

NUC970/N9H30 NuWriter 使用手冊

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 概述..... 4

 1.1 NAND 開機..... 6

 1.2 SPI 開機..... 7

 1.3 eMMC 開機 8

 1.4 USB 開機..... 9

2 安裝NuWriter驅動程式..... 10

3 USB ISP 模式..... 14

4 NuWriter 工具..... 15

 4.1 DDR/SRAM 模式..... 16

 4.1.1 操作方法..... 16

 4.2 NAND Flash模式 17

 4.2.1 Image檔案型態 17

 4.2.2 操作方法..... 25

 4.3 SPI 模式..... 29

 4.3.1 Image檔案型態 29

 4.3.2 操作方法..... 30

 4.4 eMMC 模式 32

 4.4.1 Image檔案型態 32

 4.4.2 操作方法..... 34

 4.5 Pack 模式 36

 4.5.1 操作方法..... 38

 4.5.2 製作和燒錄pack範例 43

 4.6 MTP模式 (N9H30 不支持此模式)..... 45

 4.6.1 操作方法..... 45

 4.7 Mass Production 模式..... 46

 4.7.1 操作方法..... 46

5 NuWriter 源碼介紹 48

6 韌體介紹(xusb.bin) 53

7 Revision History 56

1 概述

新唐科技NUC970/N9H30系列晶片支援下列四種開機方法：

1. eMMC 開機
2. SPI Flash 開機
3. NAND Flash 開機
4. USB ISP 開機

以上四種是依據power-setting (PA0 and PA1) 去做選擇。NuWriter工具能幫助使用者透過USB ISP模式，將Image檔案放入儲存體中，例如：eMMC 設備，SPI Flash設備或 NAND Flash設備。

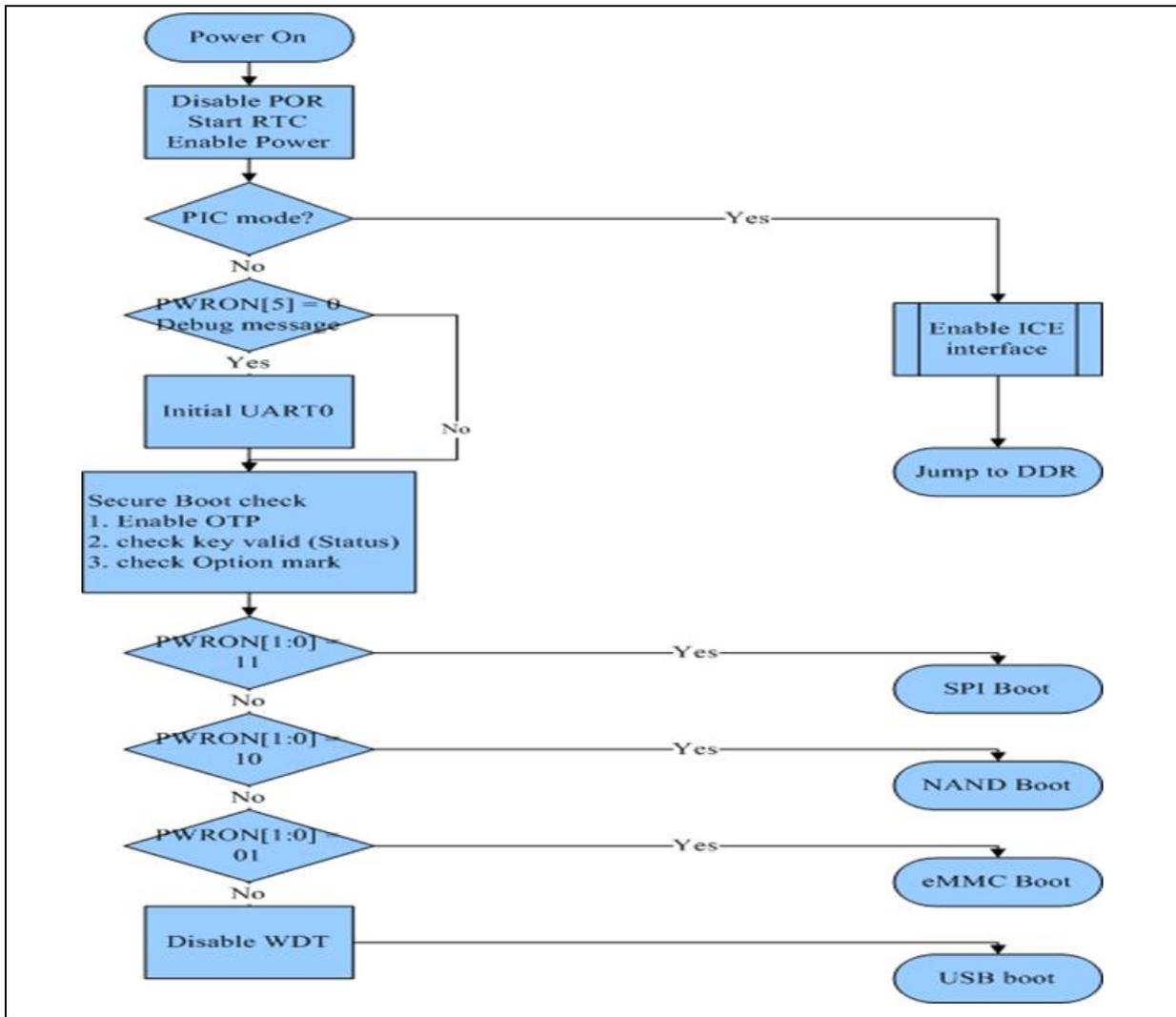


Figure 1-1 Startup Flow

上圖為NUC970/N9H30系列晶片開機流程，開機時透過power-on setting來決定禁用或使能，例如看門狗定時器禁用/使能，定時器禁用/使能，時鐘源的選擇和JTAG介面禁用/使能等，設定方法如下表：

Power-On Setting 端口	描述	Power-On Setting 寄存器
USB0_ID	USB端口0角色選擇 0 = USB端口0作為一個USB主機。	PWRON[16]

	1 = USB端口0作為一個USB設備。	
PA[1:0]	開機源選擇 00 = USB開機。 01 = eMMC開機 10 = NAND Flash開機。 11 = SPI Flash開機。	PWRON[1:0]
PA.2	系統時鐘源選擇 0 = 系統時鐘源選擇 12 MHz 晶振。 1 = 系統時鐘源選擇 UPLL 輸出。	PWRON[2]
PA.3	看門狗定時器 禁用/使能 0 = 在上電之後看門狗定時器禁用。 1 = 在上電之後看門狗定時器使能。	PWRON[3]
PA.4	JTAG 介面禁用/使能 0 = PJ[4:0] 為通用I/O端口。 1 = PJ[4:0] 為JTAG介面。	PWRON[4]
PA.5	UART 0 輸出除錯訊息使能/禁用 0 = UART 0輸出除錯訊息使能。 1 = UART 0輸出除錯訊息使能 禁用。	PWRON[5]
PA[7:6]	NAND Flash 的頁大小(Page size)選擇 00 = NAND Flash 頁大小選擇2KB。 01 = NAND Flash頁大小選擇4KB。 10 = NAND Flash頁大小選擇8KB。 11 = 保留。	PWRON[7:6]
PA[9:8]	NAND Flash ECC 選擇 00 = NAND Flash ECC選擇BCH T12。 01 = NAND Flash ECC選擇BCH T15。 10 = NAND Flash ECC選擇BCH T24。 11 = 保留。	PWRON[9:8]

Table 1-2 Power On Setting

完成後，將進入四種開機方法中的其中一種。

1.1 NAND 開機

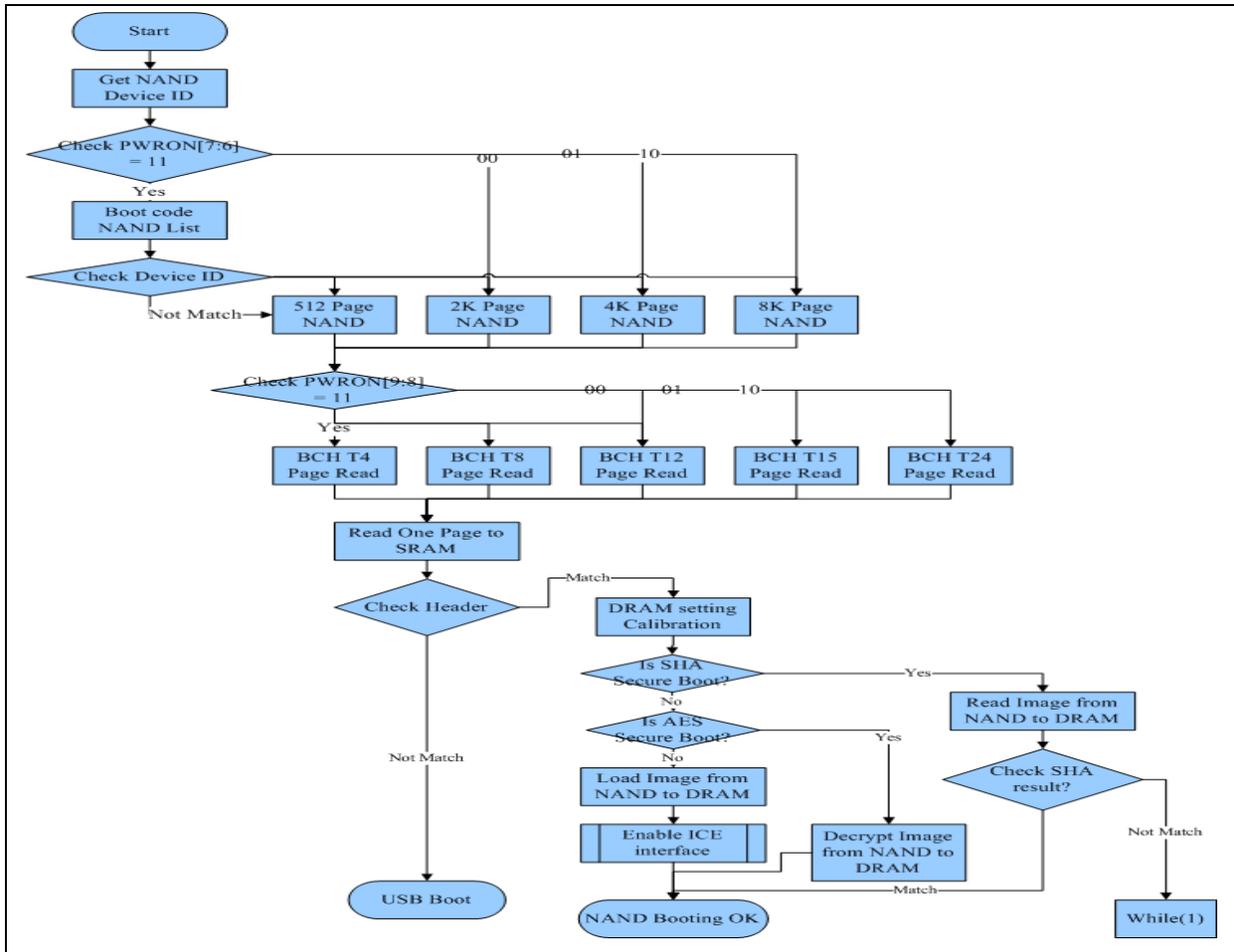


Figure 1-2 Nand Startup Flow

NAND開機流程偵測PWRON[7:6]確定Page Size，接著偵測PWRON[9:8]確定NAND Flash ECC Type，之後直接讀取NAND Flash中Block0確定檔頭中的Boot Code Marker是否正確，Boot Code Marker可以參考4.2.1.1uBoot型態，如果不正確繼續往下一個Block1來確認檔頭中的Boot Code Marker是否正確，當Block0、Block1、Block2、Block3都找不到檔頭(Header)中的Boot Code Marker時，則跳到USB開機。當檔頭偵測正確，接著做DDR初始化，判斷SHA是否使能，使能則計算SHA並且確認是否正確，否則判斷AES加密，如果有加密則作解密沒有則跳過解密，之後將剛剛偵測到的uBoot中存放的Image複製到對應的DDR的位置，並且跳過去執行，即完成NAND開機的流程。注意：uBoot中存放Image加上檔頭和DDR參數的總大小不能超過一個Block大小，即uBoot Image + Header + DDR Parmater ≤ Block Size。

注意: N9H30 不支援加密模式。

1.2 SPI 開機

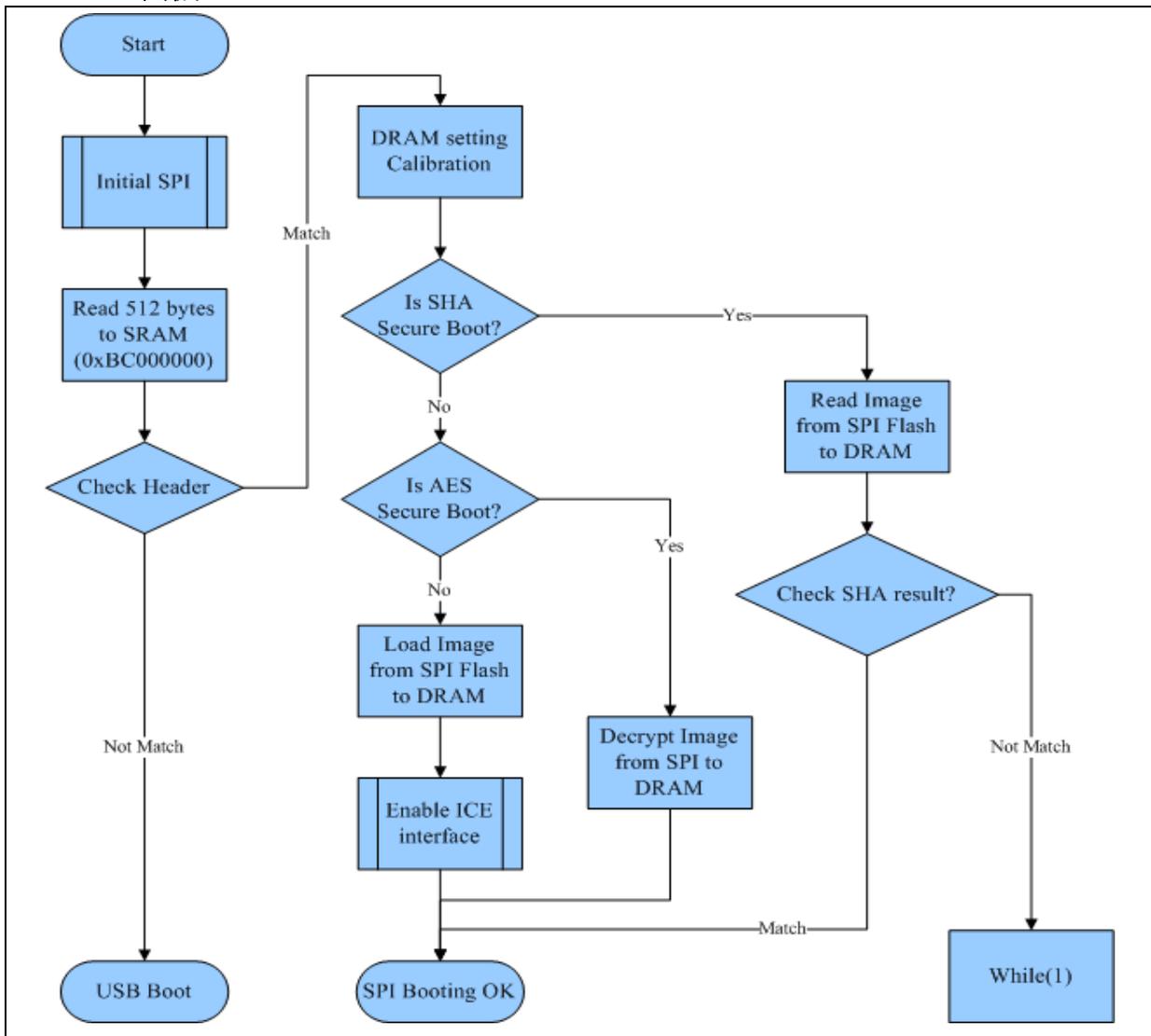


Figure 1-3 SPI Startup Flow

SPI開機後直接讀取SPI Flash中0x00000000位置來確定檔頭中的Boot Code Marker是否正確，Boot Code Marker可以參考4.3.1.1uBoot型態，如果不正確則跳到USB開機。當檔頭中的Boot Code Marker偵測正確，接著做DDR初始化，判斷SHA是否使能，使能則計算SHA並且確認是否正確，否則判斷AES加密，如果有加密則作解密沒有則跳過解密，完成後將剛剛偵測到的uBoot中存放的Imag複製到對應的DDR的位置，並且跳過去執行，即完成SPI開機的流程。

注意: N9H30 不支援加密模式。

1.3 eMMC 開機

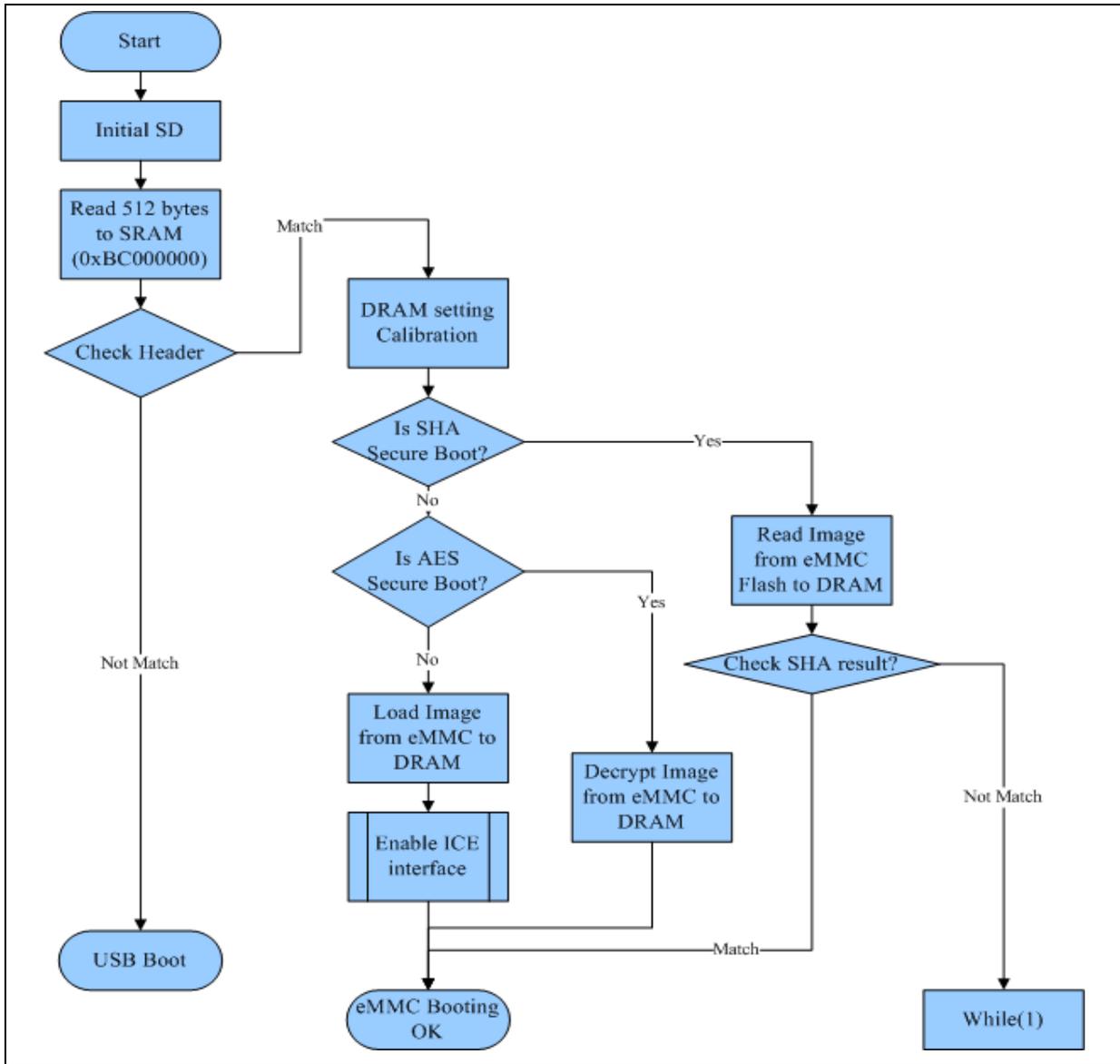


Figure 1-4 eMMC Startup Flow

eMMC開機後直接讀取eMMC中0x00000400位置來確定檔頭中的Boot Code Marker是否正確，Boot Code Marker可以參考4.4.1.1uBoot型態，如果不正確則跳到USB開機。當檔頭中的Boot Code Marker偵測正確，接著做DDR初始化，判斷SHA是否使能，使能則計算SHA並且確認是否正確，否則判斷AES加密，如果有加密則作解密沒有則跳過解密，完成後將剛剛偵測到的uBoot中存放的Imag複製到對應的DDR的位置，並且跳過去執行，即完成eMMC開機的流程。

注意: N9H30 不支援加密模式。

1.4 USB 開機

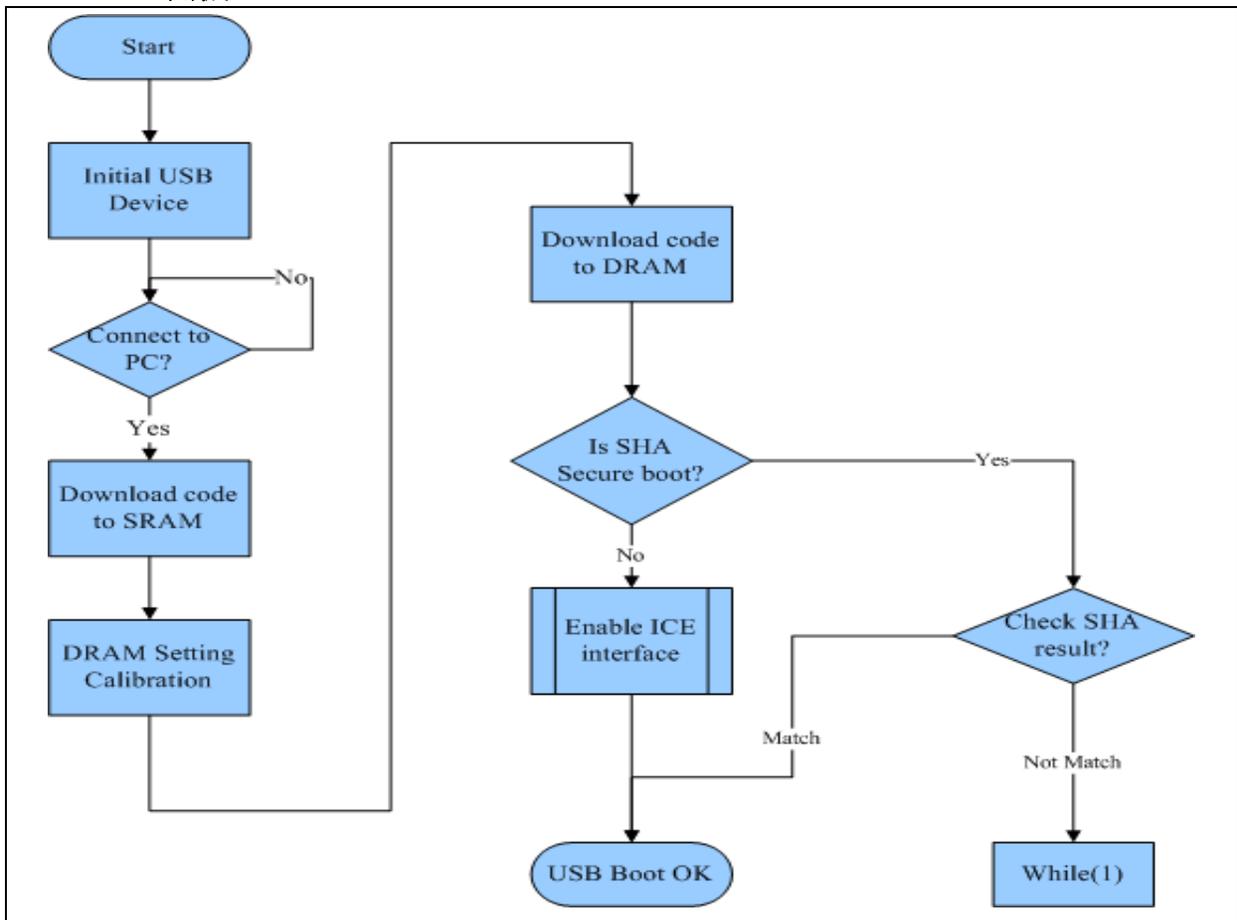


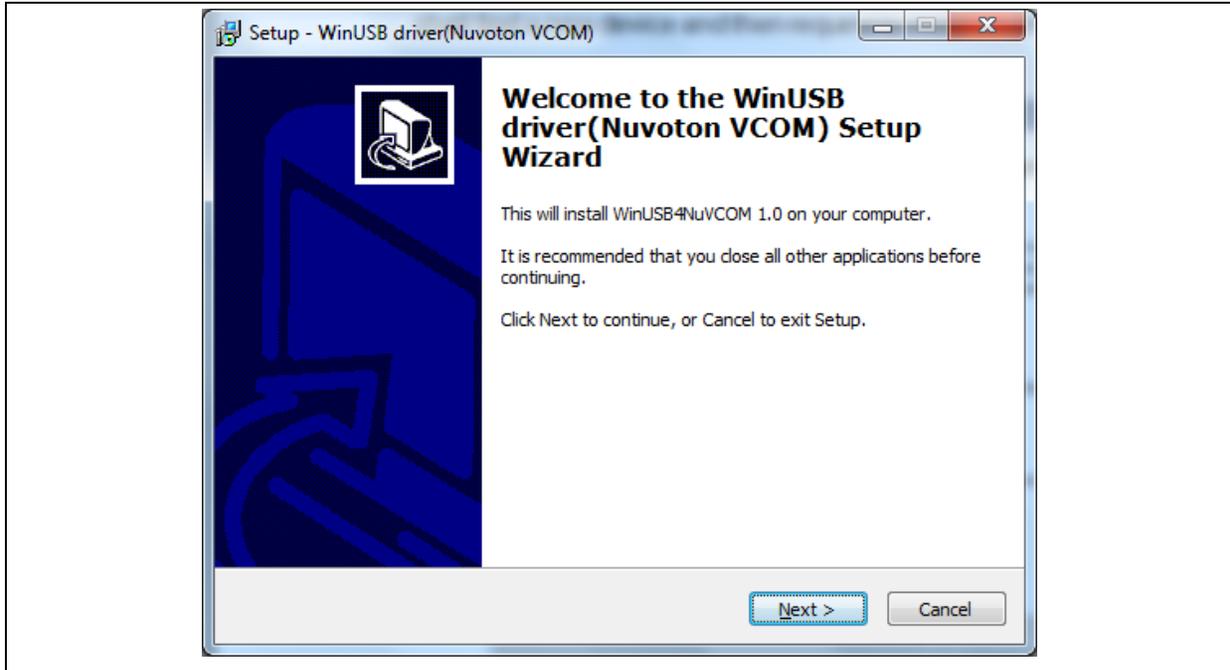
Figure 1-5 USB Startup Flow

USB開機後第一個動作會等待DDR參數初始化完成，所以會等待PC端的NuWriter連上NUC970/N9H30並且等待NuWriter傳送DDR參數做初始化，完成後馬上會再傳送xusb.bin並且跳過去執行，此時xusb.bin運作起來會與PC端的NuWriter溝通。

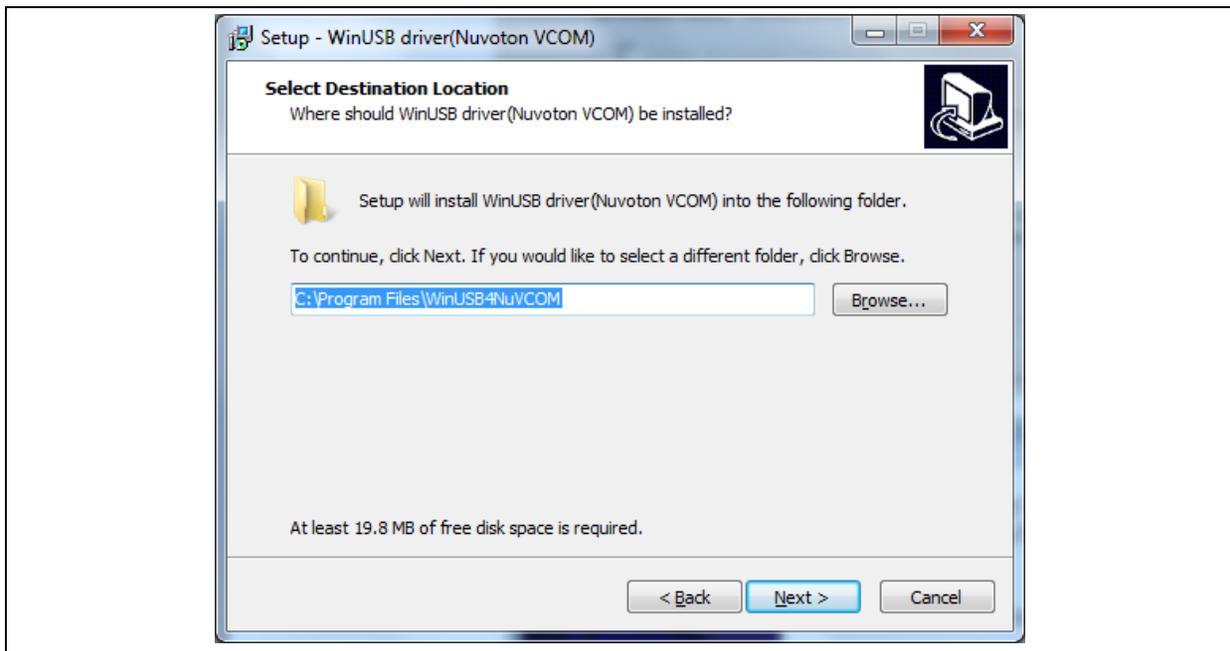
2 安裝NUWRITER驅動程式

NuWriter 必須在電腦中安裝 VCOM 驅動程式才能使用 NuWriter 工具。請依據下列步驟來安裝 WinUSB4NuVCOM 驅動程式：

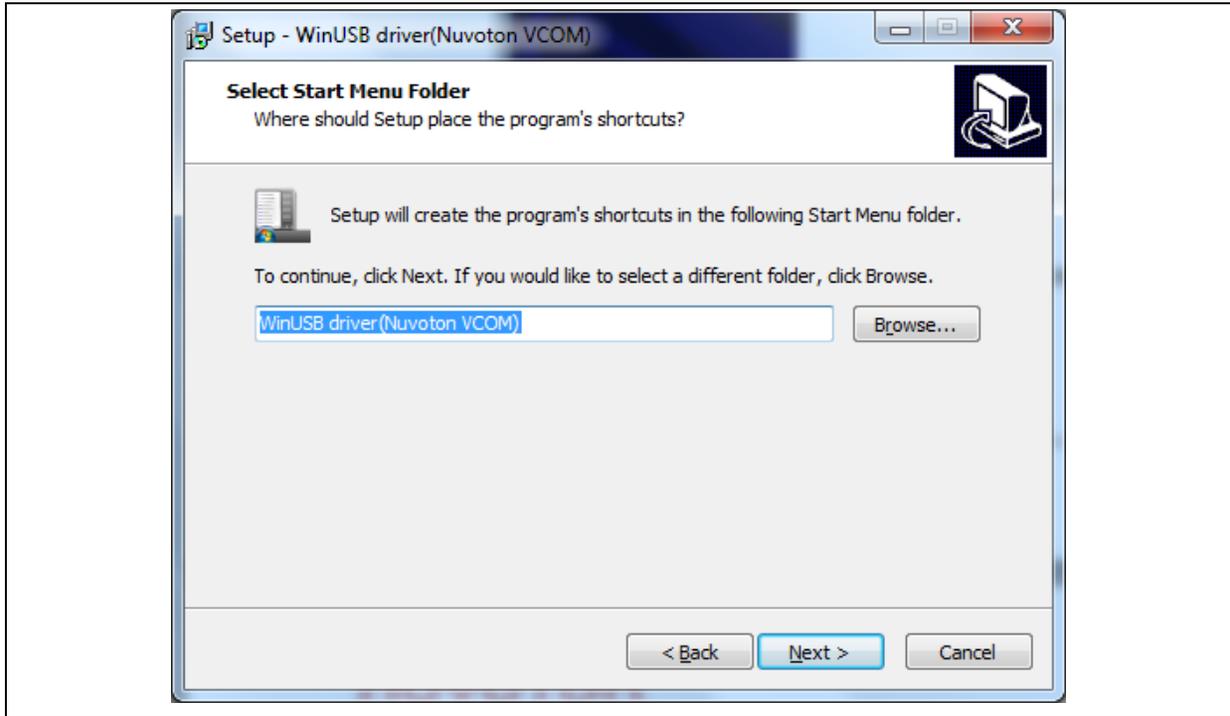
將電腦與 NUC970/N9H30 系列晶片透過 USB cable 連接起來後。在電腦中執行 WinUSB4NuVCOM.exe 開始安裝驅動程式。開啟 NUC970/N9H30 系列晶片的電源之後，Windows 會發現新的設備，然後會要求你安裝驅動程式。



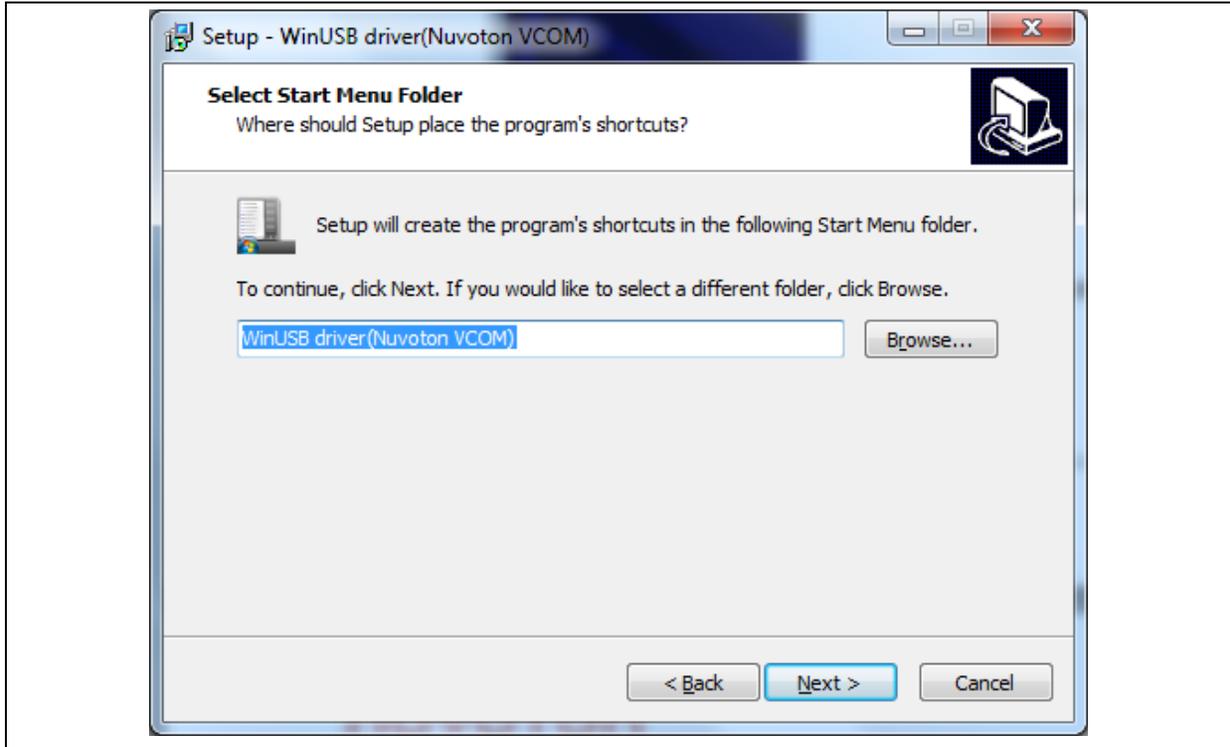
按下 “Next”。這個畫面告訴你即將要安裝 WinUSB4NuVCOM 1.0 驅動程式。如下圖：



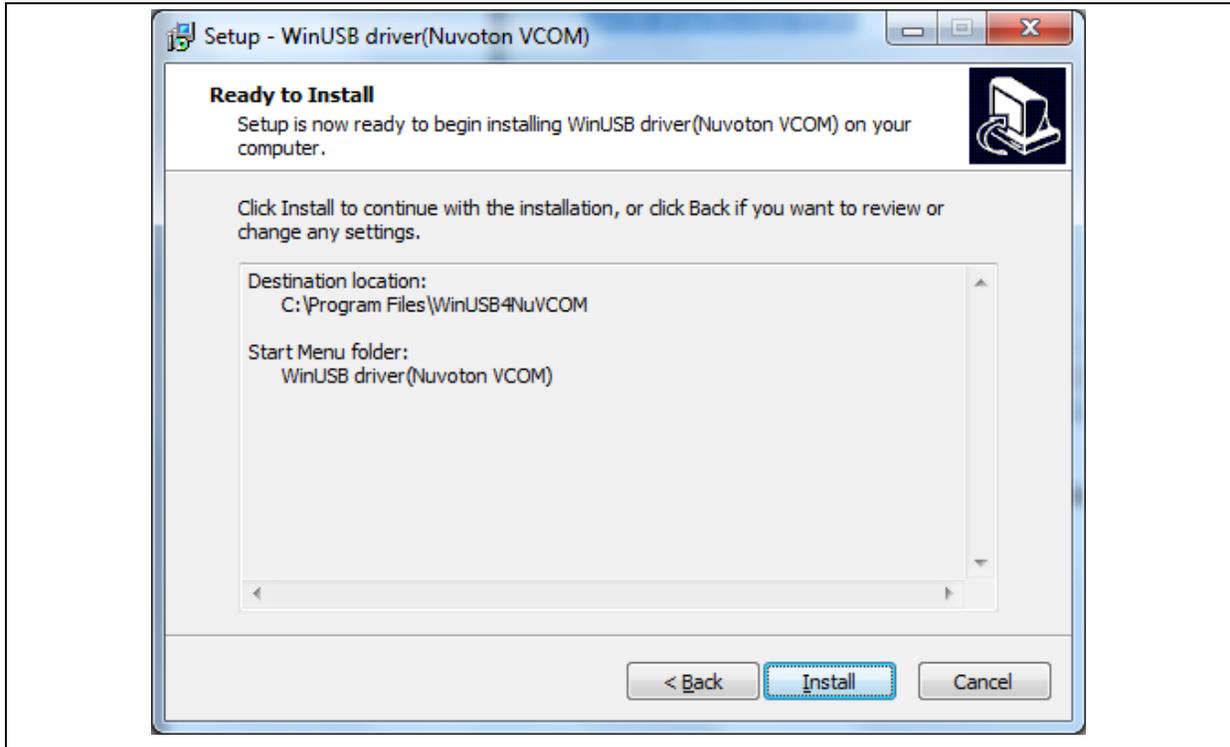
選擇使用者想要安裝的路徑或使用預設的路徑，確定以後按下 “Next”。如下圖：



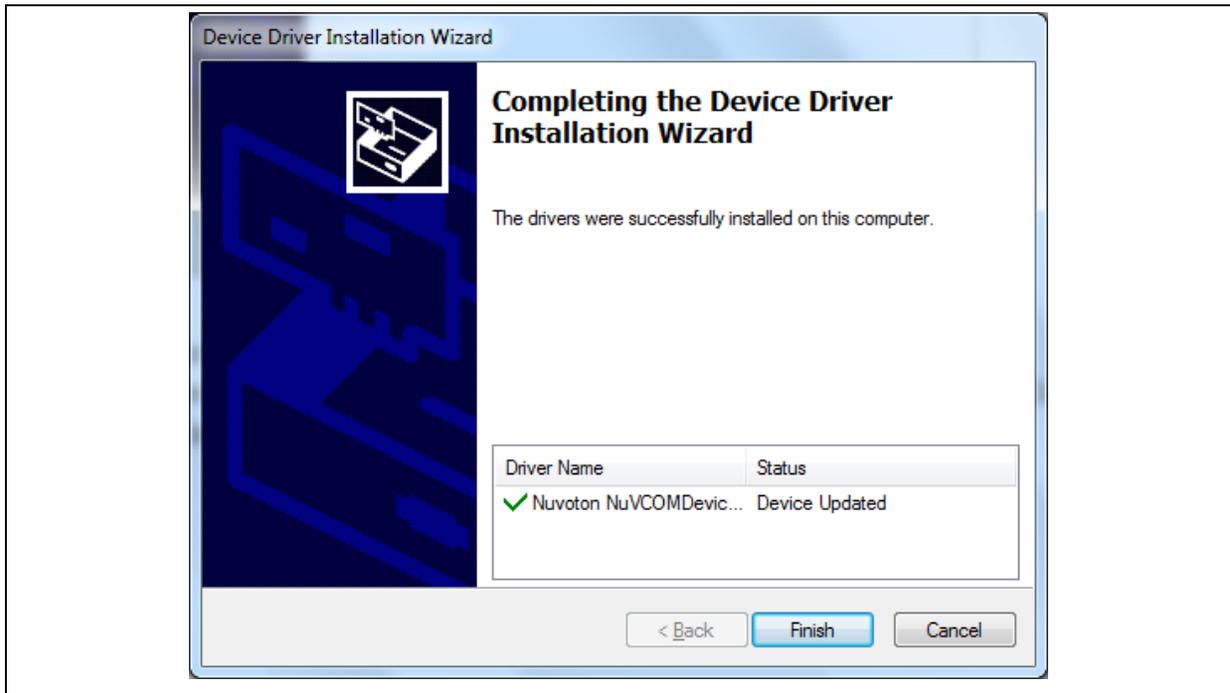
按下“Next”。如下圖：



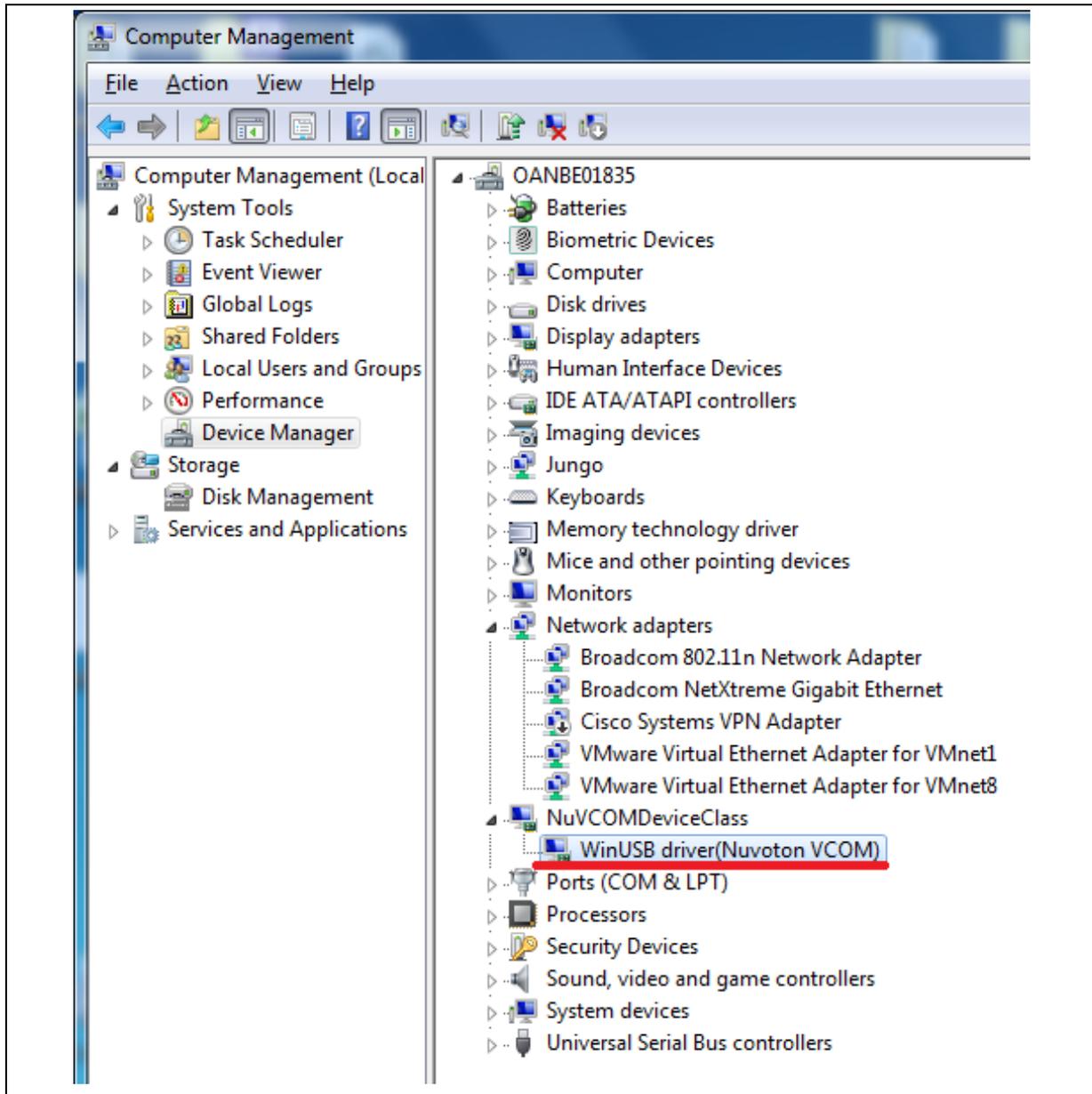
按下“Install”。如下圖：



按下“**Finish**”，完成VCOM驅動程式的安裝。如下圖：



如果VCOM驅動程式是安裝成功，可以在“Device Manager”中看到“WinUSB driver (Nuvoton VCOM)”。如下圖：



3 USB ISP 模式

NUC970/N9H30系列晶片提供Jumpers去選擇開機的方法。選擇USB ISP模式，則PA0和PA1必須設定為High。其他開機設定可以參考如下表：

開機設定	PA1	PA0
USB ISP 開機	Low	Low
eMMC 開機	Low	High
NAND 開機	High	Low
SPI 開機	High	High

開啟NUC970/N9H30系列晶片的電源並且設定為USB ISP模式和電腦上的NuWriter工具，即可開始使用。注意：如果電腦沒有找到WinUSB4NuVCOM驅動程式則NuWriter工具無法使用。

4 NUWRITER 工具

執行“nuwriter.exe”，第一個畫面如下。選擇目前晶片，目前支援NUC970/N9H30系列晶片。如果選擇NUC970/N9H30系列晶片，則必須選擇DDR參數，DDR參數依據NUC970/N9H30系列晶片的Part Number來選擇。

確定手邊的NUC970/N9H30系列晶片後並選擇完成後按下“Continue”，即可開始使用NuWriter工具。

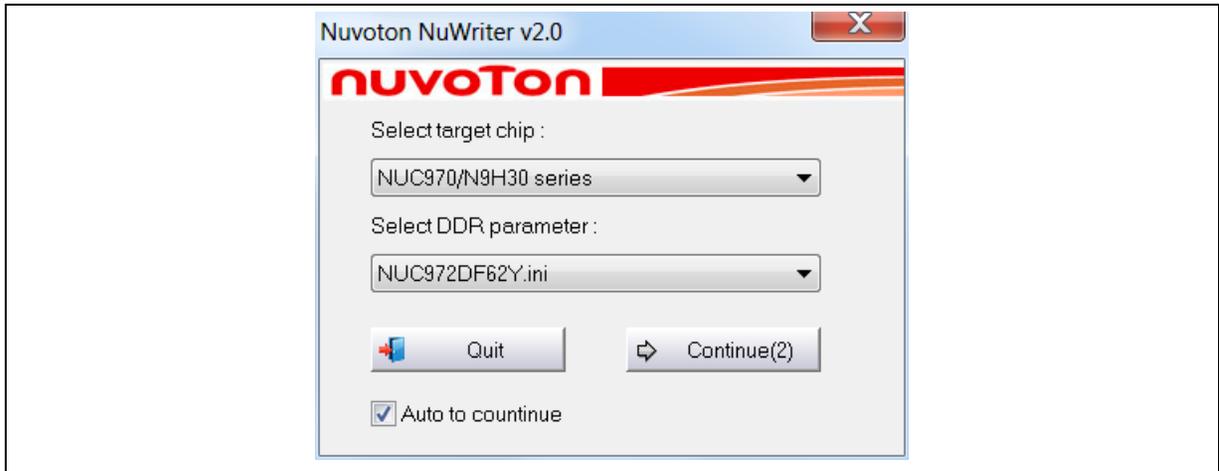


Figure 4-1 NuWriter – Set Chip/DDR

4.1 DDR/SRAM 模式

4.1.1 操作方法

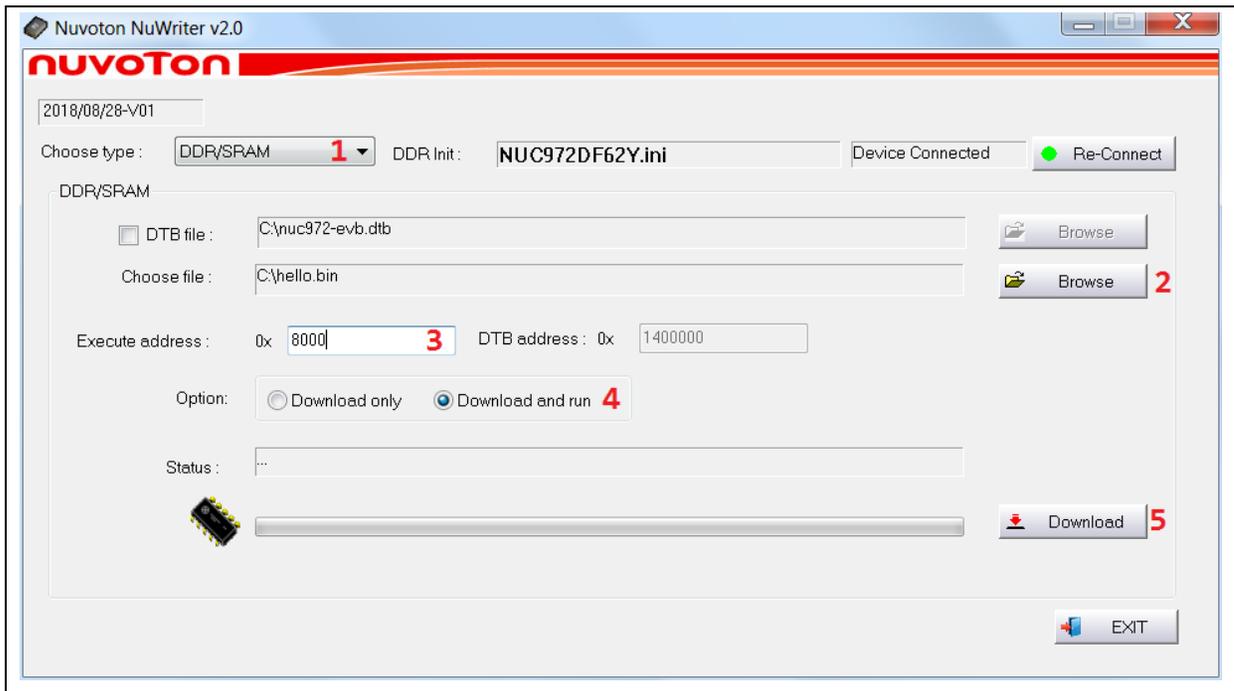


Figure 4-2 NuWriter – DDR/SDRAM mode(1)

依照上圖步驟即可以在DDR/SRAM模式下，完成將Image檔案直接下載到DDR或SRAM記憶體中。操作步驟如下：

1. 選擇“DDR/SRAM”模式。
2. 選擇Image檔案。
3. 輸入Image檔案放在DDR/SRAM的位址。注意：若要傳輸到DDR中,位址必須介於0x00000000~0x01F00000(31MB)。
4. 選擇“Download only”或是選擇“Download and run”。
5. 按下“Download”。

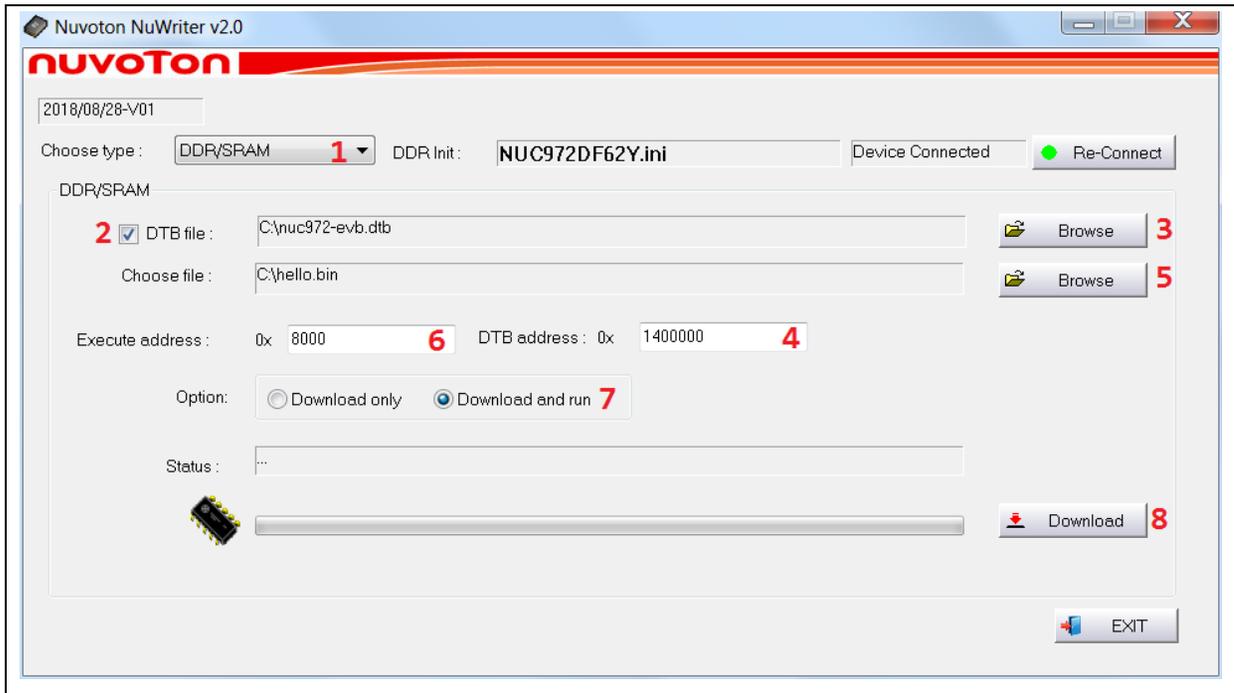


Figure 4-3 NuWriter – DDR/SDRAM mode(2)

依照上圖步驟即可以在DDR/SDRAM模式下，完成將Image檔案和Device tree檔案一起下載DDR或SRAM記憶體中。操作步驟如下：

1. 選擇“DDR/SDRAM”模式。
2. DTB 檔案打開
3. 選擇 DTB檔案
4. 輸入DTB檔案放在DDR/SDRAM的位址. 注意：不要被Image 檔案覆蓋.
5. 選擇Image檔案。
6. 輸入Image 檔案放在 DDR/SDRAM的位址。注意：若要傳輸到DDR 中,位址必須介於 0x00000000~0x01F00000(31MB)。
7. 選擇“Download only”或是選擇“Download and run”。
8. 按下“Download”。

4.2 NAND Flash模式

NAND模式 可以將Image檔案燒入到NAND Flash中，並且將Image檔案型態設定為uBoot、Data、Environment、Pack，四種型態中的其中一種。

4.2.1 Image檔案型態

4.2.1.1 uBoot 型態

主要是作為開機的第一支程式，當NUC970/N9H30系列晶片上電後會讀取uBoot型態中的Image並執行，不限定一定要存放uBoot，一般程式也是可以使用uBoot型態來作為上電後第一支執行程式。uBoot增加16bytes的檔頭(Header)和DDR參數合併，方式如下：

	0x0	0x0	0x0	0x0
0x00	Boot Code Marker	Execute Address	Image Size	Decrypt Address
0x10	DDR - Initial Marker	DDR - Counter	DDR - Address 0	DDR - Value 0
0x20	DDR - Address 1	DDR - Value 1	DDR - Address 2	DDR - Value 2
0x30	DDR - Address 3	DDR - Value 3	DDR - Address 4	DDR - Value 5
0x40	DDR - Address 5	DDR - Value 5	DDR - Dummy	DDR - Dummy
0x50	uBoot			

Figure 4-4 Boot code header

Boot Code Marker = {0x20,'T','V','N'}。

Execute Address = 透過IBR 將uboot 複製到{ Execute Address } 位置。

Image Size = {uBoot 檔案長度}。

Decrypt Address = {0xFFFFFFFF}。

DDR - Initial Marker = {0x55AA55AA}。

DDR - Counter = DDR參數的長度，是計算下圖所選擇DDR參數NUC72DF61Y.ini，可以在 NuWriter\sys_cfg\中找到此檔案。

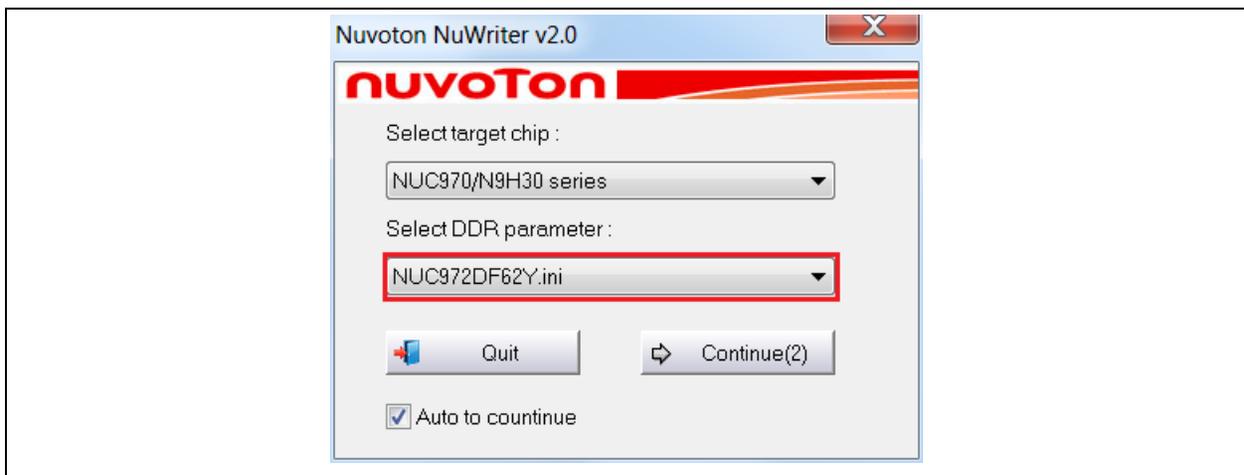


Figure 4-5 NuWriter – Select DDR parameter

舉例NUC72DF61Y.ini內容如下

```
0xB0000220=0x01000000
0xB0000264=0xC0000018
0xB0000220=0x01000018
0x55AA55AA=0x1
0x55AA55AA=0x1
```

```

0xB0001828=0x53DCD84A
0xB0001808=0x00008014
0x55AA55AA=0x1
0x55AA55AA=0x1
0x55AA55AA=0x1
0xB0001800=0x00030476
0x55AA55AA=0x1
0xB0001804=0x00000021
0x55AA55AA=0x1
0xB0001804=0x00000023
0x55AA55AA=0x1
0x55AA55AA=0x1
0x55AA55AA=0x1
0xB0001804=0x00000027
0x55AA55AA=0x1
0x55AA55AA=0x1
0x55AA55AA=0x1
0xB0001820=0x00000000
0xB0001824=0x00000000
0xB000181C=0x00004000
0xB0001818=0x00000332
0xB0001810=0x00000006
0xB0001804=0x00000027
0x55AA55AA=0x1
0x55AA55AA=0x1
0x55AA55AA=0x1
0xB0001804=0x0000002B
0xB0001804=0x0000002B
0xB0001804=0x0000002B
0xB0001818=0x00000232
0xB000181C=0x00004781
0xB000181C=0x00004401
0xB0001804=0x00000020
0xB0001834=0x00888820
0x55AA55AA=0x1
0xB0000218=0x00000008
0xB8003160=0x00008000
0xB80031A0=0x00008000
0xB000022C=0x00000100
0xB000022C=0x00000100
    
```

0xB000022C=0x00000100

DDR - Address 0 = {0xB0000220} 。 DDR - Value0 = {0x01000000} 。

DDR - Address 1 = {0xB0000264} 。 DDR - Value1 = {0xC0000018} 。

DDR - Address 2 = {0xB0000220} 。 DDR -Value2 = {0x01000018} 。

DDR - Address 3 = {0x55AA55AA} 。 DDR - Value3 = {0x1} 。

DDR - Address 4 = {0x55AA55AA} 。 DDR - Value4 = {0x1} 。

以此類推 . . .

DDR參數存放在儲存體中必須以16bytes對齊，不足的部分需要以Dummy補足。

uBoot = 存放uboot binary file 。

再燒入到Nand flash 中Block0，Block1，Block2，Block3，下圖為示意圖：

0x00000000	U-Boot	Block0
	Block0	
	U-Boot	Block1
	Block1	
	U-Boot	Block2
	Block2	
	U-Boot	Block3
	Block3	

Figure 4-6 Block 0~3 of NAND Flash

下圖為實際從NAND Flash 讀取出來的 出來的資料 如下圖表示：

並且可得知

Boot Code Marker = {0x20,'T','V','N'} 。

Execute Address = 0x00000200 。

Image Size = 0x0000A628 。

Decrypt Address = 0xFFFFFFFF 。

DDR - Initial Marker = 0x55AA55AA 。

DDR - Counter = 0x0000002A 。

DDR - Address 0 = 0xB0000220 。 DDR - Value0 = 0x01000000 。

以此類推 . . .

uBoot = address 0x00000170 ~ 0x00000170 + 0x0000A628 。

但是在NAND flash中uBoot型態有限制如下：

$$\text{uBoot Image} + \text{Header} + \text{DDR parmater} \leq \text{Block Size}$$

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f				
00000000h:	Boot Code Marker				Execute Address				Image Size				Decrypt Address							
	20	54	56	4E	00	02	00	00	28	A6	00	00	FF	FF	FF	FF				
00000010h:	Initial Marker				Counter				Address0				Value0							
	55	AA	55	AA	2A	00	00	00	20	02	00	B0	00	00	00	01				
00000020h:					Address1				Value1				Address2				Value2			
	64	02	00	B0	18	00	00	C0	20	02	00	B0	18	00	00	01				
00000030h:	AA	55	AA	55	01	00	00	00	AA	55	AA	55	01	00	00	00				
00000040h:	08	18	00	B0	30	80	00	00	AA	55	AA	55	01	00	00	00				
00000050h:	AA	55	AA	55	01	00	00	00	AA	55	AA	55	01	00	00	00				
00000060h:	00	18	00	B0	76	04	03	00	AA	55	AA	55	01	00	00	00				
00000070h:	04	18	00	B0	21	00	00	00	AA	55	AA	55	01	00	00	00				
00000080h:	04	18	00	B0	23	00	00	00	AA	55	AA	55	01	00	00	00				
00000090h:	AA	55	AA	55	01	00	00	00	AA	55	AA	55	01	00	00	00				
000000a0h:	04	18	00	B0	27	00	00	00	AA	55	AA	55	01	00	00	00				
000000b0h:	AA	55	AA	55	01	00	00	00	AA	55	AA	55	01	00	00	00				
000000c0h:	20	18	00	B0	00	00	00	00	24	18	00	B0	00	00	00	00				
000000d0h:	1C	18	00	B0	00	40	00	00	18	18	00	B0	32	03	00	00				
000000e0h:	10	18	00	B0	06	00	00	00	04	18	00	B0	27	00	00	00				
000000f0h:	AA	55	AA	55	01	00	00	00	AA	55	AA	55	01	00	00	00				
00000100h:	AA	55	AA	55	01	00	00	00	04	18	00	B0	2B	00	00	00				
00000110h:	04	18	00	B0	2B	00	00	00	04	18	00	B0	2B	00	00	00				
00000120h:	18	18	00	B0	32	02	00	00	1C	18	00	B0	81	47	00	00				
00000130h:	1C	18	00	B0	01	44	00	00	04	18	00	B0	20	00	00	00				
00000140h:	34	18	00	B0	20	88	88	00	AA	55	AA	55	01	00	00	00				
00000150h:	18	02	00	B0	08	00	00	00	60	31	00	B8	00	80	00	00				
00000160h:	Address41				Value41				Dummy				Dummy							
	A0	31	00	B8	00	80	00	00	00	00	00	00	00	00	00	00				
00000170h:	uBoot : address 0x00000170 ~ 0x0000170 + 0x0000A628																			
	13	00	00	EA	14	F0	9F	E5	10	F0	9F	E5	0C	F0	9F	E5				
00000180h:	08	F0	9F	E5	04	F0	9F	E5	00	F0	9F	E5	04	F0	1F	E5				
00000190h:	20	03	00	00	78	56	34	12	78	56	34	12	78	56	34	12				
000001a0h:	78	56	34	12	78	56	34	12	78	56	34	12	78	56	34	12				
000001b0h:	00	02	00	00	28	A6	00	00	88	AC	00	00	28	A6	00	00				
000001c0h:	DE	C0	AD	0B	00	00	0F	E1	1F	00	C0	E3	D3	00	80	E3				
000001d0h:	00	F0	29	E1	BC	D0	9F	E5	07	D0	CD	E3	00	00	A0	E3				
000001e0h:	16	05	00	EB	00	40	A0	E1	01	50	A0	E1	02	60	A0	E1				
000001f0h:	04	D0	A0	E1	8C	00	4F	E2	00	90	46	E0	06	00	50	E1				
00000200h:	06	00	00	0A	06	10	A0	E1	5C	30	1F	E5	03	20	80	E0				
00000210h:	00	06	B0	E8	00	06	A1	E8	02	00	50	E1	FB	FF	FF	3A				
00000220h:	74	00	9F	E5	74	10	9F	E5	00	20	A0	E3	01	00	50	E1				
00000230h:	02	00	00	2A	00	20	80	E5	04	00	80	E2	FA	FF	FF	EA				
00000240h:	00	00	9F	E5	00	F0	A0	E1	54	06	00	00	28	A6	00	00				
00000250h:	28	A6	00	00	28	A6	00	00	00	00	A0	E3	17	0F	07	EE				
00000260h:	17	0F	08	EE	10	0F	11	EE	23	0C	C0	E3	87	00	C0	E3				
00000270h:	02	00	80	E3	01	0A	80	E3	10	0F	01	EE	0E	C0	A0	E1				
00000280h:	0A	00	00	EB	0C	E0	A0	E1	0E	F0	A0	E1	00	00	A0	E1				
00000290h:	E8	D0	1F	E5	FE	FF	FF	EB	00	80	00	BC	28	A8	00	00				
000002a0h:	88	AE	00	00	00	00	A0	E1	00	00	A0	E1	00	00	A0	E1				
000002b0h:	68	00	9F	E5	00	10	E0	E3	00	10	80	E5	00	00	0F	E1				
000002c0h:	C0	00	80	E3	00	F0	21	E1	54	00	9F	E5	54	10	9F	E5				
000002d0h:	00	10	80	E5	50	00	9F	E5	50	10	9F	E5	00	10	80	E5				

Figure 4-7 Address 0x00000000~0x000002e0 of NAND Flash

4.2.1.2 Data型態

主要將指定的檔案放入到NAND Flash中所指定的位址，不做任何其他的處理。依據輸入的Image start offset的值(需要對齊Page Size，Page Size是依據NAND Flash規格所決定)，決定將Data存放在NAND Flash的哪個位址，如果Image start offset = 0x10000，則將Data 存放到 NAND Flash 0x10000 的位址，此方式主要可以幫助使用者依據個人需求配置NAND Flash，例如當Linux Kernel 需要由uBoot型態中Image帶起來，Image設定帶起來的位址為0x8000，則需要將Linux Kernel當作Data型態，配置到0x8000的位址。

目前支援YAFFS2與UBIFS兩種檔案系統格式。這兩種格式都可以選擇DATA型態，將做好的Image存放到NAND Flash對應的位址。讓使用者可以透過uBoot或Linux來讀取檔案系統。

YAFFS2製作In-band tags Image命令如下：(yaffs2的tag儲存在DATA區塊中)

```
# mkyaffs2 --inband-tags -p 2048 rootfs rootfs_yaffs2.img
```

--inband-tags：yaffs2的tag儲存在DATA區塊。

-p：設定NAND Flash頁的大小(Page Size)。

即可將rootfs資料夾壓縮成rootfs_yaffs2.img，再透過NuWriter放到相對應NAND Flash的位址。

輸入下列命令即可將YAFFS2 inband-tags 檔案系統掛在flash資料夾中：

```
mount -t yaffs2 -o "inband-tags" /dev/mtdblock2 /flash
```

YAFFS2的指令可以在yaffs2utils套件中找到。

UBIFS製作Image命令如下：

```
# mkfs.ubifs -F -x lzo -m 2048 -e 126976 -c 732 -o rootfs_ubifs.img -d ./rootfs
# ubinize -o ubi.img -m 2048 -p 131072 -o 2048 -s 2048 rootfs_ubinize.cfg
```

mkfs.ubifs 使用的參數說明如下：

-F：設定檔案系統未使用的空間優先mount。

-x：壓縮的格式，"lzo", "favor_lzo", "zlib" 或 "none" (預設："lzo")

-m：最小的I/O操作的大小，也就是NAND Flash一個頁的大小。

-e：邏輯擦除塊的大小(logical erase block size)。因為實體擦除塊(PEB)為128KiB，所以邏輯擦除塊設定為124KiB=126976。

-c：最大的擦除塊的號碼(maximum logical erase block count)。

-o：輸出檔案。

ubinize使用的參數說明名如下：

-o：輸出檔案。

-m：最小輸入/輸出的大小，也就是NAND Flash一個頁的大小。

-p：實體擦除塊大小，128KiB=131072。

-O：VID檔頭位移位置。

-s：使用最小輸入/輸出的大小，存放UBI檔頭。

rootfs_ubinize.cfg 內容如下：

```
[rootfs-volume]
mode=ubi
image=rootfs_ubifs.img
vol_id=0
vol_size=92946432
```

```
vol_type=dynamic
vol_name=system
vol_flags=autoresize
```

即可將rootfs資料夾壓縮成ubi.img，再透過NuWriter放到相對應NAND Flash的位址。

輸入下列命令即可將UBIFS檔案系統掛在flash資料夾中：

需要參考/sys/class/misc/ubi_ctrl/dev內容，假設內容為 10：56，則設定如下：

```
mknod /dev/ubi_ctrl c 10 56
ubiattach /dev/ubi_ctrl -p /dev/mtd2
mount -t ubifs ubi0:system /flash
```

UBIFS相關指令可以在mtd-utils套件中找到。

Linux內核也必須將對應的檔案系統設置後才可正常運作設置方式如下：

YAFFS2：

```
File systems --->
  [*] Miscellaneous filesystems --->
    <*> yaffs2 file system support
    <*> Autoselect yaffs2 format
    <*> Enable yaffs2 xattr support
```

UBIFS：

```
Device Drivers --->
  -*- Memory Technology Device (MTD) support --->
    <*> Enable UBI - Unsorted block images --->
File systems --->
  [*] Miscellaneous filesystems --->
    <*> UBIFS file system support
    [*] Advanced compression options
    [*] LZO compression support
    [*] ZLIB compression support
```

相關套件編譯：

目錄	描述
lzo-2.09.tar.gz	壓縮/解壓縮工具。 交叉編譯命令如下： \$ cd lzo-2.09 \$./configure --host=arm-linux --prefix=\$PWD/./install \$ make \$ make install

libuuid-1.0.3.tar.gz	產生獨立的序號工具. 交叉編譯命令如下: \$ cd libuuid-1.0.3 \$./configure --host=arm-linux --prefix=\$PWD/../install \$ make \$ make install
mtd-utils.tar.gz	mtd-utils源碼. 交叉編譯命令如下: 需要使用到lzo-2.09.tar.gz套件和libuuid-1.0.3.tar.gz套件 \$ cd mtd-utils \$ export CROSS=arm-linux- \$ export WITHOUT_XATTR=1 \$ export DESTDIR=\$PWD/../install \$ export LZOCPPFLAGS=-I/home/install/include \$ export LZOLDFLAGS=-L/home/install/lib \$ make \$ make install
yaffs2utils.tar.gz	yaffs2命令工具 \$ make

4.2.1.3 Environment型態

主要設定uBoot的環境變數，檔案放入到NAND Flash中所指定的位址，讓uBoot讀取時做相對應的動作。依據輸入的Image start offset的值(需要對齊Page Size)，決定將Enviroment存放在NAND Flash的哪個位址，如果Image start offset = 0x10000，則將Enviroment存放到 NAND Flash 0x10000 的位址。

env.txt 存放的是 U-Boot 的環境變數及其數值，內容舉例如下:

```
baudrate=115200
bootdelay=3
ethact=emac
ethaddr=00:00:00:11:66:88
stderr=serial
stdin=serial
stdout=serial
```

4.2.1.4 Pack型態

依據輸入Pack.bin的內容，決定將Pack內容中的Image放到相對應NANAD Flash的位址。Pack.bin的原理可以參考Pack模式章節。

4.2.1.5 壞頁處理

NAND Flash支援四種型態的方式燒入到NAND Flash中，對於壞頁則直接跳過後，假設有一資料將寫到NAND中Page10,Page11,Page12,Page13和Page14，但是當中的Page13偵測到壞頁時，直接跳過則將寫入到Page10,Page11,Page12,Page14和Page15中。

4.2.2 操作方法

4.2.2.1 新增Image

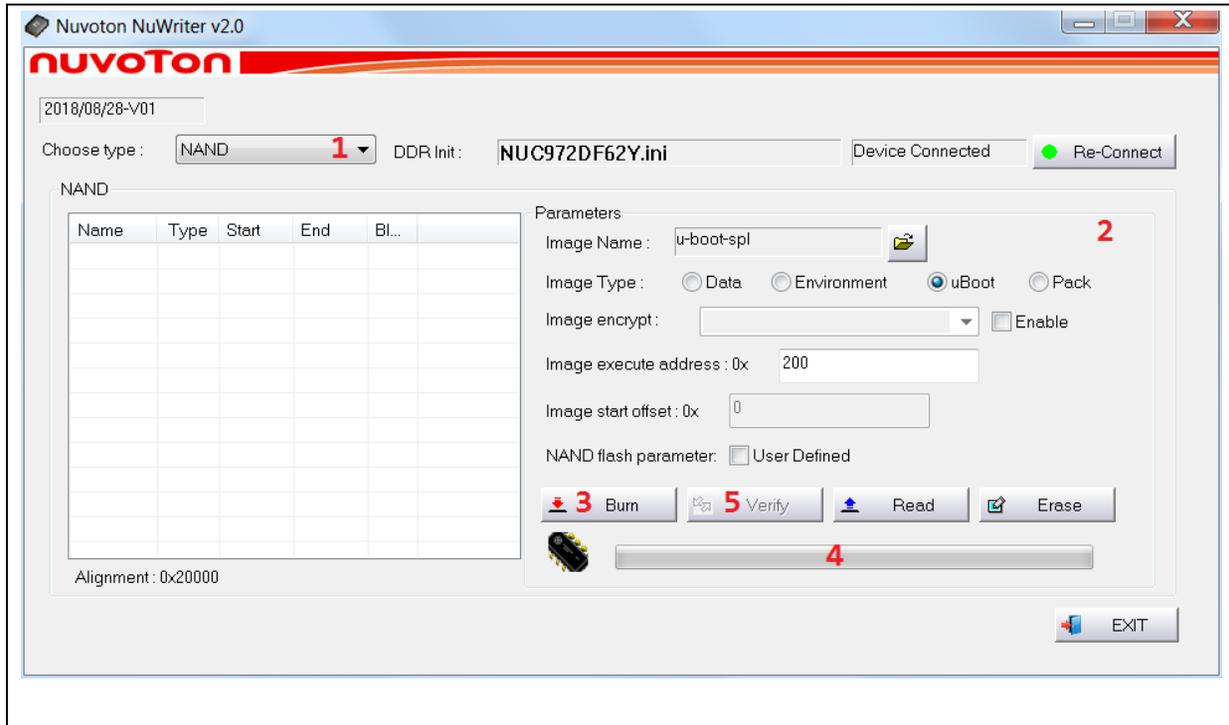


Figure 4-8 NAND – New Image

依照上圖步驟即可以完成新增Image檔案：

1. 選擇“NAND”模式，表格只會紀錄當次燒錄的Image檔案，並不會讀取NAND Flash中Image的資料。
2. 輸入Image檔案資料：
 - Image Name 選擇要燒錄的 Image檔案
 - Image Type 選擇Image型態
 - Image encrypt 設置是否需AES加密, 若是, 設置秘鑰文件, 在FS型態中無法使用AES加密 (N9H30 不支持此模式)
 - Image execute address 設置Image執行位置, 依編譯設定而輸入, 只有在uBoot型態才有效
 - Image start offset 設置Image燒錄在NAND Flash的位址
 - NAND flash parameter勾選User Defined決定是否要自訂NAND參數
3. 按下“Burn”。
4. 等待進度表完成，表格將會顯示這次燒錄完成的Image檔案。
5. 在完成後，可以選擇按下“Verify”即可確認燒入資料是否正確。

4.2.2.2 自訂NAND參數

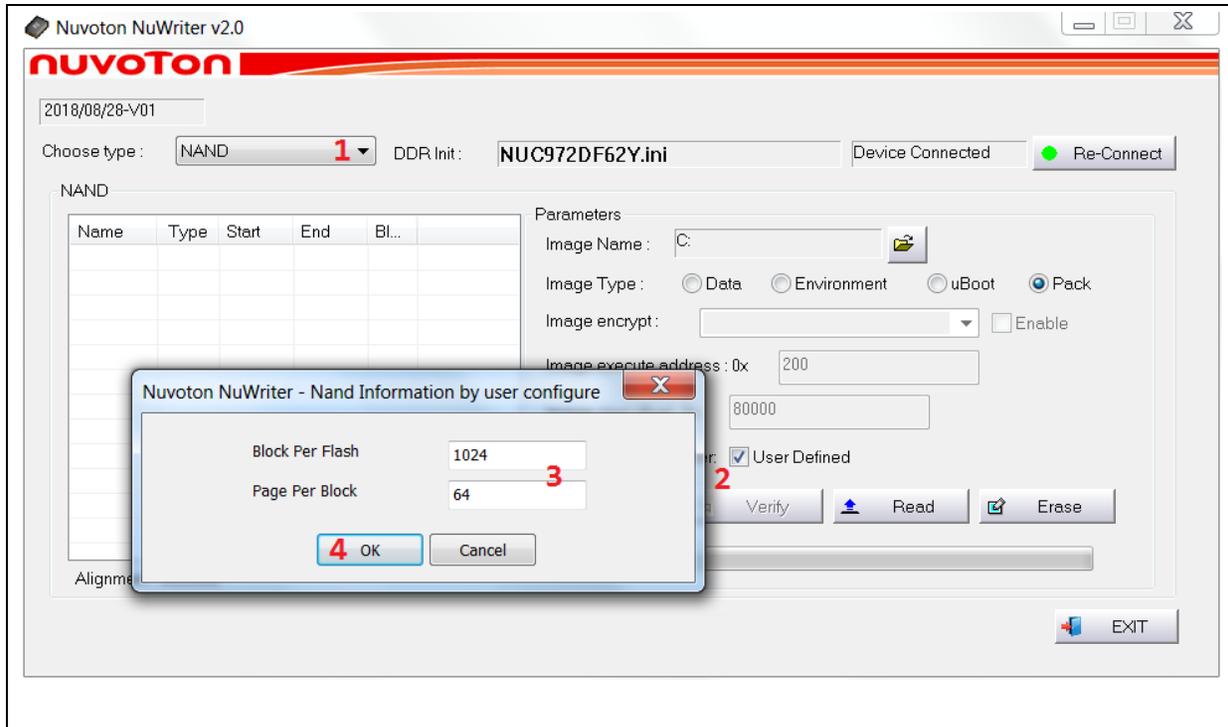


Figure 4-9 NAND – Configure Parameters

NuWriter會自動偵測NAND的ID來決定它的參數，使用者也可以自己依據NAND Flash規格來設定NAND的參數。依照上圖步驟即可以完成設定NAND參數：

1. 選擇“NAND”模式。
2. 勾選“User Defined”會跳出視窗，提供NAND參數設定。
3. 輸入Block Per Flash(總共多少個Block)及Page Per Block(每個Block有幾個Page)。
4. 按下“OK”，即可完成參數設定。

4.2.2.3 讀取Image

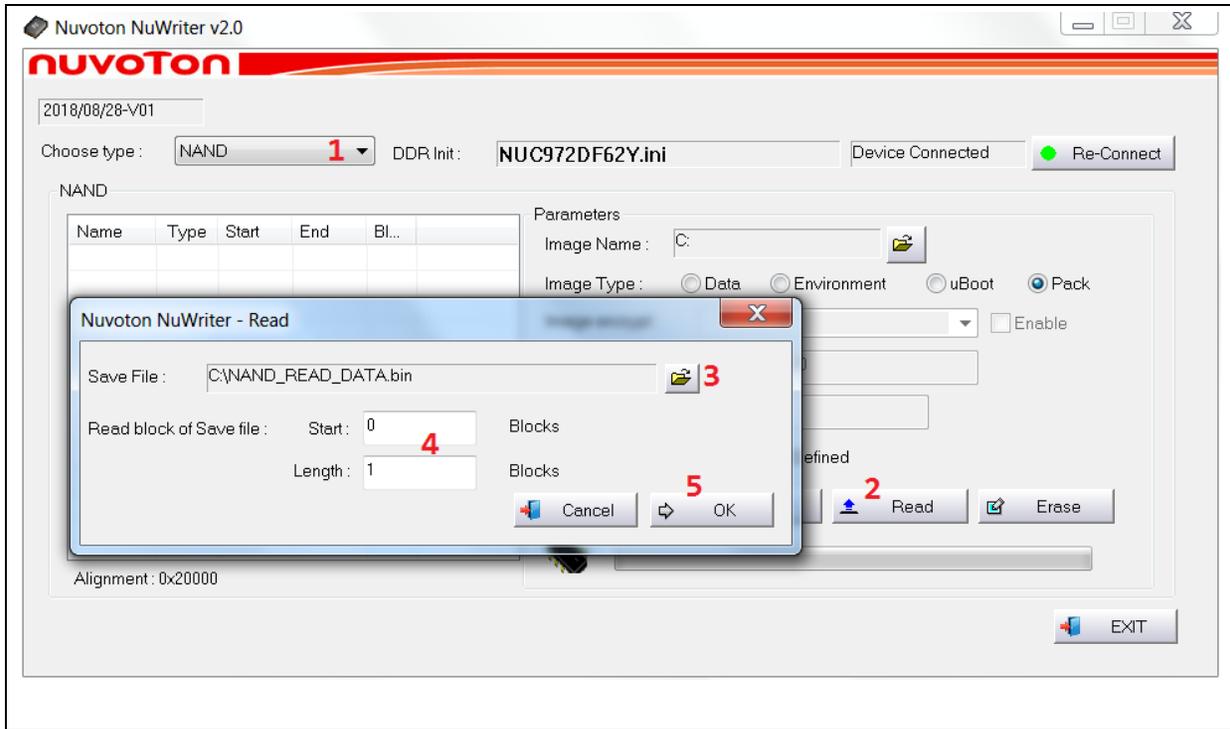


Figure 4-10 NAND – Read

依照上圖步驟即可以完成讀取Image：

1. 選擇“NAND”模式。
2. 按下“Read”。
3. 選擇要儲存檔案的位置。
4. 輸入讀回來的Blocks(每一Block大小依據NAND Flash規格來決定)。
5. 按下“OK”，即可完成Image讀取。

4.2.2.3.1 讀取模式

NuWriter 可以依據 NuWriter/path.ini 中的內容，更改讀取模式，如下圖：

1. ChipReadWithBad=0時為預設值，讀取時只會讀回資料的部分。
2. ChipReadWithBad=1時，讀取時會讀取資料加上OOB(out of band)區域。

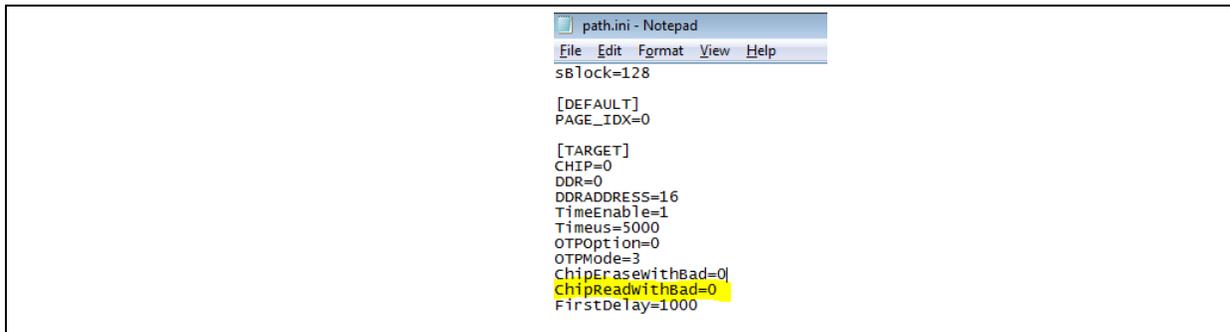


Figure 4-11 NAND – path.ini(1)

4.2.2.4 移除 Image

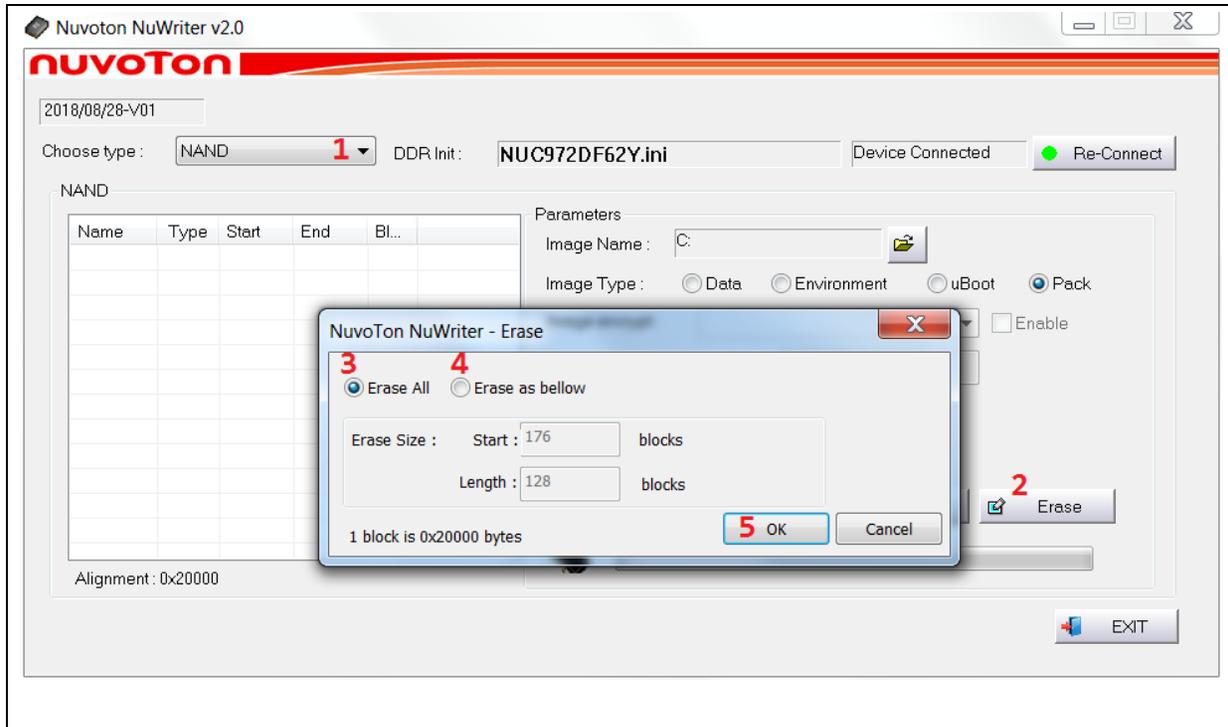


Figure 4-12 NAND – Erase

依照上圖步驟即可以完成移除Image檔案：

1. 選擇“NAND”模式。
2. 按下“Erase”
3. 選擇 Erase all or
4. 選擇 Erase as bellow
 - 輸入擦除的Blocks(每一Block大小依據NAND Flash規格來決定)
5. 按下“OK”，即可完成擦除。

4.2.2.4.1 移除模式

NuWriter 可以依據 NuWriter/path.ini 中的內容, 更改移除模式, 如下圖:

1. ChipEraseWithBad=0時為預設值, 清除時只會清除資料。
2. ChipEraseWithBad=1時, 清除時會將資料與OOB區域都會清除。

```

path.ini - Notepad
File Edit Format View Help
sBlock=128

[DEFAULT]
PAGE_IDX=0

[TARGET]
CHIP=0
DDR=0
DDRADDRESS=16
TimeEnable=1
Timeus=5000
OTPOption=0
OTPMODE=3
ChipEraseWithBad=0
ChipReadWithBad=0
FirstDelay=1000
    
```

Figure 4-13 NAND – path.ini(2)

4.3 SPI 模式

4.3.1 Image檔案型態

4.3.1.1 uBoot 型態

主要是作為開機的第一支程式，當NUC970/N9H30系列晶片上電後會讀取uBoot型態中的Image並執行，不限定一定要存放uBoot，一般程式也是可以使用uBoot型態來作為上電後第一支執行程式。uBoot增加16bytes的檔頭後和DDR參數合併，方式如下：

	0x0	0x0	0x0	0x0
0x00	Boot Code Marker	Execute Address	Image Size	Decrypt Address
0x10	DDR - Initial Marker	DDR - Counter	DDR - Address 0	DDR - Value 0
0x20	DDR - Address 1	DDR - Value 1	DDR - Address 2	DDR - Value 2
0x30	DDR - Address 3	DDR - Value 3	DDR - Address 4	DDR - Value 5
0x40	DDR - Address 5	DDR - Value 5	DDR - Dummy	DDR - Dummy
0x50	uBoot			

Figure 4-14 Boot code header

詳細的內容可以參考NAND Flash模式中的uBoot 型態章節。合併以後再燒入到SPI Flash中0x0的地址，下圖為示意圖：

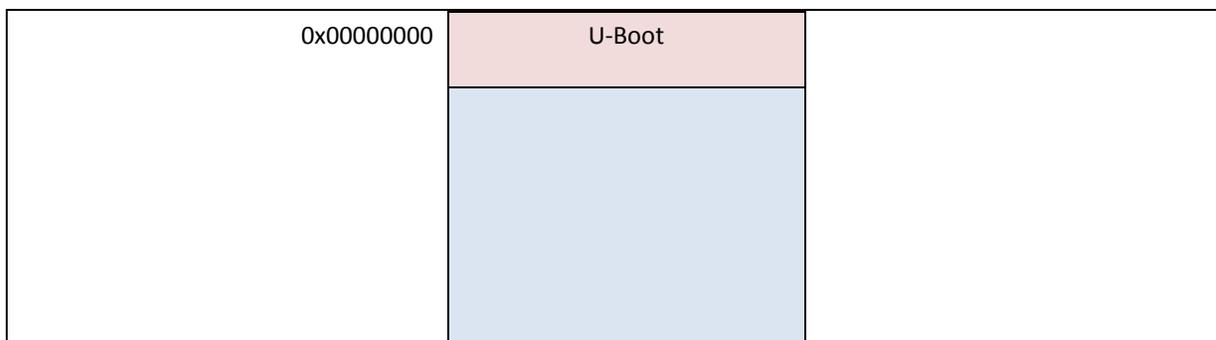




Figure 4-15 SPI Flash

4.3.1.2 Data 型態

主要將指定的檔案放入到SPI Flash中所指定的位址，不做任何其他的處理。依據輸入的Image start offset的值(需要對齊Block Size，Block Size是依據SPI Flash規格所決定)，決定將Data存放在SPI Flash的哪個位址，如果Image start offset = 0x10000，則將Data 存放到 SPI Flash 0x10000 的位址，此方式主要可以幫助使用者依據個人需求配置SPI Flash，例如當Linux Kernel 需要由uBoot型態中Image帶起來時，Linux Kernel帶起來的位址為0x8000，則需要將Linux Kernel當作Data型態，配置到0x8000的位址。

4.3.1.3 Environment 型態

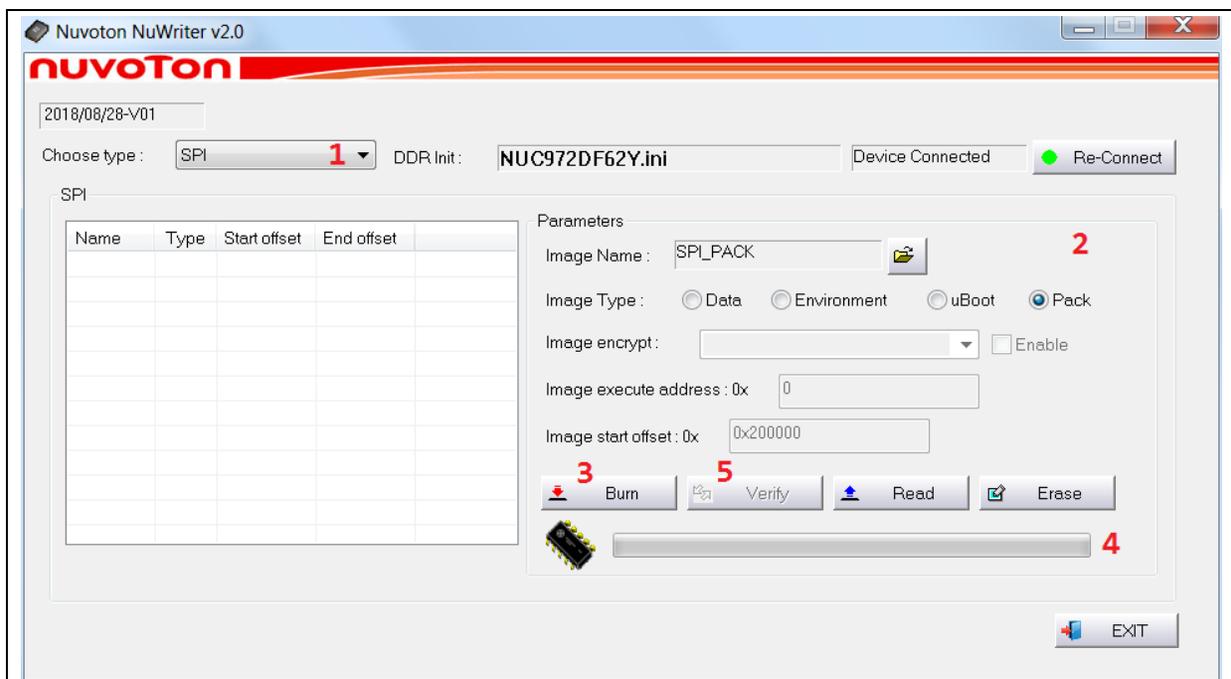
主要設定uBoot的環境變數，檔案放入到SPI Flash中所指定的位址，讓uBoot讀取時做相對應的動作。依據輸入的Image start offset的值(需要對齊Block size)，決定將Environment存放在SPI Flash的哪個位址，如果Image start offset = 0x10000，則將Environment存放到 SPI Flash0x10000 的位址。

4.3.1.4 Pack 型態

依據輸入Pack.bin 的內容，決定將Pack 內容中的Image 放到相對應SPI Flash的位址。Pack.bin的原理可以參考Pack模式章節。

4.3.2 操作方法

4.3.2.1 新增Image



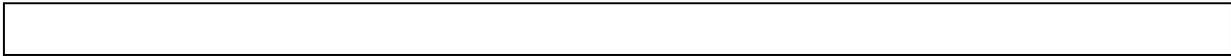


Figure 4-16 SPI – New Image

依照上圖步驟即可以完成新增Image檔案：

1. 選擇“SPI”模式，表格只會紀錄當次燒錄的Image檔案，並不會讀取SPI Flash中Image的資料。
2. 輸入Image檔案資料：
 - Image Name 選擇要燒錄的 Image檔案
 - Image Type 選擇Image型態
 - Image encrypt 設置是否需AES加密, 若是, 設置秘鑰文件. (N9H30 不支持此模式)
 - Image execute address 設置Image執行位置，依編譯設定而輸入，只有在uBoot型態才有效
 - Image start offset設置Image燒錄在SPI Flash的位址
3. 按下“Burn”。
4. 等待進度表完成，表格將會顯示這次燒錄完成的Image檔案。
5. 在完成以後，可選擇按下按下“Verify”即可確認燒入資料是否正確。

4.3.2.2 讀取Image

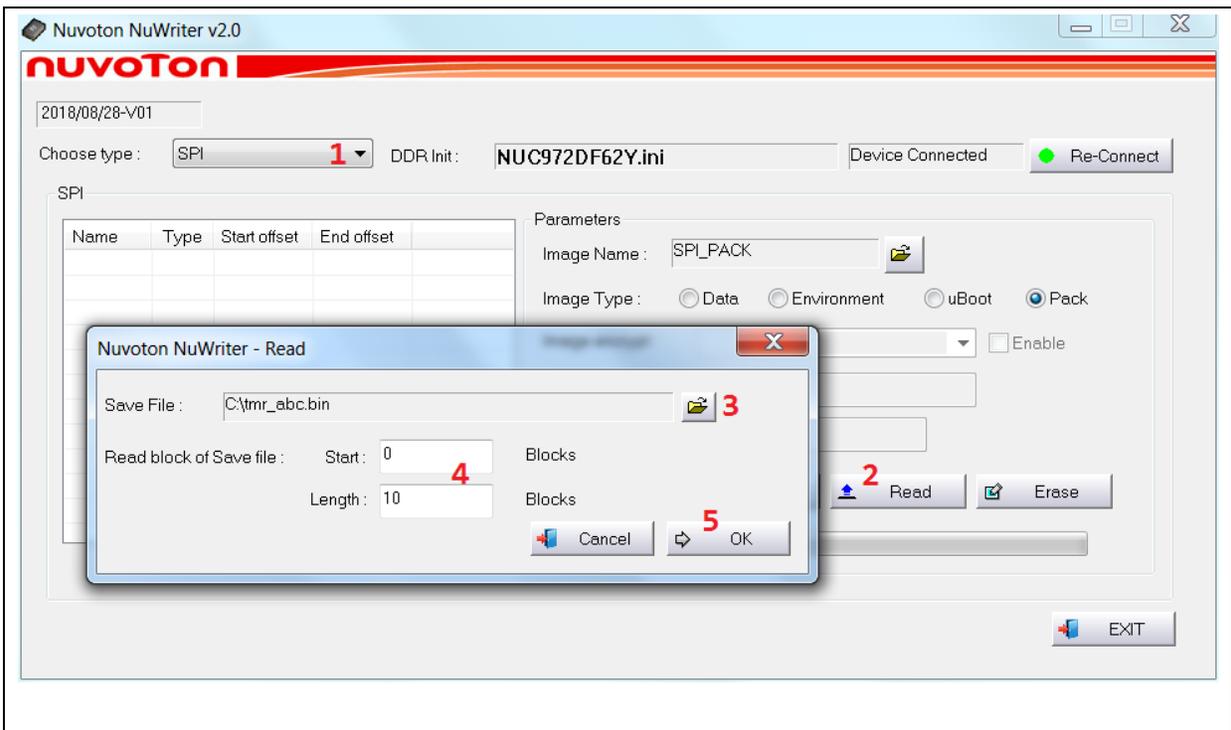


Figure 4-17 SPI – Read Image

依照上圖步驟即可以完成讀取Image：

1. 選擇“SPI”模式。
2. 按下“Read”。
3. 選擇要儲存檔案的位置。
4. 輸入讀回來的Blocks(每一Block大小依據SPI Flash規格來決定)。
5. 按下“OK”，即可完成Image讀取。

4.3.2.3 移除 Image

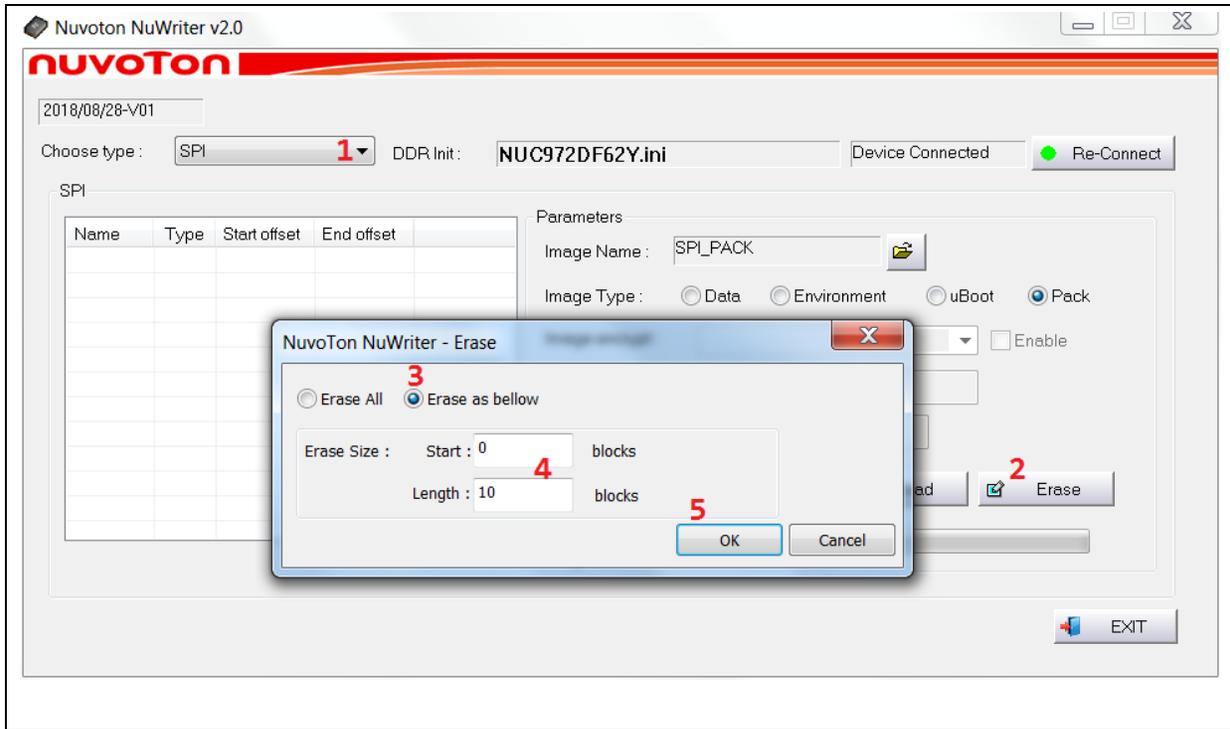


Figure 4-18 NuWriter – Erase

依照上圖步驟即可以完成移除Image檔案：

1. 選擇“SPI”模式。
2. 按下“Erase”
3. 選擇 Erase all or
4. 選擇 Erase as bellow
 - 輸入擦除的Blocks(每一Block大小依據SPI Flash規格來決定)
5. 按下“OK”，即可完成擦除。

4.4 eMMC 模式

4.4.1 Image檔案型態

4.4.1.1 uBoot 型態

主要是作為開機的第一支程式，當NUC970/N9H30系列晶片上電後會讀取uBoot型態中的Image並執行，不限定一定要存放uBoot，一般程式也是可以使用uBoot型態來作為上電後第一支執行程式。uBoot增加16bytes的檔頭後和DDR參數合併，方式如下：

	0x0	0x0	0x0	0x0
0x00	Boot Code Marker	Execute Address	Image Size	Decrypt Address
0x10	DDR - Initial Marker	DDR - Counter	DDR - Address 0	DDR - Value0

0x20	DDR - Address 1	DDR - Value 1	DDR - Address 2	DDR - Value 2
0x30	DDR - Address 3	DDR - Value 3	DDR - Address 4	DDR - Value 5
0x40	DDR - Address 5	DDR - Value 5	DDR - Dummy	DDR - Dummy
0x50	uBoot			

Figure 4-19 Boot code header

詳細的內容可以參考NAND Flash模式中的uBoot 型態章節。合併以後再燒入到eMMC 中0x400的位址，下圖為示意圖：

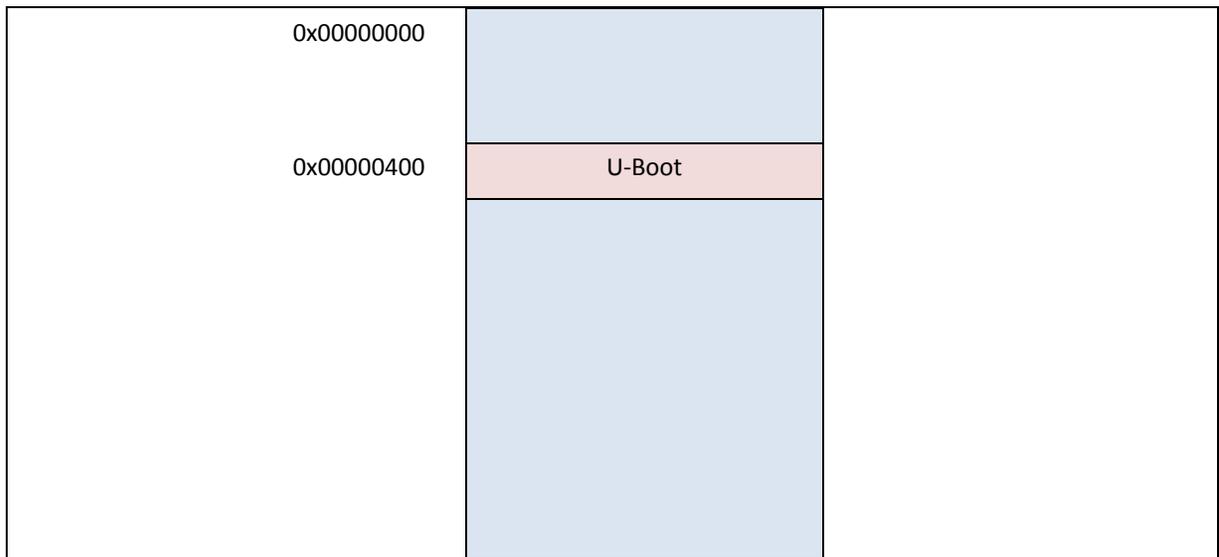


Figure 4-20 eMMC

4.4.1.2 Data 型態

主要將指定的檔案放入到eMMC中所指定的位址，不做任何其他的處理。依據輸入的Image start offset 的值(需要對齊512bytes)，決定將Data存放在eMMC的哪個位址，如果Image start offset = 0x10000，則將Data存放到eMMC 0x10000的位址，此方式主要可以幫助使用者依據個人需求配置eMMC，例如當Linux Kernel 需要由uBoot型態中Image帶起來時，Linux Kernel帶起來的位址為0x8000，則需要將Linux Kernel當作Data型態，配置到0x8000的位址。

4.4.1.3 Environment 型態

主要設定uBoot的環境變數，檔案放入到eMMC中所指定的位址，讓uBoot讀取時做相對應的動作。依據輸入的Image start offset 的值(需要對齊512bytes)，決定將Environment存放在eMMC的哪個位址，如果Image start offset = 0x10000，則將Environment存放到 eMMC 0x10000 的位址。

4.4.1.4 Pack 型態

依據輸入Pack.bin的內容，決定將Pack內容中的Image 放到相對應eMMC的位址。Pack.bin的原理可以參考Pack模式章節。

4.4.1.5 格式化(FAT32)

因為部分空間需要存放uBoot型態的Image或者其他型態的Image，所以必須設定保留空間來存放，不給使用者去做任何的修改，保留空間是以512bytes為單位，使用者可以依據自己的需要來設定保留空間的大小。注意：修改此參數可能造成既有的Image或檔案系統格式損毀。

4.4.2 操作方法

4.4.2.1 新增Image

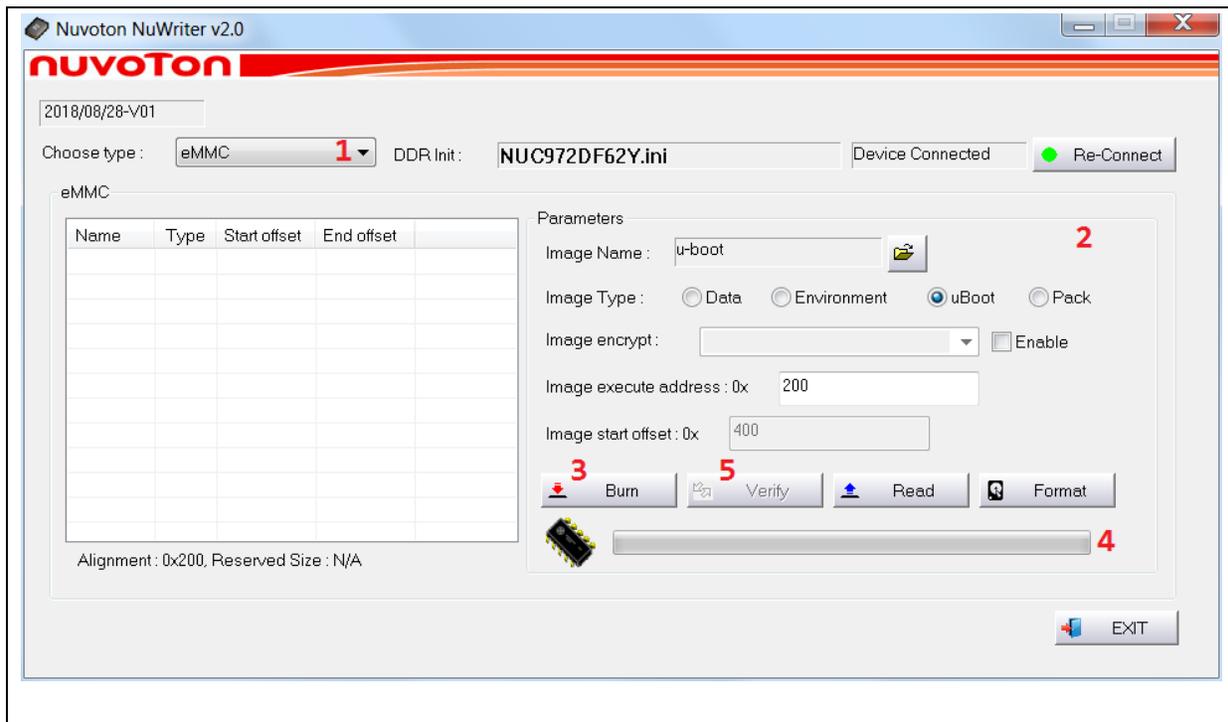


Figure 4-21 eMMC – New Image

依照上圖步驟即可以完成新增Image檔案：

1. 選擇“eMMC”模式，表格只會紀錄當次燒錄的Image檔案，並不會讀取eMMC中Image的資料。
2. 輸入Image檔案資料：
 - Image Name 選擇要燒錄的Image檔案
 - Image Type 選擇Image型態
 - Image encrypt 設置是否需AES加密，若是，設置秘鑰文件。(N9H30 不支持此模式)
 - Image execute address 設置Image執行位置，依編譯設定而輸入，只有在uBoot型態才有效
 - Image start offset設置Image燒錄在eMMC的位址
3. 按下“Burn”。
4. 等待進度表完成後，表格將會顯示這次燒錄完成的Image檔案。
5. 在完成以後，可以選擇按下“Verify”即可確認燒入資料是否正確。

4.4.2.2 讀取Image

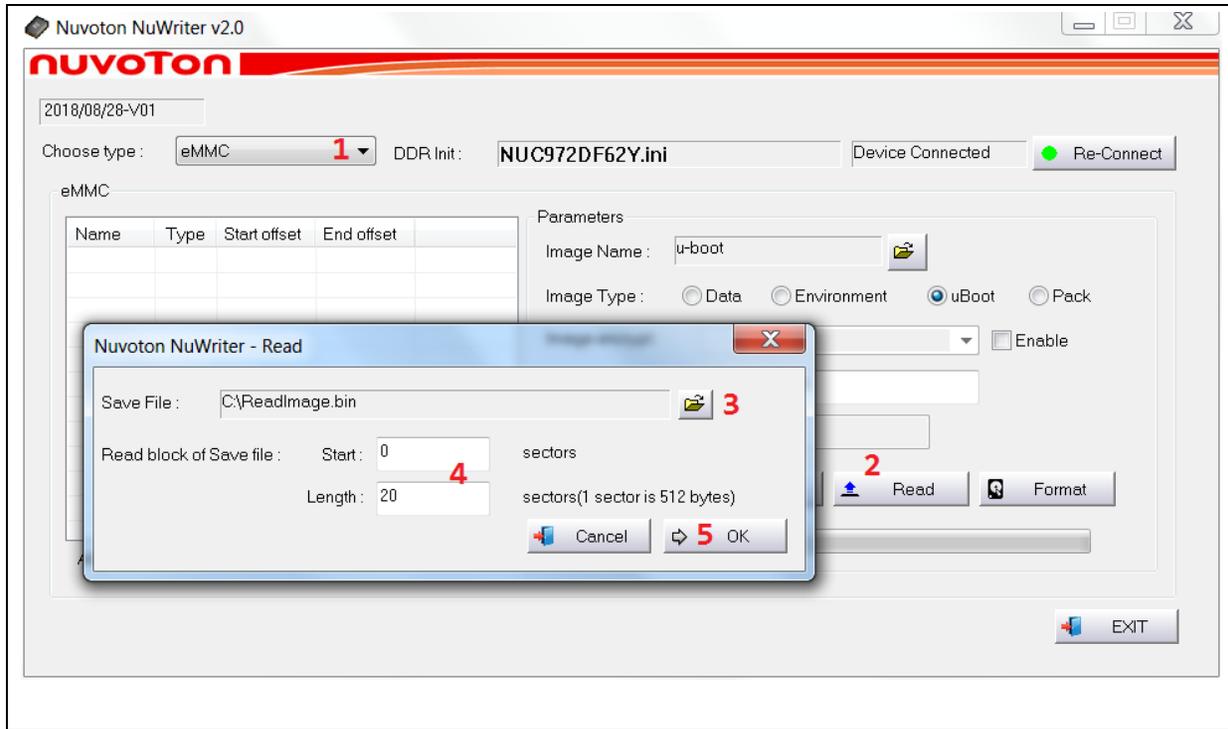


Figure 4-22 eMMC – Read

依照上圖步驟即可以完成讀取eMMC：

1. 選擇“eMMC”模式。
2. 按下“Read”。
3. 輸入儲存的檔案。
4. 輸入讀回來的sectors(1 sector is 512 bytes)。
5. 按下“OK”。即可完成。

4.4.2.3 格式化 (FAT32)

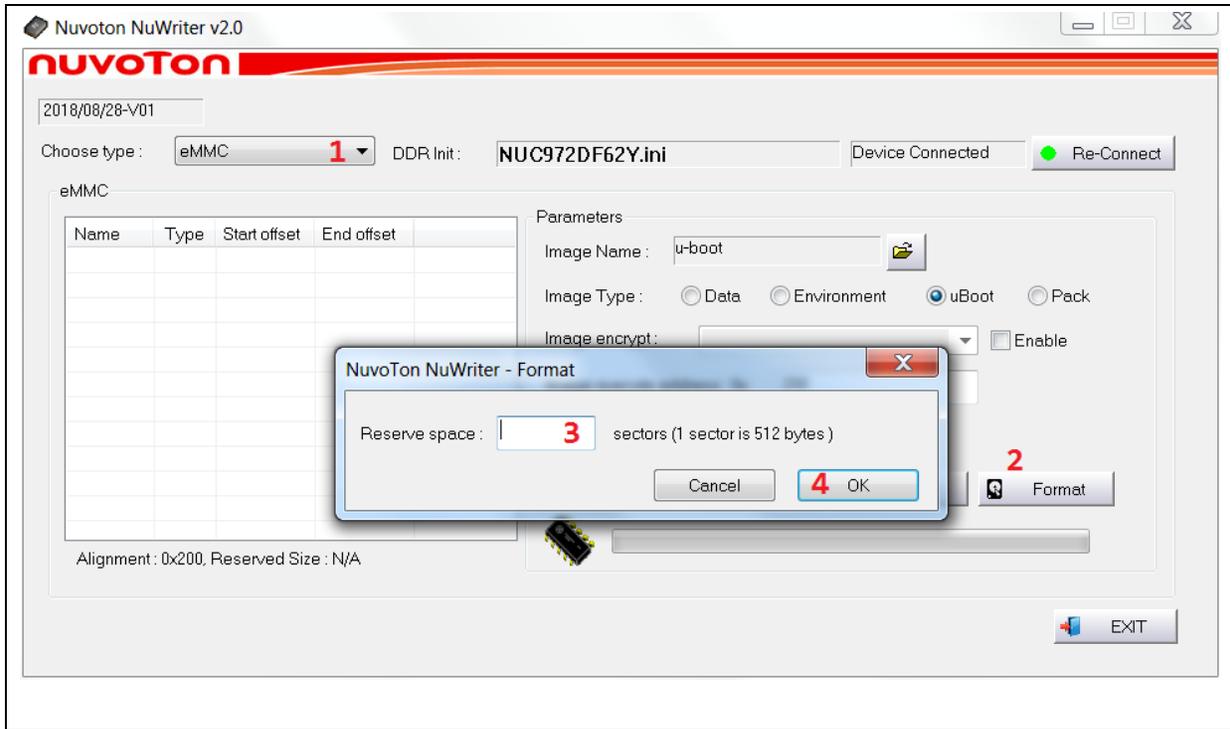


Figure 4-23 eMMC – Format

依照上圖步驟即可以完成eMMC格式化：

1. 選擇“eMMC”模式。
2. 按下“Format”。
3. 輸入保留空間(單位為512bytes)。注意：修改此參數可能造成既有的Image或檔案系統格式損毀。
4. 按下“OK”。即可完成。

4.5 Pack 模式

將每個Image檔案合併成一個Pack.bin檔案，uBoot加16bytes的檔頭後和DDR參數合併，方式如下：

	0x0	0x0	0x0	0x0
0x00	Initial Marker	File Length	File Number	Reserve
0x10	Child0-File Length	Child0-File Address	Child0-Reserve	Child0-Reserve
	Child0-data			
	Child1-File Length	Child1-File Address	Child1-Reserve	Child1-Reserve
	Child1-data			

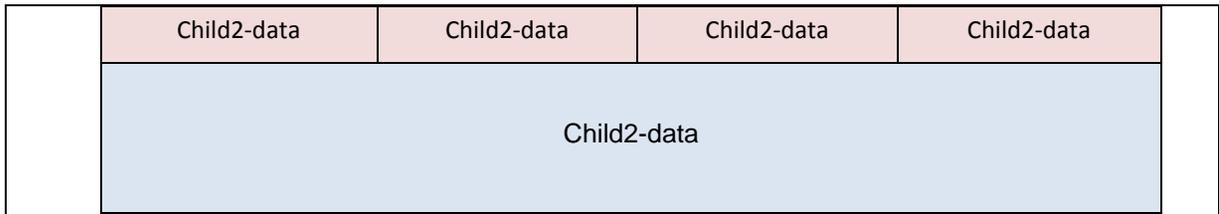


Figure 4-24 Pack header/Pack child header

Initial Marker = {0x00000005}。

File Length = {所有Image的長度}，對齊64Kbytes。

File Number = {有多少個Image檔}。

Child0-File Length = {第一個Image 資料的長度}。

Child0-File Address = {第一個Image燒入的位址}。

Child0-data = {第一個Image 資料}。

Child0-File Length = {第二個Image 資料的長度}。

Child0-File Address = {第二個Image燒入的位址}。

Child0-data = {第二個Image 資料}。

假設在NuWriter中 Pack 模式輸入兩個Image檔案分別為u-boot-spl.bin 和 u-boot.bin，如下圖所示：

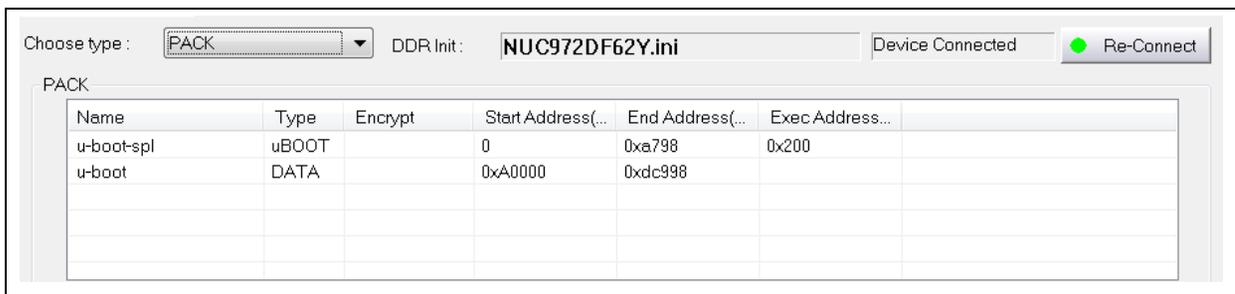


Figure 4-25 Pack Image

可以得知完成後的Pack格式如下：

Initial Marker = 0x00000005。

File Length = (0x0000a798+0x0003c998) ≅ 0x00050000。

File Number = 2。

Child0-File Length = 0x0000A798。

Child0-File Address = 0。

Child0-data = { address 0x00000000 + 0x00000020 ~ 0x00000020 + 0x0000A798}。

Child1-File Length =0x3c998。

Child1-File Address =0xA0000。

Child1-data = { address 0x0000A7B8 + 0x0000010 ~ 0x000A7C8 + 0x0003C998 }。

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	Initial Marker				File Length				File Number				Reserve			
00000000h:	05	00	00	00	00	00	05	00	02	00	00	00	FF	FF	FF	FF
	CO-File Length				CO-File Address				CO-Reserve				CO-Reserve			
00000010h:	98	A7	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	FF	FF
00000020h:	20	54	56	4E	00	02	00	00	28	A6	00	00	FF	FF	FF	FF
00000030h:	55	AA	55	AA	2A	00	00	00	20	02	00	B0	00	00	00	01
00000040h:	64	02	00	B0	18	00	00	C0	20	02	00	B0	18	00	00	01
00000050h:	AA	55	AA	55	01	00	00	00	AA	55	AA	55	01	00	00	00
00000060h:	08	18	00	B0	30	80	00	00	AA	55	AA	55	01	00	00	00
0000a760h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000a770h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000a780h:	00	C2	01	00	00	1B	B7	00	6E	75	63	39	37	30	5F	73
0000a790h:	65	72	69	61	6C	00	00	00	D0	72	00	00	00	00	00	00
0000a7a0h:	18	74	00	00	C8	73	00	00	00	74	00	00	08	73	00	00
									C1-File Length				C1-File Address			
0000a7b0h:	98	73	00	00	00	00	00	00	98	C9	03	00	00	00	0A	00
	C1-Reserve				C1-Reserve											
0000a7c0h:	FF	FF	FF	FF	FF	FF	FF	FF	13	00	00	EA	14	F0	9F	E5
0000a7d0h:	14	F0	9F	E5	14	F0	9F	E5	14	F0	9F	E5	14	F0	9F	E5
0000a7e0h:	14	F0	9F	E5	14	F0	9F	E5	80	01	40	00	E0	01	40	00
0000a7f0h:	40	02	40	00	A0	02	40	00	00	03	40	00	60	03	40	00
0000a800h:	C0	03	40	00	EF	BE	AD	DE	00	00	40	00	20	6C	03	00
0000a810h:	5C	E4	08	00	98	C9	03	00	DE	C0	AD	0B	00	00	0F	E1

Figure 4-26 Output Pack.bin

4.5.1 操作方法

Pack模式可以將多個Image檔案合併成一個Pack Image檔案，往後就可利用NuWriter將這個Pack Image直接快速的燒入到對應的存儲體中。(如 eMMC，NAND Flash，SPI Flash)。

4.5.1.1 新增Image

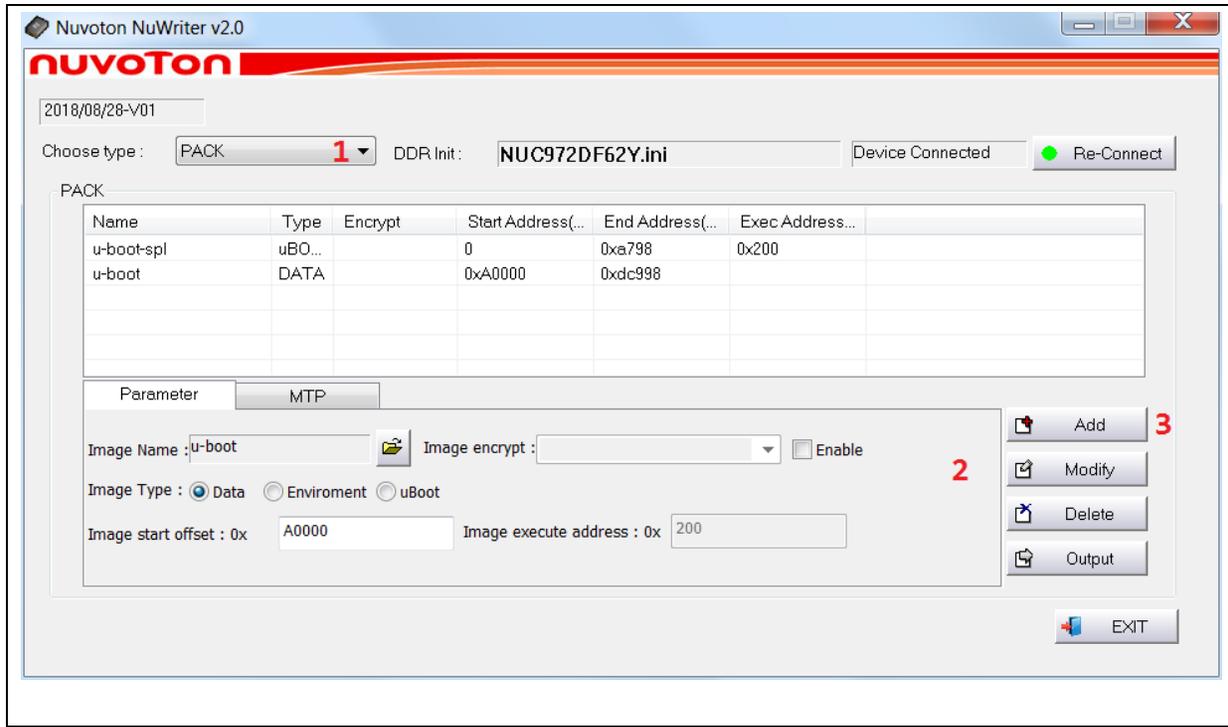


Figure 4-27 PACK – New Image

依照上圖步驟即可以完成新增image檔案：

1. 選擇“PACK” 模式。
2. 輸入Image檔案資料：
 - Image Name 選擇要燒錄的檔案
 - Image Type 選擇Image型態
 - Image encrypt 設置是否需AES加密, 若是, 設置秘鑰文件(N9H30 不支持此模式)
 - Image execute address設置Image執行位置, 依編譯設定而輸入, 只有在uBoot型態才有效
 - Image start offset 燒錄起始塊位置
3. 按下“Add”。

4.5.1.2 修改Image

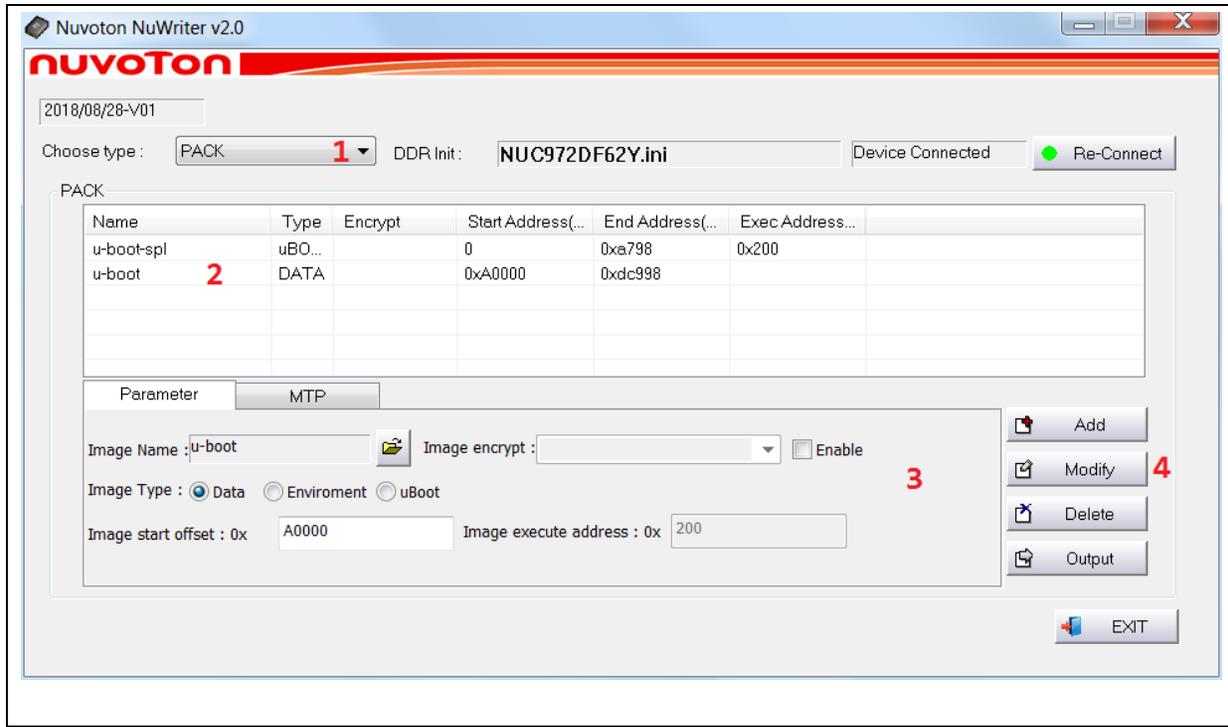


Figure 4-28 PACK – Modify Image

依照上圖步驟即可以完成修改image：

1. 選擇“PACK”模式。
2. 在表格上面連續按兩下需要修改的Image。
3. 修改Image檔案資料：
 - Image Name 選擇要燒錄的檔案
 - Image Type 選擇 System Image
 - Image encrypt 設置是否需AES加密，若是，設置秘鑰文件(N9H30 不支持此模式)
 - Image execute address 設置Image執行位置，依編譯設定而輸入，只有在uBoot型態才有效
 - Image start offset 燒錄起始塊位置
4. 按下“Modify”。

4.5.1.3 移除Image

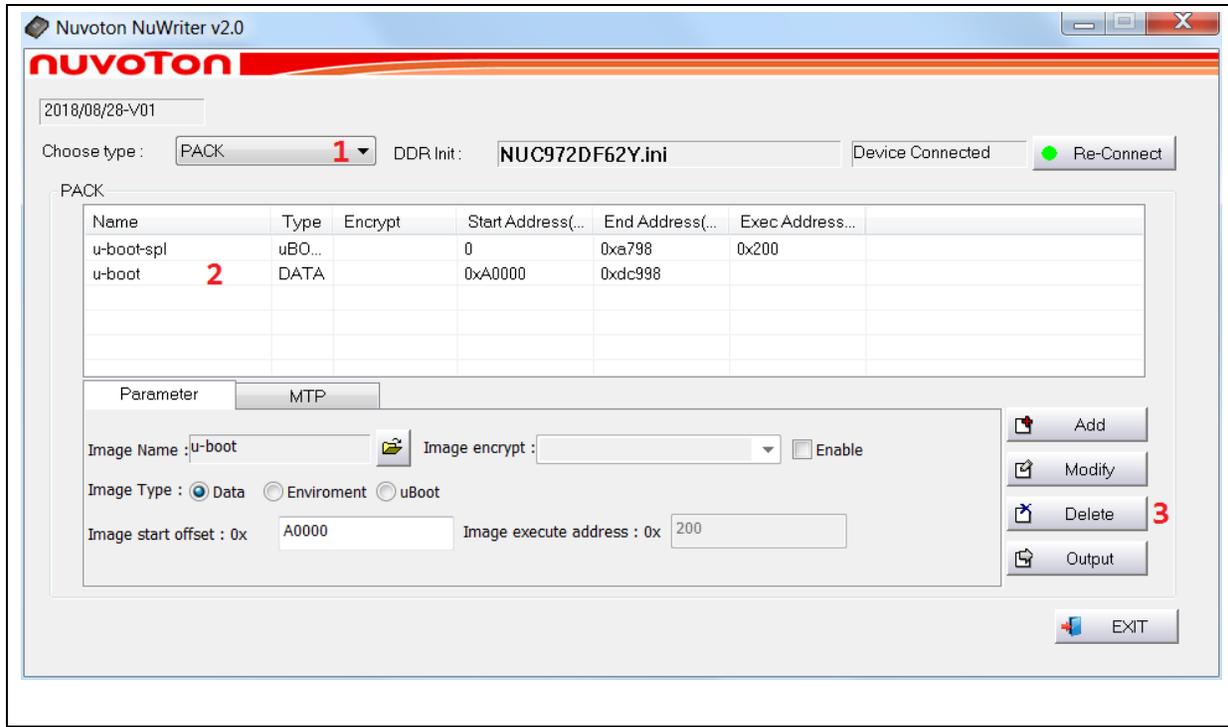


Figure 4-29 PACK – Delete Image

依照上圖步驟即可以完成移除Image：

1. 選擇“PACK”模式。
2. 在表格上點一下要刪除的Image。
3. 按下“Delete”。

4.5.1.4 輸出Pack檔案

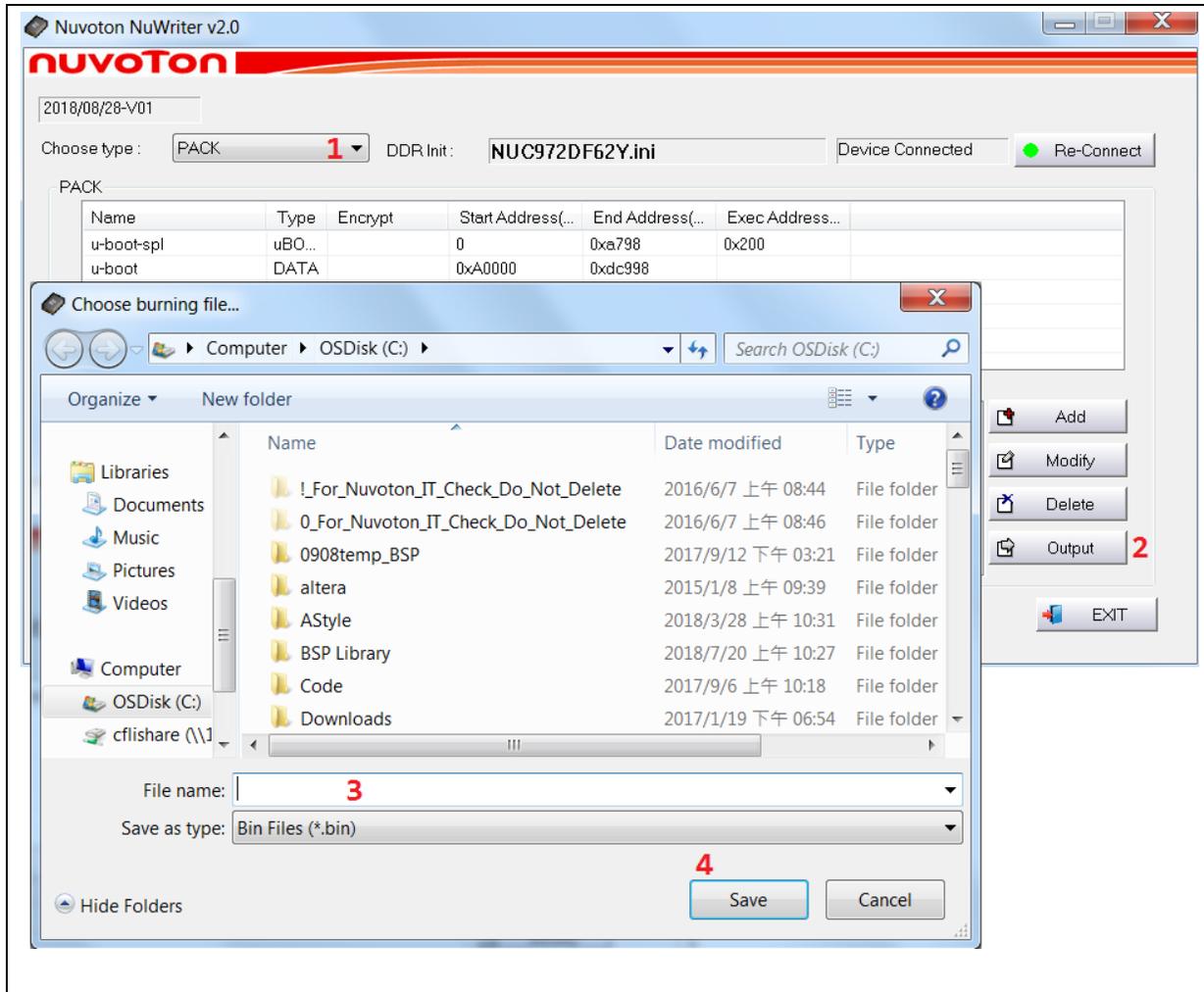


Figure 4-30 PACK – Output Pack Image

依照上圖步驟即可以完成輸出Pack檔案：

1. 選擇“PACK”模式。
2. 按下“Output”。
3. 選擇儲存的檔案。
4. 按下“Open”即可產生Pack Image。

4.5.1.5 燒錄Pack檔案

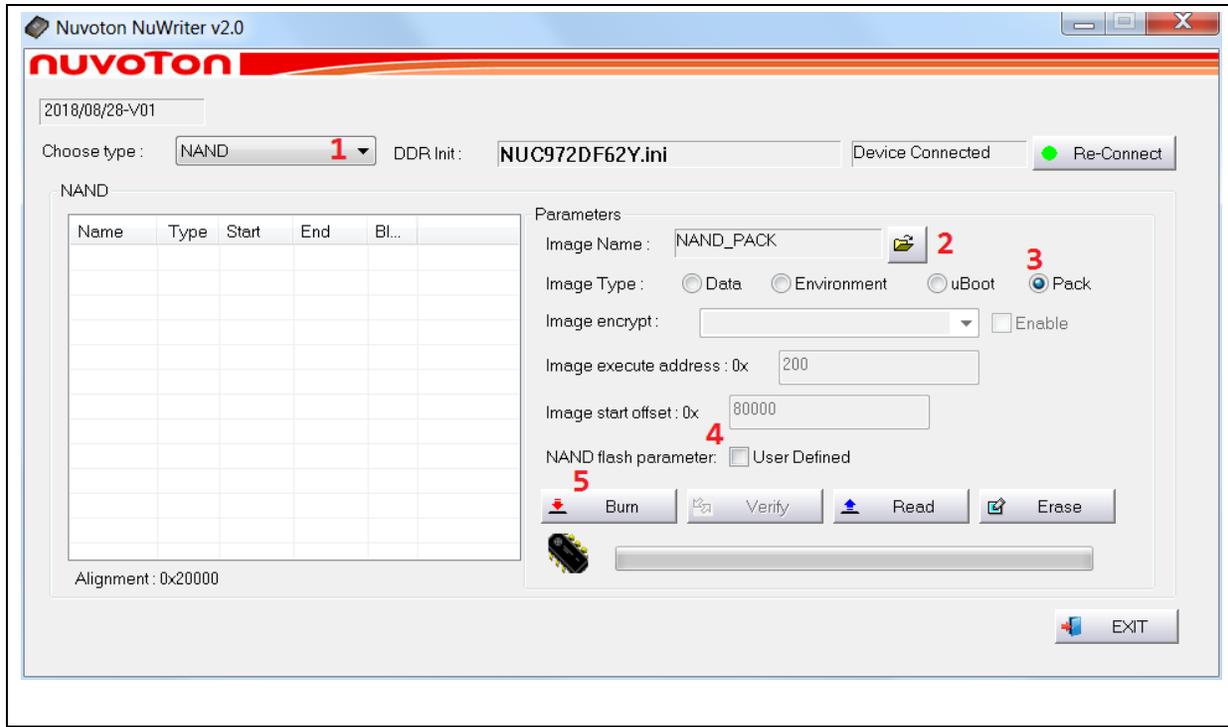


Figure 4-31 PACK – Burn Pack Image

依照上圖步驟以NAND Flash完成燒錄Pack檔案：

1. 選擇需要燒錄的存儲體(eMMC/NAND Flash/ SPI Flash)。
2. 在Image Name 選擇要之前所產生的Pack檔案。
3. Image Type選擇Pack 型態。
4. 勾選是否要使用使用者自訂的NAND參數。
5. 按下“Burn”。

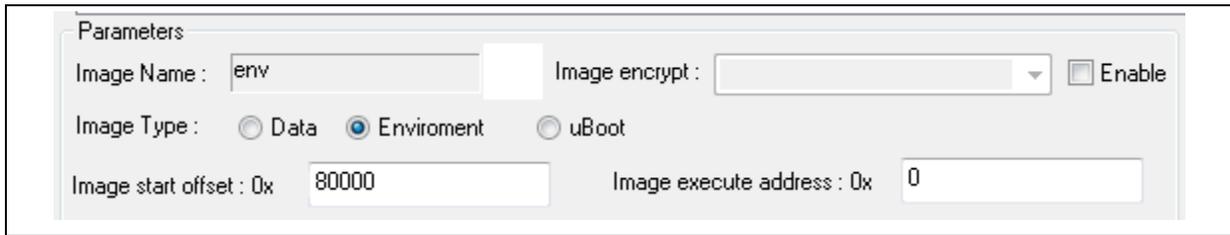
4.5.2 製作和燒錄pack範例

準備相關的檔案：

1. u-boot.bin (預設 offset 為 0xA0000，執行位置為 0xE00000)。
2. u-boot-spl.bin (預設DDR執行位置為 0x200)。
3. env.txt (預設 offset為 0x80000)。

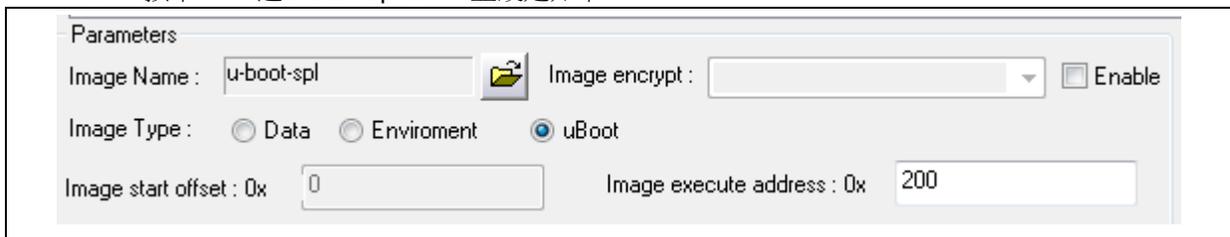
目前假設需要將env.txt、u-boot.bin和u-boot-spl.bin 製作成一個pack檔案並燒錄到NAND Flash，依照下列步驟即可以完成製作Pack檔案：

1. 選擇PACK模式。
2. 按下  選 env.txt，並設定如下：



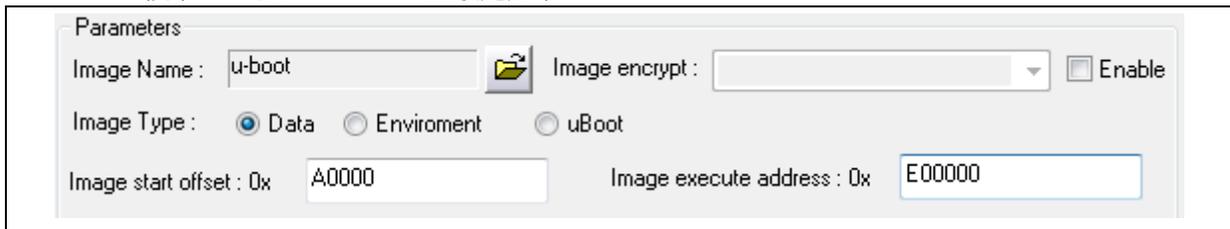
3. 按下  Add 。

4. 按下  選u-boot-spl.bin，並設定如下：



5. 按下  Add 。

6. 按下  選u-boot.bin，並設定如下：



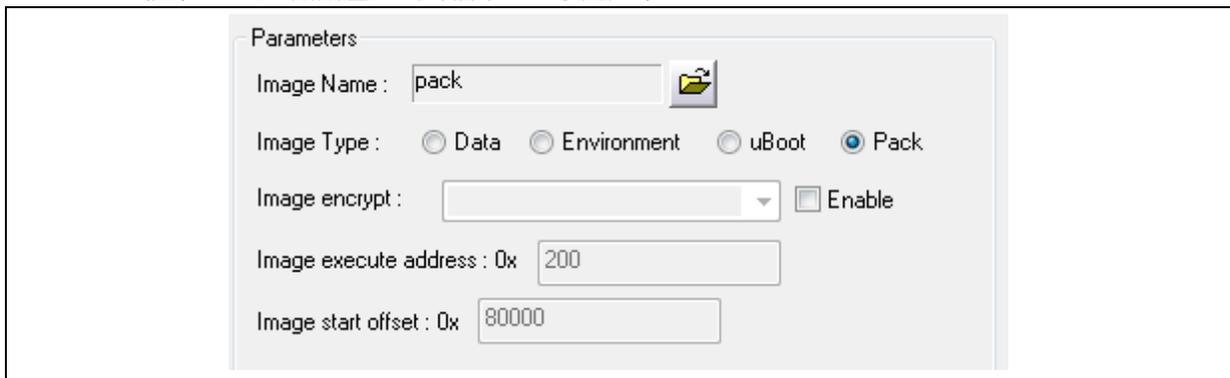
7. 按下  Add 。

8. 按下  Output，可產生一個Pack檔案。

依照下列步驟即可以完成燒錄Pack檔案：

1. 選擇NAND模式。

2. 按下  選 剛剛產生出的檔案，並設定如下：



按下  ，即可燒錄Pack檔案。

4.6 MTP模式 (N9H30 不支持此模式)

MTP模式可以將你選擇的鑰匙檔案燒入到NUC970系列晶片的MTP中，藉由此鑰匙來保護NUC970系列晶片使用到存儲體(eMMC, NAND Flash, SPI Flash)中的程式碼。

4.6.1 操作方法

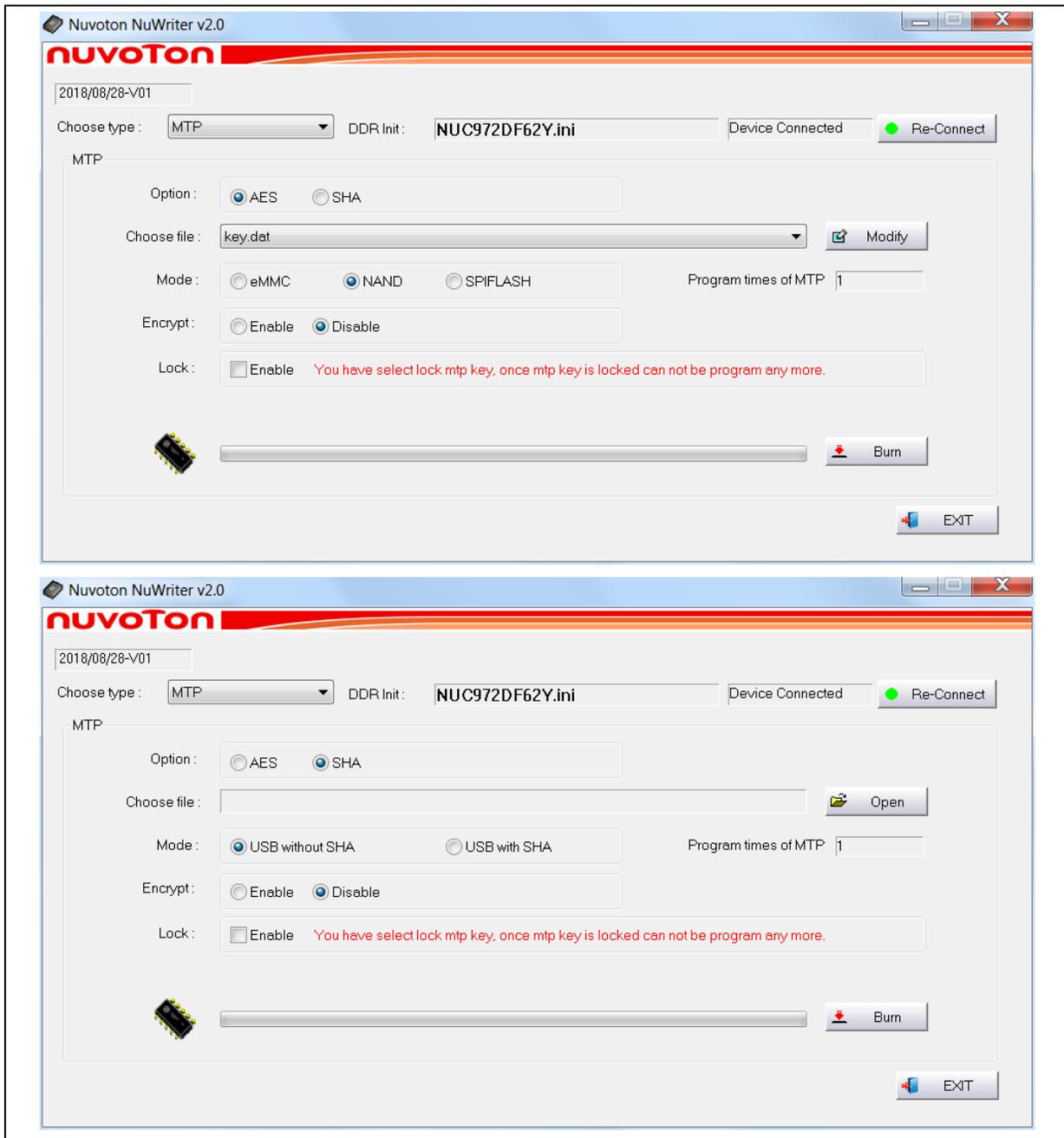


Figure 4-32 NuWriter – MTP Mode

4.6.1.1 新增key檔案

1. 進入資料夾key_cfg如下圖。

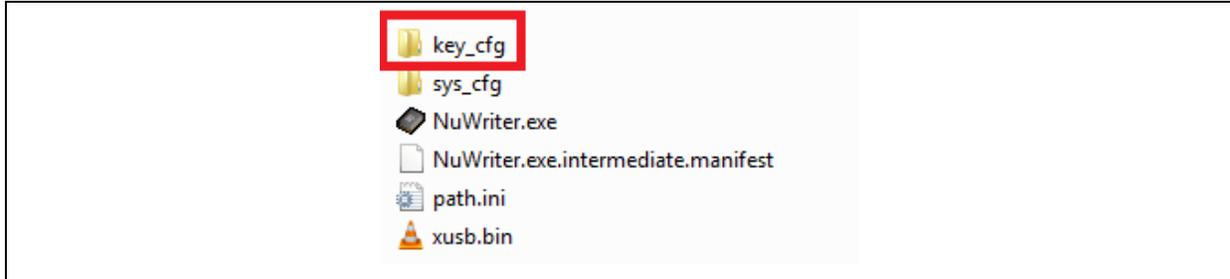


Figure 4-33 NuWriter – Folder

2. 建立文字檔和輸入密碼，密碼格式如下，第一行一定是256，之後連續8行大端模式密鑰。

```
256
0x12345678
0x23456789
0x3456789a
0x456789ab
0x56789abc
0x6789abcd
0x789abcde
0x89abcdef
```

3. 重新開啟Nu-Writer工具，並選擇“MTP”模式。
4. 選擇剛剛建立的文字檔。
5. 選擇燒入的方式：
 - 甲、保護模式選擇：
 - i. AES
 1. 開機模式選擇: eMMC, NAND, 或 SPIFLASH
 - ii. SHA
 1. 開機模式選擇: USB without SHA 或 USB with SHA
 - USB without SHA : NuWriter工具可以使用USB與NUC970溝通不需要透過SHA
 - USB with SHA : NuWriter工具使用USB與NUC970溝通需要透過SHA的驗證
6. 按下“Burn”. 即可完成燒錄動作。

4.7 Mass Production 模式

Mass Production模式針對量產設計提供多台燒錄功能，此功能會自動偵測有多少設備連接到PC，最多可以同時支援八台設備對相同的儲存裝置做燒錄動作。此模式使用的檔案必須是透過Pack模式製造產生出來的檔案，執行時會先把該儲存裝置Erase，接下來才進行Program，最後會再進行資料比對。

4.7.1 操作方法

因為Pack模式已經將所有的Image合併為一個檔案，所以使用者只需要選擇欲燒錄的檔案、儲存到哪種儲存裝置，最後再按下Start按鍵，就可以同時多台設備燒寫。若是欲燒錄的儲存裝置為NAND，就可以選擇使用或不使用自訂的NAND參數。

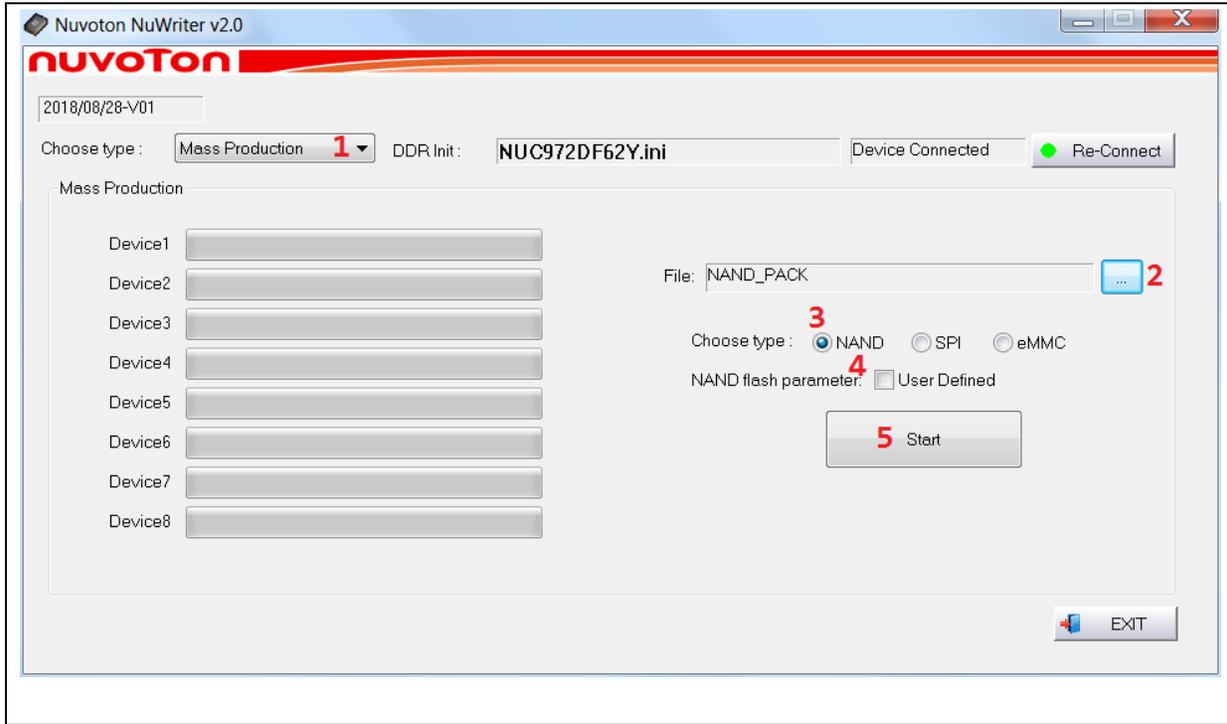


Figure 4-34 Mass Production

5 NUWRITER 源碼介紹

Nuwriter 是使用VS2008為開發環境，所以使用VS2008可以開啟NuWriter專案，開啟後如下圖示：

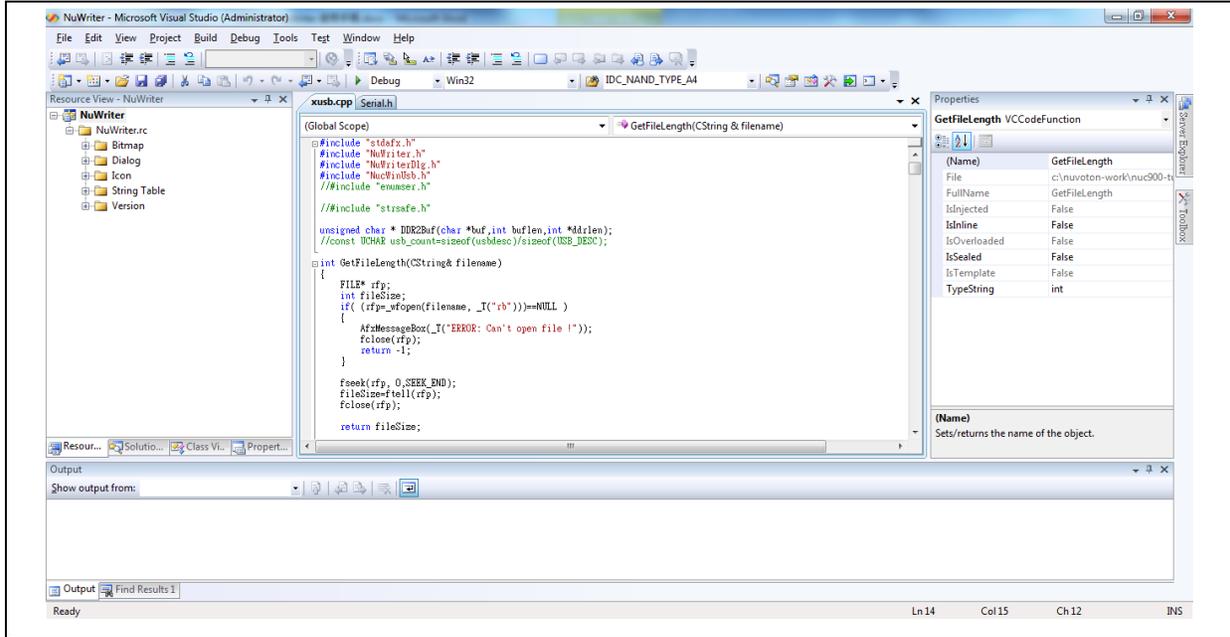


Figure 5-1 NuWriter – Project

可以透過Resource view裡面的Dialog資料夾中看見所有NuWriter的畫面，可以依據使用者需求而進行修改。

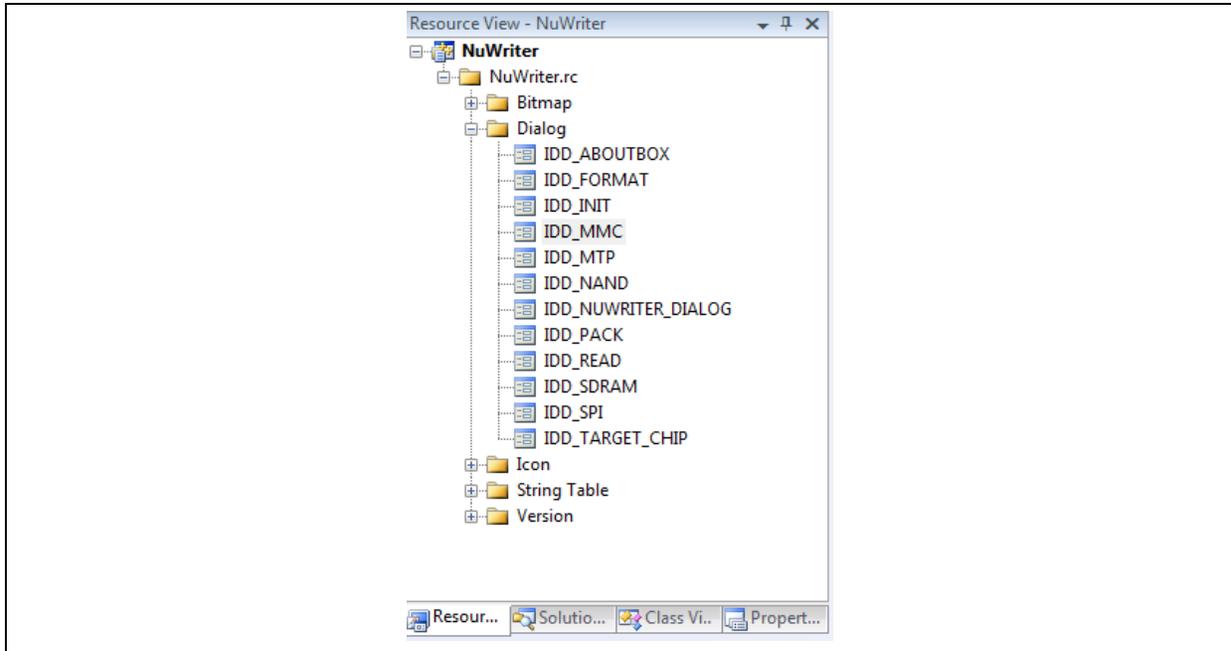


Figure 5-2 NuWriter – Resource View

假設在IDD_SRAM上面點擊兩下，即可看見下面的畫面：

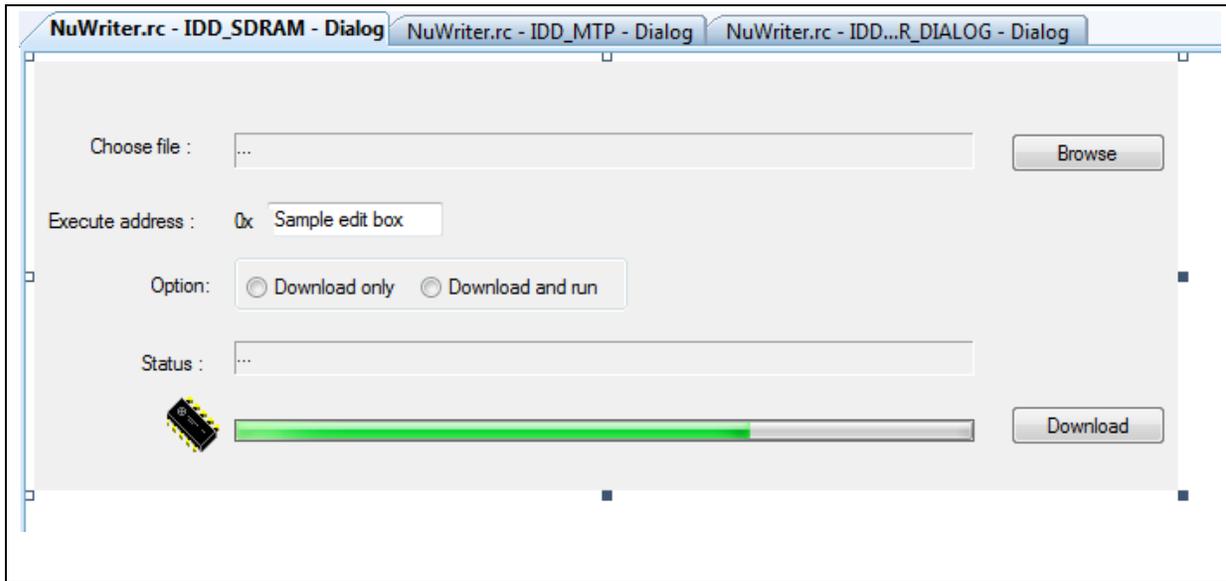


Figure 5-3 IID_SDRAM Dialog

依序再點擊Download按鈕兩下即可看見Download按鈕的程式，當NuWriter運作的時候，按下Download時所運作的程式內容如下圖：

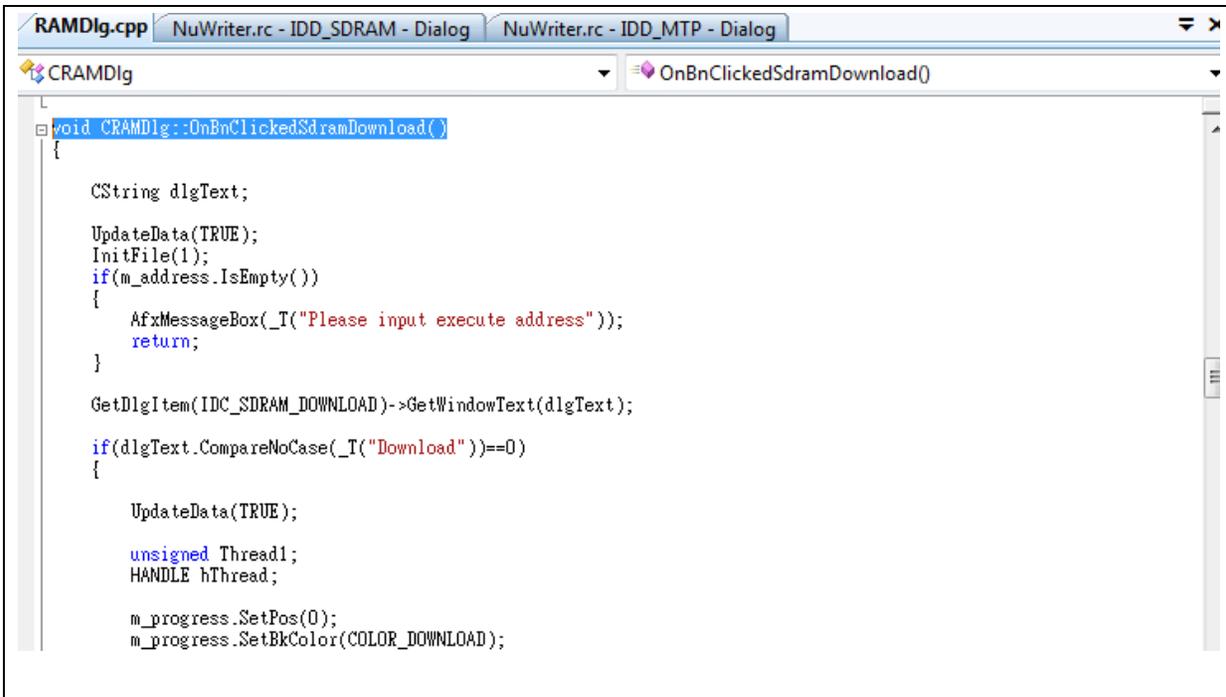


Figure 5-4 OnBnClickedSdramDownload function

在NuWriter 中是由許多的按鈕來完成所有的功能，使用者可以針對對應的按鈕來看source code，進而了解NuWriter的運作方式。

NuWriter 與NuWriterFW之間溝通主要透過NucWinUsb.h和NucWinUsb.cpp來做為之間的橋樑。

NucWinUsb.h :

```

static const GUID OSR_DEVICE_INTERFACE[] = { 0xD696BFEB, 0x1734, 0x417d,
{ 0x8A, 0x04, 0x86, 0xD0, 0x10, 0x71, 0xC5, 0x12 } };
    
```

```

typedef struct PIPE_ID
{
    UCHAR PipeInId;
    UCHAR PipeOutId;
}Pipe_Id,*PPipe_Id;

typedef struct WINUSBHANDLE
{
    HANDLE hDeviceHandle;
    WINUSB_INTERFACE_HANDLE hUSBHandle;
    Pipe_Id pipeid;
    CString DevicePath;
}SwinUsbHandle,*PSwinUsbHandle;

#define MAX_DEVICE_NUM 32
class CNucWinUsb
{
public:
    CNucWinUsb();
    SwinUsbHandle winUsbHandle[MAX_DEVICE_NUM];
    int winUsbNumber;
    UCHAR DeviceSpeed;
    /* 開啟WinUSB driver */
    BOOL EnablewinUsbDevice(void);
    /* 設定將傳到設備的模式 */
    BOOL NUC_SetType(int id,USHORT type,UCHAR * ack,ULONG len);
    /* 寫入資料到設備 */
    BOOL NUC_writePipe(int id,UCHAR *buf,ULONG len);
    /* 從設備讀取資料 */
    BOOL NUC_ReadPipe(int id,UCHAR *buf,ULONG len);
    /* 關閉winUSB driver */
    void NUC_closeHandle(void);
    /* 檢查id是否有連結到電腦 */
    BOOL NUC_CheckFw(int id);
protected:
    BOOL GetDeviceHandle(void);
    BOOL GetWinUSBHandle(void);
    BOOL GetUSBDeviceSpeed(void);
    BOOL QueryDeviceEndpoints(void);
};
    
```

```
static CNucWinUsb NucUsb;
```

假設如果有設備插入到電腦中，可以依序透過下列順序與設備溝通：

```
bResult=NucUsb.EnableWinUsbDevice(); //開啟winUsb driver
bResult=NucUsb.NUC_CheckFw(0); //確認設備0 是否有連結
if(bResult==FALSE)
{
    AfxMessageBox(_T("Please reset device and Re-connect now !!!\n"));
    return FALSE;
}
USHORT typeack=0x0;
/* 設定SDRAM模式，其他模式設定可以參考Serial.h */
bResult=NucUsb.NUC_SetType(0,SDRAM,(UCHAR *)&typeack,sizeof(typeack));
if(bResult==FALSE)
{
    AfxMessageBox(_T("typeack failed !!!\n"));
    NucUsb.NUC_CloseHandle();
    return FALSE;
}

.....跳過.....

/* 寫入SDRAM Header到設備，需要和NuwriterFw配合 */
bResult=NucUsb.NUC_writePipe(0,(UCHAR *)lpBuffer,
sizeof(SDRAM_RAW_TYPEHEAD));
if(bResult==FALSE)
{
    delete []lpBuffer;
    NucUsb.NUC_CloseHandle();
    fclose(fp);
    AfxMessageBox(_T("write SDRAM head error\n"));
    return FALSE;
}

/* 等待設備回傳ack，確定設備有接收到SDRAM Header */
bResult=NucUsb.NUC_ReadPipe(0,(UCHAR *)&ack,4);
if(bResult==FALSE)
{
    delete []lpBuffer;
    NucUsb.NUC_CloseHandle();
    fclose(fp);
}
```

```

    AfxMessageBox(_T("SDRAM head ack error\n"));
    return FALSE;
}
.....跳過.....
while(scnt>0)
{
    fread(lpBuffer,BUF_SIZE,1,fp);
    /* 寫入Data 到設備，最長一次寫4096bytes */
    bResult=NucUsb.NUC_writePipe(0,(UCHAR *)lpBuffer,BUF_SIZE);
    if(WaitForSingleObject(m_ExitEvent, 0) != WAIT_TIMEOUT)
    {
        delete []lpBuffer;
        fclose(fp);
        return FALSE;
    }

    if(bResult==TRUE)
    {
        total+=BUF_SIZE;
        pos=(int)(((float)(((float)total/(float)file_len))*100));
        posstr.Format(_T("%d%%"),pos);
        /* 等待設備回傳ack,確定設備有接收到Data */
        bResult=NucUsb.NUC_ReadPipe(0,(UCHAR *)&ack,4);
        if(bResult==FALSE || ack!=BUF_SIZE)
        {
            delete []lpBuffer;
            NucUsb.NUC_CloseHandle();
            fclose(fp);
            AfxMessageBox(_T("ACK error !"));
            return FALSE;
        }
    }
    .....跳過.....
}

```

6 韌體介紹(XUSB.BIN)

xusb.bin是使用Keil為開發環境，所以使用Keil可以開始NuWriterFW專案，開啟後如下圖示：

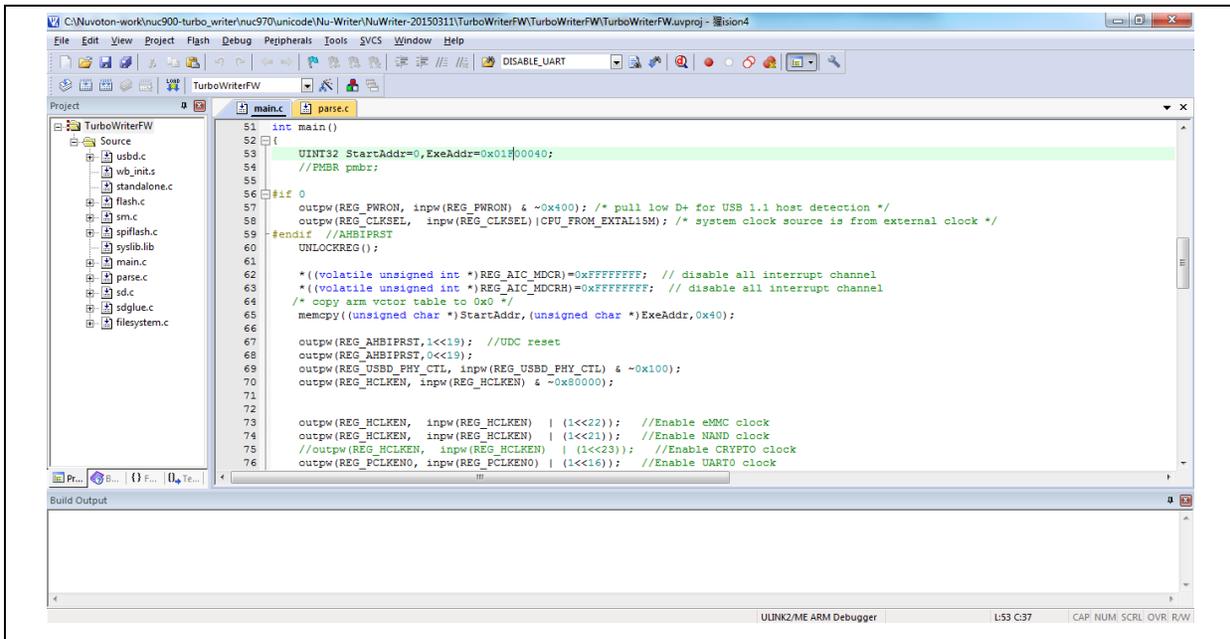


Figure 6-1 NuWriterFW – Project

Firmware code主要功能是透過NuWriter將xusb.bin 傳到設備並且執行，執行後將等待NuWriter透過USB傳送命令，根據命令作相對應的動作。程式執行後主要會while(1)在ParseFlashType(void)函數中等待USB傳送_usbdc_flash_type，如下面表示，假設需要增加從NuWriter傳來新的命令，則並須增加_usbdc_flash_type的值，以及需要修改ParseFlashType(void) 函數和usbdc_control_packet(void)函數，可以參考下面程式中註解的說明。

```

INT ParseFlashType(void){
switch (_usbdc_flash_type){
    /*可以增加接收從Nuwriter傳送過來模式的值*/
    /*
    case yyyyy:
    TODO: Add your control notification handler code here
    break;
    */
    case USBDC_FLASH_SDRAM:
        UXmodem_SDRAM();
        _usbdc_flash_type = -1;
        break;
    case USBDC_FLASH_MMC:
        UXmodem_MMC();
        _usbdc_flash_type = -1;
        break;
    case USBDC_FLASH_NAND:

```

```

        UXmodem_NAND();
        _usbd_flash_type = -1;
    break;
case USBD_FLASH_SPI:
    UXmodem_SPI();
    _usbd_flash_type = -1;
    break;
case USBD_MTP:
    UXmodem_MTP();
    _usbd_flash_type = -1;
    break;
default:
    break;
}

```

usbd.c中的usbd_control_packet()函數，如下：

```

void usbd_control_packet()
{
    .....跳過.....
    if (_usb_cmd_pkt.bRequest == 0xb0){
        /* 可以增加接收從Nuwriter傳送過來模式的值 */
        /*
        if (_usb_cmd_pkt.wValue == (xxxxx)){
            _usbd_flash_type = yyyyy;
            outpw(REG_USBD_CEP_CTRL_STAT, CEP_NAK_CLEAR);
            // clear nak so that sts stage is complete
        }
        xxxxx : Nuwriter傳過的值
        yyyyy : ParseFlashType(void) Parse的值
        */

        if (_usb_cmd_pkt.wValue == (USB_BURN_TYPE+USB_FLASH_SDRAM)){
            _usbd_flash_type = USB_FLASH_SDRAM;
            outpw(REG_USBD_CEP_CTRL_STAT, CEP_NAK_CLEAR);
            // clear nak so that sts stage is complete
        }
        else if (_usb_cmd_pkt.wValue == (USB_BURN_TYPE+USB_FLASH_NAND)){
            _usbd_flash_type = USB_FLASH_NAND;
            // clear nak so that sts stage is complete
            outpw(REG_USBD_CEP_CTRL_STAT, CEP_NAK_CLEAR);
        }
    }
}

```

```

else if (_usb_cmd_pkt.wValue == (USBBD_BURN_TYPE+USBBD_FLASH_NAND_RAW))
{
    _usb_flash_type = USBBD_FLASH_NAND_RAW;
    // clear nak so that sts stage is complete
    outpw(REG_USBBD_CEP_CTRL_STAT, CEP_NAK_CLEAR);
}
else if (_usb_cmd_pkt.wValue == (USBBD_BURN_TYPE+USBBD_FLASH_MMC)){
    _usb_flash_type = USBBD_FLASH_MMC;
    // clear nak so that sts stage is complete
    outpw(REG_USBBD_CEP_CTRL_STAT, CEP_NAK_CLEAR);
}
.....跳過.....
}
    
```

修改完成後，可以編譯成xusb.bin，如下圖所示

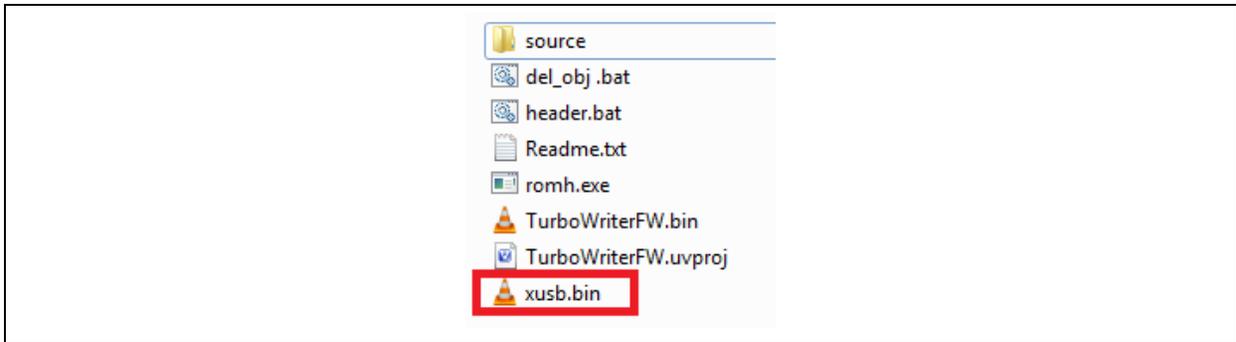


Figure 6-2 NuWriterFW – Folder

並且將xusb.bin覆蓋NuWriter資料夾中原本的xusb.bin，即完成更新NuWriterFW。

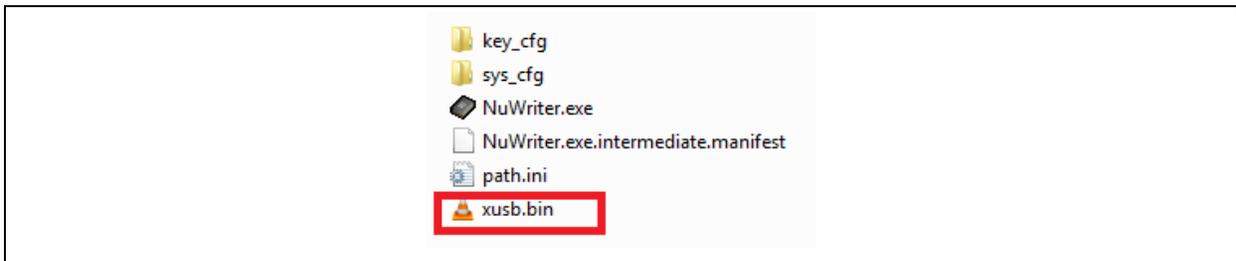


Figure 6-3 NuWriter – Folder

7 REVISION HISTORY

Date	Revision	Description
2015.04.23	1.00	初版
2015.06.02	1.00	燒錄 NAND 時以區塊為單位來燒錄
2015.06.05	1.00	Pack 模式下增加加密功能(AES)
2015.06.15	1.00	NAND 模式下,移除 FS 型態, 使用者可以改用 data 型態來燒錄
2015.07.28	1.00	當 Nuwriter 燒錄 eMMC 時,速度增加到 1Mhz. 從 eMMC 開機時, eMMC 的時脈增加到 6Mhz.
2015.08.25	1.00	NuWriter 增加 xusb16.bin, xusb.bin, xusb64.bin 分別給 nuc970 DDR 大小 (16MB,32MB,64MB) 使用
2015.08.31	1.00	在 path.ini 中增加 ChipReadWithBad 參數. NAND 讀取資料時 1: 讀取 data 區域和 OOB 區域, 0: 讀取 data 區域 在 path.ini 中增加 ChipEraseWithBad 參數. NAND 擦除資料時 1: 擦除 data 區域和 OOB 區域, 0: 擦除 data 區域
2015.09.24	1.00	修正當資料大於 31MB 的時候燒錄失敗. Pack 模式增加 MTP 功能.
2015.10.21	1.00	增加 "重新連線功能(重複 5 次)" 預防第一次連結 NUC970 時失敗.
2018.08.14	1.01	增加 N9H30 描述 增加量產模式(一對多)支援
2018.08.30	2.00	NAND 模式與量產模式, 新增使用者自訂 NAND 參數設定

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, “Insecure Usage”.

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer’s risk, and in the event that third parties lay claims to Nuvoton as a result of customer’s Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

*Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.*