例示范此种写法，左右两边结果是等价的。

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 比较运算符（Comparison）

操作数	定 义	例 子
Var1 = Var2	Var1 等于 Var2	R0 = 2 ?TRUE, X0 = R1 ?TRUE, R0.0 = 1 ? True
Var != Var	Var1 不等于 Var2	R0 != R1?TRUE ; R0 != 3 ?TRUE ; PA.0 = 1 ?TRUE
Var >= Var	Var1 大于等于 Var2	Vol >= R1?TRUE ; R0 >= 3 ?TRUE
Var > Var	Var1 大于 Var2	Random > 128 ?TRUE ; R0 > 5 ?TRUE
Var <= Var	Var1 小于等于 Var2	R0 <= R1?TRUE ; R0 <= 5 ?TRUE
Var < Var	Var1 小于 Var2	R0 < R1 ?TRUE ; R0 = 7, R0 < 5 ?TRUE

Q-Code 支持变量的比较，变量可以为：

- 1-bit 运算符：Ri.n / Xi.n / Px.n
- 4-bit 运算符：Ri / Px
- 8-bit 运算符：Xi
- **[Variable]** 中定义的变量。
- 系统变量

注意：可用的系统变量如下表：

系统变量名称	说明	支援性	范例
Vol	音量	NY5 / NY5+ / NY6 / NY7 / NX1	Vol = R1
V_Chx_Vol	语音的通道音量	NY5+ / NY6 / NY7	V_Ch0_Vol = X0
M_Chx_Vol	Melody 的个别通道音量	NY5+ / NY6 / NY7 / NX1	M_Ch10_Vol = R0
SPIVol	SPIPlay 播放的语音音量	NY6B / NY6C / NY7	SPIVol = R0
Px	IO Port	全系列	PA = R0 ^ 0x9
PxM	IO Port 输出入状态	NY4 / NY5+ / NY6 / NY7 / NY9T / NX1	PAM = PFM | 0x1

PxM.n	IO 脚位输出入状态	NY4 / NY5+ / NY6 / NY7 / NY9T / NX1	PAM.0 = PFM.0
RandomL	随机数的低位 nibble 只读，不可使用于 = 左边。	全系列	R0 = RandomL ^ 0x5
RandomH	随机数的高位 nibble 只读，不可使用于 = 左边。	全系列	R0 = RandomH | 0x9
Random	随机数 只读，不可使用于 = 左边。	全系列	X0 = Random ^ 0x33

5.2.2 Px = data?Path

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

Px 的值为某个值时，跳至 Path。

例. **PA = 3? Path ; PA 的值为 3 时，跳至 Path。**

5.2.3 Px != data?Path

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

Px 的值不为某个值时，跳至 Path。

例. **PA != 3? Path ; PA 的值不为 3 时，跳至 Path。**

5.2.4 Px.n = 0?Path

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

Px 的某一个脚的值为 0 时，跳至 Path。

例. **PA.0 = 0? Path ; PA.0 的值为 0 时，跳至 Path。**

5.2.5 Px.n != 0?Path

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

Px 的某一个脚的值不为 0 时，跳至 Path。

例. **PA.0 != 0? Path ; PA.0 的值不为 0 时，跳至 Path。**

5.2.6 Px.n = 1?Path

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

Px 的某一个脚的值为 1 时，跳至 Path。

例. **PA.0 = 1? Path ; PA.0 的值为 1 时，跳至 Path。**

5.2.7 Px.n != 1?Path

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

Px 的某一个脚的值不为 1 时，跳至 Path。

例. **PA.0 != 1? Path ; PA.0 的值不为 1 时，跳至 Path。**

5.2.8 Vol = n?Path

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

若是目前音量等于 n 的时候，则跳转至 Path。n=0~15。

例. 若音量等于第五阶时。

Vol=5?True ;音量等于第 5 阶时，则跳转至“True”。

5.2.9 Vol != n?Path

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

若是目前音量不等于 n 的时候，则跳转至 Path。n=0~15。

例. 若音量等于第五阶时。

Vol!=5?True ; 音量不等于第 5 阶时，则跳转至“True”。

5.2.10 MixCtrl = data?Path

[NY5]

MixCtrl 设定值等于 data，则跳转到 Path。

例.

```
MixCtrl=0x2?Set_MixCtrl      ;  MixCtrl 值若是不等于 0x2，则跳转至“Set_MixCtrl”。
Set_ MixCtrl: MixCtrl(0,0,4) ; 设定成 Channel2=100%音量。
```

5.2.11 MixCtrl != data?Path

[NY5]

MixCtrl 设定值不等于 data，则跳转到 Path。

例.

```
MixCtrl!=0xE?Set_MixCtrl     ;  MixCtrl 值若是不等于 0xE，则跳转至“Set_MixCtrl”。
Set_MixCtrl: MixCtrl(1,1,2)  ; 设定成 Ch0 / Ch1=50%音量+Ch2=50%音量。
```

5.2.12 RandomL = data?Path

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

这是由随机产生器产生的随机数，随机数可在 **[Option]** 页面中设定。RandomH 提供高位的 nibble 而 RandomL 提供低位的 nibble。

随机产生器的 Low-Nibble 若是等于 data，则跳转至 Path。data=0~15。

例. 随机数产生器所产生的随机数，若是 RandomL 等于 0xA，则跳转到该路径。

```
RandomL=0xA?True                ; 取到的值若是等于 Low-Nibble 则跳转至“True”。
```

5.2.13 RandomH = data?Path

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

这是由随机产生器产生的随机数，随机数可在 **[Option]** 页面中设定。RandomH 提供高位的 nibble 而 RandomL 提供低位的 nibble。

随机产生器的 High-Nibble 若是等于 data，则跳转至 Path。data=0~15。

例. 随机数产生器所产生的随机数，若是 RandomH 等于 0x3，则跳转到该路径。

```
RandomH=0x3?True                ; 取到的值若是等于 High-Nibble 则跳转至“True”。
```

5.2.14 RandomL != data?Path

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

这是由随机产生器产生的随机数，随机数可在 **[Option]** 页面中设定。RandomH 提供高位的 nibble 而 RandomL 提供低位的 nibble。

随机产生器的 Low-Nibble 若是不等于 data，则跳转至 Path。data=0~15。

例. 随机数产生器所产生的随机数，若是 RandomL 不等于 0xA，则跳转到该路径。

```
RandomL!=0xA?True               ; 取到的值若是不等于 Low-Nibble 则跳转至“True”。
```

5.2.15 RandomH != data?Path

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

这是由随机产生器产生的随机数，随机数可在 **[Option]** 页面中设定。RandomH 提供高位的 nibble 而 RandomL 提供低位的 nibble。

随机产生器的 High-Nibble 若是不等于 data，则跳转至 Path。data=0~15。

例. 随机数产生器所产生的随机数，若是 RandomH 不等于 0x3，则跳转到该路径。

```
RandomH!=0x3?True               ; 取到的值若是不等于 High-Nibbble 则跳转至“True”。
```

5.2.16 Random = data?Path

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

这是由随机产生器产生的随机数，随机数可在 **[Option]** 页面中设定。随机产生器所产生的数值若是等于 data，则跳转至 Path。。

Random=Data?Path

Random(Max)=Data?Path

Random(Min, Max)=Data?Path

Data：number.

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 0 ~ 255。
- NX1 支持 0 ~ 0xFFFFFFFE，当有指定 Min / Max 时，则为 Min ~ Max - 1。

Min：随机数最小值。若不指定则为 0。

Max：最大接受 0xFFFFFFFF。

Note:

1. NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 不支持设定 Min / Max 参数。

2. NX1 指定 Min / Max 时，产生的随机数不受 Random 选项影响。

例. Random=0x3A

Random=0x3A?True ；取到的值若是等于 **0x3A**，则跳转至**“True”**。

例. NX1

Random(0x10, 0x1000)=0x100?True ；取到的值若是等于 **0x100**，则跳转至**“True”**。

5.2.17 Random != data?Path

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

这是由随机产生器产生的随机数，随机数可在 **[Option]** 页面中设定。随机产生器所产生的数值若是不等于 data，则跳转至 Path。

Random != Data?Path

Random(Max) != Data?Path

Random(Min, Max) != Data?Path

Data：number.

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 0 ~ 255。
- NX1 支持 0 ~ 0xFFFFFFFE，当有指定 Min / Max 时，则为 Min ~ Max - 1。

Min：随机数最小值。若不指定则为 0。

Max：最大接受 0xFFFFFFFF。

Note:

1. NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 不支持设定 Min / Max 参数。

2. NX1 指定 Min / Max 时，产生的随机数不受 Random 选项影响。

例. Random=0x2A

Random!=0x3A?True ；取到的值若是不等于 **0x3A**，则跳转至**“True”**。

例. NX1

Random(0x10, 0x1000)=0x100?True ；取到的值若是不等于 **0x100**，则跳转至**“True”**。

5.2.18 Px[1 X 0 X]?Path

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

可以使用该指令来读取 Port 的特定 IO 脚位的状态，并且跳转到对应的路径。

例. 读取 PB.1 与 PB.3。

```
TR1: PB[1 X 0 X]?True                    ; PB.3=1 and PB.1=0，则跳转至“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 播放中的话，则跳转至 Path。

Ch：指定语音通道，若不指定，则视同全部通道。

- NY5+支持通道 Ch= Ch0 ~ Ch3。
- NY6 支持通道 Ch= Ch0 ~ Ch5。
- NY7 支持通道 Ch= Ch0 ~ Ch7。
- NX1 支持通道 Ch= Ch0 ~ Ch7。

例. 使用 Ch4 来播放 Voice，使用 Ch0 ~ Ch3 来播放 Melody。

```
TR1: PlayV(ch4,$V0)
TR2: PlayM($M0)
TR3: ChUsed(3)?Channel_Playing      ; 通道 3 正在使用中，则跳转至“Channel_Playing”。
```

5.2.20 Voice(Ch)?Path

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

若是指定的语音通道有 Voice 正在播放中的话，则跳转至 Path。

Ch：指定语音通道，若不指定，则视同全部语音通道。

- NY4 不能指定通道。
- NY5 / NY5+支持 Ch=0 ~ 3 或 Ch0 ~ Ch3。
- NY6 支持 Ch= Ch0 ~ Ch1。
- NY7 支持 Ch= Ch0 ~ Ch7。
- NX1 支持 Ch= Ch0 ~ Ch7。

注意：此指令仅在程序中有使用 PlayV 或 PlayVS 时才有效。

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，则跳转至“Voice_Playing”。
TR3: Voice(0)?Ch0_Playing           ; 通道 0 正在播放中，则跳转至“CH0_Playing”。
```

```
TR4: Voice(1)?Ch1_Playing          ; 通道 1 正在播放中，则跳转至“CH1_Playing”。
TR5: Voice(ch2)?Ch2_Playing        ; 通道 2 正在播放中，则跳转至“CH2_Playing”。
TR6: Voice(ch3)?Ch3_Playing        ; 通道 3 正在播放中，则跳转至“CH3_Playing”。
```

5.2.21 PauseV(Ch)?Path

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

若是指定的语音通道目前正在执行 PauseV 功能，则跳转至 Path。

Ch：指定语音通道，若不指定，则视同全部语音通道。

- NY4 不能指定通道。
- NY5 / NY5+支持 Ch=0 ~ 3 或 Ch0 ~ Ch3。
- NY6 支持 Ch=0 ~ 5 或 Ch0 ~ Ch5。
- NY7 支持 Ch=0 ~ 7 或 Ch0 ~ Ch7。
- NX1 支持 Ch=0 ~ 7 或 Ch0 ~ Ch7。

注意：此指令仅在程序中有使用 PlayV 或 PlayVS 及 PauseV 时才有效。

例. 任一语音通道的 Voice 正在暂停播放中。

```
PowerOn: InputState
TR1: PlayV(ch0,$V0), PlayV(ch1,$V1), PlayV(ch2,$V2), PlayV(ch3, $V3)
TR2: PauseV?VoicePause             ; 任一通道正在暂停播放中，则跳转至“VoicePause”。
TR3: PauseV(0)?Ch0_Pause           ; 通道 0 正在暂停播放中，则跳转至“CH0_Pause”。
TR4: PauseV(1)?Ch1_Pause           ; 通道 1 正在暂停播放中，则跳转至“CH1_Pause”。
TR5: PauseV(ch2)?Ch2_Pause         ; 通道 2 正在暂停播放中，则跳转至“CH2_Pause”。
TR6: PauseV(ch3)?Ch3_Pause         ; 通道 3 正在暂停播放中，则跳转至“CH3_Pause”。
```

5.2.22 Melody?Path

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

若是 Melody 正在播放中，则跳转至 Path。

例. Melody 正在播放中。

```
Melody?True                        ; Melody 播放中，则跳转至“True”。
```

5.2.23 PauseM?Path

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

若是目前正在执行 PauseM 功能，则跳转至 Path。

例. Melody 正在暂停播放中。

```
PauseM?True                        ; Melody 暂停中，则跳转至“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执行时间延迟功能，则跳转至 Path。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。若不指定，则表示所有的 Delay。

例.

Delay(0)?True ；**前景 Delay 执行中则跳转至“True”。**

Delay?True ；**所有前后背景中正在有 Delay 执行中则跳转至“True”。**

5.2.25 PauseD(n)?Path

[NY9T]

若是指定的前景或背景目前正在运行时间暂停功能，则跳转至 Path。

n：0=前景，1=背景 1，2=背景 2。若不指定，则表示所有的 Delay。

例. **PauseD(0)?True_Path** ；**前景 Delay 暂停执行中，则跳转至“True_Path”。**

例. **PauseD?True_Path** ；**所有前后背景中有 Delay 正在暂停执行中，则跳转至“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，则跳转至 Path。

Ch：指定 Action 通道，若不指定，则表示全部 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 1 ~ 8 或 Ch1 ~ Ch8
- NX1 支持 Ch1 ~ Ch32。

注意：此指令仅在程序中有使用 PlayA 或 PlayAS 时才有效。

例. Action 通道 1 正在执行中。

Action(1)?StopOut ；**Action 通道 1 正在执行中，则跳转至“StopOut”。**

Action?StopOut ；**Action 的任一通道正在执行中，则跳转至“StopOut”。**

5.2.27 PauseA(Ch)?Path

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

若是指定的 Action 通道目前正在执行 PauseA 功能，则跳转至 Path。

Ch：指定 Action 通道，若不指定，则视同全部 Action 通道。

- NY5+ / NY6 / NY7 / NY9T 支持 1~8 或 Ch1~Ch8
- NX1 支持 Ch1 ~ Ch32。

注意：此指令仅在程序中有使用 PlayA 或 PlayAS 时才有效。

例. 任一通道的 Action 正在暂停播放中。

```
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          ; 任一通道正在暂停播放中，则跳转至“ActionPause”。
TR3: PauseA(1)?Ch1_Pause         ; 通道 1 正在暂停播放中，则跳转至“CH1_Pause”。
TR4: PauseA(2)?Ch2_Pause         ; 通道 2 正在暂停播放中，则跳转至“CH2_Pause”。
TR5: PauseA(ch3)?Ch3_Pause       ; 通道 3 正在暂停播放中，则跳转至“CH3_Pause”。
TR6: PauseA(ch4)?Ch4_Pause       ; 通道 4 正在暂停播放中，则跳转至“CH4_Pause”。
```

5.2.28 PWMIO?Path

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

若是 PWMIO 正在播放中，则跳转至 Path。

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。若不指定，则表示所有的 PlayPWM / PlayPWMS / PWMOut / PWMOutS。

Pin：指定脚位，表示该脚位是否在使用 PWMOut / PWMOutS 输出 PWMIO。

- **Px.n**：IC 脚位。
- **EXPx_Py.n**：I/O 扩展芯片的脚位。

Ch：PlayPWM / PlayPWMS 所输出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的输出状态。

注意：

1. ***NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支持指定 n。***
2. ***NY4 / NY5 / NY6 / NY7 / NY9T 不支持指定 Pin。***
3. ***NY4 / NY5 / NY6 / NY7 / NY9T / NX1 不支持指定 Ch / PIO。***

例. PWMIO 正在播放中。

```
PWMIO?True                       ; PWMIO 播放中，则跳转至“True”Path。
PWMIO(0)?True_Path               ; 前景 PlayPWM/PlayPWMS 执行中，则跳转至
                                   “True_Path”。
PWMIO(PB.0)?True_Path            ; 若有 PWMOut 输出于 PB.0，则跳转至“True_Path”。
PWMIO(EXP0_P1.2)?True_Path       ; 若有 PWMOut 输出于第 0 颗 I/O 扩展芯片 P1.2 脚位，则
                                   跳转至“True_Path”。
```

5.2.29 PausePWM?Path

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

若是目前正在执行 PausePWM 功能，则跳转至 Path。

PausePWM?Path

PausePWM(n)?Path

PausePWM(Px.n)?Path

PausePWM(Ch)?Path

PausePWM(PIO)?Path

n：0=前景，1=背景 1，2=背景 2。若不指定，则表示所有的 PlayPWM / PlayPWMS / PWMOut / PWMOutS。

Pin：指定脚位，表示该脚位是否暂停由 PWMOut / PWMOutS 输出的 PWM-IO。

- **Px.n**：IC 脚位。
- **EXPx_Py.n**：I/O 扩展芯片的脚位。

Ch：PlayPWM / PlayPWMS 所输出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的输出状态。

注意：

1. NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支持指定 n。

2. NY4 / NY5 / NY6 / NY7 / NY9T 不支持指定 Px.n。

3. NY4 / NY5 / NY6 / NY7 / NY9T / NX1 不支持指定 Ch / PIO。

例. PWMIO 正在暂停播放中。

```
PausePWM?True_Path              ; PWMIO 暂停中，则跳转至“True”。
```

5.2.30 HoldPWM(n)?Path

[NY9T]

若是指定的前景或背景目前正在执行 HoldPWM 功能，则跳转至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，则表示所有的 HoldPWM。

注意：NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支持此指令。

```
例. HoldPWM(0)?True_Path        ; 前景 PlayPWM/PlayPWMS 暂停中，则跳转至“True_Path”。
例. HoldPWM?True_Path           ; 所有前后背景中有 PlayPWM/PlayPWMS 正在暂停执行中，则跳转至“True_Path”。
```

5.2.31 SPIPlay(Ch)?Path

[NX1]

若是指定的语音通道有 SPIPlay 正在播放中的话，则跳转至 Path。

Ch：指定语音通道，若不指定，则视同透过 SPI 播放的全部语音通道与全部 Action 通道。

- NX1 支援 Ch0 ~ Ch3。

注意：此指令仅在程序中有使用 SPIPlay / SPIPlayS 时才有效。

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，则跳转至“Voice_Playing”。

TR3: SPIPlay(ch0)?Ch0_Playing ; 通道 0 正在播放中，则跳转至“CH0_Playing”。

TR4: SPIPlay(ch1)?Ch1_Playing ; 通道 1 正在播放中，则跳转至“CH1_Playing”。

TR5: SPIPlay(ch2)?Ch2_Playing ; 通道 2 正在播放中，则跳转至“CH2_Playing”。

TR6: SPIPlay(ch3)?Ch3_Playing ; 通道 3 正在播放中，则跳转至“CH3_Playing”。

5.2.32 SPIPause(Ch)?Path

[NX1]

若是指定的语音通道目前正在执行 SPIPause 功能，则跳转至 Path。

Ch：指定语音通道，若不指定，则视同透过 SPI 播放的全部语音通道与全部 Action 通道。

- NX1 支援 Ch0 ~ Ch3。

注意：此指令仅在程序中有使用 SPIPlay / SPIPlayS / SPIPause 时才有效。

例. 任一语音通道的 SPIPlay 正在暂停播放中。

PowerOn: InputState

TR1: SPIPlay(ch0,0), SPIPlay(ch1,1), SPIPlay(ch2,2), SPIPlay(ch3, 3)

TR2: SPIPause?VoicePause ; 任一通道正在暂停播放中，则跳转至“VoicePause”。

TR3: SPIPause(ch0)?Ch0_Pause ; 通道 0 正在暂停播放中，则跳转至“CH0_Pause”。

TR4: SPIPause(ch1)?Ch1_Pause ; 通道 1 正在暂停播放中，则跳转至“CH1_Pause”。

TR5: SPIPause(ch2)?Ch2_Pause ; 通道 2 正在暂停播放中，则跳转至“CH2_Pause”。

TR6: SPIPause(ch3)?Ch3_Pause ; 通道 3 正在暂停播放中，则跳转至“CH3_Pause”。

5.2.33 Pause(n)?Path

[NY9T]

若是指定的前景或背景目前正在执行任一播放或时间延迟功能，则跳转至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，则表示所有前景与背景。

例. **Pause(0)?True_Path ; 前景暂停执行中，则跳转至“True_Path”。**

例. **Pause?True_Path ; 所有前后背景的动作正在暂停执行中，则跳转至“True_Path”。**

5.2.34 Record?Path

[NX1]

若是录音功能正在执行中，则跳转至 Path。

*例.*Record?True_Path ；录音功能执行中，则跳转至“True_Path”。

5.2.35 KRecord?Path

[NX1]

若是琴键录音功能正在执行中，则跳转至 Path。

注意：当 KRecord / KRecordS 被执行时，必须要触发 InstNoteOn 或 InstNoteOff 一次，才会正式进入琴键录音模式。

例.

KRecord?True_Path ；录音功能执行中，则跳转至“True_Path”。

5.2.36 Recorded($Rec)?Path

[NX1]

若是指定的录音区段内含录音数据，则跳转至 Path。

例.

Recorded($Rec0)?True_Path ;Rec0 录音区段内含录音数据，则跳转至“True_Path”。

5.2.37 PlayK?Path

[NX1]

若是琴键回放功能正在执行中，则跳转至 Path。

例.

PlayK?True_Path ；录音功能执行中，则跳转至“True_Path”。

5.2.38 EraseR?Path

[NX1]

若是正在擦除 SPI Flash 上的录音区段中，则跳转至 Path。

例.

EraseR?True_Path ;若正在擦除录音区段，则跳转至“True_Path”。

5.2.39 VR_VAD?Path

[NX1]

若是 VAD 超过门坎值，则跳转至 Path。

例.

```
VR_VAD?True_Path                    ;若 VAD 超过门坎值，则跳转至“True_Path”。
```

5.2.40 LEDStr?Path

[NX1]

若是正在进行 Single-String 灯串播放，则跳转至 Path。

LEDStr?Path

LEDStr(Px.n)?Path

Px.n：指定检查是否正在进行灯串播放的脚位，不指定则检查所有 Single-String 输出脚位。

例.

```
Path1: LEDStr?True_Path              ；若灯串播放中，则跳转至“True_Path”。
```

5.2.41 LEDSync?Path

[NX1]

若是正在进行 Sync-Play 灯串播放，则跳转至 Path。

LEDSync?Path

例.

```
Path1: LEDSync?True_Path             ；若灯串播放中，则跳转至“True_Path”。
```

5.2.42 LEDText?Path

[NX1]

若是正在进行 Scrolling-Text 灯串播放，则跳转至 Path。

LEDText?Path

例.

```
Path1: LEDText?True_Path             ；若灯串播放中，则跳转至“True_Path”。
```

5.2.43 Animaltalks_Record?Path

[NX1]

若是正在进行动物变音的录音，则跳转至 Path。

Animaltalks_Record?Path

例.

Path1: Animaltalks_Record?True_Path ；若动物变音录音中，则跳转至 “True_Path”。

5.2.44 Animaltalks_Play?Path

[NX1]

若是正在进行动物变音播放，则跳转至 Path。

Animaltalks_Play?Path

例.

Path1: Animaltalks_Play?True_Path ；若动物变音播放中，则跳转至 “True_Path”。

5.2.45 Animalsings_Record?Path

[NX1]

若是正在进行动物唱歌的录音，则跳转至 Path。

Animalsings_Record?Path

例.

Path1: Animalsinigs_Record?True_Path ；若动物唱歌录音中，则跳转至 “True_Path”。

5.2.46 Animalsings_Play?Path

[NX1]

若是正在进行动物唱歌播放，则跳转至 Path。

Animalsings_Play?Path

例.

Path1: Animalsings_Play?True_Path ；若动物唱歌播放中，则跳转至 “True_Path”。

5.2.47 Calibrate?Path

[NY9T]

若是触摸键校正功能正在执行中，则跳转至 Path。

例.

Calibrate?True_Path ；AutoJudge_Calibrate 或 Enforce_Calibrate 功能执行中，则跳转至“True_Path”。

5.2.48 IoExp_Exists?Path

[NY5+ / NX1]

检查 ExpID 对应的 I/O 扩展芯片是否存在，若存在则跳至“Path”。

IoExp_Exists(ExpId)?Path

ExpId：I/O 扩展芯片 ID。

5.2.49 CheckSum?Path

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

用于快速检测 ROM DATA 是否有遭到损毁。CheckSum?Path 指令会自动判读 ROM DATA 的正确性，若是 ROM DATA 正确则会跳至“Path”。

注意：

1. 当 CheckSum Command 被执行时，会停止当前所有动作。在执行 CheckSum 时，会需要一些时间。且执行完毕前将没有任何回应。

2. NX1 EF 系列不会检查整块 ROM 区，因 UDR 区块为可更新数据，不在检查范围之内。

例.

```
TEST_ROUTINE: Checksum?OK, PlayV(Ch1,Fail.wav)      ; Check Sum 正确会跳至“OK”。
                                                    ; 反之，则执行下一个指令。
OK：PlayV(Ch1, OK.wav)
```

5.2.50 SPI_CheckSum?Path

[NX1]

用于快速检测 SPI Flash 资料是否有遭到损毁。SPI_CheckSum?Path 指令会自动判读 SPI Flash 数据的正确性，若是数据正确则会跳至“Path”。

注意：当 SPI CheckSum 指令执行时，会停止当前所有动作。在执行 SPI CheckSum 时，会需要一些时间。且执行完毕前将没有任何回应。

例.

```
TEST_ROUTINE: SPI_CheckSum?OK, PlayV(Ch1, $Fail)    ; SPI checksum 正确会跳至“OK”。
                                                    ;反之，则执行下一个指令。
OK: PlayV(Ch1, $OK)
```

5.2.51 !Px[1 X 0 X]?Path

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

可以使用该指令来读取 Port 的特定 IO 脚位的状态，并且跳跃到对应的路径。

例. 读取 PB.1 与 PB.3。

```
TR1: !PB[1 X 0 X]?False_Path                ; PB.3!=1 或 PB.1!=0，则跳跃至“False_Path”。
```

False_Path: PlayV(Ch0, $V0)

5.2.52 !ChUsed(Ch)?Path

[NY5+ / NY6 / NY7 / NX1]

指定的通道若无 Voice 或 Melody 播放中的话，则跳跃至 Path。

Ch：指定语音信道，不指定，则视同全部通道。

- 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 来播放 Voice，使用 Ch0 ~ Ch3 来播放 Melody。

TR1: PlayV(Ch4, $V0)

TR2: PlayM($M0)

TR3: !ChUsed(Ch3)?Channel_NotPlaying ；通道 3 非使用中，则跳跃至“Channel_NotPlaying”。

5.2.53 !Voice(Ch)?Path

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

若是指定的语音信道无 Voice 正在播放中的话，则跳跃至 Path。

Ch：指定语音信道，若不指定，则视同全部语音信道。

- 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：此指令仅在程序中有使用 PlayV 或 PlayVS 时才有效。

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，则跳跃至“Voice_NotPlaying”。

TR3: !Voice(Ch0)?Ch0_NotPlaying ；通道 0 非播放中，则跳跃至“Ch0_NotPlaying”。

TR4: !Voice(Ch1)?Ch1_NotPlaying ；通道 1 非播放中，则跳跃至“Ch1_NotPlaying”。

TR5: !Voice(Ch2)?Ch2_NotPlaying ；通道 2 非播放中，则跳跃至“Ch2_NotPlaying”。

TR6: !Voice(Ch3)?Ch3_NotPlaying ；通道 3 非播放中，则跳跃至“Ch3_NotPlaying”。

5.2.54 !PauseV(Ch)?Path

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

若是指定的语音信道目前并未执行 PauseV 功能，则跳跃至 Path。

Ch：指定语音信道，若不指定，则视同全部语音信道。

- 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：此指令仅在程序中有使用 PlayV 或 PlayVS 及 PauseV 时才有效。

例. 任一语音信道的 Voice 正在暂停播放中。

```
PowerOn: InputState
TR1: PlayV(Ch0, $V0), PlayV(Ch1, $V1), PlayV(Ch2, $V2), PlayV(Ch3, $V3)
TR2: !PauseV?VoiceNotPause          ; 所有通道并未暂停播放，则跳跃至“VoiceNotPause”。
TR3: !PauseV(Ch0)?Ch0_NotPause      ; 通道 0 并未暂停播放，则跳跃至“Ch0_NotPause”。
TR4: !PauseV(Ch1)?Ch1_NotPause      ; 通道 1 并未暂停播放，则跳跃至“Ch1_NotPause”。
TR5: !PauseV(Ch2)?Ch2_NotPause      ; 通道 2 并未暂停播放，则跳跃至“Ch2_NotPause”。
TR6: !PauseV(Ch3)?Ch3_NotPause      ; 通道 3 并未暂停播放，则跳跃至“Ch3_NotPause”。
```

5.2.55 !Melody?Path

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

若是 Melody 非播放中，则跳跃至 Path。

例. Melody 非播放中。

```
!Melody?False_Path                  ; Melody 非播放中，则跳跃至“False_Path”。
```

5.2.56 !PauseM?Path

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

若是目前未执行 PauseM 功能，则跳跃至 Path。

例. Melody 正在暂停播放中。

```
!PauseM?False_Path                  ; Melody 并未暂停，则跳跃至“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并未运行时间延迟功能，则跳跃至 Path。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，则表示所有的 Delay。

例.

```
!Delay(0)?False_Path                ; 前景无 Delay 执行中则跳跃至“False_Path”。
```

```
!Delay?False_Path                ; 所有前后背景中皆无 Delay 执行中则跳跃至
                                 ; “False_Path”。
```

5.2.58 !PauseD(n)?Path

[NY9T]

若是指定的前景或背景目前并未执行 PauseD，则跳跃至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，则表示所有的 Delay。

```
例. !PauseD(0)?False_Path        ; 前景并未执行 PauseD 中，则跳跃至“False_Path”。
例. !PauseD?False_Path           ; 所有前后背景中并未执行 PauseD，则跳跃至
                                 ; “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，则跳跃至 Path。

Ch：指定 Action 通道，若不指定，则表示全部 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 Ch1 ~ Ch8
- NX1 支援 Ch1 ~ Ch32。

注意：此指令仅在程序中有使用 PlayA 或 PlayAS 时才有效。

例. Action 通道 1 正在执行中。

```
!Action(Ch1)?StartOut            ; Action 通道 1 不在执行中，则跳跃至“StartOut”。
!Action?StartOut                 ; Action 的全部通道都不在执行中，则跳跃至“StartOut”。
```

5.2.60 !PauseA(Ch)?Path

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

若是指定的 Action 通道目前并未执行 PauseA 功能，则跳跃至 Path。

Ch：指定 Action 通道，若不指定，则视同全部 Action 通道。

- NY5+ / NY6 / NY7 / NY9T 支持 1 ~ 8 或 Ch1 ~ Ch8
- NX1 支援 1 ~ 20 或 Ch1 ~ Ch32。

注意：此指令仅在程序中有使用 PlayA 或 PlayAS 时才有效。

例. 任一通道的 Action 正在暂停播放中。

```
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      ; 所有通道并未暂停播放，则跳跃至“ActionNotPause”。
TR3: !PauseA(1)?Ch1_NotPause     ; 通道 1 并未暂停播放，则跳跃至“Ch1_NotPause”。
TR4: !PauseA(2)?Ch2_NotPause     ; 通道 2 并未暂停播放，则跳跃至“Ch2_NotPause”。
```

TR5: !PauseA(ch3)?Ch3_NotPause ；通道 3 并未暂停播放，则跳跃至“Ch3_NotPause”。

TR6: !PauseA(ch4)?Ch4_NotPause ；通道 4 并未暂停播放，则跳跃至“Ch4_NotPause”。

5.2.61 !PWMIO?Path

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

若是 PWMIO 非播放中，则跳跃至 Path。语法如下：

!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，则表示所有的 PlayPWM / PlayPWMS / PWMOut / PWMOutS。

Pin：指定脚位，表示该脚位是否在使用 PWMOut / PWMOutS 输出 PWMIO。

- **Px.n**：IC 脚位。
- **EXPx_Py.n**：I/O 扩展芯片的脚位。

Ch：PlayPWM / PlayPWMS 所输出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的输出状态。

注意：

1. ***NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支持指定 n。***
2. ***NY4 / NY5 / NY6 / NY7 / NY9T 不支持指定 Pin。***
3. ***NY4 / NY5 / NY6 / NY7 / NY9T / NX1 不支持指定 Ch / PIO。***

例. PWMIO 正在播放中。

!PWMIO?False_Path ；PWMIO 非播放中，则跳跃至“False_Path”。

!PWMIO(0)?False_Path ；前景 PlayPWM/PlayPWMS 非执行中，则跳跃至“False_Path”。

!PWMIO(PB.0)?False_Path ；若无 PWMOut 输出于 PB.0，则跳跃至“False_Path”。

!PWMIO(EXP0_P1.2)?False_Path ；若无 PWMOut 输出于第 0 颗 I/O 扩展芯片 P1.2 脚位，则跳跃至“False_Path”。

5.2.62 !PausePWM?Path

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

若是目前并未执行 PausePWM 功能，则跳跃至 Path。语法如下：

!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，则表示所有的 PausePWM / PlayPWMS / PWMOut / PWMOutS。

Pin：指定脚位，表示该脚位是否暂停由 PWMOut / PWMOutS 输出的 PWM-IO。

- **Px.n**：IC 脚位。
- **EXPx_Py.n**：I/O 扩展芯片的脚位。

Ch：PlayPWM / PlayPWMS 所输出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的输出状态。

注意：

1. NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支持指定 n。

2. NY4 / NY5 / NY6 / NY7 / NY9T 不支持指定 Pin。

3. NY4 / NY5 / NY6 / NY7 / NY9T / NX1 不支持指定 Ch / PIO。

例. PWMIO 正在暂停播放中。

```
!PausePWM?False_Path                ; PWMIO 并未暂停，则跳跃至“False_Path”。
```

5.2.63 !HoldPWM(n)?Path

[NY9T]

若是指定的前景或背景目前并未执行 HoldPWM 功能，则跳跃至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，则表示所有的 HoldPWM。

注意：NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支持此指令。

例.

```
!HoldPWM(0)?False_Path      ; 前景 PlayPWM / PlayPWMS 非暂停中，则跳跃至“False_Path”。
!HoldPWM?False_Path         ; 所有前后背景中无 PlayPWM / PlayPWMS 正在暂停执行中，则跳跃至“False_Path”。
```

5.2.64 !SPIPlay(Ch)?Path

[NX1]

若是指定的语音信道无 SPIPlay 正在播放中的话，则跳跃至 Path。

Ch：指定语音信道，若不指定，则视同透过 SPI 播放的全部语音信道与全部 Action 信道。

- NX1 支援 Ch0 ~ Ch3。

注意：此指令仅在程序中有使用 SPIPlay / SPIPlayS 时才有效。

例.

```
PowerOn: InputState
TR1: SPIPlay(Ch0, 0), SPIPlay(Ch1, 1), SPIPlay(Ch2, 2), SPIPlay(Ch3, 3)
TR2: !SPIPlay?Voice_NotPlaying      ; 无 SPIPlay 播放，则跳跃至“Voice_NotPlaying”。
```

```
TR3: !SPIPlay(Ch0)?Ch0_NotPlaying   ; 通道 0 无 SPIPlay 播放，则跳跃至“Ch0_NotPlaying”。
TR4: !SPIPlay(Ch1)?Ch1_NotPlaying   ; 通道 1 无 SPIPlay 播放，则跳跃至“Ch1_NotPlaying”。
TR5: !SPIPlay(Ch2)?Ch2_NotPlaying   ; 通道 2 无 SPIPlay 播放，则跳跃至“Ch2_NotPlaying”。
TR6: !SPIPlay(Ch3)?Ch3_NotPlaying   ; 通道 3 无 SPIPlay 播放，则跳跃至“Ch3_NotPlaying”。
```

5.2.65 !SPIPause(Ch)?Path

[NX1]

若是指定的语音信道目前并非在执行 SPIPause 功能，则跳跃至 Path。

Ch：指定语音信道，若不指定，则视同透过 SPI 播放的全部语音信道与全部 Action 信道。

- NX1 支援 Ch0 ~ Ch3。

注意：此指令仅在程序中有使用 SPIPlay / SPIPlayS / SPIPause 时才有效。

例. 任一语音信道的 SPIPlay 正在暂停播放中。

```
PowerOn: InputState
TR1: SPIPlay(Ch0, 0), SPIPlay(Ch1, 1), SPIPlay(Ch2, 2), SPIPlay(Ch3, 3)
TR2: !SPIPause?VoiceNotPause        ; 所有通道皆非暂停播放中，则跳跃至
                                    ; “VoiceNotPause”。
TR3: !SPIPause(Ch0)?Ch0_NotPause    ; 通道 0 非暂停播放中，则跳跃至“Ch0_NotPause”。
TR4: !SPIPause(Ch1)?Ch1_NotPause    ; 通道 1 非暂停播放中，则跳跃至“Ch1_NotPause”。
TR5: !SPIPause(Ch2)?Ch2_NotPause    ; 通道 2 非暂停播放中，则跳跃至“Ch2_NotPause”。
TR6: !SPIPause(Ch3)?Ch3_NotPause    ; 通道 3 非暂停播放中，则跳跃至“Ch3_NotPause”。
```

5.2.66 !Pause(n)?Path

[NY9T]

若是指定的前景或背景并未执行 Pause，则跳跃至 Path。

n：0=前景，1=背景 1，2=背景 2。n 不指定，则表示所有前景与背景。

注意：NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支持此指令。

```
例. !Pause(0)?False_Path    ; 前景并未执行 Pause，则跳跃至“False_Path”。
例. !Pause?False_Path       ; 所有前后背景并未执行 Pause，则跳跃至“False_Path”。
```

5.2.67 !Record?Path

[NX1]

若是录音功能未在执行中，则跳跃至 Path。

```
例. !Record?False_Path      ; 无录音功能执行中，则跳跃至“False_Path”。
```

5.2.68 !KRecord?Path

[NX1]

若是琴键录音功能非执行中，则跳跃至 Path。

注意：当 KRecord / KRecordS 被执行时，必须要触发 InstNoteOn 或 InstNoteOff 一次，才会正式进入琴键录音模式。

例.

```
!KRecord?False_Path                ; 录音功能非执行中，则跳跃至“False_Path”。
```

5.2.69 !Recorded($Rec)?Path

[NX1]

若是指定的录音区段内不包含录音数据，则跳跃至 Path。

例.

```
!Recorded($Rec0)?False_Path        ; Rec0 录音区段内不包含录音数据，则跳跃至
                                   ; “False_Path”。
```

5.2.70 !PlayK?Path

[NX1]

若是琴键回放功能非执行中，则跳跃至 Path。

例.

```
!PlayK?False_Path                  ; 琴键回放功能非执行中，则跳跃至“False_Path”。
```

5.2.71 !EraseR?Path

[NX1]

若非正在擦除录音区段中，则跳跃至 Path。

例.

```
!EraseR?False_Path                 ; 若录音区段擦除非进行中，则跳跃至“False_Path”。
```

5.2.72 !VR_VAD?Path

[NX1]

若是 VAD 未超过门坎值，则跳跃至 Path。

例.

```
!VR_VAD?False_Path                    ; 若 VAD 未超过门坎值，则跳跃至“False_Path”。
```

5.2.73 !LEDStr?Path

[NX1]

若未进行 Single-String 灯串播放，则跳跃至 Path。

!LEDStr?Path

!LEDStr(Px.n)?Path

Px.n：指定检查是否正在进行 Single-String 灯串播放的脚位，不指定则检查所有 Single-String 输出脚位。

例.

```
Path1: !LEDStr?False_Path             ; 若灯串非播放中，则跳跃至“False_Path”。
```

5.2.74 !LEDSync?Path

[NX1]

若是未进行 Sync-Play 灯串播放，则跳跃至 Path。

!LEDSync?Path

例.

```
Path1: !LEDSync?False_Path            ; 若灯串 Sync-Play 非播放中，则跳跃至“False_Path”。
```

5.2.75 !LEDText?Path

[NX1]

若是未进行 Scrolling-Text 灯串播放，则跳跃至 Path。

!LEDText?Path

例.

```
Path1: !LEDText?False_Path            ; 若灯串非播放中，则跳跃至“False_Path”。
```

5.2.76 !Animaltalks_Record?Path

[NX1]

若非正在进行动物变音的录音，则跳跃至 Path。

!Animaltalks_Record?Path

例.

Path1: !Animaltalks_Record?False_Path ；若未进行动物变音录音中，则跳跃至“False_Path”。

5.2.77 !Animaltalks_Play?Path

[NX1]

若是未在进行动物变音的播放，则跳跃至 Path。

!Animaltalks_Play?Path

例.

Path1: !Animaltalks_Play?False_Path ；若动物变音非播放中，则跳跃至“False_Path”。

5.2.78 !Animalsings_Record?Path

[NX1]

若非正在进行动物唱歌的录音，则跳跃至 Path。

!Animalsings_Record?Path

例.

Path1: !Animalsings_Record?False_Path ；若未进行动物唱歌录音中，则跳跃至“False_Path”。

5.2.79 !Animalsings_Play?Path

[NX1]

若是未在进行动物唱歌的播放，则跳跃至 Path。

!Animalsings_Play?Path

例.

Path1: !Animalsings_Play?False_Path ；若动物唱歌非播放中，则跳跃至“False_Path”。

5.2.80 !Calibrate?Path

[NY9T]

若是触摸键校正功能非执行中，则跳跃至 Path。

例.

!Calibrate?False_Path ；AutoJudge_Calibrate 或 Enforce_Calibrate 功能皆非执行中，则跳跃至“False_Path”。

5.2.81 !IoExp_Exists?Path

[NY5+ / NX1]

检查 ExpID 对应的 I/O 扩展芯片是否存在，若不存在则跳至“Path”。当未指定 ExpID 时则检查所有的 ID。

!IoExp_Exists?Path

!IoExp_Exists(ExpId)?Path

ExpId：I/O 扩展芯片 ID。

5.2.82 !CheckSum?Path

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

用于快速检测 ROM DATA 是否有遭到损毁。!CheckSum?Path 指令会自动判读 ROM DATA 的正确性，若是 ROM DATA 不正确则会跳至“Path”。

注意：

1. 当 CheckSum Command 被执行时，会停止当前所有动作。在执行 CheckSum 时，会需要一些时间。且执行完毕前将没有任何回应。

2. NX1 EF 系列不会检查整块 ROM 区，因 UDR 区块为可更新数据，不在检查范围之内。

例.

```
TEST_ROUTINE: Checksum?NotOK, PlayV(Ch1, Pass.wav); Check Sum 不正确会跳至“NotOK”。
                                                    ；反之，则执行下一个指令。
NotOK：PlayV(Ch1, Fail.wav)
```

5.2.83 !SPI_CheckSum?Path

[NX1]

用于快速检测 SPI Flash 资料是否有遭到损毁。!SPI_CheckSum?Path 指令会自动判读 SPI Flash 数据的正确性，若是数据不正确则会跳至“Path”。

注意：当 SPI CheckSum 指令执行时，会停止当前所有动作。在执行 SPI CheckSum 时，会需要一些时间。且执行完毕前将没有任何回应。

例.

```
TEST_ROUTINE: !SPI_CheckSum?NotOK, PlayV(Ch1, $Pass) ; SPI checksum 不正确会跳至
                                                    ; “NotOK”。反之，则执行下一个指令。
NotOK: PlayV(Ch1, $Fail)
```

5.2.84 If-Else

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

If-Else 是 Condition?Path 的另一种写法。

```
Voice?{ PA.0 = 0 }:{ PA.0 = 1 }
```

可写成如下程序片段：

```
If(Voice)
{
  PA.0 = 0
}
Else
{
  PA.0 = 1
}
```

如果 Condition 成立时执行 If 区块内的程序，若 Condition 不成立则执行 Else 区块内的程序。执行后继续执行 If-Else 之后的程序。

If(Condition) { … }
If(Condition) { … } Else { … }
Condition: 判断条件。

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 指令会依照 Ri 的值切换目前的路径到相对应的路径。这个路径可以是一个前景路径，背景路径或“X”（代表没有动作）。当全部切换后的路径都为“X”时，表示此时的路径都可以跳过，换言之，这个 Switch Statement 的步骤也可以不必写。若是在 **[Path]** 内没有被定义到的会执行后面的指令。

Ri 的内容若是 0 则跳转至 Path0，若为 1 则跳转至 Path1，依此类推。

```
例. R0=3, Switch(R0)=[ Path1, Path2, X, Path3], Path4      ; R0=3，跳转至 Path3。
例. R0=4, Switch(R0)=[ Path1, Path2, X, Path3], Path4      ; R0 超出 Switch command 定义的范
                                                           ; 围。
```

5.2.86 Switch(Px) = [Path0, Path1, Path2,..Path15]

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

用法同 Switch(Ri)，Switch 指令会依照 Px 的值切换目前的路径到相对应的路径。

Px 的内容若是 0 则跳转至 Path0，若为 1 则跳转至 Path1，依此类推。

```
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 定义的
                                                           ; 范围，执行下一个指令。
```

5.2.87 Switch(RandomL) = [Path0, Path1, Path2,... Path15]

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

Switch 指令会依照 RandomL 的值切换目前的路径到相对应的路径。这个路径可以是一个前景路径，背景路径或“X”（代表没有动作）。若是在 **[Path]**内没有被定义到的会执行后面的指令。

RandomL 的内容若是 0 则跳转至 Path0，若为 1 则跳转至 Path1，依此类推。

```
例. Switch(RandomL)=[ Path1, Path2, X, Path3], Path4 ; 依照 RandomL 值切换到对应路径。
```

5.2.88 Switch(RandomH) = [Path0, Path1, Path2,… Path15]

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

Switch 指令会依照 RandomH 的值切换目前的路径到相对应的路径。这个路径可以是一个前景路径，背景路径或“X”（代表没有动作）。若是在 **[Path]**内没有被定义到的会执行后面的指令。

RandomH 的内容若是 0 则跳转至 Path0，若为 1 则跳转至 Path1，依此类推。

例. Switch(RandomH)=[Path1, Path2, X, Path3], Path4; 依照 RandomH 值切换到对应路径。

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 指令会依照 Xi 的值切换目前的路径到相对应的路径。这个路径可以是一个前景路径，背景路径或“X”（代表没有动作）。

Xi 的内容若是 0 则跳转至 Path0，若为 1 则跳转至 Path1，依此类推。

```
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 所定义的范围，
                                                      ; 跳过 Switch Command，执行下一个
                                                      ; 指令。
```

5.2.90 Switch(Random) = [Path0, Path1, Path2,… Path255]

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

Switch 指令会依照 Random 的值切换目前的路径到相对应的路径。这个路径可以是一个前景路径，背景路径或“X”（代表没有动作）。Random 的内容若是 0 则跳转至 Path0，若为 1 则跳转至 Path1，依此类推。

```
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
                                                             ;义的范围，跳过 Switch
                                                             ; Command，执行下一个指令。
```

5.2.91 Switch(Px[d x d x]) = [Path0, Path1, Path2,… Path15]

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

可以使用该指令来读取 Port 的特定 IO 脚位的状态，并将 IO 状态组合成一个数值，根据此数值跳转到对应的路径。Switch 指令会依照有效位“d”的值切换目前的路径到相对应的路径。这个路径可以是一个前景路径，背景路径或“X”（代表没有动作）。此功能主要用来达到 bonding option 的功能。

例. 读取 PB.1 与 PB.3，两个位总共可以组合出四种结果。

```
TR1: Switch( PB[d x d x] ) = [P0, P1, P2, P3]  ; 利用 PB1 与 PB3 来组合出四种结果。
P0: PlayV(Ch0, $V0)                            ; 当 PB1 及 PB3 皆为 0，执行 P0 路径。
P1: PlayV(Ch0, $V2)                            ; 当 PB1 为 1 及 PB3 为 0，执行 P1 路径。
P2: PlayV(Ch0, $V8)                            ; 当 PB1 为 0 及 PB3 为 1，执行 P2 路径。
P3: PlayV(Ch0, $V10)                           ; 当 PB1 及 PB3 皆为 1，执行 P3 路径。
```

5.2.92 While

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

重复执行 {...} 中的程序，直到 Condition 不成立为止。

表达式的测试会在每次执行循环之前进行。因此，While 循环会执行零次或多次。

执行程序如下所示：

1. 评估 Condition。 如果 Condition 不成立，While 会结束并执行其后的指令。
2. 执行 {...} 中的程序。
3. 跳回步骤 1。

在 {...} 中执行路径或是条件跳跃等指令，While 也会终止。

While(Condition) { ... }

Condition：判断条件。*Q-Code* 支持的判断式同条件跳跃指令。

5.2.93 Do-While

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

Do-While 语句可让您重复 {...} 内的程序，直到 Condition 不成立为止。

执行 {...} 中的程序后，会评估 Condition 是否成立。因此，循环主体一律至少执行一次。

执行程序如下所示：

1. 执行 {...} 中的程序。
2. 接下来会评估 Condition。如果 Condition 不成立，Do-While 会结束并执行其后的指令。如果 Condition 仍成立，则会从步骤 1 开始重复执行 {...} 内的程序。

在 {...} 中执行路径或是条件跳跃等指令，Do-While 也会终止。

Do { ... } While(Condition)

Condition：判断条件。*Q-Code* 支持的判断式同条件跳跃指令。

例. 将 1 到 9 的加总储存至 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 循环。

进入循环前，Var 会被设定为 Start，循环进行中会逐次递增或递减 Var 的值，并执行 {...} 中的程序，直

到 Var 等于 End 为止。

1. 1..10 会产生 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 的结果。
2. 1..<10 会产生 1, 2, 3, 4, 5, 6, 7, 8, 9 的结果。
3. 1..10 step 3 会产生 1, 4, 7, 10 的结果。
4. 1..<10 step 3 会产生 1, 4, 7 的结果。
5. 1..10 step -1 不会进入循环。

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 在循环进行时，会依照 Increment 的值进行递增或递减。

Start：循环变量 Var 的初始值。

End：循环变量 Var 的最终值。

Increment：每次循环执行时，会使用 Increment 的值进行递增或递减，仅接受常数值，可为负值。不指定则为 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，每次间隔 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	-	-	-

针对 IO 脚位操作指令，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]

将 Px 的值存到 Ri。

例. **PB= 0x3, R0=PB ; R0=0x3**

5.3.2 Ri = PxM

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

将 Port 输出/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]

将 Px 的第 n 个 bit，输出到 Ri 的第 n 个 bit。

例. **PB=0x3, R0=0x0, R0=PB ; R0=0x3**

5.3.4 PxM = data

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

设定 I/O Port 的输出和输入模式。

例. **PBM=0x5 ; PB.0=输入模式，PB.1=输出模式**
; PB.2=输入模式，PB.3=输出模式

5.3.5 PxM = Ri

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

由 Ri 来设定 Port 的输出入模式。

例. **R0=0x3, PBM=R0 ; PBM=0x3**

5.3.6 PxM.n = 0

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

设定 Port 第 n 脚的设定成输出模式。

例. **PBM.0=0 ; PB.0=输出模式**

5.3.7 PxM.n = 1

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

设定 Port 第 n 脚的设定成输入模式。

例. **PBM.0=1 ; PB.0=输入模式**

5.3.8 Px = data

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

Px 输出数值 data。

例. **PB=0x5** ; **PB=0x5**

5.3.9 Px = Ri

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

Px 输出 Ri 的内容值。

例. **R0=0x3, PB=R0** ; **PB=0x3**

5.3.10 Px = Py

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

将 Py Port 的内容值输出到 Px Port。

例. **PB=0x3, PA=PB** ; **PA=0x3**

5.3.11 !Px

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

将 Px 值反相后输出。

例. **PB=0x3, !PB** ; **PB=0xC**

5.3.12 !Px.n

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

将 Px 的第 n 个 bit，反相后输出。

例. **PB=0x3, !PB.0** ; **PB=0x2**

5.3.13 Px.n = 0

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

将 Px 的第 n 个 bit 设为 0。

例. **PB=0xD, PB.3=0** ; **PB=0x5**

5.3.14 Px.n = 1

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

将 Px 的第 n 个 bit 设为 1。

例. **PB=0x5, PB.3=1 ; PB=0xD**

5.3.15 Px.n = Ri.n

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

将 Ri 的第 n 个 bit，输出到 Px 的第 n 个 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]

将 Py 的第 n 个 bit，输出到 Px 的第 n 个 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]

将 Py 与 Ri 的值相加后输出到 Px。

例. **PB=0x3, R0=0x4, PB=PB+R0 ; PB=0x7**

5.3.18 Px = Py – Ri

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

将 Py 与 Ri 的值相减后输出到 Px。

例. **PB=0x3, R0=0x2, PB=PB-R0 ; PB=0x1**

5.3.19 Px = Py | Ri

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

将 Py 与 Ri 的值做 OR 后输出到 Px。

例. **PB=0x3, R0=0x4, PB=PB | R0 ; PB=0x7**

5.3.20 Px = Py ^ Ri

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

将 Py 与 Ri 的值做 XOR 后输出到 Px。

例. **PB=0x3, R0=0x4, PB=PB^R0 ; PB=0x7**

5.3.21 Px = Py & Ri

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

将 Py 与 Ri 的值做 AND 后输出到 Px。

例. **PB=0x3, R0=0x4, PB=PB&R0 ; PB=0x0**

5.3.22 Ri = Px + Rj

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

将 Px 与 Rj 的值相加后存到 Ri。

例. **R0=0x3, PB=0x4, R1=PB+R0 ; R1=0x7**

5.3.23 Ri = Px – Rj

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

将 Px 与 Rj 的值相减后存到 Ri。

例. **R0=0x3, PB=0x4, R1=PB-R0 ; R1=0x1**

5.3.24 Ri = Px | Rj

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

将 Px 与 Rj 的值 OR 后存到 Ri。

例. **PB=0x3, R0=0x4, R1=PB|R0 ; R1=0x7**

5.3.25 Ri = Px ^ Rj

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

将 Px 与 Rj 的值 XOR 后存到 Ri。

例. **PB=0x3, R0=0x4, R1=PB^R0 ; R1=0x7**

5.3.26 Ri = Px & Rj

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

将 Px 与 Rj 的值 AND 后存到 Ri。

例. **PB=0x3, R0=0x4, R1=PB&R0 ; R1=0x0**

5.3.27 Px = Py + data

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

将 Py 与 data 的值相加后输出到 Px。

例. **PB=0x3, PB=PB+0x4 ; PB=0x7**

5.3.28 Px = Py – data

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

将 Py 与 data 的值相减后输出到 Px。

例. **PB=0x3, PB=PB-0x2 ; PB=0x1**

5.3.29 Px = Py | data

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

将 Py 与 data 的值做 OR 后输出到 Px。

例. **PB=0x3, PB=PB | 0x4 ; PB=0x7**

5.3.30 Px = Py ^ data

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

将 Py 与 data 的值做 XOR 后输出到 Px。

例. **PB=0x3, PB=PB^0x4 ; PB=0x7**

5.3.31 Px = Py & data

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

将 Py 与 data 的值做 AND 后输出到 Px。

例. **PB=0x3, PB=PB&0x4 ; PB=0x0**

5.3.32 Ri = Px + data

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

将 Px 与 data 的值相加后存到 Ri。

例. **PB=0x3, R1=PB+0x4 ; R1=0x7**

5.3.33 Ri = Px – data

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

将 Px 与 data 的值相减后存到 Ri。

例. **PB=0x3, PB=PB-0x2 ; R1=0x1**

5.3.34 Ri = Px | data

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

将 Px 与 data 的值 OR 后存到 Ri。

例. **PB=0x3, R1=PB | 0x4 ; R1=0x7**

5.3.35 Ri = Px ^ data

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

将 Px 与 data 的值 XOR 后存到 Ri。

例. **PB=0x3, R1=PB^0x4 ; R1=0x7**

5.3.36 Ri = Px & data

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

将 Px 与 data 的值 AND 后存到 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]

此指令针对 Px 的输出控制。各脚位可选择以下种类的输出方式：

参数	说明	NY4	NY5	NY5+	NY6	NY7	NY9T	NX1
X	表示该脚位维持原来的状态。	✓	✓	✓	✓	✓	✓	✓
0	表示该脚位输出低电平。	✓	✓	✓	✓	✓	✓	✓
1	表示该脚位输出高电平。	✓	✓	✓	✓	✓	✓	✓
A	表示该脚位输出 Action（该功能仅对于播放整个 VIO 文件有效）。	✓	✓	✓	✓	✓	✓	✕
P	表示该脚位输出 PWM-IO（该功能仅对 PlayPWM 播放 MPIOx 时有效）。	✕	✕	✓	✕	✕	✕	✕
FD	表示随播放的声音音量大小变化（Flash Dynamic）。	✓	✓	✓	✓	✓	✕	✓
Q	表示该脚位跟随 *Quick-IO* 的 QIO 信号变化。	✓	✓	✓	✕	✕	✕	✓

例. 若要将 PB 输出 15，则下达 PB=[1 1 1 1]。

例. 若要将 PB 输出 5，则下达 PB=[0 1 0 1]。

例. 若要将 PB 输出 5，且不影响其它 bit 原本的输出，则下达 PB=[X 1 0 1]。

例. 若要将 PB.3 输出跟随着音量大小闪动，则下达 PB=[FD X X X]。

例. 若要将 PB.3 输出跟随着 *Quick-IO* 的信号闪动，则下达 PB=[Q X X X]。

例. 使用 IO 指令设定来播放整个 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 输出 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 输出 VIO1 的 A3 ~ A10。
[Path]
PowerOn: PA=[A A A A], PB=[A A A A]   ; 设定 PA 及 PB 输出 Action。
Path1: PlayA(Ch4, $MVIO0)             ; 播放 MVIO0 于通道 4。
Path2: PlayA(Ch4, $MVIO1)             ; 播放 MVIO1 于通道 4。
```

5.3.38 [Px.n …] = [1 X 0 Q]

[NX1]

针对个别脚位的输出控制。各脚位可选择的输出方式如下：

参数	说明
X	表示该脚位维持原来的状态。
0	表示该脚位输出低电平。
1	表示该脚位输出高电平。
FD	表示随播放的声音音量大小变化（Flash Dynamic）。
Q	表示该脚位跟随 *Quick-IO* 的 QIO 信号变化。

5.3.39 Px.n = 1KHz(time)

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

Px.n 输出几秒 1KHz 的方波。

Time：0~15，单位为秒。time=0，将为循环输出。

注意：该指令被执行时，将不响应任何动作，直到执行完毕。

例. **PB.3=1KHz(5)** ; **PB3 输出 5 sec 1KHz 的方波。**

5.3.40 XiL = Px

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

将 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]

将 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]

将 Px 的第 n 个 bit，输出到 XiL 的第 n 个 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]

将 Px 的第 n 个 bit，输出到 XiH 的第 n 个 bit。

例. **PB=0x5, X0=0x0, X0H.2 = PB.0 ; X0H = 0x4**

5.3.44 Xi.n = Px.n

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

将 Px 的第 n 个 bit，输出到 Xi 的第 n 个 bit。

例. **PB=0x5, X0=0x0, X0.2 = PB.0 ; X0 = 0x4**

5.3.45 Xi = [Px, Py]

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

将 Px 的值存到 Xi 的 High-Nibble，将 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 输出 Xi 的 Low-Byte。

例. **X0L=0x3, PB=X0L ; PB = 0x3**

5.3.47 Px = XiH

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

Px 输出 Xi 的 High-Byte。

例. **X0H=0x3, PB=X0H ; PB = 0x3**

5.3.48 Px.n = XiL.n

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

将 XiL 的第 n 个 bit，输出到 Px 的第 n 个 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]

将 XiH 的第 n 个 bit，输出到 Px 的第 n 个 bit。

例. **PB=0x0, X0=0x50, PB.2=X0H.0 ; B = 0x4**

5.3.50 Px.n = Xi.n

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

将 Xi 的第 n 个 bit，输出到 Px 的第 n 个 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 输出 Xi 的 High-Byte 高位值，Py 输出 Xi 的 Low-Byte。

例. **X0H=0x3, X0L=0x1, [PB, PA]=X0 ; PB = 0x3, PA = 0x1**

5.4 路径指令（Path Command）

Path Command				
ASM	C-Code	BG	BreakFG	StopFG
StopBG	StopBG1	StopBG2	StopBG3	Subroutine

Label(Pathname)	Macro	1ms_RET	4ms_RET	-

5.4.1 ASM

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

在 *Q-Code* 中用户可以插入汇编语言模块，在 **[ASM]** 段落中加入汇编语言模块，并且在 *Q-Code* 中使用。

例. 使用 ASM 来控制 IC。

```
[ASM]
LED
{
mvla    b'0101'
mpg     pa_page
mvam    pa_state
mvam    pa
}

[Path]
TR1: LED                                    ; 调用 Assembly。
```

5.4.2 C-Code

[NX1]

在 *Q-Code* 中用户可以插入 C 语言模块，在 **[C-Code]** 段落中加入 C 语言模块，并且在 *Q-Code* 中使用。

例. 使用 C 来控制 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)
```

若有打开 XIP (Execute in place) 功能，C-Code 也可以指定部份程序存放于 XIP 区段。在函数原型宣告增加 __XIP 属性即可。

例. 指定函数存放于 XIP 区段

[C-Code]

```
void func1(void) __XIP;
void func1(void)
{
    // do something
}
```

除了直接使用 __XIP 属性，也可以藉由预先定义的 _SPI_XIP 宏得知目前有无打开 XIP (Execute in place) 功能，若有打开 XIP 则 _SPI_XIP 宏为 1，否则为 0。建议编写 C-Code 的时候使用以下范例的方式参考 _SPI_XIP 宏，并另行定义 XIP 属性，如此一来当 Q-Code_NX1 的 XIP (Execute in place) 功能打开/关闭状态改变时，C-Code 不需要额外做修正。用户自行定义的宏请尽量避免以底线起头，避免与 Q-Code_NX1 核心（Firmware kernel）功能冲突。

例. 参考 _SPI_XIP 宏

[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]

背景调用指令，用户可以在任一地方来调用背景。*Q-Code* 提供最多 3 层的背景供 User 使用，在不同的层背景可以调用另一层的背景。

BG(BG1)

BG(BG1, BG2)

BG(BG1, BG2, BG3)

BG(BG1, BG2, BG3, BG4)

BG(BG1, BG2, BG3, BG4, BG5)

BG1：背景 1 路径。

- **[Background1]** 中定义的背景路径名称。
- **X**：不改变当前状态。

- **OFF**：停止背景 1 路径。

BG2：背景 2 路径。

- **[Background2]** 中定义的背景路径名称。
- **X**：不改变当前状态。
- **OFF**：停止背景 2 路径。

BG3：背景 3 路径。

- **[Background3]** 中定义的背景路径名称。
- **X**：不改变当前状态。
- **OFF**：停止背景 3 路径。

BG4：背景 4 路径。

- **[Background4]** 中定义的背景路径名称。
- **X**：不改变当前状态。
- **OFF**：停止背景 4 路径。

BG5：背景 5 路径。

- **[Background5]** 中定义的背景路径名称。
- **X**：不改变当前状态。
- **OFF**：停止背景 5 路径。

注意：

1. ***NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 仅支持 BG1 / BG2***
2. ***NX1 支持 BG1 / BG2 / BG3 / BG4 / BG5。***
3. ***BG1 仅能调用 [Background1]背景的动作，不能调用 [Background2] 的动作。反之 BG2 / BG3 / BG4 / BG5 亦同。***

例. 利用背景指令来达成 3Hz 闪灯。

[Path]

PowerOn: BG(OUT1,OUT2), Delay(1), Stop ；调用背景 1 与背景 2 后，会执行后面的指令。

[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]

该指令可以立即停止目前的前景的 PlayV、PlayM 或 Delay 等指令，并且执行前景指令后的下一个指令。

例. 文件$V0 在播放 1 秒后停止，并立即执行后面的 StopV 指令。

[Path]

TR1: PlayV($V0)&[BG1], StopV

[Background1]

BG1: Delay(1), BreakFG

5.4.5 StopFG

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

可以停止当前前景的动作。

例. 利用 StopFG 指令来停止前景动作。

[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 延迟 10 秒后，即停止前景的动作。

5.4.6 StopBG

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

停止背景 1、背景 2 与背景 3 当前的动作。

例. 利用 StopBG 背景指令来停止背景动作。

[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 当前的动作。

例. 利用 StopBG1 背景指令来停止背景 1 动作，背景 2 依旧会动作。

[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 当前的动作。

例. 利用 StopBG2 背景指令来停止背景 2 动作，背景 1 依旧会动作。

[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 当前的动作。

例. 利用 StopBG3 背景指令来停止背景 3 动作，背景 1 与背景 2 依旧会动作。

[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], Delay(0.5), PB=[1 1 X X X], Delay(0.5), OUT3

5.4.10 StopBG4

[NX1]

停止背景 4 当前的动作。

5.4.11 StopBG5

[NX1]

停止背景 5 当前的动作。

5.4.12 Subroutine

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

在 *Q-Code* 中，用户可以插入 *Q-Code* 子程序，在 **[Subroutine]** 段落中加入 *Q-Code* 模块，并且在 *Q-Code* 中使用。

例. 使在 *Q-Code* 中插入子程序。

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

[Path]
TR1:SUB, PlayV(Ch1, $V2)                    ; 执行完“SUB”后，会执行 PlayV(Ch1, $V2)。
```

5.4.13 Label(Pathname)

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

可以在程序中不需要使用”：”来设定路径名称，来减少程序分段。

例.

```
TR1: PlayV(Ch1, $V0), Label(Pathname), PlayV(Ch1, $V1) , Pathname
                                                ; 播放完$V0 后，不停播放$V1。
```

等同以下程序

```
TR1: PlayV(Ch1, $V0), Label_Loop
Label_Loop : PlayV(Ch1, $V1) , Label_Loop       ; 播放完$V0 后，不停播放$V1。
```

5.4.14 Macro

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

在 *Q-Code* 中，用户可以插入宏子程序，在 **[Macro]** 段落中加入宏子程序，并且在 *Q-Code* 中使用。

例. 在 *Q-Code* 中插入宏。

```
[Macro]
MAC:PA=0xF, Delay(0.166)

[Path]
TR1: MAC, PlayV($V1)                    ; 执行完 Macro “MAC”后，会执行 PlayV(Ch1,$V1)。
```

5.4.15 1ms_RET

[NY9T]

1ms 路径返回。

注意：

1. 进入 1ms 路径后，一定要执行 1ms_RET 指令，否则系统将会出现不能预期的错误。

2. 若是 TimeBase=4ms，则 1ms 路径不会被执行。

例.

```
[Option]
TimeBase = 1ms
[Path]
PowerOn: Delay(10), PA=0
1ms: !PA, 1MS_RET                    ；执行完!PA 后，执行 1MS_RET 时，会返回主程序。
```

5.4.16 4ms_RET

[NY9T]

4ms 路径返回。

注意：进入 4ms 路径后，一定要执行 4ms_RET 指令，否则系统将会出现不能预期的错误。

例.

```
[Option]
TimeBase = 1ms or 4ms
[Path]
PowerOn: Delay(10), PA=0
4ms: !PA, 4MS_RET                    ；执行完!PA 后，执行 4MS_RET 时，会返回主程序。
```

5.5 语音指令（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]

Q-Code 提供用户简易的指令来达成播放音源文件的方式。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 文件播放结束后，执行下一个指令。

PlayVS：Voice 文件开始播放，立即执行下一个指令。

Ch：指定要播放的语音通道。

- NY4 不能指定播放通道。
- NY5 / NY5+支持的语音通道为 Ch0 ~ Ch3 或是 0 ~ 3。
- NY6 支持的语音通道为 Ch0 ~ Ch5 或是 0 ~ 5。
- NY7 支持的语音通道为 Ch0 ~ Ch7 或是 0 ~ 7。
- NX1 支持的语音通道为 Ch0 ~ Ch7 或是 0 ~ 7。

Voice：指定要播放的语音。

- Voice Label。
- Ri（可支持到 1~4 个 Ri）或是 Xi（可支持到 1~2 个 Xi），由 Ri / Xi 的值决定要播放的是第几个 voice 文件。若是 Ri / Xi 带有"++"则在播放后将其递增。.
- Record Label 在 **[Record]** 中定义的录音区段。

PlaySpeed：指定要播放的速度，若不指定则使用语音文件的采样频率做为播放速度。可为 Frequency 或 Xi，支持 4K ~ 44.1K。

- NY4 / NY5 使用 Ri / Xi 指定播放速度，Ri / Xi 与播放速度对应表可参考 NY4 / NY5 频率计算公式
- NY5+ / NY6 / NY7 则是使用 Ri / Xi 的值做为播放速度，R0 = 0xA，则播放速度即为 10k。
- NX1 不支持指定播放速度。

Storage：播放的档案所在的储存空间，若不指定则为内部空间。

- NY4 / NY5 / NY5+ / NY6 / NY7 不支持指定储存空间。
- NX1 支援 SPI0 / SPI1 / UDR。

注意：NY4 / NY5 中，当使用 Voicename{*n}指令时，系统会按照其之前一个 Voice 的采样率来播放 Voice，其自身没有默认的采样率，所以，如果程序中的 Voice 使用了不同的采样率，可能引起同一首 Voicename{*n}多次执行时使用不同的采样率来播放。

例. 指定欲播放的文件

```
P1: PlayV(Ch1,$V1,6k)/PlayVS(Ch1,$V1,6k) ; 使用 Voice Label 指定欲播放的文件
P2: PlayV(ch0, X1:X0, 10k), PB.3=1        ; 播放被寄存器内容值所指到的音档（若 X1=0x01，
                                            X0=0xF2，则播放编号 V498 音文件），于语音通道 0，
                                            播放频率 10k，结束后执行 PB.3=1。
P3: PlayV(ch0, R3:R2:R1:R0, 10k), PB.3=1  ; 播放被寄存器内容值所指到的音档（若 R3=0x0，
                                            R2=0x3，R1=0XF，R0=0x1，则播放编号 V1009 音文
                                            件），于语音通道 0，播放频率 10k，结束后执行 PB.3=1。
P4: PlayV(ch0, X1:X0++,10k), PB.3=1       ; 播放被寄存器内容值所指到的音档（若 X1=0x01，
```

X0=0xF2，则播放编号 V498 音档，X0 自动加 1，X0=0xF3），于语音通道 0，播放频率 10k，结束后执行 PB.3=1

P5: PlayVS(ch0, X1:X0, 10k), PB.3=1 ；播放被寄存器内容值所指到的音档（若 X1=0x01，X0=0xF2，则播放编号 V498 音文件），于语音通道 0，播放频率 10k，播放同时执行 PB.3=1。

P6: PlayVS(ch0, R3:R2:R1:R0, 10k), PB.3=1 ；播放被寄存器内容值所指到的音档（若 R3=0x0，R2=0x3，R1=0XF，R0=0x1，则播放编号 V1009 音文件），于语音通道 0，播放频率 10k，播放同时执行 PB.3=1。

P7: PlayVS(ch0, X1:X0++,10k), PB.3=1 ；播放被寄存器内容值所指到的音档（若 X1=0x01，X0=0xF2，则播放编号 V498 音档，X0 自动加 1，X0=0xF3），于语音通道 0，播放频率 10k，播放同时执行 PB.3=1。

例. 指定播放速度

```
P1: PlayV(Ch1, $V0,6.35k) / PlayVS(Ch1, $V0,6.35k)      ; 直接设定播放速度
P2: PlayV(Ch1, $V0) / PlayVS(Ch1, $V1)          ; 不写播放速度使用该文件的采样频率为播放速度
P3: PlayV(Ch1, $V0, R1:R0)                      ; 使用 2 个 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. 重复播放

```
P1: PlayV(Ch1, $V0) *3                          ; 重复播放 3 次
```

例. 播放语音同时执行背景路径

```
PlayV(Ch1, $V0)&[BG1,BG2] / PlayVS(Ch1, $V1)&[BG1,BG2]  ; 播放语音同时执行 BG1 与 BG2
PlayV(Ch1, $V0)&[BG1,X] / PlayVS(Ch1, $V0)&[BG1,X]      ; 播放语音同时执行 BG1，Background2 保持不变。
PlayV(Ch1, $V0)&[OFF,X] / PlayVS(Ch1, $V0)&[OFF,X]      ; 播放语音并关闭 Background1，Background2 保持不变。
```

例. 根据 **[QIO]** 设定播放语音并输出 QIO 信号。.

```
[Voice File]
V1 = Qio.nyq                                    ; 加入 voice 文件。
[Output State]
Output_0: Q   Q   Q   Q   Q   Q   Q   Q   ; 使用 PA 及 PB 输出 QIO 信号。
[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 输出 Q9 ~ Q16。
[Path]
Path1: Output_0, PlayV(Ch0, $V1)                ; 播放 V1 并根据 MQIO 设定输出 QIO 信号。
```

例. 播放 SPI Flash 上的语音。

```
[Variable]
Var8: R0
[Memory]
SPI0_File = Spi.spiprj
[Path]
PowerOn: R0 = 0
Path1: PlayV(Ch0, R0++, SPI0)                ; 播放 SPI Flash 上由 R0 所指定的语音。
```

5.5.1.1 PlayV / PlayVS with Table

使用读表的方式来播放 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
; 将 Table 指定位置值存放至 R0，播放 R0 指定音源文件（R0=0x5，播放编号 V5 音源文件），于语音通道 0，结束后执行 PB.3=1。
TR2: TableL(Tab,1,1,X0), PlayV(ch0, X0), PB.3=1
; 将 Table 指定位置值存放至 X0，播放 X0 指定音源文件（X0=0x05，播放编号 V5 音源文件），于语音通道 0，结束后执行 PB.3=1。
TR3: TableL(Tab,1,1,X0), PlayV(ch0, $V0, X0), PB.3=1
; 将 Table 指定位置值存放至 X0，播放编号 V0 音源文件，于语音通道 0，播放频率为 X0（X0=0x05，播放频率为 5K），结束后执行 PB.3=1。
TR4: TableL(Tab,1,1,R0), PlayVS(ch0, R0), PB.3=1
; 将 Table 指定位置值存放至 R0，播放 R0 指定音源文件（R0=0x5，播放编号 V5 音源文件），于语音通道 0，播放同时执行 PB.3=1。
TR5: TableL(Tab,1,1,X0), PlayVS(ch0, X0), PB.3=1
; 将 Table 指定位置值存放至 X0，播放 X0 指定音源文件（X0=0x05，播放编号 V5 音源文件），于语音通道 0，播放同时执行 PB.3=1。
TR6: TableL(Tab,1,1,X0), PlayVS(ch0, $V0, X0), PB.3=1
; 将 Table 指定位置值存放至 X0，播放编号 V0 音源文件，于语音通道 0，播放频率为 X0（X0=0x05，播放频率为 5K），播放同时执行 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，则 Voice 播放完毕后，再执行下一个指令。没有 Voice 播放则立即执行下一个指令。

Ch：选择等待播放结束的语音通道，不带此参数，则表示等待全部语音通道的 Voice 播放结束。

- NY4 不能指定通道。
- NY5 / NY5+支持的语音通道为 Ch0 ~ Ch3。
- NY6 支持的语音通道为 Ch0 ~ Ch5。
- NY7 支持的语音通道为 Ch0 ~ Ch7。
- NX1 支持的语音通道为 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)          ; 与 TR1 的执行结果相同。
```

5.5.3 PauseV(Ch)

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

用来暂停指定通道的 Voice 文件播放。

Ch：指定要暂停播放的语音通道。不指定，则暂停全部语音通道的播放。

- NY4 不能指定通道。
- NY5 / NY5+支持的语音通道为 Ch0 ~ Ch3。
- NY6 支持的语音通道为 Ch0 ~ Ch5。
- NY7 支持的语音通道为 Ch0 ~ Ch7。
- NX1 支持的语音通道为 Ch0 ~ Ch3。

例.

```
PowerOn: InputState
TR1: PlayV(Ch0,$V0), PlayV(Ch1,$V1), PlayV(Ch2,$V2), PlayV(Ch3, $V3)
TR2: PauseV                               ; 暂停所有的 Voice 播放。
TR3: ResumeV                              ; 恢复所有的 Voice 播放。
TR4: PauseV(0)                            ; 暂停语音通道 0 的 Voice 播放。
TR5: ResumeV(0)                           ; 恢复语音通道 0 的 Voice 播放。
TR6: PauseV(1)                            ; 暂停语音通道 1 的 Voice 播放。
TR7: ResumeV(1)                           ; 恢复语音通道 1 的 Voice 播放。
TR8: PauseV(2)                            ; 暂停语音通道 2 的 Voice 播放。
TR9: ResumeV(2)                           ; 恢复语音通道 2 的 Voice 播放。
```

```
TR10: PauseV(3)                    ; 暂停语音通道 3 的 Voice 播放。
TR11: ResumeV(3)                   ; 恢复语音通道 3 的 Voice 播放。
```

5.5.4 ResumeV(Ch)

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

用来恢复指定通道的 Voice 文件播放。

Ch：指定要恢复播放的语音通道。不指定，则恢复全部语音通道的播放。

- NY4 不能指定通道。
- NY5 / NY5+支持的语音通道为 Ch0 ~ Ch3。
- NY6 支持的语音通道为 Ch0 ~ Ch5。
- NY7 支持的语音通道为 Ch0 ~ Ch7。
- NX1 支持的语音通道为 Ch0 ~ Ch3。

以下情况指令不再有效：

1. Voice 文件已经播放结束。
2. 有执行过 StopV 或 Stop 指令。

例.

```
PowerOn: InputState
TR1: PlayV(ch0,$V0), PlayV(ch1,$V1), PlayV(ch2,$V2), PlayV(ch3, $V3)
TR2: PauseV                        ; 暂停所有的 Voice 播放。
TR3: ResumeV                       ; 恢复所有的 Voice 播放。
TR4: PauseV(0)                     ; 暂停语音通道 0 的 Voice 播放。
TR5: ResumeV(0)                    ; 恢复语音通道 0 的 Voice 播放。
TR6: PauseV(1)                     ; 暂停语音通道 1 的 Voice 播放。
TR7: ResumeV(1)                    ; 恢复语音通道 1 的 Voice 播放。
TR8: PauseV(2)                     ; 暂停语音通道 2 的 Voice 播放。
TR9: ResumeV(2)                    ; 恢复语音通道 2 的 Voice 播放。
TR10: PauseV(3)                    ; 暂停语音通道 3 的 Voice 播放。
TR11: ResumeV(3)                   ; 恢复语音通道 3 的 Voice 播放。
```

5.5.5 StopV(Ch)

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

用来停止指定通道的 Voice 文件播放。

Ch：指定要停止播放的语音通道。不指定，则停止全部语音通道的播放。

- NY4 不能指定通道。
- NY5 / NY5+支持的语音通道为 Ch0 ~ Ch3。
- NY6 支持的语音通道为 Ch0 ~ Ch5。

- NY7 支持的语音通道为 Ch0 ~ Ch7。
- NX1 支持的语音通道为 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 的代号或文件名来直接播放语音时才能使用该指令，而不能用于 PlayV 的指令下。要使用该指令，需要在 play 前先下达一次 Freq=nK 播放速度的指令，之后的 play 动作会延用上次指定的播放速度。

CH：指定播放的语音通道

- NY4 不能指定通道。
- NY5 / NY5+支持的语音通道为 Ch0 ~ Ch3。

n：播放速度，n=4K~44.1K（频率对照表请参考频率计算公式）。

例. **Freq=10k, $V0**

5.5.7 V_Chx_Freq = nK

[NY5+]

修改正在播放语音的通道 S.R.。设定的 S.R.数值范围为 4K~44.1K。此设定不会影响下一次播放语音的 S.R.。

Chx：指定播放的语音通道。

- NY5+支持的语音通道为 Ch0~Ch3。

注意：当指定的语音通道没有在播放语音，此指令无效。

例. **V_Ch0_Freq = 10K ; 将语音通道 0 的 S.R.设为 10K。**

5.5.8 V_Chx_Vol = n / V_Chx_Vol = Xi

[NY5+ / NY6 / NY7]

修改正在播放语音的通道音量。设定的音量数值范围为 0~255，可为立即值或用 Xi 来控制；设为 0 时表示该语音通道为静音，设为 255 时表示将该语音通道音量维持在最大音量。

Chx：指定播放的语音通道。

- NY5+支持的语音通道为 Ch0 ~ Ch5。
- NY6 支持的语音通道为 Ch0 ~ Ch5。
- NY7 支持的语音通道为 Ch0 ~ Ch7。

注意：当指定的通道没有在播放语音，此指令无效。

```
例. V_Ch0_Vol = 200          ; 将语音通道 0 的音量设为 200，约原本音量的 78%。
例. V_Ch1_Vol = X0           ; 取 X0 内的数值当成通道 1 的音量。
```

5.5.9 Xi = V_Chx_Vol

[NY5+ / NY6 / NY7]

读取正在播放语音的通道音量。

Chx：指定播放的语音通道。

- NY5+支持的语音通道为 Ch0 ~ Ch3。
- NY6 支持的语音通道为 Ch0 ~ Ch5。
- NY7 支持的语音通道为 Ch0 ~ Ch7。

```
例. X0 = V_Ch0_Vol           ; 读取语音通道 0 的音量，并将结果存放 X0。
```

5.5.10 SBC_Loop_On

[NX1]

SBC 循环播放。

注意：

1. 当使用循环播放功能时，SBC-1 和 SBC-2 分别仅提供单一通道可使用。

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 音档播放。
TR2: SBC_Loop_On             ; 打开循环播放。
TR3: SBC_Loop_Off            ; 停止循环播放。
```

5.5.11 SBC_Loop_Off

[NX1]

停止 SBC 循环播放。停止循环播放后，仍会把音档完整播放完毕。

注意：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 循环播放。

注意：

1. 当使用循环播放功能时，仅提供单一通道可循环播放。

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 音档播放。
TR2: ADPCM_Loop_On                      ; 打开循环播放。
TR3: ADPCM_Loop_Off                     ; 停止循环播放。
```

5.5.13 ADPCM_Loop_Off

[NX1]

停止 ADPCM 循环播放。停止循环播放后，仍会把音档完整播放完毕。

注意：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 设定升采样。

ADPCM_UpSampling(Factor)

Factor：升采样倍数，可设定 1~8。

注意：

1. 当使用升采样功能时，仅提供单一通道可升采样。

2. 升采样后不可大于 64 KHz。

例.

```
PowerOn: ADPCM_UpSampling(2)            ; ADPCM 使用 2 倍升采样。
TR1: PlayV(ch0, $V0)                    ; ADPCM 音檔播放。
```

5.5.15 ADM_Loop_On

[NX1]

ADM 循环播放。

注意：

1. 当使用循环播放功能时，仅提供单一通道可循环播放。

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 音档播放。
TR2: ADM_Loop_On                        ; 打开循环播放。
TR3: ADM_Loop_Off                       ; 停止循环播放。
```

5.5.16 ADM_Loop_Off

[NX1]

停止 ADM 循环播放。停止循环播放后，仍会把音档完整播放完毕。

注意：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 设定升采样。

ADM_UpSampling(Factor)

Factor：升采样倍数，可设定 1~8。

注意：

1. 当使用升采样功能时，仅提供单一通道可升采样。

2. 升采样后不可大于 64 KHz。

例.

```
PowerOn: ADM_UpSampling(2)              ; ADM 使用 2 倍升采样。
TR1: PlayV(Ch0, $V0)                    ; ADM 音檔播放。
```

5.5.18 PCM_Loop_On

[NX1]

PCM 循环播放。

注意：

1. 当使用循环播放功能时，仅提供单一通道可循环播放。

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 音档播放。
TR2: PCM_Loop_On                        ; 打开循环播放。
TR3: PCM_Loop_Off                       ; 停止循环播放。
```

5.5.19 PCM_Loop_Off

[NX1]

停止 PCM 循环播放。停止循环播放后，仍会把音档完整播放完毕。

注意：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 播放列表的档案数量。

ReadFileCountV(Ri)

ReadFileCountV(Ri, Storage)

Ri：用于取得数量的变量，取得的数据为 0~65535。

Storage：用于指定特定储存空间的档案数量，可使用 UDR / SPI0 / SPI1，不指定时则为 **[Voice File]** 内的数量。

例.

```
[Variable]
Var16: voice_max=0, voice_idx=0
[Path]
PowerOn: ReadFileCountV(voice_max)          ; 取得 Voice File 数量。
TR1:     voice_idx++,                       ; voice_idx+1。
         voice_idx>=voice_max?{voice_idx=0}, ; index 超过边界，重置为 0。
         PlayV(ch0, voice_idx)              ; 播放 voice_idx 曲目。
```

5.6 录音指令（Record Command）

Record Command				
Record	RecordS	WaitRN	StopR	EraseR
EraseRS	WaitEN	-	-	-

5.6.1 Record / RecordS

[NX1]

进行录音。录音完成后，可以通过 PlayV 播放。

Record(Label {,Time})

RecordS (Label {, Time})

Record：待录音结束后，执行下一个指令。

RecordS：开始录音后，立即执行下一个指令。

Label：**[Record]** 中定义的录音区段名称。

Time：录音的长度，未定则为该录音区段的长度。

例.

```
[Path]
TR1: Record($Rec0)                    ; 进行录音，使用 Rec0 区段。
TR2: PlayV(ch0, $Rec0)                    ; 使用 PlayV 播放录音结果
```

5.6.2 WaitRN

[NX1]

若录音正在进行中，则在录音完毕后，再执行下一个指令。没有在录音则立即执行下一个指令。

注意：Record=RecordS+WaitRN。

例.

```
[Path]
TR1: Record($Rec0)
TR2: RecordS($Rec0), WaitRN           ; 与 TR1 的执行结果相同。
```

5.6.3 StopR

[NX1]

用来停止录音。

例.

```
PowerOn: InputState
TR1R: Record($Rec0)
```

```
TR2F: StopR                                    ; 停止当前的录音。
```

5.6.4 EraseR / EraseRS

[NX1]

对 SPI Flash 上的录音区段进行擦除动作。

指令格式如下：

EraseR(Label) { & [BG1 {,BG2 {,BG3}}] }
EraseRS (Label) { & [BG1 {,BG2 {,BG3}}] }

EraseR：待擦除结束后，执行下一个指令。
EraseRS：开始擦除后，立即执行下一个指令。
Label：**[Record]** 中定义的录音区段名称。

例.

```
EraseR($Rec0)
```

5.6.5 WaitEN

[NX1]

若 SPI Flash 擦除正在进行中，则在擦除完毕后，再执行下一个指令。没有在擦除则立即执行下一个指令。

注意：

1. EraseR=EraseRS+WaitEN。
2. 打开即擦即录功能时，此指令不可使用。

例.

```
[Path]
TR1: EraseR($Rec0)
TR2: EraseRS($Rec0), WaitEN                    ; 与 TR1 的执行结果相同。
```

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* 提供用户简易的指令来达成播放组合句子的方式。指令格式如下：

PlayS(Parameter)

PlayS：待句子组合播放结束后，执行下一个指令。

Parameter：设定欲播放的组合句子代号　　*例.* **PlayS($S1)**

注意：NY5+ / NY6 / NY7 / NX1 中。句子功能主要用来将多个资源文件（Melody、Speech 及 Action 等）组合成一个句子使用，因此 Sentence 段落中只允许 Play 及 Delay 等指令，其余指令不可使用在 Sentence 段落内。

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($S1)

[Background2]
BG3: PlayS($S1)
```

5.7.2 WaitSN(n)

[NY6 / NY7 / NX1]

若有句子正在播放，则句子播放完毕后，再执行下一个指令。没有句子播放中则立即执行下一个指令。

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，则等待句子播放完毕后返回，执行下一个指令。

例. NY7 使用 WaitSN 指令

```
[Path]
TR1: PlayS($S1)
TR2: WaitSN(0), PB.0=1                    ; 待句子播放完毕，将 PB.0 设为 1。
```

5.7.3 PauseS(n)

[NY6 / NY7 / NX1]

可以暂停指定的句子播放。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，则暂停所有的句子播放。

例.

```
PowerOn: InputState
TR1: PlayS($S1)
TR2: PauseS                               ; 暂停所有的句子播放。
TR3: ResumeS                              ; 恢复所有的句子播放。
```

```
TR4: PauseS(0)                    ; 暂停前景的句子播放。
TR5: ResumeS(0)                   ; 恢复前景的句子播放。
```

5.7.4 ResumeS(n)

[NY6 / NY7 / NX1]

可以恢复指定的句子播放。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，则恢复所有的句子播放。

以下情况指令不再有效：

1. 句子已经播放结束。
2. 有执行过 StopS 或 Stop 指令。

例.

```
PowerOn: InputState
TR1: PlayS($S1)
TR2: PauseS                       ; 暂停所有的句子播放。
TR3: ResumeS                      ; 恢复所有的句子播放。
TR4: PauseS(0)                    ; 暂停前景的句子播放。
TR5: ResumeS(0)                   ; 恢复前景的句子播放。
```

5.7.5 StopS(n)

[NY6 / NY7 / NX1]

用来停止句子播放。可以用 n 参数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，则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：在对 SPI Flash 下任何指令前，请确定 SPI Flash 在稳定可读写状态。依 SPI Flash 型号的不同，上电后至稳定可读写的时间也不同，约 200us ~ 15ms 不等。PCB 板上的电容也会影响上电后 SPI

Flash 到达可工作电压的时间，以及下电后放电时间。因此，若 SPI Flash 下电后重新上电应检查 SPI Flash 状态或适当地延时后再作读写。

5.8.1 SPIPlay / SPIPlayS

[NY6B / NY6C / NY7 / NX1]

Q-Code 提供用户利用简易的指令来达成播放存在于 SPI Flash 的文件，支持 2 组 SPI 界面的设定，但这 2 组 SPI 接口无法同时播放。指令格式如下：

SPIPlay ({Ch, } Index {++}{, SPIGroup}) {*n} {&[BG1 {,BG2 {,BG3}}]}

SPIPlayS ({Ch, } Index {++}{, SPIGroup}) {&[BG1 {,BG2 {,BG3}}]}

SPIPlay ({Ch, } Index {++}, Px.n) {*n} {&[BG1 {,BG2 {,BG3}}]}

SPIPlayS ({Ch, } Index {++}, Px.n) {&[BG1 {,BG2 {,BG3}}]}

SPIPlay：待 SPI 文件播放结束后，执行下一个指令。

SPIPlayS：SPI 文件开始播放，立即执行下一个指令。

Ch：指定要使用的输出通道 (***NX1 Only***)。

Index：播放文件的索引值，可为立即值或变量。

- 使用立即值索引
- 使用 Ri / Xi 索引，可使用 1 ~ 4 个 Ri 或是 1 ~ 2 个 Xi 的组合(NY6 / NY7) 或是单一变数 (NX1)。

SPIGroup：选择播放的 SPI 接口。

- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

Px.n：当 SPI 文件为 action 时输出的脚位。

NY7 的 SPIPlay 功能所提供的语音转文件数据格式和所支持的采样率如下：

Sample Rate	PCM8	PCM12	ADPCM6
7.8K	✓	✓	✓
15.6K	✓	✓	✓

注意：

1. ***语音文件需使用 SPI Encoder 转成.bin 文件后，再烧录到 SPI Flash 才可通过 Q-Code 播放。***
2. ***Encode 后的文件，其 Header 会包含文件型态（句子或语音），用户只需要给予 Index 即可播放。***
3. ***播放的 Index 如超过 SPI Flash 内的文件数，则会改播放 Index 0 的文件。***
4. ***NY6 / NY7 中，SPI 文件无法跟语音(PlayV / PlayVS)或 melody(PlayM / PlayMS)同时播放，两者会互相打断。***
5. ***NY6 / NY7 使用 SPIPlay 进行 action 播放时，固定使用 Ch1。***
6. ***使用 NX1 时，有指定 Px.n 参数时，表示用于 action 播放，Ch 参数表示 action 通道，可以使用 Ch1 ~ Ch32，若选择播放的文件不为 action 则此指令无作用。未指定 Px.n 参数时，表示用于 Voice / MIDI 播放，Ch 参数表示声音输出通道，仅可使用 Ch0 ~ Ch7，此时若选择播放的文件不为 Voice***

/ MIDI 时则此指令无作用。

7. NY6 同时播放 SPI flash 中的 voice 与 action 时，若语音压缩率为 1 ~ 9 且采样率超过 16kHz，或是语音压缩率为 10 ~ PCM 且采样率超过 10kHz，则 action 不会输出。

例. NY7 指定欲播放的文件索引值。

```
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 界面。

```
SPIPlay(1, SPI1) / SPIPlay(1, SPI2)           ; 指定 SPI 界面
SPIPlay(R1:R0)                                ; 不指定 SPI 接口，使用 SPI1
```

例. NY6 播放 SPI Flash 中的 action。

```
SPIPlay(0, PA.3)                              ; 播放 Index 0，将 action 输出至 PA.3。
```

例. NX1 播放 SPI Flash 中的 action。

```
SPIPlay(Ch1, 0, PA.3, SPI0)                   ; 播放 Index 0，使用 Ch1 将 action 输出至 PA.3。
```

例.

```
P1: SPIPlay(5), PB.3=1                        ; 播放 Index 5，结束后执行 PB.3=1。
P2: SPIPlayS(5), PB.3=1                       ; 播放 Index 5 后，马上执行 PB.3=1。
P3: SPIPlay(R1:R0, SPI1)                      ; 使用 R1:R0 为索引值来播放文件，R1 为高四位，
                                              R0 为低四位。
P4: SPIPlay(X0++)                             ; 使用 X0 为索引值来播放文件，播放后 X0 自动加 1。
```

5.8.2 SPIWaitN(Ch)

[NY6B / NY6C / NY7 / NX1]

等待 SPIPlay 播放完毕后，再执行下一个指令，没有 SPIPlay 正在执行则立即执行下一个指令。

Ch：选择等待播放结束的通道。不带此参数，则表示全部通道的播放结束。此参数作用于声音输出通道，而非 action 通道。有指定此参数时，仅作用于播放中的语音。未指定此参数则作用于所有经由 SPIPlay 播放的媒体。

- NY6 / NY7 不支持指定通道。
- NX1 支持 Ch0 ~ Ch7。

5.8.3 SPIStop(Ch)

[NY6B / NY6C / NY7 / NX1]

用来停止 SPI 播放。

Ch：选择停止播放的通道。若省略不写则相当于全部停止播放。此参数作用于声音输出通道，而非 action 通道。有指定此参数时，仅作用于播放中的语音。未指定此参数则作用于所有经由 SPIPlay 播放的媒体。

- NY6 / NY7 不支持指定通道。

- NX1 支持 Ch0 ~ Ch7。

5.8.4 SPIPause(Ch)

[NY6B / NY6C / NY7 / NX1]

用来暂停 SPI 播放。

Ch：选择暂停播放的通道。若省略不写则全部暂停播放。此参数作用于声音输出通道，而非 action 通道。有指定此参数时，仅作用于播放中的语音。未指定此参数则作用于所有经由 SPIPlay 播放的媒体。

- NY6 / NY7 不支持指定通道。
- NX1 支持 Ch0 ~ Ch7。

注意：如果在没有 SPIPlay 的情况下，此动作会被忽略。

5.8.5 SPIResume(Ch)

[NY6B / NY6C / NY7 / NX1]

用来恢复 SPI 播放。此指令不带参数。

Ch：选择恢复播放的通道。若省略不写则全部恢复播放。此参数作用于声音输出通道，而非 action 通道。有指定此参数时，仅作用于播放中的语音。未指定此参数则作用于所有经由 SPIPlay 播放的媒体。

- NY6 / NY7 不支持指定通道。
- NX1 支持 Ch0 ~ Ch7。

5.8.6 SPIVol = n / SPIVol = Ri

[NY6B / NY6C / NY7 / NX1]

设定 SPIPlay 播放的语音音量。

n：音量，可为立即值或用 Ri 来控制。

- NY6 数值范围为 0~15；n=0 时表示为静音，n=15 时表示将音量维持在最大音量。
- NY7 数值范围为 0~4；n=0 时表示为静音，n=4 时表示将音量维持在最大音量。

```
例. SPIVol = 3          ; 将 SPI 语音音量设为 3。
例. SPIVol = R0         ; 取 R0 内的数值当成 SPI 语音的音量。
```

5.8.7 Ri = SPIVol

[NY6B / NY6C / NY7 / NX1]

读取 SPIPlay 播放的语音音量。

```
例. R0 = SPIVol         ; 读取 SPI 语音音量，并存于 R0。
```

5.8.8 SPIGetIndex

[NY6B / NY6C / NY7 / NX1]

读取 SPI Flash 内的文件索引数目。

SPIGetIndex(Result{, SPIGroup})

Result：文件索引数目

- NY6 / NY7 支持 RI:Rk:Rj:Ri 或 Xi:Xj 的组合，其中 Ri=bit0 ~ 3，Rj=bit4 ~ 7，Rk=bit8 ~ 11，RI=bit12 ~ 15 / Xi=bit0 ~ 7，Xj=bit8 ~ 15。
- NX1 仅支持一个变量。

SPIGroup：选择读取的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

例. **SPIGetIndex(R1:R0)** ；读取第一组 SPI 的文件索引值数目。

例. **SPIGetIndex(R2:R1:R0, SPI2)** ；读取第二组 SPI 的文件索引值数目。

例. **SPIGetIndex(X0)** ；读取第一组 SPI 的文件索引值数目。

例. **SPIGetIndex(X1:X0, SPI2)** ；读取第二组 SPI 的文件索引值数目。

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. ***在对 SPI Flash 下任何指令前，请确定 SPI Flash 在稳定可读写状态。依 SPI Flash 型号的不同，上电后至稳定可读写的时间也不同，约 200us ~ 15ms 不等。PCB 板上的电容也会影响上电后 SPI Flash 到达可工作电压的时间，以及下电后放电时间。因此，若 SPI Flash 下电后重新上电应检查 SPI Flash 状态或适当地延时后再作读写。***
2. ***不同的 SPI Flash 型号对于 Qual Mode 的设置不同，需留意 QE bit 是否已适当地设置。***

5.9.1 SPI_WREN

[NY6B / NY6C / NY7 / NX1]

对 SPI Flash 传送 Write Enable 指令，将 status register 的 WEL 位（bit1）设为 1，使 SPI Flash 处于可写入状态。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. ***此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0x06。***
2. ***指令执行后，status register 的 WEL 位（bit1）会被设为 1，此时 SPI Flash 的写入动作才有效，用户可通过 SPI_RDSR 指令确认位状态。***

例.

```
SPI_WREN                        ; 传送 Write Enable 指令。
```

例. NY7 / NX1

```
SPI_WREN(SPI1)                  ; 通过 SPI1 传送 Write Enable 指令。
```

例. NY7

```
SPI_WREN(SPI2)                  ; 通过 SPI2 传送 Write Enable 指令。
```

5.9.2 SPI_WRDIS

[NY6B / NY6C / NY7 / NX1]

对 SPI Flash 传送 Write Disable 指令，将 status register 的 WEL 位（bit1）清为 0，使 SPI Flash 处于不可写入的状态。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. ***此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0x04。***
2. ***指令执行后，status register 的 WEL 位（bit1）会被清为 0，此时 SPI Flash 的写入动作无效，用户可通过 SPI_RDSR 指令确认位状态。***

例.

```
SPI_WRDIS                       ; 传送 Write Disable 指令。
```

例. NY7 / NX1

```
SPI_WRDIS(SPI1)                 ; 通过 SPI1 传送 Write Disable 指令。
```

例. NY7

```
SPI_WRDIS(SPI2)                 ; 通过 SPI2 传送 Write Disable 指令。
```

5.9.3 SPI_RDID(Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

对 SPI Flash 传送 Read Identification 指令，读取 SPI Flash 的 Manufacturer ID、Memory Type 及 Capacity 等信息。

Result：欲存放读回信息的变量，可为 Ri 或 Xi 的组合，需要 24-bit；高 8 位为 Manufacturer ID，中 8 位 Memory Type，低 8 位为 Capacity。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0x9F。

例. NY6B / NY6C / NY7

[Path]

TR1: SPI_RDID(R5:R4:R3:R2:R1:R0)

; 通过 SPI1 读取 SPI Flash 信息，将 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)

; 通过 SPI2 读取 SPI Flash 信息，将 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]

对 SPI Flash 传送 Chip Erase 指令，执行全内存擦除动作。

擦除动作执行时，系统不会进入睡眠模式，待擦除动作完成后，会自动执行“SPI_EraseEnd”路径。用户亦可利用 SPI_RDSR 指令读取 status register 的 WIP 位（bit0）状态，当擦除进行中，此位为 1，擦除完成，此位为 0。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。

- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. ***此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0xC7。***
2. ***此指令会自动将 WEL 位设为 1，用户不需额外执行 SPI_WREN 指令。***
3. ***依据不同大小的 SPI Flash，擦除动作可能需耗时数秒至数十秒，此期间除 SPI_RDSR 指令外，其余指令无效。***
4. ***擦除动作完成后，会自动执行"SPI_EraseEnd"路径，用户可通过此路径知道擦除动作完成时间点，以便执行后续动作。***

例. NY6B / NY6C / NY7 / NX1

[Path]

TR1: SPI_CE

; 对 SPI Flash 传送 Chip Erase 指令，执行全内存擦除动作。

例. NY7

[Path]

TR1: SPI_CE(SPI2)

; 通过 SPI2 对 SPI Flash 传送 Chip Erase 指令，执行全内存擦除动作。

例. NY6B / NY6C / NY7 / NX1

[Path]

Erase: SPI_CE, R0=1

SPI_EraseEnd: R0=0

; 对 SPI Flash 传送 Chip Erase 指令，擦除时将 R0 设为 1；擦除完成后，将 R0 设为 0。

5.9.5 SPI_SE(Addr, Count, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

对 SPI Flash 传送 Sector Erase 指令，执行 sector 擦除动作。一个 sector 的大小为 4KB，用户可指定一次要擦除的 sector 数量。

擦除动作执行时，系统不会进入睡眠模式，待擦除动作完成后，会自动执行"SPI_EraseEnd"路径。用户亦可利用 SPI_RDSR 指令读取 status register 的 WIP 位（bit0）状态，当擦除进行中，此位为 1，擦除完成，此位为 0。

Addr：指定要擦除的地址，可为立即值或由变量指定；一个 sector 大小为 4KB，使用立即值寻址时，需填入 24-bit 地址，但其中 Bit[11:0]为无效位；若使用变数寻址时，则仅需填入有效位 Bit[23:12]。

Count：要进行擦除的 sector 数量，范围 1~16，不填则默认为 1。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. *此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0x20。*
2. *此指令会自动将 WEL 位设为 1，用户不需额外执行 SPI_WREN 指令。*
3. *根据擦除的 sector 数量，擦除动作可能需耗时数十 ms 到数秒，此期间除 SPI_RDSR 指令外，其余指令无效。*
4. *擦除动作完成后，会自动执行"SPI_EraseEnd"路径，用户可通过此路径知道擦除动作完成时间点，以便执行后续动作。*

例. NY6B / NY6C / NY7

[Path]

TR1: SPI_SE(0x4321, 3)

; 将 SPI Flash 的 0x4000~0x6FFF 地址数据擦除。

TR2: R0=0x2, R1=0x1, SPI_SE(R1:R0, 2)

; 将 SPI Flash 的 0x12000~0x13FFF 地址数据擦除。

例. NY7

[Path]

Erase: X0=0x23, R2=0x1, SPI_SE(R2:X0, SPI1), R3=1

SPI_EraseEnd: R3=0

; 通过 SPI1 将 SPI Flash 的 0x123000~0x123FFF 地址数据擦除，擦除时将 R3 设为 1；擦除完成后，将 R3 设为 0。

例. NX1

[Variable]

Var32: Addr

Var8: R3

[Path]

TR1: SPI_SE(0x4321, 3)

; 透过 SPI0 将 SPI Flash 的 0x4000 ~ 0x6FFF 地址数据抹除。

TR2: Addr = 0x12, SPI_SE(Addr, 2, SPI1)

; 透过 SPI1 将 SPI Flash 的 0x12000 ~ 0x13FFF 地址数据抹除。

Erase: Addr = 0x123, SPI_SE(Addr, SPI0), R3=1

SPI_EraseEnd: R3=0

; 透过 SPI0 将 SPI Flash 的 0x123000 ~ 0x123FFF 地址数据抹除，抹除时将 R3 设为 1；抹除完成后，将 R3 设为 0。

5.9.6 SPI_BE(Addr, Count, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

对 SPI Flash 传送 Block Erase 指令，执行 block 擦除动作。一个 block 的大小为 64KB，用户可指定一次要擦除的 block 数量。

擦除动作执行时，系统不会进入睡眠模式，待擦除动作完成后，会自动执行“SPI_EraseEnd”路径。用户亦可利用 SPI_RDSR 指令读取 status register 的 WIP 位（bit0）状态，当擦除进行中，此位为 1，擦除完成，此位为 0。

Addr：指定要擦除的地址，可为立即值或由变量指定；一个 block 大小为 64KB，使用立即值寻址时，需填入 24-bit 地址，但其中的 Bit[15:0]为无效位；若使用变数寻址时，则仅需填入有效位 Bit[23:16]。

Count：要进行擦除的 block 数量，范围 1~16，不填则默认为 1。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. ***此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0xD8。***
2. ***此指令会自动将 WEL 位设为 1，用户不需额外执行 SPI_WREN 指令。***
3. ***根据擦除的 block 数量，擦除动作可能需耗时数秒，此期间除 SPI_RDSR 指令外，其余指令无效。***
4. ***擦除动作完成后，会自动执行“SPI_EraseEnd”路径，用户可通过此路径知道擦除动作完成时间点，以便执行后续动作。***

例. NY6B / NY6C / NY7 / NX1

[Path]

TR1: SPI_BE(0x14321, 2)

; 将 SPI Flash 的 0x10000~0x2FFFF 地址数据擦除。

TR2: R0=0x3, SPI_BE(R0, 3)

; 将 SPI Flash 的 0x30000~0x5FFFF 地址数据擦除。

例. NY7

[Path]

Erase: X0=0x17, SPI_BE(X0, SPI1), R3=1

SPI_EraseEnd: R3=0

; 通过 SPI1 将 SPI Flash 的 0x170000~0x17FFFF 地址数据擦除，擦除时将 R3 设为 1；擦除完成后，将 R3 设为 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]

对 SPI Flash 传送 Deep Power-down 指令，使之进入 deep power-down 模式，**建议系统进入睡眠模式前，执行此指令，以节省 SPI Flash 耗电。**

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. 此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0xB9。

2. SPI Flash 进入 deep power-down 后，除 SPI_RDP 唤醒指令外，其余指令无效。

例. NY6B / NY6C / NY7 / NX1

```
SPI_DP                          ；传送 Deep Power-down 指令。
```

例. NY7 / NX1

```
SPI_DP(SPI1)                    ；通过 SPI1 传送 Deep Power-down 指令。
```

例. NY7

```
SPI_DP(SPI2)                    ；通过 SPI2 传送 Deep Power-down 指令。
```

5.9.8 SPI_RDP(Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

对 SPI Flash 传送 Release Deep Power-down 指令，将其从 deep power-down 模式中唤醒，并读回 8-bit 的 Device ID。

Result：储存读回的 Device ID，可以为 Ri 或是 Xi 的组合，不填则表示不读取 ID。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0xAB。

例. NY7

[Path]

TR1: SPI_RDP(R1:R0)

; 通过 SPI1 将 SPI Flash 从 deep power-down 模式唤醒，并将 Device ID 储存在[R1:R0]。

TR2: SPI_RDP(SPI1)

; 通过 SPI1 将 SPI Flash 从 deep power-down 模式唤醒，不读取 Device ID。

TR3: SPI_RDP(X0, SPI2)

; 通过 SPI2 将 SPI Flash 从 deep power-down 模式唤醒，并将 Device ID 储存在 X0。

例. NX1

[Variable]

Var8: DevID

[Path]

TR1: SPI_RDP(DevID)

; 透过 SPI0 将 SPI Flash 从 deep power-down 模式唤醒，并将 Device ID 储存在 DevID。

TR2: SPI_RDP(SPI0)

; 透过 SPI0 将 SPI Flash 从 deep power-down 模式唤醒，不读取 Device ID。

TR3: SPI_RDP(DevID, SPI1)

; 透过 SPI1 将 SPI Flash 从 deep power-down 模式唤醒，并将 Device ID 储存在 DevID。

5.9.9 SPI_WRSR(Data, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

对 SPI Flash 传送 Write Status Register 指令，用来修改 Status Register 的数据。

Data：欲写入 Status Register 的数据，可为立即值，或 Ri / Xi 的组合。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. 此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0x01。

2. 此指令会自动将 WEL 位设为 1，用户不需额外执行 SPI_WREN 指令。

例. NY7

[Path]

TR1: SPI_WRSR(0x18)

; 通过 SPI1 写入 0x18 到 Status Register。

TR2: R0=0x8, R1=0x1, SPI_WRSR(R1:R0, SPI1)

; 通过 SPI1 写入 0x18 到 Status Register。

TR3: X0=0x18, SPI_WRSR(X0, SPI2)

; 通过 SPI2 写入 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]

对 SPI Flash 传送 Read Status Register 指令，用来读取 Status Register 的数据。

Result：储存读回的 Status Register 数据，可为 Ri / Xi 的组合。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0x05。

例. NY7

[Path]

TR1: SPI_RDSR(R1:R0)

; 通过 SPI1 读取 Status Register 数据，并储存到[R1:R0]。

TR2: SPI_RDSR(X0, SPI1)

; 通过 SPI1 读取 Status Register 数据，并储存到 X0。

TR3: SPI_RDSR(X0, SPI2)

; 通过 SPI2 读取 Status Register 数据，并储存到 X0。

例. **NX1**

[Variable]

Var8: SR

[Path]

TR1: SPI_RDSR(SR)

; 透过 SPI0 读取 Status Register 数据，并储存到 SR。

TR2: SPI_RDSR(SR, SPI0)

; 透过 SPI0 读取 Status Register 数据，并储存到 SR。

TR3: SPI_RDSR(SR, SPI1)

；透过 SPI1 读取 Status Register 数据，并储存到 SR。

5.9.11 SPI_WRD(Addr, Data, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

对 SPI Flash 传送 Page Program 指令，用来写入数据到指定地址，用户可搭配 SPI_TX 指令一次写入多个字节数据。当欲写入数据传送完毕后，需下达 SPI_CS_Off 指令，以执行写入动作。写入动作执行时，用户需利用 SPI_RDSR 指令读取 status register 的 WIP 位(bit0)状态，当写入进行中，此位为 1，写入完成，此位为 0。

Addr：要写入数据的 24-bit 地址，可为立即值或是由变量指定；由变量指定时，高位的变量可省略，省略的位默认为 0。

Data：要写入的数据，可为立即值或是由变数指定。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. ***此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0x02。***
2. ***指令会自动将 WEL 位设为 1，用户不需额外执行 SPI_WREN 指令。***
3. ***指令执行时会自动将 CS 脚位设为低电平，当欲写入数据传送完毕后，需下达 SPI_CS_Off 指令，以执行写入动作。***
4. ***当 SPI Flash 执行写入时，status register 的 WIP 位(bit0)会被设为 1，用户可通过 SPI_RDSR 指令读取位状态，当状态为 0，则表示写入完成。***
5. ***Page Program 指令一次最多能写入 256 个字节数据，每传送一字节数据，写入地址会自动加 1，但仅针对低 8 位，假设当前写入地址为 0x1FF，加 1 后会变 0x100。***
6. ***根据写入的数据数，可能需耗时数百 ms，此期间除 SPI_RDSR 指令外，其余指令无效。***

例. NY7

[Path]

TR1: SPI_WRD(0x4321, 0x32), SPI_CS_Off

；通过 SPI1 将 0x32 写入到 0x4321 地址。

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)

；通过 SPI2 将 0x34 写入到 0x12FF 地址，0x56 写入到 0x1200 地址，0x78 写入到 0x1201 地址。

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)

；通过 SPI1 将 0xAB 写入到 0x4321 地址，0xCD 写入到 0x4322 地址，0xEF 写入到 0x4323 地址。

例. NX1

[Variable]

Var32: Addr

Var8: Data

[Path]

TR1: SPI_WRD(0x4321, 0x32), SPI_CS_Off

; 透过 SPI0 将 0x32 写入到 0x4321 地址。

TR2: Addr = 0x12FF, Data=0x34, SPI_WRD(Addr, Data, SPI1),

SPI_TX(0x56, SPI1), SPI_TX(0x78, SPI1), SPI_CS_Off(SPI1)

; 透过 SPI1 将 0x34 写入到 0x12FF 地址，0x56 写入到 0x1200 地址，0x78 写入到 0x1201 地址。

TR3: SPI_WRD(0x4321, 0xAB), Data = 0xCD, SPI_TX(Data),

Data=0xEF, SPI_TX(Data), SPI_CS_Off

; 透过 SPI0 将 0xAB 写入到 0x4321 地址，0xCD 写入到 0x4322 地址，0xEF 写入到 0x4323 地址。

5.9.12 SPI_RDD(Addr, Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

对 SPI Flash 传送 Read Data 指令，从指定地址读取数据，用户可搭配 SPI_RX 指令一次读取多个字节数据。数据读取完毕后，需下达 SPI_CS_Off 指令，以结束读取动作。

Addr：指定要读取数据的 24-bit 地址，可为立即值或由变量指定；由变量指定时，高位的变量可省略，省略的位默认为 0。

Result：欲存放读取数据的变量。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. ***此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0x03。***
2. ***指令执行时会自动将 CS 脚位设为低电平，当数据读取完毕后，需下达 SPI_CS_Off 指令，以结束读取动作。***
3. ***Read 指令每读取一字节数据，读取地址会自动加 1。***

例. NY7

[Path]

TR1: SPI_RDD(0x4321, R1:R0), SPI_CS_Off

; 通过 SPI1 将 0x4321 地址内数据储存到[R1:R0]。

TR2: R0=0x3, X1=0x12, SPI_RDD(X1:R0, X2, SPI2), SPI_CS_Off(SPI2)

; 通过 SPI2 将 0x123 地址内数据储存到 X2。

TR3: SPI_RDD(0x1234, X0, SPI1), SPI_RX(R3:R2, SPI1), SPI_RX(X2, SPI1), SPI_CS_Off(SPI1)

; 通过 SPI1 将 0x1234 地址内数据储存到 X0，0x1235 地址内数据储存到[R3:R2]，0x1236 地址内数据

储存到 X2。

例. NX1

[Variable]

Var32: Addr

Var8: Data, Data0, Data1, Data2

[Path]

TR1: SPI_RDD(0x4321, Data), SPI_CS_Off

; 透过 SPI0 将 0x4321 地址内数据储存到 Data。

TR2: Addr=0x123, SPI_RDD(Addr, Data, SPI1), SPI_CS_Off(SPI1)

; 透过 SPI1 将 0x123 地址内数据储存到 Data。

TR3: SPI_RDD(0x1234, Data0, SPI1), SPI_RX(Data1, SPI1), SPI_RX(Data2, SPI1), SPI_CS_Off(SPI1)

; 透过 SPI1 将 0x1234 地址内数据储存到 Data0，0x1235 地址内数据储存到 Data1，0x1236 地址内数据储存到 Data2。

5.9.13 SPI_GetAddr(Index, Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

用户可通过 SPI Encoder 的 User Defined Sections 页面加入自定义信息文件或保留区块，再利用此指令取得各区块的地址，以执行读写或擦除动作。

当用户加入文件或定义保留区时，加入的顺序(Sec 栏位)即为索引值，将欲取得区块地址的索引值带入指令内参数，即可获得对应的地址；SPI Encoder 详细的操作方式请参考 SPI Encoder 使用手册。

Index：指定要取得区块地址的索引值，可为立即值或由变量指定，共 16-bit，有效范围为 0~65535；由变量指定时，高位的变量可省略，省略的位默认为 0。

Result：指定储存区块开始地址的变量，共 24-bit，高位的变量可省略不储存。

SPIGroup：选择对应的 SPI 界面。

- NY6 / NX1 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：此指令兼容于九齐 N25Q 系列 SPI Flash，指令码为 0x03。

例. NY7

[Path]

TR1: SPI_GetAddr(0x8, X2:X1:R1:R0)

; 通过 SPI1 读取索引值 8 的区块地址，并将地址的 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)

; 通过 SPI2 读取索引值 0x23 的区块地址，并将地址的 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)

；通过 SPI1 读取索引值 0x12 的区块地址，并将地址存在[R7:R2]。

例. NX1

[Variable]

Var32: Addr

Var16: Index

[Path]

TR1: SPI_GetAddr(0x8, Addr)

；透过 SPI0 读取索引值 8 的区块地址，并将地址存在 Addr。

TR2: Index=0x23, SPI_GetAddr(Index, Addr, SPI1)

；透过 SPI1 读取索引值 0x23 的区块地址，并将地址存在 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]

将 CS 脚位设为低电平。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：当任一 CS 脚位设为低电平时，系统将无法进入睡眠模式。

例.

SPI_CS_On **；将 SPI1 的 CS 脚位设为低电平。**

SPI_ CS_On(SPI1) **；将 SPI1 的 CS 脚位设为低电平。**

SPI_ CS_On(SPI2) **；将 SPI2 的 CS 脚位设为低电平。**

5.10.2 SPI_CS_Off(SpiGroup)

[NY6B / NY6C / NY7 / NX1]

将 CS 脚位设为高电平。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

例.

```
SPI_CS_Off                          ; 将 SPI1 的 CS 脚位设为高电平。
SPI_ CS_Off(SPI1)                   ; 将 SPI1 的 CS 脚位设为高电平。
SPI_ CS_Off(SPI2)                   ; 将 SPI2 的 CS 脚位设为高电平。
```

5.10.3 SPI_TX(Data, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

通过 SCK 及 MOSI 脚位传送 8-bit 数据到 SPI Flash；此指令可以用来传送指令或搭配 SPI_WRD 指令写入数据等，亦可用此指令组合出 *Q-Code* 不支持的指令。

Data：欲传送的 8-bit 数据，可为立即值或是由变数指定。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. ***此指令仅通过 SCK 及 MOSI 脚位送出 8-bit 数据，不影响 CS 脚位状态，因此用户需自行控制 CS 脚电平。***
2. ***此指令需搭配 SPI_WREN 及 SPI_WRD 指令才能将数据写入到 SPI Flash 内，写入方式请参考 SPI_WRD 指令说明。***

例. NY7

```
[Path]
TR1: SPI_CS_On, SPI_TX(0xB9), SPI_CS_Off
; 透过 SPI1 传送 0xB9 指令到 SPI Flash。
TR2: R0=0xB, R1=0xA, SPI_CS_On(SPI2), SPI_TX(R1:R0, SPI2), SPI_CS_Off(SPI2)
; 透过 SPI2 传送 0xAB 指令到 SPI Flash。
TR3: X0=0x06, SPI_CS_On(SPI1), SPI_TX(X0, SPI1), SPI_CS_Off(SPI1)
; 透过 SPI1 传送 0x06 指令到 SPI Flash。
```

例. NX1

```
[Variable]
Var8: Data
[Path]
TR1: SPI_CS_On, SPI_TX(0xB9), SPI_CS_Off
```

; 透过 SPI0 传送 0xB9 指令到 SPI Flash。

TR2: Data=0xAB, SPI_CS_On(SPI1), SPI_TX(Data, SPI1), SPI_CS_Off(SPI1)

; 透过 SPI1 传送 0xAB 指令到 SPI Flash。

TR3: Data=0x06, SPI_CS_On(SPI0), SPI_TX(Data, SPI0), SPI_CS_Off(SPI0)

; 透过 SPI0 传送 0x06 指令到 SPI Flash。

5.10.4 SPI_RX(Result, SPIGroup)

[NY6B / NY6C / NY7 / NX1]

通过 SCK 及 MISO 脚位从 SPI Flash 接收 8-bit 数据；此指令可用来读取 ID 或搭配 SPI_RDD 指令接收 SPI Flash 内数据等，亦可用此指令组合出 Q-Code 不支持的指令。

Result：读取的 8-bit 数据，可为 Ri / Xi 的组合。

SPIGroup：选择对应的 SPI 界面。

- NY6 不支持指定 SPIGroup。
- NY7 可为 SPI1 或 SPI2，不填则预设为 SPI1。
- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

注意：

1. 此指令仅通过 SCK 及 MISO 脚位接收 8-bit 数据，不影响 CS 脚位状态，因此用户需自行控制 CS 脚电平。

2. 此指令需搭配 SPI_RDD 指令才能读取 SPI Flash 内数据，读取方式请参考 SPI_RDD 指令说明。

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]

调整 SCK 透过 IHRC 除频的倍率。

Divisor：除频倍率，可设定 1 / 2 / 4 / 8 / 16 / 32 / 64 / 128，当 IHRC>32MHz 时默认值为 2，其余默认值为 1。

SPIGroup：选择读取的 SPI 界面。

- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

例.

```
SPI_CLKDIV(16, SPI1)                    ; 将 SPI1 的 SCK 的频率设定为 IHRC 的 1/16。
```

5.10.6 SPI_CPOL(Polarity, SPIGroup)

[NX1]

调整 SCK 闲置时的极性。

Polarity：0 (Low)、1 (High)，默认值为 0。

SPIGroup：选择读取的 SPI 界面。

- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

例.

```
SPI_CPOL(0, SPI1)                       ; 将 SPI1 的 CLK 闲置极性设为 Low。
```

5.10.7 SPI_CPHA(Phase, SPIGroup)

[NX1]

调整 SCK 的数据采样时机。

Phase：0 (1st Edge)、1 (2nd Edge)，默认值为 0。

SPIGroup：选择读取的 SPI 界面。

- NX1 可为 SPI0 或 SPI1，不填则预设为 SPI0。

例.

```
SPI_CPHA(0, SPI1)                       ; 将 SPI1 的 CLK 采样设置为 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]

对 Embedded Flash 传送 Sector Erase 指令，执行 sector 抹除动作。一个 sector 的大小为 512B，使用者可指定一次要抹除的 sector 数量。

抹除动作执行时，系统不会进入睡眠模式。

EF_SE(Addr)

EF_SE(Addr, Count)

Addr：指定要抹除的地址，可为立即值或由变量指定；一个 sector 大小为 512B，使用立即值寻址时，需填入 20-bit 地址，但其中 Bit[8:0]为无效位；若使用变数寻址时，则仅需填入有效位 Bit[20:9]。

Count：要进行抹除的 sector 数量，范围 1 ~ 16，不填则预设为 1。

注意：根据抹除的 sector 数量，抹除动作可能需耗时数十 ms 到数秒，此期间其余指令无效。

例.

[Variable]

Var32: Addr

[Path]

TR1: EF_SE(0x4321, 3)

; 将 Embedded Flash 的 0x4200 ~ 0x47FF 地址数据抹除。

TR2: Addr=0x12, EF_SE(Addr, 2)

; 将 Embedded Flash 的 0x2400 ~ 0x27FF 地址数据抹除。

5.11.2 EF_WRD

[NX1 EF]

对 Embedded Flash 传送 Program 指令，用来写入数据到指定地址。

EF_WRD(Addr, Data)

Addr：要写入数据的 20-bit 地址，可为立即值或是由变量指定；由变量指定时，高位的变量可省略，省略的位默认为 0。

Data：要写入的数据，可为立即值或是由变数指定。

注意：根据写入的资料数，可能需耗时数百 ms，此期间其余指令无效。

例.

[Variable]

```
Var32: Addr
Var8: Data
[Path]
TR1: EF_WRD(0x4321, 0x32)
; 将 0x32 写入到 0x4321 地址。
TR2: Addr=0x12FF, Data=0x34, EF_WRD(Addr, Data)
; 将 0x34 写入到 0x12FF 地址。
```

5.11.3 EF_RDD

[NX1 EF]

对 Embedded Flash 传送 Read Data 指令，从指定地址读取数据。

EF_RDD(Addr, Result)

Addr：指定要读取数据的 20-bit 地址，可为立即值或由变量指定；由变量指定时，高位的变量可省略，省略的位默认为 0。

Result：欲存放读取数据的变量。

例.

```
[Variable]
Var32: Addr
Var8: Data
[Path]
TR1: EF_RDD(0x4321, Data)
; 将 0x4321 地址内数据储存到 Data。
TR2: Addr=0x123, EF_RDD(Addr, Data)
; 将 0x123 地址内数据储存到 Data。
```

5.11.4 EF_GetAddr

[NX1 EF]

使用者可透过 Q-Code 的 User Defined Sections 页面加入自定义信息文件或保留区块，再利用此指令取得各区块的地址，以执行读写或抹除动作。

当使用者加入档案或定义保留区时，加入的顺序(Sec 字段)即为索引值，将欲取得区块地址的索引值带入指令内参数，即可获得对应的地址。

EF_GetAddr(Index, Result)

Index：指定要取得区块地址的索引值，可为立即值或由变量指定，共 16-bit，有效范围为 0 ~ 65535；由变量指定时，高位的变量可省略，省略的位默认为 0。

Result：指定储存区块开始地址的变量，共 20-bit，高位的变量可省略不储存。

例.

[Variable]

Var32: Index, Addr

[Path]

TR1: EF_GetAddr(0x8, Addr)

；读取索引值 8 的区块地址，并将地址存在 Addr。

TR2: Index=0x23, EF_GetAddr(Index, Addr)

；读取索引值 0x23 的区块地址，并将地址存在 Addr。

5.12 Storage 指令（Storage Command）

Comparator Command				
Storage_Save	-	-	-	-

5.12.1 Storage_Save

[NX1]

将 **[Storage Variable]** 中所定义的变量储存至 SPI Flash / Embedded Fash 中。

5.13 比较器指令（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]

用来启动 Comparator 功能；若带有计时源频率，则可以将两次比较成立的时间撷取下来，即 Capture 功能，计时源频率分别有 62.5K、250K、1M、4M。

用户打开 Comparator 或 Capture 功能，当比较器成立发生中断时，将会立即执行相对应的系统路径。

注意：

1. 第一次启动 Capture 时，Comparator 成立撷取的时间，会是错误的。

2. Capture 占用到 Timer 资源，无法再使用 Timer 计时功能。

例. 打开 Comparator 功能，成立时执行 INT_CMP 系统路径。

```
TR1:CMP_ON                    ；打开 Comparator 功能。
INT_CMP:!PB                   ；Comparator 成立时，立即将 PB 输出反向。
```

例. 打开 Capture 功能，Comparator 成立时执行 INT_CMP 系统路径，若 Comparator 持续没有成立，

则执行 INT_TMR 系统路径。

```
TR1:CMP_ON(1M)                  ; 打开 Capture 功能。
INT_CMP:CMP_Read(X0)            ; Comparator 成立时，立即读取撷取的时间。
INT_TMR:!PB                     ; Comparator 持续不成立时，立即将 PB 输出反向。
```

5.13.2 CMP_OFF

[NY6B / NY6C]

关闭 Comparator 功能。

```
例. TR2:CMP_OFF                 ; 关闭 Comparator 功能。
```

5.13.3 CMP_Read(Rj:Ri) / CMP_Read(Xi)

[NY6B / NY6C]

读取 Capture 的计数值。

例. 将 Capture 的计数值存放到 X0。

```
INT_CMP: CMP_Read(X0)           ; 读取 Capture 的计时值，并存放到 X0 寄存器。
```

例. 将 Capture 的计数值存放到 R0,R1。

```
INT_CMP: CMP_Read(R1:R0)        ; 读取 Capture 的计时值，并存放到 R0、R1 寄存器。
```

5.13.4 CMP_CNT_ON(Count)

[NY6B / NY6C]

用来启动外部输入计数功能，需带入计数次数，当次数达到将执行 INT_TMR 系统路径。

注意：Counter 占用到 Timer 资源，无法再使用 Timer 功能。

例. 打开 Counter 功能，设定计数 10 次 PB 输出反向。

```
TR1: CMP_CNT_ON(10)             ; 打开 Counter 功能，并设定计数 10 次。
INT_TMR:!PB                     ; 计数达到 10 次，立即将 PB 输出反向。
```

5.13.5 CMP_CNT_OFF

[NY6B / NY6C]

关闭 Counter 功能。

```
例. TR2:CMP_CNT_OFF             ; 关闭 Counter 功能。
```

5.14 定时器指令（Timer Command）

Counter Command				
TMR_ON	TMR_OFF	TMR_Read(Rj:Ri)	TMR_Read(Xi)	-

5.14.1 TMR_ON

[NY5+ / NY6 / NX1]

用来启动 Timer 计时功能，需带入计时时间。

TMR_On(Time)

TMR_On(Timer, Time)

Timer：定时器。

- NY5+支援 TMR0 / TMR1 / TMR2 / TMR3。
- NX1 支援 TMR0 / TMR1 / TMR2 / TMR3 / PWMA / PWMB。

Time：计时时间。

- NY5+支援 64 ~ 256us。
- NY6 支援 1.00ms ~ 4.00ms。
- NX1 OTP 计时时间随系统频率而有不同：
  - High Clock Frequency 为 12MHz 时，为 64 ~ 5461us。
  - High Clock Frequency 为 16MHz 时，为 64 ~ 4096us。
  - High Clock Frequency 为 24MHz 时，为 64 ~ 2730us。
  - High Clock Frequency 为 32MHz 时，为 64 ~ 2048us。
- NX1 EF 支援 64 ~ 1365us。

注意：

1. NY6 不支持 Timer 参数。

2. NY6 计时时间位于 1.00ms ~ 2.00ms 之间时，每次计时最小单位为 64us；当位于 2.01ms ~ 4.00ms 时，每次计时最小单位为 256us。若最小时间单位无法整除所设定的时间时，每次中断时间会有误差。

3. NY5+同时只能开启 1 个定时器功能，最后启动的定时器会将之前的关闭。

例. 打开 Timer 计时，每 1.5ms 中断一次，中断发生时执行 INT_TMR 系统路径。

```
TR1: TMR_ON(1.5ms)          ; 打开 Timer 计时功能，每 1.5ms 中断一次。
INT_TMR:!PA                 ; 每次 Timer 中断，立即将 PA 输出反向。
```

5.14.2 TMR_OFF

[NY5+ / NY6 / NX1]

关闭 Timer 功能。

TMR_Off

TMR_Off(Timer)

Timer：定时器。

- NX1 支援 TMR0 / TMR1 / TMR2 / TMR3 / PWMA / PWMB。

注意：NY5+ / NY6 不支持 Timer 参数。

例. **TR2:TMR_OFF** ；关闭 **Timer** 功能。

5.14.3 TMR_Read(Rj:Ri) / TMR_Read(Xi)

[NY6]

读取 Timer 的计数值。

例. 将 Timer 的计数值存放到 X0。

TR1: TMR_Read(X0) ；读取 **Timer** 的计时值，并存放到 **X0** 寄存器。

例. 将 Timer 的计数值存放到 R0,R1。

TR1: **TMR_Read(R1:R0)** ；读取 **Timer** 的计时值，并存放到 **R0**、**R1** 寄存器。

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* 提供用户简易的指令来达成播放音乐文件的方式。

PlayM ({Ch, }Parameter {++} {, Storage}) {*n} {& [BG1{,BG2 {,BG3}}] }

PlayMS ({Ch, }Parameter {++} {, Storage}) {& [BG1{,BG2 {,BG3}}] }

PlayM：待 Melody 文件播放结束后，执行下一个指令。

PlayMS：Melody 文件开始播放，立即执行下一个指令。

Ch：所要使用的声音输出通道。

- NY5 / NY5+ / NY6 / NY7 不支持指定通道。
- NX1 支持 0 ~ 3 或是 Ch0 ~ Ch3。

Parameter：欲播放的音乐文件。

- Melody file 文件代号

- Ri（可支持到 1~4 个 Ri）或是 Xi（可支持到 1~2 个 Xi），由 Ri / Xi 的值决定要播放的是第几个 melody 文件。若是 Ri / Xi 带有"++"则在播放后将其递增。

Storage：播放的档案所在的储存空间，若不指定则为内部空间。

- NY5 / NY5+ / NY6 / NY7 不支持指定储存空间。
- NX1 支援 SPI0 / SPI1。

例.

```
P1: PlayM($M0),PB.3=1                  ; 播放 Melody 音乐文件，结束后执行 PB.3=1。
P2: PlayM(R3:R2:R1:R0),PB.3=1          ; 播放被寄存器内容值所指到的音乐文件(若 R3=0x0，
                                       R2=0x3，R1=0xF，R0=0x1，则播放编号 M1009 音乐
                                       文件)，结束后执行 PB.3=1。
P3: PlayM(X1:X0),PB.3=1                ; 播放被寄存器内容值所指到的音乐文件(若 X1=0x01，
                                       X0=0xF2，则播放编号 M498 音乐文件)，结束后执行
                                       PB.3=1。
P4: PlayM(X1:X0++),PB.3=1              ; 播放被寄存器内容值所指到的音乐文件(若 X1=0x01，
                                       X0=0xF2，则播放编号 M498 音乐文件，且 X0 自动加 1，
                                       X0=0xF3)，结束后执行 PB.3=1。
P5: PlayMS($M0),PB.3=1                 ; 播放 Melody，播放同时执行 PB.3=1。
P6: PlayMS(R3:R2:R1:R0),PB.3=1         ; 播放被寄存器内容值所指到的音乐文件(若 R3=0x0，
                                       R2=0x3，R1=0xF，R0=0x1，则播放编号 M1009 音乐
                                       文件)，播放同时执行 PB.3=1。
P7: PlayMS(X1:X0),PB.3=1               ; 播放被寄存器内容值所指到的音乐文件(若 X1=0x01，
                                       X0=0xF2，则播放编号 M498 音乐文件)，播放同时执行
                                       PB.3=1。
P8: PlayMS(X1:X0++),PB.3=1             ; 播放被寄存器内容值所指到的音乐文件(若 X1=0x01，
                                       X0=0xF2，则播放编号 M498 音乐文件，且 X0 自动加 1，
                                       X0=0xF3)，播放同时执行 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

使用读表的方式来播放 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
; 将 Table 指定位置值存放至 R0，播放 R0 指定音乐文件(R0=0x5，播放编号 M5 音乐文件)，结束后执行 PB.3=1。
TR2: TableL(Tab,1,1,X0), PlayM(X0), PB.3=1
; 将 Table 指定位置值存放至 X0，播放 X0 指定音乐文件(X0=0x05，播放编号 M5 音乐文件)，结束后执行 PB.3=1。
TR4: TableL(Tab,1,1,R0), PlayMS(R0), PB.3=1
; 将 Table 指定位置值存放至 R0，播放 R0 指定音乐文件(R0=0x5，播放编号 M5 音乐文件)，播放同时执行 PB.3=1。
TR5: TableL(Tab,1,1,X0), PlayMS(X0), PB.3=1
; 将 Table 指定位置值存放至 X0，播放 X0 指定音乐文件(X0=0x05，播放编号 M5 音乐文件)，播放同时执行 PB.3=1。
```

动作	指令	说明
代号播放	PlayM($M0)	播放 Label “M0” 这个音乐文件。
RAM 播放	PlayM(X0)	播放 X0 内容值所指到音乐文件。
重复播放的次数	PlayM($M0)*n	Label “M0”这个音乐文件会被重复播放 n 次。
在播放声音档时一并调用背景动作	PlayM($M0)&[BG1,BG2]	在播放 Label “M0” 时会一并调用背景 1 与背景 2 的动作。
	PlayM($M0)&[X,BG2]	不改变 BG1 当前的动作。
	PlayM($M0)&[BG1,X]	不改变 BG2 当前的动作。
	PlayM($M0)&[OFF,BG2]	停止 BG1 当前的动作。
	PlayM($M0)&[BG1,OFF]	停止 BG2 当前的动作。

例.

```
[Melody Database]
D:\.qcODE\TEST\ADSR.qmd
; M0 = Music.mid, 4 channel
; M1 = Start.mid, 4 channel
```

```
[Path]
TR1: PlayM($M0)                    ; 播放编号 M0 的 Melody 文件。
TR2: PlayM($M0)*3                  ; 播放编号 M0 的 Melody 文件三次。
TR3: PlayM($M0)&[BG1, BG2]         ; 播放编号 M0 文件，并且同时调用背景 1 与背景 2 的
                                     动作。

[Background1]
BG1: ……

[Background2]
BG2: ……
```

5.15.2 WaitMN

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

若有 Melody 正在播放中，则 Melody 播放完毕后，再执行下一个指令。没有 Melody 播放则立即执行下一个指令。

注意：PlayM=PlayMS+WaitMN。

例.

```
[Path]
TR1: PlayM($M0)
TR2: PlayMS($M0), WaitMN           ; 与 TR1 的执行结果相同。
```

5.15.3 PauseM

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

PauseM 暂停当前 MIDI 音乐文件的播放。

例.

```
PowerOn: PlayM($M0)
TR1: PauseM                        ; 暂停 Melody 播放。
```

5.15.4 ResumeM

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

ResumeM 指令可用来恢复 MIDI 音乐文件的播放，以下情况指令不再有效：

1. Melody 文件已经播放结束。
2. 有执行过 StopM 或 Stop 指令。

例.

```
PowerOn: PlayM($M0)
TR1: PauseM                        ; 暂停 Melody 播放。
```

```
TR2: ResumeM                    ; 恢复 Melody 播放。
```

5.15.5 StopM

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

StopM 指令是立即停止当前播放的 Melody 文件。

例.

```
PowerOn: PlayM($M0)
TR1: StopM                      ; 停止 Melody 播放。
```

5.15.6 Instrument(Ch, i)

[NY5+ / NY6 / NY7 / NX1]

Instrument 指令是用来改变正在播放 melody 的音色。

Ch：为要改变音色的 MIDI 通道，

- NY5+支援 1 ~ 4、10。
- NY6 支持 1 ~ 6、10。
- NY7 支持 1 ~16。
- NX1 支持 1 ~16。

i：为 GM 音色编号，可用立即值或用变量控制，范围为 0~127。

注意：

1. 此指令必须在 melody 播放时调用才有效，在 melody 播放前调用无效。

2. 使用变量指定 GM 音色时，如果该音色不存在，有可能会造成音色切换错误或播放异常的现象。

```
例. Instrument(Ch2, 1)          ; 将 MIDI 通道 2 的音色强制改为 1 号音色。
例. Instrument(Ch1, R1:R0)      ; 将 MIDI 通道 1 的音色强制改为 R1:R0 的 GM 音色编
                                  号，R1 为高 3 位，R0 为低 4 位。
例. Instrument(Ch3, X0)         ; 将 MIDI 通道 3 的音色强制改为 X0 的 GM 音色编号。
```

5.15.7 M_Chx_Vol = n / M_Chx_Vol = Ri

[NY5+ / NY6 / NY7 / NX1]

修改正在播放 Melody 的个别通道音量。设定的音量可为立即值或用 Ri 来控制。

Chx：指定 MIDI 通道。

- NY5+支援 1 ~ 4、10。
- NY6 支持 1 ~ 6、10。
- NY7 支持 1 ~ 16。
- NX1 支持 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：当 melody 开始播放时，会将每个 MIDI 通道的音量默认为 15，因此在播放前使用此指令无效。

例. **M_Ch1_Vol = 10** ；将 MIDI 通道 1 的音量设为 10，约原本音量的 67%。

例. **M_Ch1_Vol = R0** ；取 R0 内的数值当成 MIDI 通道 1 的音量。

5.15.8 Ri = M_Chx_Vol

[NY5+ / NY6 / NY7 / NX1]

读取正在播放 Melody 的个别通道音量。

Chx：指定 MIDI 通道，

- NY5+支援 1 ~ 4、10。
- NY6 支持 1 ~ 6、10。
- NY7 支持 1 ~ 16。
- NX1 支持 1 ~ 16。

注意：当 melody 开始播放时，会将每个通道的音量默认为 15。

例. **R0 = M_Ch1_Vol** ；读取 MIDI 通道 1 的音量，并存于 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 为增加的百分比，以 5%为单位，当增加比例不为 5 的整数倍时会取四舍五入。

注意：此指令必须在 melody 播放时调用才有效，在 melody 播放前调用无效。

例. **Tempo=100**

[Path]

PowerOn: PlayMS($M0), Tempo + 10 ；拍子速度增加 10%，变成 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 为降低的百分比，以 5%为单位，当降低比例不为 5 的整数倍时会取四舍五入。

注意：此指令必须在 melody 播放时调用才有效，在 melody 播放前调用无效。

例. **Tempo=100**

[Path]

```
PowerOn: PlayMS($M0), Tempo - 5          ; 拍子速度降低 5%，变成 95%。
```

5.15.11 Tempo++

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

Tempo++ 指令。依据 Melody 原始的 Tempo 值加快一阶。

注意：

1. *NY5 tempo 增量请参考 Tempo=n 指令说明。*
2. *NY5+ / NY6 / NY7 / NX1 每一阶加快播放速度5%，此指令必须在melody 播放时调用才有效，在melody 播放前调用无效。*

例.

```
PowerOn: PlayMS($M0)
TR1: Tempo++                             ; 拍子速度变成 105%。
```

5.15.12 Tempo--

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

Tempo-- 指令。依据 Melody 原始的 Tempo 值减慢一阶。

注意：

1. *NY5 tempo 增量请参考 Tempo=n 指令说明。*
2. *NY5+ / NY6 / NY7 / NX1 每一阶加快播放速度5%，此指令必须在melody 播放时调用才有效，在melody 播放前调用无效。*

例.

```
PowerOn: PlayMS($M0)
TR1: Tempo--                             ; 拍子速度变成 95。
```

5.15.13 TempoRst

[NY5+ / NY6 / NY7 / NX1]

TempoRst 指令。可以恢复 Melody 文件原本的播放速度。

注意：此指令必须在 melody 播放时调用才有效，在 melody 播放前调用无效。

例.

```
Tempo=100
PowerOn: PlayMS($M0), Tempo - 5
TR1: TempoRst                            ; 拍子速度恢复成 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
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 **；拍子速度变成 100。**

5.15.15 Tempo(Rj:Ri) / Tempo(Xi)

[NY5]

根据目前 RAM 的值来设定 Tempo 值。

Tempo 值的计算公式如下：

Time base = 2.048ms

Value = 60000(ms) / Tempo / 24 / Timebase

例. **Tempo=100**

PowerOn: X0=12, Tempo(X0) **；设定拍子速度为 100。**

5.15.16 ReadTempo(Rj:Ri) / ReadTempo(Xi)

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

将目前的 Tempo 设定值，存放到 RAM。

例.

[Path]

TR1: PlayM($M0) **；播放 Melody。**

TR2: ReadTempo(X0), [PD,PC]=X0 **；将 Tempo 的设定值输出到 PD、PC。**

5.15.17 Mute_On(Ch)

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

打开静音功能，当 melody 播放时，Mute_on 指令可以把指定的 MIDI 通道作静音。

Ch：要静音的 MIDI 通道，不指定则为全部 MIDI 通道。

- NY5 支持 0 ~ 3。
- NY5+支援 1 ~ 4、10。
- NY6 支持 1 ~ 6、10。
- NY7 支持 1 ~ 16。
- NX1 支持 1 ~ 16。

注意：

1. NY5+ / NY6 / NY7 使用此指令必须在 melody 播放时调用才有效，在 melody 播放前调用无效。

2. NX1 于 Qcode6.50 起，Mute_On 指令将不受 melody 播放限制。

例.

[Path]

PowerOn : PlayM($M0)

TR1: Mute_on(1) **; Melody 播放时，打开 MIDI 通道 1 为静音功能。**

TR2: Mute_on(3) **; Melody 播放时，打开 MIDI 通道 3 为静音功能。**

5.15.18 Mute_Off(Ch)

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

关闭静音功能，当有静音模式被打开，可以利用 Mute_off 指令把指定的 MIDI 通道还原。

Ch：关闭静音功能的通道。

- NY5 支持 0 ~ 3。
- NY5+支援 1 ~ 4、10。
- NY6 支持 1 ~ 6、10。
- NY7 支持 1 ~ 16。
- NX1 支持 1 ~ 16。

注意：NY5+ / NY6 / NY7 使用此指令必须在 melody 播放时调用才有效，在 melody 播放前调用无效。

例.

[Path]

PowerOn : PlayM($M0)

TR1: Mute_on(1) **; Melody 播放时，打开 MIDI 通道 1 为静音功能。**

TR2: Mute_on(3) **; Melody 播放时，打开 MIDI 通道 3 为静音功能。**

TR3: Mute_off(1) **; 取消 MIDI 通道 1 的静音。**

TR4: Mute_off(3) **; 取消 MIDI 通道 3 的静音。**

5.15.19 OKON_On / SingleNote_On

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

One-Key-One-Note 功能打开，用户可以使用 OKON_On 指令打开此功能。

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，指令必须在 melody 播放时调用才有效，在 melody 播放前调用无效。*
3. *NX1 通道固定在 MIDI 通道 1。*

例.

```
[Path]
PowerOn:
TR1: PlayM($M0)                ; 播放音乐文件。
TR2: OKON_On                   ; 打开 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 功能关闭，用户可以使用 OKON_Off 指令关闭此功能。

注意：NY5+ / NY6 / NY7 此指令必须在 melody 播放时调用才有效，在 melody 播放前调用无效。

例.

```
[Path]
PowerOn:
TR1: PlayM($M0)                ; 播放音乐文件。
TR2: OKON_On                   ; 打开 One-Key-One-Note 功能，播放 MIDI 通道 1 的音符。
TR3: OKON_Off                  ; 关闭 One-Key-One-Note 功能。
```

5.15.21 OKON_Play / SinglePlay

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

用户可以使用 OKON_Play 指令开始进行 One-Key-One-Note 的播放动作。

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，指令必须在 melody 播放时调用才有效，在 melody 播放前调用无效。*
3. *NX1 通道固定在 MIDI 通道 1。*

例.

```
[Path]
PowerOn:
TR1: PlayM($M0)                ; 播放音乐文件。
TR2: OKON_On                   ; 打开 One-Key-One-Note 功能，播放通道 1 的音符。
TR3: OKON_Play                 ; 播放单音。
```

5.15.22 OKON_SustainOn

[NY5+]

开启 OKON 的延音功能。当开启时，OKON_Play 执行时，若 midi 音符有 sustain 段，则该音符持续输出不间断，直到下达 OKON_SustainEnd 指令为止。

例.

```
[Input State]
Input_0: TR1 TR2 TR3 TR4/TR4_R
[Path]
PowerOn: Input_0
TR1: PlayMS($M0), OKON_On        ; 播放音乐文件，并开启 OKON 功能。
TR2: OKON_SustainOn              ; 开启 OKON 的延音功能。
TR3: OKON_SustainOff             ; 关闭 OKON 的延音功能。
TR4: OKON_Play                   ; 按下按键，开始播放音符，当 OKON 延音开启时，持续播
                                 ; 放 sustain 段。
TR4_R: OKON_SustainEnd           ; 放开按键，结束音符的播放。
```

5.15.23 OKON_SustainOff

[NY5+]

关闭 OKON 的延音功能。当关闭时，OKON_Play 执行时，若 midi 音符有 sustain 段，则依照 Q-MIDI 中设定的时间长度播放。

5.15.24 OKON_SustainEnd

[NY5+]

结束 OKON 播放中音符的延音效果，若延音效果的时间小于 Q-MIDI 中设定的时间，则依据 Q-MIDI 中设定的时间长度播放，若延音效果的时间大于 Q-MIDI 中设定的时间，则音符直接收音结束。

5.15.25 DynamicOn

[NY5+ / NY6 / NY7]

开启通道动态分配功能，功能开启后，播放 melody 的 note 时，程序会依照当时 synthesizer 通道使用状况，分配一个适合的通道来播放，因此谱曲时，将 note 谱在通道 1 ~ 4、10 (NY5+) / 1 ~ 6、10 (NY6) / 1 ~ 16 (NY7)皆可正常播放。但是仍须注意同时间 note 数不能超过 synthesizer 通道数。程序默认为 DynamicOn。

注意：指令必须在 melody 播放时调用才有效，在 melody 播放前调用无效。

例.

```
PowerOn: PlayMS($M0)
TR1: DynamicOn                    ; 打开动态分配功能。
```

5.15.26 DynamicOff

[NY5+ / NY6 / NY7]

关闭通道动态分配功能，功能关闭后，播放 melody 的 note 时，程序会使用与 note 通道相同的 synthesizer 通道来播放，因此谱曲时，只能将 note 谱在通道 0 ~ 3、7 (NY5+) / 1 ~ 6、10 (NY6) / 1 ~ 8 (NY7)，且同一个通道同时间只能有一个 note，否则前面的 note 会被后面的 note 取代。另外，当 MIDI 通道正在播放语音或单音时，该通道所有的 note 会被忽略，直到语音或单音播放完毕才会继续播放。

注意：

1. ***指令必须在 melody 播放时調用才有效，在 melody 播放前調用无效。***
2. ***当动态分配关闭，Percussion 的 note 会被指定到 hardware 通道 3 (NY5+) / 5 (NY6) / 7 (NY7)播放。***

例.

```
PowerOn: PlayMS($M0)
TR1: DynamicOff                   ; 关闭动态分配功能。
```

5.15.27 StopMNote

[NY7]

停止当前正在播放的 MIDI 音符，但不会停止 MIDI 本身的播放，也就是说，当前音符被停止后，后续新的音符仍会正常播放。除此之外，指令也不会影响键盘单音音符的播放，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 暂停播放时，马上将音符结束，避免暂停时因为尾音过长造成效果不协调。

例.

```
PauseM, StopMNote                 ; 暂停 MIDI 播放，并停止当前正在播放的 MIDI 音符。
```

5.15.28 MIDI_Pitch(Semitone)

[NX1]

此指令用于升降 MIDI 的播放音调。

Semitone：设定范围由 Q-MIDI 限制，最大±12 个半音，默认值为 0。

例.

```
MIDI_Pitch(-3), PlayMS(Ch0, $M0)  ; 音调降低 3 个半音并播放 MIDI。
```

5.15.29 Mask_On(Ch)

[NX1]

遮蔽 MIDI 通道功能，当 melody 播放时，Mask_On 指令可以把指定的 MIDI 通道遮蔽，使该 MIDI 通道除能以降低 CPU 负载。

Ch：要遮蔽的 MIDI 通道，不指定则为全部 MIDI 通道。

- NX1 支援 1 ~ 16。

例.

[Path]

PowerOn : PlayM(Ch0, $M0)

TR1: Mask_on(1) ; 开启 MIDI 通道 1 遮蔽功能。

TR2: Mask_on(3) ; 开启 MIDI 通道 3 遮蔽功能。

5.15.30 Mask_Off(Ch)

[NX1]

取消遮蔽 MIDI 通道功能，当 MIDI 通道遮蔽被开启，可以利用 Mask_Off 指令把指定的 MIDI 通道还原。

Ch：取消遮蔽功能的通道。不指定则为全部 MIDI 通道。

- NX1 支援 1 ~ 16。

例.

[Path]

PowerOn : PlayM(Ch0, $M0)

TR1: Mask_on(1) ; 开启 MIDI 通道 1 遮蔽功能。

TR2: Mask_on(3) ; 开启 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 曲目清单的档案数量。

ReadFileCountM(Ri)

ReadFileCountM(Ri, Storage)

Ri：用于取得数量的变量，取得的数据为 0~65535。

Storage：指定储存空间，可使用 SPI0 / SPI1，不指定时则为 [**Melody Database**] 内的曲目数量。

例.

[Variable]

Var16: midi_max=0, midi_idx=0

[Path]

```
PowerOn: ReadFileCountM(midi_max)  ; 取得 midi 曲目数量。
TR1:     midi_idx++,                ; midi_idx+1。
         midi_idx>=midi_max?{midi_idx=0}, ; index 超过边界，重置为 0。
         PlayM(ch0, midi_idx)       ; 播放 midi_idx 曲目。
```

5.15.32 MIDI_Loop_On

[NX1]

MIDI 循环播放。

注意：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               ; 开启循环播放。
TR3: MIDI_Loop_Off              ; 停止循环播放。
```

5.15.33 MIDI_Loop_Off

[NX1]

停止 MIDI 循环播放。停止循环播放后，仍会把音档完整播放完毕。

注意：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 键盘指令（Keyboard Command）

Instrument Command				
InstNoteOn	InstNoteOff	InstNoteAllOff	DrumNoteOn	DrumNoteOff
NoteVibrato	Gliss	MaxSingleNote	LongInst_HoldTime	ShortInst_HoldTime
KRecord	KRecordS	WaitKRN	StopKR	PlayK
PlayKS	WaitKN	StopK	-	-

5.16.1 InstNoteOn(Index, Note , Vol) & InstNoteOff(Note)

[NY5+ / NY6 / NY7 / NX1]

InstNoteOn 指令和 InstNoteOff 指令可用作一般乐器的单音键盘功能，但两个指令必须同时使用。当按下一个键时，InstNoteOn 指令会播放一个音符并占用一个通道，直到释放按键或播放完毕为止。当释放按键时，InstNoteOff 指令将停止播放并释放占用的通道。

Index：指定 GM 乐器音色编号。

Note：{“C1”, “C#1”, “D1”, “D#1”,”E1”, “F1”, “F#1”, “G1”, “G#1”, “A2”, “A#2”, “B2”…. “B8”}。

Vol：单音的 velocity，范围为 0~127，可用立即值或变数指定。

例.

```
Path1: InstNoteOn (0, C1, 127)    ; Falling Edge。播放 0 号一般音色，音高 C1，力度 127
Path2: InstNoteOff (C1)           ; Rising Edge。释放 C1 音高单音
```

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。释放 C3 音高单音
```

例.

```
Path1: InstNoteOn (0, C4, X1)     ; Falling Edge。播放 0 号一般音色，音高 C4，使用 X1 指定力度
Path2: InstNoteOff (C4)           ; Rising Edge。释放 C4 音高单音
```

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。释放 C4 音高单音
```

5.16.2 InstNoteAllOff

[NX1]

InstNoteAllOff 指令可用做一般乐器与打击乐器的键盘功能，InstNoteAllOff 将释放当前所有播放中的一般乐器与打击乐器的播放音符。

例.

```
Path1: InstNoteAllOff             ; 释放当前所有播音的音符
```

5.16.3 DrumNoteOn(Index, Vol) & DrumNoteOff(Index)

[NY5+ / NY6 / NY7]

DrumNoteOn 指令和 DrumNoteOff 指令可用作打击乐器的单音键盘功能，但两个指令必须同时使用。当按下一个键时，DrumNoteOn 指令会播放一个音符并占用一个通道，直到释放按键或播放完毕为止。当释放按键时，DrumNoteOff 指令将停止播放并释放占用的通道。

Index：指定 GM 打击乐器音色编号，可用立即值或变数指定。

Vol：单音的 velocity，范围为 0~127，可用立即值或变数指定。

例.

```
Path1: DrumNoteOn (0, 127)        ; Falling Edge。播放 0 号打击音色，力度 127。
Path2: DrumNoteOff (0)            ; Rising Edge。释放 0 号打击音色单音。
```

例.

```
Path1: DrumNoteOn (X0, 127)       ; Falling Edge。使用 X0 指定打击音色，力度 127。
Path2: DrumNoteOff (X0)           ; Rising Edge。释放 X0 指定打击音色单音。
```

例.

```
Path1: DrumNoteOn (35, X1)          ; Falling Edge。播放 35 号打击音色，使用 X1 指定力度。
Path2: DrumNoteOff (35)             ; Rising Edge。释放 35 号打击音色单音。
```

例.

```
Path1: DrumNoteOn (X0, X1)          ; Falling Edge。使用 X0 指定打击音色，使用 X1 指定力度。
Path2: DrumNoteOff (X0)             ; Rising Edge。释放 X0 指定打击音色单音。
```

5.16.4 NoteVibrato = n / NoteVibrato = Ri

[NY7 / NX1]

当音色通过 *Q-MIDI* 设定 vibrato 效果时，用户可利用此指令来打开单音的 vibrato 效果。”n”为 vibrato 控制参数，可为立即值或用 Ri 来控制，数值范围为 0~8，0 表示关闭，1~8 可控制效果的 depth，数值越大，表示 depth 越大。

注意：Percussion 音色不支持 vibrato 功能。

例.

```
Path1: NoteVibrato = 8, InstNoteOn (0, C1, 127)   ; 打开 vibrato 效果，播放 0 号一般音色，
                                                   ; 音高 C1，力度 127。
Path2: NoteVibrato = 0                            ; 关闭 vibrato 效果。
```

5.16.5 Gliss(Note, Semitone, Time)

[NY7 / NX1]

当单音播放时，用户可利用此指令达到 glissando 的效果。

Note：套用此效果的单音音高。

Semitone：指定变频的半音数，单次变频范围为+/-12 个半音。

Time：指定变频的时间。

- NY7 可支持的时间为 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 音色不支持 glissando 效果。***
2. ***此指令仅在符合 note 参数条件的单音播放时间内执行才有效；根据不同条件，变频时间约有+/-10%的误差。***
3. ***NY7 要使用 gliss，须以 Q-MIDI 制作音色时须加入含有 Wheel 句柄的.mid 文件，以定义变频的范围，否则 Gliss 指令将无效。***
4. ***针对 NY7，此指令的最大变频范围为原本的 sub-patch 范围加上 pitch bend 的变频范围。例如，原本 sub-patch 为 C3~D4，pitch bend 范围为+/-4 个半音，则指令有效变频范围则为 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 单音后，先用 300ms 降低两个半音为 D#4，再用 100ms 降低 4 个半音为 B3。

5.16.6 MaxSingleNote = n / MaxSingleNote = Ri

[NY5+ / NY6 / NY7]

此指令允许用户任意调整最大琴键音数目，以方便在电子琴应用时，根据不同功能，调整琴键音占用的通道数，确保有足够的通道提供给 MIDI 播放使用。

例.

Path1: MaxSingleNote = 3 **；将最大琴键音数目设定为 3 个。**

Path2: R0=5, MaxSingleNote = R0 **；通过 R0 将最大琴键音数目设定为 5 个。**

注意：此指令的设定值必须小于或等于 Melody_MaxSingleNote 这个 option 的设定值，否则无效。

5.16.7 LongInst_HoldTime(Time)

[NX1]

此指令用于设定有含 Sustain 的长音乐器 NoteOn 后，音符最长播放时间。

Time：指定长音乐器最长播放时间，可支持时间为 1ms ~ 131068ms，设定为 0 时为无限长度播音，默认值为 0。

例.

Path1: LongInst_HoldTime(16000) **；设定长音乐器最长播放 16 秒后进行收音。**

Path2: LongInst_HoldTime(0) **；设定长音乐器无限长度播放。**

5.16.8 ShortInst_HoldTime(Time)

[NX1]

此指令用于设定没有 Sustain 的短音乐器 NoteOff 后，要延迟多少时间才衔接 Envelope Release 段。

Time：指定短音乐器延迟时间衔接 Envelope Release，可支持时间为 0ms~1020ms，默认值为 0。

例.

Path1: ShortInst_HoldTime(300)

；设定短音乐器 NoteOff 后延迟 300ms 衔接 Envelope Release 段。

5.16.9 KRecord / KRecordS

[NX1]

进行琴键录音，用于记录 InstNoteOn / InstNoteOff 的 Note 与 Vol。录音完成后，可以透过 PlayK 播放。

KRecord(Label)

KRecordS(Label)

KRecord：待录音结束后(包含 Timeout)，执行下一个指令。

KRecordS：开始录音后，立即执行下一个指令。

Label：**[Record]** 中定义的录音区段名称。

注意：

1. *当 Krecord / KRecordS 被执行时，必须要触发 InstNoteOn 或 InstNoteOff 一次，才会正式进入琴键录音模式，Timeout 计时功能始启动。*
2. *当 Krecord / KRecordS 被执行时，任何一次 InstNoteOn / InstNoteOff 都会重置 Timeout 记时。*
3. *KRecord / KRecordS 不支援 Realtime Erasing，需搭配 EraseR / EraseRS 擦除录音空间。*
4. *录音空间抵达上限、发生 Timeout、使用 PlayKS 时，将视为 KRecord 正常结束，KRecord 会继续执行下一个指令。*

例.

```
[Path]
TR1: ERASER($Rec0),KRecord($Rec0)          ; 擦除 Rec0 后使用 KRecord 进行琴键录音。
TR2: PlayK($Rec0)                          ; 使用 PlayK 回放琴键录音结果。
TR3: InstNoteOn (0, C4, 127)               ; Falling Edge。播放 0 号音色，音高 C4，力度 127
TR4: InstNoteOff (C4)                      ; Rising Edge。释放 C4 音高单音。
TR5: InstNoteOn (0, D4, 127)               ; Falling Edge。播放 0 号音色，音高 D4，力度 127
TR6: InstNoteOff (D4)                      ; Rising Edge。释放 D4 音高单音。
TR7: InstNoteOn (0, E4, 127)               ; Falling Edge。播放 0 号音色，音高 E4，力度 127
TR8: InstNoteOff (E4)                      ; Rising Edge。释放 E4 音高单音。
```

5.16.10 WaitKRN

[NX1]

若有琴键录音正在执行，则琴键录音完毕后，再执行下一个指令。没有琴键录音则立即执行下一个指令。

注意：KRecord=KRecordS+WaitKRN。

例.

```
[Path]
TR1: KRecord($Rec0)
TR2: KRecordS($Rec0), WaitKRN              ; 与 TR1 的执行结果相同。
```

5.16.11 StopKR

[NX1]

StopKR 指令是立即停止当前琴键录音。

例.

```
TR1: KRecordS($Rec0)
TR2: StopKR                                ; 停止琴键录音。
```

5.16.12 PlayK / PlayKS

[NX1]

进行琴键回放指令。录音完成后，可以透过 PlayK 播放。

PlayK(Label, Instrument)

PlayKS(Label, Instrument)

PlayK：琴键回放结束后，执行下一个指令。

PlayKS：开始琴键回放后，立即执行下一个指令。

Label：**[Record]** 中定义的录音区段名称。

Instrument：使用指定的乐器播放琴键录音，支持 0 ~ 127。

例.

[Path]

TR1: ERASER($Rec0),KRecord($Rec0) **；擦除 Rec0 后使用 KRecord 进行琴键录音。**

TR2: PlayK($Rec0, 0) **；使用 PlayK 回放琴键录音结果，使用乐器编号 0。**

5.16.13 WaitKN

[NX1]

若有琴键回放正在执行，则琴键回放完毕后，再执行下一个指令。没有琴键回放则立即执行下一个指令。

注意：PlayK=PlayKS+WaitKN。

例.

[Path]

TR1: PlayK($Rec0)

TR2: PlayKS($Rec0), WaitKN **；与 TR1 的执行结果相同。**

5.16.14 StopK

[NX1]

StopK 指令是立即停止当前琴键回放。

例.

TR1: PlayKS($Rec0)

TR2: StopK **；停止琴键回放。**

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 可将输出音量调至最大 (Vol=15)。

例. **PowerOn: Vol_Max** ；音量设定成最大。

5.17.2 Vol_Min

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

Vol_Min 可将输出音量调至最小。(Vol =0)

注意：NY5 / NX1 音量=0，并非为静音。

例. **PowerOn: Vol_Min** ；音量设定成最小。

5.17.3 Vol = n

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

用户可以藉由 Vol command 来调整适合的音量。

n：欲设定的音量。

- NY5 / NY5+ / NY6 / NY7 / NX1，n = 0 ~ 15.

注意：NY5 / NX1 音量=0，并非为静音。

例. **Vol=7** ；将 **Volume** 设置成第八阶。

5.17.4 Vol = Ri

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

音量可以藉由 Ri 来控制。

注意：NY5 / NX1 音量=0，并非为静音。

例. **R1=3, Vol=R1** ；由 **R1** 来控制 **Volume**。

5.17.5 Vol++

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

Vol++为目前的音量设定递加一阶（最大为 15）。

例. Vol=8, Vol++ ; 则音量=9。

5.17.6 Vol--

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

Vol--为目前的音量设定递减一阶，Vol 最小为 0。

注意：NY5 / NX1 音量=0，并非为静音。

例. **Vol=8, Vol--** ; 则音量=7。

5.17.7 Ri = Vol

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

Ri=Vol 可以把目前的音量设定取出，存入 Ri。

例. **R1=Vol** ; 将目前 Volume 的阶数取出后放在 R1。

5.17.8 Px = Vol

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

可以把目前的音量输出到 Port。

例. **PB=Vol** ; 将目前 Volume 的阶数输出到 PB。

5.17.9 Vol += n

[NX1]

Vol += n 可将目前的音量增加 n 阶，若目前音量增加 n 超过 15，则设定为 15。

例. **Vol += 2**

5.17.10 Vol -= n

[NX1]

Vol -= n 可将目前的音量减少 n 阶，若目前音量增加 n 小于 0，则设定为 0。

注意：NX1 音量=0，并非为静音。

例. **Vol -= 2**

5.17.11 VolX1

[NY6]

音量的阶数为原本 Vol 的阶数。

例. **Vol=8, VolX1 ; 音量=8。**

5.17.12 VolX2

[NY6]

音量的阶数为原本 Vol 的阶数乘 2。

注意：使用 VolX2 可将音量增大，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 控制指定声音输出通道的音量。n 介于 0~15 之间。代表音量大小。

CHx 代表 channel，对应到 PlayV / PlayVS、SPIPlay / SPIPlayS、PlayM / PlayMS 等指令所使用的声音输出通道。

注意：音量=0，为静音。

例.

SPIPlay(Ch0,0),Ch0_vol=5 ; 设定 Ch0 音量为 5

5.17.14 PP_Gain = n

[NX1 OTP]

用户可以藉由 PP_Gain 指令来调整适合的 Push-Pull 增益，n=0~16。PP_Gain=0 表示静音。PP_Gain=16，输出音量最大。PP_Gain 预设值为 9。PP_Gain 设定值对应 NY7 Push-Pull Volume 设定如下：

NX1 PP_Gain	Push-Pull Volume
16	130%
15	122%
14	114%
13	107%
12	100%
11	94%
10	88%
9	83%
8	77%
7	71%

NX1 PP_Gain	Push-Pull Volume
6	66%
5	60%
4	55%
3	50%
2	45%
1	40%
0	静音

例. **PP_Gain=9** ; 将 PP 增益设置成第 9 阶。

5.17.15 PP_Gain = Ri

[NX1 OTP]

用户可以藉由 PP_Gain 指令来调整适合的 Push-Pull 增益，Ri=0~16。

例. **R1=3, PP_Gain=R1** ; 由 R1 来控制 PP 增益。

5.17.16 PP_Gain++

[NX1 OTP]

PP_Gain++为目前的 Push-Pull 增益设定递加一阶，PP_Gain 最大为 16。

例. **PP_Gain=8, PP_Gain++** ; 则 PP 增益=9。

5.17.17 PP_Gain--

[NX1 OTP]

PP_Gain--为目前的 Push-Pull 增益设定递减一阶，PP_Gain 最小为 0。

例. **PP_Gain=8, PP_Gain--** ; 则 PP 增益=7。

5.17.18 Ri = PP_Gain

[NX1 OTP]

Ri=PP_Gain 可以把目前的 Push-Pull 增益设定取出，存入 Ri。

例. **R1=PP_Gain** ; 将目前 PP 增益设定取出后放在 R1。

5.17.19 PGA_Gain = n

[NX1]

用户可以藉由 PGA_Gain 指令来调整适合的 PGA 增益，可以对麦克风收录的信号做放大。n=0~31，0

为最小增益（仍然有信号），31 为最大。默认值为 12。

PGA_Gain 的设定值对输入信号的增益倍率请参考以下列表。用户可以依应用来调整 PGA_Gain，以语音识别而言，倍率建议为 60~70 之间，放大倍率若过大会造成饱和波形失真不利于辨识的判断。

请留意 NX1_FDB Ver.A 2016/11/21 上的 Ropi (R10) 为 5.1KΩ，NX1_FDB Ver.B 2017/10/5 (非正式的蓝色板) 的 Ropi (R5)为 2KΩ，NX1_FDB Ver.B 2018/8/22 (正式的蓝色板) 的 Ropi 为 0Ω 且不可更换。

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** **；将 PGA Gain 设定为 10。**

5.17.20 PGA_Gain = Ri

[NX1]

藉由寄存器设定 PGA_Gain 值。

例. **R1=10, PGA_Gain=R1** ；将 PGA Gain 设定为 10。

5.17.21 Ri = PGA_Gain

[NX1]

将目前 PGA_Gain 值读到寄存器。

例. **R1 = PGA_Gain** ；将 PGA Gain 的值存到 R1。

5.17.22 Ri = MixCtrl

[NY5]

将目前的 Mix Ctrl 设定值，存放到 RAM。

例. **R0=MixCtrl** ；将目前的 MixCtrl 设定存到 R0。

5.17.23 Px = MixCtrl

[NY5]

将目前的 Mix Ctrl 设定值，输出到 Port。

例. **PB=MixCtrl** ；将目前的 MixCtrl 设定输出到 PB。

5.17.24 MixCtrl

[NY5 / NX1]

控制各频道的混音输出。

NY5 系列可以使用 CH0，CH1，CH2，CH3。可用设定如下：

MixCtrl Command	说明	设定值
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，关闭其余 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，关闭其余 channel。	0xE
MixCtrl(0, 0, 2, 2)	CH2 / CH3 各一半音量。	0xC

NX1 控制各频道的混音输出，须填入百分比，共有 0%，25%，50%，75%，100%几种选项。

例. **MixCtrl(50%,25%,75%,100%)** **；将目前的 MixCtrl 设定输出到 PB。**

注意：

1. ***NY5 中，若用户未写 MixCtrl 时，系统默认值为 MixCtrl(1, 1, 1, 1)。***
2. ***NX1 中，若用户未写 MixCtrl 时，系统默认值为 MixCtrl(100%, 100%, 100%, 100%)。***

5.18 触摸键指令（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]

打开触摸键扫描功能。若是打开该功能，则触摸键会持续做扫描。

注意：

1. ***系统默认值为 TouchKey_ON，PowerOn 后无须另外再下达一次 TouchKey_ON 指令。***
2. ***PowerOn 后，触摸键扫描速度默认为 TouchKey_Scan_Normal。***
3. ***TouchKey_ON 后，会保持上一次设定的扫描状态。***

5.18.2 TouchKey_OFF

[NY9T]

关闭触摸键扫描功能。若是关闭该功能，则触摸键不做扫描。

例.

[Input State]

KEY1:

[Path]

PowerOn: TouchKey_OFF, Delay(0.3), KEY1, TouchKey_ON

; PowerOn 时，将触摸键扫描功能关闭，待延迟 0.3 秒后，才打开触摸键扫描。

5.18.3 TouchKey_CLR

[NY9T]

用于清除当前触摸键状态，当按键状态被清除后，触摸键会重新扫描。

例.

[Input State]

KEY1: TR1

[Path]

PowerOn: KEY1 ; TouchKey State 设置成 KEY1 状态。

TR1: PlayA(Ch1,$VIO0), TouchKey_CLR

; 按下 TR1 后，执行 PlayA(Ch1,$VIO0)。执行完 PlayA(Ch1,$VIO0)后，执行 TouchKey_CLR 指令，即清除当前的触摸键的状态，重新扫描触摸键状态。当 VIO0 播放完了之后，如果 TR1 有被按住时，程序会继续执行 PlayA(Ch1,$VIO0)，如果 TR1 没有被按住时，则程序进入 Sleep 状态。

5.18.4 TouchKey_Scan_Slow

[NY9T]

打开触摸键慢速扫描功能，则触摸键会间歇性对按键做扫描，可以达到较省电的效能。

注意：

1. 进入睡眠前，建议切换成 TouchKey_Scan_Slow 以达到省电。

2. 当触摸键被侦测到时，会自动切换至一般扫描模式。

3. 若按键有被按住时，则不会进入慢速扫描模式。

例.

[Path]

PowerOn: TouchKey_ON

Sleep: TouchKey_Scan_Slow ; 将触摸键扫描切换成慢速扫描模式。

5.18.5 TouchKey_Scan_Normal

[NY9T]

打开触摸键快速扫描功能，则触摸键会回复为一般扫描方式。（默认值为 TouchKey_Scan_Normal）

注意：当 TouchKey_Scan_Normal 时，触摸键的反应速度会较 TouchKey_Scan_Slow 快。

例.

[Path]

PowerOn: TouchKey_ON

Sleep: TouchKey_Scan_Slow ; 将触摸键扫描切换成慢速扫描模式。

TR1: TouchKey_Scan_Normal ; 将触摸键扫描切换成一般扫描模式。

5.18.6 TouchKey_Sensitivity(Level)

[NY9T / NX1]

切换触摸键的灵敏度，数值越小越灵敏，NY9T 共有八段可调整，默认值为 4。NX1 共有四段可调整，

默认值为 0。

Level：设定触摸键的灵敏度。可直接指定灵敏度的阶数或是由寄存器或外部电阻来调整，范围 0 ~ 7。Radj 阶数表请参考 ReadRadj(Ri)章节中说明。NX1 仅提供直接设定，范围 0 ~ 3。

- 直接设定灵敏度阶数（0~7） *例.* **TouchKey_Sensitivity(4)**
- Ri（支持 Ri） *例.* **TouchKey_Sensitivity(R0)**
- Radj（支持外部电阻调整灵敏度） *例.* **TouchKey_Sensitivity(Radj)**

例. 如何利用 bonding option，选择不同的触摸键灵敏度。

[Path]

```
PowerOn: Switch(PE[x x d d])= [Sens0, Sens1, Sens2, Sens3]    ; PE[1:0]为 bonding option。
Sens0: TouchKey_Sensitivity(0)                    ; PE[1:0]为 00 时，则灵敏度最高。
Sens1: TouchKey_Sensitivity(2)                    ; PE[1:0]为 01 时，则灵敏度介于最高和一般之间。
Sens2: TouchKey_Sensitivity(4)                    ; PE[1:0]为 10 时，则灵敏度一般。
Sens3: TouchKey_Sensitivity(7)                    ; PE[1:0]为 11 时，则灵敏度最低。
```

例. 如何利用寄存器因应不同模式下的行为，动态调整触摸键灵敏度。

[Path]

```
PowerOn:
Main: R0=0, R_Mode_Count>=8?SetSens, R0=4, SetSens
SetSens: TouchKey_Sensitivity(R0)                 ; 则灵敏度依据 R0 的数值设定。
```

例. 如何利用外部电阻上电时来选择不同的触摸键灵敏度。

[Path]

```
PowerOn: TouchKey_Sensitivity(Radj)          ; 根据 IC 侦测到外部电阻的阶数，自动调整灵敏度。
```

注意：当外部电阻未连接或是电阻值超过范围时，则该指令将会维持原本的灵敏度。

5.18.7 Calibrate_ON

[NY9T]

打开触摸键的灵敏度校正功能（默认值为打开）。

注意：NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支持此指令。

5.18.8 Calibrate_OFF

[NY9T]

关闭触摸键的灵敏度校正功能。

5.18.9 AutoJudge_Calibrate

[NY9T]

自动触摸键的灵敏度校正功能。打开后会自动更正环境变量，但若有触摸键被按住，则会终止校正程序。

例. 每 4 秒进行触摸键校正一次

[Path]

```
4Sec: AutoJudge_Calibrate                ; 每隔 4 秒钟，则触摸键会自动更正一次，但触摸键校正功能若被关闭的话，将不会进行校正。
TR1R: Calibrate_ON                       ; 打开触摸键校正功能。
TR2R: Calibrate_OFF                      ; 关闭触摸键校正功能。
```

5.18.10 Enforce_Calibrate_Normal

[NY9T / NX1]

强制触摸键的灵敏度校正功能。打开后会自动更正环境变量，若当前有触摸键被按住，仍会强迫进行校正。

注意：

1. ***当 Calibrate_OFF 时，则执行到 Enforce_Calibrate 指令时，触摸键将不会进行校正，并且会执行接续 Enforce_Calibrate_Normal 后面的指令。***
2. ***强烈建议每间隔一段时间就进行触摸键校正，以避免因环境改变而造成触摸键可能会反应不佳的状况。***
3. ***若是按键被长按超过 40 秒，则建议强制校正一次。***
4. ***Enforce_Calibrate_Normal 执行完毕后，若有程序中含有"Enforce_Calibrate"路径时，则会先执行该路径。***
5. ***Enforce_Calibrate_Normal 执行完毕后，是否会执行按键放开路径将与用户是否有清除触摸键状态有关。若不清除触摸键状态，则会自动执行按键放开的路径。***

例. 按键长按 40 秒，则强迫触摸键校正一次

[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 秒钟，则触摸键会自动更正一次。且当按键持续被按下 40 秒后，会强迫进行校正一次。

TR1R: Calibrate_ON                       ; 打开触摸键校正功能。
TR2R: Calibrate_OFF                      ; 关闭触摸键校正功能。
TR3R: KeyPress=1, Count=0, Play…..       ; 当 KEY3 被按下后，则 KeyPress=1（表示按键被按下）。
TR3F: KeyPress=0, Count=0                ; 当 KEY3 被放开后，则 KeyPress=0（表示按键被放开）。

Enforce_Calibrate: SW_Reset
; 执行完 Enforce_Calibrate 后，若不需记忆原本的状态，可直接进行软件重置，来恢复成开机时的状
```

态。若没有进行软件重置的话，将会执行按键放开的动作。

5.18.11 Enforce_Calibrate_Sleep

[NY9T]

强制触摸键的灵敏度校正功能。打开后会自动更正环境变量，若当前有触摸键被按住，仍会强迫进行校正。

注意:

1. 当 Calibrate_OFF 时，则执行到 Enforce_Calibrate_Sleep 指令时，触摸键将不会进行校正，并且会执行接续 Enforce_Calibrate_Sleep 后面的指令。

2. 强烈建议每间隔一段时间就进行触摸键校正，以避免因环境改变而造成触摸键可能会反应不佳的状况。

3. Enforce_Calibrate_Sleep 执行完毕后，即使程序中含有"Enforce_Calibrate"路径亦不会执行该路径。

4. Enforce_Calibrate_Sleep 执行后，不需等待灵敏度校正完成，即可进入 Sleep。

例. 每 0.5 秒，则强迫触摸键校正一次

[Path]

```
500ms: Enforce_Calibrate_Sleep                    ; 每隔 500 毫秒，则触摸键会自动更正一次。
```

5.18.12 Ri = TouchKey(Px)

[NY9T]

将触摸键状态存到 Ri，以 Port 为单位，每次读取指定 Port 的 4 支脚位状态。NY9T001A/004A series：x = PA，NY9T008A series：x = PA / PB，NY9T016A series：x = PA~PD。

```
例.R0=TouchKey(PA)                                ; 将触摸键 PA.0~PA.3 当前的状态存到 R0。
```

5.18.13 Var = Touchkey_Count(TouchKey)

[NX1]

此指令将指定的 TouchKey 扫描 Count 数值回传至变数，用于调整 TouchKey 设定。

TouchKey：欲回传 Count 数值的 TouchKey，依照使用数量输入范围为 0 ~ (n - 1)。

```
例. Var32_Count=TouchKey_Count(0)
; 将 TouchKey0 的 Count 数值传至变数 Var32_Count。
```

5.18.14 Var = Touchkey_BGCount(TouchKey)

[NX1]

此指令将指定的 TouchKey 背景 Count 数值回传至变数，用于调整 TouchKey 设定。

TouchKey：欲回传 Count 数值的 TouchKey，依照使用数量输入范围为 0 ~ (n - 1)。

```
例. Var32_BGCount=TouchKey_BGCount(0)
; 将 TouchKey0 的 Count 数值传至变数 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) 指令为间接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 Low-Nibble。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

Rx/Xx：Rx/Xx 内容值代表 X 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Ry/Xy：Ry/Xy 内容值代表 Y 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Ri：将 Table 取到的数据放入 Ri 内。

5.19.2 TableM(TableName, Rx/Xx, Ry/Xy, Ri)

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

TableM(TableName, Rx/Xx, Ry/Xy, Ri) 指令为间接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 Middle-Nibble。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

Rx/Xx：Rx/Xx 内容值代表 X 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Ry/Xy：Ry/Xy 内容值代表 Y 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Ri：将 Table 取到的数据放入 Ri 内。

5.19.3 TableH(TableName, Rx/Xx, Ry/Xy, Ri)

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

TableH(TableName, Rx/Xx, Ry/Xy, Ri) 指令为间接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 High-Nibble。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

Rx/Xx：Rx/Xx 内容值代表 X 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Ry/Xy：Ry/Xy 内容值代表 Y 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Ri：将 Table 取到的数据放入 Ri 内。

5.19.4 Table(TableName, Rx/Xx, Ry/Xy, Rh, Rm, Rl)

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

间接寻址指令，可用来读取在 **[Table]** 段落中定义数值。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

Rx/Xx：Rx/Xx 内容值代表 X 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Ry/Xy：Ry/Xy 内容值代表 Y 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Rh：将 Table 取到的 High-Nibble 数据放入 Rh 内。

Rm：将 Table 取到的 Middle-Nibble 数据放入 Rm 内。

Rl：将 Table 取到的 Low-Nibble 数据放入 Rl 内。

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) 指令为直接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 Low-Nibble。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

X：代表 X 轴的地址。

Y：代表 Y 轴的地址。

Ri：将 Table 取到的数据放入 Ri 内。

5.19.6 TableM(TableName, X, Y, Ri)

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

TableM(TableName, X, Y, Ri) 指令为直接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 Middle-Nibble。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

X：代表 X 轴的地址。

Y：代表 Y 轴的地址。

Ri：将 Table 取到的数据放入 Ri 内。

5.19.7 TableH(TableName, X, Y, Ri)

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

TableH(TableName, X, Y, Ri) 指令为直接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 High-Nibble。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

X：代表 X 轴的地址。

Y：代表 Y 轴的地址。

Ri：将 Table 取到的数据放入 Ri 内。

5.19.8 Table(TableName, X, Y, Rh, Rm, Rl)

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

直接寻址指令，可用来读取在 **[Table]** 段落中定义数值。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

X：代表 X 轴的地址。

Y：代表 Y 轴的地址。

Rh：将 Table 取到的 High-Nibble 数据放入 Rh 内。

Rm：将 Table 取到的 Middle-Nibble 数据放入 Rm 内。

Rl：将 Table 取到的 Low-Nibble 数据放入 Rl 内。

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) 指令为间接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 Low-Byte。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

Rx/Xx：Rx/Xx 内容值代表 X 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。。

Ry/Xy：Ry/Xy 内容值代表 Y 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Xi：Table 取到的数据放入 Xi 内。

5.19.10 TableH(TableName, Rx/Xx, Ry/Xy, Xi)

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

TableH(TableName, Rx/Xx, Ry/Xy, Xi) 指令为间接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 High-Byte。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

Rx/Xx：Rx/Xx 内容值代表 X 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Ry/Xy：Ry/Xy 内容值代表 Y 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Xi：将 Table 取到的数据放入 Xi 内。

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) 指令为间接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 High-Byte 和 Low-Byte。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

Rx/Xx：Rx/Xx 内容值代表 X 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Ry/Xy：Ry/Xy 内容值代表 Y 轴的地址，可选择用 4-bit 或 8-bit 变量寻址。

Xh：将 Table 取到的高位值放入 Xh 内。

Xl：将 Table 取到的低位值放入 Xl 内。

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) 指令为直接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 Low-Byte。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

X：代表 X 轴的地址。

Y：代表 Y 轴的地址。

Xi：将 Table 取到的数据放入 Xi 内。

5.19.13 TableH(TableName, X, Y, Xi)

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

TableH(TableName, X, Y, Xi) 指令为直接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 High-Byte。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

X：代表 X 轴的地址。

Y：代表 Y 轴的地址。

Xi：将 Table 取到的数据放入 Xi 内。

5.19.14 Table(TableName, X, Y, Xh, Xl)

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

Table(TableName, X, Y, Xh, Xl) 指令为直接寻址指令，可用来读取在 **[Table]** 段落中定义数值的 High-Byte 和 Low-Byte。

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

X：代表 X 轴的地址。

Y：代表 Y 轴的地址。

Xh：将 Table 取到的高位值放入 Xh 内。

Xl：将 Table 取到的低位值放入 Xl 内。

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]

间接寻址指令，可用来读取在 **[Table]** 段落中定义数值。

Table(TableName, VarX, VarY, VarR)

VarR = TableName[VarX][VarY]

TableName：是一个在 **[Table]** 段落中以字符定义的 table 名称。

VarX：X 轴的地址。

- 立即值。
- **[Variable]** 中定义的变量。

VarY：Y 轴的地址。

- 立即值。
- **[Variable]** 中定义的变量。

VarR：将 Table 取到的数据放入 VarR 内。

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 红外线指令（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_TX_Busy?Path	IR_Carrier_On	IR_Carrier_Off

5.20.1 IR_TX=data

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

直接 IR 代码发射。

```
例. IR_TX=0xF or IR_TX=15                 ; IR 发射代码 0x0F。
例. IR_TX=0xFFF or IR_TX=4095             ; IR 发射代码 0xFFF。
例. IR_TX=0xFFFF or IR_TX=65535           ; IR 发射代码 0xFFFF。
```

5.20.2 IR_TX(Rl:Rk:Rj:Ri)

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

提供使用 RAM 的方式来发射代码。Ri=bit0~3，Rj=bit4~7，Rk=bit8~11，Rl=bit12~15。

例. 4-bit mode 时，R0=0xF。

```
TR1: IR_TX(R0)                            ; 红外线发射代码 0xF。
```

例. 8-bit mode 时，R0=0xF，R1=0x3。

```
TR1: IR_TX(R1:R0)                         ; 红外线发射代码 0x3F。
```

例. 12-bit mode 时，R0=0xF，R1=0xF，R2=0x1。

```
TR1: IR_TX(R2:R1:R0)                      ; 红外线发射代码 0x13F。
```

例. 16-bit mode 时，R0=0xF，R1=0x3，R2=0x01，R3=0x0。

```
TR1: IR_TX(R3:R2:R1:R0)                   ; 红外线发射代码 0x013F。
```

5.20.3 IR_TX(Xj:Xi)

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

提供使用 RAM 的方式来发射代码。Xj：为 High-Byte；Xi：为 Low-Byte。

例. 4-bit mode 时，X0=0x3F。

```
TR1: IR_TX(X0)                            ; 红外线发射代码 0x0F。
```

例. 8-bit mode 时，X0=0x3F。

TR1: IR_TX(X0) ; 红外线发射代码 **0x3F**。

例. 12-bit mode 时，X0=0x3F，X1=0x01。

TR1: IR_TX(X1, X0) ; 红外线发射代码 **0x13F**。

例. 16-bit mode 时，X0=0x3F，X1=0x01。

TR1: IR_TX(X1:X0) ; 红外线发射代码 **0x013F**。

5.20.4 IR_TX_WaitN

[NY5+ / NX1]

若有 IR 数据正在传送中，则等待至数据传送完毕后，再执行下一个指令。IR 数据没有传送则立即执行下一个指令。

5.20.5 IR_RX_ON

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

打开红外线接收功能。若是没有打开该功能，则红外线无法接收。

例. **IR_RX_ON** ; 打开红外线接收功能。

5.20.6 IR_RX_OFF

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

关闭红外线接收功能。

例. **IR_RX_OFF** ; 关闭红外线接收功能。

5.20.7 [Rl, Rk, Rj, Ri] = IR_RX

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

用户可通过指令将 RX 收到的数据存在指定的 RAM，Ri=bit0~3，Rj=bit4~7，Rk=bit8~11，Rl=bit12~15。

注意：在未收到 RX 数据前，使用此指令会读取到未知数据。

例. 假设 IR 设定为 12-bit 模式

[R0] = IR_RX ; 将 **RX** 的 **Bit[3:0]**存到 **R0**。

[R1, R0] = IR_RX ; 将 **RX** 的 **Bit[3:0]**存到 **R0**，**Bit[7:4]**存到 **R1**。

[R2, R1, R0] = IR_RX ; 将 **RX** 的数据由低到高依序存入 **R0**、**R1** 及 **R2**。

5.20.8 [Xj, Xi] = IR_RX

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

用户可通过指令将 RX 收到的数据存在指定的 RAM，Xi=bit0~7，Xj=bit8~15。

注意：在未收到 RX 数据前，使用此指令会读取到未知数据。

例. 假设 IR 设定为 12-bit 模式

```
[X0] = IR_RX                    ; 将 RX 的 Bit[7:0]存到 X0。
[X1, X0] = IR_RX                ; 将 RX 的 Bit[7:0]存到 X0，Bit[11:8]存到 X1，X1[7:4]
                                  会清为 0。
```

5.20.9 IR_RX = data?Path

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

当接收到的 IR 代码等于 data 值时，跳到 Path。

```
例. IR_RX = 0? Path             ; 当接收到的代码为 0 时，跳到 Path。
```

5.20.10 IR_RX != data?Path

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

当接收到的 IR 代码不等于 data 值时，跳到 Path。

```
例. IR_RX != 0? Path            ; 当接收到的代码不为 0 时，跳到 Path。
```

5.20.11 IR_TX_Busy?Path

[NX1]

当 IR 正在传送数据时，跳到 Path。

5.20.12 IR_Carrier_On

[NX1]

启动 IR 载波输出，提供手动操作的控制指令。

5.20.13 IR_Carrier_Off

[NX1]

关闭 IR 载波输出，提供手动操作的控制指令。

5.21 串行控制传送指令（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]

可连接一般微控制器，将 NY9T 当成微控制器的按键。提供 3 种不同的传输协议，分别是 SPI_Like、NY3 Serial_Trigger、IR_Trigger。仅支持 16-bit 模式。

SPI_Like：使用 2 根脚位，达成类似 SPI 传输协议。

NY3 Serial_Trigger：使用 2 个脚位来连接 NY3 系列，可将 NY9T 当成 NY3 的按键。详细的连接方式，请参阅 NY3C/3D 规格书。

IR_Trigger：使用 1 个脚位来仿真 IR 的接收信号，可直接使用目前的 NY4/5/7 系列的 IR 通信。

Mode：选择 Serial Control 信号的通信协议或是依照脚位来做自动选择通信协议的模式。

- 自动选择模式 — ***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：传输数据。

- data（支持 16-bit） — ***例.* SC_TX(Auto, 0xFFFF)**
- Ri（支持到 4 个 Ri） — ***例.* SC_TX(Auto, R3:R2:R1:R0)**
- Xi（支持到 2 个 Xi） — ***例.* SC_TX(Auto, X1:X0)**

注意：

1. ***使用该功能前，需先在 Option 中设定传输协议。***
2. ***左边的寄存器为最高位，信号由最高位开始传送，传输的数据固定为 16-bit。***

5.21.2 SC_TX(Mode, Rl:Rk:Rj:Ri)

[NY9T008A / NY9T016A]

提供使用 RAM 的方式来传输串行信号。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 脚位，自动侦测使用的传输协议，输出 [R3:R2:R1:R0] 寄存器内容值的信号。

例. SPI_Like 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(SPI_Like, R3:R2:R1:R0)

；由 Serial Control 脚位，使用 SPI_Like 传输协议，输出 [R3:R2:R1:R0] 寄存器内容值的信号。

例. NY3 Serial_Trigger 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(NY3, R3:R2:R1:R0)

；由 Serial Control 脚位，使用 NY3 Serial_Trigger 传输协议，输出 [R3:R2:R1:R0] 寄存器内容值的信号。若寄存器内容值为皆为 0 时，仅输出 Reset 信号。

例. IR_Trigger 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(IR_Trigger, R3:R2:R1:R0)

；由 Serial Control 脚位，使用 IR_Trigger 传输协议，输出 [R3:R2:R1:R0] 寄存器内容值的信号。

5.21.3 SC_TX(Mode, Xj:Xi)

[NY9T008A / NY9T016A]

提供使用 RAM 的方式来传输串行信号。Xi=bit0~7，Xj=bit8~15。

例. Auto_Detect 模式

X1=0x0, X0=0, SC_TX(Auto, X1:X0)

；由 Serial Control 脚位，自动侦测使用的传输协议，输出 [X1:X0] 寄存器内容值的信号。

例. SPI_Like 模式

X1=0x0, X0=0, SC_TX(SPI_Like, X1:X0)

；由 Serial Control 脚位，使用 SPI_Like 传输协议，输出 [X1:X0] 寄存器内容值的信号。

例. NY3 Serial_Trigger 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(NY3, X1:X0)

；由 Serial Control 脚位，使用 NY3 Serial_Trigger 传输协议，输出 [X1:X0] 寄存器内容值的信号。

若寄存器内容值为皆为 0 时，仅输出 Reset 信号。

例. IR_Trigger 模式

R3=0x0, R2=0, R1=0, R0=0, SC_TX(IR_Trigger, X1:X0)

；由 Serial Control 脚位，使用 IR_Trigger 传输协议，输出 [X1:X0] 寄存器内容值的信号。

5.22 串行数据接收指令（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		-	-	--

5.22.1 SC_RX_ON

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

打开串行数据接收功能。若是没有打开该功能，则红外线无法接收。

***例.* SC_RX_ON** **；打开串行数据接收功能。**

5.22.2 SC_RX_OFF

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

关闭串行数据接收功能。

***例.* SC_RX_OFF** **；关闭串行数据接收功能。**

5.22.3 [Rl, Rk, Rj, Ri] = SC_RX

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

用户可通过指令将接收到的串行数据存在指定的 RAM，Ri=bit0~3，Rj=bit4~7，Rk=bit8~11，Rl=bit12~15。

注意：在未收到串行数据前，使用此指令会读取到未知数据。

例.

```
[R0] = SC_RX                    ; 将接收到数据的 Bit[3:0]存到 R0。
[R1, R0] = SC_RX                ; 将接收到数据的 Bit[3:0]存到 R0，Bit[7:4]存到 R1。
[R2, R1, R0] = SC_RX            ; 将接收到的数据由低到高依序存入 R0、R1 及 R2。
```

5.22.4 [Xj, Xi] = SC_RX

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

用户可通过指令将接收到的串行数据存在指定的 RAM，Xi=bit0~7，Xj=bit8~15。

注意：在未收到串行数据前，使用此指令会读取到未知数据。

例.

```
[X0] = SC_RX                    ; 将接收到数据的 Bit[7:0]存到 X0。
[X1, X0] = SC_RX                ; 将接收到数据的 Bit[7:0]存到 X0，Bit[15:8]存到 X1。
```

5.22.5 SC_RX = data?Path

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

当接收到的串行数据等于 data 值时，跳到 Path。

```
例. SC_RX = 0? Path             ; 当接收到的数据为 0 时，跳到 Path。
```

5.22.6 SC_RX != data?Path

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

当接收到的串行数据不等于 data 值时，跳到 Path。

```
例. SC_RX != 0? Path            ; 当接收到的数据不为 0 时，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]

传输 I2C 信号。

I2C_TX(Data)
I2C_TX(Rj:Ri)
I2C_TX(Xi)

Data：欲传送的立即值。
Rj:Ri：使用 RAM 的方式來传输 I2C 信号。Ri=bit0 ~ 3，Rj=bit4 ~ 7。
Xi：使用 RAM 的方式來传输 I2C 信号。Xi=bit0 ~ 7。

5.23.2 I2C_RX

[NY5+ / NX1]

用户可透过指令将接收到的 I2C 数据存在指定的 RAM。

[Rj, Ri] = I2C_RX
[Xi] = I2C_RX
Var = I2C_RX

Rj, Ri：将接收的 I2C 数据存放至 RAM，Ri=bit0 ~ 3，Rj=bit4 ~ 7。
Xi：将接收的 I2C 数据存放至 RAM，Xi=bit0 ~ 7。
Var：将接收的 I2C 数据存放至 RAM。

5.23.3 I2C_RX=data?Path

[NY5+ / NX1]

当接收到的 I2C 数据等于 data 值时，跳到 Path。

5.23.4 I2C_RX!=data?Path

[NY5+ / NX1]

当接收到的 I2C 数据不等于 data 值时，跳到 Path。

5.23.5 I2C_Ack?Path

[NY5+ / NX1]

当主控时，数据传送完毕，判断接收端是否有响应 ACK 信号，若有则跳至 Path。

5.23.6 I2C_Start

[NY5+ / NX1]

当主控时，传送装置地址，并开始由被动装置读数据或写数据至被动装置。

I2C_Start

I2C_Start(Address, Mode)

I2C_Start(Address, Bits, Mode)

Address：装置位址。

- data（支援 0 ~ 255）
- Ri（支援 1 ~ 2 个 Ri）
- Xi（支援 1 个 Xi）

Bits：可为 7 / 10，预设为 7。

Mode：传输模式。可为 R / W。

注意：只有在 I2C 为 Master 模式才可使用此指令。

5.23.7 I2C_Stop

[NY5+ / NX1]

当主控时，传送结束信号，通知装置通信结束。

注意：只有在 I2C 为 Master 模式才可使用此指令。

5.23.8 I2C_MReadAck

[NY5+ / NX1]

主控输出 clock 以接收装置回传数据，接收完成响应 ACK 信号。

注意：只有在 I2C 为 Master 模式才可使用此指令。

5.23.9 I2C_MReadNAck

[NY5+ / NX1]

主控输出 clock 以接收装置回传数据，接收完成不回应 ACK 信号。

注意：只有在 I2C 为 Master 模式才可使用此指令。

5.23.10 I2C_Reset

[NX1]

重设 I2C。

注意：只有在 I2C 为 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]

传输 UART 信号。

UART_TX(Data)

UART_TX(Rj:Ri)

UART_TX(Xi)

Data：欲传送的立即值。

Rj:Ri：使用 RAM 的方式來传输 UART 信号。Ri=bit0 ~ 3，Rj=bit4 ~ 7。

Xi：使用 RAM 的方式來传输 UART 信号。Xi=bit0 ~ 7。

5.24.2 UART_TX_WaitN

[NY5+ / NX1]

若有 UART 数据正在传送中，则等待至数据传送完毕后，再执行下一个指令。UART 数据没有传送则立即执行下一个指令。

5.24.3 UART_RX

[NY5+ / NX1]

[Rj, Ri] = UART_RX

Xi = UART_RX

用户可透过指令将接收到的 UART 数据存在指定的 RAM，Ri=bit0 ~ 3，Rj=bit4 ~ 7，Xi=bit0 ~ 7。

5.24.4 UART_RX=data?Path

[NY5+ / NX1]

当接收到的 UART 数据等于 Data 值时，跳到 Path。

5.24.5 UART_RX!=data?Path

[NY5+ / NX1]

当接收到的 UART 数据不等于 Data 值时，跳到 Path。

5.24.6 UART_TX_Busy?Path

[NX1 EF]

当 UART TX 正在进行中，则跳到 Path。

5.25 脉冲调变 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	WaitPN	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 提供用户简易的指令来达成 PWM-IO 输出的方式。指令格式如下：

PWMOut：待 PWM-IO 输出的时间结束后，才执行下一个指令。

PWMOutS：PWM-IO 输出后，立即执行下一个指令。

Pin：可指定任一输出脚。

- **Px.n**：IC 脚位。
- **EXPx_Py.n**：I/O 扩展芯片的脚位。

Duty：可使用立即值指定输出百分比及阶数或使用变量指定阶数。

- 使用立即值输出阶数 ***例.*** **PWMOut(PC.0, 12, 1s)**
- 使用立即值输出百分比 ***例.*** **PWMOut(PC.0, 30%, 1s)**
- 使用 Ri 输出阶数（固定使用 2 个 Ri） ***例.*** **PWMOut(PC.0, R1:R0, 1s)**
- 使用 Xi 输出阶数（固定使用 1 个 Xi） ***例.*** **PWMOut(PC.0, X0, 1s)**

Time：可以设定 PWM-IO 输出的时间，不填写的话默认为 16ms，若指定 loop 则持续输出不停止。（时间=16ms~30sec）

Type：选择使用软件(0)或硬件(1) PWM 输出功能，默认为软件(0)。

注意：NY4 / NY5 / NY6 / NY7 / NX1 不支持 Type 参数。

5.25.1.2 PWMOut / PWMOutS (Para8, Para7, Para6, Para5, Para4, Para3, Para2, Para1)

[NY9T]

PWMOut 指令是用来设定全部 PWM-IO 脚位的 Duty cycle，不需要经过 PlayPWM 指令即可直接输出。

Para1~8：代表各个 pin 编号，Para1=PE.0，Para2=PE.1，依此类推。（只有设定成 PWM-IO 的脚位才能设定）

- 直接设定输出的百分比（0%~100%） *例.* **PWMOut (30%, 50%, 70%, 5%)**
- 直接设定输出的阶数（0~255） *例.* **PWMOut (30, 50, 70, 5)**

Time：可以设定 PWM-IO 输出的时间，不填写的话预设为 16ms。（时间=16ms~30sec）

注意：NY9T 使用两个 PWM-IO 输出指令，必须要间隔 16ms 以上才不会发生动作不正确的情况。

5.25.1.3 PWMOut / PWMOutS (@PWMEn, Percentage, Time)

[NY9T]

PWMEn：脚位及文件对应的方式是依照 **[I/O]** 段落中所设定的 PWM-IO 输出脚位顺序及 Action 文件中的 Label “A1~A8” 的顺序，最多对应至 PE.0~PF.3，共 8 个 pin。PWMEn=1 or 0，1=该脚位在播放 Action 文件时，打开 PWM-IO 输出功能；0=不启动。（未填的部分会自动补 0，高位在最左边）

- 直接设定要输出的脚位 *例.* **PWMOut(@1111, 30%)**
- Ri（支持到 2 个 Ri） *例.* **PWMOut(R1:R0, 30%)**
- Xi（支持到 1 个 Xi） *例.* **PWMOut(X0, 30%)**
- 填写 X 的话，会维持上一次的设定。 *例.* **PWMOut(X, 30%)**

Percentage：设定输出的百分比或阶数。

- 直接设定输出的百分比（0%~100%） *例.* **PWMOut(@1111, 30%)**
- 直接设定输出的阶数（0~255） *例.* **PWMOut(@1111, 127)**
- Ri（支持到 2 个 Ri） *例.* **PWMOut(@1111, Rj:Ri)**
- Xi（支持到 1 个 Xi） *例.* **PWMOut(@1111, Xi)**

Time：可以设定 PWM-IO 输出的时间，不填写的话预设为 16ms。（时间=16ms~30sec）

例.

[Path]

```
TR1: PWMOut(@1111, 30%)                    ; PE.0~PE.3 输出 30%的亮度。
TR2: PWMOut(@1111, 30)                     ; PE.0~PE.3 输出 30/256 的亮度。
TR3: R1=0x1, R0=0xE, PWMOut(@1111, R1:R0) ; PE.0~PE.3 输出 30/256 的亮度。
TR4: X0=0x1E, PWMOut(@1111,X0)             ; PE.0~PE.3 输出 30/256 的亮度。
TR5: PWMOut(@1111, 30%, 16ms)              ; PE.0~PE.3 输出 30%亮度及播放 16 毫秒的时间。
TR6: PWMOut(@1111, 30, 2s)                 ; PE.0~PE.3 输出 30/256 的亮度及 2 秒的时间。
```

注意：

1. ***NY9T 使用两个 PWM-IO 输出指令，必须要间隔 16ms 以上才不会发生动作不正确的情况。***
2. ***若 Percentage 参数使用寄存器来决定的话，将会占用 DATA ROM 来存放 PWM-IO 信号 (DATA Size 计算方式 = (pin number+1)*256)。***

5.25.2 PlayPWM / PlayPWMS

5.25.2.1 PlayPWM / PlayPWMS

[NY5+]

使用者可在 **[PWM-IO]** 段落中设定要使用 PWM-IO 输出的信号组合，透过此指令来将信号播放出来。

PlayPWM({Px.n, }Ch, $PIO) { & [BG1 {,BG2}] }

PlayPWM({Px.n, }Ch, $PIO, Loop) { & [BG1 {,BG2}] }

PlayPWMS({Px.n, }Ch, $PIO)

PlayPWMS({Px.n, }Ch, $PIO, Loop)

PlayPWM：待 Action 文件播放结束后，执行下一个指令。

PlayPWMS：Action 文件开始播放，立即执行下一个指令。

Px.n：输出脚位。

- 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]** 中指定的输出组态。

- 直接指定代号(SPIOx 及 MPIOx)。
- Ri（可支持到 1~3 个 Ri）或是 Xi（可支持到 1~2 个 Xi），由 Ri / Xi 的值决定要播放的是第几个 PWM-IO 文件。若是 Ri / Xi 带有"++"则在播放后将其递增。若使用 2 个 Xi，则高 4 位无效。

Loop：让此 Action 信号循环播放，直到有其他的指令来停止它，如 StopPWM(若不使用此功能则不需写出此参数)。

注意：

1. NY9T 中，两个 PWM-IO 输出指令，必须要间隔 16ms 以上才不会发生动作不正确的情况。

2. NY5+中，当指定 Px.n 时，仅能使用 [PWM-IO] 中的单一信号输出；未指定 Px.n 时，则只能使用[PWM-IO] 中的多信号输出。

例. NY5+

```
[Action File]
VIO0 = Action1.vio                          ; Action 檔中，含有 ActionLabel "A1~A6"的信号。
[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 输出 VIO0.A1，PB.1 输出 VIO0.A2，以此类推。
```

Path2: PD=[P P P P], PlayPWM(Ch2, $MPIO2) ; PD.0 输出 VIO0.A4，PD.1 输出 VIO0.A3，以此类推。

Path3: PlayPWM(PB.0, Ch1, $SPIO1) ; PB.0 输出 VIO0.A6。

5.25.2.2 PlayPWM(@PWMEn) / PlayPWMS(@PWMEn)

[NY9T]

用户可由 *Q-Visio* 来绘制要输出的 Action 信号，在 *Q-Code* 中可使用 PWM-IO 指令来将信号播放出来。

PlayPWM(@PWMEn, $VIOLabel) { & [BG1 {,BG2}] }

PlayPWM(@PWMEn, $VIOLabel, Extension) { & [BG1 {,BG2}] }

PlayPWMS(@PWMEn, $VIOLabel)

PlayPWMS(@PWMEn, $VIOLabel, Extension)

PlayPWM：待 Action 文件播放结束后，执行下一个指令。

PlayPWMS：Action 文件开始播放，立即执行下一个指令。

PWMEn：脚位及文件对应的方式是依照 **[I/O]** 段落中所设定的 PWM-IO 输出脚位顺序及 Action 文件中的 Label "A1~A8" 的顺序，最多对应至 PE.0~PF.3，共 8 个 pin。PWMEn=1 or 0，1=该脚位在播放 Action 文件时，打开 PWM-IO 输出功能；0=不启动。（未填的部分会自动补 0，高位在最左边）

- 直接设定要输出的脚位 *例.* **PlayPWM(@1111, $VIO0, 4)**
- Ri（支持到 2 个 Ri） *例.* **PlayPWM(R1:R0, $VIO0, 4)**
- Xi（支持到 1 个 Xi） *例.* **PlayPWM(X0, $VIO0, 4)**
- 填写 X 的话，会维持上一次的设定 *例.* **PlayPWM(X, $VIO0, 4)**

VIOLabel：加入多个 VIO 档时，须指定所要引用的 Action File 中的哪一组。

- 直接设定要输出的文件 *例.* **PlayPWM(@1111, $VIO0)**
- Ri（支持到 4 个 Ri） *例.* **PlayPWM(@1111, R3:R2:R1:R0)**
- Xi（支持到 2 个 Xi） *例.* **PlayPWM(@1111, X1:X0)**

Extension：可将该信号整个依照倍数来延长，节省 ROM Size。Extension=1/2/4/8，1 倍可省略不写。

- 直接赋值。 *例.* **PlayPWM(@1111, $VIO0, 4)**
- Ri（支持到 1 个 Ri） *例.* **PlayPWM(@1111, $VIO0, R0)**
- 填写 X 的话，会维持上一次的设定 *例.* **PlayPWM(@1111, $VIO0, X)**

注意：

1. ***两个 PWM-IO 输出指令，必须要间隔 16ms 以上才不会发生动作不正确的情况。***
2. ***PWMEn 参数的最左边对应有设定成 PWM-IO 输出脚位的最高位；Action 文件则是 A1 对应到最低位。***
3. ***Action 信号会依照 PWM-IO 打开的数量来对应，A1 会以 PE0 为优先对应脚位，若 PE0 未被设定成 PWM-IO，则对应至 PE1，依此类推。***
4. ***若是播放 VIO 文件，且同时输出多个信号时，须注意各信号的输出时间长度必须相等，若时间长***

度不相等时，则会以输出时间最长的信号为基准，并且将其余信号输出时间不足的部分强制输出 0%。

5. 注意：Ri 的内容值必须为 1/2/4/8，非这四个数值时，皆会变成 1 倍。

例.

```
[Action File]
VIO0 = Action1.vio                    ; Action 档中，含有 ActionLabel “A1~A6”的信号。
VIO1 = Action2.vio                    ; Action 档中，含有 ActionLabel “A1~A6”的信号。

[Path]
Path1: PlayPWM(@1100, $VIO0)          ; 播放“VIO0”，输出到 PE.2, PE.3。
Path2: PlayPWM(@0011, $VIO0)          ; 播放“VIO0”，输出到 PE.0, PE.1。
Path3: PlayPWM(@1111, $VIO1)          ; 播放“VIO1”，输出到 PE.0, PE.1, PE.2, PE.3。
Path4: PlayPWM(@1111, $VIO1, 4)       ; VIO1 信号延长 4 倍来播放，输出到 PE.0, PE.1, PE.2,
                                        PE.3。
Path5: PlayPWM(@1111, $VIO1), PlayA…  ; 播放“VIO1”完，立即执行 PlayA 指令。
Path6: PlayPWMS(@1111, $VIO1), PlayA…; 播放“VIO1”后，立即执行 PlayA 指令。
```

5.25.3 WaitPN

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

等待 PWM-IO 输出完毕后，再执行下一个指令，没有 PWM-IO 正在执行则立即执行下一个指令。

WaitPN

WaitPN(Px.n)

WaitPN(Ch)

WaitPN(PIO)

Pin：指示要等待 PWMOut / PWMOutS 输出的脚位。

- **Px.n**：IC 脚位。
- **EXPx_Py.n**：I/O 扩展芯片的脚位。

Ch：PlayPWM / PlayPWMS 所输出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的输出状态。

注意：

1. NY4 / NY5 / NY6 / NY7 / NY9T 不支持 Pin 参数。

2. NX1 不支持 Ch / PIO 参数。

5.25.4 StopPWM

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

用来停止 PWM-IO 输出。

StopPWM

StopPWM(Pin)

StopPWM(Ch)

StopPWM(PIO)

Pin：指示要停止 PWMOut / PWMOutS 输出的脚位。

- **Px.n**：IC 脚位。
- **EXPx_Py.n**：I/O 扩展芯片的脚位。

Ch：PlayPWM / PlayPWMS 所输出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的输出状态。

注意：

1. NY4 / NY5 / NY6 / NY7 / NY9T 不支持 Pin / Ch / PIO 参数。

2. NX1 不支持 Ch / PIO 参数。

5.25.5 PausePWM

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

暂停目前的 PWM-IO 输出。

- NY9T，PausePWM 对所有的 PWM-IO 都是暂停输出，IC 可进入睡眠。
- NY4 / NY5 / NY6 / NY7，不支持 PlayPWM，PausePWM 会暂停 PWM 计时，维持当下 duty 持续输出，IC 不进入睡眠。
- NY5+
  - PausePWM 对 PWMOut 的输出，会暂停 PWM 计时，维持当下 duty 持续输出，IC 不进入睡眠。
  - PausePWM 对 PlayPWM 的输出，暂停输出，IC 可进入睡眠。

PausePWM

PausePWM(Pin)

PausePWM(Ch)

PausePWM(PIO)

Pin：指示要暂停 PWMOut / PWMOutS 输出的脚位。

- **Px.n**：IC 脚位。
- **EXPx_Py.n**：I/O 扩展芯片的脚位。

Ch：PlayPWM / PlayPWMS 所输出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的输出状态。

注意：

1. 如果在没有 PWM-IO 输出的情况下，此动作会被忽略。

2. NY4 / NY5 / NY6 / NY7 / NY9T 不支持 Pin / Ch / PIO 参数。

3. NX1 不支持 Ch / PIO 参数。

5.25.6 ResumePWM

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

用来恢复 PWM-IO 输出，继续计数时间。

ResumePWM

ResumePWM(Pin)

ResumePWM(Ch)

ResumePWM(PIO)

Pin：指示要恢复 PWMOut / PWMOutS 输出的脚位。

- **Px.n**：IC 脚位。
- **EXPx_Py.n**：I/O 扩展芯片的脚位。

Ch：PlayPWM / PlayPWMS 所输出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的输出状态。

注意：

1. NY4 / NY5 / NY6 / NY7 / NY9T 不支持 Pin / Ch / PIO 参数。

2. NX1 不支持 Ch / PIO 参数。

5.25.7 HoldPWM

[NY5+ / NY9T]

暂停目前的 PWM-IO 输出，会维持目前的 duty 输出，但不进入 Sleep 模式。

- NY9T，HoldPWM 可暂停 PWMOut / PlayPWM 的 PWM-IO 输出。
- NY5+，HoldPWM 只可暂停 PlayPWM 的 PWM-IO 输出，对 PWMOut 无作用。

HoldPWM

HoldPWM(Ch)

HoldPWM(PIO)

Ch：PlayPWM / PlayPWMS 所输出的 PWM-IO 通道。

PIO：表示 PlayPWM / PlayPWMS 全部通道的输出状态。

注意：

1. NY9T 不支持 Ch / PIO 参数。

2. NY5+的 HoldPWM 对 PWMOut 无效。

5.25.8 PWMCtrl (@PWMEn, Extension)

[NY9T]

可以开关在 **[I/O]** 段落中已被设定的 PWM-IO 脚位。PWMEn 的使用方式同 PlayPWM 指令，请参考 5.25.2。

例. 在播放 PWM-IO（PE.0~PE.3）的过程中，可以关闭或打开 PWM-IO 的输出。

```
[Path]
PowerOn: PWMCtrl(@1111,1)                ; PWMIO 预设为 1111，Extension=1。
TR1: PlayPWM(X, $VIO0)                   ; 播放”VIO0”，输出到 PE。
```

```
TR2: PWMCtrl(@0011,X)          ; 关闭 PE.2, PE.3 的输出。
TR3: PWMCtrl(X, 4)             ; 改变 Extension 为 4 倍。
```

5.25.9 PWMDuty

[NY5+ / NX1]

修改脚位上的 PWM-IO 输出。

PWMDuty(Pin, Duty)

Pin：指定要修改 PWM-IO 输出的脚位。

- **Px.n**：IC 脚位。
- **EXPx_Py.n**：I/O 扩展芯片的脚位。

Duty：可使用立即值指定输出百分比及阶数或使用变量指定阶数。

- 使用立即值输出阶数
- 使用立即值输出百分比
- 使用 Ri 输出阶数(固定使用 2 个 Ri)
- 使用 Xi 输出阶数(固定使用 1 个 Xi)

注意：PWMDuty 只有在 PWM-IO 进行中才有效。

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 中断指令（Interrupt Command）

Interrupt Command				
INT_ON	INT_OFF	INT_RET	INT = n	-

5.26.1 INT_ON

[NY5 / NY5+ / NX1]

打开时间中断。

注意：

1. 即便是中断打开后，若是无播放指令或延迟指令在执行，则系统会自动进入睡眠模式。

2. NX1 于 PowerOn 时，预设为 INT_ON，使用者无需再呼叫。

3. NX1 的 INT_ON 与 INT_OFF 必须配对使用。

例. NY5 / NY5+

[Path]

```
PowerOn: KEY1, INT_ON                    ; 打开时间中断。
```

例. NX1

```
[Variable]
Var8: count500ms = 0
[Path]
PowerOn: KEY1, INT_ON
500ms: INT_OFF, count500ms++, INT_ON     ; 修改 count500ms 前，将中断关闭，避免同时修改变
                                         量造成错误。
[Interrupt]
RTC_2Hz: count500ms++
```

5.26.2 INT_OFF

[NY5 / NY5+ / NX1]

关闭时间中断。

注意：

1. NX1 的 INT_OFF 与 INT_ON 必须配对使用。

2. NX1 处理器中断程序请勿关闭过久，避免影响系统层级的运作。

例.

```
[Path]
PowerOn: KEY1, INT_OFF                   ; 关闭时间中断。
```

5.26.3 INT_RET

[NY5 / NY5+]

返回中断向量。

注意：有使用跳转指令后，一定要使用 INT_RET 来做返回中断向量的动作，否则将会造成动作不正常。

例.

```
[Path]
PowerOn: KEY1, INT_ON, INT=1.024
Interrupt: R0=0xF?Change, PA=0x0,        ; R0 的值若是等于 0xF，且会跳转到“Change”，
           INT_RET                       ; 否则执行下一个命令。
Change: PA=0xF, INT_RET                  ; 执行完 PA=0xF 后，当执行 INT_RET 时，会返回中
                                         ; 断向量。
```

5.26.4 INT = n

[NY5 / NY5+]

设定时间中断间距。n = 0.256 / 0.512 / 1.024，单位 ms。

例.

[Path]

PowerOn: KEY1, INT_ON, INT=1.024

Interrupt: !PA, INT_RET **；每 1ms 发生时间中断一次，且将 PA 反相输出。**

5.27 时间延迟指令（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 组的 delay，在前景与两个背景分别提供一组。

time 单位：ms（毫秒）或 sec（秒）（默认为秒），最小值=4ms。未满 4ms，无条件进位。

- NY4 / NY5 / NY6 / NY7 / NY9T 最大值=15s。
- NY5+ / NX1 最大可至 0x7FFFFFFF 毫秒。

注意：

1. NX1 于 Q-Code 6.51(含)以前存在-2.34%计时误差，源自于 RTC_1024Hz 与 1ms 的时间差异，(0.9765625ms - 1ms) / 1ms ≒ -2.34%。Q-Code 6.60(含)之后透过软件进行补偿修正。

2. NX1 以 RTC 进行计时模拟，存在 I_LRC 的±1.5%计时误差范围。

3. NX1 采用分时多任务系统架构，注意 1,注意 2 计时误差不包含多任务系统的任务切换耗时。

例. 要执行一个 0.1 sec 的延迟。

[Path]

PowerOn: Delay(0.1) **；延迟 0.1 秒。**

或

PowerOn: Delay(100ms) **；延迟 0.1 秒。**

5.27.2 Delay(Rk:Rj:Ri)

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

根据 RAM 的内容值来设定 Delay 时间。Ri=低位，Rj=中位，Rk=高位。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 时间是以 4.096ms 为最小单位。
- NX1 时间最小单位为 1ms。

例. NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 中，时间延迟 1 sec，如何计算 Ri，Rj，Rk 要填入的值。

1000 ms / 4.096 ms = 244.1（无条件进位），将 245 转换成 Hex，则等于 0xF5。（计数器的最小单位为

4.096ms）

TR1: R2=0x0, R1=0xF, R0=0x5, Delay(R2:R1:R0) ; 延迟 1 sec。

例. NX1 中，时间延迟 1 sec，直接填入 1000 即可。

[Variable]

Var16: Delay_Time

[Path]

TR1: Delay_Time=1000, Delay(Delay_Time) ; 延迟 1 sec。

5.27.3 WaitDN(n)

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

若有 Delay 正在执行中，则 Delay 执行完毕后，再执行下一个指令。没有 Delay 执行中则立即执行下一个指令。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，则等待所有的 delay 执行完毕。

例.

[Path]

TR1: Delay(5)

TR2: WaitDN(1), Out ; 若背景 1 的正在运行时间延迟，则时间延迟执行
; 完后返回执行 Out 指令。

[Background1]

BG1: Delay(5) ; BG1 执行 5 秒时间延迟。

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，则停止所有的 Delay。

例.

[Path]

TR1: BG1 ; 调用 BG1。

TR2: StopD(1) ; 停止 BG1 的时间延迟。

[Background1]

BG1: Delay(5) ; BG1 执行 5 秒时间延迟。

例.

[Path]

PowerOn: BG1, Delay(3) ; 前景延迟 3 秒。

TR1: StopD ; 停止前景和 BG1 的时间延迟。

[Background1]

```
BG1: Delay(5)                    ; BG1 执行 5 秒时间延迟。
```

5.27.5 PauseD(n)

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

可以暂停指定的 Delay。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，则暂停所有的 Delay。

```
[Path]
TR1: BG1                         ; 调用 BG1。
TR2: PauseD(1)                   ; 暂停 BG1 的时间延迟。
TR3: ResumeD(1)                  ; 恢复 BG1 的时间延迟。
[Background1]
BG1: Delay(5)                    ; BG1 执行 5 秒时间延迟。
```

5.27.6 ResumeD(n)

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

可以恢复指定的 Delay。

n：0=前景，1=背景 1，2=背景 2，3=背景 3。n 不指定，则恢复所有的 Delay。

```
[Path]
TR1: BG1                         ; 调用 BG1。
TR2: PauseD(1)                   ; 暂停 BG1 的时间延迟。
TR3: ResumeD(1)                  ; 恢复 BG1 的时间延迟。
[Background1]
BG1: Delay(5)                    ; BG1 执行 5 秒时间延迟。
```

5.27.7 SDelay(time)

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

SDelay 可用来设定短暂的延迟（Short Delay）。time 单位：毫秒或微秒（默认为毫秒）。

注意：

1. ***在延迟过程中无法处理其它的动作，仅能在延迟结束后执行下一个步骤。***
2. ***NY4 / NY5 / NY5+ / NY9T 中的时间范围为 50us ~ 12500us，每次增加 50us。***
3. ***NY6 / NY7 中的时间范围为 10us ~ 4000us，每次增加 10us。***
4. ***NX1 中的时间范围为 1us ~ 4000us，每次增加 1us。***

例. 要执行一个 250us 的延迟。

```
DELAY_TEST: SDelay(250us)        ; 延迟 250us。
```

5.28 动作指令（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]

用户可由 *Q-Visio* 来绘制要输出的 Action 信号，在 *Q-Code* 中可使用 Action 指令来将信号播放出来。

PlayA (Px.n, Ch, {VIOLabel.)Label{, Extension}{, Loop}) {*n} { &[BG1 {,BG2}] }

PlayAS (Px.n, Ch, {VIOLabel.)Label{, Extension}{, Loop}) { &[BG1 {,BG2}] }

PlayA(Ch, VIOLabel{, Extension}{, Loop}) {*n} { & [BG1 {,BG2}] }

PlayAS(Ch, VIOLabel{, Extension}{, Loop}) { & [BG1 {,BG2}] }

PlayA：待 Action 文件播放结束后，执行下一个指令。

PlayAS：Action 文件开始播放，立即执行下一个指令。

Px.n：可指定任一输出脚。

Ch：Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 Ch1 ~ Ch8。
- NX1 支持 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]** 中所定义的多信号组态。*例.* **PlayA(Ch1, $MVIO0, R0)**
- 指定 **[Action]** 中所定义的单信号组态。*例.* **PlayA(PE.0, Ch1, $SVIO0, R0)**

Label：指定欲播放.vio 文件的单一信号。

Extension：可将该信号整个依照倍数来延长，以节省 ROM Size。Extension 的范围为 1~15。

- 直接赋值。 *例.* **PlayA(PE.0, Ch 1, $VIO.A1, 4)**
- Ri（支持到 1 个 Ri） *例.* **PlayA(PE.0, Ch 1, $VIO.A1, R0)**

Loop：让此 Action 信号循环播放，直到有其他的指令来停止它，如 StopA(若不使用此功能则不需写出此参数)。

注意：

1. NX1 不支持播放单一 VIO 文件，仅能播放 VIO 文件中的单一信号。

2. NY4 / NY5 / NY5+ / NY6 / NY7 / NX1 不支持 Extension 参数。

3. *若是播放 VIO 文件，且同时输出多个信号时，须注意各信号的输出时间长度必须相等，若时间长度不相等时，则会以输出时间最长的信号为基准，并且将其余信号输出时间不足的部分强制输出 0%。*
4. *由于 NY9T Action 计数器的分辨率仅有 1ms，因此变化的范围较小，所以不建议用来输出 Ascend / Descend 信号。*
5. *打开 Loop 功能，且同时输出多个信号时，用户需注意所有输出的信号时间长度皆为相同，这样才能无接缝地输出各信号，避免造成信号不连续的问题。*

例.

```
Path1: PlayA(PB.0, Ch 4, $A1)              ; 播放 Label “A1” 于通道 4，输出到 PB.0。
Path2: PlayAS(PB.1, Ch3, $EyeMotion)       ; 播放 Label “EyeMotion” 于通道 3，输出到 PB.1。
Path3: PlayAS(PB.2, Ch2, $A3)              ; 播放 Label “A3” 于通道 2，输出到 PB.2。
Path4: PlayA(PB.2, Ch2, $VIO1.A3)          ; 播放 VIO1 中的 A3 信号于通道 2，输出到 PB.2。
```

例. 使用 **[Action]** 输出多笔 action 信号。

```
[Action File]
VIO0 = Action1.vio                         ; 加入 action 文件。
VIO1 = Action2.vio
[Output State]
Output_0: A   A   A   A   A   A   A   A    ; 使用 PA 及 PB 作为 action 信号输出。
[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 作为输出脚位。
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 输出 action 信号至 PA 及 PB。
TR2: Output_0, PlayA(Ch1, $MVIO1)          ; 使用 MVIO0 输出 action 信号至 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 播放完毕后，再执行下一个指令。没有 Action 播放，则立即执行下一个指令。

Ch：选择 Action 通道。不带此参数则表示全部的 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 Ch1 ~ Ch8。
- NX1 支持 Ch1 ~ Ch32。

5.28.3 PauseA(Ch)

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

暂停正在播放中的指定通道的 Action。

Ch：选择 Action 通道。不带此参数则表示全部的 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 /NY9T 支持 Ch1 ~ Ch8。
- NX1 支持 Ch1 ~ Ch32。

5.28.4 ResumeA(Ch)

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

恢复播放指定通道的 Action。

Ch：选择 Action 通道。不带此参数则表示全部的 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 Ch1 ~ Ch8。
- NX1 支持 Ch1 ~ Ch32。

5.28.5 HoldA(Ch)

[NY9T]

暂停正在播放中的 Action，并维持目前的输出状态。

Ch：选择 Action 通道，Ch=Ch1 ~ Ch8。不带此参数则表示全部 Action 通道。

5.28.6 StopA(Ch)

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

停止选择的 Action 通道。

Ch：选择 Action 通道。不带此参数则表示全部的 Action 通道。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 Ch1 ~ Ch8。
- NX1 支持 Ch1 ~ Ch32。

例.

[Path]

```
TR1: PlayA(PB.0, 1, $A0)          ; 播放$A0 于 Action 通道 1，且输出到 PB0。
TR2: PauseA(Ch1)                  ; 暂停执行 Action 通道 1 的动作。
TR3: Resume(Ch 1)                 ; 恢复执行 Action 通道 1 的动作。
TR4: StopA(Ch 1)                  ; 停止执行 Action 通道 1 的动作。
TR5: WaitAN(Ch 1), Out            ; 若 Action 通道 1 正在执行，Action 通道 1 执行
                                  ; 完后返回执行 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_Birghtness
LEDText_Play	LEDText_PlayS	LEDText_Stop	LEDText_Clear	LEDText_Brightness

5.29.1 LEDStr_Play / LEDStr_PlayS

[NX1]

Single-String 灯串播放指令，播放灯串文件。

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：待灯串播放结束后，执行下一个指令。

LEDStr_PlayS：灯串开始播放，立即执行下一个指令。

Px.n：输出脚位。

LEDLabel：指定要播放的 Single-String LED 文件。

Index：播放文件的索引值，可为立即值或变量。

Storage：LED 文件位置。支持 SPI0 / SPI1，不指定则为内部 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 档案(.csv)。
[Path]
TR1: LEDStr_PlayS(PA.2, $LED0)                ; 播放 LED0 的数据，输出至 PA.2。
TR2: SpiPlayS(ch,0), LEDStr_Play(PA.2, 0)     ; 播放第 0 首 Speech 和第 0 个灯串。
```

5.29.2 LEDStr_Stop

[NX1]

灯串播放停止指令，停止播放中的 Single-String 灯串。

LEDStr_Stop

LEDStr_Stop(Px.n)

Px.n：指定欲停止 Single-String 灯串输出的脚位，不指定则停止所有播放中的 Single-String 灯串。

例.

```
TR1: LEDStr_Stop                    ; 停止播放灯串。
```

5.29.3 LEDStr_Clear

[NX1]

灯串清除指令，点灭 Single-String 灯串上所有 LED。

LEDStr_Clear

LEDStr_Clear(Px.n)

Px.n：指定欲清除 Single-String 灯串输出的脚位，不指定则清除所有 Single-String 灯串。

例.

```
TR1: LEDStr_Clear                   ; 点灭灯串。
```

5.29.4 LEDStr_Brightness

[NX1]

灯串亮度设定指令，调整 Single-String 灯串亮度，n=0 ~ 15。

例.

```
TR1: LEDStr_Brightness(15)          ; 灯串设定最亮。
```

5.29.5 LEDSync_Play / LEDSync_PlayS

[NX1]

Sync-Play 灯串播放指令，播放灯串文件。

LEDSync_Play(LEDLabel)

LEDSync_Play(index)

LEDSync_Play(index, Storage)

LEDSync_PlayS(LEDLabel)

LEDSync_PlayS(index)

LEDSync_PlayS(index, Storage)

LEDSync_Play：待灯串播放结束后，执行下一个指令。

LEDSync_PlayS：灯串开始播放，立即执行下一个指令。

LEDLabel：指定要播放的 Sync-Play LED 文件。

Index：播放文件的索引值，可为立即值或变量。

Storage：LED 文件位置。支持 SPI0 / SPI1，不指定则为内部 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 档案(.csv)。
[Path]
TR1: LEDSync_PlayS($LED0)                    ; 播放 LED0 的数据，输出至 PA.2。
TR2: SpiPlayS(ch,0), LEDSync_Play(PA.2, 0)   ; 播放第 0 首 Speech 和第 0 个灯串。
```

5.29.6 LEDSync_Stop

[NX1]

灯串播放停止指令，停止播放中的 Sync-Play 灯串。

LEDSync_Stop

例.

```
TR1: LEDSync_Stop                            ; 停止播放灯串。
```

5.29.7 LEDSync_Clear

[NX1]

灯串清除指令，点灭 Sync-Play 灯串上所有 LED。

LEDSync_Clear

例.

```
TR1: LEDSync_Clear                           ; 点灭灯串。
```

5.29.8 LEDSync_Brightness

[NX1]

Sync-Play 灯串亮度设定指令，调整灯串亮度，n=0 ~ 15。

例.

```
TR1: LEDSync_Brightness(15)                  ; 灯串设定最亮。
```

5.29.9 LEDText_Play / LEDText_PlayS

[NX1]

Single-String 灯串播放指令，播放灯串文件。

LEDText_Play(LEDLabel)

LEDText_Play(index)

LEDText_Play(index, Storage)

LEDText_PlayS(LEDLabel)

LEDText_PlayS(index)

LEDText_PlayS(index, Storage)

LEDText_Play：待灯串播放结束后，执行下一个指令。

LEDText_PlayS：灯串开始播放，立即执行下一个指令。

LEDLabel：指定要播放的 Scrolling-Text LED 文件。

Index：播放文件的索引值，可为立即值或变量。

Storage：LED 文件位置。支持 SPI0 / SPI1，不指定则为内部 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 档案(.csv)。
[Path]
TR1: LEDText_PlayS($LED0)                    ; 播放 LED0 的数据，输出至 PA.2。
TR2: SpiPlayS(ch,0), LEDText_Play(0)         ; 播放第 0 首 Speech 和第 0 个灯串。
```

5.29.10 LEDText_Stop

[NX1]

灯串播放停止指令，停止播放中的 Scrolling-Text 灯串。

LEDText_Stop

Px.n：指定欲停止灯串输出的脚位，不指定则停止所有播放中的灯串。

例.

```
TR1: LEDText_Stop                            ; 停止播放灯串。
```

5.29.11 LEDText_Clear

[NX1]

灯串清除指令，点灭 Scrolling-Text 灯串上所有 LED。

LEDText_Clear

例.

```
TR1: LEDText_Clear                    ; 点灭灯串。
```

5.29.12 LEDText_Brightness

[NX1]

灯串亮度设定指令，调整 Scrolling-Text 灯串亮度，n=0 ~ 15。

例.

```
TR1: LEDText_Brightness(15)           ; 灯串设定最亮。
```

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]

扫描所有 QFID Group 的状态。Ri[7:0]为 Group 的状态，对应的 bit 为 1 表示有扫描到该 Group 的 Tag，若为 0 则无该 Group 的 Tag。

5.30.2 QFID_TagId (GroupID, Rl:Rk:Rj:Ri)

[NY5+ / NY7 / NX1]

扫描所有 Tag 的状态。

GroupID：为指定的 QFID Group ID，可为立即值或用变数来指定。

Ri：为 Tag0 ~ Tag3 的状态。对应的 bit 为 1 表示有侦测到对应的 Tag，若为 0 则无。

Rj：为 Tag4 ~ Tag7 的状态。对应的 bit 为 1 表示有侦测到对应的 Tag，若为 0 则无。

Rk：为 Tag8 ~ Tag11 的状态。对应的 bit 为 1 表示有侦测到对应的 Tag，若为 0 则无。

Rl：为 Tag12 ~ Tag15 的状态。对应的 bit 为 1 表示有侦测到对应的 Tag，若为 0 则无。

注意：NY5+ / NY7 不可指定 GroupID。

5.30.3 QFID_TagInput (GroupID, ID, Ri)

[NY5+ / NY7 / NX1]

读回 Tag 上三支 I/O 的输入状态。

GroupID：为指定的 QFID Group ID。

ID：为指定的 Tag ID，范围为 0~15，可为立即值或用变数来指定。

Ri：为读回的 I/O 状态，Ri[0]表示 I/O1 状态，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]

打开 QFID 扫描功能，开机后 QFID 功能默认为关闭。。

5.30.5 QFID_Off

[NY5+ / NY7 / NX1]

关闭 QFID 扫描功能。

5.30.6 QFID_SlowOn

[NY5+ / NY7]

打开 QFID 的低速扫描模式，模式打开后，QFID 会以扫描一次休息数次的模式动作，休息的次数由 QFID Option 内的设定值决定，详细说明请参考 QFID 段落小节；打开此模式可以降低消耗电流，但反应速度也会同时降低。

注意：

1. ***此指令与一般指令的 Slow 指令不能同时执行，否则会造成休息时间长度错误。***
2. ***使用此指令前必须在 QFID Option 内选择 slow 脚，并在脚位上并联一颗电阻及一颗电容，否则动作会不正常；电阻及电容值请参考 Option 内的 QFID Reader Application Circuit。***

5.30.7 QFID_SlowOff

[NY5+ / NY7]

关闭 QFID 的低速扫描模式。

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]

代码发射。

Ri：为指定的代码，可为立即值或用变数来指定。

例.

```
[Option]
WaveID = TX                                  ; 打开 WaveID TX 功能。
WaveID_CommandCount = 8
[Path]
TR1: WaveID_TX(5)                            ; 发射代码 5。
TR2: WaveID_TX(R0)                           ; 发射代码 R0。
```

5.31.2 WaveID_RX_ON

[NX1]

打开 WaveID 接收功能。

例.

```
[Path]
TR1: WaveID_RX_ON                            ; 打开接收。
```

5.31.3 WaveID_RX_OFF

[NX1]

关闭 WaveID 接收功能。

例.

```
[Path]
TR1: WaveID_RX_OFF                           ; 关闭接收。
```

5.31.4 WaveID_RX

[NX1]

用户可通过指令将接收到的 WaveID 代码存在指定的变量。

例.

```
[Option]
WaveID = RX                                  ; 开启 WaveID RX 功能。
WaveID_CommandCount = 8
[Variable]
Var8: WaveId_Value
[Path]
```

```
TR1: WaveID_RX_On                          ; 启用 WavdID RX。
WaveID_RX: WaveId_Value = WaveID_RX        ; 将接收到代码存到 Var0。
```

5.32 听声辨位指令（Sound Localization Command）

Sound Localization Command				
SL_On	SL_Off	SL_RX	-	-

5.32.1 SL_On

[NX1]

开启听声辨位扫描功能，开机后默认为关闭。

例.

```
[Option]
SoundLoc = Enable                          ; 开启听声辨位功能。
SoundLoc_Amp_Pin = PA.15
SoundLoc_Amp_Shutdown_Control = Active_Low
[Variable]
Var8: R_Angle
[Path]
TR1: SL_On                                 ; 启用听声辨位。
SL_RX: R_Angle=SL_RX, R_Angle=30?SL_30, R_Angle=60?SL_60
SL_30: PlayV(Ch0, $V0)                     ; 当 R_Angle 为 30，播放语音$V0。
SL_60: PlayV(Ch0, $V1)                     ; 当 R_Angle 为 60，播放语音$V1。
```

5.32.2 SL_Off

[NX1]

关闭听声辨位扫描功能。

5.32.3 Var=SL_RX

[NX1]

读取听声辨位侦测到的角度，将数值存放于 Var，读回的角度可为 0 / 30 / 60 / 90 / 120 / 150 / 180 / 210 / 240 / 270 / 300 / 330 共 12 个方位值，建议搭配 SL_RX 路径使用。

5.33 聲音偵測指令（Sound Detect Command）

Sound Detect Command				
SoundDetect_On	SoundDetect_Off	-	-	-

5.33.1 SoundDetect_On

[NX1]

启用声音侦测功能。

例.

[Option]
SoundDetect = Enable
SoundDetect_High_Threshold = 300
SoundDetect_Low_Threshold = 300
SoundDetect_Active_Time = 80ms
SoundDetect_Mute_Time = 650ms
[Path]
TR1: SoundDetect_On ；启用声音侦测。
Sound_Detect: PA.0 = 1 ；侦测到声音时，PA.0 输出 1。
Sound_Mute: PA.0 = 0 ；当不再侦测到声音超过 150ms 时，PA.0 输出 0。

5.33.2 SoundDetect_Off

[NX1]

停用声音侦测功能。

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]

启用音高侦测功能。

例.

[Option]
PitchDetect = Tone ；侦测 Pure-Tone。
PitchDetect_Range = Middle ；侦测范围 1000 ~ 3000Hz。

```
[Variable]
Var16: Pitch_Value
[Path]
TR1: PitchDetect_On                    ; 开启音高侦测。
TR2: PitchDetect_Off                   ; 关闭音高侦测。
Pitch_Detected: ReadPitch(Pitch_Value) ; 侦测到 1000 ~ 3000Hz 内的 pure-tone，
                                       ; 将侦测到的音高储存至 Pitch_Value。
```

5.34.2 PitchDetect_Off

[NX1]

停用音高侦测功能。

5.34.3 ReadPitch

[NX1]

读取侦测到的音高。

ReadPitch(Var)

Var：将侦测到的音高储存至 Var，范围 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]

打开 RFC 扫描功能，开机后 RFC 功能默认为关闭。

5.35.2 RFC_Off

[NY5+ / NY6 / NY7 / NX1]

关闭 RFC 扫描功能。

5.35.3 RFC_Level(Ri)

[NY5+ / NY6 / NY7 / NX1]

读取侦测到的 RFC 数值，将数值存放于 Ri。

注意：

1. *开机后需使用 RFC_On 指令将 RFC 功能打开，并等待初始时间约 30ms 后才能利用此指令取到正确的 RFC 数值，否则可能会取到未知数值。*
2. *使用此指令前必须在 RFC 脚位接上一个电位器及一颗电容，电路接法请参考 RFC Option 内的 RFC Application Circuit。*

5.36 ADC 输入指令（ADC Input Command）

ADC Input Command				
ADC_Input	-	-	-	-

5.36.1 ADC_Input

[NX1]

读取模拟数字转换的值。

ADC_Input(Px.n, Var)

Px.n：由指定脚位读取模拟数字转换结果。

Var：将读取的模拟数字转换结果存放于 Var，结果范围为 0 ~ 0xFFF0。

例.

```
[Option]
ADC_Input_Pins = PA.1, PA.2
[Variable]
Var16: Result
[Path]
TR: ADC_Input(PA.2, Result)          ; 读取 PA.2 模拟数字转换的值，
                                     ; 并将结果存放于 Result。
```

5.37 语音识别指令（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 用于定义语音指令的状态，VR 状态是指当语音指令被辨识后所应该作出的反应，用户可以设

定不同的 VR State，在不同的情况下语音指令被辨识后所作出的反应也可以不同。语音指令可以分成多个群组，但同时只有一个指令群组能被辨识。

例.

```
[VR]
VRG1_0: VR1   VR2   VR3                    ; 语音识别指令群组 1 状态 0
VRG2_0: VR4   VR5                          ; 语音识别指令群组 2 状态 0
[Path]
PowerOn: VRG1_0                            ; VR State 设置成 VRG1_0 状态，只有第一组指令
                                           可以被辨识
```

5.37.2 VR_ON

[NX1]

启用语音识别功能。启用语音识别功能后，*Q-Code_NX1* 会不断进行辨识是否有语音指令。请注意当播放声音时，语音识别功能会自动暂停，直到声音播放结束，语音识别功能才会自动回复，继续进行语音指令的辨识。启用语音识别功能之前要先设定 VR State，在语音识别功能打开时可以随时切换 VR State。

例.

```
[VR]
VRG1_0: VR1   VR2   VR3
[Path]
PowerOn: VRG1_0, VR_ON                     ; 打开语音识别功能
```

5.37.3 VR_OFF

[NX1]

关闭语音识别功能。

例.

```
[Input State]
KEY1: TR1   TR2   X /TR4
[VR]
VRG1_0: VR1   VR2   VR3
[Path]
PowerOn: KEY1,VRG1_0,VR_ON
TR1: VR_OFF                                ; 关闭语音识别功能
```

5.37.4 VR_VAD=n

[NX1]

设定语音识别时输入音量的阀值。语音启动检测 VAD（Voice Activity Detection）技术用于检测语音信号是否存在，设定其阀值也就是在某个音量以上才进行辨识。阀值愈大可以滤除愈大的背景噪声，但也可能将小音量的语音指令滤除；而阀值太小可能无法将背景噪声排除以防止误动作，因此用户需以使用环境来考虑阀值。当未设定时，默认值为 4。

此 VAD 阀值相当于 CSMT（Cyberon CSpotter Modeling Tool）中的 energy threshold，惟两者阀值的定义不相同，用户可择一使用，若 energy threshold > 0，则可设定 VR_VAD_Off；若 VR_VAD_On 则 energy threshold 可设为 0。阀值设定后还需以 VR_VAD_On 指令启动。预设是 VR_VAD_Off。

n：阀值，可为立即值或变数，n= 0 ~ 16。若需要设定此表以外之值可利用 C Code 段落来直接调用底层的 API。

阀值	平均音量（%）	平均音量（Sample）	说明
0	0.00%	0	不滤除背景噪声，即不启用 VAD
1	0.46%	150	适用于非常安静的封闭房间
2	0.61%	200	适用于有轻微背景噪音但没有说话的封闭房间
3	0.76%	250	适用于有背景噪音或音乐但没有说话的房间
4	0.92%	300	适用于 3 公尺外有说话的房间
5	1.07%	350	适用于 1 公尺外有说话的房间
6	1.22%	400	适用于 1 公尺内近距离有轻微活动的房间
7	1.53%	500	适用于 1 公尺内近距离有活动的房间
8	1.83%	600	适用于 1 公尺内近距离有说话声的房间
9	2.14%	700	适用于较安静的开放空间
10	2.44%	800	适用于有些微背景噪音的开放空间
11	3.66%	1000	适用于有比较大的背景噪音的开放空间
12	3.66%	1200	适用于有较大的背景噪音或近距离有说话声的开放空间
13	4.58%	1500	适用于近距离 1 公尺内有活动，或有说话声的开放空间
14	6.10%	2000	适用于近距离 1 公尺内有活动，或有较大说话声的开放空间
15	9.16%	3000	适用于较吵杂的外围环境，需要大音量才能辨识
16	13.73%	4500	适用于非常吵杂的外围环境如展场，需要更大音量才能辨识

注意：若 energy threshold > 0，RAM 会多耗费约 300 bytes；而 VAD 仅多耗费 12 bytes。

例.

```
VR_VAD=1                                    ; 设定 VAD 阀值为 1
```

5.37.5 VR_VAD_On

[NX1]

打开语音识别的 VAD 功能。当启动 VAD，语音指令的音量须大于阀值，才会被判读和辨识。

例.

```
VR_VAD_On                                    ; 打开 VAD
```

5.37.6 VR_VAD_Off

[NX1]

关闭语音识别的 VAD 功能。

例.

```
VR_VAD_Off                                   ; 关闭 VAD
```

5.37.7 VRGC_Timeout_CLR

[NX1]

清除 VRGC state 的 Timeout 定时器。当 VRGC 任何群组中有指令被辨识成功时，Timeout 定时器即被启动，当语音指令组合已经全部的辨识完成，可使用此指令清除 Timeout 定时器。

例.

```
VRGC_Timeout_CLR                             ; 清除 Timeout 定时器
```

5.37.8 Ri = VR_HitScore

[NX1]

取得目前已成功辨识的语音指令的分数，分数愈高表示愈符合语音指令。必须在语音指令成功辨识之后才可使用此指令。

例.

```
R0 = VR_HitScore                             ; 将语音指令辨识分数储存至变量 R0
```

5.37.9 Ri = VR_HitID

[NX1]

取得目前已成功辨识的语音指令的索引值。必须在语音指令成功辨识之后才可使用此指令。

例.

```
R0 = VR_HitID                                ; 将语音指令索引值储存至变量 R0
```

5.37.10 VR_Loading

[NX1]

取得语音识别执行时，实时的 CPU 负载，单位为百分比

例.

```
R0 = VR_Loading                              ; 将 CPU 负载储存至变量 R0
```

5.37.11 VT_Training

[NX1]

训练语音指令，训练成功的指令，可于语音识别时使用。

例.

```
VT_Training(1)                    ; 开始训练第一组语音指令。
```

5.37.12 VT_Delete

[NX1]

删除已训练成功的语音指令。

例.

```
VT_Delete(1)                      ; 删除第一组语音指令。
```

5.37.13 VT_DeleteAll

[NX1]

删除所有训练成功的语音指令。

例.

```
VT_DeleteAll                      ; 删除所有语音指令。
```

5.37.14 VT_TrainingNum

[NX1]

设定训练时录音的次数，最多 3 次，最少 1 次，次数越多会越容易识别。Voice Tag 预设 3 次，Voice Password 预设 1 次。

例.

```
VT_TrainingNum=3                  ; 录音次数设定 3 次。
```

5.38 变音效果指令（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

5.38.1 PitchChange

[NX1]

改变声音的音高。声音音高最高可调整为正常音高的两倍，声音音高最低可调整为正常音高的一半。

PitchChange(Channel, Pitch)

Channel：要改变的通道

Pitch：常数或变数，范围必须落在-12 ~ +12 之间。

数值	说明
-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	关闭 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 等同于 PitchChange_Off。***
2. ***若程序使用两个通道，PitchChange 同时只能使用一通道，不允许同时两通道 PitchChange。***
3. ***PitchChange 所指定的通道与 SpeedChange 所指定的通道同时操作必须使用相同通道。***
4. ***此指令只对 Voice 文件有作用，MIDI 文件播放不受此效果影响。***

5. 此指令只对取样频率小于或等于 16KHz 的 Voice 文件有作用。

6. 此指令只对 ADPCM / PCM / SBC-1 / CELP 有作用。

例. **PitchChange(ch1, 1)**

5.38.2 PitchChange_Off

[NX1]

关闭 PitchChange 效果。

注意：必须有使用 PitchChange 方可使用 PitchChange_Off。

5.38.3 SpeedChange

[NX1]

改变声音播放速度，声音播放速度最高可调整为正常播放速度的两倍。声音播放速度最低可调整为正常播放速度的一半。

SpeedChange(channel, number)

Channel：作用通道

Number：常数或变数，范围必须落在-12 ~ +12 之间。

数值	说明
-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	关闭 SpeedChange
1	1.08X
2	1.17X
3	1.25X
4	1.33X
5	1.42X

数值	说明
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 等同于 SpeedChange_Off。*
2. *若程序使用两个通道，SpeedChange 同时只能使用一通道，不允许同时两通道 SpeedChange。*
3. *PitchChange 所指定的通道与 SpeedChange 所指定的通道必须相同。*
4. *此指令只对 Voice 文件有作用，MIDI 文件播放不受此效果影响。*
5. *此指令只对取样频率小于 16K 的 Voice 文件有作用。*
6. *此指令只对 ADPCM / PCM / SBC-1 / CELP 有作用。*

例. **SpeedChange(ch1, 1)**

5.38.4 SpeedChange_Off

[NX1]

关闭 SpeedChange 效果。

注意：必须有使用 SpeedChange 方可使用 SpeedChange_Off。

5.38.5 Robot1

[NX1]

用来产生机器人讲话的效果。

Robot1(Channel, number)

Channel：通道

Number：范围 0 ~ 15

注意：

1. *此指令只对 Voice 文件有作用，MIDI 文件播放不受此效果影响。*
2. *此指令只对 ADPCM / PCM / SBC-1 / CELP 有作用。*

5.38.6 Robot1_Off

[NX1]

关闭 Robot1 效果。

注意：必须有使用 Robot1 方可使用 Robot1_Off。

5.38.7 Robot2

[NX1]

用来产生机器人讲话的效果。

Robot2(Channel, Type,Thres)

Channel：语音通道

Type：范围 0 ~ 2，表示不同音阶

Thres：范围 0~7，噪音门坎值，一般语音播放因为没有外界噪声，可设成 0。

注意：

1. 若程序使用两个通道，Robot2 同时只能使用一通道，不允许同时两通道 Robot2。

2. 此指令只对 Voice 文件有作用，MIDI 文件播放不受此效果影响。

3. 此指令只对 ADPCM / PCM / SBC-1 / CELP 有作用。

例. **Robot2(ch0, 2,0)**

5.38.8 Robot2_Off

[NX1]

关闭 Robot2 效果。

注意：必须有使用 Robot2 方可使用 Robot2_Off。

5.38.9 Robot3

[NX1]

用来产生机器人讲话的效果。

Robot3(Channel, Type, Thres)

Channel：语音通道

Type：范围 0 ~ 2，表示不同音阶

Thres：范围 0~7，噪音门坎值，一般语音播放因为没有外界噪声，可设成 0

注意：

1. 若程序使用两个通道，Robot3 同时只能使用一通道，不允许同时两通道 Robot3。

2. 此指令只对 Voice 文件有作用，MIDI 文件播放不受此效果影响。

3. 此指令只对 ADPCM / PCM / SBC-1 / CELP 有作用。

例. **Robot3(ch0, 2,0)**

5.38.10 Robot3_Off

[NX1]

关闭 Robot3 效果。

注意：必须有使用 Robot3 方可使用 Robot3_Off。

5.38.11 Robot4

[NX1]

用来产生机器人讲话的效果。

Robot4(Channel)

Channel：语音通道，仅支持 Ch0。

注意：

1. 此指令仅支持 ADPCM。

2. 此指令只对 Voice 文件有作用，MIDI 文件播放不受此效果影响。

例. **Robot4(Ch0)**

5.38.12 Robot4_Off

[NX1]

关闭 Robot4 效果。

注意：必须有使用 Robot4 方可使用 Robot4_Off。

5.38.13 Echo

[NX1]

用来产生回音的效果。

Echo(Channel, Level)

Channel：语音通道

Level：范围 0 ~ 7，表示不同回音程度，7 是最长，此参数仅对 SBC/ADPCM Chann1/CELP/PCM 有效

Option ADPCM_ECHO_DELAY：short 表示短的回音效果，long 表示长的回音效果，此 option 仅对 Channel0 且声音压缩为 ADPCM 才有效

注意：

1. 若程序使用两个通道，Echo 同时只能使用一通道，不允许同时两通道 Echo。

2. ADPCM Channel0 的 Echo 效果较佳，建议使用 Echo 音效时，选择 ADPCM 并用 Channel0 播放。

3. 此指令只对 Voice 文件有作用，MIDI 文件播放不受此效果影响。

4. 此指令只对 ADPCM / PCM / SBC-1 / CELP 有作用。

例.

```
[Option]
ADPCM_ECHO_DELAY = Short                ; Echo 回音 delay 选择短的
[Path]
PowerOn: PlayV(ch0,$V0), Echo(ch0,0)    ; 播放 Echo 音效
```

5.38.14 Echo_Off

[NX1]

关闭 Echo 效果。

注意：必须有使用 Echo 方可使用 Echo_Off。

5.38.15 Reverb

[NX1]

用来产生回响的效果。

Reverb(Channel, Level)

Channel：语音通道

Level：范围 0 ~ 7，表示不同回响程度，7 程度最高

注意：

1. 若程序使用两个通道，Reverb 同时只能使用一通道，不允许同时两通道 Reverb。

2. 此指令只对 Voice 文件有作用，MIDI 文件播放不受此效果影响。

3. 此指令只对 ADPCM / PCM / SBC-1 / CELP 有作用。

5.38.16 Reverb_Off

[NX1]

关闭 Reverb 效果。

注意：必须有使用 Reverb 方可使用 Reverb_Off。

5.38.17 Darth

[NX1]

用来产生黑武士的效果。

Darth(Channel, Pitch, Type)

Channel：语音通道，仅支持 Ch0。

Pitch：范围-12 ~ 12，音高设定，设定同 **PitchChange**。

Type：范围 0 ~ 3，设定黑武士类别，金属音设定 0，金属音+沙哑音设定 1/2，沙哑音设定 3。

注意：

1. 此指令仅支持 ADPCM。

2. 此指令只对 Voice 文件有作用，MIDI 文件播放不受此效果影响。

例. **Darth(Ch0,-4,2)**

5.38.18 Darth_Off

[NX1]

关闭 Darth 效果。

注意：必须有使用 Darth 方可使用 Darth_Off。

5.38.19 AnimalRoar

[NX1]

用来产生 AnimalRoar 的效果。

AnimalRoar(Channel, Speed, Pitch, Reverb)

Channel：语音通道，仅支持 Ch0。

Speed：范围-12 ~ 12，速度设定，设定同 **SpeedChange** 的 Number 参数。

Pitch：范围-12 ~ 12，音高设定，设定同 **PitchChange**。

Reverb：范围 0 ~ 3。

注意：

1. 此指令仅支持 ADPCM。

2. 此指令只对 Voice 文件有作用，MIDI 文件播放不受此效果影响。

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]

关闭 AnimalRoar 效果。

注意：必须有使用 AnimalRoar 方可使用 AnimalRoar_Off 。

5.39 实时播放指令（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]

实时播放(大声公)指令。

注意：使用该指令时，语音通道最多支持两通道。

例.

```
[Option]
RT_Sampling_Rate = 8000                ; 实时播放的取样频率 8k，亦可设定 12k 或 16k
[Path]
PowerOn: RT_Play                       ; 实时播放
```

5.39.2 RT_Play_Off

[NX1]

关闭实时播放。

注意：必须有使用 RT_Play 方可使用 RT_Play_Off。

5.39.3 RT_PitchChange

[NX1]

实时播放的音高设定。声音音高最高可调整为正常音高的两倍，声音音高最低可调整为正常音高的一半。

RT_PitchChange(Pitch)

Pitch：常数或变数，范围必须落在-12 ~ +12 之间。

数值	说明
-12	0.5X
-11	0.53X
-10	0.56X
-9	0.59X
-8	0.63X
-7	0.67X

数值	说明
-6	0.72X
-5	0.75X
-4	0.79X
-3	0.84X
-2	0.89X
-1	0.94X
0	关闭 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 等同于 RT_PitchChange_Off。

例. **RT_PitchChange(1)**

5.39.4 RT_PitchChange_Off

[NX1]

关闭实时播放音高变化。

注意：必须有使用 RT_PitchChange 方可使用 RT_PitchChange_Off。

5.39.5 RT_Robot1

[NX1]

用来产生机械音的效果。

RT_Robot1(Number)

Number：范围 0 ~ 15。

例. **RT_Robot1(1)**

5.39.6 RT_Robot1_Off

[NX1]

关闭实时播放器械音。

注意：必须有使用 RT_Robot1 方可使用 RT_Robot1_Off。

5.39.7 RT_Robot2

[NX1]

用来产生机械音的效果。

RT_Robot2(Type,Thres)

Type：范围 0 ~ 2，表示不同音阶。

Thres：范围 0~7，噪音门坎值，数值越高门坎值越高，适合较吵杂的环境。

例. **RT_Robot2(1,3)**

5.39.8 RT_Robot2_Off

[NX1]

关闭实时播放器械音。

注意：必须有使用 RT_Robot2 方可使用 RT_Robot2_Off。

5.39.9 RT_Robot3

[NX1]

用来产生机械音的效果。

RT_Robot3(Type,Thres)

Type：范围 0 ~ 2，表示不同音阶。

Thres：范围 0~7，噪音门坎值，数值越高门坎值越高，适合较吵杂的环境。

例. **RT_Robot3(1,3)**

5.39.10 RT_Robot3_Off

[NX1]

关闭实时播放器械音。

注意：必须有使用 RT_Robot3 方可使用 RT_Robot3_Off。

5.39.11 RT_Robot4

[NX1]

用来产生机械音的效果。

例. **RT_Robot4**

5.39.12 RT_Robot4_Off

[NX1]

关闭实时播放器械音。

注意：必须有使用 RT_Robot4 方可使用 RT_Robot4_Off。

5.39.13 RT_Echo

[NX1]

用来产生回音的效果。

Option RT_ECHO_DELAY：short 表示短的回音效果，long 表示长的回音效果。

例.

```
[Option]
RT_ECHO_DELAY = Short                    ; Echo 回音 delay 选择短的
[Path]
PowerOn: RT_Play, RT_Echo                ; 实时播放使用 Echo 音效
```

5.39.14 RT_Echo_Off

[NX1]

关闭实时播放 Echo 功能。

注意：必须有使用 RT_Echo 方可使用 RT_Echo_Off。

5.39.15 RT_Reverb

[NX1]

用来产生回响的效果。

RT_Reverb(Level)

Level：范围 0 ~ 7，表示不同回响程度，7 程度最高

例. **RT_Reverb(7)**

5.39.16 RT_Reverb_Off

[NX1]

关闭实时播放 Reverb 功能。

注意：必须有使用 RT_Reverb 方可使用 RT_Reverb_Off。

5.39.17 RT_Ghost

[NX1]

用来产生鬼声的效果。

RT_Ghost(Pitch,Type)

Pitch：范围-12 ~ 12，音高设定，女生建议设定 1~12，男生建议设定-1~-12。

Type：范围 0 ~ 2，模拟鬼讲话建议设定 0~1，模拟鬼叫声建议设定 2。

例. **RT_Ghost(12,2)**

5.39.18 RT_Ghost_Off

[NX1]

关闭实时播放鬼声功能。

注意：必须有使用 RT_Ghost 方可使用 RT_Ghost_Off。

5.39.19 RT_Darth

[NX1]

用来产生黑武士的效果。

RT_Darth(Pitch,Type)

Pitch：范围-12 ~ 12，音高设定，设定同 **RT_PitchChange**。

Type：范围 0 ~ 3，设定黑武士类别，金属音设定 0，金属音+沙哑音设定 1/2，沙哑音设定 3。

例. **RT_Darth(-4,2)**

5.39.20 RT_Darth_Off

[NX1]

关闭实时播放黑武士功能。

注意：必须有使用 RT_Darth 方可使用 RT_Darth_Off 。

5.39.21 RT_Chorus

[NX1]

用来产生合音的效果。

RT_Chorus(Pitch, Vol_1, Vol_2, Vol_3, Gain)

Pitch：范围-12 ~ 12，音高设定，设定同 **RT_PitchChange**，合音音轨 1 与合音音轨 3 是可随着 Pitch 调整音调，合音音轨 2 维持原音。

Vol_1：范围 0 ~ 100，设定合音音轨 1 的音量。

Vol_2：范围 0 ~ 100，设定合音音轨 2 的音量。

Vol_3：范围 0 ~ 100，设定合音音轨 3 的音量。

Gain：范围 0 ~ 10，设定整体音量增益。

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]

关闭实时播放合音功能。

注意：必须有使用 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。设定的音量可为立即值或用变量控制。

n：通道音量，支持 0 ~ 15。

注意：必须有使用 RT 相关指令才能使用 RT_Vol 指令。

例. **RT_Vol = 10** ；**将 RT 的音量设为 10，约原本音量的 67%。**

例. **RT_Vol = R0** ；**取 R0 内的数值当成 RT 的音量。**

5.39.24 Var = RT_Vol

[NX1]

读取 RT 的音量。

注意：必须有使用 RT 相关指令才能使用 RT_Vol 指令。

例. **R0 = RT_Vol** ；**读取 RT 的音量，并存于 R0。**

5.40 动物变音指令（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]

开启动物变音功能。

注意：使用动物变音前必须叫用 Animaltalks_On 以配置系统资源。

例.

[Path]

TR1: AnimalTalks_On ；开启动物变音功能。

5.40.2 Animaltalks_Off

[NX1]

关闭动物变音功能。

注意：动物变音使用完毕后必须叫用 Animaltalks_Off 以回收系统资源。

例.

[Path]

TR1: AnimalTalks_Off ；关闭动物变音功能。

5.40.3 Animaltalks_Record / Animaltalks_RecordS

[NX1]

进行动物变音录音。录音完成后，可以透过 AnimalTalks_Play / Animaltalks_PlayS 播放。

Animaltalks_Record(Label {,Time}) { & [BG1 {,BG2 {,BG3}}] }

Animaltalks_RecordS (Label {, Time})

Animaltalks_Record：待录音结束后，执行下一个指令。

Animaltalks_RecordS：开始录音后，立即执行下一个指令。

Label：**[Record]** 中定义的录音区段名称。

Time：录音的长度，未定则为该录音区段的长度。

例.

[Path]

```
TR1: Animaltalks_Record($Rec0)          ; 进行动物变音录音，使用 Rec0 区段。
TR2: Animaltalks_Play                   ; 使用 Animaltalks_Play 播放录音结果。
```

5.40.4 Animaltalks_StopR

[NX1]

用来停止动物变音录音。

例.

```
[Path]
TR1: Animaltalks_Record($Rec0)
TR2: Animaltalks_StopR                  ; 停止当前的录音。
```

5.40.5 Animaltalks_Play / Animaltalks_PlayS

[NX1]

进行动物变音播音。

Animaltalks_Play：待播放结束后，执行下一个指令。

Animaltalks_PlayS：开始播放后，立即执行下一个指令。

注意：动物变音播放会直接使用有音效的 Audio 信道，原播放信道将会直接停止。

例.

```
[Path]
TR1: Animaltalks_Play                   ; 使用 Animaltalks_Play 播放录音结果。
```

5.40.6 Animaltalks_Stop

[NX1]

用来停止动物变音播音。

例.

```
[Path]
TR1: Animaltalks_Play
TR2: Animaltalks_Stop                   ; 停止当前的播音。
```

5.40.7 Animaltalks_SetVoice

[NX1]

设定 AnimalTalks_Play / Animaltalks_PlayS 时使用的动物原音音档。

Animaltalks_SetVoice(Index {, Storage})

Index：动物音文件索引。

Storage：播放的动物音文件所在的储存空间，若不指定则为内部空间，可使用 udr、spi0、spi1。

例.

[Path]

```
TR1: Animaltalks_SetVoice(2, spi0)        ; 设定动物变音音檔，使用 SPI0 的索引 2 音檔。
TR2: Animaltalks_Play                     ; 使用 Animaltalks_Play 播放录音结果。
```

5.40.8 Animaltalks_LongSound_On

[NX1]

设定长动物音模式，会略过静音段的播放。

Animaltalks_LongSound_On(Factor)

Factor：播放段落持续时间系数，设定范围 1% ~ 65535%。

例.

[Path]

```
TR1: Animaltalks_SetVoice(2, spi0)        ; 设定动物变音音檔，使用 SPI0 的索引 2 音檔。
TR2: Animaltalks_LongSound_On(300%)       ; 设定长音音档播放时间系数 300%。
TR3: Animaltalks_Play                     ; 使用 Animaltalks_Play 播放录音结果。
```

5.40.9 Animaltalks_LongSound_Off

[NX1]

关闭长动物音模式。

例.

[Path]

```
TR1: Animaltalks_SetVoice(2, spi0)        ; 设定动物变音音檔，使用 SPI0 的索引 2 音檔。
TR2: Animaltalks_LongSound_Off            ; 关闭长音音文件模式。
TR3: Animaltalks_Play                     ; 使用 Animaltalks_Play 播放录音结果。
```

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]

开启动物唱歌功能。

注意：使用动物唱歌前必须叫用 Animalsings_On 以配置系统资源。

例.

[Path]

TR1: AnimalSings_On ；开启动物唱歌功能。

5.41.2 Animalsings_Off

[NX1]

关闭动物唱歌功能。

注意：动物唱歌使用完毕后必须叫用 Animalsings_Off 以回收系统资源。

例.

[Path]

TR1: AnimalSings_Off ；关闭动物唱歌功能。

5.41.3 Animalsings_Record / Animalsings_RecordS

[NX1]

进行动物唱歌的录音。录音完成后，可以透过 AnimalSings_Play / AnimalSings_PlayS 播放。

Animalsings_Record(Label {,Time})

Animalsings_RecordS (Label {, Time})

Animalsings_Record：待录音结束后，执行下一个指令。

Animalsings_RecordS：开始录音后，立即执行下一个指令。

Label：**[Record]** 中定义的录音区段名称。

Time：录音的长度，未定则为该录音区段的长度。

例.

[Path]

TR1: Animasings_Record($Rec0) ；进行动物唱歌的录音，使用 **Rec0** 区段。

TR2: Animalsings_Play ；使用 **Animalsings_Play** 播放录音结果。

5.41.4 Animalsings_StopR

[NX1]

用来停止动物唱歌的录音。

例.

[Path]

TR1: Animalsings_Record($Rec0)

TR2: Animalsings_StopR ; 停止当前的录音。

5.41.5 Animalsings_Play / Animalsings_PlayS

[NX1]

进行动物唱歌的播音。

Animalsings_Play：待播放结束后，执行下一个指令。

Animalsings_PlayS：开始播放后，立即执行下一个指令。

注意：动物唱歌的播放会直接使用有音效的 Audio 信道，原播放信道将会直接停止。

例.

[Path]

TR1: Animalsings_Play ; 使用 Animalsings_Play 播放录音结果。

5.41.6 Animalsings_Stop

[NX1]

用来停止动物唱歌的播音。

例.

[Path]

TR1: Animalsings_Play

TR2: Animalsings_Stop ; 停止当前的播音。

5.41.7 Animalsings_SetVoice

[NX1]

设定 AnimalSings_Play / AnimalSings_PlayS 时使用的动物原音音档。

Animalsings_SetVoice(Index)

Animalsings_SetVoice(Index, Storage)

Index：动物音文件索引。

Storage：播放的动物音档所在的储存空间，若不指定则为内部空间，可使用 udr、spi0、spi1。

例.

[Path]

TR1: Animalsings_SetVoice(2, spi0) ; 设定动物唱歌音档，使用 SPI0 的索引 2 音檔。

```
TR2: Animalsings_Play                ; 使用 Animalsings_Play 播放录音结果。
```

5.41.8 Animalsings_LongSound_On

[NX1]

设定长动物音模式，会略过静音段的播放。

例.

```
[Path]
TR1: Animalsings_SetVoice(2, spi0)   ; 设定动物变音音档，使用 SPI0 的索引 2 音檔。
TR2: Animalsings_LongSound_On        ; 设定长音音文件模式。
TR3: Animalsings_Play                ; 使用 Animalsings_Play 播放录音结果。
```

5.41.9 Animalsings_LongSound_Off

[NX1]

关闭长动物音模式。

例.

```
[Path]
TR1: Animalsings_SetVoice(2, spi0)   ; 设定动物变音音档，使用 SPI0 的索引 2 音檔。
TR2: Animalsings_LongSound_Off       ; 关闭长音音文件模式。
TR3: Animalsings_Play                ; 使用 Animalsings_Play 播放录音结果。
```

5.41.10 Animalsings_NC_On

[NX1]

启动动物音降噪功能。

例.

```
[Path]
TR1: Animalsings_NC_On               ; 设定动物音降噪功能。
```

5.41.11 Animalsings_NC_Off

[NX1]

关闭动物音降噪功能。

例.

```
[Path]
TR1: Animalsings_NC_Off              ; 关闭动物音降噪功能。
```

5.41.12 Animalsings_NC_Auto

[NX1]

设定动物音自动判断降噪功能。

例.

[Path]

TR1: Animalsings_NC_Auto **；设定动物音自动判断降噪功能。**

5.42 I/O 扩展芯片指令（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]

用于改变现在I/O扩展芯片上的按键对应的型态。使用者可以在 **[Input State Exp0] / [Input State Exp1] / [Input State Exp2] / [Input State Exp3]** 中设定不同的 Input State，藉以控制不同按键状态的需求。

例.

[Input State Exp0]

IoExp0_KEY1: TR1 TR2 X /TR4

[Path]

PowerOn: IoExp0_KEY1 **；I/O 扩展芯片的按键设置成 IoExp0_KEY1 状态。**

5.42.2 IoExp Output State

[NY5+ / NX1]

用于改变现在 I/O 扩展芯片输出脚位的状态。使用者可以在 **[Output State Exp0] / [Output State Exp1] / [Output State Exp2] / [Output State Exp3]** 中设定不同的 Output State，藉以控制各输出脚位的状态。

例.

[Output State Exp0]

IoExp0_Output_0: 0 1 0 1

[Path]

PowerOn: IoExp0_Output_0 ; I/O 扩展芯片输出脚位设置成 IoExp0_Output_0 状态。

5.42.3 IoExp_Key_CLR

[NY5+ / NX1]

用于清除当前 I/O 扩展芯片的按键状态，当按键状态被清除后，I/O 扩展芯片会重新扫描按键。

IoExp_Key_CLR

IoExp_Key_CLR(ExpId)

ExpId：I/O 扩展芯片 ID。

5.42.4 IoExp_Key_ON

[NY5+ / NX1]

开启 I/O 扩展芯片按键扫描功能。若是开启该功能，则 I/O 扩展芯片会持续做扫描按键。

IoExp_Key_ON

IoExp_Key_ON(ExpId)

ExpId：I/O 扩展芯片 ID。

注意：系统默认值为 IoExp_Key_ON，PowerOn 后无须另外再下达一次 IoExp_Key_ON 指令。

5.42.5 IoExp_Key_OFF

[NY5+ / NX1]

关闭 I/O 扩展芯片按键扫描功能。若是关闭该功能，则 I/O 扩展芯片不做按键扫描。

IoExp_Key_OFF

IoExp_Key_Off(ExpId)

ExpId：I/O 扩展芯片 ID。

5.42.6 IoExp_Sleep

[NY5+ / NX1]

讓所有的 I/O 扩展芯片進入休眠。

注意：唤醒后须等待 1ms 才能对 I/O 扩展芯片下达指令。

5.42.7 IoExp_ErrId

[NY5+ / NX1]

當與 I/O 扩展芯片通訊失敗時，可使用此指令取得 I/O 扩展芯片 ID。

Var = IoExp_ErrId

Var：取得的 I/O 扩展芯片 ID。

5.42.8 IoExp_ReadInfo

[NY5+ / NX1]

讀取 I/O 扩展芯片的資訊。

IoExp_ReadInfo(ExpId, Body, Func, FwVer)

ExpId：I/O 扩展芯片 ID。
Body：I/O 扩展芯片的 Body 代码。
Func：I/O 扩展芯片的功能代码。
FwVer：I/O 扩展芯片的韧体版本。

5.42.9 IoExp_ReadSN

[NY5+ / NX1]

每次针对 I/O 扩展芯片执行指令时，都会增加 SN 的计数，此指令用来读取 I/O 扩展芯片的 SN。

IoExp_ReadSN(ExpId, SN)

ExpID：I/O 扩展芯片 ID。
SN：I/O 扩展芯片的 SN。

5.42.10 IoExp_SetInputPullHigh

[NY5+ / NX1]

将 I/O 扩展芯片上的脚位设定为带上拉电阻的输入模式。

IoExp_SetInputPullHigh(Port, Value)

Port：I/O 扩展芯片上的 Port，例如：Exp0_P0 为编号 0 的 I/O 扩展芯片上的 P0。
Value：可为数字或变量，要设定脚位为带上拉电阻的输入模式，需将此参数的对应 bit 设为 1。

5.42.11 IoExp_SetInputPullLow

[NY5+ / NX1]

将 I/O 扩展芯片上的脚位设定为带下拉电阻的输入模式。

IoExp_SetInputPullLow(Port, Value)

Port：I/O 扩展芯片上的 Port，例如：Exp0_P0 为编号 0 的 I/O 扩展芯片上的 P0。

Value：可为数字或变量，要设定脚位为带下拉电阻的输入模式，需将此参数的对应 bit 设为 1。

5.42.12 IoExp_SetInputFloating

[NY5+ / NX1]

将 I/O 扩展芯片上的脚位设定为不带上拉电阻的输入模式。

IoExp_SetInputFloating(Port, Value)

Port：I/O 扩展芯片上的 Port，例如：Exp0_P0 为编号 0 的 I/O 扩展芯片上的 P0。
Value：可为数字或变量，要设定脚位为不带上拉电阻的输入模式，需将此参数的对应 bit 设为 1。

5.42.13 IoExp_SetOutputHigh

[NY5+ / NX1]

将 I/O 扩展芯片上的脚位设定为输出 High。

IoExp_SetOutputHigh(Port, Value)

Port：I/O 扩展芯片上的 Port，例如：Exp0_P0 为编号 0 的 I/O 扩展芯片上的 P0。
Value：可为数字或变量，要设定脚位为输出 High，需将此参数的对应 bit 设为 1。

5.42.14 IoExp_SetOutputLow

[NY5+ / NX1]

将 I/O 扩展芯片上的脚位设定为输出 Low。

IoExp_SetOutputLow(Port, Value)

Port：I/O 扩展芯片上的 Port，例如：Exp0_P0 为编号 0 的 I/O 扩展芯片上的 P0。
Value：可为数字或变量，要设定脚位为输出 Low，需将此参数的对应 bit 设为 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]

用于改变现在按键对应的型态。用户可以在 **[Input State]** 中设定不同的 Input State，藉以控制不同按键状态的需求。

例.

```
[Input State]
KEY1: TR1   TR2   X   / TR4

[Path]
PowerOn: KEY1                    ; Input State 设置成 KEY1 状态。
```

5.43.2 Output State

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

用于改变现在输出脚位的状态。使用者可以在 **[Output State]** 中设定不同的 Output State，藉以控制各输出脚位的状态。

例.

```
[Output State]
Output_0: 0   1   0   1

[Path]
PowerOn: Output_0                ; Output State 设置成 Output_0 状态。
```

5.43.3 Action Mark State

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

Q-Visio 编辑文件时，可以在 action 文件中插入 Mark 标记。当 *Q-Code* 读到标记时，会依照当前的 Action Mark 状态自动执行对应的路径。用户可以在[Action Mark]段落中定义 Action Mark 的状态。

Action Mark 的标记编号最多可有 255 个，从 A1 至 A255。在 *Q-Code* 中，用户仅需要加入.vio 文件及定义好标记对应的路径即可。最多可以定义 255 组设定。

用户在播放 Action 文件时，可以用 *Q-Visio* 插入 Mark 来标记 Action Mark。标记可由 A1 至 A255。当播放 Action 档时，可以藉由 Action Mark State 来切换目前要对应的 Action Mark 的路径。最多可定义 255

个 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                          ; 当读取到 Action Mark 编号 1，即会执行 AM1。
AM2: PB=0x2                          ; 当读取到 Action Mark 编号 2，即会执行 AM2。
AM3: PB=0x3                          ; 当读取到 Action Mark 编号 3，即会执行 AM3。
AM4: PB=0x0                          ; 当读取到 Action Mark 编号 4，即会执行 AM4。
```

5.43.4 Wave Mark State

[NY4 / NY5 / NY5+ / NX1]

用户在播放 Voice 文件时，可以由 *Quick-IO* 来插入 Mark 来标记 Wave Mark。标记最多可有[M01]至[M15]。当播放 Voice 档时，可以藉由 WaveMark State 来切换目前要对应的 WaveMark 的路径。最多可定义 255 个 WaveMark State。

```
[WaveMark]
;                M1      M2      M3      M4
WaveMark1:       WM1     WM2     WM3     WM4

[Path]
PowerOn: WaveMark1, PlayV($V0)

[Background1]
WM1: PB=0x1                          ; 当 QIO 发生，且 Mark 编号为 1，即会执行 WM1。
WM2: PB=0x2                          ; 当 QIO 发生，且 Mark 编号为 2，即会执行 WM2。
WM3: PB=0x3                          ; 当 QIO 发生，且 Mark 编号为 3，即会执行 WM3。
WM4: PB=0x0                          ; 当 QIO 发生，且 Mark 编号为 4，即会执行 WM4。
```

5.43.5 MelodyMark State

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

用户在播放 Melody 文件时，可以由 *Cakewalk* 或 *Q-MIDI* 来插入 Mark 来标记 Melody Mark。Melody Mark 标记最多可有 M1 至 M255。当播放 Melody 时，可以藉由 Melody Mark State 来切换目前要对应的 Melody Mark 的路径。最多可定义 255 个 Melody Mark State。

```
[MelodyMark]
;                M1      M2      M3      M4
```

```
MelodyMark1:   MM1      MM2      MM3      MM4

[Path]
PowerOn: MelodyMark1, PlayM($M0)

[Background1]
MM1: PB=0x1                    ; 当读取到 Melody Mark 时，且 Mark 编号为 1，即会执行 MM1。
MM2: PB=0x2                    ; 当读取到 Melody Mark 时，且 Mark 编号为 2，即会执行 MM2。
MM3: PB=0x3                    ; 当读取到 Melody Mark 时，且 Mark 编号为 3，即会执行 MM3。
MM4: PB=0x0                    ; 当读取到 Melody Mark 时，且 Mark 编号为 4，即会执行 MM4。
```

5.43.6 NoteOn State

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

用户在播放 Melody 文件时，若不使用 Cakewalk 或 *Q-MIDI* 来插码。可以通过 **[Note On]** 来执行每个音符被读取时，会执行的动作。藉由 NoteOn State 来切换目前要对应的路径，最多可定义 255 个 NoteOn State。

- NY5 针对 Ch0 ~ Ch3。
- NY5+针对 Ch1 ~ Ch4、Ch10。
- NY6 针对 MIDI 通道 Ch1 ~ Ch6、Ch10。
- NY7 针对 MIDI 通道 Ch1 ~ Ch16。
- NX1 针对 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    ; 切换 Note On 状态。
NO1: CH1, R0=1                           ; 设定 Note On State=CH1。
NO2: CH2, R0=0                           ; 设定 Note On State=CH2。
CH1_ON: BG1                              ; 执行 CH1 的 Note On 时，调用 BG1。
CH2_ON: BG2                              ; 执行 CH2 的 Note On 时，调用 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+]

当使用者使用 Q-Visio 编辑文件时，可以随意在 action 文件中插入 Mark 标记。当 Q-Code 使用 PlayPWM 播放时读到标记，会依照当前的 PWM-IO Mark 状态自动执行对应的路径。使用者可以在 **[PWM-IO Mark]** 段落中定义 PMW-IO Mark 的状态。

PWM-IO Mark 的标记编号最多可有 255 个，从 A1 至 A255。最多可以定义 255 组设定。

例. 在 Q-Visio 中，插入 A1~A4 的码。

```
[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                    ; 当读取到 PWM-IO Mark，且 Mark 编号为 1，即会执行 PM1。
PM2: PB=0x2                    ; 当读取到 PWM-IO Mark，且 Mark 编号为 2，即会执行 PM2。
PM3: PB=0x3                    ; 当读取到 PWM-IO Mark，且 Mark 编号为 3，即会执行 PM3。
PM4: PB=0x0                    ; 当读取到 PWM-IO Mark，且 Mark 编号为 4，即会执行 PM4。
```

5.43.8 QFID State

[NY5+ / NY7 / NX1]

用于改变现在 QFID 反应的状态。用户可以在 **[QFID]** 中设定不同的 QFID State，藉以控制不同 QFID Tag 侦测的需求。

例.

```
[QFID]
QFIDGroup0_0: TR1   TR2   X /TR4
[Path]
PowerOn: QFIDGroup0_0                ; QFID State 设置成 QFIDGroup0_0 状态。
```

5.43.9 Key_CLR

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

用于清除当前按键状态，当按键状态被清除后，按键会重新扫描。

例.

```
[Input_State]
```

KEY1：TR1　TR2

[Path]

PowerOn: KEY1　　; Input State 设置成 KEY1 状态。

TR1: PlayV(Ch0, $V0), KEY_CLR

; 按下 TR1 后，执行 PlayV(Ch0, $V0)。执行完 PlayV(Ch0, $V0)后，执行 Key_CLR 指令，即清除当前的按键的状态，重新扫描按键状态。当 V0 播放完了之后，如果 TR1 有被按住时，程序会继续执行 PlayV(ch0, $V0)，如果 TR1 没有被按住时，则程序进入 Sleep 状态。

5.43.10 Key_ON

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

打开按键扫描功能。若是打开该功能，则按键会持续做扫描。

注意：系统默认值为 Key_ON，PowerOn 后无须另外再下达一次 Key_ON 指令。

5.43.11 Key_OFF

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

关闭按键扫描功能。若是关闭该功能，则按键不做扫描。

例.

[Input_State]

KEY1: TR1

[Path]

PowerOn: KEY_OFF, DELAY(0.3), KEY1, KEY_ON

; PowerOn 时，将按键扫描功能关闭，待延迟 0.3 秒后，才打开按键扫描。

5.43.12 Stop

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

停止当前所有动作包 PlayV、PlayM、Delay、PlayA、PlayPWM、PWMOut 及所有背景动作。

注意：NY5 中使用 Stop 指令后，中断亦会被关闭。

例.

[Path]

PowerOn: PlayV(ch0,$V0) & [BG1, BG2]

TR1: Stop　　; 停止全部的播放，Delay 及背景动作。

[Background1]

BG1: …..

[Background2]

BG2: …..

5.43.13 Pause(n)

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

可以暂停指定的前景、背景 1、背景 2 或背景 3 的所有动作，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，则暂停所有的执行。

例.

```
[Path]
TR1: Pause(1)                          ; 暂停背景 1 的所有动作。
[Background1]
BG1: PlayV(Ch0, $V0), Delay(5)         ; BG1 执行 PlayV(Ch0, $V0)及 5 秒时间延迟。
```

5.43.14 Resume(n)

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

可以恢复指定的前景、背景 1、背景 2 或背景 3 的所有动作，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，则恢复所有的执行。

例.

```
[Path]
TR1: Pause(1)                          ; 暂停背景 1 的所有动作。
TR2: Resume(1)                         ; 恢复背景 1 的所有动作。
[Background1]
BG1: PlayV(ch0, $V0), Delay(5)         ; BG1 PlayV(Ch0, $V0)及 5 秒时间延迟。
```

5.43.15 ReadChannel(Rj:Ri) / ReadChannel(Xi)

[NY5 / NY5+ / NY6 / NY7]

将目前的通道的使用状况，存放到 RAM。

例.

```
[Path]
TR1: PlayV(CH2, $V0)                   ; 播放 Voice。
TR2: ReadChannel(X0), [PC,PB]=X0       ; 将 Channel 的使用状况输出到 PC 与 PB。
```

5.43.16 PauseDown

[NY5]

在执行完 PauseDown 指令后，会立即做 Ramp Down 动作。执行完 PauseDown 指令后，仅能使用 ResumeUp 指令来恢复播放。（仅对 Channel2 有效）

注意：

1. ***PauseDown 指令仅对于 Channel2 有效，仅有 Channel0 / Channel1 播放的话无法暂停播放。若下达 PauseDown 指令时，Channel0 / Channel1 正在播放的话会一并暂停。***
2. ***使用 PauseDown 指令，可以进入 Sleep。***

例.

[Path]

```
TR1: PlayV(ch2, $V0)                ; 播放 Voice。
TR2: PauseDown                      ; 暂停播放，且进入 IC 会进入 Sleep。
TR3: ResumeUp                       ; 恢复播放。
```

5.43.17 ResumeUp

[NY5]

恢复 PauseDown 指令所暂停的播放动作。ResumeUp 指令仅能恢复使用 PauseDown 指令所暂停的播放。

例.

[Path]

```
TR1: PlayV(ch2, $V0)                ; 播放 Voice。
TR2: PauseDown                      ; 暂停播放，且 IC 会进入 Sleep。
TR3: ResumeUp                       ; 恢复播放。
```

5.43.18 Audio_Loop_On

[NX1]

Audio_Loop_On 指令用于开启指定语音通道循环播放功能。

Audio_Loop_On(Ch)

Ch：指定要开启循环播放的语音通道。

- NX1 支持的语音通道为 Ch0 ~ Ch7。

5.43.19 Audio_Loop_Off

[NX1]

Audio_Loop_Off 指令用于关闭指定语音通道循环播放功能。

Audio_Loop_Off(Ch)

Ch：指定要关闭循环播放的语音通道。

- NX1 支持的语音通道为 Ch0 ~ Ch7。

5.43.20 Audio_ON

[NY5 / NY5+ / NY6 / NY7]

打开 Audio 输出。

注意：

1. 系统默认值为 Audio_ON，PowerOn 后无须另外再下达一次 Audio_ON 指令。

2. 若使用 DAC 或 Push-Pull 输出，则需注意在恢复 Audio 输出时，可能会有“bo”声。

5.43.21 Audio_OFF

[NY5 / NY5+ / NY6 / NY7]

关闭 Audio 输出。

注意：

1. 若将为 Audio 设置为 OFF，将会一直维持直到再下达一次 Audio_ON 指令。播放音档才会有声音输出。

2. 若使用 DAC 或 Push-Pull 输出，则需注意在关闭 Audio 输出时，可能会有“ bo”声。

3. 当使用 PWM / DAC 输出时，调用 Audio_OFF 会使 PWM / DAC 输出脚位成为 floating 状态。

例.

```
[Path]
TR1: PlayV(Ch0,$V0)                ; 播放 Voice。
TR2: Audio_ON                      ; 打开声音输出。
TR3: Audio_OFF                     ; 关闭声音输出。
```

5.43.22 AudioMode=n

[NY5+ / NY6 / NY7]

选择声音输出的模式。

n：声音输出的模式。支持 DAC / PWM / PP。

- NY5+仅能提供 DAC 与 PWM 输出。
- NY6A 仅提供 PWM 输出。
- NY6B / NY6C 提供 DAC 与 PWM 输出。
- NY7A 提供 DAC 与 PWM 输出。
- NY7B / NY7C 提供 DAC 与 Push-Pull 输出。

例. NY7A

```
[Path]
TR1: AudioMode=DAC                 ; 设定成 DAC 输出。
TR2: AudioMode=PWM                 ; 设定成 PWM 输出。
TR3: PlayV(Ch0, $V0)               ; 播放音源文件。
```

例. NY7B / NY7C

```
[Path]
```

```
TR1: AudioMode=DAC                  ; 设定成 DAC 输出。
TR2: AudioMode=PP                   ; 设定成 Push-Pull 输出。
TR3: PlayV(Ch0, $V0)                ; 播放音源文件。
```

5.43.23 NoiseFilter_ON(Ch)

[NY6 / NY7 / NX1]

打开噪声滤波功能。

NoiseFilter_On

NoiseFilter_On(Ch)

Ch：0 ~ 5 或 Ch0 ~ Ch5，若不指定则打开所有语音通道的噪声滤波功能。

- NY6 支持 0 ~ 5 或 Ch0 ~ Ch5。
- NY7 / NX1 不支持指定语音通道。

注意：

1. ***该功能一经打开后，直到关闭前会一直维持打开的状态。***
2. ***此功能不允许在声音播放中使用。***
3. ***默认为打开。***

5.43.24 NoiseFilter_OFF(Ch)

[NY6 / NY7 / NX1]

关闭噪声滤波功能。

NoiseFilter_Off

NoiseFilter_Off(Ch)

Ch：0 ~ 5 或 Ch0 ~ Ch5，若不指定则打开所有语音通道的噪声滤波功能。

- NY6 支持 0 ~ 5 或 Ch0 ~ Ch5。
- NY7 / NX1 不支持指定语音通道。

注意：

1. ***该功能一经打开后，直到关闭前会一直维持打开的状态。***
2. ***此功能不允许在声音播放中使用。***

例. 若语音听起来有背景噪音时，可以藉由打开 Noise Filter 来进行降噪。

[Path]

```
TR1: PlayV(Ch0,$V0)                 ; 播放 Voice。
TR2: NoiseFilter_ON                 ; 打开 Noise Filter。
TR3: NoiseFilter_OFF                ; 关闭 Noise Filter。
```

5.43.25 EQ_Filter

[NX1]

开启音频滤波器滤波功能并指定滤波器参数。

EQ_Filter(Ch, Index)

Ch：语音通道。

- NX1 支援 Ch0 ~ Ch7。

Index：音频滤波器参数索引。

- 0 ~ 滤波器索引最大值。

注意：

1. 请搭配音频滤波器 (Audio Filter) 进行参数设置。

2. 此功能仅支持 SBC1 / ADPCM 使用。

例. 若希望语音使用音频滤波器，可以藉由设定 EQ_Filter 来实现。

```
[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 的滤波器参数。
```

5.43.26 EQ_Filter_Off

[NX1]

关闭音频滤波器的滤波功能。

EQ_Filter_Off(Ch)

Ch：语音通道。

- NX1 支援 Ch0 ~ Ch7。

注意：必须有使用 EQ_Filter 方可使用 EQ_Filter_Off。

5.43.27 AGC_On

[NX1]

打开 AGC 功能。在 *Q-Code* 使用 VR 或 Record 指令时，AGC (Auto Gain Control)可自动调节麦克风增益，避免音量过大造成信号饱和的情形，可提升 VR 辨识率与减少 Record 杂音的状况。

注意：若项目包含 VR，且 IC Body 不为 NX12P44，预设为打开。

5.43.28 AGC_Off

[NX1]

关闭 AGC 功能。在 *Q-Code* 使用 VR 或 Record 指令时，AGC(Auto Gain Control)可自动调节麦克风增益，避免音量过大造成信号饱和的情形，可提升 VR 辨识率与减少 Record 杂音的状况。

注意：若项目包含 VR，且 IC Body 不为 NX12P44，预设为打开。

例. 若录音在大音量时听起来有杂音，可藉由打开 AGC 降低杂音发生。

[Path]

```
TR1: Record($Rec0,5s)                ; 开始录音
TR2: PlayV(CH0,$Rec0)                ; 播放录音内容
TR3: AGC_On                          ; 打开 AGC
TR4: AGC_Off                         ; 关闭 AGC
```

5.43.29 RampUp

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

在 *Q-Code* 要播放 Voice 文件或是 Melody 文件时，系统会自动帮 User 做 RampUp 的动作。但若是 User 有特殊的需求，而必须先做 RampUp 这个动作，在这边我们也提供了 RampUp 的指令以供 User 来使用。

注意：

1. ***若已经执行过 RampUp 指令后，则在 IC 未进入睡眠前或执行 RampDn 指令前，系统不会再执行 RampUp 动作。***
2. ***NY5 / NY5+ / NY6 / NY7 在延迟过程中无法处理其它的动作，仅能在延迟结束后执行下一个步骤。***
3. ***NX1 在执行 RampUp 后，会立即执行下一道指令，RampUp 运行时间由选项 Ramp Up Time 决定。***

例.

```
RampUp, Delay(1), PlayV…..
; 执行过 RampUp 后，PlayV / PlayM 不会再执行 RampUp 动作。
```

例. 有 OP Amp 应用时，且有控制脚位。

[Path]

```
TR1: RampUp, PB.3=0, PlayV…..
; 执行完 RampUp 后，将 OP Amp 致能。播放音源文件时就不会有“bo”声。
```

5.43.30 RampDown

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

在 *Q-Code* 系统中，当声音或者是 Melody 档播放完毕后会直接帮 User 做 RampDown 的动作。但若是 User 有特殊的需求，仍可使用指令来达成。

注意：

1. ***若是在进 Sleep 前，会有一段延迟时间的话，则会造成耗电或者是有杂音的问题。在这边我们提供了 RampDown 指令让 User 可以自行关闭 Audio Output，以解决耗电及杂音的问题。***
2. ***NY5 / NY5+ / NY6 / NY7 在延迟过程中无法处理其它的动作，仅能在延迟结束后执行下一个步骤。***
3. ***NX1 在执行 RampDown 后，会立即执行下一道指令，RampDown 运行时间由选项 Ramp Down Time 决定。***

例.

```
PlayV….. , RampDown, Delay(15)
; 播放完毕后仍会延迟 15 秒才会进入睡眠模式，此时必须要先做 RampDn 动作，来关闭声音输出以避免“bo”声及耗电。
```

5.43.31 AutoSleep_On

[NX1]

Q-Code 在 IC 闲置时会自动进入睡眠模式。

注意：系统默认值为 AutoSleep_ON，PowerOn 后无须另外再下达一次 AutoSleep_ON 指令。

5.43.32 AutoSleep_Off

[NX1]

Q-Code 在 IC 闲置时不进入睡眠模式。

注意：若将 AutoSleep 设置为 OFF，IC 将不会进入睡眠，直到再下达一次 AutoSleep_ON 指令。

5.43.33 Sleep

[NX1]

设定当系统不动作时，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）是为了要清除 Watch Dog Counter 用。若是在 *Q-Code* 里面执行过多的算数逻辑指令或是其它指令，而其间都没有执行过 Play 或者是延迟命令，则系统可能会由于运行时间过长，会将 IC 当成是异常执行，因而 Reset IC。这时就可以通过 WDT_CLR 指令来强制 Reset Counter。

例. **WDT_CLR ; Watch Dog Clear。**

5.43.35 Repeat

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

提供 repeat function 让 user 可以藉由 repeat function 来达到省 Code size 的目的。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 总共提供 3 组，分别对于前景及两个背景都可独立使用而不互相干扰。
- NX1 总共提供 4 组，分别对于前景及三个背景都可独立使用而不互相干扰。

{ … } *n

n: 重复次数。

- NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T 支持 1 ~ 15。
- NX1 支持 1 ~ 255。

例. 单一个 section repeat 时，利用 LOOP counter 来达到 repeat 功能。

```
{ PlayV($V0), PlayV($V1), PlayV($v2) } * 3    ; "{"…. "}" 号内的程序会执行 3 次。
```

5.43.36 Background

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

执行指令时一并启动背景路径。

… & [BG1]
… & [BG1, BG2]
… & [BG1, BG2, BG3]
… & [BG1, BG2, BG3, BG4]
… & [BG1, BG2, BG3, BG4, BG5]

BG1：呼叫 Background1 的程序。

- BG1 可以呼叫 [**Background1**] 底下的动作。
- X：不改变当前 Background1 path 的动作。
- OFF：停止当前 Background1 path 的动作。

BG2：呼叫 Background2 的程序。

- BG2 可以呼叫 [**Background2**] 底下的动作。
- X：不改变当前 Background2 path 的动作。
- OFF：停止当前 Background2 path 的动作。

BG3：呼叫 Background3 的程序（NX1 Only）。

- BG3 可以呼叫 [**Background3**] 底下的动作。
- X：不改变当前 Background3 path 的动作。
- OFF：停止当前 Background3 path 的动作。

BG4：呼叫 Background4 的程序（NX1 Only）。

- BG4 可以呼叫 [**Background4**] 底下的动作。
- X：不改变当前 Background4 path 的动作。
- OFF：停止当前 Background4 path 的动作。

BG5：呼叫 Background5 的程序（NX1 Only）。

- BG5 可以呼叫 [**Background5**] 底下的动作。
- X：不改变当前 Background5 path 的动作。
- OFF：停止当前 Background5 path 的动作。

例.

```
PlayV($V0) & [BG1, BG2]                    ; 播放 V0 时一并执行 BG1 及 BG2。
```

5.43.37 Slow

[NY5+ / NY6 / NY7]

切换 Sleep 模式为 Slow 模式，此模式可用于长时间的计数或是需要固定时间检查信号的应用。当用户执行此指令后，原本的 Sleep 模式将会由 slow 模式来取代，系统固定时间会自动唤醒一次，唤醒后会执行 WakeUp 路径一次，而在进入 Slow 模式前也会执行 Sleep 路径一次。用户可使用 SlowOff 指令来关闭 Slow 模式，恢复成 sleep 模式。Slow 模式唤醒的时间为中断时间的 16 倍，即中断 256us 会在 4ms 唤醒一次，中断 1ms 会在 16ms 唤醒一次。

例. 低速模式切换。

[Path]

```
TR1: Slow                                   ; 切换成低速模式。
```

5.43.38 SlowOff

[NY5+ / NY6 / NY7]

用来关闭 Slow 模式，恢复成 Sleep 模式。

例. 关闭低速模式。

[Path]

```
TR1: SlowOff                                ; 关闭低速模式。
```

5.43.39 END

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

在此的 END 指令并非强制停止目前所有正在执行的动作。若碰到 END 指令后，后面的指令将不会被执行。当所有的程序全部皆执行完毕的话，系统才会允许进入睡眠模式。若是用户需要完全停止目前正在执行的所有动作，并且进入睡眠模式，可以使用 Stop 指令来停止有目前正在执行的动作。

注意：系统编译完后会自动在每行结束加上“END”指令，故可以不需要在每行结束时下达“END”指令。

例. **PlayV….. , RampDown, Delay(15), END**

5.43.40 ReadRollingCode

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

滚动码（Rolling Code）为 20-bit 流水号，每次在写入 OTP 时会自动增加，可赋予每一个 OTP 独特的 ID。使用此指令可取得写入时赋予的流水号，并存放到寄存器。寄存器存放的内容为：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) ；将读出的 rolling code 存到 R0~R4。

例. NX1

[Variable]

Var32: Code

[Path]

TR1: ReadRollingCode(Code) ；将读出的 rolling code 存到 Code。

5.43.41 ChMode(n)

[NY6 / NY7]

设定通道数量。

n：通道数量。

- NY6 支持 2 / 4 / 6（预设为 6）
- NY7 支持 2 / 4 / 6 / 8。（预设为 8）

注意：

1. 当语音或 melody 播放中，切换通道数会造成采样率错误，因而播放异常。请避免在播放中切换通道数。

2. NY6 使用 6 通道，语音采样率最高仅支持 41.6KHz，其他通道数皆可达 44.1KHz。

例.

ChMode(4), PlayV(Ch1,$V1) ；设定使用 4 通道来播放 V1。

音质	可用通道编号
高质量	0 ~ 1
优良	0 ~ 3
中等	0 ~ 5
一般	0 ~ 7

5.43.42 ReadRadj(Ri)

[NY9T]

读取外部电阻值，指令会回传 0x0~0x7 数值，若没有接外部电阻值时，则回传 0xF，此读取行为并不会直接影响触摸键的灵敏度，用户可以根据读取的数值，自行定义如何对应到 8 阶的灵敏度设定。

建议电阻值（±1%）	Level	Ri
750K	0	0x0
360K	1	0x1
180K	2	0x2
120K	3	0x3
82K	4	0x4
51K	5	0x5
33K	6	0x6

建议电阻值（±1%）	Level	Ri
16K	7	0x7
Other	-	0xF

注意：若不使用外部电阻调整灵敏度功能时，需将 Radj 脚位接到 VDD。

例.

[Path]

PowerOn: ReadRadj(R0), Switch(R0)=[Sens0, Sens1, Sens2, Sens3]　；将外部电阻的阶数读出。

Sens0: TouchKey_Sensitivity(0)　　；设定灵敏度最高。

Sens1: TouchKey_Sensitivity(2)　　；设定灵敏度介于一般和最高之间。

Sens2: TouchKey_Sensitivity(4)　　；设定灵敏度一般。

Sens3: TouchKey_Sensitivity(7)　　；设定灵敏度最低。

5.43.43 SW_Reset

[NY9T]

SW_Reset（Software Reset）可恢复成上电时的初始状态。与硬件 Reset 不同处在于 IC 不会重新读取硬件的初始设定，仅是重置 RAM 及 IO 状态。该指令被执行后，程序会回到启始进入点，SW_Reset 后面的指令将不会被执行。

注意：在执行 Enforce_Calibrate 后，若想要重置整个系统可利用 SW_Reset 指令或是在 Option 中选用 SW_Reset = Enable。

例. **Enforce_Calibrate: SW_Reset　　；Software Reset。**

5.43.44 Debounce(Time)

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

Debounce 指令提供给用户一种方法同时改变 Direct 及 Matrix 按键扫描的 Debounce 时间。Time 单位：ms（毫秒）或 sec（秒）（默认为秒），最小值=0ms，最大值=1s。

注意：

1. *Debounce(Time)指令会同时影响 Direct 及 Matrix 按键。*
2. *NY9T 中，Debounce option 与 Debounce(Time)指令，只能择一使用，且时间须为 4ms 的倍数。*

例.

[Path]

PowerOn: KEY1, KEY_ON

TR1: Debounce(16ms)　　；设定 Direct 及 Matrix 按键的 Debounce 时间为 16ms。

TR2: Debounce(0.06)　　；设定 Direct 及 Matrix 按键的 Debounce 时间为 60ms。

5.43.45 Direct_Debounce(Time)

[NY5+ / NY6 / NY7]

Direct_Debounce 指令提供给用户一种方法来改变 Direct 按键扫描的 Debounce 时间。Time 单位：ms（毫秒）或 sec（秒）（默认为秒），最小值=0ms，最大值=1s。

注意：Direct_Debounce(Time)指令仅对 Direct 按键有效，不影响 Matrix 按键。

例.

[Path]

PowerOn: KEY1, KEY_ON

TR1: Direct_Debounce(16ms) ; **设定 Direct 按键的 Debounce 时间为 16ms。**

TR2: Direct_Debounce(0.06) ; **设定 Direct 按键的 Debounce 时间为 60ms。**

5.43.46 Matrix_Debounce(Time)

[NY5+ / NY6 / NY7]

Matrix_Debounce 指令提供给用户一种方法来改变 Matrix 按键扫描的 Debounce 时间。Time 单位：ms（毫秒）或 sec（秒）（默认为秒），最小值=0ms，最大值=1s。

注意：Matrix_Debounce(Time)指令仅对 Matrix 按键有效，不影响 Direct 按键。

例.

[Path]

PowerOn: KEY1, KEY_ON

TR1: Matrix_Debounce(16ms) ; **设定 Matrix 按键的 Debounce 时间为 16ms。**

TR2: Matrix_Debounce(0.06) ; **设定 Matrix 按键的 Debounce 时间为 60ms。**

5.43.47 Ri = LVD

[NY5+ / NY6B / NY6C / NX1]

读取 LVD 的阶数，并将 LVD 的阶数存入指定的 RAM Ri。

阶数的对应如下：

阶数	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. 将 LVD 的阶数存入 R0。

TR1: R0 = LVD ; **将 LVD 的阶数存到 R0。**

5.43.48 Enforce_Wakeup

[NX1]

Enforce_Wakeup 指令提供给使用者于 Standby Mode 时，从 RTC_2Hz 路径唤醒系统的方法。

注意：Enforce_Wakeup 指令仅于 Standby Mode 条件下 RTC_2Hz 路径内有效。

例.

```
[Variable]
Var8:event_trig=0, count_500ms=0
[Path]
PowerOn:
main: event_trig=1?TR1                          ; 确认唤醒事件进行播音。
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 秒唤醒一次。
```

5.43.49 GetMicVol

[NX1]

GetMicVol 指令透过撷取麦克风 ADC 读值，进行估测音量信息提供。

GetMicVol(Var)

Var：将读取的结果存放于 Var，结果范围为 0~65535。

注意：

1. 仅于麦克风运行期间能取得数值，麦克风未运行时会得到 0。

2. 依当前使用环境与麦克风灵敏度等因素，会有 5%不等的底噪，此音量信息仅供粗略的趋势参考。

例.

```
[Variable]
Var16: var_mic=0
[Path]
PowerOn: RT_Play                                ; 透过 RT 启动麦克风。
8ms: GetMicVol(var_mic),                        ; 每 8ms 读取一次麦克风估测音量。
     var_mic>6000?{PA.1=0}:{PA.1=1}             ; 音量大于 6000 设置 PA.1=0，反之 PA.1=1。
```

5.43.50 Millis

[NX1]

Millis 指令透过读取 NX1 计数信息取得初始化后的运行时间 (毫秒)，数据长度为 Var32。

Millis(Var)

Var：将读取的结果存放于 Var。

注意：睡眠时将不会继续计数。

例.

```
[Variable]
Var32: millis_var
[Path]
PowerOn: Millis(millis_var)                    ; 取得运行时间。
```

5.43.51 Srand

Srand 会设定用于产生一系列随机数的起始点。

Srand(Seed)

Seed：随机数种子。

5.44 侦错指令（Debug Command）

Debug Command				
Printf	-	-	-	-

5.44.1 Printf

[NX1]

用于透过 UART TX 硬件输出除错信息，使用方式如 C 语言的 printf。

Printf(Format_str, Args)

Format_str：信息字符串。

Args：变量信息，数量可随需求增减，可不使用。

注意：

1. 需开启 UART TX 功能才可使用此指令。

2. Format_str 中支持的变量输出型别有%o、%d、%u、%x 以及%X。

3. Q-Code 不会检查 Format_str 中使用的变量的型别与数量，使用者需自行注意。

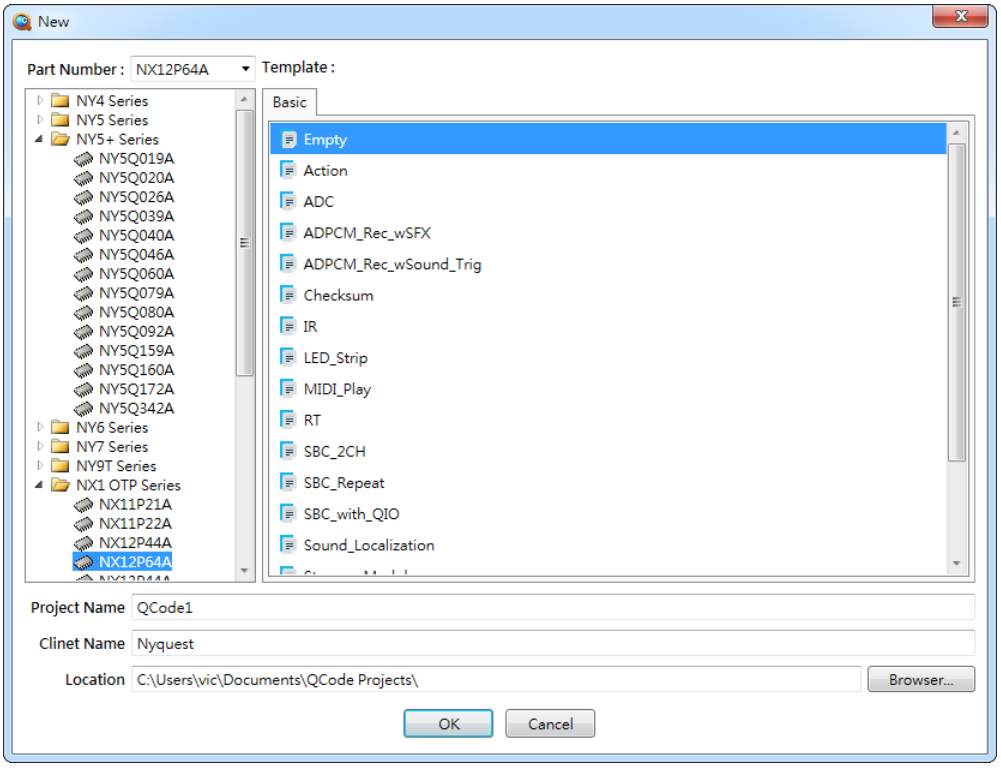
例.

```
[Variable]
Var32: cnt = 0
[Path]
PowerOn: Printf("Hello\r\n")                   ; 输出 Hello。
8ms: cnt++                                     ; 每 8ms 累加一次 cnt。
500ms: Printf("cnt%d\r\n", cnt)                ; 每 500ms 输出当前 cnt 的值。
```

6 附录

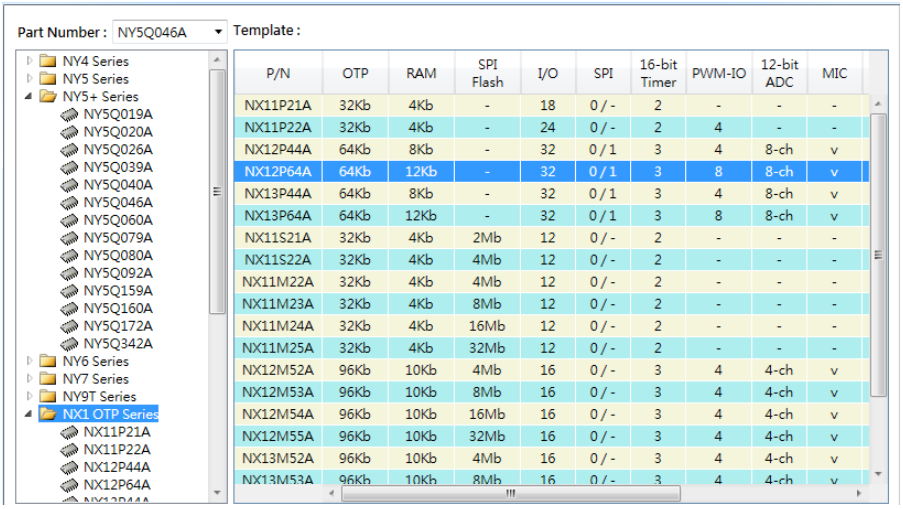
6.1 新建窗口

[文件] → [新建]



Part Number：选择 IC 的型号。

Template：选择要使用的功能样本，建立档案的内容。点击 IC 的型号，则右方显示样板选单如上图。若点击系列目录，则右方会显示系列产品信息如下图。



Project Name：输入项目名称。

Client Name：输入客户名称，以保护客户的权益。

Location：档案的目录。

6.2 相关工具介绍

6.2.1 相关软件工具

在使用 *Q-Code* 编写程序时，我们不仅仅只要用到这一个工具，我们还需要用到几个相关的工具。例如，当我们在编写完程序时，就需要编译产生一个.bin 和.htm 文件（这是投 Code 的重要文件），此时，我们就需要用到 *NYASM*，它应该在安装 *Q-Code* 时同步安装。

Q-MIDI：此软件是用来编辑 MIDI 与音色库，产生的文件为.md3，可在 NY5 / NY5+ / NY6 / NY7 / NX1 系列中使用。

Quick-I/O：此软件是用来画出输出脚的信号，需结合.wav 文件使用，所产生的文件为.nyq。

Q-Visio：此软件是用来画出输出脚的信号，所产生的文件为.vio。

Q-Touch：此软件是用来扫描触摸键的灵敏度，所产生的文件为.t9x。

Q-Writer：此软件是用来将.bin 文件烧录在 Flash Demo Board、Romter 或 OTP 上以供验证。

注意：在安装 Q-Code 时，最好是将用到的相关工具都同步安装，以上工具的使用请参考相关的用户手册。

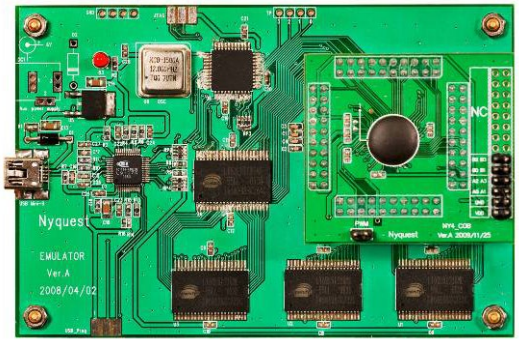
6.2.2 相关硬件工具

当程序编写完成并编译后，用户可以将程序下载至 ICE 上进行验证，或者通过烧录器（FDB_Writer 或 Q-FDB_Writer）烧录至 Demo 板上进行演示。

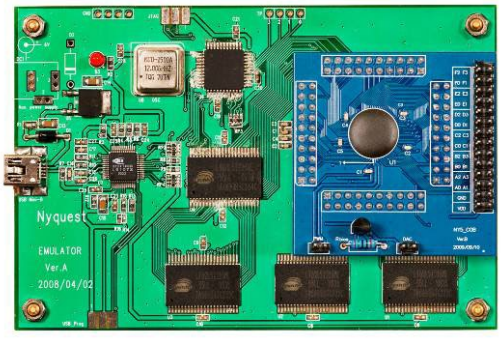
何谓 ICE? ICE 是 In Circuit Emulator 的缩写，中文叫实体仿真器或仿真器。只需要将 ICE 通过 USB 连接至个人计算机，便可以将编译好的程序下载至 ICE 进行验证。（有关 ICE 硬件安装请参考 *NYIDE* 用户手册。）

注意：

1. ***ICE 有分为 V1 / V2（下图中 NY4 / NY5 使用的是 ICE V1，NY7 则是 ICE V2）。***
2. ***ICE V1 仅支持 NY4 / NY5。***
3. ***ICE V2 支持 NY4 / NY5 / NY5+ / NY6 / NY7 / NY9T。***



NY4_COB on ICE V1



NY5_COB on ICE V1

NY7_COB on ICE V2

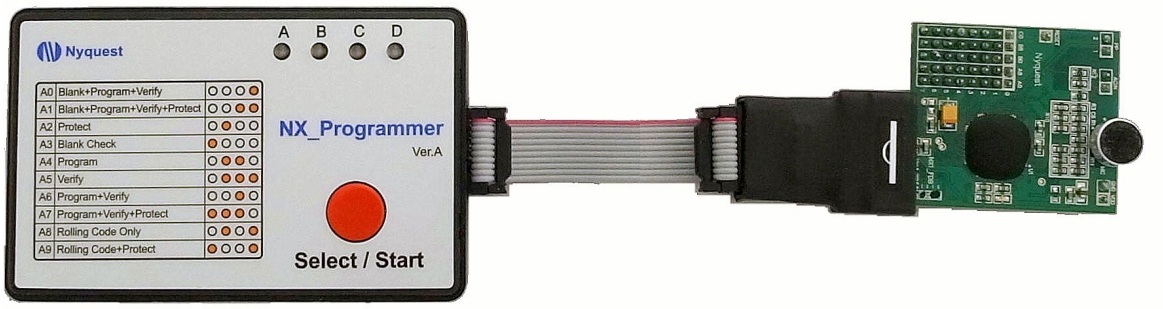
何谓 *Q-Writer*？ *Q-Writer* 是一个图形界面的烧录系统，让用户能够快速的将程序产生的.bin 文件烧录至 FDB（Flash Demo Board）演示板中，或者下载到 Romter 验证，也可直接通过 *Q-Writer* 直接烧录至 OTP 来进行验证。用户可以用 FDB_Writer 烧录，也可以用 *Q-Writer* 烧录（有关 *Q-Writer* 的软件和硬件安装及使用请参考 *Q-Writer* 用户手册）。

FDB_Writer：如下图所示。

Q-FDB_Writer：如下图所示。

Romter：如下图所示。

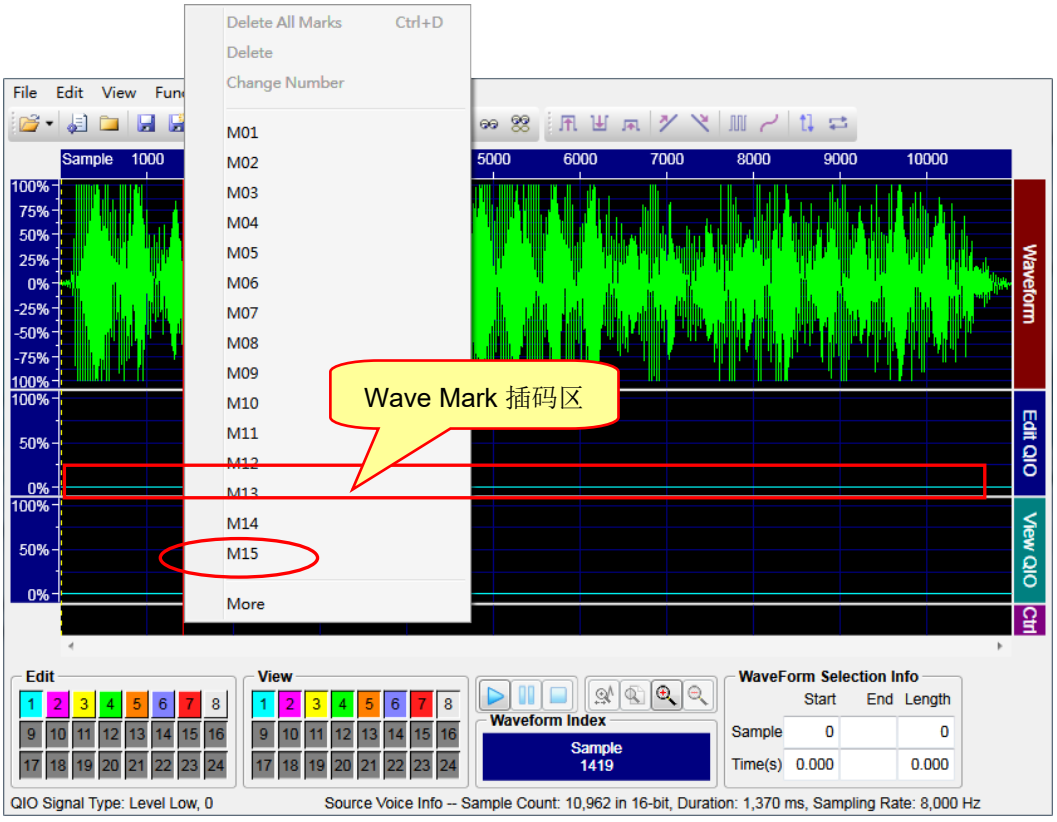
NX_Programmer：如下图所示。



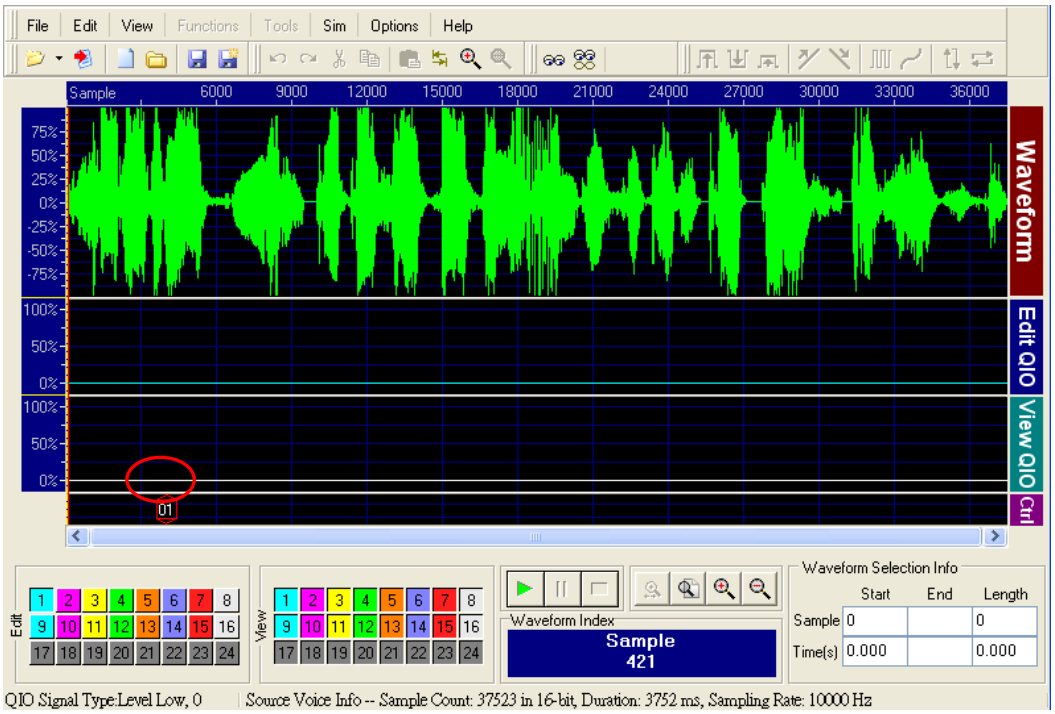
6.3 使用 *Quick-IO* 在.wav 文件中插入控制码

在 *Quick-IO* 中，有提供插入控制码的功能。让用户可以在任一个位置插入控制码，提供 *Q-Code* 判读。当 *Q-Code* 读取到 WaveMark 时，会执行 **[WaveMark]** section 中相对应的路径。

当打开一个.wav 文件后，将鼠标游标移到 Ctrl 栏，点击右键，即会出现 Mark Number。



选择好要插入的 Mark Number 后，在控制栏中会出现插入的 Mark，此时就完成一个插码的动作。如下图：



6.4 在 Q-Code 程序中使用 Q-Sound

在 *Q-Code* 使用 *Q-Sound* 的 Split 功能，可依据下列的步骤进行操作：

第一步：如何打开 ***Q-Sound*** 窗口。

第二步：如何打开 **Split** 窗口。

第三步：设定 **Noise Gate**。

第四步：设定 **Minimum Mute Length**。

第五步：进行 **Auto Mark**。

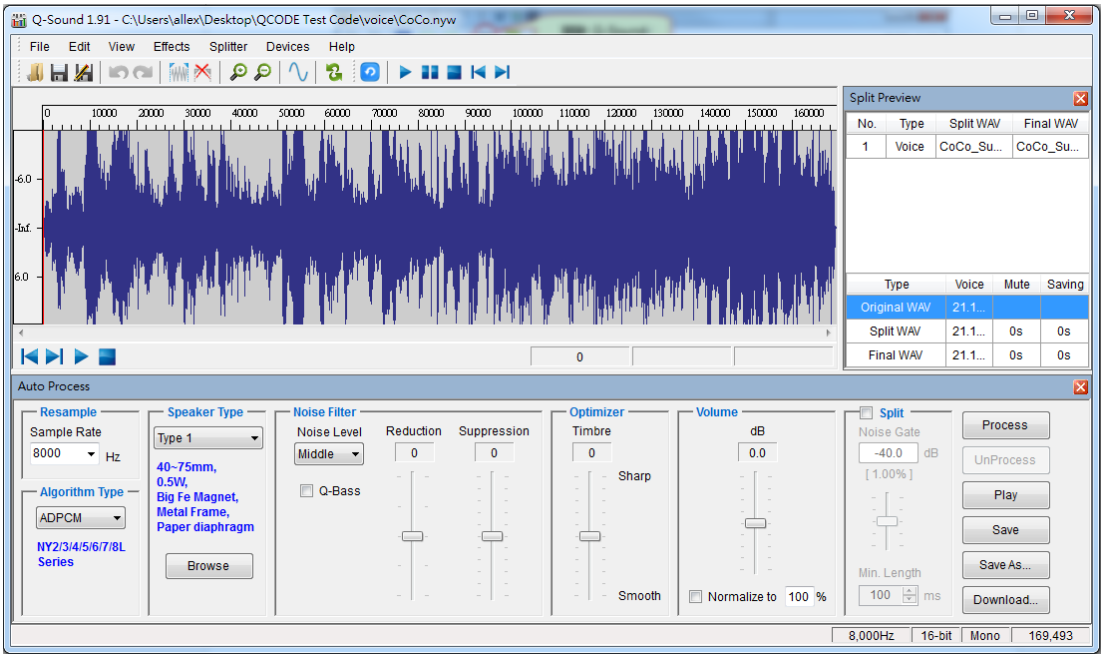
第六步：调整 **Mark**。

第七步：在 ***Q-Code*** 的程序中使用.nyw 文件。

第一步：如何打开 *Q-Sound* 窗口。

在 *Q-Code* 的段落菜单中单击 Voice File 段落中的 Add File，打开 Voice File 对话框。

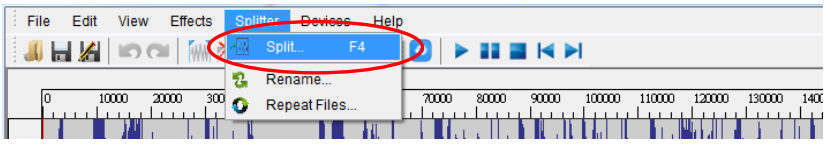
单击  打开 *Q-Sound*。



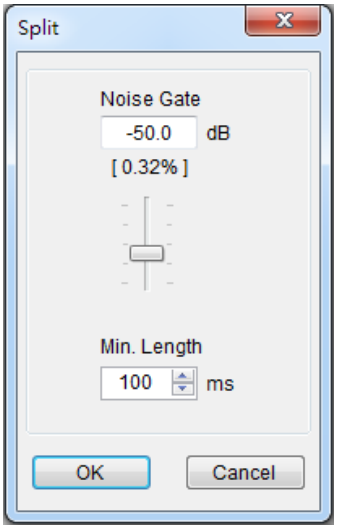
注意：支持.wav 和.nyw 二种文件格式。

第二步：如何打开 Split 窗口。

按下菜单 [Splitter]。

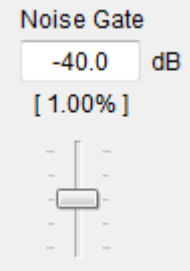


点击 Split 打开 Split 窗口。



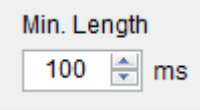
第三步：设定 Noise Gate。

调整 Noise Gate 的值，设定静音区段音量的范围。



第四步：设定 Minimum Mute Length。

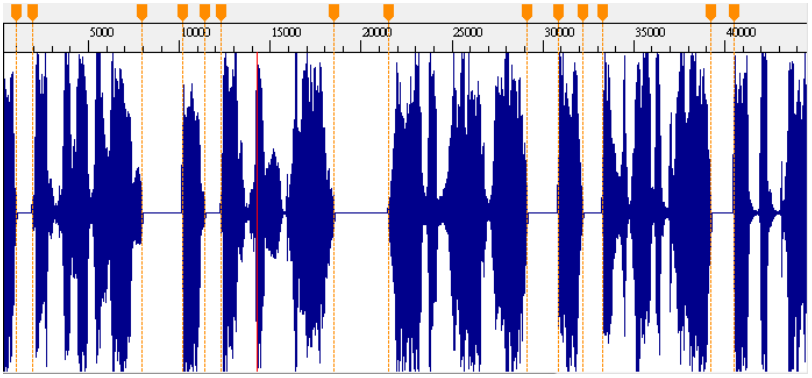
调整 Minimum Mute Length 的值，来设定静音区段的最小长度。大于这个长度的静音段，*Q-Sound* 才会进行处理。



第五步：进行 Auto Mark。

当设定完成后，*Q-Sound* 会依照 Noise Gate 及 Minimum Mute Length 的设定，自动插入标记。

完成后，会在以下对话框中显示出静音段的数量，并于 Split Preview 窗口产生预计切割的状态。



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预期，请回到第二步继续操作。

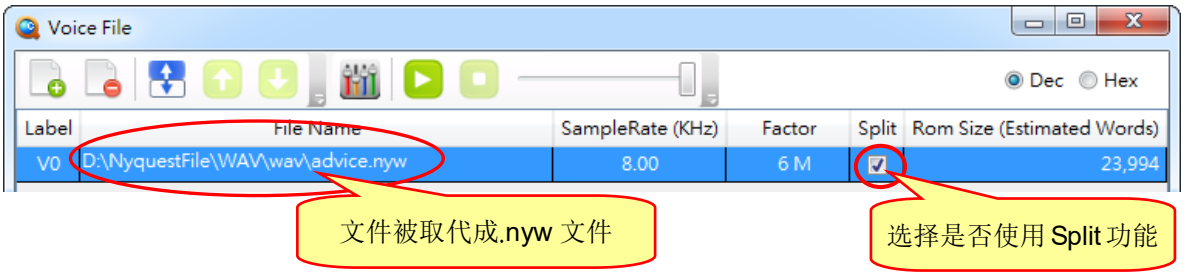
第六步：调整 Mark。

若是想要手动调整标记，可将光标移至标记上，此时会出现  ，压住左键即可进行微调。

完成后，检视 Split Preview 窗口中的文件切割状态是否符合预期。若是不符合预期，请重复第二步和第三步的操作，直至符合预期。确定无误后，按下保存并关闭 *Q-Sound*。

第七步：在 *Q-Code* 的程序中使用.nyw 文件。

关闭 *Q-Sound* 回到 Voice File 对话框，进行功能设定。（如果未勾选 Split，即使已经打开 *Q-Sound* 完成插码并保存，仍然不会进行切割。）



按 OK 关闭 Voice File 对话框后，会在 Voice File 区块自动产生程序码。

```
[Voice File]
V0 = D:\Demo\advice.nyw /5 /fs /s
```

播放声音文件“V0”的方式，与一般未使用 *Q-Sound* 处理过的音文件，采用相同的播放指令，用户可以当成播放单一声音文件的方式来处理。

```
[Path]
PowerOn:playV($v0), input_0
p1: playV($v0)
p2: playV($v0)
```

当语音文件包含相当程度的静音，编译（Build）使用 Split 功能的 *Q-Code* 程序，会发现占用的 ROM Size 变小了。下图是同一个 *Q-Code* 程序，未使用与使用 Split 功能编译后，在 Status bar 上的 ROM Size 信息。

注意：更详细的 Q-Sound 功能，请参阅 Q-Sound 用户手册。

6.5 *Q-Code* 指令表

6.5.1 NY4 *Q-Code* 指令表

6.5.1.1 算数逻辑指令（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 路径指令（Path Command）

Path Command				
ASM	BG	BreakFG	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro

6.5.1.5 语音指令（Voice Command）

Voice Command				
PlayV	PlayVS	Voicename	WaitVN(Ch)	PauseV(Ch)
ResumeV(Ch)	StopV(Ch)	Fre.qcH = 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)
TableH(TableName, X, Y, Ri)	Table(TableName, X, Y, Rh, Rm, Rl)

Table Command	
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 红外线指令（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 串行数据接收指令（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 脉冲调变 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	WaitPN	StopPWM	PausePWM
ResumePWM	-	-	-	-

6.5.1.11 时间延迟指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.1.12 动作指令（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	WaveMark 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 算数逻辑指令（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]				
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 路径指令（Path Command）

Path Command				
ASM	BG	BreakFG	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro

6.5.2.5 语音指令（Voice Command）

Voice Command				
PlayV	PlayVS	Voicename	WaitVN(Ch)	PauseV(Ch)
ResumeV(Ch)	StopV(Ch)	Fre.qcH = nK	-	-

6.5.2.6 句子指令（Sentence Command）

Sentence Command				
PlayS	-	-	-	-

6.5.2.7 Melody 指令（Melody Command）

Melody Command				
PlayM	PlayMS	WaitMN	PauseM	ResumeM
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 红外线指令（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 串行数据接收指令（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 脉冲调变 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	WaitPN	StopPWM	PausePWM
ResumePWM	-	-	-	-

6.5.2.13 中断指令（Interrupt Command）

Interrupt Command				
INT_ON	INT_OFF	INT_RET	INT = n	-

6.5.2.14 时间延迟指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.2.15 动作指令（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	WaveMark State	MelodyMark State
NoteOn 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 算数逻辑指令（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				

4-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 路径指令（Path Command）

Path Command				
ASM	BG	BreakFG	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro

6.5.3.5 语音指令（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 计数器指令（Counter Command）

Counter Command				
TMR_ON	TMR_OFF	-	-	-

6.5.3.8 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:Rj)	ReadTempo(Xi)	Mute_On(Ch)	Mute_Off(Ch)	OKON_On
OKON_Off	OKON_Play	OKON_SustainOn	OKON_SustainOff	OKON_SustainEnd
DynamicOn	DynamicOff	-	-	-

6.5.3.9 键盘指令（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 红外线指令（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 串行数据接收指令（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）

PWMIO 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）

PWMIO Command			
UART_TX	UART_TX_WaitN	UART_RX	UART_RX=data?Path
UART_RX!=data?Path		-	-

6.5.3.16 脉冲调变 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	WaitPN	StopPWM	PausePWM
ResumePWM	-	-	-	-

6.5.3.17 中断指令（Interrupt Command）

Interrupt Command				
INT_ON	INT_OFF	INT_RET	INT = n	-

6.5.3.18 时间延迟指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.3.19 动作指令（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 扩展芯片指令（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		-		

6.5.3.23 一般指令（MISC Command）

MISC Command				
Input State	Outut State	Action Mark State	WaveMark State	MelodyMark State
NoteOn 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 算数逻辑指令（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 路径指令（Path Command）

Path Command				
ASM	BG	BreakFG	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro

6.5.4.5 语音指令（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(Rl:Rk:Rj:Ri, SPIGroup)		SPIGetIndex(Xj:Xi, 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_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 比较器指令（Comparator Command）

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 计数器指令（Counter 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 键盘指令（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 红外线指令（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 串行数据接收指令（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 脉冲调变 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	WaitPN	StopPWM	PausePWM
ResumePWM	-	-	-	-

6.5.4.19 时间延迟指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.4.20 动作指令（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)	Debounce(Time)	Ri = LVD
Direct_Debounce(Time)		Matrix_Debounce(Time)		

6.5.5 NY7 Q-Code 指令表

6.5.5.1 算数逻辑指令（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	
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]		

Flow Control Command				
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 路径指令（Path Command）

Path Command				
ASM	BG	BreakFG	StopFG	StopBG
StopBG1	StopBG2	Subroutine	Label(Pathname)	Macro

6.5.5.5 语音指令（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 键盘指令（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 红外线指令（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 串行数据接收指令（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 脉冲调变 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	WaitPN	StopPWM	PausePWM
ResumePWM	-	-	-	-

6.5.5.17 时间延迟指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.5.18 动作指令（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 算数逻辑指令（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 路径指令（Voice 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 触摸键指令（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 串行控制传送指令（Serial Control TX Command）

Serial Control TX Command		
SC_TX(Mode, data)	SC_TX(Mode, Ri:Rj:RI:Rk)	SC_TX(Mode, Xi:Xj)

6.5.6.9 脉冲调变 IO 指令（PWMIO Command）

PWMIO Command				
PWMOut	PWMOutS	WaitPN	PlayPWMS	WaitPN
StopPWM	PausePWM	ResumePWM	HoldPWM	PWMCtrl

6.5.6.10 时间延迟指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	WaitDN(n)	StopD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.6.11 动作指令（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 算数逻辑指令（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 FD An]	-	-	-

6.5.7.4 路径指令（Path Command）

Path Command				
C-Code	BG	StopFG	StopBG	StopBG1
StopBG2	StopBG3	StopBG4	StopBG5	Subroutine
Label(Pathname)	Macro	-	-	-

6.5.7.5 语音指令（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 录音指令（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 定时器指令（Timer Command）

Counter 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 键盘指令（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 触摸键指令（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 红外线指令（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 中断指令（Interrupt Command）

Interrupt Command				
INT_ON	INT_OFF	-	-	-

6.5.7.22 时间延迟指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.7.23 动作指令（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_Birghtness
LEDText_Play	LEDText_PlayS	LEDText_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 听声辨位指令（Sound Localization Command）

Sound Localization Command				
SL_On	SL_Off	SL_RX	-	-

6.5.7.28 声音侦测指令（Sound Detect Command）

Sound Detect Command				
SoundDetect_On	SoundDetect_Off	-	-	-

6.5.7.29 音高侦测指令（Sound 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 输入指令（ADC Input Command）

ADC Input Command				
ADC_Input	-	-	-	-

6.5.7.32 语音识别指令（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

6.5.7.33 变音效果指令（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 实时播放指令（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	-	-	-	-

6.5.7.35 动物变音指令（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 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			

6.5.7.37 I/O 扩展芯片指令（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	-

MISC Command				
ReadRollingCode		Ri = LVD	Enforce_Wakeup	GetMicVol
Millis	Srand	-	-	-

6.5.7.39 侦错指令（Debug Command）

Debug Command				
Printf	-	-	-	-

6.5.8 NX1 EF Q-Code 指令表

6.5.8.1 算数逻辑指令（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
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		-

Flow Control Command				
!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 路径指令（Path Command）

Path Command				
C-Code	BG	StopFG	StopBG	StopBG1
StopBG2	StopBG3	StopBG4	StopBG5	Subroutine
Label(Pathname)	Macro	-	-	-

6.5.8.5 语音指令（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 录音指令（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 定时器指令（Timer Command）

Counter 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 键盘指令（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 触摸键指令（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 Command	
Table(TableName, VarX, VarY, VarR)	-

6.5.8.19 红外线指令（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 中断指令（Interrupt Command）

Interrupt Command				
INT_ON	INT_OFF	-	-	-

6.5.8.23 时间延迟指令（Delay Command）

Delay Command				
Delay(time)	Delay(Ri:Rj:Rk)	WaitDN(n)	StopD(n)	PauseD(n)
ResumeD(n)	SDelay(time)	-	-	-

6.5.8.24 动作指令（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_Birghtness
LEDText_Play	LEDText_PlayS	LEDText_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 听声辨位指令（Sound Localization Command）

Sound Localization Command				
SL_On	SL_Off	SL_RX	-	-

6.5.8.29 声音侦测指令（Sound Detect Command）

Sound Detect Command				
SoundDetect_On	SoundDetect_Off	-	-	-

6.5.8.30 音高侦测指令（Sound 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 输入指令（ADC Input Command）

ADC Input Command

ADC_Input	-	-	-	-

6.5.8.33 语音识别指令（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

6.5.8.34 变音效果指令（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 实时播放指令（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 动物变音指令（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 侦错指令（Debug Command）

Debug Command				
Printf	-	-	-	-

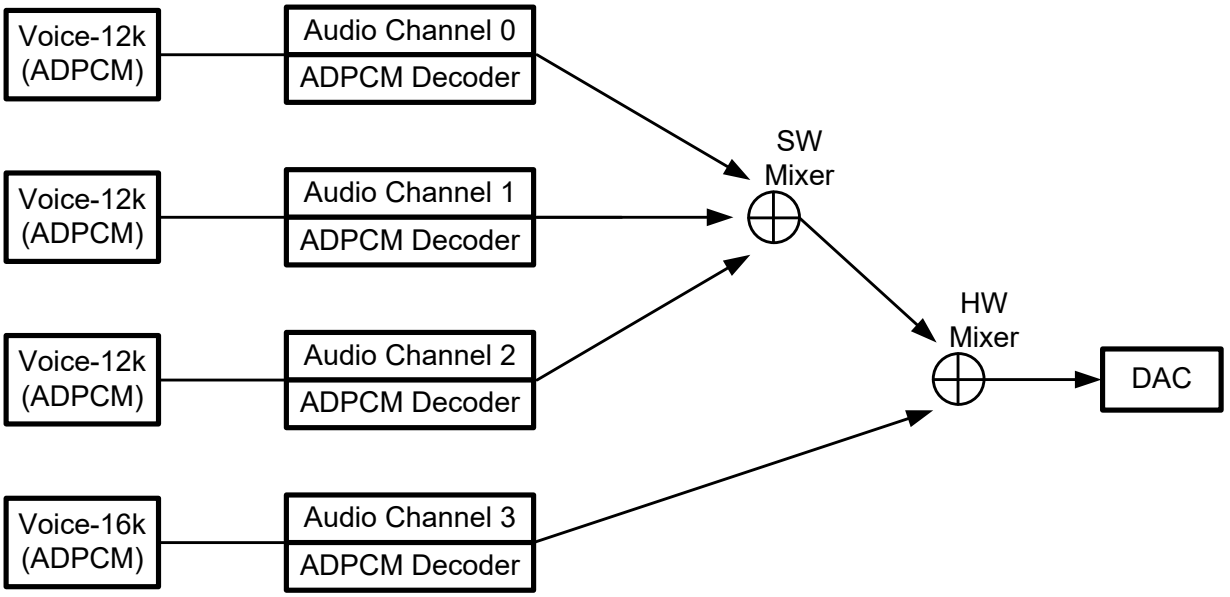
6.6 NX1 声音输出通道

NX1(底层为 C_MODULE)支持八个语音同时通过软件通道输出，软件通道分别为：Audio Channel 0 ~ Audio Channel 7，底层程序会根据音乐的取样频率，自动对应到硬件通道的 HW_CH0 与 HW_CH1。同时只能播放两种不同的取样频率，若同时播放三种不同的取样频率，则会造成播音异常。若有使用语音录音或是音效(SpeedChange / PitchChange / RobotX / Echo / Reverb / Darth)，软件通道同时可以支持的通道数会降为两通道。

语音格式上，最多支持两组 SBC-1、两组 SBC-2、八组 ADPCM、八组 ADM、三组 PCM、一组 CELP 和一组 MIDI 译码算法，使用上须注意译码算法的组数，如果同一时间需要播放的语音超过支持的组数，语音会无法成功播放。多通道的应用需要设定算法的使用状况，设定方法请参考 RAM Usage。

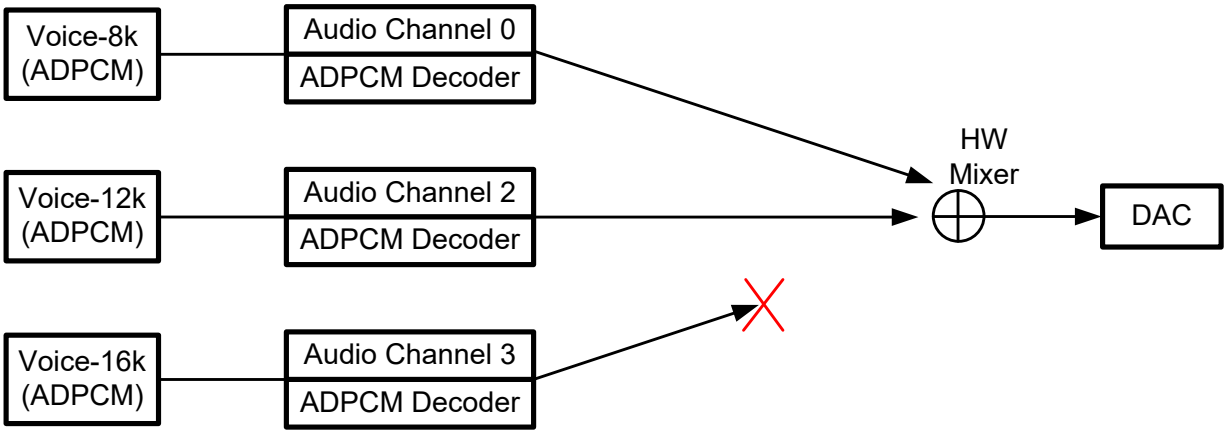
例. 四通道 ADPCM 同时播放，总共两种 sample rate

TR1: PlayVS(Ch0, $v0), PlayVS(Ch1, $v1) , PlayVS(Ch2, $v2) , PlayV(Ch3, $v3)



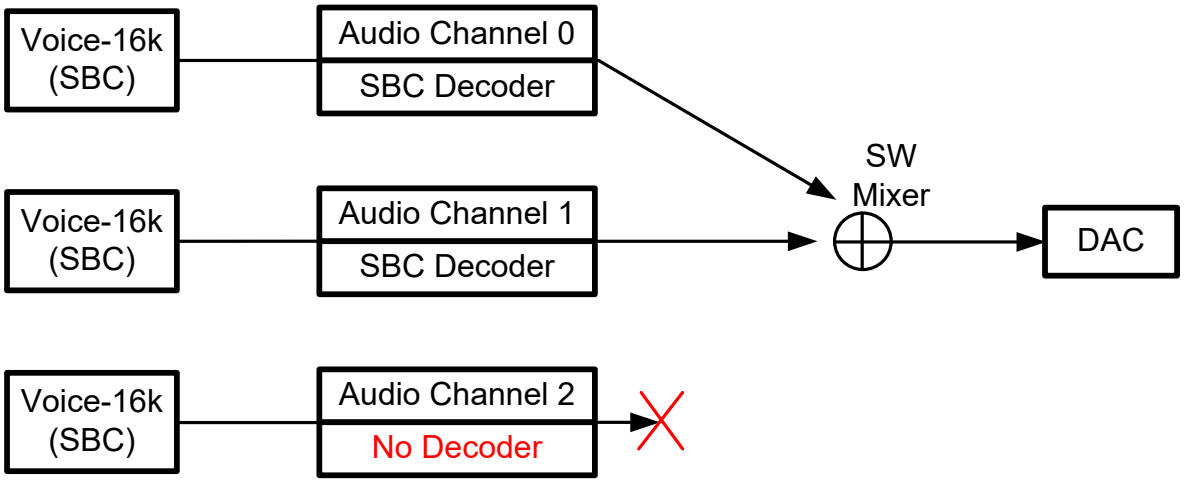
例. 三通道 ADPCM 总共三种 sample rate，无法同时播放

TR1: PlayVS(Ch0, $v0), PlayVS(Ch1, $v1) , PlayV(Ch2, $v2)



例. 三通道 SBC 无法同时播放，

TR1: PlayVS(Ch0, $v0), PlayVS(Ch1, $v1) , PlayV(Ch2, $v2)



6.7 RAM 资源使用说明

在 *Q-Code* 中，使用到特定的功能或是指令时，会占用到 RAM。根据使用不同的功能或是指令，占用的 RAM 数量皆不同。系统占用的规则为从编号后面的 RAM 开始往前占用。占用 RAM 的功能与数量关系由以下各章节说明。

6.7.1 NY4 RAM 资源使用说明

共有两个 Page 分别为 Page0、Page1。每个 Page 共有 48 个 RAM，总共有 96 的 RAM 可以使用。其中的 76 个 RAM 开放给 User 使用，其余为系统占用。

注意：实际的 RAM 配置地址，系统会依照使用的功能随机做配置。

Q-Code 功能	说明	使用 RAM 数量
系统占用	程序流程控制。NY4A 与 NY4B 分别占用不同数量。	20 / 22
Action	使用 PlayA / PlayAS 时，每一 Action 通道占用 9 个 RAM。 若打开压缩功能，则每一 Action 通道会多使用 1 个 RAM。	(9+n) * ch
Action Mark State	超过 15 个 action mark state 时，会占用 2 个 RAM。	1~2
	路径超过 15 个 Path 时，会再占用 1 个 RAM。	1~2
BackGround	使用背景 1 路径时。超过 15 个 Path 时，会再占用 1 个 RAM。	1~2
	使用背景 2 路径时。超过 15 个 Path 时，会再占用 1 个 RAM。	1~2
Delay	使用 Delay 时，每一层使用 3 个 RAM。系统占用 2 个 RAM。	3*n+2
FD	使用 FD 时，有使用的 Port 会占用 1 个 RAM。	1~2
Input State	超过 15 个 Input State 时，会占用 2 个 RAM。	1~2
IR	使用 IR TX 时。DATA Length 每增加 4-bit 就多占用 1 个 RAM。	1~4
	使用 IR RX 时。DATA Length 每增加 4-bit 就多占用 1 个 RAM。	1~4
Key Function	Matrix 每一根输出 pin 对应一组输入 port 会占用 3 个 RAM。 Direct 每一 port 会占用 3 个 RAM。 Debounce 超过 60ms 时，会占用 2 个 RAM。 系统占用 2 个 RAM。	(Key / 4)*3 + 3~4
Random	使用 Random 时。当 Random 超过 16 个，会再占用 1 个 RAM。	3~4
Repeat	在前景使用 Repeat 功能时。	1
	在背景 1 使用 Repeat 功能时。	1
	在背景 2 使用 Repeat 功能时。	1
Subroutine	在 Subroutine 段落中，提供巢状堆栈功能。 每调用第一层就需要占用 4 个 RAM，调用第二层时会再占用 4 个 RAM。	4 / 8
PauseD	使用 PauseD 时，每一层占用 3 个 RAM。	3*n
PlayVS	使用 PlayVS 时。	1
QIO	使用 QIO 时。每使用 1 个 Port，则会占用一个 RAM。系统	1*n+5

Q-Code 功能	说明	使用 RAM 数量
	占用 5 个 RAM。	
Wait	使用 WaitVN 时，占用 1 个 RAM，系统占用 1 个 RAM。	1+1
	使用 WaitDN 时，占用 1 个 RAM，系统占用 1 个 RAM。	1+1
	使用 WaitAN 时，占用 2 个 RAM，系统占用 1 个 RAM。	2+1
Wave Mark	使用 Wave Mark 时。当 Wave Mark State 超过 15 个，会再占用 1 个 RAM。	2~3

6.7.2 NY5 RAM 资源使用说明

NY5 共有四个 Page 分别为 Page0、Page1、Page2、Page3。每个 Page 共有 56 个 RAM，总共有 224 的 RAM 可以使用。其中的 188 个 RAM 开放给用户使用，其余为系统占用。

注意：实际的 RAM 配置地址，系统会依照使用的功能随机做配置。

Q-Code 功能	说明	使用 RAM 数量
系统占用	程序流程控制。(NY5*20 I/O; NY5C*24 I/O)	36 / 36 / 36 / 38
Action	使用 PlayA / PlayAS 时，每一 Action 通道占用 10 个 RAM。 若打开压缩功能，则每一 Action 通道会多使用 1 个 RAM。	(10+n)*ch
	使用 PlayA / PlayAS 时，且设定 InterruptService = Action，则会多占用 7 个 RAM。	(10+n)*ch+7
Action Mark State	超过 15 个 action mark state 时，会占用 2 个 RAM。	1~2
	路径超过 15 个 Path 时，会再占用 1 个 RAM。	1~2
BackGround	使用背景 1 路径时。超过 15 个 Path 时，会再占用 1 个 RAM。	1~2
	使用背景 2 路径时。超过 15 个 Path 时，会再占用 1 个 RAM。	1~2
Delay	使用 Delay 时，每一层使用 3 个 RAM。系统占用 2 个 RAM。	3*n+2
FD	使用 FD 时，有使用的 Port 会占用 1 个 RAM。	1~6
Input State	超过 16 个 Input State 时，会占用 2 个 RAM。	1~2
IR	使用 IR TX 时。DATA Length 每增加 4-bit 就多占用 1 个 RAM。	1~4
	使用 IR RX 时。DATA Length 每增加 4-bit 就多占用 1 个 RAM。	1~4
Interrupt Path	使用 Interrupt Path 时。	8~12
Key Function	Matrix 每一根输出 pin 对应一组输入 port 会占用 3 个 RAM。 Direct 每一 port 会占用 3 个 RAM。 Debounce 超过 60ms 时，会占用 2 个 RAM。 系统占用 2 个 RAM。	(Key / 4)*3 + 3~4
Melody Mark	使用 Melody Mark 时。当 Melody Mark 编号超过 15 时，会再占用 1 个；当 Melody Mark State 超过 15 个，会再占用 1 个 RAM。	2~4

Q-Code 功能	说明	使用 RAM 数量
Note On	使用 Note On 时。当 Note On State 超过 15 个，会再占用 1 个 RAM。	2~3
PauseD	使用 PauseD 时，每一层占用 3 个 RAM。	3*n
PauseDown / ResumeUp	使 PauseDown / ResumeUp 时，则占用 4 个 RAM。	4
PauseV	使用 PauseV 时。	1
PlayM	使用 PlayM + 4 通道 PCT Mode。	99
	使用 PlayM + 4 通道 ADSR Mode（自动打开 InterruptService = Melody）。	117
	使用 PlayM 时，且设定 InterruptService = Melody，则会多占用 7 个 RAM。	7
PlayMS	使用 PlayMS 功能时，除了原本 PlayM 外，需多占用 1 个。	1
PlayVS	在前景使用 PlayVS 时，占用 1 个 RAM。	1
	在背景 1 使用 PlayVS 时，占用 1 个 RAM。	1
	在背景 2 使用 PlayVS 时，占用 1 个 RAM。	1
QIO	使用 QIO 时，每使用 1 个 Port，则会占用一个 RAM。系统占用 6 个 RAM。	1*n+6
	使用 QIO 时，且设定 InterruptService = QIO，则会多占用 10 个 RAM。	(1*n+6)+10
Random	使用 Random 时。当 Random 超过 16 个，会再占用 1 个 RAM。	3~4
Repeat	在前景使用 Repeat 功能时。	1
	在背景 1 使用 Repeat 功能时。	1
	在背景 2 使用 Repeat 功能时。	1
Single Play	使用 Single Play 时。	2
Subroutine	在 Subroutine 段落中，提供巢状堆栈功能。 每调用第一层就需要占用 4 个 RAM，调用第二层时会再占用 4 个 RAM。	4 / 8
Wait	使用 WaitVN 时，占用 2 个 RAM，系统占用 1 个 RAM。	2+1
	使用 WaitDN 时，占用 1 个 RAM，系统占用 1 个 RAM。	1+1
	使用 WaitAN 时，占用 2 个 RAM，系统占用 1 个 RAM。	2+1
	使用 WaitMN 时，占用 1 个 RAM，系统占用 1 个 RAM。	1+1
Wave Mark	使用 Wave Mark 时。当 Wave Mark State 超过 15 个，会再占用 1 个 RAM。	2~3

6.7.3 NY5+ RAM 资源使用说明

总共有 215 的 RAM 可以使用。系统会依照使用的功能多寡，动态决定所有功能所需的 RAM 数量；因此将不必要的功能关闭以增加可使用的 RAM 数量。可使用的 RAM 数量会在项目执行完 Build 动作后列在 Build Message 窗口内。

6.7.4 NY6 RAM 资源使用说明

共有六个 Page：Page0 ~ Page5。每个 Page 共有 56 个 RAM，总共有 336 的 RAM 可以使用。系统会依照使用的功能多寡，动态决定所有功能所需的 RAM 数量；因此将不必要的功能关闭以增加可使用的 RAM 数量。可使用的 RAM 数量会在项目执行完 Build 动作后列在 Build Message 窗口内。

6.7.5 NY7 RAM 资源使用说明

共有两个 Page 分别为 Page0、Page1。每个 Page 共有 224 个 RAM，总共有 448 的 RAM 可以使用。系统会依照使用的功能多寡，动态决定所有功能所需的 RAM 数量；因此将不必要的功能关闭以增加可使用的 RAM 数量。可使用的 RAM 数量会在项目执行完 Build 动作后列在 Build Message 窗口内。

6.7.6 NY9T RAM 资源使用说明

在 *Q-Code* 中，使用到特定的功能或是指令时，会占用到 RAM。根据使用不同的功能或是指令，占用的 RAM 数量皆不同。系统占用的规则为从编号后面的 RAM 开始往前占用。占用 RAM 的功能与数量关系表，如下表所示。

NY9T001A / NY9T004A RAM 共有 1 个 Page 为 Page0，总共有 48 的 RAM 可以使用。其中的 41 个 RAM 开放给 User 使用，其余为系统占用。

注意：实际的 RAM 配置地址，系统会依照使用的功能随机做配置。

Q-Code 功能	说明	使用 RAM 数量
Reserved	程序流程控制。	7
AutoJudge_Calibrate	使用触摸键自动更正时，占用 1 个 RAM。	1
Enforce_Calibrate	使用触摸键强制自动更正时，占用 1 个 RAM。	1
Process	使用前景指令时，占用 4 个 RAM。	4
	使用背景 1 的指令时，占用 4 个 RAM。	4
	使用背景 2 的指令时，占用 4 个 RAM。	4
IO	根据不同 Body 的输出埠数量，所占用的数量有所不同。	1*n
Action	使用 PlayA / PlayAS 时，每一通道占用 9 个 RAM。	9
	使用 Period 功能时，每一通道占用 1 个 RAM。	1
	使用 Extension 功能时，每一通道占用 6 个 RAM。	6
	使用 Loop 功能时，通道 1~4 会占用 1 个 RAM，通道 5~8	1~2

Q-Code 功能	说明	使用 RAM 数量
	会占用 1 个 RAM。	
	播放整组 VIO 时，会根据定义的输出埠数量占用 RAM。	1*n
	使用 PlayA / PlayAS 时，且设定 Action = INT，则会多占用 6~7 个 RAM。	8*ch+(6~7)
PWM-IO	使用 PlayPWM / PlayPWMS / PWMOut 时，占用 1 个 RAM。	1
BackGround	使用背景 1 路径时。超过 15 个 Path 时，会再占用 1 个 RAM。	1~2
	使用背景 2 路径时。超过 15 个 Path 时，会再占用 1 个 RAM。	1~2
Delay	使用 Delay 时，每一层使用 3 个 RAM。系统占用 2 个 RAM。	3*n+2
Input State	超过 15 个 Input State 时，会占用 2 个 RAM。	1~2
Key	每个 Input Port 最多对应 4 个键，原则上每 4 个键，会占用 1 个 RAM。	(Key/4)+2
	Debounce 超过 60ms 时，会占用 2 个 RAM。	1~2
Touch-Key	每个 Input Port 最多对应 4 个键，原则上每 4 个键，会占用 1 个 RAM。	(Key/4)+3
Random	使用 Random 时。当 Random 超过 16 个，会再占用 1 个 RAM。	1~2
Repeat	在前景使用 Repeat 功能时。	1
	在背景 1 使用 Repeat 功能时。	1
	在背景 2 使用 Repeat 功能时。	1
Subroutine	使用 Subroutine 功能时，占用 4 个 RAM。	4
Wait	使用 WaitDN 时，占用 1 个 RAM，系统占用 1 个 RAM。	1+1
	使用 WaitAN 时，占用 1 个 RAM，系统占用 1 个 RAM。	1+1
	使用 WaitPN 时，占用 1 个 RAM，系统占用 1 个 RAM。	1+1
Special Path	使用 4ms 路径时，占用 1 个 RAM。	1
	使用 Enforce_Calibrate 路径时，占用 1 个 RAM。	
Radj	使用 Radj 相关指令时，占用 1 个 RAM。	1

NY9T008A / NY9T016A RAM 共有两个 Page 分别为 Page0、Page1。每个 Page 共有 48 个 RAM，总共有 96 的 RAM 可以使用。其中的 89 个 RAM 开放给 User 使用，其余为系统占用。

注意：实际的 RAM 配置地址，系统会依照使用的功能随机做配置。

Q-Code 功能	说明	使用 RAM 数量
Reserved	程序流程控制。	7
AutoJudge_Calibrate	使用触摸键自动更正时，占用 1 个 RAM。	1

Q-Code 功能	说明	使用 RAM 数量
Enforce_Calibrate	使用触摸键强制自动更正时，占用 1 个 RAM。	1
Process	使用前景指令时，占用 4 个 RAM。	4
	使用背景 1 的指令时，占用 4 个 RAM。	4
	使用背景 2 的指令时，占用 4 个 RAM。	4
IO	根据不同 Body 的输出埠数量，所占用的数量有所不同。	1*n
Arithmetic Logic	使用乘除法时，会占用 2 个 RAM。	2
Action	使用 PlayA / PlayAS 时，每一通道占用 9 个 RAM。	9
	使用 Period 功能时，每一通道占用 1 个 RAM。	1
	使用 Extension 功能时，每一通道占用 6 个 RAM。	6
	使用 Loop 功能时，通道 1~4 会占用 1 个 RAM，通道 5~8 会占用 1 个 RAM。	1~2
	播放整组 VIO 时，会根据定义的输出埠数量占用 RAM。	1*n
	使用 PlayA / PlayAS 时，且设定 Action = INT，则会多占用 7~8 个 RAM。	9*ch+(7~8)
PWM-IO	使用 PlayPWM / PlayPWMS / PWMOut 时，占用 1 个 RAM。	1
BackGround	使用背景 1 路径时。超过 15 个 Path 时，会再占用 1 个 RAM。	1~2
	使用背景 2 路径时。超过 15 个 Path 时，会再占用 1 个 RAM。	1~2
Delay	使用 Delay 时，每一层使用 3 个 RAM。系统占用 2 个 RAM。	3*n+2
Input State	超过 15 个 Input State 时，会占用 2 个 RAM。	1~2
Key	每个 Input Port 最多对应 4 个键，原则上每 4 个键，会占用 1 个 RAM。	(Key/4)+2
	Debounce 超过 60ms 时，会占用 2 个 RAM。	1~2
Touch-Key	每个 Input Port 最多对应 4 个键，原则上每 4 个键，会占用 1 个 RAM。	(Key/4)+3
Random	使用 Random 时。当 Random 超过 16 个，会再占用 1 个 RAM。	1~2
Repeat	在前景使用 Repeat 功能时。	1
	在背景 1 使用 Repeat 功能时。	1
	在背景 2 使用 Repeat 功能时。	1
Subroutine	使用 Subroutine 功能时，占用 4 个 RAM。	4
Wait	使用 WaitDN 时，占用 1 个 RAM，系统占用 1 个 RAM。	1+1
	使用 WaitAN 时，占用 2 个 RAM，系统占用 1 个 RAM。	2+1
	使用 WaitPN 时，占用 1 个 RAM，系统占用 1 个 RAM。	1+1

Q-Code 功能	说明	使用 RAM 数量
Special Path	使用 4ms 路径时，占用 1 个 RAM。	1
	使用 Enforce_Calibrate 路径时，占用 1 个 RAM。	
Serial Control	使用 Serial Control 功能时，占用 4 个 RAM。	4
Radj	使用 Radj 相关指令时，占用 1 个 RAM。	1

6.8 Ri 与 Xi 对应表

Ri（4-bit RAM）与 Xi（8-bit RAM）对应表。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 频率计算公式

6.9.1 NY4 / NY5 频率计算公式

当使用 PlayV 来播放.wav 文件时，时间计数器会影响播放的速度。播放速度的公式如下：

TM =（F_{TCS} / F_{SR}） - 1

TM：时间计数器数值。

F_{TCS}：系统频率（播放语音时为 1MHz）。

F_{SR}：播放速度。

播放频率对照表：

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 来设定播放速度时

```
TR1: X0=0x63, PlayV($V0,X0)          ; 播放速度为 10KHz
TR2: PlayV($V0,X0)                   ; 播放速度为 8KHz
```

6.10 特殊路径中不可使用的功能

特殊路径 512us、1ms、2ms、4ms、8ms、500ms、1sec、4sec、Int_256us、Int_512us、Int_1ms、interrupt 不支持以下指令。

- **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。
  - 其余 touchkey 指令，除 500ms / 1sec / 4sec 路径以外皆不可使用。
- **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 路径外皆不可使用。
- **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 路径外皆不可使用。

6.11 串行信号控制通信协议（Serial Control Protocol）

可连接一般微控制器，将 NY9T 当成微控制器的按键。提供 3 种不同的传输协议，分别是 SPI_Like、NY3 Serial_Trigger、IR_Trigger。用户可以设定成自动侦测或是特定的传输协议。

SPI_Like：使用 2 根脚位，达成类似 SPI 传输协议。

NY3 Serial_Trigger：使用 2 个脚位来连接 NY3 系列，可将 NY9T 当成 NY3 的按键。详细的连接方式，请参阅 NY3C/3D 规格书。

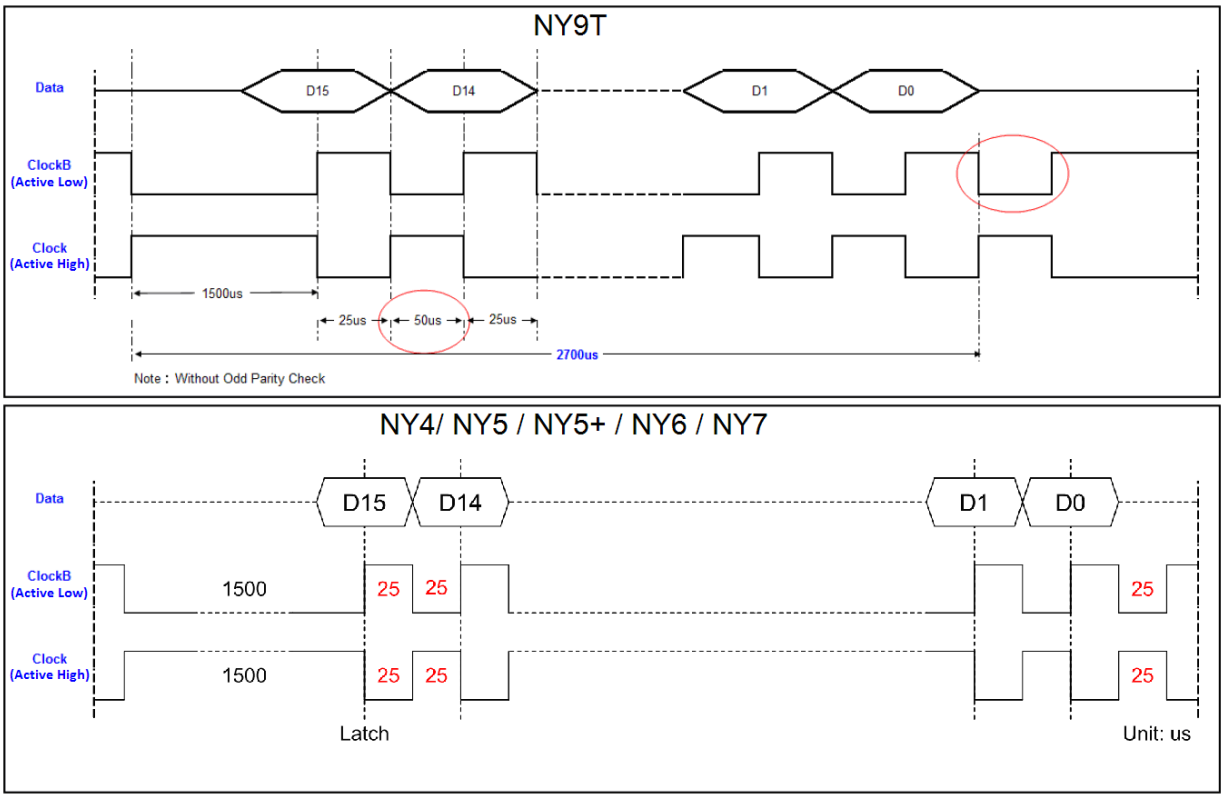
IR_Trigger：使用 1 个脚位来仿真 IR 的接收信号，可直接使用目前的 NY4/5/7 系列的 IR 通信。

自动侦测：可由 3 根脚位的设定方式来决定，NY9T 要使用何种传输协议。自动侦测的脚位设定方式如下表所示：

自动模式侦测			
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

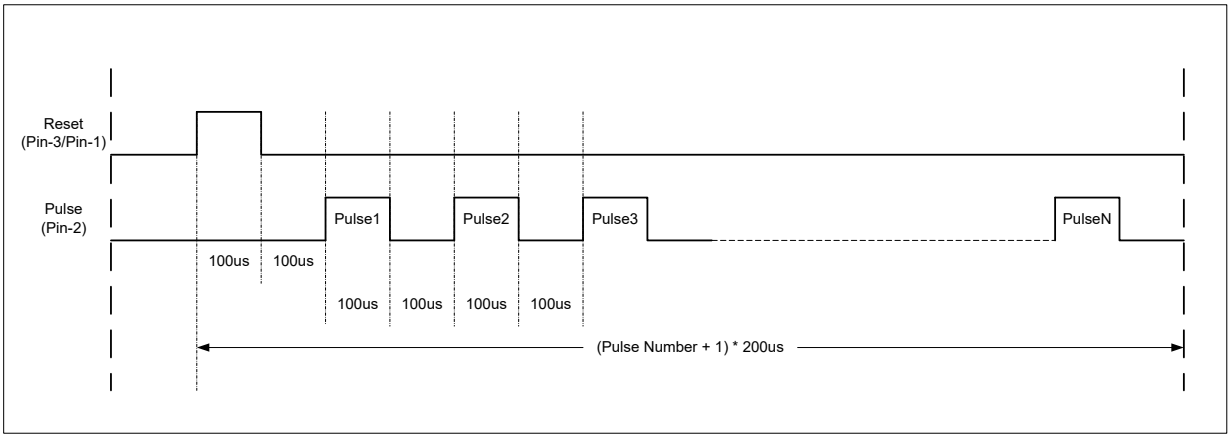
Serial Control 传输协议的时序图如下所示：

SPI_Like Timing

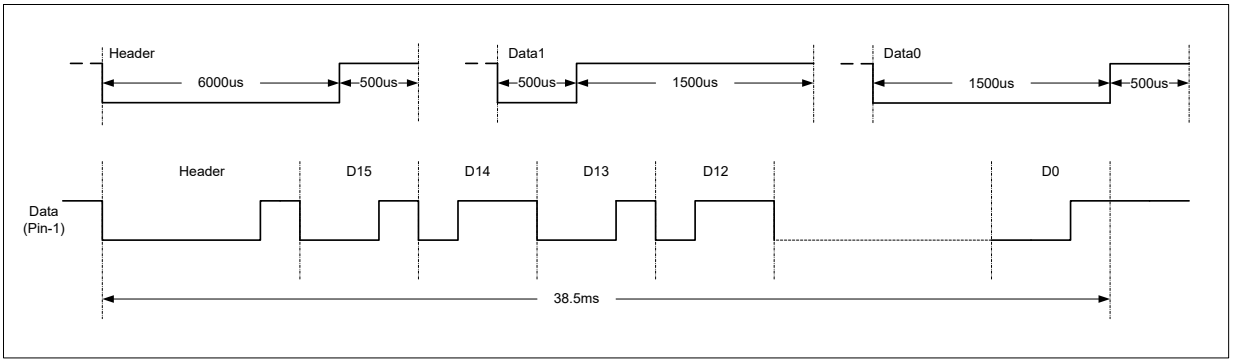


*Note: 上图中的 25us 为最小值，范围为 **25us ~ 100us**。*

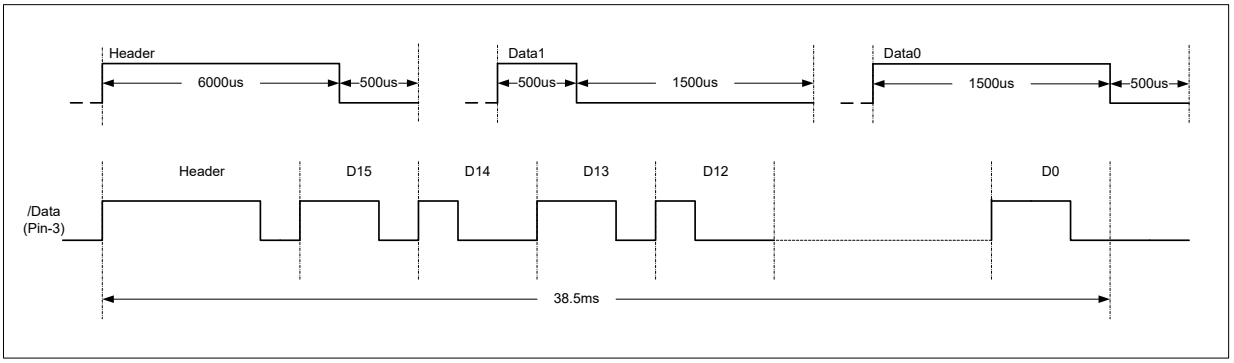
NY3 Serial_Trigger Timing



IR_Trigger(Active Low)



IR_Trigger(Active High)



6.12 NY5 与 NY5+音量阶数对映

当转换 NY5 的.qc 至 NY5+时，由于硬件上的差异，因此音量指令的阶数与 NY5 输出并不相同，以下表格列出在不同设定下，输出音量相同的阶数对映。

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

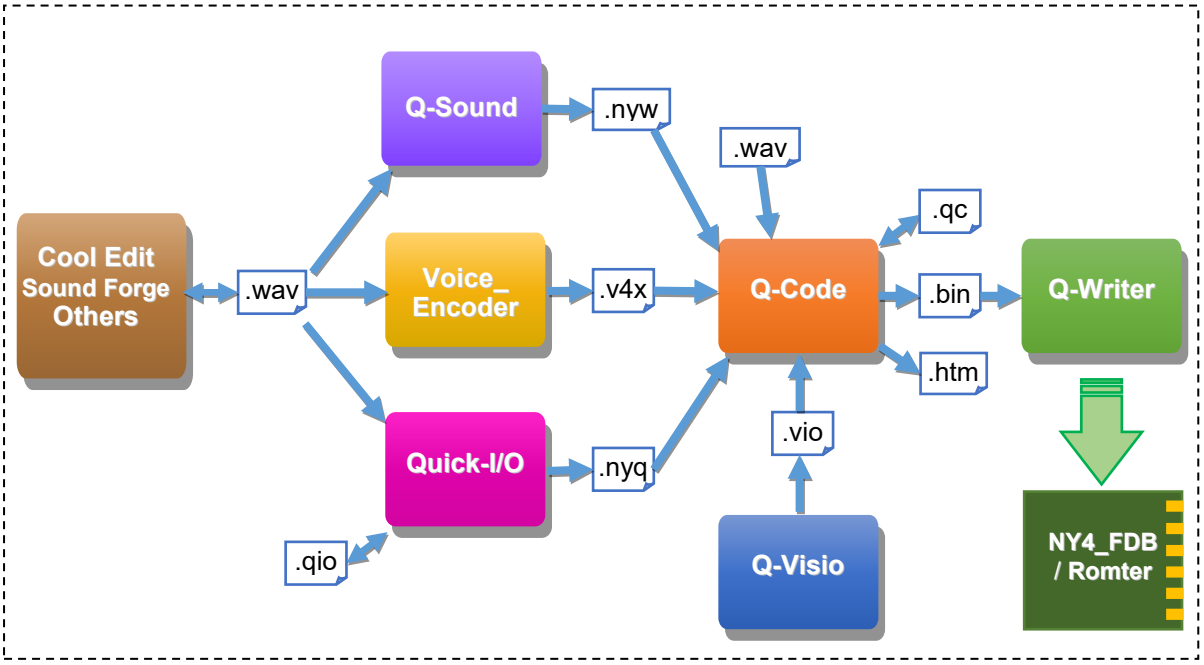
Voltage = 3.0V & PWM Current = Large																
	Vol															
NY5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NY5+	4	7	10	11	12	12	13	13	14	14	15	15	15	15	15	15

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

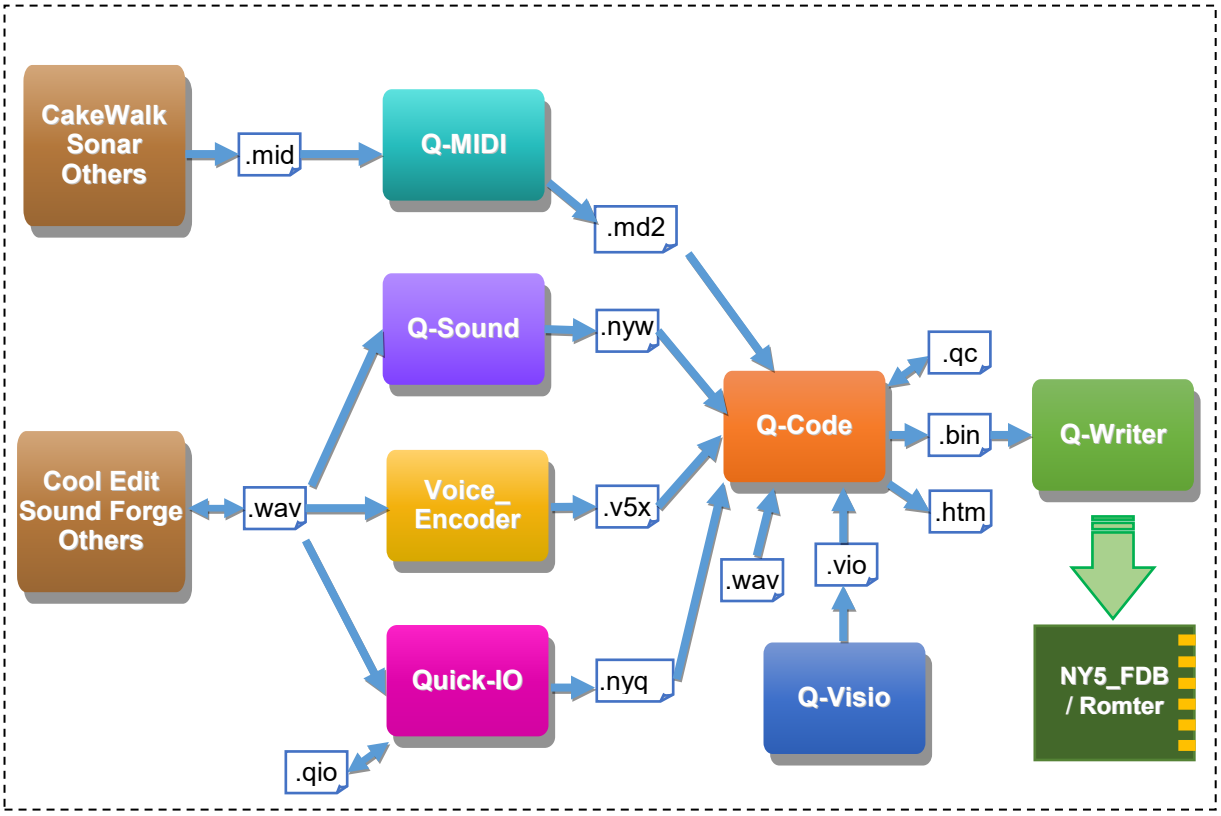
Voltage = 4.5V & PWM Current = Large																
	Vol															
NY5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NY5+	3	6	7	8	9	10	11	12	13	14	15	15	15	15	15	15

6.13 Q-Code 开发流程图

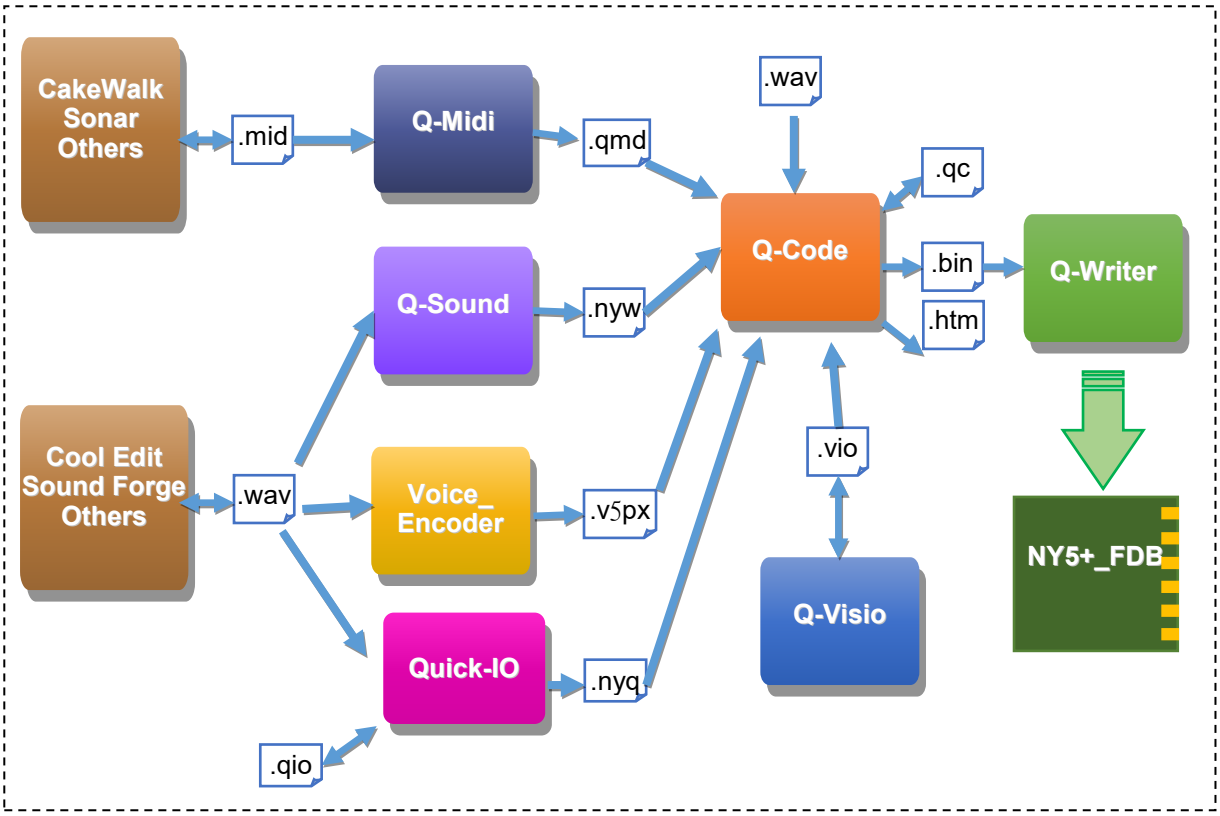
6.13.1 NY4 开发流程图



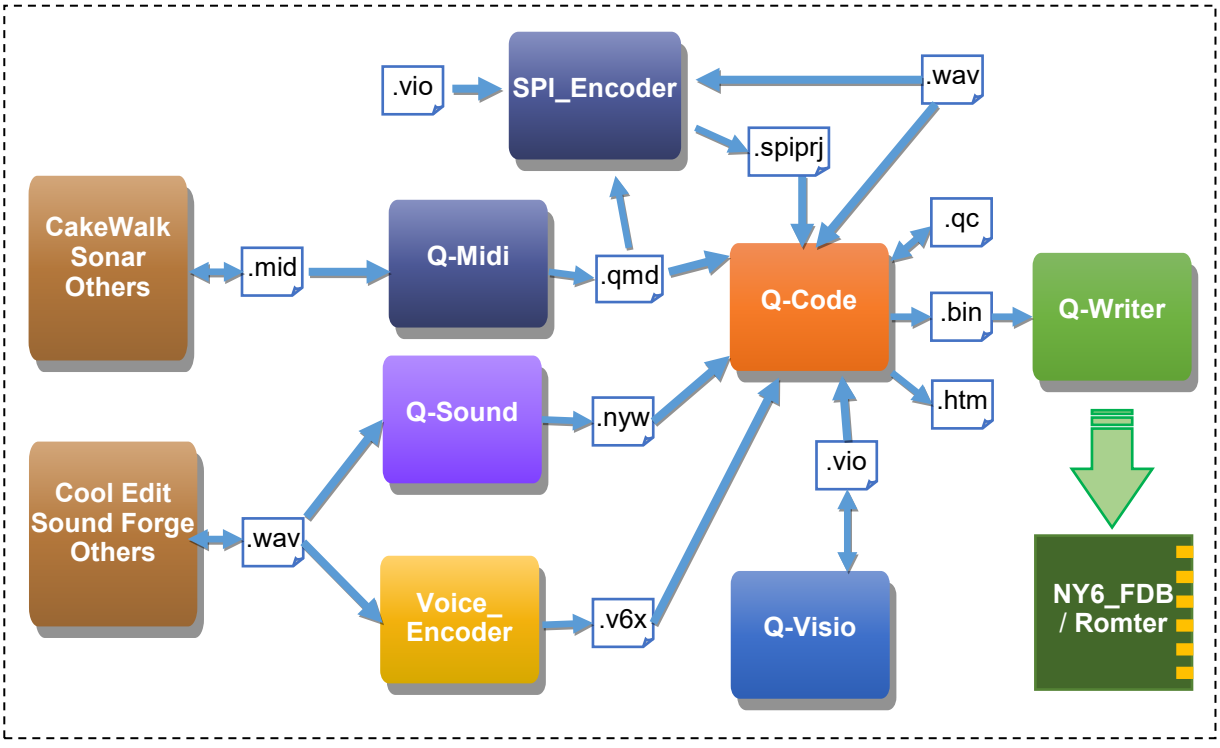
6.13.2 NY5 开发流程图



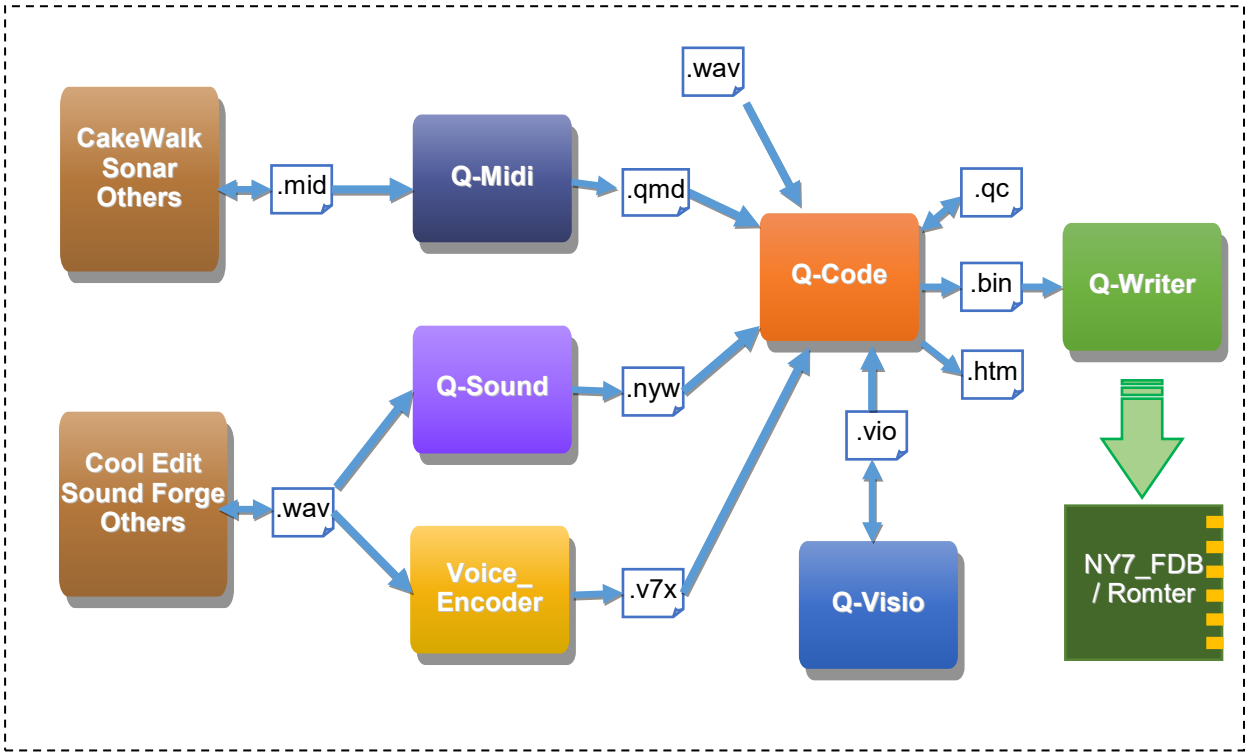
6.13.3 NY5+开发流程图



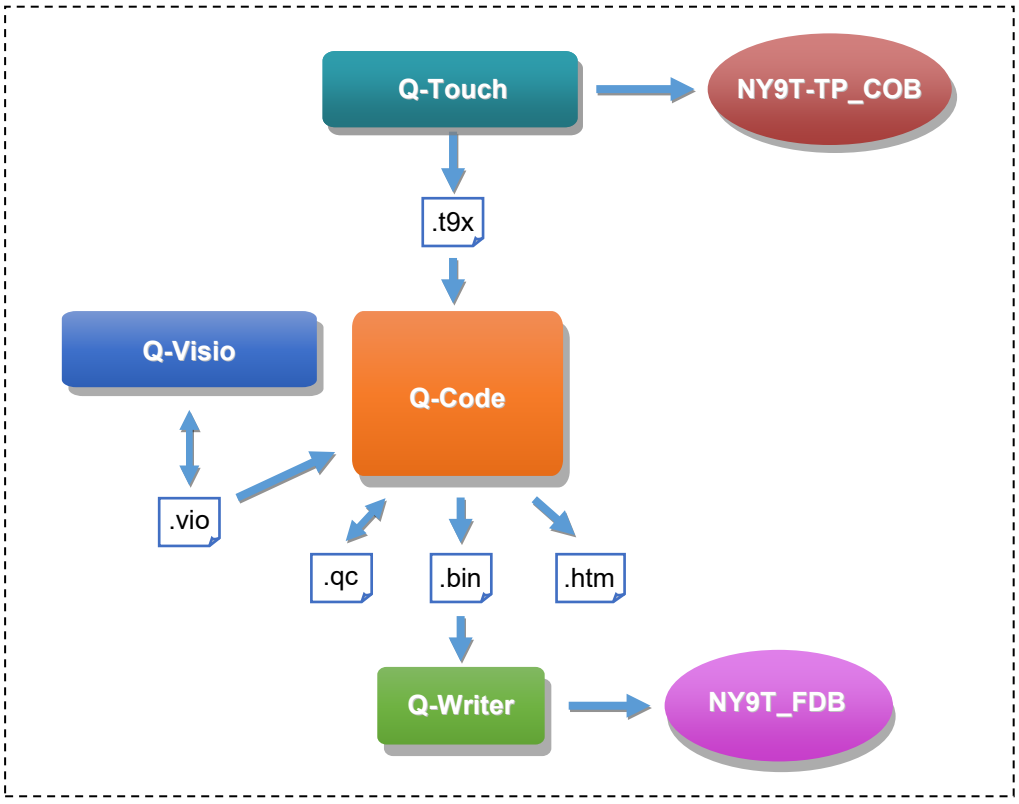
6.13.4 NY6 开发流程图



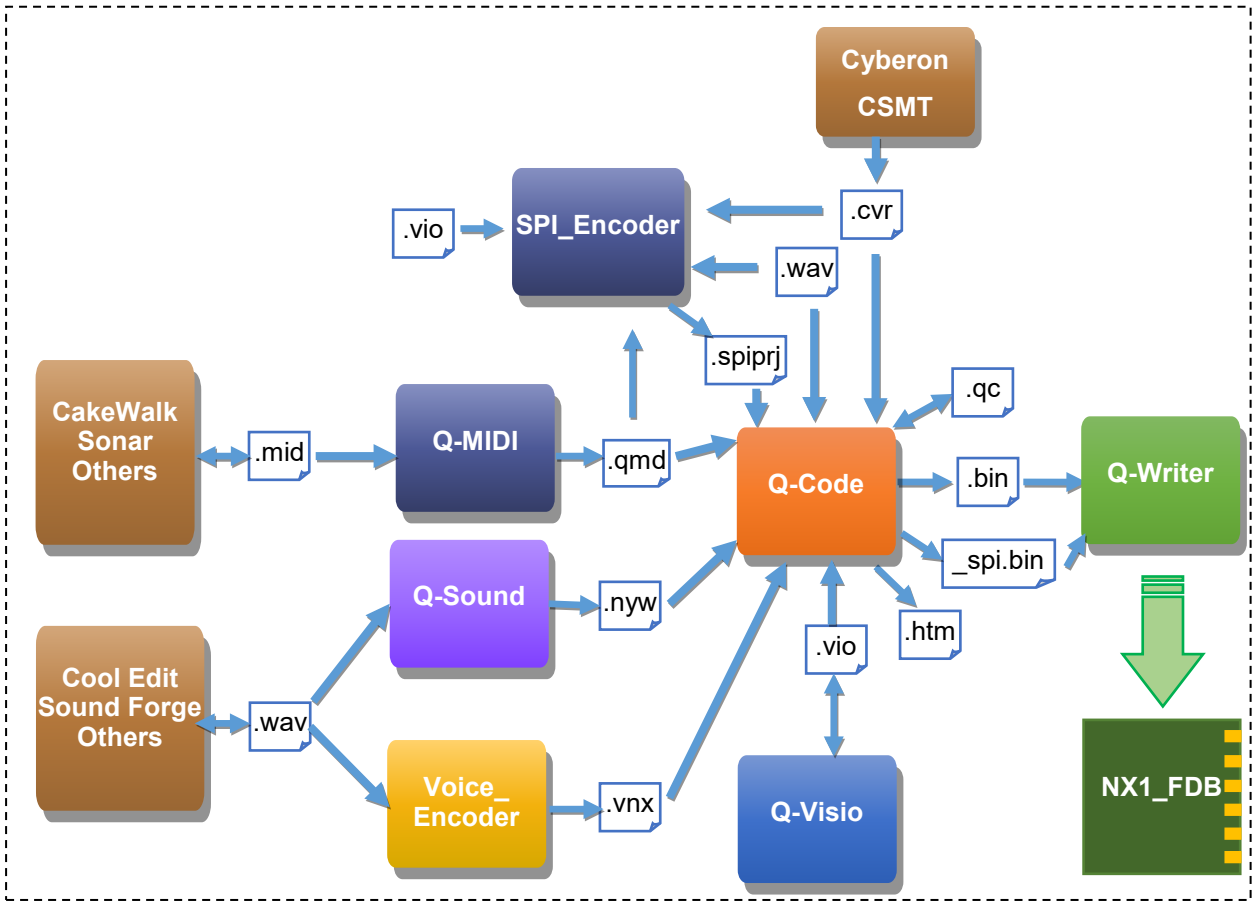
6.13.5 NY7 开发流程图



6.13.6 NY9T 开发流程图

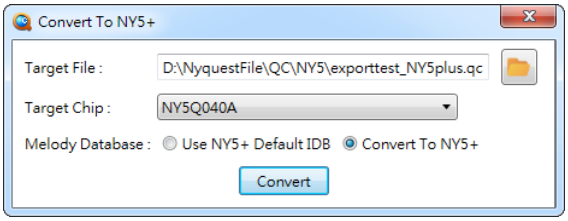


6.13.7 NX1 开发流程图



6.14 Convert To N5+ 说明

按下 Convert To N5+ 会出现下列视窗。



Target File（文件路径）：设定文件路径。

Target Chip（目标芯片）：设定转换的 IC。尽量选择脚位数目和 ROM Size 相近的 IC，可以避免转换后功能的误差。

Melody Database：选择 .md2 转换成 .qmd 要使用哪一种 IDB，可直接使用目前 .md2 的 IDB 或选择使用 NY5+预设的 IDB。

转换成功后，系统会自动开启 .qc 并用预设的浏览器显示下图的转换报告。

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 才支持此功能。

7 改版记录

版本	日期	内 容 描 述	修正页
2.0	2012/01/02	1. 整合 NY4 / NY5 Q-Code 用户手册。 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) 算数逻辑指令 – 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	内 容 描 述	修正页
		(2) 条件跳转指令 – 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) 路径指令 – 新增 Break 指令。 – 新增 StopFG 指令。 – 新增 Label(Pathname) 指令。 (5) 语音指令 – 新增 PlayVS 指令。 – 新增 WaitVN 指令。 (6) 红外线指令	

版本	日期	内 容 描 述	修正页
		– 新增 TX(Ri:Rj:Rk:Rl）指令。 (7) 时间延迟指令 – 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) 动作指令 – 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) 算数逻辑指令 – 新增 Ri.n = Rj.n 指令。	

版本	日期	内 容 描 述	修正页
		– 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) 条件跳转指令 – 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	内 容 描 述	修正页
		– 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) 路径指令 – 新增 Break 指令。 – 新增 StopFG 指令。 – 新增 Label(Pathname)指令。 (5) 语音指令 – 新增 PlayVS 指令。 – 新增 WaitVN(Ch)指令。 (6) 音乐指令 – 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) 红外线指令 – 新增 TX(Ri:Rj:Rk:Rl）指令。 (8) 时间延迟指令 – 新增 Delay(Ri:Rj:Rk) 指令。 – 新增 WaitDN(n) 指令。	

版本	日期	内 容 描 述	修正页
		– 新增 StopD(n) 指令。 – 新增 PauseD(n) 指令。 – 新增 ResumeD(n) 指令。 (9) 动作指令 – 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. 附录 (1) 新增相关工具介绍。 (2) 新增 *Q-Code* 基本功能介绍。 (3) 新增 *Q-Code* 应用实例。 (4) 新增在 *Q-Code* 中，使用 *WAVE_Splitter*。 (5) 修改 *Q-Code* 指令表。 (6) 修改 RAM 资源使用说明。 (7) 修改 Ri 与 Xi 对应表。 (8) 修改 *Q-Code* 开发流程图。	

版本	日期	内 容 描 述	修正页
2.1	2012/01/31	1. 增加 ASM 使用说明。 2. 修正 NY4 *Q-Code* RAM 资源使用说明。 3. 修正 NY5 *Q-Code* RAM 资源使用说明。	53, 131 258 260
2.2	2012/03/23	1. WDT_Clear 指令更名成 WDT_CLR。 2. 修正 Interrupt Path 中，所禁用的指令。 3. 修正 NY4 *Q-Code* RAM 资源使用说明。 4. 修正 NY5 *Q-Code* RAM 资源使用说明。	98, 198 137 258 260
2.3	2012/05/31	1. 新增寄存器 PlayVS 播放指令。 2. 新增寄存器 PlayTS 播放指令。 3. 新增寄存器 PlayMS 播放指令。 4. 增加 Table PlayV / PlayVS 范例说明。 5. 增加 Table PlayT / PlayTS 范例说明。 6. 增加 Table PlayM / PlayMS 范例说明。 7. 更换 IC Body 图示。 8. 修改语音段落说明。 9. 修改在 *Q-Code* 中如何播放 *Q-Sound* 处理后的文件说明。 10. 移除 Multi-channel PlayV 指令。 11. 增加 Audio_ON / Audio_OFF 注意事项。 12. 修改在 *Q-Code* 程序中使用 *Q-Sound* 说明。	79, 160 167 170 80, 161 168 171 33, 102 35, 106 37, 108 160 197 248
2.4	2012/08/30	1. 修改执行（Run）菜单内容。 2. 修改 ICE/Romter 连接状态（ICE / Romter Connect Status）内容。 3. *Q-Sound* 取代 *WAVE_Splitter* 在语音段落的功能。 4. 修改音乐文件段落加入 / 删除文件图示。 5. 修改乐器段落加入 / 删除文件图示。 6. 新增 Configure Download 功能。 7. 修改按键侦测时间选项。 8. 新增特殊路径 — 主循环 9. 新增一般指令 — Debounce(Time)指令。 10. 修改 NY4 RAM 资源使用说明。 11. 修改 NY5 RAM 资源使用说明。 12. 修改 Q-Code 开发流程图。	26 31 35, 106 110 126 267 35, 105 57, 136 97, 138 258 260 268

版本	日期	内 容 描 述	修正页
2.5	2012/11/01	1. 新增支持 NY5AxxxB、NY5BxxxB IC 系列。 2. 新增系统内频可选 1 或 2 MHz 功能。 3. NY5AxxxB IC 新增支持 PWM Current 功能。 4. 变更 FrameRate 为 Action_FrameRate。	102 103 - 34, 105
2.6	2012/11/30	1. NY4 选项部分 (1) 新增 Action Compression 功能。 2. NY4 段落部分 (1) 修改 Action File 接口。 (2) 新增 Action 输出参数设定。 3. NY4 指令部分 (1) IO 指令 – Px=[1 X 0 FD An Q] 指令新增 Action 参数。 (2) Action 指令 – 新增多通道信号播放功能。 4. NY5 选项部分 (1) 新增 Action Compression 功能。 5. NY5 段落部分 (1) 修改 Action File 接口。 (2) 新增 Action 输出参数设定。 6. NY5 部分 (1) IO 指令 – Px=[1 X 0 FD An Q] 指令新增 Action 参数。 (2) Action 指令 – 新增多通道信号播放功能。 7. 修改 RAM 资源使用说明。	 34 38 46 73 93 105 108 120 154 189 258, 260
2.7	2013/02/27	1. 新增 Time Base 设定。 2. 新增 Action Loop 功能。	103 93, 189
2.8	2014/05/29	新增滚动码指令。	99, 183
2.9	2014/08/21	NY5AxxxA 增加 Tone / ToneS 指令。	173
3.0	2015/02/10	1. 新增 Action 信号输出支持 Drive / Sink 设定。 2. 修改 Random 功能的使用 RAM 数量。	38, 111 259, 261, 263

版本	日期	内 容 描 述	修正页
3.1	2015/05/21	1. 更新界面图片。 2. 更新帮助(Help)菜单。 3. 更新 QIO Custom 程序范例。 4. 修改主循环、4 毫秒和中断路径上可使用的指令类型。 5. 新增 IR_RX 路径。 6. 新增[Ri,Rj,Rk,Rl] = RX / [Xi, Xj] = RX 到红外线指令。	- 27 56, 133 57, 58, 136, 136 59, 138 90, 135
4.0	2015/08/28	1. 合并 NY7。 2. 新增串行数据接收（Serial Control）。 3. 移除 StopDelay。 4. 修改 RX / TX 指令为 IR_RX / IR_Tx。 5. 更新 SDelay 时间范围。 6. 修改 ReadTempo 说明。	- 136, 161, 165, 170 - - 140 117
4.1	2015/11/25	1. 移除 Debounce(Time)指令。 2. 修改上电触发说明。 3. 修改按键侦测时间说明。 4. 新增矩阵扫描设定时间说明。 5. 新增 Ri 与 Xi 对应说明。 6. 新增 SPIVol 指令说明。 7. 修改 Instrument(Ch, i)指令说明，支持变量控制 GM 音色。 8. 修改 InstNoteOn 及 DrumNoteOn 指令说明。 9. 新增 Gliss(Note, Semitone, Time)指令说明。 10. 修改 Ri 与 Xi 对应表。	- 39 40 40 77 110 114 124, 125 125 174
4.2	2016/02/02	1. 更新 Shortcut icon。 2. 更新 NY7 上拉电阻说明。 3. NY4 不支持 vol 指令。 4. 更新 Background2 描述。	24, 25 48 82, 126 76

版本	日期	内 容 描 述	修正页
4.3	2016/05/31	1. 修改按键扫描间隔说明。 2. 新增最大琴键音数说明。 3. 更新 I / O 段落说明。 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. 修改查表指令说明。 8. 新增脉冲调变 IO 指令说明。 9. 新 增 Direct_Debounce(Time) 、 Matrix_Debounce(Time) 及 Debounce(Time)指令。 10. 更新 NY7 Q-Code 指令表说明。 11. 更新 NY4 / NY5 RAM 资源使用说明。	41 42 50 86, 87 107 112 131 140 155 170 174, 175
4.4	2016/08/30	1. 编译(Complie)新增重新编译(Rebuild)功能。 2. 更新输入状态段落(Input State)说明。	25 54
5.0	2016/11/30	1. 移除 NY5AA。 2. 新增 MaxSingleNote 指令。 3. 合并 NY9T。	- 131 187, 194, 200, 203
5.1	2017/02/24	1. 新增 QFID 功能。 2. 新增 Action Mark。	52, 163 69, 164
5.2	2017/05/22	1. 新增 Export Project 功能说明。 2. 新增 RFC。 3. 修改变量运算说明。	23 48, 165 90
5.3	2017/08/21	1. 修改 Conditional Jump 指令说明。 2. 新增 Xi = V_Chx_Vol /Ri = SPIVo / I Ri = M_Chx_Vol 指令。 3. 新增 Enforce_Calibrate_Normal / Enforce_Calibrate_Sleep 指令，并移除 Enforce_Calibrate 指令。	92 120, 124, 129 143, 143

版本	日期	内 容 描 述	修正页
5.4	2017/11/27	1. 新增 IR CRC 说明。 2. 更新 PWMIO 说明。 3. 更新 RFC 说明。 4. 新增 QFID Slow。 5. 更新 MixCtrl 指令说明。 6. 更新 PWM 指令说明。 7. 新增指令 QFID_SlowOn / QFID_SlowOff。 8. 新增指令 RFC_On / RFC_Off。 9. 更新 Audio_OFF 指令说明。	42 48 49 54 141 156 166 167 173
5.5	2018/02/27	1. 更新 QFID 段落。 2. 新增擦除完成（SPI_EraseEnd）指令。 3. SPIReadIndex 指令改为 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 说明。 2. 更新 Mute_On / Mute_Off 指令说明。 3. 新增 HoldA 指令。	48 144, 145 176
5.7	2018/08/30	1. 更新 Scankey Interval 选项说明。 2. 修改指令说明格式。	47 -
5.8	2018/12/07	1. 合并 NY6、NX1。	-
5.9	2019/2/27	1. 更新操作画面。 2. 更新 Action Mark State 指令说明。	- 353
6.0	2019/05/31	1. 新增 VR Timeout Extend 选项。 2. 新增 VR_VAD?Path / AGC_On / AGC_Off 指令。 3. 更新 PGA_Gain 指令说明。	75 115, 220, 220 177
6.1	2019/08/29	1. 更新 PGA_Gain 指令说明。 2. 更新 SDelay 指令说明。 3. 更新 Debounce(time)指令说明。	174 198 221

版本	日期	内 容 描 述	修正页
6.2	2019/11/22	1. 更新 FD 说明。 2. 更新 Factor 说明。 3. 更新 WaitSN 说明。 4. 更新 M_CHx_Vol 说明。	44 50 139 161
6.3	2020/03/20	1. 修改编辑接口。 2. 新增侦错模式。 3. 修改信息窗口。 4. 修改随机产生器选项说明。 5. 修改 Debounce / PowerOnTrigger 选项说明。 6. 修改 Before_PowerOn 说明。 7. 新增 SBC_Loop_On / SBC_Loop_Off 指令。 8. 修改 MixCtrl 指令说明。 9. 新增 SFX 指令。 10. 新增 Real Time Play 指令。 11. 修改 NX1 声音输出通道说明。	21 22 22 46 47 - 139 178 209 214 259
6.4	2020/06/05	1. 修改 RAM_Usage 选项说明。 2. 新增 Variable_Compatible 选项说明。 3. 修改 Record 说明。 4. 修改 Output State 说明。 5. 修改 Px=[0 1 An Q FD…] 说明。 6. 新增 [Px.n Px.n…] = [0 1 Q…] 指令。 7. 新增 Ri=VR_Loading 指令。 8. 修改 ReadRollingCode 指令说明。	50 51 57 73 125 126 210 230
6.5	2020/8/31	1. 新增 LED String 段落。 2. 修改 Touchkey 说明。 3. 修改 SPI Flash 说明。 4. 修改 I/O 段落说明。 5. 修改语音识别段落说明。 6. 新增 WaveID_RX / VT_BeforeRecord / VT_BeforeTraining / VT_AddTagFail 路径。 7. 新增 MIDI_Pitch / LongInst_Holdtime / ShortInst_HoldTime 指令。 8. 修改键盘指令说明。	63 38 43 67 79 106, 109 178, 182 179, 180, 181

版本	日期	内 容 描 述	修正页
		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 图片。	
		2. 更新 Record 段落说明。	61
		3. 新增 ADPCM_Loop_On / ADPCM_Loop_Off 指令。	147
		4. 新增 Mask_On / Mask_Off 指令。	178, 179
		5. 新增琴键录音功能。	182 - 184
6.7	2021/01/18	新增 Melody_OKON_BgNote 选项。	53
6.8	2021/05/20	新增 NY5+系列。	-
6.9	2021/09/30	1. NY5+移除 PauseS / ResumeS / StopS / WaitSN 指令支持。	-
		2. 加入 NX1 EF series。	-
		3. NY5+支持 **[QIO Custom]** 段落。	101
7.0	2021/11/30	1. 更新 Voice Output 选项说明。	41
		2. 更新 **[Record]** 段落说明。	70
		3. 更新 **[I/O]** 段落说明。	76
		4. 更新 LVD_mVn 路径说明。	117
		5. 更新 Ri = LVD 指令说明。	272
		6. 新增 Robot4 / RT_Robot4。	251, 256
7.1	2022/2/25	1. 新增 UART / I2C 说明。	55
		2. 新增 **[PWM-IO]** 段落说明。	87
		3. 更新 LVD_mVn 路径说明。	117
		4. 移除 LVD_LVn 路径说明。	-
		5. 修改指令支持说明。	
		6. 新增 UART / I2C 指令。	222
		7. 更新 PWM 指令说明。	225
		8. 新增 RT_Darth / RT_Darth_Off。	258

版本	日期	内 容 描 述	修正页
7.2	2022/5/31	1. 修改 FD 选项说明。	55
		2. 新增听声辨位功能。	61, 119, 242
		3. 更新 LVD_mVn / LVD_Max 系统路径说明。	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 指令说明。	267
		8. 更新 Ri=LVD 指令说明。	282
7.3	2022/8/31	1. 更新 Oscillator NX1 说明。	9
		2. 更新 SPI Flash 说明。	50
		3. 更新 I2C 说明。	56
		4. 新增 LVD 说明。	60
		5. 更新 **[Record]** 说明。	72
		6. 新增 Recorded?Path 指令。	139
		7. 更新 PausePWM / HoldPWM 指令说明。	234, 235
7.4	2022/11/30	1. 新增 Audio Filter。	68
		2. 新增 Common ROM Code。	69
		3. 更新 **[Record]** 说明。	76
		4. 新增 **[QIO]**。	94
		5. 新增 **[Interrupt]** 。	117
		6. 新增 Q-Code 特殊路径。	119
		7. 更新 PWMDuty 指令说明。	246
		8. 更新音效指令说明。	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 内容。	6
		2. 更新 Sound Localization 内容。	65
		3. 更新 Realtime Play 内容。	67
		4. 新增 Sound Effect。	68
		5. 更新 QIO 内容。	96
		6. 新增 V_Chx_Freq 指令。	174

版本	日期	内 容 描 述	修正页
		7. NX1 新增支援 SPI Flash Command。	84
		8. 修改 Gliss 指令说明。	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 与 NY5+音量阶数对映。	349
		5. I2C 新增 NX1。	49, 241
7.7	2023/8/31	1. 修改 SPI 相关指令范例。	-
		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	内 容 描 述	修正页
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 参数。 3. NX1 支持 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 EstimateMicVol 指令。 7. NX1 支持 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。	115, 328 249 349
8.3	2025/05/29	1. 更新简介。 2. 更新 Touch Key。	24 50

版本	日期	内 容 描 述	修正页
		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 指令更名为 BreakFG。	208
		7. 修改 Delay 指令内容。	304
		8. Action 指令修改 NX1 支持通道范围。	307
		9. 新增 Srand 指令。	362
8.4	2025/08/29	1. 更新功能菜单	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 循环。	198
		9. SBC / ADPCM / ADM / PCM / MIDI loop 指令标示为过期。	224 ~ 226, 268
		10. 新增 IR_Carrier_On / IR_Carrier_Off 指令。	293
		11. 新增 Audio_Loop 功能。	359